Programación Paralela • KVM • Home Assistant • Pantalla I2C • Archlinux

ODROID Magazine

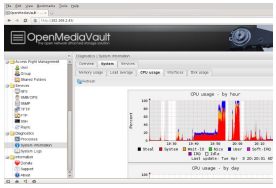
Año Cuatro
Num.#46
Oct 2017

Manten tus archivos seguros y disponibles para todas tus aplicaciones personales

MONTA TU PROPIO HOME SERVER *Multimedia*

Un Swarm de Swarms:

- ODROID-C2 Docker Swarm
- ODROID-MC1 Docker Swarm



Montar tu Propio Home Server: Cómo Almacenar una Gran Cantidad de Archivos Multimedia

October 1, 2017

Cómo montar un almacenamiento conectado en red (NAS) en casa para automatizar la copia de seguridad de los datos del teléfono móvil, administrar y compartir datos en Internet, transmitir vídeos, descargar y administrar Torrents en un smartphone, o alojar un blog personal.

KVM en el ODROID-XU4

October 1, 2017

Esta es una guía paso a paso sobre cómo activar KVM en un ODROID-XU4. Esta guía solo está disponible para las versiones de u-boot odroidxu4-v2017.05 y Linux kernel 4.9.x.

Mi ODROID-C2 Docker Swarm – Parte 2: Implementando una pila en un Swarm

October 1, 2017

Docker 1.13.x ha introducido la nueva función de implementación de pila Docker que permite la utilización de una pila completa de aplicaciones en el Swarm. Una pila es un conjunto de servicios que componen una aplicación. Esta nueva función implementa automáticamente múltiples servicios que están vinculados entre sí, eliminando así ▶️

Juegos Linux: Sistema de Entretenimiento Móvil

October 1, 2017

Vi la oportunidad de crear mi propio sistema de entretenimiento móvil usando algunos componentes disponibles en Hardkernel. Este proyecto es bastante sencillo y muy apropiado para principiantes, incluso para niños.

Cómo instalar ArchLinux con Full Disk Encryption en un ODROID-C2

October 1, 2017

La Full Disk Encryption (FDE) protege nuestros datos del acceso no autorizado en el caso de que alguien logre acceder físicamente a los medios de almacenamiento. En este artículo, voy a describir cómo instalar ArchLinux con Full Disk Encryption en el OROID-C2. El método de encriptación es LUKS con el ▶️

Módulo LCD I2C: Utilizando la LCD serie 1602 16×2 TWI

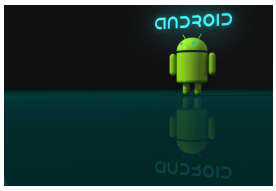
October 1, 2017

La pantalla del módulo LCD serie I2C TWI 1602 16×2 para Arduino JD es la solución ideal para visualizar datos técnicos y mucha más información.

GamODROID-C0: Una Consola de Juegos Retro portátil Basada en ODROID

October 1, 2017

Para este nuevo proyecto, deseaba algo más potente para ejecutar juegos de N64, Dreamcast y PSX, y también algún que otro juego nativo de Linux. No hay muchas opciones de bajo consumo con suficiente CPU + GPU para esto, así que elegí un ODROID-C0



Desarrollo de Android: Proveedor de contenido de Android

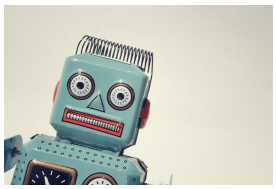
October 1, 2017

Al igual que ocurre con cualquier otro sistema operativo, Android necesita internamente tener un almacenamiento de persistencia para almacenar información del sistema. En este artículo vamos a echar un vistazo a algunos de los proveedores de contenido que utiliza internamente el sistema operativo.

Programación en Paralelo ODROID-MC1: Primeros Pasos

October 1, 2017

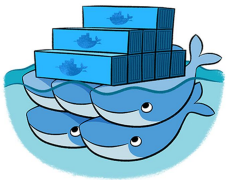
Esta guía no pretende enseñarte a escribir programas en paralelo sobre el ODROID-MC1. Está dirigida a proporcionarte un entorno acondicionado para experimentar con MPJ Express, una implementación de referencia de la API mpiJava 1.2.



Home Assistant: Scripts para Personalizar el Sistema

October 1, 2017

En este artículo, profundizaremos en la personalización de Home Assistant, creando nuestros propios scripts para recopilar datos de sensores remotos y otros dispositivos de control. También veremos varias formas de comunicarnos con sensores remotos.



ODROID-MC1 Docker Swarm: Guía de Inicio

October 1, 2017

El personal de Hardkernel ha montado una gran instalación informática en forma de clúster para probar la estabilidad de Kernel 4.9. El cluster consistía en 200 ODROID-XU4 (es decir, un total neto de 1600 núcleos de CPU y 400 GB de RAM),

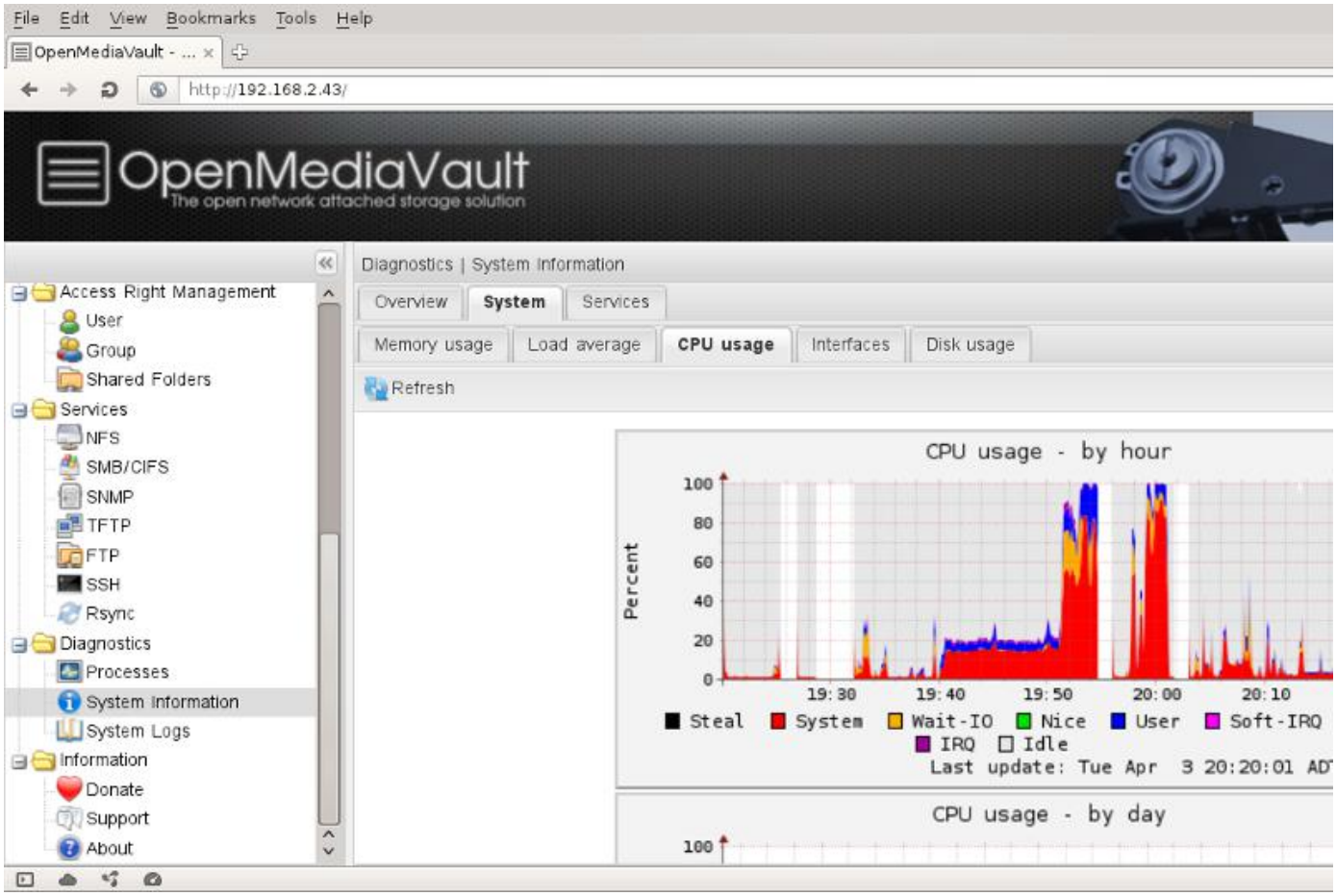
Conociendo un ODROIDian: Brian Kim, Ingeniero de Hardkernel

October 1, 2017

Conociendo un ODROIDian es una sección mensual en la que puedes conocer y aprender de las personas que disfrutan usando productos Hardkernel.

Montar tu Propio Home Server: Cómo Almacenar una Gran Cantidad de Archivos Multimedia

October 1, 2017 By odroidinc.com Linux, Mecaniqueo



¿Por qué necesitarías un servidor de almacenamiento conectado en red (NAS) en casa?

- Copia de seguridad automática de los datos de tu Smartphone
- Administrar y compartir datos en Internet
- Transmitir y reproducir videos guardados
- Descargar y administrar Torrents en un smartphone
- Alojar un blog personal
- Habilitar SSL por seguridad

Componentes necesarios:

- Servicio de Internet
- Un router WiFi
- Un ordenador de escritorio o un ordenador portátil, como un MacBook Pro
- Un ODROID-HC1 con su fuente de alimentación
- Una Tarjeta MicroSD para el sistema operativo
- Un Cable LAN para conectar el ODROID-HC1 al router WiFi
- Una Unidad de disco duro (2,5 pulgadas) para almacenar los datos multimedia

Figura 1 – Home Server usando un ODROID-HC1

También necesitas conocer un poco el sistema operativo, además de Open Media Vault (www.openmediavault.org), que nos permitirá instalar y administrar un almacenamiento conectado en red sin necesidad de tener conocimientos avanzados.

Preparación

En primer lugar, descarga Open Media Vault (OMV) para el ODROID-HC1 desde http://bit.ly/2xogExP a tu ordenador. Consulta el archivo readme.txt para conocer el nombre de usuario y la contraseña.

Nombre de usuario de la interfaz web = admin
Contraseña de la interfaz web = openmediavault

Nombre de usuario de la consola/ssh = root
Contraseña de la consola/ssh (3.0.75+) = openmediavault

Figura 2 – Imagen de Open Media Vault descargada

A continuación, utiliza un adaptador USB con una tarjeta microSD de 8GB, abre Etcher (etcher.io) para grabar el sistema operativo, tal y como se muestra en la Figura 3. Asegúrate de descomprimir el archivo .7z antes de seleccionarlo en Etcher.

