Mondragon Unibertsitatea

Dockers: Containers

Docker: Containers

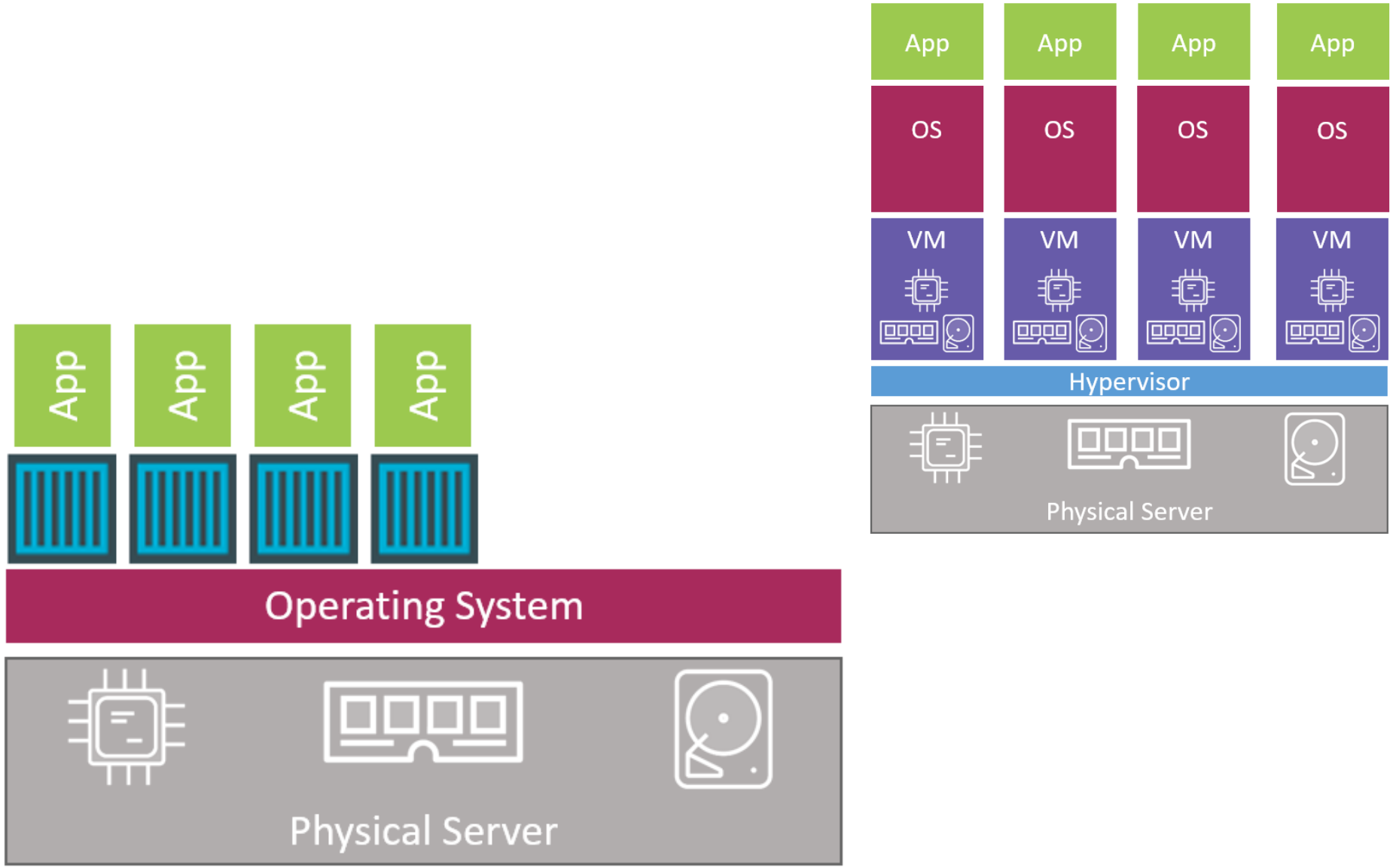
Mondragon Unibertsitatea
Goi Eskola Politeknikoa

- Objetivos:
  - ¿Qué son los contenedores Docker (Docker containers)?
  - Cómo llevar a cabo las operaciones básicas.

