|  |  |
| --- | --- |
| **Comandos** | **Descripción** |
| **Command line docker** | |
| **docker run {name\_contein}** | Ejecuta un contenedor especificado. |
| **docker ps** | Ver los contenedores corriendo o en ejecución. |
| **docker ps -q** | Nos permite ver los id de los contenedores que se están ejecutando. |
| **docker ps -l** | Nos muestra el ps del último proceso, este o no ejecutando. |
| **docker run –name <custom\_name> <image\_name>** | Nos permite crear un contenedor con un nombre personalizado a partir de una imagen. |
| **docker rename <old\_name> <image\_name>** | Nos permite renombrar un contenedor. |
| **docker rm <container\_id or container\_name>** | Elimina un contenedor. |
| **docker rm -f <container\_id or container\_name>** | Fuerza a eliminar un contenedor especificado. |
| **docker run -it <image\_name>** | La bandera -**it** hace referencia al **modo interactivo**, es decir, descarga y ejecuta una imagen de docker. |
| **docker exec -it <image\_name>** | Nos permite en un contenedor que ya existe y esta corriendo ejecutar un comando o un proceso dentro del container. |
| **docker run -it --rm -v <path-to-sock-of-docker>:<path-of-sock-destino> <name\_image>** | Nos permite ejecutar un sock de forma interactiva con otro sock de docker para su comunicación y luego, de forma automática eliminarlo una vez terminada la ejecución del mismo.  **NOTA**: Esto sólo funciona en Linux y Mac. |
| **docker run -it -rm -v <path-to-sock-of-docker>:<path-of-sock-destino> -v $(which docker):/bin/docker wagoodman/dive <name\_image\_to\_inspect>** | Nos permite ejecutar un contenedor de forma interactiva, eliminarlo de forma automática una vez pare la ejecución, levanta un volumen sock local (container) a un destino dentro del contenedor. Descargar e instalar la herramienta **Dive** para inspeccionar docker desde otro contenedor.  **NOTA**: Ejecutar esto en entornos controlados únicamente, porque puede llegar a ser muy peligroso. |
| **docker run --name <name\_container> -d <image\_name> tail -f /dev/null** | Nos permite ejecutar de forma continua un contenedor. |
| **docker inspect --format '{{.State.Pid}}' <name\_container>** | Nos permite obtener el numero de proceso de un contenedor. |
| **kill -9 <number\_process>** | Mata un proceso determinado. Solo funciona en linux. |
| **docker restart $(docker ps -q)** | Reinicia todos los contenedores que están corriendo. |
| **docker stop $(docker ps -aq)** | Para todos los contenedores que están corriendo. |
| **docker stop <name\_container>** | Nos permite enviarle la señal **SIGTERM** a un contenedor. Esto quiere decir que, parará nuestro container (siempre que sea posible).  **NOTA**: Sí hacemos **docker stop** y nuestro container no se para de forma instantánea, es decir, no responde, docker entonces le enviará SIGKILL para forzar la salida.  Para ver el status de salida, podemos usar **docker ps -l** si chequeamos en la columna **STATUS**, podemos ver el código de salida del último proceso.  Sí tenemos un código mayor a 128, esto indica una salida por una excepción o una señal no manejada correctamente. |
| **docker kill <name\_image>** | Nos permite enviar una señal **SIGTERM** a un contenedor determinado. |
| **docker exec <name\_container> ps -ef** | Nos permite listar el proceso de un contenedor especifico. |
| **docker run -d --name <name> -p {port\_machine\_local}:{port\_of\_container} <name\_container>** | Permite exponer un contenedor en un determinado puerto en background. |
| **docker logs <name\_container>** | Permite ver los log del contenedor. |
| **docker logs -f <name\_container>** | Permite ver los log del contenedor en tiempo real. |
| **docker logs --tail 10 -f <name\_container>** | Permite ver los últimos 10 log. |
| **docker exec -it <name\_container> bash** | Ejecuta un bash del contenedor especificado. |
| **docker run -d --name db -v /path/absolute/of/folder/my\_computer:/path/of/tool/container** | Permite hacer un **bind mounts**, es decir, permite especificar una ruta donde se persistan los datos de la imagen del contenedor.  **NOTA**: A la izquierda va la ruta completa de la carpeta de la máquina local. A la derecha va la ruta de la herramienta dentro del contenedor. |
| **docker volume create <name\_volume>** | Crea un volumen. |
| **docker run -d --name db --mount src=<name\_volume>,dst=/path/of/dist/container** | Permite montar un contenedor dentro de un volumen especificado.  **NOTA**: Primero debemos especificar que queremos montar con **src**. Sí el volumen no existe, se crea en esta instancia.  Segundo el destino, es decir, adonde va a estar montado, se realiza con la propiedad **dst**. |
| **docker cp <path/of/file/to/copy> <name\_container>:<path/inside/container>** | Nos permite copar un archivo desde el host local dentro del contenedor. |
| **docker cp <name\_container>:<path/inside/container> <path/host/loca>** | Nos permite copiar archivos desde el contenedor all host local. |
| **docker history <name\_image>** | Nos permite ver el historial de una imagen. |
| **docker pull <name\_image:tag>** | Nos permite traer una imagen desde un repositorio remoto. |
| **docker build -t <base\_image>:<tag\_image\_name> <dockerfile context>** | Nos permite crear una imagen propia. <dockerfile context> es el directorio o ruta actual. |
| **docker run -it <image>:<image\_tag> <command>** | Nos permite correr nuestro contenedor. |
| **docker run -it –rm** | Nos permite ejecutar un contenedor de forma interactiva y elimina automáticamente el contenedor una vez que haya terminado su ejecución. |
