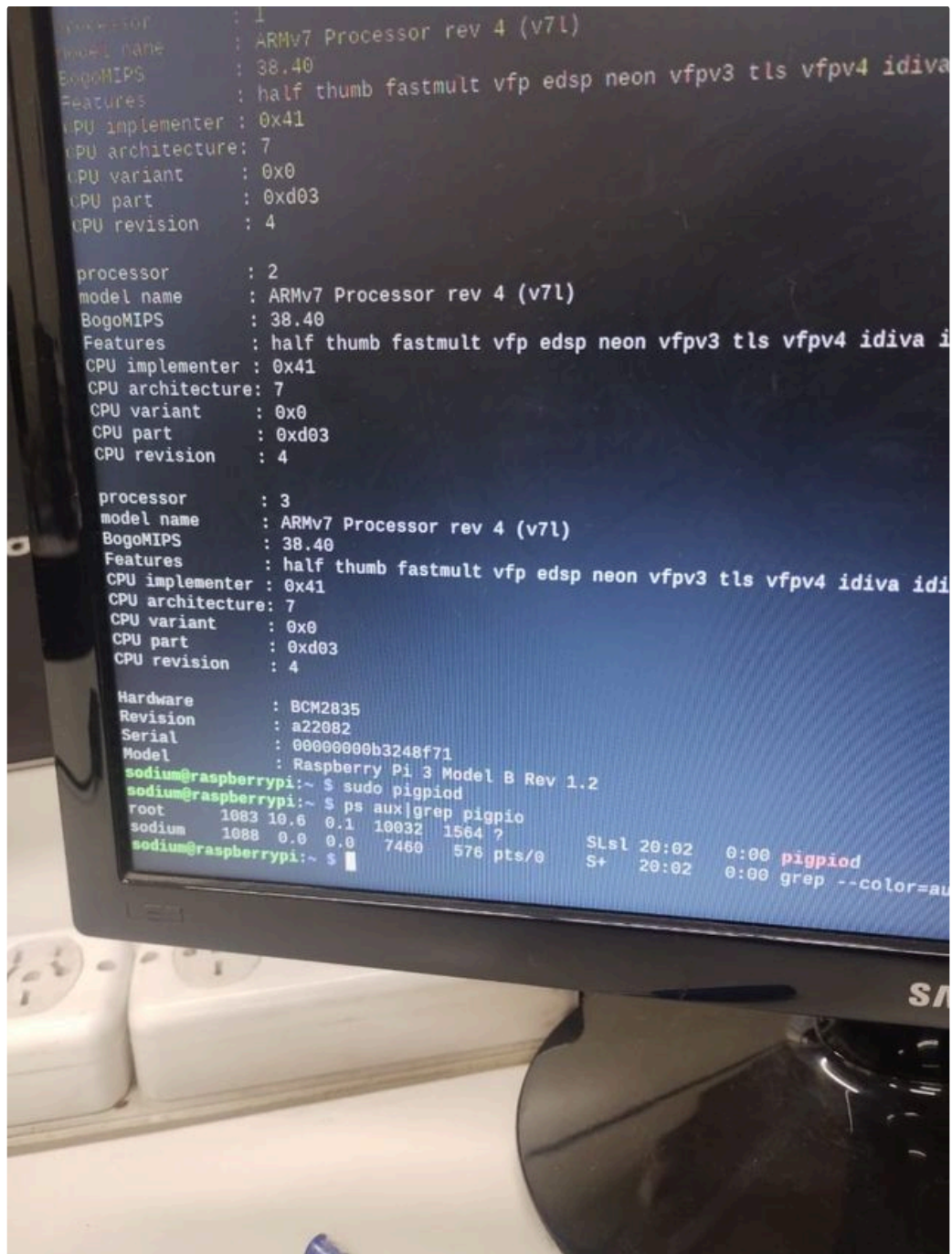


5. Análisis de PIGPIO

Análisis del Problema [↗](#)

1. Se analizó si el problema al ejecutar pigpio en qemu, puede ser el tipo de procesador que emula. Por lo tanto se comparó con el utilizado por el HW físico y se determinó que son iguales.