Docker Containers

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Containers vs VMs



Docker Containers

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Docker Container

Un contenedor consta de un sistema operativo, archivos añadidos por el usuario y metadatos. Cada contenedor se construye a partir de una imagen. Esa imagen le dice a Docker qué contiene el contenedor, qué proceso ejecutar cuando se lanza el contenedor, y una variedad de otros datos de configuración. La imagen de Docker es de sólo lectura. Cuando Docker ejecuta un contenedor a partir de una imagen, añade una capa de lectura-escritura sobre la imagen (usando un sistema de archivos de unión) en la que tu aplicación puede ejecutarse.

Docker Engine (Motor)

El host Docker se crea como parte de la instalación de Docker en tu máquina. Una vez que su host Docker ha sido creado, entonces le permite gestionar imágenes y contenedores. Por ejemplo, la imagen se puede descargar y los contenedores se pueden iniciar, detener y reiniciar.

Docker Containers

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Docker Client

- El cliente se comunica con el host Docker y le permite trabajar con imágenes y contenedores.
- Comprueba si tu cliente está funcionando utilizando el siguiente comando:

$ docker –v

- La versión exacta del Cliente y del Servidor se puede ver utilizando el comando $ docker version. Salida:

Cliente:
Versión: 17.09.0-ce-rc3
Versión de la API: 1.32
Versión de Go: go1.8.3
Confirmación Git: 2357fb2
Construido: Thu Sep 21 02:31:18 2017
SO/Arch: darwin/amd64

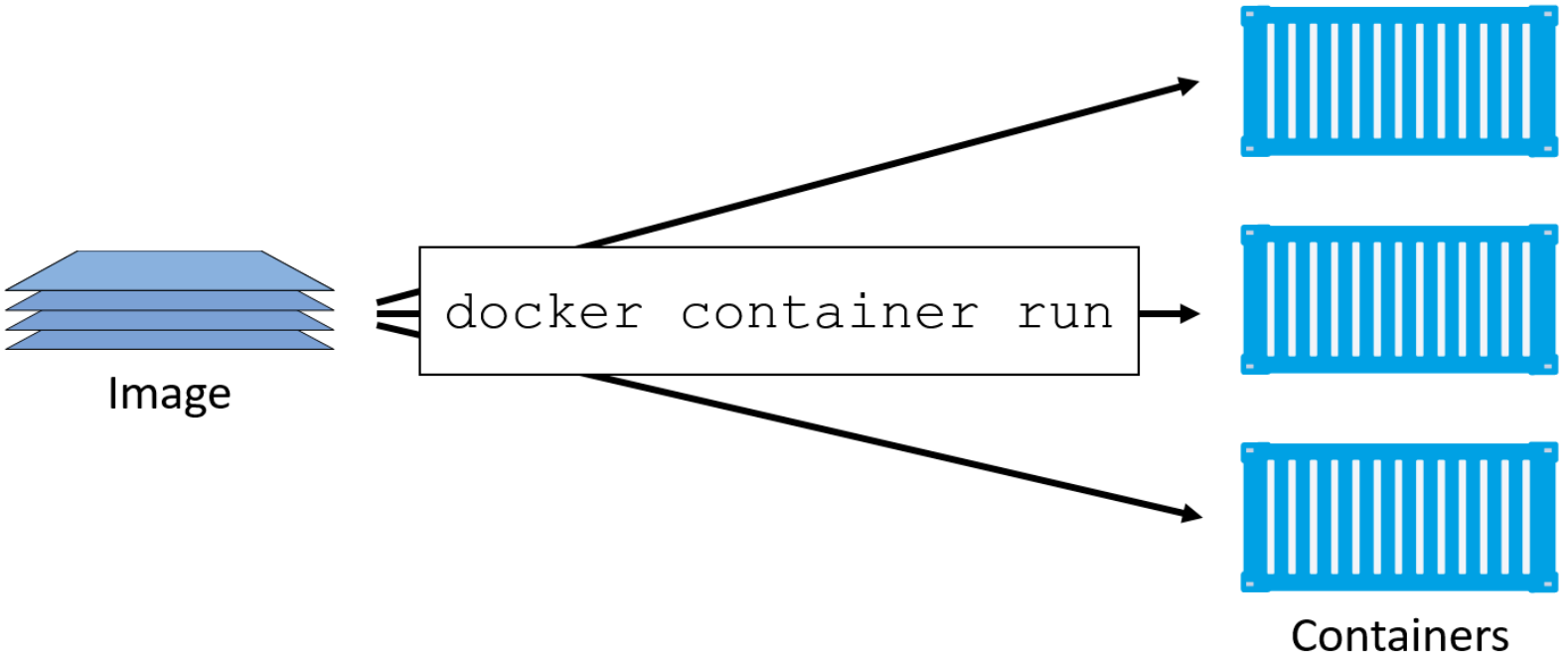
Servidor:
Versión: 17.09.0-ce-rc3
Versión de la API: 1.32 (versión mínima 1.12)
Versión de Go: go1.8.3
Confirmación Git: 2357fb2
Construido: Thu Sep 21 02:36:52 2017
SO/Arch: linux/amd64
Experimental: true