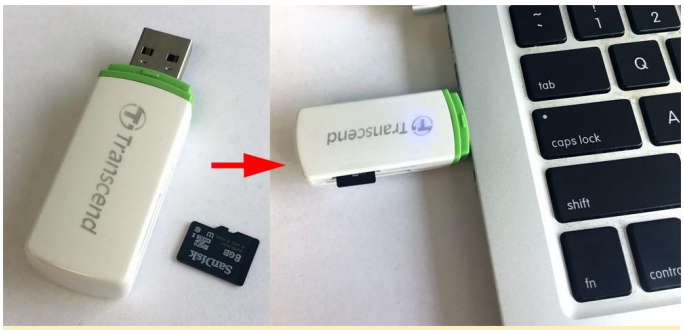


Figura 3 – Insertando el adaptador USB y la tarjeta microSD en el ordenador

Figura 4: el archivo de imagen descomprimido tiene un nombre de archivo diferente al archivo .7z

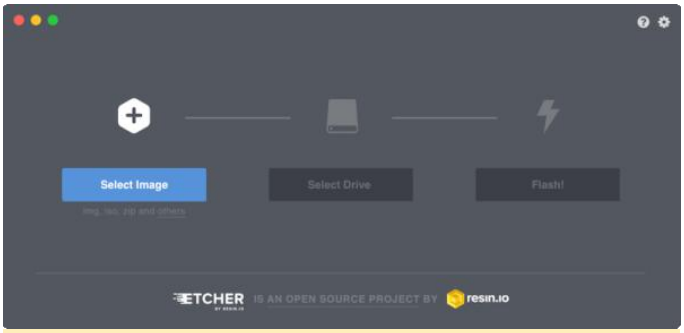


Figura 5: Etcher te permite escribir imágenes pre-configuradas en una tarjeta microSD

Configuración general

Inserta la SD con la imagen completada de Open Media Vault en el ODROID-HC1, luego desliza e inserta la unidad de disco duro en el conector SATA. Conecta el cable LAN desde el router WiFi al HC1 y utiliza la fuente de alimentación para encenderlo. Para el primer arranque necesitarás aproximadamente unos 10 minutos. Con otro cable LAN, conecta el ordenador al mismo router WiFi al que está conectado el HC1.

A continuación, descarga e instale Angry IP Scanner (http://bit.ly/2wCMeII) y escanea las direcciones IP de

los dispositivos conectados. El nombre de equipo aparecerá como odroidxu4.local. Abre un navegador e introduce la dirección del ODROID-HC1.

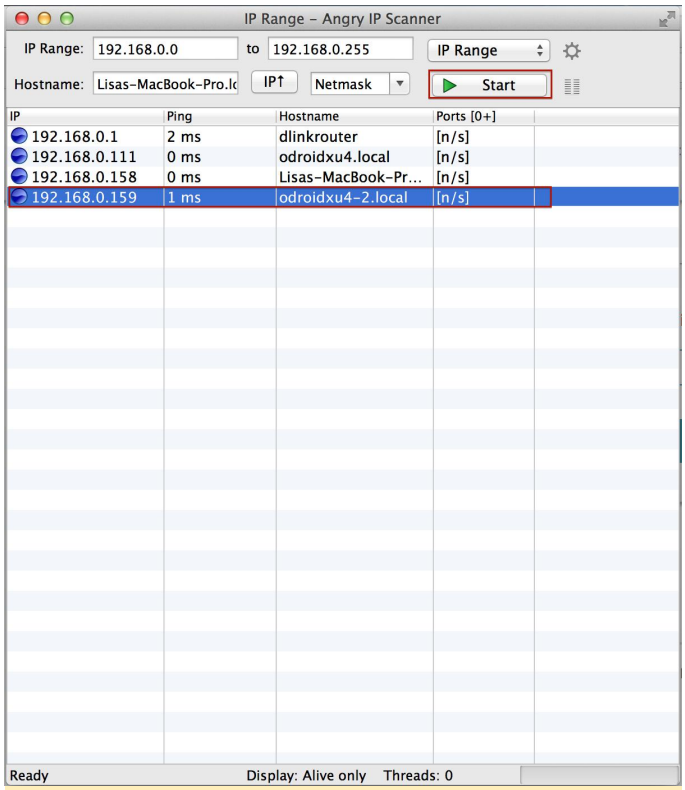


Figura 6: Escaneando las direcciones IP locales para localizar la dirección IP del ODROID-HC1

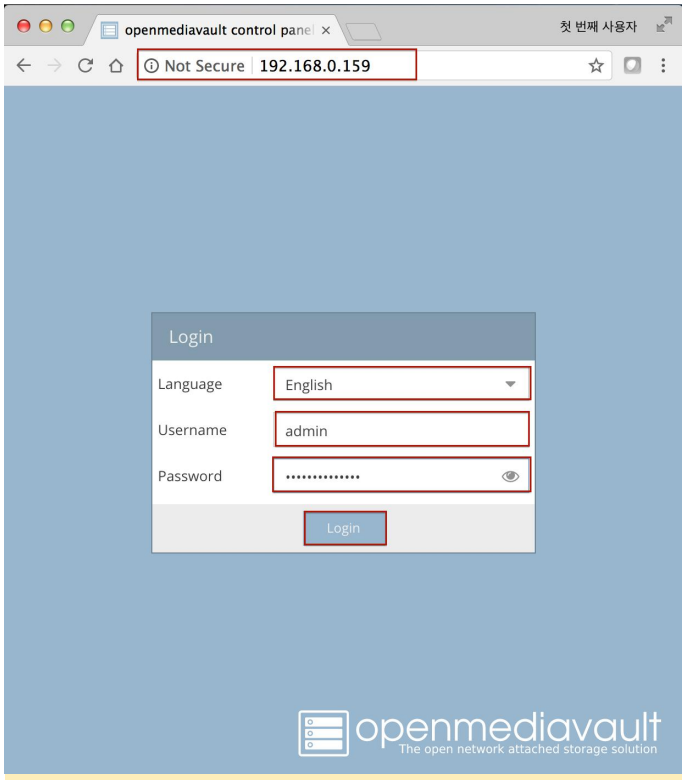


Figura 7 – Iniciando sesión en la interfaz web de Open Media Vault

Como ya he mencionado, el nombre de usuario y la contraseña por defecto se encuentran en readme.txt en http://bit.ly/2xogExP.

Nombre de usuario de la interfaz web = admin
Contraseña de la interfaz web = openmediavault

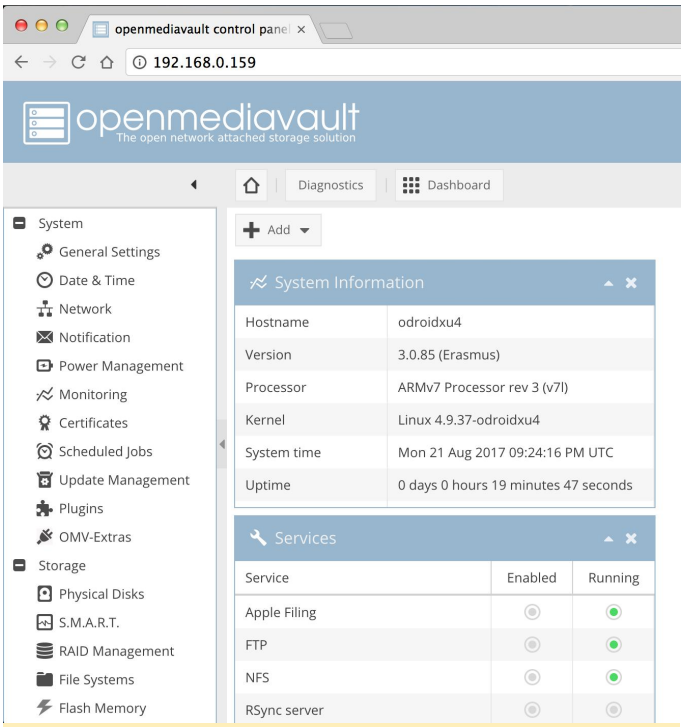


Figura 8 – Pantalla de inicio de la interfaz web de Open Media Vault

Dirígete a “System -> Date & Time” y cambia la zona horaria según tu ubicación actual, luego activa la opción “Use NTP server -> Save -> Apply”.

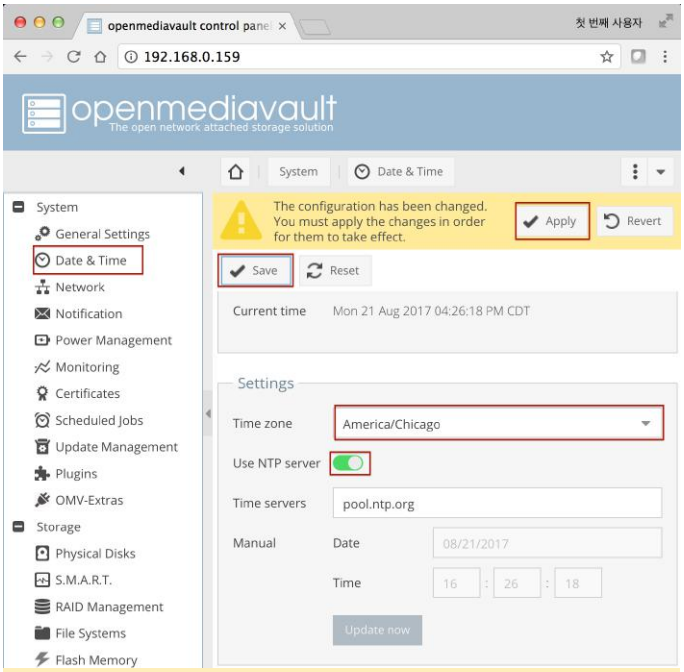


Figura 9 – Actualizando la fecha y hora en Open Media Vault

También puedes cambiar el tiempo de espera de la sesión a “0” para no cerrar sesión tras una cierta

cantidad de tiempo de inactividad seleccionando “General Settings -> Session timeout -> 0 -> Save -> Apply -> Yes “.

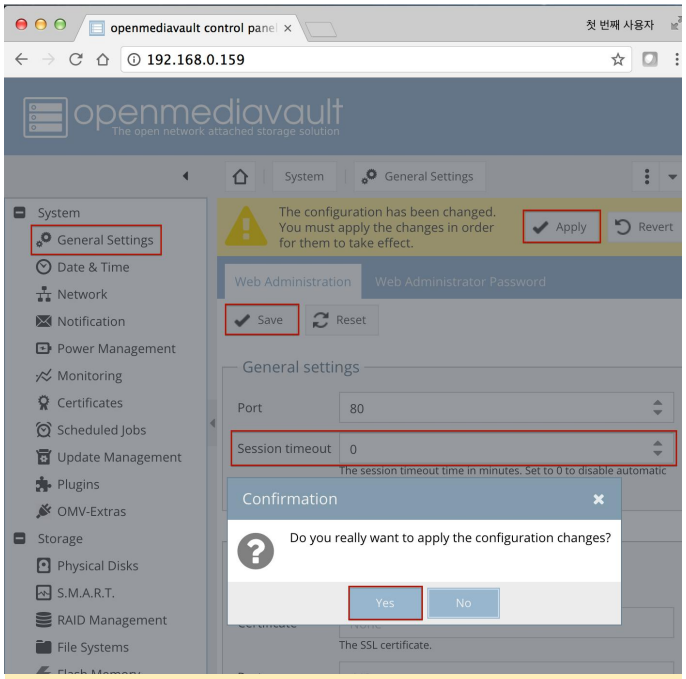


Figura 10: Guardando los cambios de la configuración en Open Media Vault

A continuación, actualiza el sistema a la última versión seleccionando ” Update Management -> Check Package information -> Upgrade “, vuelva a cargar la página una vez que haya finalizado la actualización, luego reinicie ODROID-HC1 utilizando la opción “Reboot” en la interfaz web de Open Media Vault