HW físico



```
processor       : 1
model name     : ARMv7 Processor rev 4 (v7l)
BogoMIPS      : 38.40
Features      : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part      : 0xd03
CPU revision   : 4

processor       : 2
model name     : ARMv7 Processor rev 4 (v7l)
BogoMIPS      : 38.40
Features      : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva i
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part      : 0xd03
CPU revision   : 4

processor       : 3
model name     : ARMv7 Processor rev 4 (v7l)
BogoMIPS      : 38.40
Features      : half thumb fastmult vfp edsp neon vfpv3 tls vfpv4 idiva idi
CPU implementer : 0x41
CPU architecture: 7
CPU variant    : 0x0
CPU part      : 0xd03
CPU revision   : 4

Hardware      : BCM2835
Revision      : a22082
Serial        : 0000000b3248f71
Model         : Raspberry Pi 3 Model B Rev 1.2
sodium@raspberrypi:~$ sudo pigpiod
sodium@raspberrypi:~$ ps aux|grep pigpio
root      1083  0.0  0.1 10032 1564 ?
sodium    1088  0.0  0.0   7460  576 pts/0    Ssl  20:02   0:00 pigpiod
sodium@raspberrypi:~$
```

HW simulado en qemu

```

pi@raspberrypi:~$ cat /proc/cpuinfo
processor       : 0
BogoMIPS      : 125.00
Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0x0
CPU part      : 0xd03
CPU revision  : 4

processor       : 1
BogoMIPS      : 125.00
Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0x0
CPU part      : 0xd03
CPU revision  : 4

processor       : 2
BogoMIPS      : 125.00
Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0x0
CPU part      : 0xd03
CPU revision  : 4

processor       : 3
BogoMIPS      : 125.00
Features      : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant   : 0x0
CPU part      : 0xd03
CPU revision  : 4

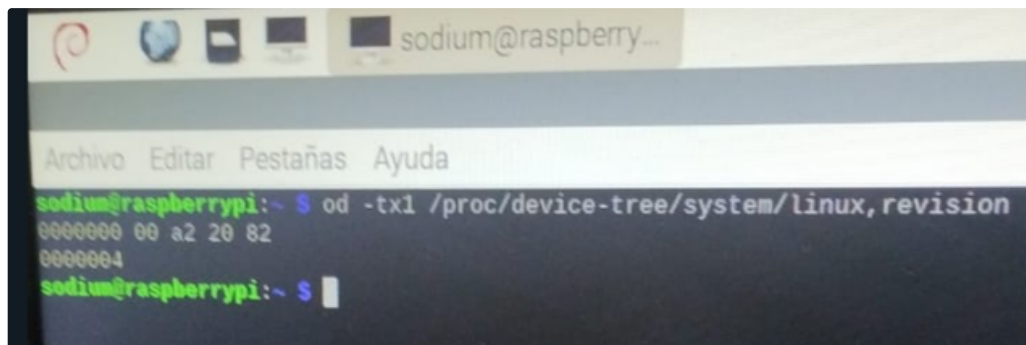
Hardware      : BCM2835
Model         : Raspberry Pi 3 Model B+
pi@raspberrypi:~$

```




2. Entonces se intento generar el comando indicado por [Pi3B+ wrong revision · Issue #3161 · raspberrypi/linux](#) el cual es

```
1 $ od -tx1 /proc/device-tree/system/linux,revision
```

Al ejecutarlo muestra el siguiente contenido:



3. No obstante no se pudo ejecutar en qemu ya que no se encontraba el archivo linux, revision dentro de QEMU
4. Entonces se decidió copiar el archivo los archivos del directorio system de la raspberry pi fisica y copiarla dentro de qemu. Entre los que se hallaba el archivo **linux,revision**

		
linux,revision 16 jul 2024, 11:54 pm	Preview unavailable name 16 jul 2024, 11:54 pm	linux,serial 16 jul 2024, 11:54 pm

Sin embargo, no se pudo copiar dichos archivos, dado que no se permiten copiarlas dentro del directorio /proc, debido a que forma parte del kernel.

```
root@raspberrypi:/proc/device-tree# mkdir system
mkdir: cannot create directory 'system': Operation not permitted
root@raspberrypi:/proc/device-tree#
```

 [Raspberry Pi hardware - Raspberry Pi Documentation](#)

Luego de ver que no se puede copiar el archivo, investigando al parecer todo lo que está dentro del path /proc, pertenece a un path que se genera pero a punta a la memoria del kernel.

Debido a esto, quizás es conveniente descargar el proyecto de PIGPIO y reformar el Path.

Docker de PIGPIO

Investigando sobre PIGPIO, se encontró documentacion donde mencionaba Dockers.

 [pigpio library](#)

Other Languages

There are several third party projects which provide wrappers for pigpio.

Some are listed here:

- [Docker](#) Note that pigpio does not support or accept issues relating to problems of running in docker. Use the docker projects own [issue tracker](#) for that (zinen)
- [Erlang](#)(skvamme)
- [Forth](#)(skvamme)
- [Java](#) JNI wrapper around the pigpio C library (mattlewis)
- [Java](#) via diozero, a high level wrapper around pigpio, Pi4J, wiringPi etc (mattlewis)
- [Java](#) (nkolban)
- [.NET/mono](#) (unosquare)
- [Node.js](#) A wrapper for the pigpio C library (fivdi)
- [Node.js](#) A client for pigpio socket interface (guymcswain)
- [Perl](#) (Gligan Calin Horea)
- [Ruby](#) (Nak)
- [Smalltalk](#)(Instantiations)
- [Xojo](#)(UBogun)
- [Xojo](#)(Eugene Dakin)

Al revisar el link, se encontró un docker que posiblemente implemente lo que necesitamos de PIGPIO dentro de un Docker:

 [GitHub - zinen/docker-alpine-pigpiod: Base for a tiny docker image containing pigpio. Aims to support all models of raspberry pi.](#)

<https://hub.docker.com/r/zinen2/alpine-pigpiod>

dockerhub Explore Repositories Organizations Search Docker Hub

Explore / zinen2/alpine-pigpiod

zinen2/alpine-pigpiod ☆4 Pulls 9.8K

By [zinen2](#) • Updated 3 months ago

Tiny image contains pigpio/pigpiod to control the gpio. Aims to support all models of raspberry pi

[IMAGE](#)

Overview Tags

alpine-pigpiod on github

docker pulls: 9.8K image size: 3.59 MB pigpio: v79 Docker hub auto publish image: passing

Docker image containing pigpiod. Can e.g. be used with node-red to access GPIOs on a Raspberry Pi by installing the package `node-red-node-pi-gpio`.

Since this Docker image is based on the Linux distribution alpine, the image is very small (~6MB).

Usage(32bit system):

```
docker run -it -p 8888:8888 --device /dev/gpiochip0 zinen2/alpine-pigpiod
```

On 64bit system or if you hit access problems try running privileged:

```
docker run -it -p 8888:8888 --privileged zinen2/alpine-pigpiod
```

Note that this container must run on the Raspberry Pi itself. But the GPIOs are accessible for other devices on the network at port 8888.

Docker Pull Command

```
docker pull zinen2/alpine-pigpiod
```

Copy

Al descargarlo y correrlo, lanzó la misma excepción de la cual partimos, por lo que en realidad este docker esta pensado para que se descargue sobre un Raspberry Pi real, correrlo y usarlo, sin la necesidad de hacer la instalación del servicio como lo hicimos en las pruebas.

Análisis de código fuente de PIGPIO [↗](#)

Al llegar al punto de no poder ejecutar PIGPIO dentro de Docker, se comenzó a analizar el código fuente para intentar sortear la excepción que lanza el servicio dentro de QEMU.

Debido a esto, se continuó con los siguientes pasos:

1. Descargamos el código fuente de la página:

[GitHub - joan2937/pigpio: pigpio is a C library for the Raspberry which allows control of the General Purpose Input Outputs \(GPIO\).](#) ,

