

Introducción a Docker

# UD 05. Caso práctico

## 03 - Acelerando juegos de prueba con “tmpfs”

---



Autor: Sergi García Barea

Actualizado Febrero 2025


## Licencia



**Reconocimiento – NoComercial - CompartirIgual (BY-NC-SA):** No se permite un uso comercial de la obra original ni de las posibles obras derivadas, la distribución de las cuales se debe hacer con una licencia igual a la que regula la obra original.

## Nomenclatura

A lo largo de este tema se utilizarán distintos símbolos para distinguir elementos importantes dentro del contenido. Estos símbolos son:

 **Importante**

 **Atención**

 **Interesante**

1. Introducción	3
2. Paso 1: preparando el contenedor MySQL	3
3. Paso 2: Probando el contenedor con MySQL	4

## UD05. CASO PRÁCTICO 03

### 1. INTRODUCCIÓN

El uso de sistemas de ficheros en memoria como *“tmpfs”* o *“ramfs”* es recomendable cuando se quiere obtener una mayor velocidad de entrada/salida, a costa de la persistencia. Esto es útil cuando un servidor debe usar un fichero, para acelerar los test de juegos de pruebas, etc.

**! Atención:** es posible que si tienes un disco SSD, no puedas notar mejoría en este caso práctico.

En este caso práctico, vamos a preparar un servidor MySQL cuya información estará en memoria (con un volumen *“tmpfs”*). Este servidor MySQL estará pensado como un contenedor que se iniciará y estará disponible para probar una aplicación.

Utilizaremos para este propósito una base de datos de test *“dummy”* que se adjunta al caso práctico. Esta ha sido obtenida usando <http://filldb.info/dummy>.

La imagen en la que nos basaremos será [https://hub.docker.com/\\_/mysql](https://hub.docker.com/_/mysql)

### 2. PASO 1: PREPARANDO EL CONTENEDOR MySQL

En primer lugar, supondremos que la base de datos la hemos descargado en el directorio *“/home/sergi/dump.sql”*. A continuación proponemos dos comandos, uno para utilizar el contenedor sin *“tmpfs”* y otro para cargar la información de MySQL en volumen *“tmpfs”*.

#### Contenedor sin *“tmpfs”*:

```
docker run -d --rm --name mysqlsintmpfs -p 3306:3306 -v
/home/sergi/dump.sql:/docker-entrypoint-initdb.d/dump.sql -e
MYSQL_ALLOW_EMPTY_PASSWORD=TRUE -e MYSQL_USER=cefireuser -e MYSQL_PASSWORD=cefirepass
mysql:5.6
```

Hemos creado un contenedor que se eliminará una vez parado (parámetro *--rm*), con el puerto 3306 del contenedor mapeado a la máquina anfitrión.

Algunas cosas concretas de MySQL:

- Mapeamos el fichero *“dump.sql”* que contiene la base de datos, al directorio *“/docker-entrypoint-initdb.d”*. El motivo de esto, es que tal como indica la ayuda de la imagen de Docker *“mysql”*, al crearse el contenedor, todo fichero *“.sh”*, *“.sql”* y *“.sql.gz”* será procesado en orden alfabético.
  - Con esto conseguimos que al iniciar el contenedor se rellene la base de datos, lanzando el SQL contenido en el fichero *“dump.sql”*.
- La variable de entorno **MYSQL\_ALLOW\_EMPTY\_PASSWORD** indica que el usuario root no tiene password.
- Las variables de entorno de usuario y password, crean un usuario con permisos de root para poder utilizarlo en conexiones remotas.

### Contenedor usando “tmpfs”:

```
docker run -d --rm --name mysqlcontmpfs -p 3307:3306 -v  
/home/sergi/dump.sql:/docker-entrypoint-initdb.d/dump.sql --tmpfs  
/var/lib/mysql:rw,noexec,nosuid,size=1024m -e MYSQL_ALLOW_EMPTY_PASSWORD=TRUE -e  
MYSQL_USER=cefireuser -e MYSQL_PASSWORD=cefirepass mysql:5.6
```

Similar al anterior, solo que el directorio donde se almacena la información de MySQL, **“/var/lib/mysql”** lo almacenamos en memoria con un volumen que usa **“tmpfs”**.

### 3. PASO 2: PROBANDO EL CONTENEDOR CON MySQL

Antes de comenzar, es recomendable esperar un minuto o dos desde la creación del contenedor para asegurarnos que la base de datos se ha llenado correctamente.

Para acceder a la base de datos, aunque podemos acceder con los puertos expuestos (usando usuario **“cefireuser”** y password **“cefirepass”**) y ese sería el caso que usaría una aplicación que quiere usar esta base de datos como test, para simplificar el caso práctico nosotros entraremos en el contenedor utilizando los siguientes comandos:

#### Para acceder a la versión sin “tmpfs”:

```
docker exec -it mysqlsintmpfs bash
```

#### Para acceder a la versión con “tmpfs”:

```
docker exec -it mysqlcontmpfs bash
```

Una vez dentro mediante **“docker exec”**, podemos conectarnos usando:

```
mysql -u root
```

Nos permitirá acceder sin password, ya que definimos la variable de entorno que lo permitía durante la creación del contenedor.

Ahí se pueden lanzar consultas para apreciar la diferencia de rendimiento. Según la máquina y el sistema operativo, esta diferencia puede ser pequeña o imperceptible, o notable.

Una secuencia de prueba dentro del cliente MySQL podría ser:

```
USE test;  
SELECT SQL_NO_CACHE * FROM posts;
```

Con esta consulta, usamos la base de datos **“test”** y con **SQL\_NO\_CACHE** indicamos que no utilice la caché en la consulta. Esta consulta obtiene 11100 resultados.

En el momento de realizar estas pruebas, el autor del documento con **“tmpfs”** tardaba 0.02 segundos y sin **“tmpfs”** 0.04 segundos.

**! Atención:** para simplificar el ejemplo las pruebas son de lectura de datos. Pero sobre todo se nota un incremento de rendimiento en contextos de escritura de datos.