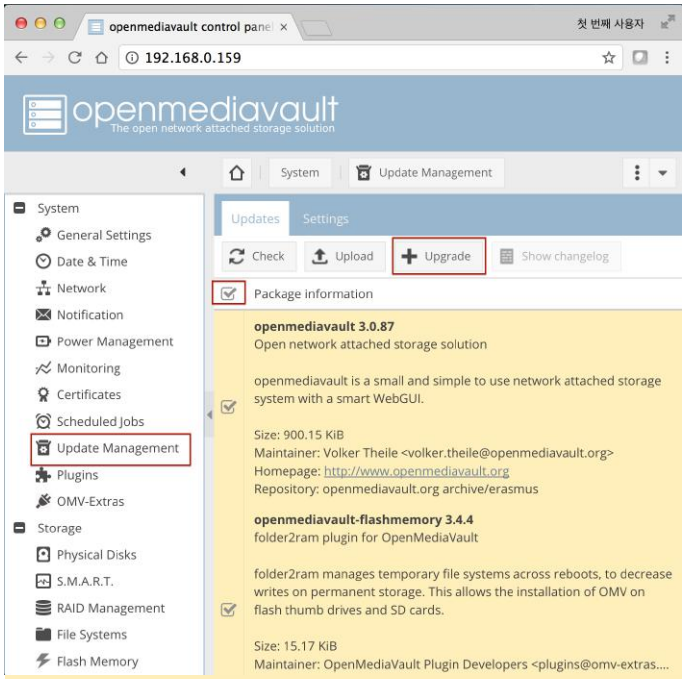


Figura 11 – Actualizando a la última versión de Open Media Vault

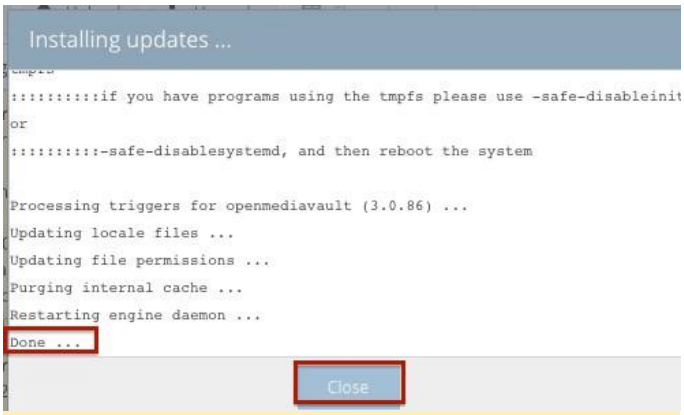


Figura 12: La actualización de Open Media Vault se ha completado

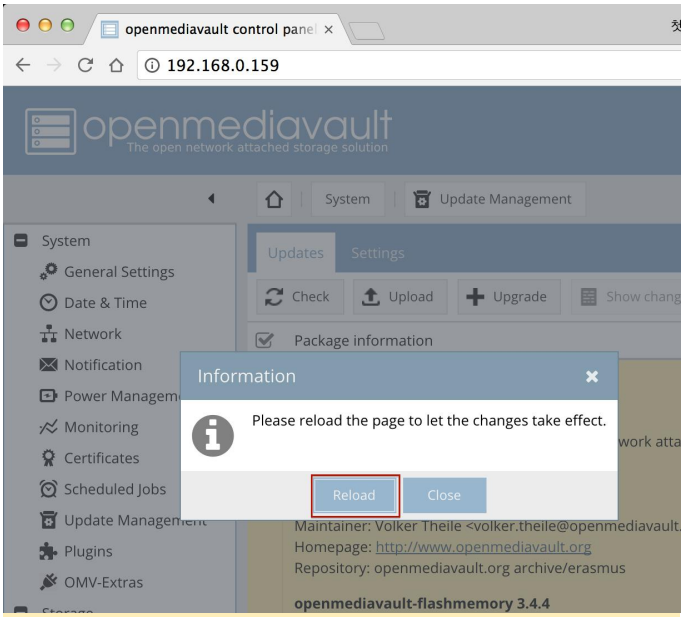


Figura 13: la página se debe volver a cargar tras completarse la actualización de Open Media Vault

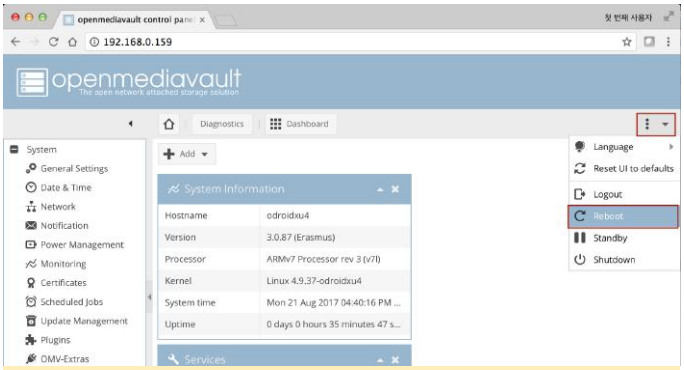


Figura 14: Seleccionando la opción “Reboot” en la interfaz web de Open Media Vault

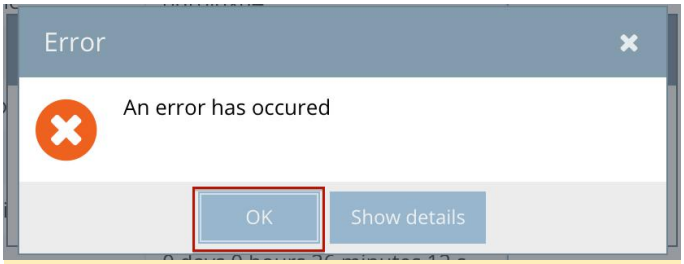
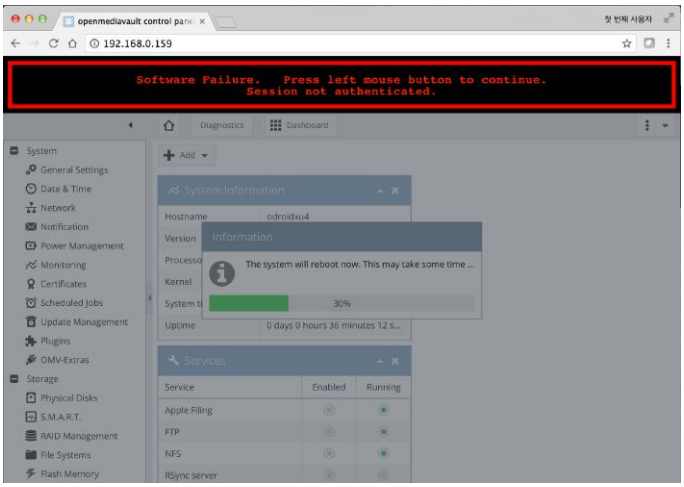


Figura 15 y 16: Ignora los mensajes de error después de presionar “Reboot”



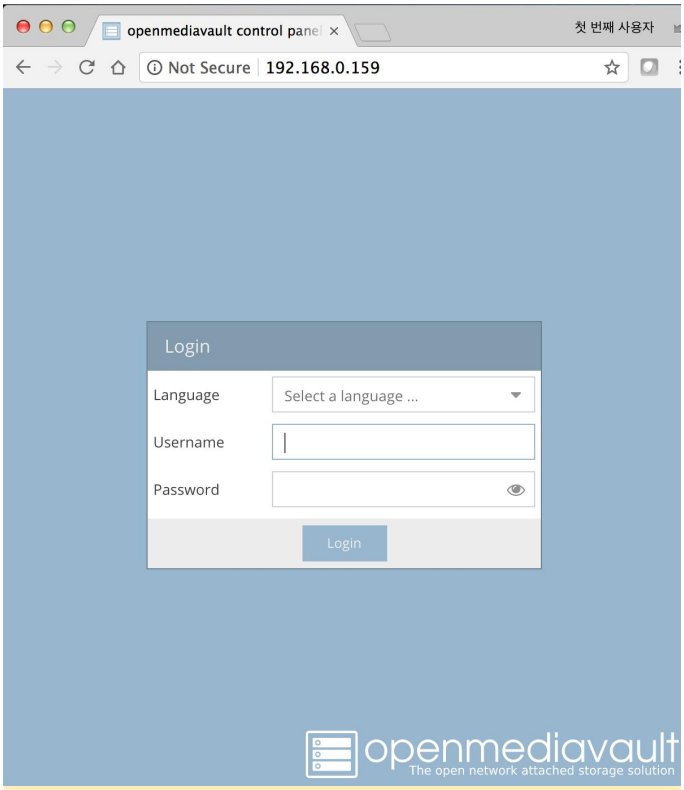
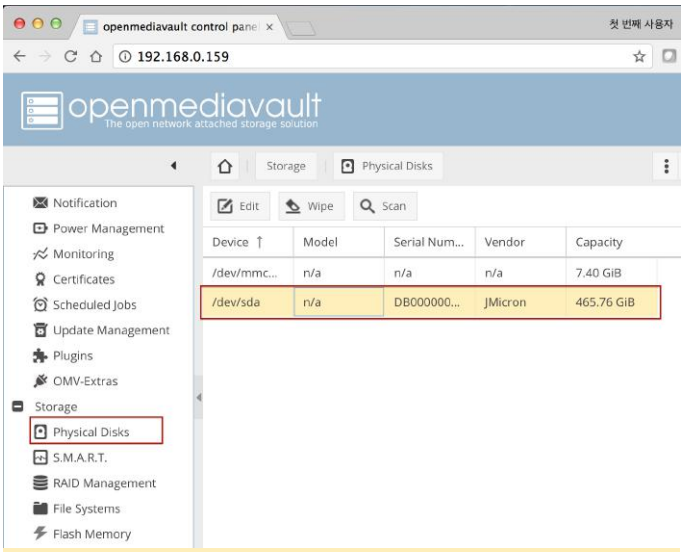


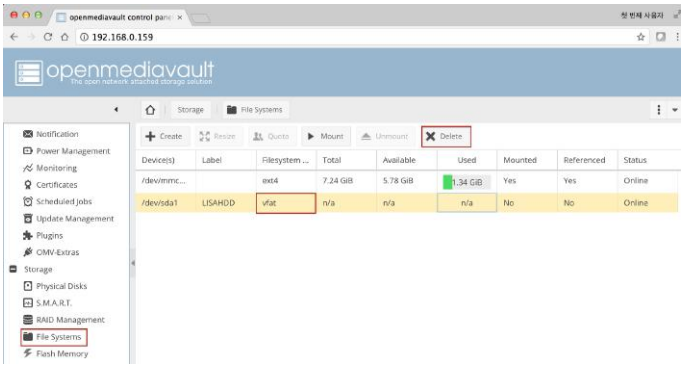
Figura 17: inicia sesión en la interfaz web de Open Media Vault tras haberse completado el reinicio

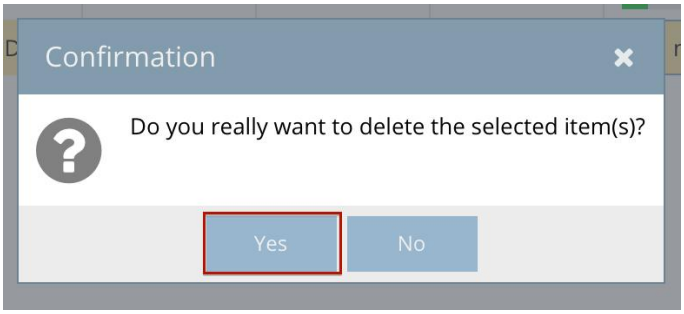
Definir los permisos

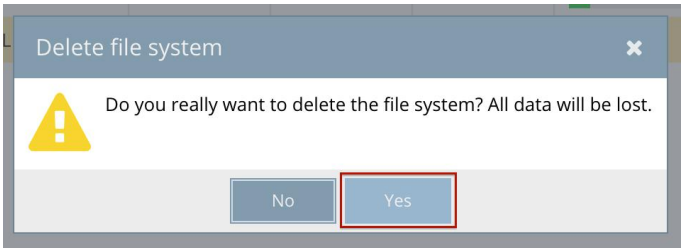
El disco duro debe estar en formato ext4 para que sea compatible con Open Media Vault. Si el sistema de archivos del disco duro no es ext4, necesitarás crear un nuevo sistema de archivos, tal y como se muestra en la Figura 18.

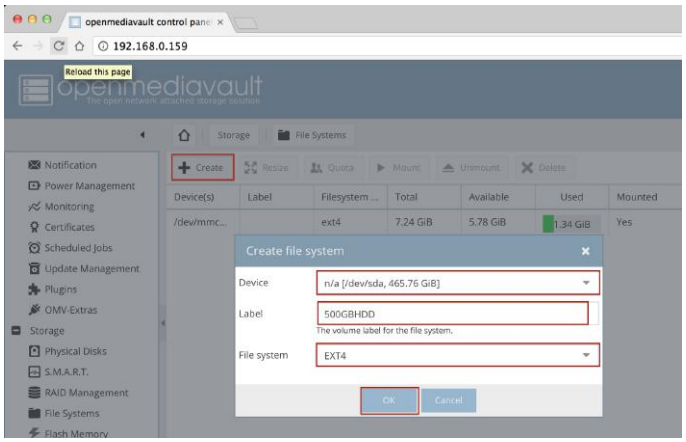


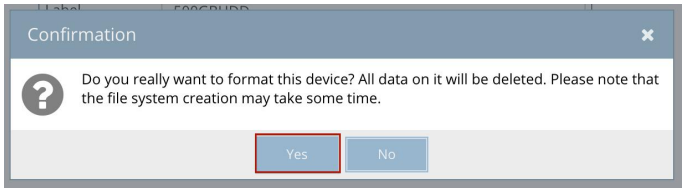
Figuras 18 – 24 – Formateando del disco duro en ext4

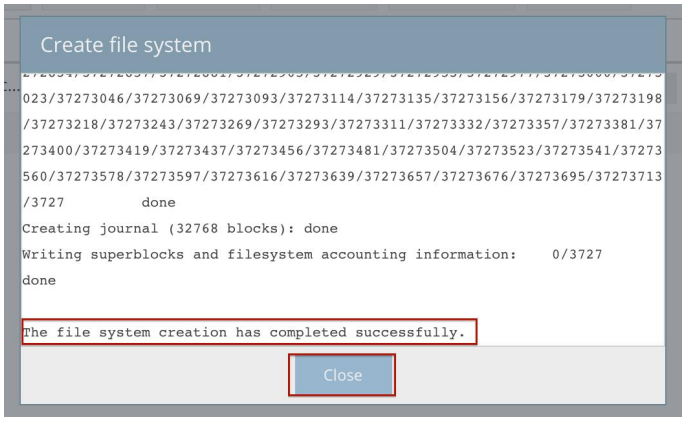




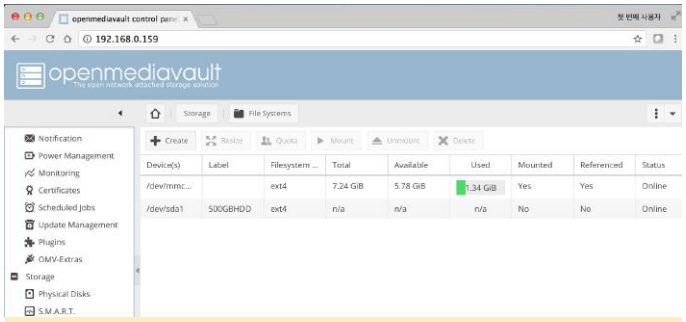




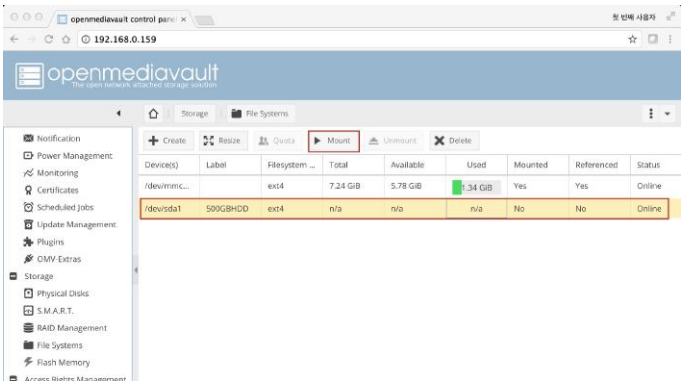


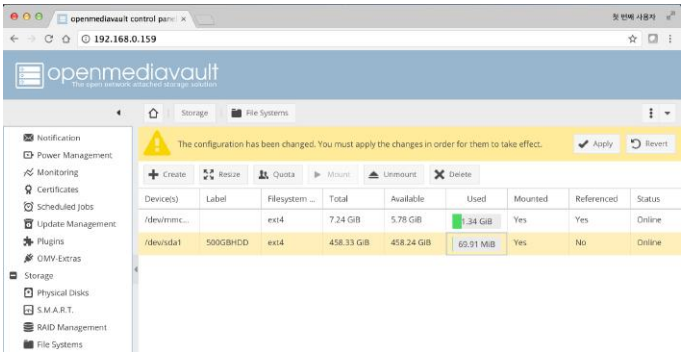


Una vez completado el formato, selecciona “Mount” tal y como se muestra en las Figuras 19 y 20.

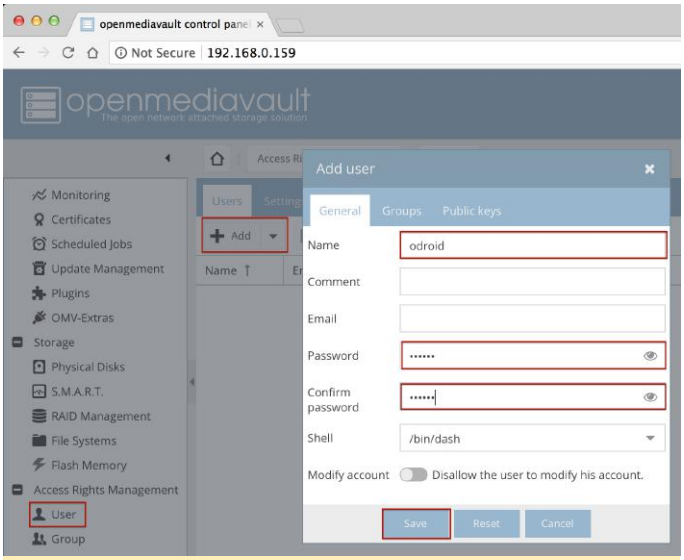


Figuras 25, 26 y 27 – Montando la unidad de disco duro recién formateada

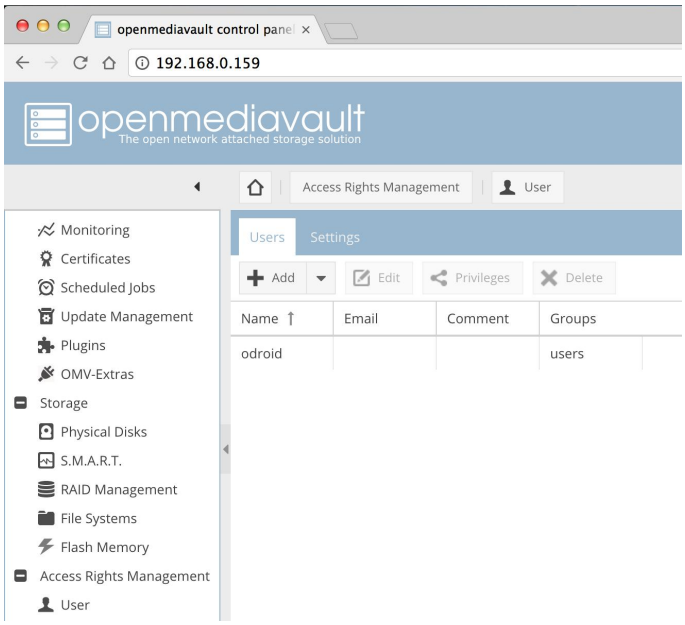




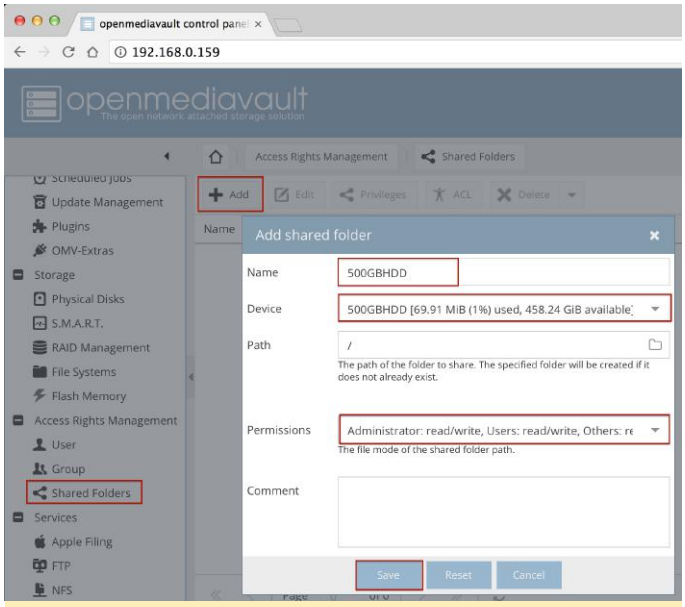
El siguiente paso es registrar los usuarios que tendrán permisos para transferir datos hacia/desde el servidor.



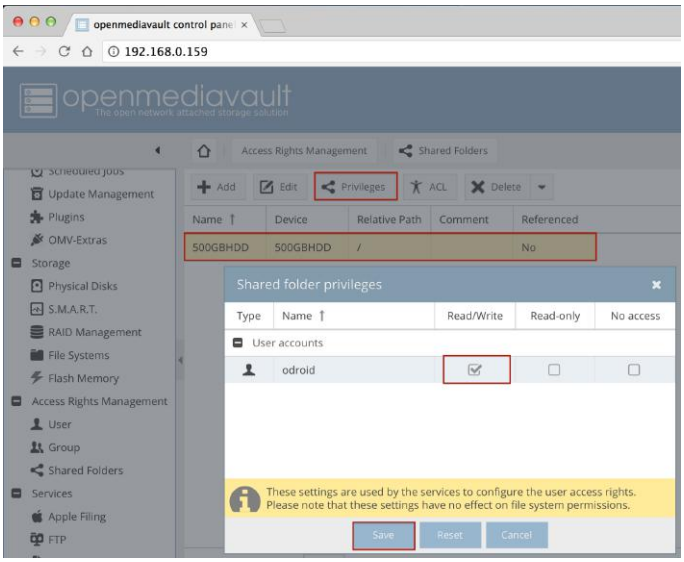
Figuras 28 y 29: Registrando del usuario “odroid” para poder transferir datos desde/hacia el servidor



Una vez creado el usuario, crea una carpeta compartida seleccionando ” Shared Folders -> Add -> Name -> Select Device -> Set Permissions -> Save “. A cada usuario se le debe conceder los correspondiente sprivilegios. Concede al usuario “odroid” los privilegios de lectura/escritura para la carpeta compartida y guarda la configuración.



Figuras 30 y 31: Creando la carpeta compartida y asignando los privilegios individuales de usuario



ACL es otro tipo de permiso que se deben conceder, tal y como se describe en **http://bit.ly/2xn98sb**. El usuario “odroid” necesita permisos de lectura/escritura/ejecución, y al resto de usuarios se les pueden conceder los permisos que sean necesarios.