2. Compilamos el código y lo instalamos según se indica en el sitio del desarrollador:

[pigpio library](#)

```

pi@raspberrypi:~/pigpio-master$ make
gcc -O3 -Wall -pthread -fpic -c -o pigpio.o pigpio.c
sudo make install
gcc -O3 -Wall -pthread -fpic -c -o command.o command.c
gcc -shared -pthread -Wl,-soname,libpigpio.so.1 -o libpigpio.so.1 pigpio.o command.o
ln -fs libpigpio.so.1 libpigpio.so
strip --strip-unneeded libpigpio.so
size libpigpio.so
   text    data     bss     dec    hex filename
 298171   18672   611640   920483   e0ba3 libpigpio.so
gcc -O3 -Wall -pthread -fpic -c -o pigpiod_if.o pigpiod_if.c
gcc -shared -pthread -Wl,-soname,libpigpiod_if.so.1 -o libpigpiod_if.so.1 pigpiod_if.o command.o
ln -fs libpigpiod_if.so.1 libpigpiod_if.so
strip --strip-unneeded libpigpiod_if.so
size libpigpiod_if.so
   text    data     bss     dec    hex filename
   61052    8712   49304   119068   1d11c libpigpiod_if.so
gcc -O3 -Wall -pthread -fpic -c -o pigpiod_if2.o pigpiod_if2.c
gcc -shared -pthread -Wl,-soname,libpigpiod_if2.so.1 -o libpigpiod_if2.so.1 pigpiod_if2.o command.o
ln -fs libpigpiod_if2.so.1 libpigpiod_if2.so
strip --strip-unneeded libpigpiod_if2.so
size libpigpiod_if2.so
   text    data     bss     dec    hex filename
  83346    8720    2936    95002   1731a libpigpiod_if2.so
gcc -O3 -Wall -pthread -c -o x_pigpio.o x_pigpio.c
gcc -o x_pigpio x_pigpio.o -L. -lpigpio -pthread -lrt
gcc -O3 -Wall -pthread -c -o x_pigpiod_if.o x_pigpiod_if.c
gcc -o x_pigpiod_if x_pigpiod_if.o -L. -lpigpiod_if -pthread -lrt
gcc -O3 -Wall -pthread -c -o x_pigpiod_if2.o x_pigpiod_if2.c
gcc -o x_pigpiod_if2 x_pigpiod_if2.o -L. -lpigpiod_if2 -pthread -lrt
gcc -O3 -Wall -pthread -c -o pig2vcd.o pig2vcd.c
gcc -o pig2vcd pig2vcd.o
strip pig2vcd
gcc -O3 -Wall -pthread -c -o pigpiod.o pigpiod.c
gcc -o pigpiod pigpiod.o -L. -lpigpio -pthread -lrt
strip pigpiod
gcc -O3 -Wall -pthread -c -o pigs.o pigs.c
gcc -o pigs pigs.o command.o
strip pigs

```

3. Ejecutamos el código y llegamos al mismo problema:

```

Windows PowerShell
copying pigpio.py -> build/lib
running install_lib
copying build/lib/pigpio.py -> /usr/local/lib/python3.9/dist-packages
byte-compiling /usr/local/lib/python3.9/dist-packages/pigpio.py to pigpio.cpython-3
running install_egg_info
Writing /usr/local/lib/python3.9/dist-packages/pigpio-1.78.egg-info
install -m 0755 -d /usr/local/man/man1
install -m 0644 p*.1 /usr/local/man/man1
install -m 0755 -d /usr/local/man/man3
install -m 0644 p*.3 /usr/local/man/man3
ldconfig
pi@raspberrypi:~/master/pigpio-master$ sudo pigpio
sudo: pigpio: command not found
pi@raspberrypi:~/master/pigpio-master$ sudo pigpiod
2024-05-22 23:43:58 gpioHardwareRevision: unknown revision=0
2024-05-22 23:43:58 initCheckPermitted:
+-----+
|Sorry, this system does not appear to be a raspberry pi. |
|aborting. |
+-----+
Can't initialise pigpio library
pi@raspberrypi:~/master/pigpio-master$

```

4. Como conclusión, entendemos que este código que descargamos pareciera ser igual al descargado e instalado por el comando del sistema operativo.

5. Luego, se realizaron los mismos pasos pero en el Raspberry Pi real y funcionó, por lo que terminados de confirmar de esta manera que el código fuente descargado es el correcto.

Comandos Usados:

Para copiar el archivo

```
1 scp pigpio.c pi@localhost:/home/pi
```

Para buscar

```
1 sudo find / -name pigpiod
```

Para conectarte por SSH

```
1 ssh -p 5022 pi@localhost
```

[🔗 Raspberry Pi: debugging with gdb, command line](#)

```
1 target extended-remote localhost:2345
```

Para ejecutar dentro de GDB

```
1 set remote exec-file /usr/local/bin/pigpiod
2 file /usr/local/bin/pigpiod
```

Modificación de código fuente de PIGPIO [↗](#)

Una vez validado que el código fuente era el correcto, lo que hicimos fue descargarlo en el sistema operativo externo al docker, realizar modificaciones y por medio de comandos de SSH realizar la descarga de los archivos modificados dentro del docker para poder luego compilarlo y ejecutarlo.