| **docker login -u <you\_user> -p <you\_password>** | Nos permite loguearnos en docker hub. |
| **cat ~/.docker/config.json** | Verifica el logueo de docker hub. Sí la llave ‘**auth**’ contiene algo, se logueo correctamente. |
| **docker tag <image>:<tag> <you\_username>/<image>:<tag>** | Nos permite retaguear una imagen para subir al repositorio. |
| **docker push <you\_username>/<image>:<tag>** | Nos permite subir una imagen. |
| **touch dockerfile** | Nos permite crear un archivo de docker. |
| **docker commit** | Es muy parecido al commit de git.  Guarda el **estado** de la capa mutable en un **tag** de la imagen a partir de la cual se genero el contenedor creando una capa mas para crear uno o mas contenedores a partir de esta nueva imagen. |
| **Docker run –rm <image>** | Nos permite eliminar automáticamente el contenedor luego de pararlo |
| **Docker run –rm -p <port\_local>:<port\_container> <image>** | Nos permite eliminar automáticamente el contenedor luego de pararlo y con la bandera -p exponemos el puerto del contenedor en la máquina local. |
| **docker network create --atachable <name\_network>** | Nos permite crear una red y que otros contenedores se conecten a ella. |
| **docker network inspect <name\_network>** | Nos permite inspeccionar una red determinada. |
| **docker network connect <name\_network> <name\_container>** | Nos permite conectar una red especifica a un contenedor determinado. |
| **docker run -d -name <name\_app> -p <port\_local>:<port\_container> --env <name\_var\_env>=<protocol\_name>://<name\_container\_db>:<port\_db>/<name\_database> <name\_image>** | Nos permite ejecutar un contenedor pasandole variables de entorno de una conexión a una base de datos. Ejemplo:  **docker run -d -name app -p 3000:3000 --env MONGO\_URL=mondodb://db:27017/test platzi** |
| **docker buils -t <name\_image>** | Nos permite crear una imagen de Docker a partir de un archivo **Dockerfile**. |
| **docker run --name <name\_image> <name\_container> <hostname>** | Nos permite pasar un parámetro a **docker run** y hacer algo con él, pero previamente se tiene que agregar el ENTRYPOINT en el **Dockerfile**. |
| **docker build -t <name\_image> -f <path-dockerfile> <context>** | Nos permite ejecutar un container levantando un dockerfile especificado. |
| **Administración de ambiente** | |
| **docker ps -a** | Permite ver toda la lista de contenedores corriendo o los que fueron ejecutados, es decir, los apagados. |
| **docker system prune** | Nos permite borrar todo lo que no se esta usando (image, volume, network, etc). |
| **docker container prune** | Elimina todos los contenedores apagados. |
| **docker network prune** | Elimina todas las redes que no estén en uso. |
| **docker volume prune** | Elimina todos los volúmenes que no estén en uso. |
| **docker image prune** | Elimina todas las imágenes que no se estén usando. |
| **docker rm -f $(docker ps -aq)** | Elimina TODOS los contenedores que estén en ejecución o apagados. |
| **docker network ls** | Nos permite listar las redes disponibles en los contenedores. |
| **docker volume ls** | Nos permite ver todos los volúmenes. |
| **docker image ls** | Nos permite listar las imágenes que tenemos actualmente.  **NOTA**: Sí se listan imágenes iguales por alguna razón, esas imágenes no se repiten, docker apunta el puntero a los mismos archivos y por ende, es únicamente una imagen, es decir, no ocupa doble logar en disco. |
| **docker run -d --name <name\_container> --memory <number>g <name\_image>** | Nos permite limita el uso de la memoria para un contenedor especifico.  **NOTA**: Si un contenedor no arranca**,** con **docker inspect <container\_id or container\_name>** podemos inspeccionar el container y fijarnos en el indice “State”: **“OOMKill”: true** lo cual significa **Out of Memory Kill** (matar sin memoria), si OOMKill esta en **true**, esto se debe que el container no puede ejecutarse por falta de memoria. |
| **docker stats** | Nos permite ver cuantos recursos ocupa docker en nuestro SO. |
| **docker inspect <container\_id or container\_name>** | Permite inspeccionar la configuración del contenedor |
| **Docker compose** | |
| **docker-compose up** | Nos permite levantar todas nuestras configuraciones que tengamos en los archivos **.yml** de nuestros servicios. |
| **docker-compose up -d** | Nos permite levantar los servicios en dettach (en background) |
| **docker-compose down** | Nos permite bajar nuestros servicios. |
| **docker-compose build** | Nos permite reconstruir nuestros servicios. |
| **docker-compose logs** | Nos permite ver los logs de nuestros contenedores. |
| **docker-compose logs -f** | Nos permite ver los logs en modo **follow**. |
| **docker-compose logs <service\_name>** | Nos permite ver los logs de un contenedor en especifico. |
| **docker-compose exec <service\_name> <command>** | Nos permite correr un comando en un contenedor determinado. |
| **docker-compose run –rm <service\_name> <command>** | Nos permite correr un nuevo contenedor basado en un servicio. |
| **docker-compose –rm –service-ports <service\_name> <command>** | Nos permite correr un nuevo contenedor basado en un servicio exponiendo un puerto. |
| **docker-compose up -d** | Nos permite Levantar un servicio desde nuestro archivo.yml en modo **--detach** |
| **docker-compose up -d --scale <name\_container>=<number\_instance>** | Nos permite crear más de una instancia de un contenedor a la vez.  **NOTA**: Deben estar en puertos distintos, de lo contrario arrojará un error. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |