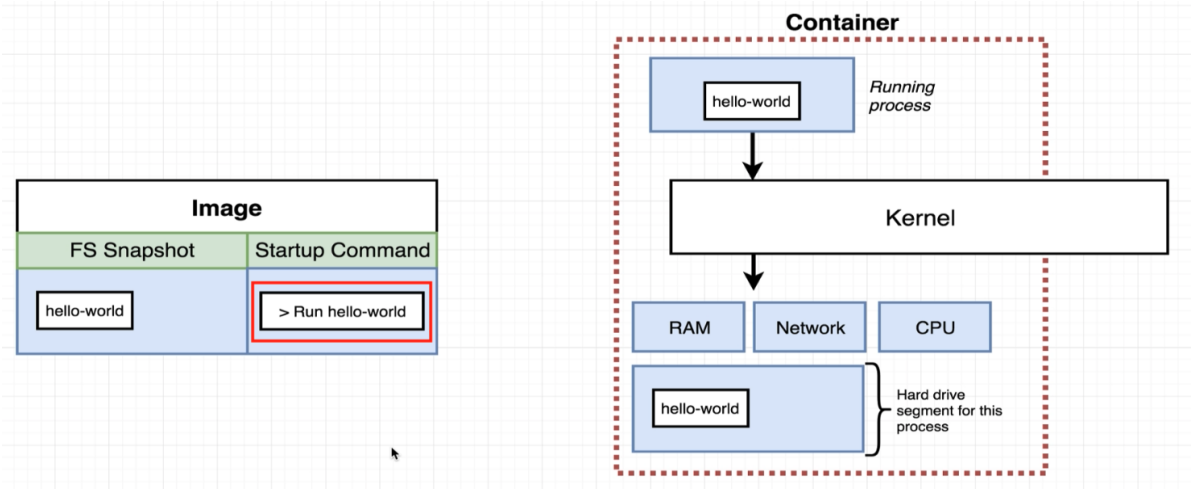
Docker run – Anular el comando por default

Ahora vamos ahora a ver el comando docker run y algunas variaciones de este.

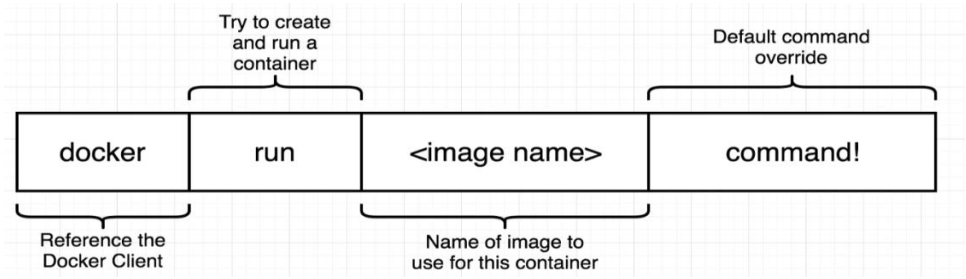
Antes de ver el siguiente tipo de variaciones de este comando, vamos a recapitular rápidamente lo que sucede detrás de escenas con el comando docker run.

Recordemos que, cada vez que ejecutamos el comando *docker run* con una imagen, no sólo obtenemos esa *snapshot*, sino que también obtenemos la ejecución de este comando *run* que se supone se debe ejecutar después de que se crea el contenedor:



Por lo tanto, la variación del comando *docker run* que vamos a ver nos permitirá anular o hacer *override* este comando *run* por default.

Así es como lo haremos:



Ejecutamos *docker run*, el nombre de la imagen, y después de eso añadimos un comando alternativo que se ejecutará dentro del contenedor después de que se inicie. Esto es un *override*, por lo tanto, cualquier comando por default que se incluya dentro de la imagen no se ejecutará.

Vamos a ejecutar entonces el comando *docker run*, pero esta vez utilizaremos una imagen diferente llamada *busybox*:

```
~ % docker run busybox
```

Después de este comando, añadimos el comando alternativo para que se ejecute dentro del contenedor después de que se crea. Así que añadimos el comando *echo hi there*:

```
~ % docker run busybox echo hi there
```

Esto de aquí es el override:

```
~ % docker run busybox echo hi there
```

El comando echo va a imprimir "*hi there*" en nuestra terminal.