El conjunto completo de comandos se puede ver usando docker --help.

Docker Containers: Idea General

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

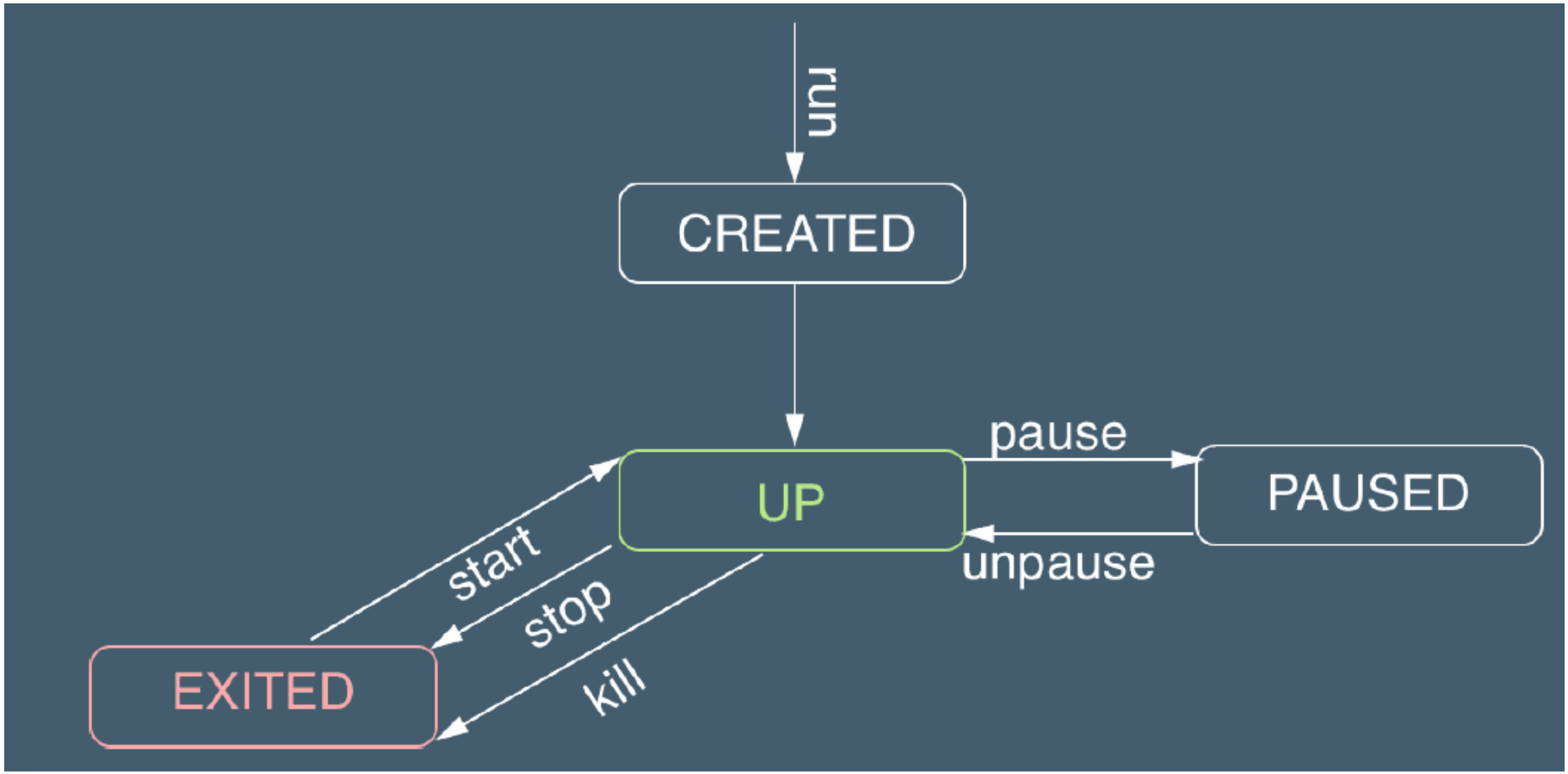
- Un contenedor es la instancia en tiempo de ejecución de una imagen.
- De la misma manera que podemos iniciar una máquina virtual (VM) desde una plantilla de máquina virtual, iniciamos uno o más contenedores desde una sola imagen.
- La gran diferencia entre una VM y un contenedor es que los contenedores son más rápidos y ligeros.



Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Ciclo de vida del Container



Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

- Forma más simple de **iniciar** un contenedor:

```
$ docker container run
```

- El comando puede tener muchos argumentos, pero la forma más básica es pasarle la imagen y la aplicación que queramos que ponga en marcha:

```
$ docker container run <image> <app>.
```

- El ejemplo, pone en marcha un contenedor con Ubuntu Linux y la aplicación que ponemos en marcha es el shell bash:

```
$ docker container run -it tomcat:latest /bin/bash
```

- Los contendores están en marcha mientras la aplicación está corriendo. En el ejemplo anterior, el contenedor terminará cuando cerremos el shell bash.

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Iniciar un contenedor simple

- Una vez extraída la imagen, el demonio crea el contenedor y ejecuta la aplicación especificada dentro de él (/bin/bash).
- Al ejecutar, vemos que el intérprete de comandos del shell ha cambiado y ahora nos encuentramos dentro del contenedor. En el ejemplo citado, el intérprete de comandos del shell ha cambiado a algo así como *root@3027eb644874:/#*. El número largo después de la @ son los primeros 12 caracteres del ID único del contenedor.
- Intente ejecutar algunos comandos básicos dentro del contenedor. Es posible que note que algunos comandos no funcionan. Esto se debe a que las imágenes que utilizamos, como casi todas las imágenes de los contenedores, están altamente optimizadas para los contenedores y son versiones muy simplificadas.
  - No tienen todos los comandos y paquetes normales instalados. Ejemplo: en alpine:latest, el /bin/bash no funciona, no tiene instalado ni el bash. Prueba en el contenedor de tomcat el comando **ping**

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Procesos de los Contenedores

Ejecuta el comando: ps –elf (muestra los procesos en marcha)

- Aunque puede parecer que hay dos procesos ejecutándose en la salida anterior, no los hay.
  - El primer proceso en la lista, con PID 1, es el shell de Bash que le dijimos al contenedor que se ejecutara.
  - El segundo proceso es el comando ps -elf que ejecutamos para producir la lista. Se trata de un proceso de corta duración que ya ha salido en el momento en que se visualiza el mensaje.
- Si escribe exit, para salir del shell de Bash, el contenedor también saldrá (terminará). Un contenedor no puede existir sin un proceso en marcha.
- Matando el terminal bash mata el único proceso del contenedor, resultando en que el contenedor también sea matado.
- Pulse Ctrl-PQ para salir del contenedor sin terminarlo.
  - Si lo hace, volverá a la shell del host de su Docker y dejará el contenedor en funcionamiento en segundo plano. Compruébelo con el siguiente comando:

    ```
    $ docker container ls
    ```
- El contenedor sigue funcionando y puede volver a conectar su terminal a él con el comando docker container exec:

  ```
  $ docker container exec -it CONTAINER_ID bash
  ```

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Procesos de los Contenedores

```
$ docker container exec -it CONTAINER_ID bash
```

- El intérprete de comandos del intérprete de órdenes ha cambiado de nuevo al contenedor.
- Si ejecuta el comando ps de nuevo, verá **dos** procesos Bash. Esto se debe a que el comando docker container exec creó un **nuevo** proceso Bash y se adjuntó al mismo.
- En este caso, escribir exit en este shell no terminará el contenedor, porque el proceso original de Bash continuará ejecutándose.
  - Escribe exit para salir del contenedor y verifica que sigue funcionando con un contenedor.

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

```
$ docker container run tomcat:latest sleep 10
```

¿Qué pasará?

```
$ docker container run tomcat:latest /bin/bash
```

¿Qué ocurre?

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

- En caso de que queramos **parar** el contenedor de forma manual:

```
$ docker container stop container_name
   o
$ docker container stop container_Id
```

- Para su **reinicio**:

```
$ docker container start container_name
   o
$ docker container start container_Id
```

- Para **borrar** un contenedor para siempre:

```
$ docker container rm container_name
   o
$ docker container rm container_Id
```

- Para **ver** los contenedores que tenemos:

```
$ docker container ls (sólo activos)
$ docker container ls -a(todos, hay más opciones según lo que nos interese ver)
```

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Ciclo de vida del Container

- Vamos a recorrer todo el ciclo de vida de los contenedores y comprobar la persistencia de datos.

```
$ docker container run --name percy -it tomcat:latest /bin/bash
```

- Ese es nuestro contenedor creado, y lo llamamos "percy" por persistente
- Ahora pongámoslo a trabajar escribiéndole algunos datos.

```
root@9cb2d2fd1d65:/# cd tmp
root@9cb2d2fd1d65:/tmp# ls -l
root@9cb2d2fd1d65:/tmp# echo “Kaixo!!!" > newfile
root@9cb2d2fd1d65:/tmp# ls -l
root@9cb2d2fd1d65:/tmp# cat newfile
```

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Ciclo de vida del Container

- Ctrl-PQ para salir del contenedor sin terminarlo.

```
$ docker container stop percy
$ docker container ls
$ docker container ls –a
$ docker container start percy
$ docker container ls
$ docker container exec -it percy bash
root@9cb2d2fd1d65:/# cd tmp
root@9cb2d2fd1d65:/# ls –l
root@9cb2d2fd1d65:/# cat newfile
```

Verificamos con esto que al parar el contenedor no destruimos ni el contenedor ni los datos que se alojan en él.

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Ciclo de vida del Container

- Ahora terminaremos el contenedor y lo borraremos del sistema.
- Es possible borrar un contenedor en marcha si añadimos el flag **-f** al comando `docker container rm`

  $ docker container rm <container> -f
- De todas formas, es mejor si lo hacemos en dos pasos:
  - primero parar el contenedor
  - luego borrarlo.
- Ej
- Si estamos en el terminal del contenedor (percy), primero debemos ir al terminal del Docker host mediante Ctrl-PQ.

```
$ docker container stop percy
$ docker container rm percy
$ docker container ls –a
```

Intentar levantar de nuevo el contenedor, con la misma imagen y el mismo nombre (percy). ¿Existe el fichero que antes hemos creado?

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Contenedores autorecuperable con políticas de reinicio

- A menudo es una buena idea ejecutar contenedores con una política de reinicio.
- Es una forma de autorecuperación que permite a Docker reiniciarlas automáticamente después de que ocurran ciertos eventos o fallos (flag **--restart**)
- Las políticas de reinicio se aplican por contenedor y pueden configurarse imperativamente en la línea de comandos como parte de los comandos de ejecución de contenedores de docker, o de forma declarativa en archivos Compose para su uso con Docker Compose y Docker Stacks.
- Actualmente, existen las siguientes políticas de reinicio:
  - no: No reiniciar automáticamente el contenedor. (por defecto)
  - on-failure: Reinicia el contenedor si sale debido a un error, que se manifiesta como un código de salida distinto de cero.
  - Siempre/Always: el más sencillo. Reiniciar siempre el contenedor si se detiene. Si se detiene manualmente, se reinicia sólo cuando el demonio Docker se reinicia o el propio contenedor se reinicia manualmente.
  - unless-stopped: similar a always, excepto que cuando el contenedor se detiene (manualmente o de otra manera), no se reinicia incluso después de que el demonio Docker se reinicie.

doc
doc
docker run -it --rm -p 8888:8080 tomcat:9.0

Politeknikoa

Ejemplo Contenedor que ofrece un Servicio Web (para práctica)

- Vamos a poner en marcha un contenedor desde una imagen. La imagen ejecuta un servidor web en el puerto 8080.
- Usa el comando $docker container stop y $docker container rm para borrar todos los contenedores del sistema.

```
$ docker container run -d --name webserver -p 80:8080 tomcat:latest
```

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Ejemplo Servicio Web

- Note que el indicador del terminal no ha cambiado. Esto se debe a que iniciamos este contenedor en segundo plano con el flag **-d**. Iniciar un contenedor en segundo plano no lo conecta a su terminal.
- Otros flags y su significado:
  - -d flag en vez de -it. -d pone en marcha el modo **d**aemon, y le dice al contenedor que lo ejecute en background.
  - -p 80:8080. El flag -p flag mapea puertos en el Docker host a los puertos dentro del contenedor. En el ejemplo, estamos mapeando el puerto 80 del Docker host al puerto 8080 del contenedor. El tráfico que llegue al Puerto 80 del Docker host será direccionado al puerto 8080 dentro del contenedor.
  - También le decimos qué imagen utilizar: `millarramendi/web-docker-ci:pi`

```
$ docker container ls
```

Ahora ya el contenedor está en marcha y los puertos mapeados y podemos conectarnos mediante un web browser a la dirección IP del **Docker host** en el puerto 80.

Docker Containers: Información detallada

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

Inspeccionando containers

Al generar/build una imagen Docker, es possible incrustar un commando para listar la aplicación que quieres que el contenedor utilice al hacer run de esa imagen.

```
$ docker image inspect tomcat
[
    {
        "Id": "sha256:07e574331ce3768f30305519...49214bf3020ee69bba1",
        "RepoTags": [
            “tomcat:latest"
            <Snip>
            ],
            "Cmd": [
              "/bin/sh",
              "-c",
              "#(nop)  CMD  [\"/bin/sh\"  \"-c\"  \"cd  /src  \u0026\u0026
node \
              ./app.js\"]"
            ],
<Snip>
```

Docker Containers: Los comandos

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

- **`$ docker container run:`** pone en marcha nuevos contenedores. En su forma más simple, acepat una imagen y un comando como argumento. La imagen que se usa para crear el contenedor y el comando es la aplicación que queremos que el contenedor ponga en marcha.
  - El siguiente ejemplo pondrá en marcha un contenedor Ubuntu ejecutando como aplicación su terminal: **$ docker container run –it ubuntu /bin/bash.**
- **`Ctrl-PQ:`** separa su shell del terminal de un contenedor y deja el contenedor en funcionamiento (UP) en segundo plano.
- **`$ docker container ls:`** lista todos los contenedores que están en marcha (UP) . Si añadimos el flag -a flag veremos también contenedores que están parados/stopped (Exited).
- **`$ docker container exec:`** permite ejecutar un nuevo proceso dentro del contenedor que ya está en marcha. Es muy útil para enlazar por ejemplo el terminal del contenedor con el terminal del Docker host
  - Este comando iniciará un nuevo terminal Shell dentro del contenedor y lo conectará al terminal del Docker host:

`$ docker container exec –it container-name or container-id bash`.

