 Docker run

| docker run -it --name=cont1 ubuntu /bin/bash |
| --- |

* Crea un contenedor con la imagen “ubuntu” (al no especificar, toma versión “latest”), le establece un nombre “cont1” y lanza en modo interactivo una shell “bash”.

| docker run -d -p 1200:80 nginx |
| --- |

* Crea un contenedor con la versión “latest” de la imagen “nginx” y lo lanza en “background”, exponiendo el puerto 80 del contenedor en el puerto 1200 de la máquina anfitrión.

| docker run -it -e MENSAJE=HOLA ubuntu:14.04 bash |
| --- |

* Crea un contenedor con la imagen “ubuntu”, versión “14.04” y establece la variable de entorno “MENSAJE”.

 Docker ps

| docker ps |
| --- |

* Muestra información de los contenedores en ejecución.

| docker ps -a |
| --- |

* Muestra información de todos los contenedores, tanto parados como en ejecución.

 Docker start/stop/restart

| docker start micontenedor |
| --- |

* Arranca el contenedor con nombre “mi contenedor”.

| docker start -ai micontenedor |
| --- |

* Arranca el contenedor con nombre “mi contenedor”, enlazando el comando ejecutado al arranque a la entrada y salida estándar de la terminal del anfitrión.

 Docker exec

| docker exec -it -e FICHERO=prueba cont bash |
| --- |

* Lanza en el contenedor “cont” (que debe estar arrancado) el comando “bash”, estableciendo la variable de entorno “FICHERO” y enlazando la ejecución de forma interactiva a la entrada y salida estándar del anfitrión.

| docker exec -d cont touch /tmp/prueba |
| --- |

* Lanza en el contenedor “cont” (que debe estar arrancado) el comando “touch /tmp/prueba”. Este comando se ejecuta en segundo plano, generando el fichero “/tmp/prueba”.

 Docker attach

| docker attach idcontainer |
| --- |

* Enlaza nuestra terminal la entrada/salida de nuestra al proceso en segundo plano del contenedor “idcontainer”.

 Docker logs

| docker logs -n 10 idcontainer |
| --- |

* Muestra las 10 últimas líneas de la salida estándar producida por el proceso en ejecución en el contendor.

 Docker cp

| docker cp idcontainer:/tmp/prueba ./ |
| --- |

* Copia el fichero “/tmp/prueba” del contenedor “idcontainer” al directorio actual del anfitrión.

| docker cp ./miFichero idcontainer:/tmp |
| --- |

* Copia el fichero “miFichero” del directorio actual del anfitrión a la carpeta “/tmp” del contenedor.

 Gestión de imágenes

| docker images |
| --- |

* Información de imágenes locales disponibles.

| docker search ubuntu |
| --- |

* Busca la imagen “ubuntu” en el repositorio remoto (por defecto Docker Hub).

| docker pull alpine |
| --- |

* Descarga localmente imagen “alpine”.

| docker history alpine |
| --- |

* Muestra la historia de creación de la imagen “alpine”.

| docker rmi ubuntu:14.04 |
| --- |

* Elimina localmente la imagen “ubuntu” con tag “14.04”.

| docker rmi $(docker images -q) |
| --- |

* Borra toda imagen local que no esté siendo usada por un contenedor.

| docker rm IDCONTENEDOR |
| --- |

* Borra un contenedor con IDCONTENEDOR.

| docker stop $(docker ps -a -q) |
| --- |

* Para todos los contenedores del sistema.

| docker rm $(docker ps -a -q) |
| --- |

* Borra todos los contenedores parados del sistema.

| docker system prune -a |
| --- |

* Borra todas las imágenes y contenedores parados del sistema.

 Gestión de redes

| docker network create redtest |
| --- |

* Creamos la red “redtest”

| docker network ls |
| --- |

* Nos permite ver el listado de redes existentes.

| docker network rm redtest |
| --- |

* Borramos la red “redtest”.

| docker run -it --network redtest ubuntu /bin/bash |
| --- |

* Conectamos el contenedor que creamos a la red “redtest”.

| docker network connect IDRED IDCONTENEDOR |
| --- |

* Conectamos un contenedor a una red.

| docker network disconnect IDRED IDCONTENEDOR |
| --- |

* Desconectamos un contenedor de una red

 Volúmenes

| docker run -d -it --name appcontainer -v /home/sergi/target:/app nginx:latest |
| --- |

* Creamos un contenedor y asignamos un volumen con “binding mount”.

| docker run -d -it --name appcontainer -v micontenedor:/app nginx:latest |
| --- |

* Creamos un contenedor y asignamos un volumen Docker llamado “micontenedor”.

| docker volume create/ls/rm mivolumen |
| --- |

* Permite crear, listar o eliminar volúmenes Docker.

| docker run -d -it --tmpfs /app nginx |
| --- |

* Permite crear un contenedor y asociar un volumen “tmpfs”.

| docker run --rm --volumes-from contenedor1 -v /home/sergi/backup:/backup ubuntu bash -c "cd /datos && tar cvf /backup/copiaseguridad.tar ." |
| --- |

* Permite realizar una copia de seguridad de un volumen asociado a “contenedor1” y que se monta en “/datos”. Dicha copia finalmente acabará en “/home/sergi/backup” de la máquina anfitrión.

| docker volume rm $(docker volume ls -q) |
| --- |

* Permite eliminar todos los lúmenes de tu máquina.

 Ejemplo básico de fichero “docker-compose.yml”

| version: "3.9" services:  db:  image: mariadb:10.11.2  volumes:  - db\_data:/var/lib/mysql  environment:  MARIADB\_ROOT\_PASSWORD: somewordpress  MARIADB\_DATABASE: wordpress  MARIADB\_USER: wordpress  MARIADB\_PASSWORD: wordpress  wordpress:  image: wordpress:latest  ports:  - "8000:80"  environment:  WORDPRESS\_DB\_HOST: db:3306  WORDPRESS\_DB\_USER: wordpress  WORDPRESS\_DB\_PASSWORD: wordpress  WORDPRESS\_DB\_NAME: wordpress volumes:  db\_data: |
| --- |

 Principales comandos de “Docker Compose”

| docker compose up -d |
| --- |

* Inicia el sistema definido en “***docker-compose.yml***” en segundo plano. Genera y descarga imágenes.

| docker compose down |
| --- |

* Detiene y elimina los contenedores según la configuración de “***docker-compose.yml***”.

| docker compose build/pull |
| --- |

* Construye/descarga las imágenes de contenedores según la configuración de “***docker-compose.yml***”.

| docker compose ps |
| --- |

* Muestra información de los contenedores según la configuración de “***docker-compose.yml***”.