Si ejecutamos el comando, podemos ver el resultado en nuestra terminal:

```
jorge@MacBook-Pro-de-Jorge ~ %
jorge@MacBook-Pro-de-Jorge ~ % docker run busybox echo hi there
hi there
jorge@MacBook-Pro-de-Jorge ~ %
```

Ahora, vamos a ver otra variación de este comando, pero hagámoslo un poco más interesante. Vamos a añadir como override el comando *ls*:

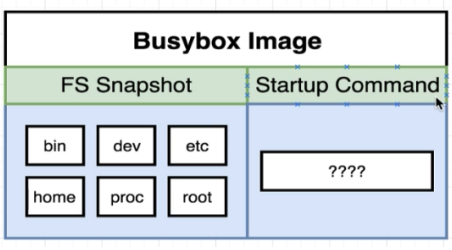
```
jorge@MacBook-Pro-de-Jorge ~ % docker run busybox ls
```

El comando *ls* nos permite imprimir todos los archivos y carpetas dentro de un directorio. Asi que, si ejecutamos el comando, podremos ver el resultado en la terminal:

```
jorge@MacBook-Pro-de-Jorge ~ % docker run busybox ls
bin
dev
etc
home
lib
lib64
proc
root
sys
tmp
usr
var
jorge@MacBook-Pro-de-Jorge ~ %
```

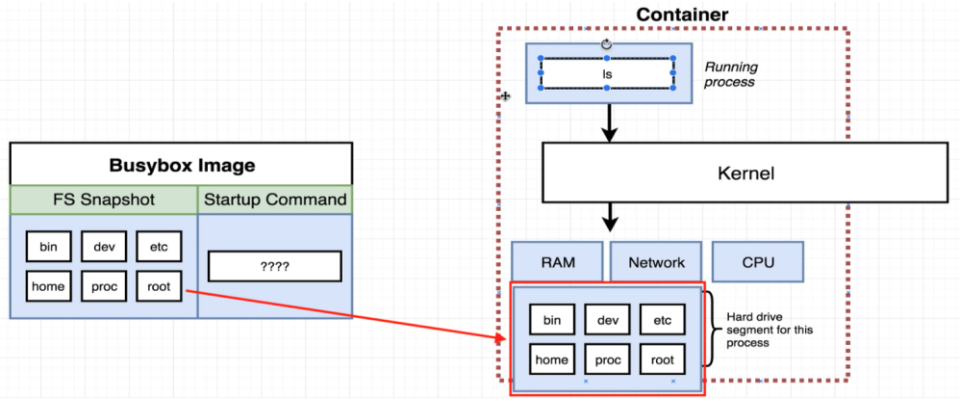
Si usted está en Windows, este listado de carpetas puede ser confuso, ya que este es un listado de carpetas que existen en el contenedor dentro del cual ejecutamos el comando *ls*.

Recordemos qué ocurre exactamente cuándo creamos un contenedor a partir de una imagen. Aquí tenemos la imagen *busybox*:



Esta imagen tiene un snapshot de un sistema de archivos por default y algún comando que se ejecuta cuando el contenedor es creado.

La imagen *busybox* contiene las carpetas que vimos previamente, al ejecutar el comando *ls*. Así que, cuando creamos un contenedor a partir de esa imagen, tomamos ese snapshot del sistema de archivos y lo metemos como carpetas en el nuevo contenedor:



Entonces, al ejecutar el comando *ls*, lo que se imprime es un listado de archivos que existen dentro del contenedor, y ese sistema de archivos es precisamente que se copia de la imagen hacia el contenedor.

