

# Consideraciones de seguridad relativas al uso y despliegue de Lightning en los laboratorios docentes de la UC3M

Pablo Toribio, 15 de julio de 2018

## Descripción

Este documento detalla la configuración de seguridad que debe utilizarse para el despliegue y uso del simulador de redes Lightning. Para entenderlo se requiere un conocimiento superficial de Docker y de los espacios de nombres (namespaces) del kernel Linux.

## Restricciones

El simulador de redes Lightning (<https://github.com/ptoribi/lightning>) fue diseñado con dos características que tienen implicaciones en la forma en que se ejecuta Docker y en su configuración de seguridad:

**[1]** Al ser ejecutado, el simulador comparte entre el SO anfitrión y cada uno de los containers de tipo host un **directorio común** (se creará dentro del directorio **home** del usuario que arrancó la aplicación, tendrá como nombre predeterminado **lightning-shared-folder**, y se montará en el directorio **/home** de cada uno de los containers), en el que tanto el usuario del SO anfitrión que arrancó el simulador como el usuario de los containers deben poder escribir con los **mismos identificadores de usuario (UID) y grupo primario (GID)**. Estos identificadores deberán ser los pertenecientes al usuario que arrancó el simulador, de forma que pueda escribir y manipular ficheros sobre el directorio compartido sin problemas y de la misma forma que lo haría sobre cualquier otro directorio de su home de usuario.

**[2]** El simulador lanza una ventana de comandos por cada uno de los containers arrancados. En el caso de las máquinas de tipo host dicha ventana deberá ejecutar la shell de comandos **bash** y ser iniciada por un usuario con los **mismos identificadores de usuario (UID) y grupo primario (GID)** que el usuario que arrancó el simulador.

Las características **[1]** y **[2]** necesitan, por tanto, la restricción siguiente: el usuario del container Docker deberá tener la misma configuración **UID:GID** que el usuario del SO anfitrión que arrancó el simulador.

# Seguridad en Docker

## 1.- Aislamiento de containers mediante el uso de espacios de nombres

De forma predeterminada la plataforma Docker a la vez que arranca un container crea determinados espacios de nombres asociados al mismo con el fin de aislar sus recursos (procesos, pilas de red, etc.) de forma que estos no puedan ver o afectar a recursos de otro container o del sistema operativo anfitrión.

De forma general el sistema operativo anfitrión sí puede ver los recursos de otros espacios de nombres, pero no al contrario.

### 1.1- El espacio de nombres de usuario

Uno de los espacios de nombres más interesantes es el **espacio de nombres de usuario**, que en concreto aísla los identificadores de usuarios (UIDs) y grupos (GIDs).

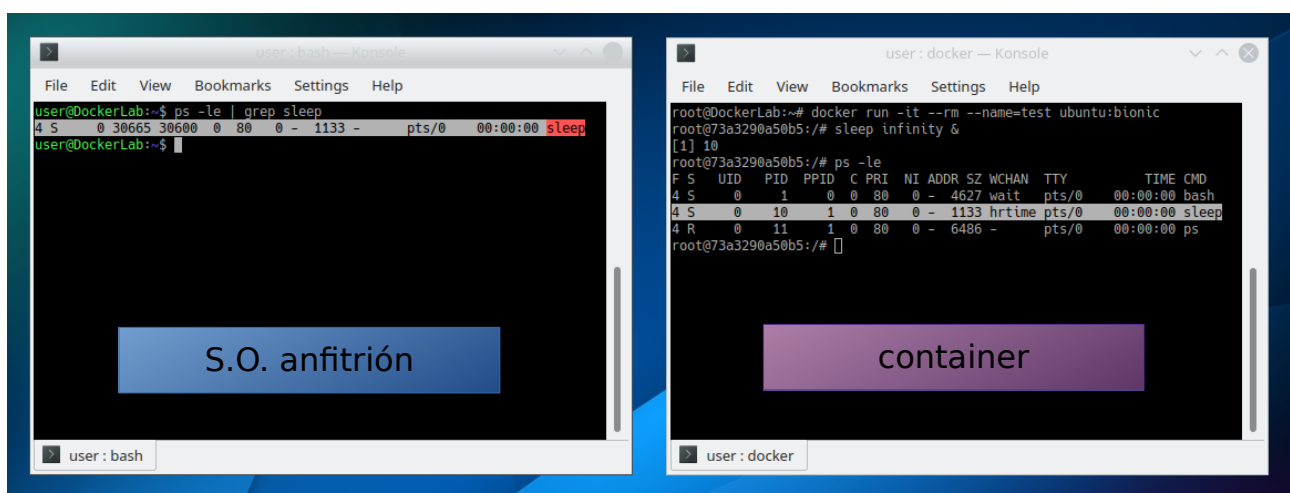
Además de dicho aislamiento, se puede configurar el demonio de Docker para que utilice un **mapeo**, de forma que un ID de usuario/grupo de un container sea visto por el SO anfitrión como otro ID completamente diferente.