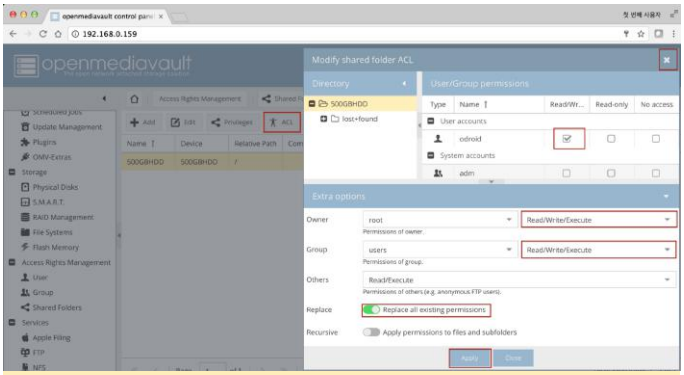
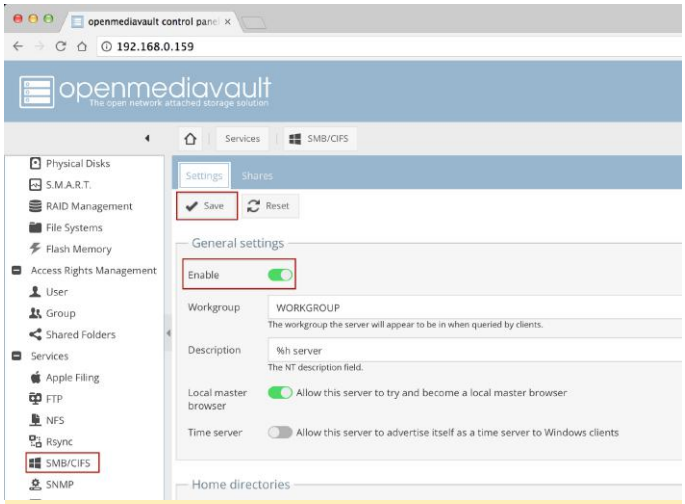


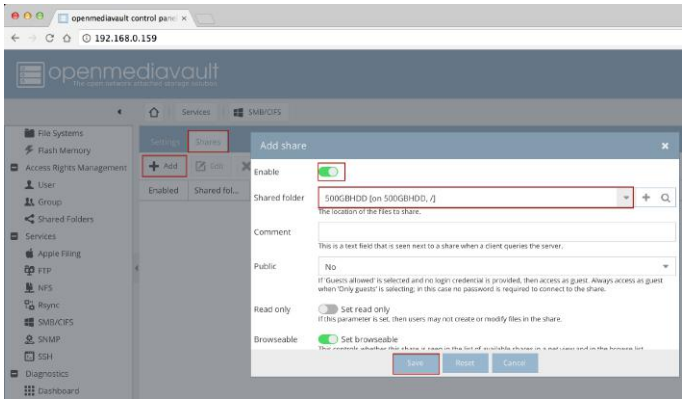
Figura 32: Concediendo permisos ACL al usuario “odroid”

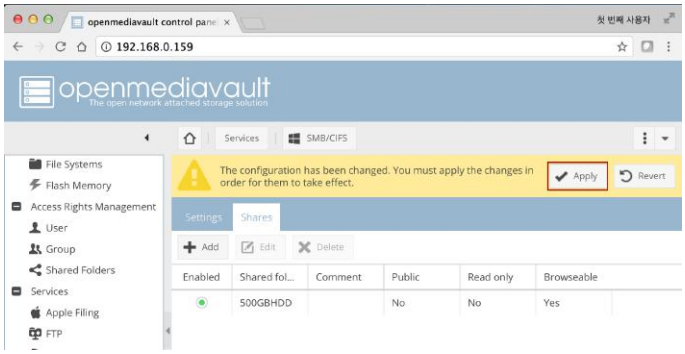
Transferencia de datos usando Samba

El servidor se puede compartir con el grupo de trabajo utilizando Samba (SMB). Haz clic en “Apply” para ver la carpeta compartida.



Figuras 33, 34 y 35: compartir el servidor usando Samba





Ten en cuenta que si tienes dos o más dispositivos o carpetas compartidas idénticas, tu ordenador puede cambiarle el nombre a uno de ellos. Por ejemplo, si tiene dos ODROID-HC1 conectados al router, reconocerá el primero como odroidxu4 y nombrará el segundo como odroidxu4-2, para diferenciarlos. Si no ves los dos automáticamente, intenta reiniciar el ordenador.

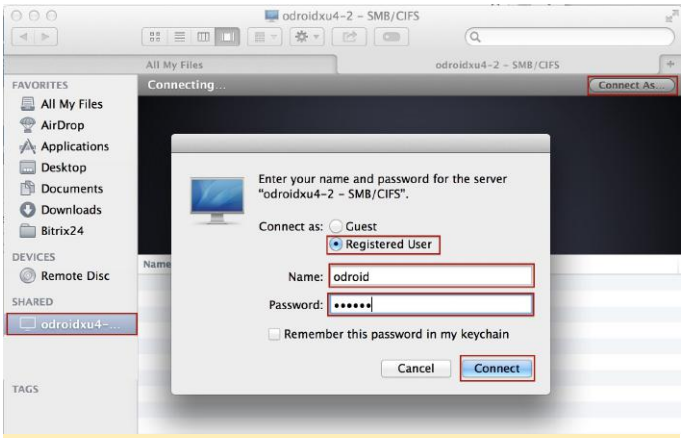


Figura 36: Accediendo a la carpeta compartida desde un ordenador conectado a la red

Abre Finder y marca “Shared” para ver el servidor compartido odroidxu4, que es el ODROID-HC1. Haz clic en “Connect As” e introduce el nombre y la contraseña que coinciden con el nombre de usuario y la contraseña que se crearon en el servidor. Una vez establecida la conexión, se pueden transferir los archivos y las carpetas desde y hacia el servidor ODROID-HC1.

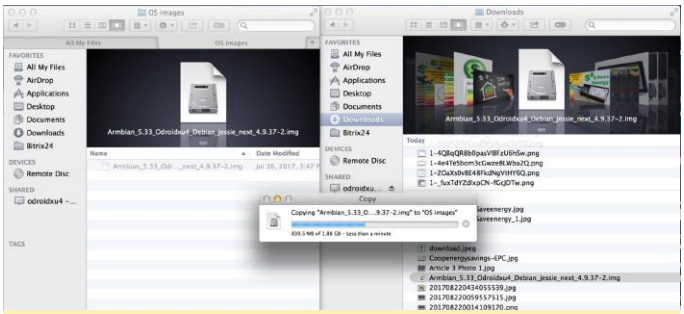


Figura 37 – Copiando archivos y carpetas al ODROID-HC1 usando Samba

Transferencia de datos mediante FTP

El Protocolo de Transferencia de Archivos (FTP) es un protocolo de red estándar utilizado para la transferencia de archivos entre un cliente y un servidor dentro de una red informática. En primer lugar, activa el FTP en Open Media Vault tal y como se muestra en la Figura 38.

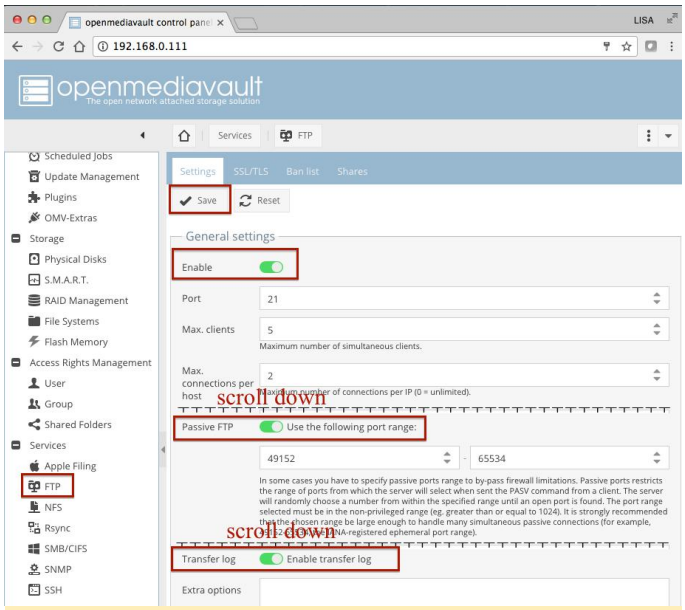


Figura 38: Activando el FTP en Open Media Vault

A continuación, habilita la carpeta compartida seleccionando “Services -> FTP -> Shares -> Add -> Enable -> select Shared folder -> Save”.

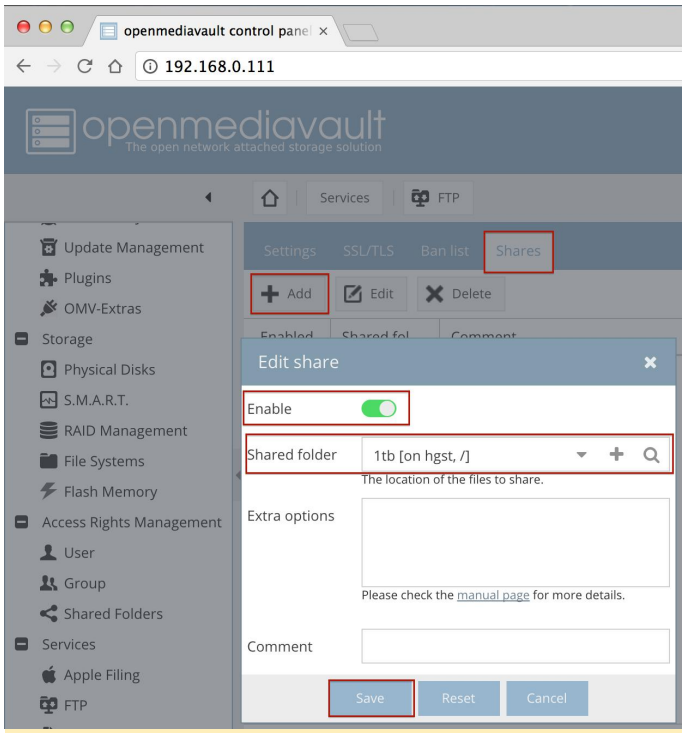
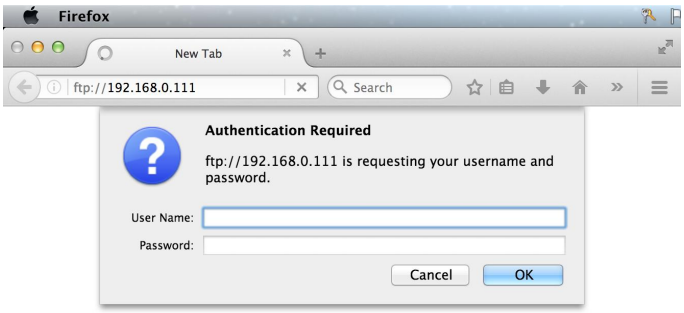
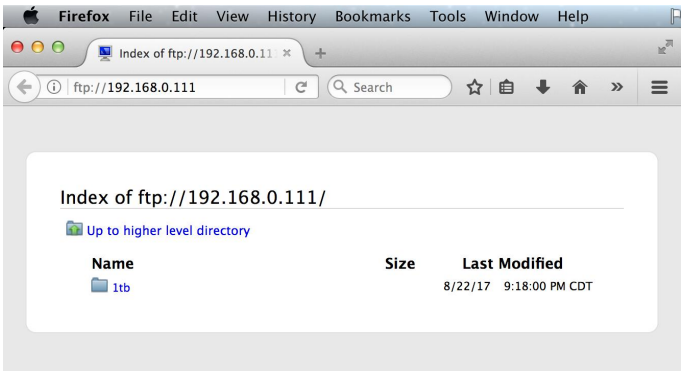


Figura 39: Seleccionando la carpeta FTP compartida en Open Media Vault

Tras habilitar el FTP, los archivos se podrán transferir hacia/desde el servidor visitando ftp://192.168.0.111 en tu navegador, utiliza la dirección de tu servidor ODROID-HC1 en lugar de 192.168.0.111.