1. La primer modificación realizada, fue cambiar el path de donde tomar el archivo que indicaba el hardware. De esa manera avanzó pero lanzó otra excepción en otra parte del código.
2. Al revisar el código vemos que llegó hasta acá:


```

C piggpio.c
8246 {
8275 {
8284 {
8316     else if (model == 17) gpioMask = PI_DEFAULT_UPDATE_MASK_PI4B;
8317     else
8318         gpioMask = PI_DEFAULT_UPDATE_MASK_UNKNOWN;
8319 }
8320     gpioMaskSet = 1;
8321 }
8322
8323 #ifndef EMBEDDED_IN_VM
8324 if (!gpioCfg.internals & PI_CFG_NOSIGHANDLER)
8325     sigSetHandler();
8326 #endif
8327
8328 if (initPeripherals() < 0) return PI_INIT_FAILED;
8329
8330 if (initAllocDMAMem() < 0) return PI_INIT_FAILED;
8331
8332 /* done with /dev/mem */
8333
8334 if (fdMem != -1)
8335 {
8336     close(fdMem);
8337     fdMem = -1;
8338 }
8339
8340 param.sched_priority = sched_get_priority_max(SCHED_FIFO);
8341
8342 if (gpioCfg.internals & PI_CFG_RT_PRIORITY)
8343     sched_setscheduler(0, SCHED_FIFO, &param);
8344
8345 initClock(1); /* initialise main clock */
8346
8347 atexit(gpioTerminate);
8348

```

```

2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1034]=00050000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1035]=00051000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1036]=00052000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1037]=00053000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1038]=00054000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1039]=00055000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1040]=00056000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1041]=00057000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1042]=00058000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1043]=00059000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1044]=0005A000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1045]=0005B000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1046]=0005C000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1047]=0005D000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1048]=0005E000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1049]=0005F000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1050]=00060000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1051]=00061000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1052]=00062000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1053]=00063000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1054]=00064000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1055]=00065000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1056]=00066000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1057]=00067000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1058]=00068000
2024-05-23 01:24:59 initAllocDMAMem: dmaIBus[1059]=00069000
*** INPUT DMA CONTROL BLOCKS ***
2024-05-23 01:24:59 sigHandler: Unhandled signal 11, terminating

2024-05-23 01:24:59 gpioTerminate:

```

A partir de aquí, se comenzaron a realizar los siguientes pasos de forma cíclica:

1. Agregar logs, modificaciones o impresiones en pantalla para entender donde está el problema
2. Copiar los archivos modificados dentro del docker pos SSH.
3. Compilar usando el comando Make
4. Instalar el servicio

5. Ejecutar el servicio
6. Revisar los logs en pantalla.

Todos estos pasos fueron ejecutados reiteradas veces hasta el punto que se analizó intentar utilizar el GDB para poder debuggear el código directamente dentro de Docker.

Las pruebas fueron fallidas debido a que el archivo con el cual se compila el proyecyo makefile posee muchos pasos y no genera los símbolos necesarios para que el GDB pueda debuggear.

Entonces se analizó intentar debuggear con una herramienta mas moderna como Visual Studio pero en forma remota, ya que dentro del docker no contamos con interface de usuario.

Se revisaron varios videos para intentar implementar Debugging Remoto como los siguiente:

[Remote Development on VSCode with SSH](#)

[FPP Remote C/C++ Debugging with VS Code](#)

Se logró debuggear un proyecto simple en C conectado por SSH a un Linux.

Al intentar aplicar la misma metodología pero ahora con el proyecto real dentro del docker, no se logró debido a que no se habilitaban las herramientas de debug en el equipo remoto.

Intuimos que es por un problema de que el Docker posee todos sus puertos cerrados y posiblemente exista algún puerto que requiere VSCode para poder hacer el debug además del propio SSH.

Llegado a este punto, desistimos avanzar por aquí por la complejidad que implicaba sólo por el hecho de fixear el código fuente y poder correrlo.

i Aca se habla sobre simular el GPIO dentro de QEMU, aunque no entendí bien como usarlo. Parece una versión vieja:

[D. - Simulating Raspberry PI GPIO interaction with QEMU](#)

Abajo de todo tiene un link a [GitHub - berdav/qemu-rpi-gpio: Simulate GPIOs in qemu-based Raspberry PI](#)

Se intentó correr usando:

```
docker run -it --rm -e "BOOT=https://downloads.raspberrypi.com/raspbios_arm64/images/raspbios_arm64-2024-03-15/2024-03-15-raspbios-bookworm-arm64.img.xz" -e "RAM_SIZE=""0.6G"" -p 8006:8006 --device=/dev/kvm --cap-add NET_ADMIN qemu/qemu-docker
```

Pero la imagen debe descomprimirse previo al llamado.

i Simulador Online

[Create python code](#)

i Se encontró una imagen de docker que corre qemu directamente, sin un Linux.

<https://hub.docker.com/r/qemux/qemu-docker>

Este parece ser un Orsquetador

<https://hub.docker.com/r/balenalib/raspberrypi3-node>