#### 1.1.1- Ejecución sin mapeo de IDs

Supongamos que se arranca un container Docker de nombre “test”, que se accede a él como usuario **root**, y que dentro de dicho container se ejecuta el proceso “sleep”.

```
root@DockerLab:~# docker run -it --rm -v /tmp:/tmp --name=test ubuntu:bionic
root@73a3290a50b5:/# sleep infinity &
```

En la siguiente imagen se puede observar cómo el proceso “sleep” tiene **UID=0** tanto en el container como a los ojos del sistema operativo anfitrión:



Este proceso no podrá ser visto por otro container ejecutándose en la misma máquina (gracias al aislamiento del namespace de procesos), pero sí por el sistema operativo anfitrión, y además con el mismo ID de usuario, dado que el mapeo no ha sido configurado.

### 1.1.2- Ejecución con mapeo de IDs

Supongamos que ahora se desea configurar el mapeo de IDs, para ello se deben editar dos ficheros de forma obligatoria y uno de forma opcional:

- **/etc/subuid** → mapeo de IDs de usuario (usuario:inicio:longitud\_rango).
- **/etc/subgid** → mapeo de IDs de grupo (grupo:inicio:longitud\_rango).
- **/etc/docker/daemon.json** (opcional)→ especifica el mapeo de UIDs y GIDs que utilizará de forma predeterminada el demonio de Docker al arrancar (independientemente del usuario que cree el container).

Supongamos que el sistema operativo anfitrión tiene un usuario y un grupo llamado HomerSimpson, y que para dicho usuario y grupo se desea establecer un mapeo de IDs.

El fichero **/etc/subuid** deberá contener al menos:

```
HomerSimpson:100000:65536
```

El fichero **/etc/subgid** deberá contener al menos:

```
HomerSimpson:100000:65536
```

El fichero **/etc/docker/daemon.json** (opcional) deberá contener al menos:

```
{  
  "userns-remap": "HomerSimpson:HomerSimpson"  
}
```

En caso de que hayamos creado el fichero daemon.json, bastará con reiniciar el servicio de Docker:

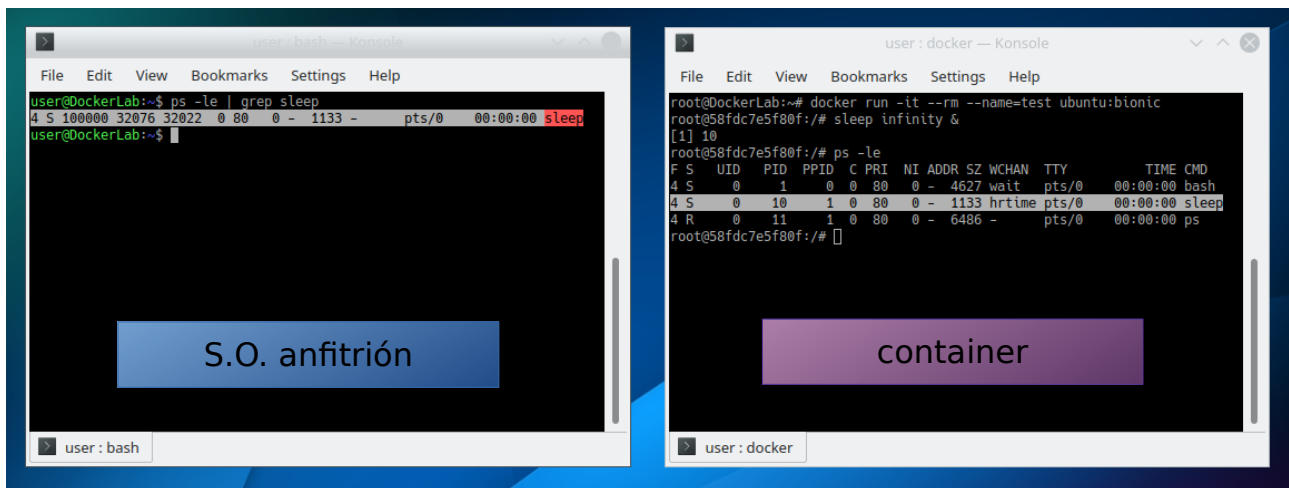
```
root@DockerLab:~# service docker restart
```

En caso contrario, deberemos para el demonio de Docker y especificar el mapeo de usuario y grupo de forma manual al volver a ejecutarlo:

```
root@DockerLab:~# service docker stop
```

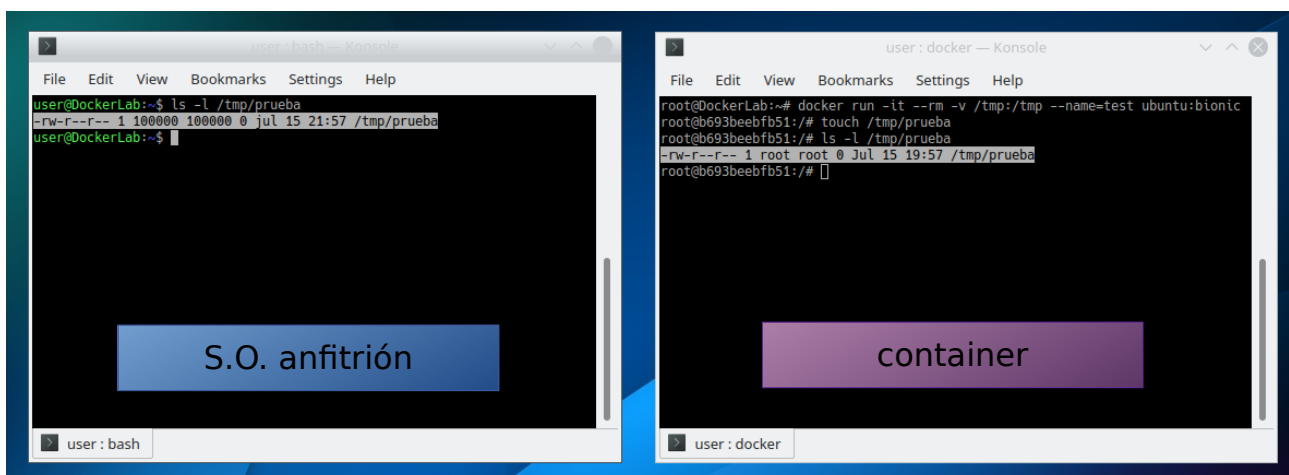
```
root@DockerLab:~# dockerd --userns-remap="HomerSimpson:HomerSimpson"
```

Supongamos que una vez efectuadas en el sistema las configuraciones se reproduce el mismo ejemplo anterior. El resultado se muestra en la siguiente imagen, donde se puede observar que el proceso sleep tiene un **UID=0** en el container pero ha sido mapeado a un **UID=100000** en el sistema operativo anfitrión:



Por supuesto el mapeo de IDs no afecta sólo a procesos, sino también a cualquier fichero creado por el usuario.