Figuras 40 y 41: visitando el servidor de Open Media Vault a través de FTP usando Firefox



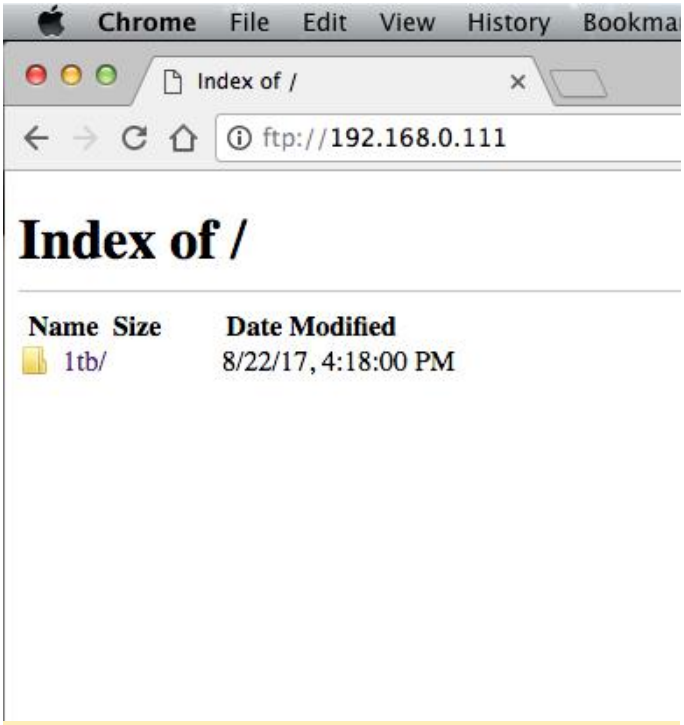
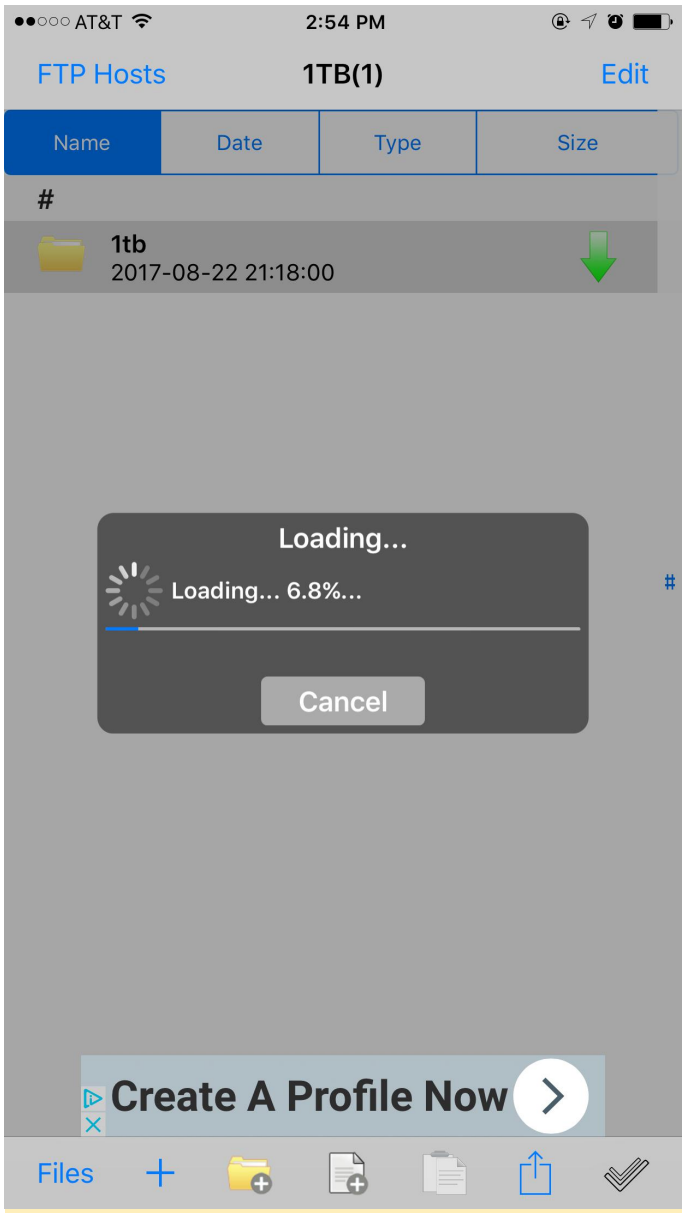
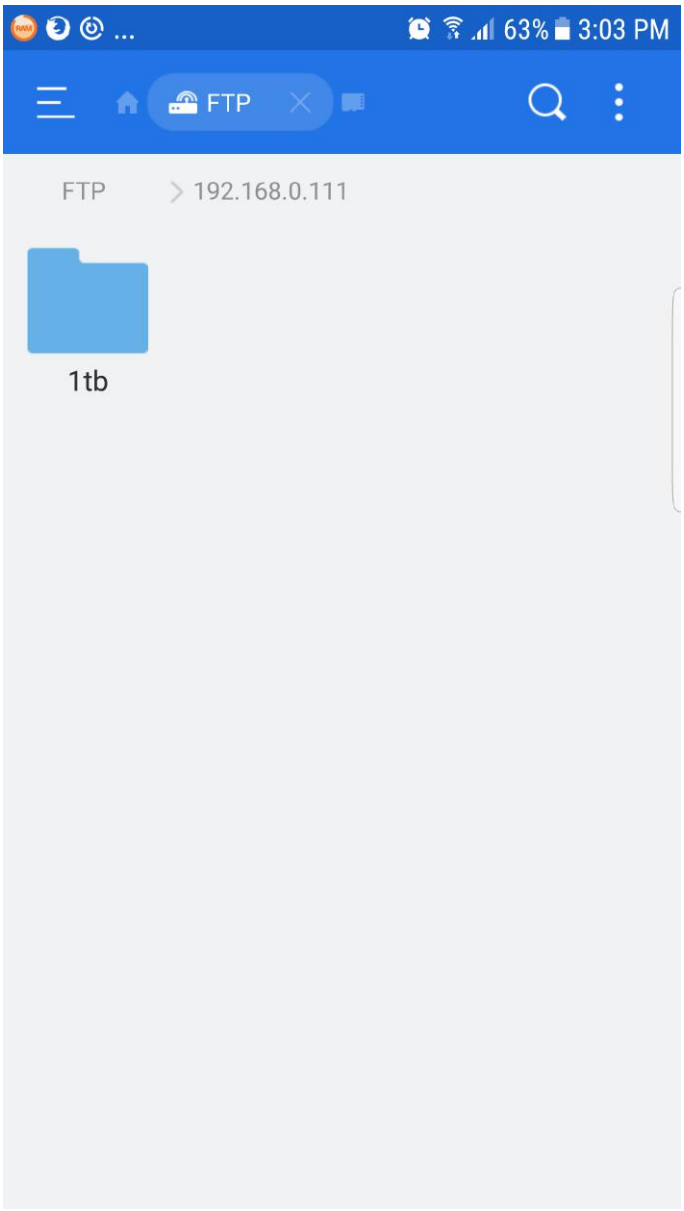


Figura 42: Visitando el servidor de Open Media Vault a través de FTP usando Chrome

Después, instala un FTP en su teléfono movil, usando una aplicación como FTP Sprite para iPhone o ES File Explorer para Android.

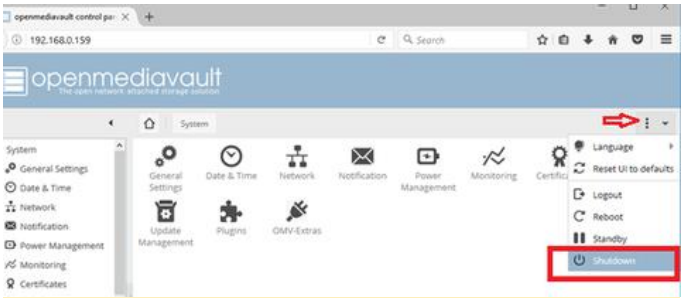


Figuras 43 y 44: Accediendo al servidor de Open Media Vault usando FTP con un smartphone

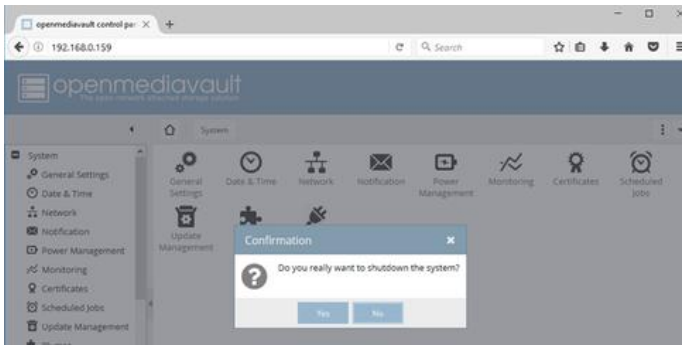


Cierre del sistema

En la interfaz web de Open Media Vault, justamente debajo del banner, haz clic en los tres puntos verticales de la derecha y selecciona “Shutdown”.



Figuras 45 y 46: Cerrando el servidor a través del menú Open Media Vault



Cuando aparezca la pantalla que se muestra en la Figura 47, significara que tu sistema operativo ha dejado de funcionar y el LED azul deberías verlo apagado en el ODROID-HC1. Llegados a este punto, puedes desconectar la fuente de alimentación y extraer la tarjeta microSD. Sigue este procedimiento de apagado cada vez que necesites cambiar el disco duro, actualizar el sistema operativo en la tarjeta microSD o simplemente desenchufar la alimentación. Esto te ayudará a evitar daños en el ODROID-HC1.

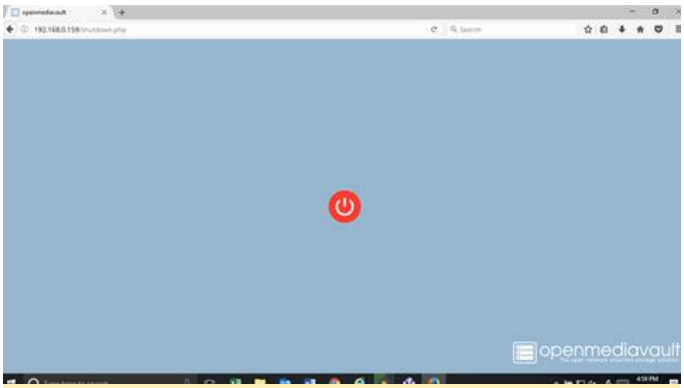
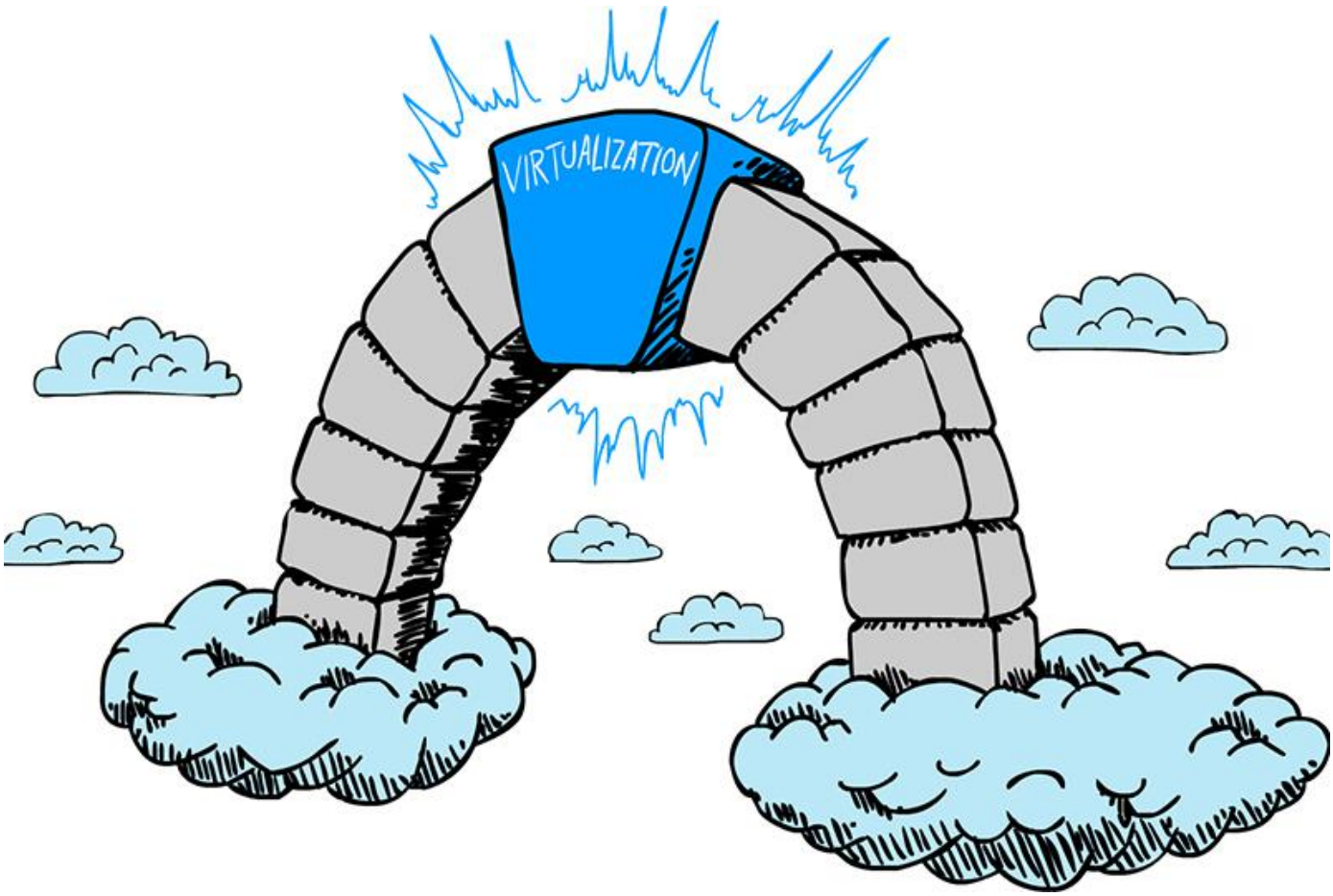