Ahora, ¿por qué usamos la imagen *busybox* en lugar de la imagen *hello-world* que estábamos usando antes? Bueno, vamos a la terminal y trataremos de ejecutar la imagen *helo-world* con el comando *ls*:

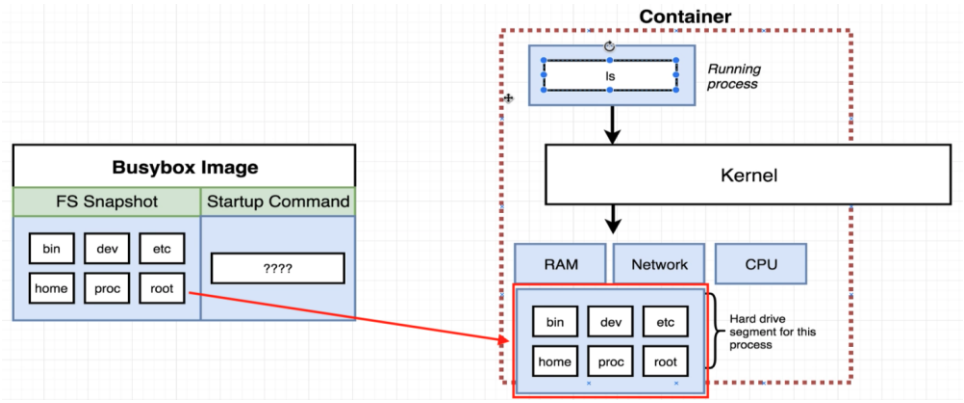
```
jorge@MacBook-Pro-de-Jorge ~ % docker run hello-world ls
docker: Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container proc
ess: exec: "ls": executable file not found in $PATH: unknown.
jorge@MacBook-Pro-de-Jorge ~ %
```

Puede ver que en este caso aparece un mensaje de error. Ahora vamos a ejecutar el comando echo con esta misma imagen:

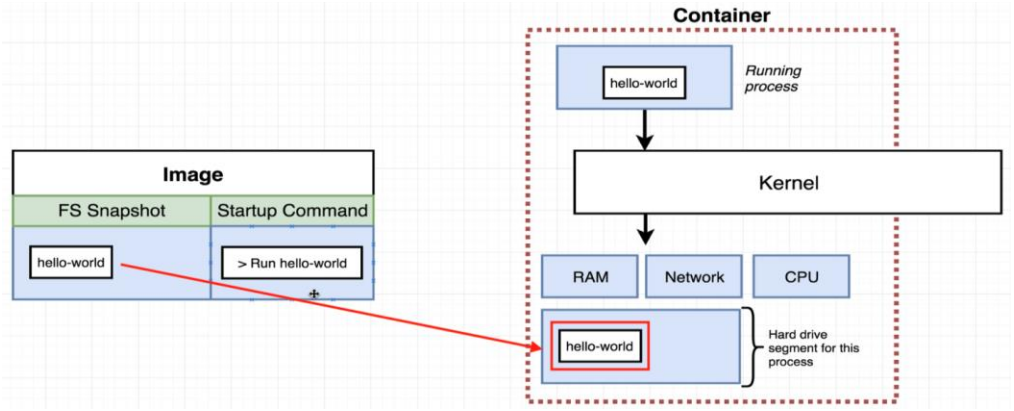
```
jorge@MacBook-Pro-de-Jorge ~ % docker run hello-world echo hi there
docker: Error response from daemon: failed to create shim task: OCI runtime create failed: runc create
 failed: unable to start container process: exec: "echo": executable file not found in $PATH: unknown.
ERRO[0000] error waiting for container: context canceled
jorge@MacBook-Pro-de-Jorge ~ %
```

Podemos ver que también aparece un mensaje de error similar. Entonces, ¿qué es lo que pasa?

Cuando ejecutamos los comandos alternos echo y ls con la imagen busybox, esos comandos funcionan porque son dos programas que existen dentro de la snapshot del sistema de archivos:



Sin embargo, con nuestra imagen hello-world, lo único que existe dentro de esa snapshot del sistema de archivos es un solo programa, un solo archivo que todo lo que hace es hacer eco o imprimir un mensaje específico cuando se inicia el contenedor:



Por lo tanto, estos comandos alternativos que estamos ejecutando se basan en el sistema de archivos incluido con la imagen, y si tratamos de ejecutar un comando dentro del contenedor que utiliza un programa que no está contenido en el sistema de archivos, entonces tendremos un error.