En la siguiente imagen se muestra el resultado de crear un fichero en el directorio **/tmp** (accedido conjuntamente por el container y el sistema operativo anfitrión). Desde el container se ve el fichero creado por el usuario y grupo 0:0 (root:root), mientras que desde el sistema operativo anfitrión el fichero se ve como creado por el usuario y grupo 100000:100000, debido al mapeo de IDs:



## 2.- Restricción de recursos mediante el uso de grupos de control (cgroups)

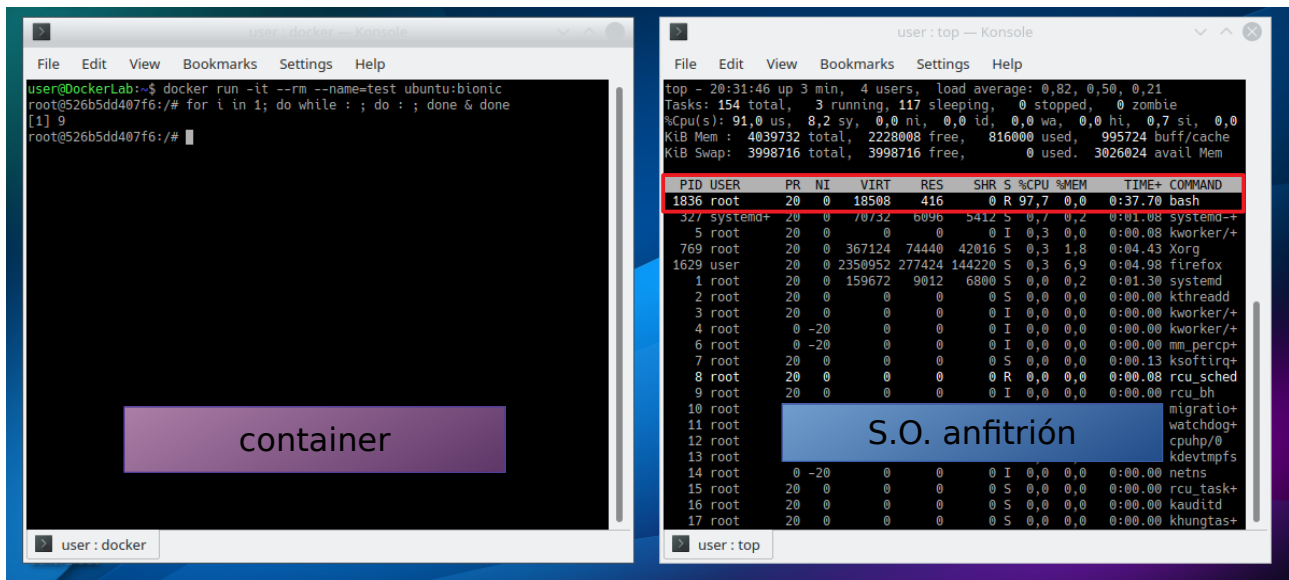
Los grupos de control o cgroups establecen limitaciones a los recursos hardware que puede emplear un container Docker.

De forma predeterminada cuando un container es creado a menos que se especifiquen unas determinadas restricciones, este ve como disponibles todas las CPUs y memoria de la máquina anfitrión, sin limitación ninguna.

Veamos un ejemplo en el que en una máquina que dispone de una única CPU de un sólo núcleo se crea un container en el que se ejecuta un proceso que intenta monopolizar el uso de la CPU.