Docker Containers: Los comandos

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

- **`$ docker container stop:`** detiene un contenedor en marcha y lo lleva al estado de Exited (0). Para ello, emite un SIGTERM al proceso con PID 1 dentro del contenedor. Si el proceso no se ha limpiado y detenido en 10 segundos, se emitirá un SIGKILL para detener por la fuerza el contenedor. docker container stop acepta IDs de contenedor y nombres de contenedor como argumentos
- **`$docker container start:`** reiniciará un contenedor detenido (Exited). Al iniciar, le puede pasar como argumento el nombre o ID del contenedor.
- **`$ docker container rm:`** borrará un contenedor parado (Stopped). Se puede especificar el contenedor mediante su nombre o ID. Se recomienda parar el contenedor mediante el comando docker container stop antes de borrar (docker container rm).
- **`$ docker container inspect:`** muestra la configuración detallada e información de tiempo de ejecución del contenedor. Acepta como argumento los nombers e IDs del contenedor.
- **`$ docker container logs:`** recoge los logs del contenedor.

Docker Containers: Práctica

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

1. Siguiendo las instrucciones de las notas expuestas en clase:
   - Poner en marcha un contenedor desde una imagen. La imagen ejecuta un servidor web en el puerto 8080 (ejemplo de clase).

   $ docker run --name webstatic -d -p 8888:80 dockersamples/static-site

   - Una vez que el contenedor está en marcha y los puertos mapeados conectar mediante un web browser a la dirección IP del **Docker host** en el puerto 8888. ¿Cuál es la respuesta?
   - Parar y borrar todos los contenedores del sistema.

2. Contenedores interactivos
   - Crear un contenedor utilizando la imagen centos:7 y conectar su bash terminal en modo interactivo
   - Verificar el sistema de ficheros del contenedor y crea un nuevo fichero.
     - # echo 'Hello there...' > test.txt
   - Sal del contenedor mediante el comando exit.
   - Ejecuta el siguiente comando: $ docker container run -it centos:7 bash
   - Trata de encontrar el fichero test.txt en este nuevo contenedor. Sal de este contenedor de la misma forma que saliste del anterior.
3. Reconectar Contenedores
   - Lista todos los contenedores que tiene el Docker host (también los que están Stopped)

Docker Containers: Práctica

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

3. Reconectar Contenedores
   - Reinicia el contenedor que creaste primero utilizando su CONTAINER ID (la que tiene la imagen centos).
   - Verifica que el contenedor está en marcha
   - Chequea si existe el fichero test.txt y verifica su contenido.
   - Ejecuta: $ docker container ls -a --no-trunc. ¿Cuál es su output? ¿Qué nos aporta el flag –no-trunc?
   - Listar los contenedores pero solo mostrando su ID.
   - Listar el ultimo contenedor que hemos creado.
   - Listar los contenedores que están en estado EXIT mediante el flag –filter.
   - Borra todos los contenedores.
4. Contenedores separados y Logeo
   - Poner en marcha un nuevo contenedor con la imagen centos:7 y lanza el commando ping al 127.0.0.1
   - Inspeccionar los logs generados por este contenedor mediante el comando log
   - Inspeccionar la información de este contenedor mediante el comando inspect

Docker Containers: Práctica

Mondragon Unibertsitatea
Goi Eskola Politeknikoa

5. Borrar Contenedores
   - Pon en marcha tres contenedores en modo background.
   - Lista los contenedores que estan en estado exit (mediante filter)
   - Borra los contenedores que estan parados (exit) y confirma que están borrados
   - Ahora, haz lo mismo pero con los contenedores que están en marcha. En este caso, existen diferentes opciones para hacerlo. Describe cada una de ellas.
   - Borra todos los contenedores que existen en el Sistema.

Mondragon Unibertsitatea

Goi Eskola Politeknikoa

Eskerrik asko

Muchas gracias

Thank you