Figura 47: pantalla post-cierre del Open Media Vault

Para comentarios, preguntas o sugerencias, visita el artículo original en **https://medium.com/p/6a3771d9172**.

KVM en el ODROID-XU4

October 1, 2017 By Brian Kim ODROID-XU4, Tutoriales



Esta es una guía paso a paso sobre cómo activar KVM en un ODROID-XU4. Esta guía solo está disponible para las versiones de u-boot odroidxu4-v2017.05 y Linux kernel 4.9.x. El primer paso es recompilar el kernel. KVM necesita el arch timer en lugar de MCT (Multi-Core Timer), que es el temporizador por defecto del ODROID-XU4 (exynos5422-odroidxu4-kvm.dtb). Las configuraciones relacionadas con la virtualización están en el archivo odroidxu4_kvm_defconfig.

```
$ sudo apt update
$ sudo apt install git
$ git clone --depth 1
https://github.com/hardkernel/linux -b
odroidxu4-4.9.y
$ cd linux
$ make odroidxu4_kvm_defconfig
$ make -j8
$ sudo make modules_install
$ sudo cp arch/arm/boot/zImage
/media/boot/zImage_kvm
$ sudo cp arch/arm/boot/dts/exynos5422-
odroidxu4-kvm.dtb /media/boot/
```

Modifica el archivo boot.ini cambiando “zImage” por “zImage_kvm” y “exynos5422-odrooidxu4.dtb” por “exynos5422-odrooidxu4-kvm.dtb” /media/boot/boot.ini

```
(......)
# Load kernel, initrd and dtb in that sequence
fatload mmc 0:1 0x40008000 zImage_kvm
(......)
if test "${board_name}" = "xu4"; then fatload
mmc 0:1 0x44000000 exynos5422-odrooidxu4-
kvm.dtb; setenv fdtloaded "true"; fi
(......)
```

Reinicia el ODROID-XU4, después verifica si KVM está activado una vez que el proceso de arranque haya finalizado:

```
$ dmesg | grep HYP
[ 0.096589] CPU: All CPU(s) started in HYP
```

```
mode.
[ 0.777814] kvm [1]: HYP VA range:
c0000000:ffffffff
$ dmesg | grep kvm
[ 0.777771] kvm [1]: 8-bit VMID
[ 0.777793] kvm [1]: IDMAP page: 40201000
[ 0.777814] kvm [1]: HYP VA range:
c0000000:ffffffff
[ 0.778642] kvm [1]: Hyp mode initialized
successfully
[ 0.778713] kvm [1]: vgic-v2@10484000
[ 0.779091] kvm [1]: vgic interrupt IRQ16
[ 0.779127] kvm [1]: virtual timer IRQ60
$ cat /proc/interrupts | grep arch_timer
58: 0 0 0 0 0 0 0 0 GIC-0 29 Level arch_timer
59: 0 1857 1412 1345 16986 6933 5162 3145 GIC-
0 30 Level arch_timer
```

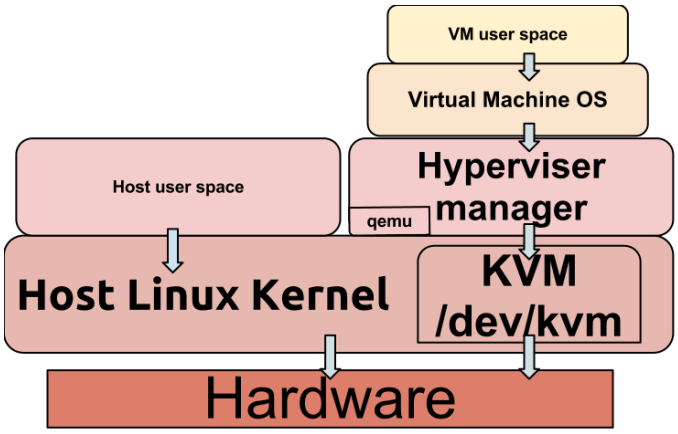


Figura 1 Arquitectura de la máquina virtual

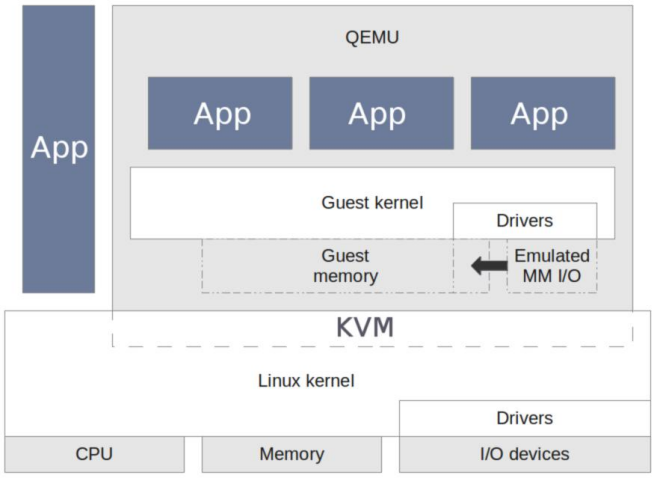


Figura 2 – Arquitectura de la máquina virtual

Ubuntu Mínimo 16.04.3 ejecutándose con QEMU y KVM/ARM

Para continuar con esta sección, asegúrate de que KVM ya esté activado, con 4 GB o más de espacio de almacenamiento disponible. En esta sección ejecutaremos la imagen Ubuntu Minimal 16.04.3 en la máquina virtual usando QEMU y KVM/ARM.

Para empezar, instala qemu-system-arm con el cual vamos a virtualizar la máquina arm y los paquetes requeridos:

```
$ sudo apt update
$ sudo apt install qemu-system-arm kpartx
```

A continuación, prepara las imágenes dtb y del kernel del Sistema Opertaivo invitado. Es necesario configurar la frecuencia del reloj para el temporizador en el archivo dts agregando la línea “clock-frequency = <100000000>;” en el nodo del temporizador.

```
$ wget
https://www.kernel.org/pub/linux/kernel/v4.x/l
inux-4.13.tar.xz
$ tar Jxvf linux-4.13.tar.xz
$ cd linux
$ nano arch/arm/boot/dts/vexpress-v2p-ca15-
tc1.dts
```

arch/arm/boot/dts/vexpress-v2p-ca15-tc1.dts

```
timer {
compatible = "arm,armv7-timer";
interrupts = <1 13 0xf08>,
<1 14 0xf08>,
<1 11 0xf08>,
<1 10 0xf08>;
clock-frequency = <100000000>;
};
```

Compila y copia las imágenes dtb y zImage en el directorio de trabajo:

```
$ make vexpress_defconfig
$ make menuconfig
```

Enable the block layer —> [*] Support for large (2TB+) block devices and files

```
$ make zImage dtbs -j8
$ cp arch/arm/boot/zImage ../
$ cp arch/arm/boot/dts/vexpress-v2p-ca15-
tc1.dtb ../
$ cd ..
```

Prepara la imagen del sistema de archivos root de Ubuntu Minimal descargando la imagen Ubuntu

minimal 16.04.3 y genera la imagen del sistema de archivos root desde la imagen.

```
$ wget
https://odroid.in/ubuntu_16.04lts/ubuntu-
16.04.3-4.9-minimal-odroid-xu4-20170824.img.xz
$ unxz ubuntu-16.04.3-4.9-minimal-odroid-xu4-
20170824.img.xz
$ sudo kpartx -a ubuntu-16.04.3-4.9-minimal-
odroid-xu4-20170824.img
$ sudo dd if=/dev/mapper/loop0p2 of=ubuntu-
minimal-16.04.3.img
$ sudo kpartx -d ubuntu-16.04.3-4.9-minimal-
odroid-xu4-20170824.img
```

Modifica el sistema de archivos root para el entorno invitado eliminando la configuración y el archivo específicos de ODROID:

```
$ mkdir rootfs
$ sudo mount ubuntu-minimal-16.04.3.img rootfs
$ cd rootfs
$ sudo rm ./first_boot
$ sudo rm ./etc/fstab
$ sudo touch ./etc/fstab
$ cd ..
$ sudo umount rootfs
```

Ejecuta qemu, donde el host es Ubuntu Mate 16.04.3/kernel 4.9.50, y el invitado es Ubuntu Mínimal 16.04.3/kernel 4.13.

```
$ qemu-system-arm -M vexpress-a15 -smp 2 -cpu
host
-enable-kvm -m 512 -kernel zImage -dtb
vexpress-v2p-ca15-tc1.dtb
-device virtio-blk-device,drive=virtio-blk
-drive file=ubuntu-minimal-
16.04.3.img,id=virtio-blk,if=none
-netdev user,id=user -device virtio-net-
device,netdev=user
-append "console=tty1 root=/dev/vda rw
rootwait fsck.repair=yes"
```

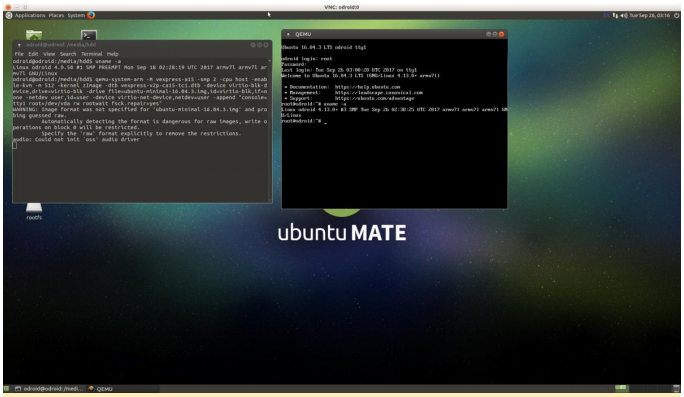


Figura 3: el sistema operativo Host ejecuta el Kernel LTS 4.9.50 mientras el sistema operativo invitado ejecuta el kernel 4.13 con flujo ascendente

Mi ODROID-C2 Docker Swarm – Parte 2: Implementando una pila en un Swarm

October 1, 2017 By Andy Yuen Docker

En la Parte 1, puse en práctica servicios en mi clúster ODROID-C2 utilizando la línea de comando Docker. Funciona, pero debería haber una mejor forma de llevar a cabo la implementación, especialmente cuando una aplicación necesita varios componentes que trabajen conjuntamente. Docker 1.13.x ha introducido la nueva función de implementación de pila Docker que permite la utilización de una pila completa de aplicaciones en el Swarm. Una pila es un conjunto de servicios que componen una aplicación. Esta nueva función implementa automáticamente múltiples servicios que están vinculados entre sí, eliminando así la necesidad de definir cada uno por separado. En otras palabras, esto es docker-componer en modo swarm. Para hacer esto, tengo que actualizar mi Docker Engine V1.12.6 que instalé usando apt-get desde el repositorio de software de Ubuntu a la V1.13.x. Teniendo ya compilado la V1.13.1 en mi ODROID-C2 cuando experimentaba sin éxito con el modo swarm hace algunos meses, tal y como comenté en mi anterior artículo, sólo es cuestión de actualizar todos mis nodos ODROID-C2 a la V1.13.1 y estaré listo para trabajar.