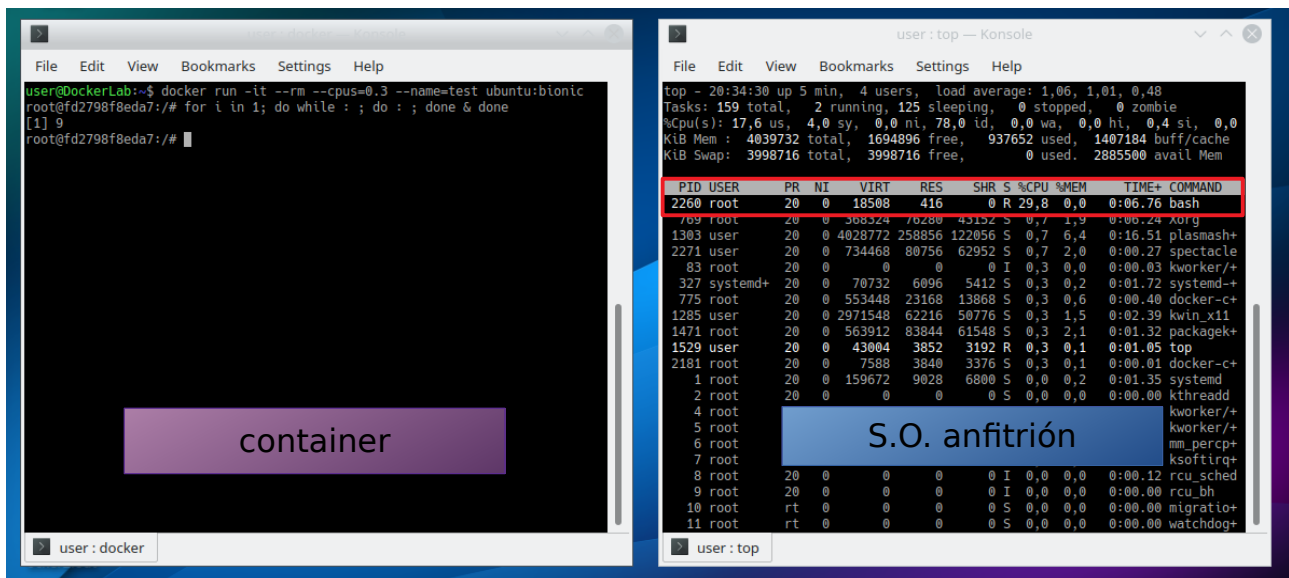
En primer lugar, sin especificar restricciones al crear el container:

```
root@DockerLab:~# docker run -it --rm --name=test ubuntu:bionic
root@526b5dd407f6:/# for i in 1; do while : ; do : ; done & done
```



En segundo lugar, se realiza la misma operación pero especificando un límite de uso de la CPU en ese container del 30%:

```
root@DockerLab:~# docker run -it --rm --cpus=0.3 --name=test ubuntu:bionic
root@fd2798f8eda7:/# for i in 1; do while : ; do : ; done & done
```



Lightning no impone restricciones hardware mediante grupos de control a los containers que crea, de forma que estos pueden emplear tanta memoria y ciclos de CPU como requieran, por lo que en todo caso un usuario podría bloquear la máquina creando en un container un proceso que haga un uso excesivo de recursos hardware.

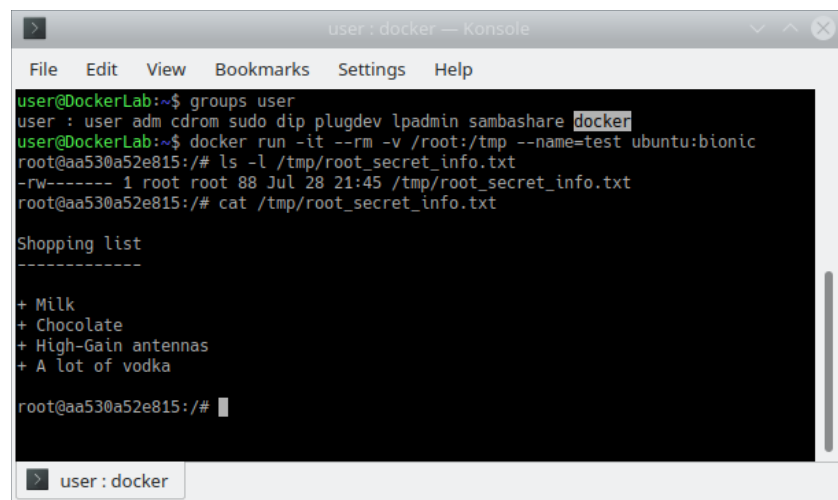
Las máquinas que se emplean en los Laboratorios de Ingeniería Telemática de la UC3M son estaciones de trabajo y no servidores compartidos, por lo que el hecho de que un usuario malintencionado pudiera bloquear momentáneamente una máquina aprovechando la falta de restricciones hardware no supondría un riesgo para la seguridad ni dejaría sin capacidad un recurso productivo.

### 3.- Usuarios que pueden hacer peticiones al demonio de Docker

Además del usuario **root**, cualquier usuario perteneciente al grupo “**docker**” puede hacer peticiones al demonio de Docker con privilegios plenos. Esto es especialmente peligroso dado que dicho usuario podría crear containers con argumentos tales como **-v** que permite montar cualquier directorio del sistema operativo anfitrión en el container, o **--cap-add** que permite darle acceso administrativo (capabilities) al container para ejecutar determinadas acciones, entre otros.

En el siguiente ejemplo, el usuario **user**, miembro del grupo **docker**, crea un container en el que se monta en su directorio **/tmp** el directorio **/root** del sistema operativo anfitrión, con las consiguientes consecuencias que ello supone:

```
root@DockerLab:~# docker run -it --rm -v /root:/tmp --name=test ubuntu:bionic
```



```
user@DockerLab:~$ groups user
user : user adm cdrom sudo dip plugdev lpadmin sambashare docker
user@DockerLab:~$ docker run -it --rm -v /root:/tmp --name=test ubuntu:bionic
root@aa530a52e815:/# ls -l /tmp/root_secret_info.txt
-rw----- 1 root root 88 Jul 28 21:45 /tmp/root_secret_info.txt
root@aa530a52e815:/# cat /tmp/root_secret_info.txt

Shopping list
-----

+ Milk
+ Chocolate
+ High-Gain antennas
+ A lot of vodka

root@aa530a52e815:/#
```

Lightning no

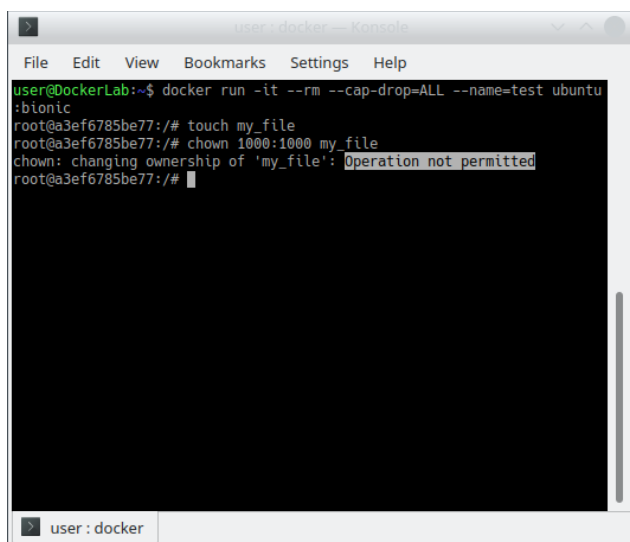
requiere que los usuarios que lo ejecuten pertenezcan al grupo “**docker**”, sino que en cada ejecución el programa hace las peticiones al demonio de Docker mediante **sudo**. Esto permite que a pesar de emplear **sudo**, Lightning cree los containers correspondientes con unos argumentos seguros y muy restringidos, y que fuera del programa el usuario no pueda emplear Docker de forma libre.

## 4.- Asignación de capabilities a un container

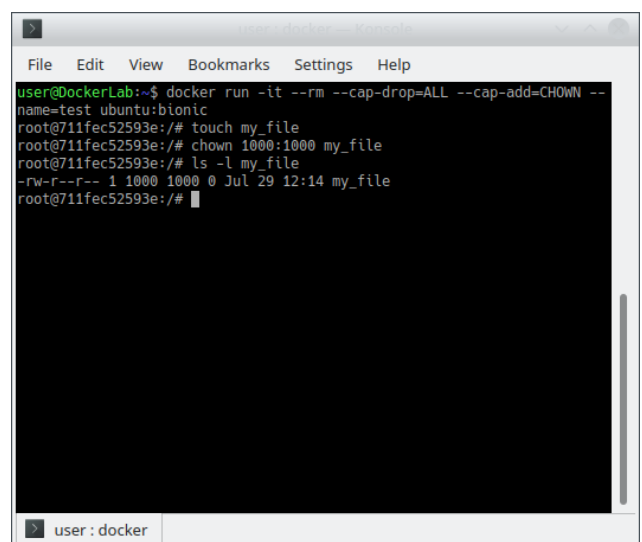
La asignación de **capabilities** permite la ejecución de ciertas acciones con privilegios administrativos (también llamados de root o de superusuario). Lightning intenta, en cuanto a seguridad se refiere, seguir siempre el **principio de mínimo privilegio**: al crear cada container en primer lugar se eliminan todas las capabilities que se conceden por defecto (`--cap-drop=ALL`), y a continuación se asignan, una por una, las estrictamente necesarias (`--cap-add`).

```
--cap-drop=ALL '#ptoribi: drop all capabilities in order to have full control by adding one by one.'\n--cap-add=CHOWN '#Make arbitrary changes to file UIDs and GIDs (see chown(2)).'\n--cap-add=KILL '#Bypass permission checks for sending signals.'\n--cap-add=SETGID '#Make arbitrary manipulations of process GIDs and supplementary GID list.'\n--cap-add=SETUID '#Make arbitrary manipulations of process UIDs.'\n--cap-add=NET_ADMIN '#Perform various network-related operations.'\n--cap-add=NET_RAW '#Use RAW and PACKET sockets.'\n
```

En el siguiente ejemplo se crea un container en el que se crea un fichero y posteriormente se intenta cambiar su propietario (UID:GID); en primer lugar sin asignar capabilities al container, y en segundo lugar asignando la capability CHOWN:



```
user@DockerLab:~$ docker run -it --rm --cap-drop=ALL --name=test ubuntu :bionic
root@a3ef6785be77:/# touch my_file
root@a3ef6785be77:/# chown 1000:1000 my_file
chown: changing ownership of 'my_file': Operation not permitted
root@a3ef6785be77:/#
```



```
user@DockerLab:~$ docker run -it --rm --cap-drop=ALL --cap-add=CHOWN --name=test ubuntu:bionic
root@711fec52593e:/# touch my_file
root@711fec52593e:/# chown 1000:1000 my_file
root@711fec52593e:/# ls -l my_file
-rw-r--r-- 1 1000 1000 0 Jul 29 12:14 my_file
root@711fec52593e:/#
```

## 5.- Seguridad del sistema ejecutándose en el container

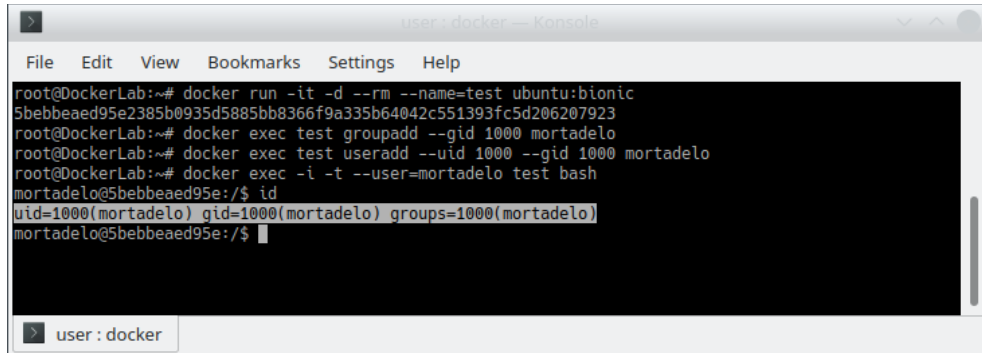
A pesar de que teóricamente un container está aislado del SO anfitrión gracias a chroot, es una buena práctica emplear técnicas de **hardening** en el sistema operativo del container.

La técnica más elemental consiste en que el usuario logueado en el container no sea el root (0:0), sino un usuario normal (bajamente privilegiado). Además se deberá cuidar que dicho usuario no pueda elevar sus privilegios logueándose como root o empleando el comando “sudo” de forma indiscriminada.

En el caso de que la versión de Docker que se está empleando tuviera alguna vulnerabilidad que permitiera al usuario “escapar” del container y crear procesos en el sistema operativo anfitrión, estos serían ejecutados con bajos privilegios y sin suponer un problema para la seguridad.

En el siguiente ejemplo se arranca un container Docker, se crea en él un usuario no privilegiado y se accede al container mediante dicho usuario:

```
root@DockerLab:~# docker run -it -d --rm --name=test ubuntu:bionic
root@DockerLab:~# docker exec test groupadd --gid 1000 mortadelo
root@DockerLab:~# docker exec test useradd --uid 1000 --gid 1000 mortadelo
root@DockerLab:~# docker exec -i -t --user=mortadelo test bash
mortadelo@5bebbaed95e:/$
```

A screenshot of a terminal window titled "user: docker — Konsole". The terminal shows a series of commands executed from a root shell on a host named "DockerLab". The commands are: "docker run -it -d --rm --name=test ubuntu:bionic", "docker exec test groupadd --gid 1000 mortadelo", "docker exec test useradd --uid 1000 --gid 1000 mortadelo", and "docker exec -i -t --user=mortadelo test bash". The output shows the container ID "5bebbaed95e2385b0935d5885bb8366f9a335b64042c551393fc5d206207923". After the final command, the prompt changes to "mortadelo@5bebbaed95e:/\$". The user then runs "id", which outputs "uid=1000(mortadelo) gid=1000(mortadelo) groups=1000(mortadelo)". The terminal window has a menu bar with "File", "Edit", "View", "Bookmarks", "Settings", and "Help". A status bar at the bottom shows "user: docker".

```
user: docker — Konsole
File Edit View Bookmarks Settings Help
root@DockerLab:~# docker run -it -d --rm --name=test ubuntu:bionic
5bebbaed95e2385b0935d5885bb8366f9a335b64042c551393fc5d206207923
root@DockerLab:~# docker exec test groupadd --gid 1000 mortadelo
root@DockerLab:~# docker exec test useradd --uid 1000 --gid 1000 mortadelo
root@DockerLab:~# docker exec -i -t --user=mortadelo test bash
mortadelo@5bebbaed95e:/$ id
uid=1000(mortadelo) gid=1000(mortadelo) groups=1000(mortadelo)
mortadelo@5bebbaed95e:/$
```

Lightning cuida celosamente este detalle, accediendo a los containers Docker mediante un usuario creado previamente en ellos que coincide en nombre, UID y GID con el usuario del SO anfitrión que lanzó la aplicación. Además a ese usuario sólo se le otorga permiso para ejecutar como root ciertos comandos no considerados peligrosos: ip, ifconfig, etc.

## Conclusiones



<https://docs.docker.com/engine/security/security/>  
<https://docs.docker.com/engine/security/usersns-remap/>  
<https://docs.docker.com/engine/reference/commandline/dockerd/>  
<https://success.docker.com/article/introduction-to-user-namespaces-in-docker-engine>  
[https://docs.docker.com/config/containers/resource\\_constraints/](https://docs.docker.com/config/containers/resource_constraints/)  
<https://www.certs.es/blog/linux-capabilities>  
<http://man7.org/linux/man-pages/man7/namespaces.7.html>  
[http://man7.org/linux/man-pages/man7/user\\_namespaces.7.html](http://man7.org/linux/man-pages/man7/user_namespaces.7.html)  
<http://man7.org/linux/man-pages/man5/subuid.5.html>  
<http://man7.org/linux/man-pages/man5/subgid.5.html>