La pila httpd-visualizer

Lo primero que hice fue utilizar las mismas aplicaciones (httpd y Visualizer) que en mi anterior artículo usando 'docker stack deploy'. Para hacer esto, necesitaba crear un archivo yaml. En realidad esta es la versión "3" del archivo yaml de docker-compose. Esto era relativamente fácil de hacer ya que no es necesaria la persistencia de datos. Aquí tienes el archivo yaml:

```
version: "3"
services:
 httpd:
   # simple httpd demo
   image: mrdreambot/arm64-busybox-httpd
```

```
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
      resources:
        limits:
          cpus: "0.1"
          memory: 20M
    ports:
      - "80:80"
    networks:
      - httpd-net
  visualizer:
    image: mrdreambot/arm64-docker-swarm-
visualizer
    ports:
      - "8080:8080"
    volumes:
      -
"/var/run/docker.sock:/var/run/docker.sock"
    deploy:
      placement:
        constraints: [node.role == manager]
    networks:
      - httpd-net
networks:
  httpd-net:
```

Ten en cuenta que el uso de “Networks” en el archivo yaml no es estrictamente necesario. Si lo omitimos, dDocker creará una red de superposición por defecto como verás en una sección más adelante. ¡Las 2 aplicaciones, en este caso, no necesitan hablar entre ellas de todos modos! Para implementarlo, simplemente cambia al directorio donde se encuentra el archivo yaml y ejecuta el comando:

```
$ docker stack deploy -c simple-stacks.yml
httpd-dsv
```

Esto crea una pila llamada httpd-dsv. Puedes conocer el estado de la pila usando varios comandos de pila, tal y como se muestra en la Figura 1.

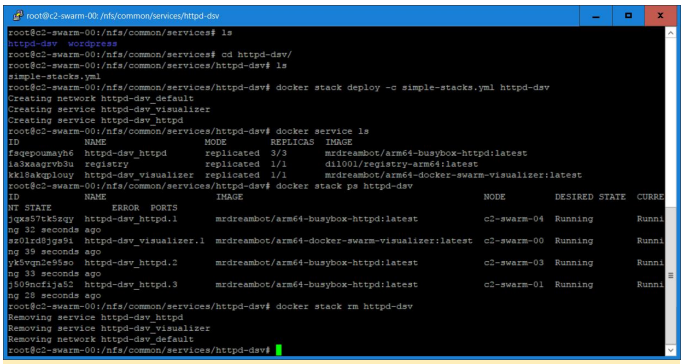


Figura 1: Comandos de pila httpd dsv

Puedes apuntar tu navegador al nodo gestor swarm o a cualquier otro nodo del swarm en el puerto 8080 para visualizar la implementación utilizando el Visualizer. La figura 2 muestra un pantallazo de la pantalla de VuShell con la visualización tomada de una implementación de pila anterior:

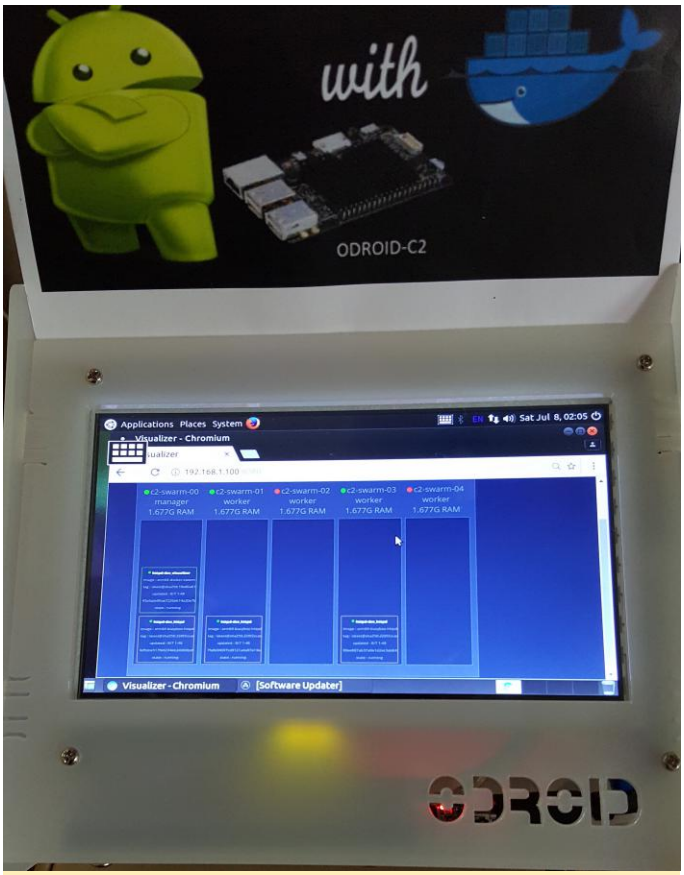


Figura 2: Visualizador VuShell

Para desmontar la pila, introduce el siguiente comando:

```
$ docker stack rm httpd-dsv
```

Migrando mi blog de WordPress a swarm

Para hacer un ejemplo más realista de una implementación de pila, he decidido realizar una

prueba migrando mi blog al swarm. Esto es muy útil para mí, ya que me permite iniciar mi blog fácilmente en otro entorno cuando ocurre un desastre. Para hacer esto, tenía que preparar unas cuantas cosas:

- Crear una copia de la base de datos de WordPress usando mysqldump para crear: mysql.dmp.
- Utilizar un editor de texto para reemplazar todas las referencias de mi dominio (mrdreambot.ddns.net) en el archivo .dmp por la dirección IP del gestor swarm que es 192.168.1.100.
- Hacer una copia (comandos Tar) del directorio /var/www/html que contiene scripts y recursos cargados
- Elegir las imágenes de docker a utilizar: mrdreambot/arm64-mysql y arm64v8/wordpress.
- Con todo lo anterior, podía crear una implementación de pila docker para mi blog de WordPress.

Estado de persistencia utilizando volúmenes

bind-mount

El primer método por el que opte fue el de usar directorios del host como volúmenes de datos (también llamados volúmenes bind-mount) para la persistencia de datos. El contenido del archivo yaml se muestra a continuación:

```
version: '3'
services:
  db:
    image: mrdreambot/arm64-mysql
    volumes:
      -
/nfs/common/services/wordpress/db_data:/u01/my
3306/data
      -
/nfs/common/services/wordpress/db_root:/root
    environment:
      MYSQL_ROOT_PASSWORD: Password456
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpressuser
      MYSQL_PASSWORD: Password456
    deploy:
      restart_policy:
        condition: on-failure
      placement:
        constraints: [node.role == manager]
  wordpress:
    depends_on:
      - db
    image: arm64v8/wordpress
    volumes:
      -
/nfs/common/services/wordpress/www_src/html:/u
sr/src/wordpress
      -
/nfs/common/services/wordpress/www_data:/var/w
ww/html
    ports:
      - 80:80
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpressuser
      WORDPRESS_DB_PASSWORD: Password456
      WORDPRESS_DB_NAME: wordpress
    deploy:
      replicas: 3
      restart_policy:
        condition: on-failure
            placement:
        constraints: [node.role == manager]
```

Las figuras 3 y 4 muestran las capturas de pantalla para la implementación de la pila.

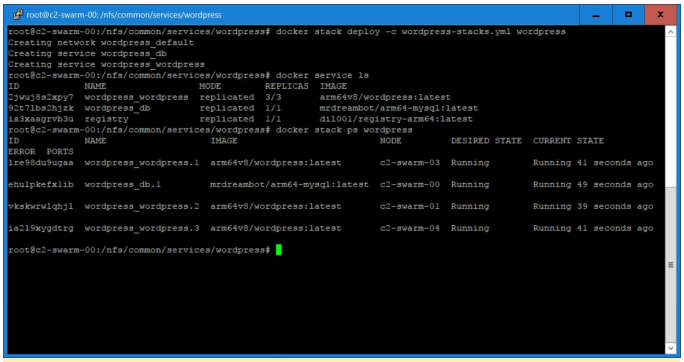


Figura 3: Implementación del volumen bind-mount de WordPress

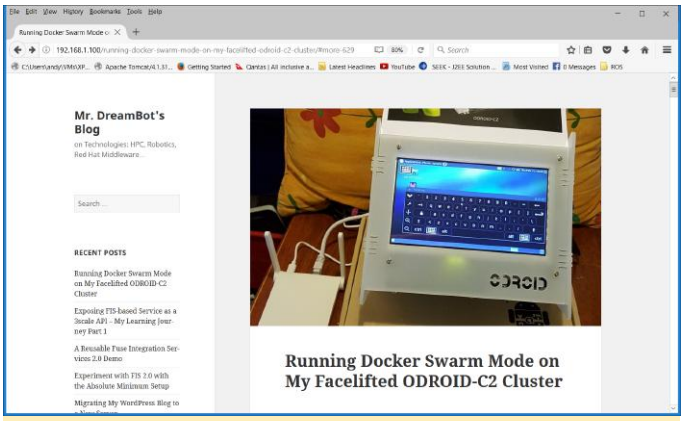


Figura 4: WordPress ejecutándose

Posiblemente habrás notado que el sitio de WordPress ha perdido parte de su apariencia

personalizada, ya que la imagen del docker arm64v8/wordpress no ofrece ninguna librería o personalización PHP. Como he comentado anteriormente, si no defines las Redes en tu archivo yaml, docker crea automáticamente una red de superposición 'wordpress_default' para la puesta en funcionamiento automática. La red de superposición es necesaria para que WordPress pueda hacer referencia a la base de datos MySQL utilizando su nombre "db" tal y como se define en el archivo yaml:

```
WORDPRESS_DB_HOST: db: 3306
```

Los volúmenes de datos justifican ciertas explicaciones. Lo primero a tener en cuenta es que todos los directorios de host utilizados como volúmenes de datos están montados en NFS y accesibles a todos los nodos del swarm.

/nfs/common/services/wordpress/db_data:/u01/my3306/data

El directorio de host /nfs/common/services/wordpress/db_data es un directorio vacío. Es el equivalente al directorio del contenedor /u01/my3306/data donde se encuentra la base de datos MySQL. A continuación, se describe como se crea su contenido.

/nfs/common/services/wordpress/db_root:/root

He completado previamente el directorio del host /nfs/common/services/wordpress/db_root con 2 archivos:

- run.sh: el script de inicio de MySQL que reemplaza al que se encuentra en el directorio /root del contenedor. Este script es el punto de entrada al contenedor MySQL. Cambié el script para buscar el archivo mysql.dmp ubicado también en /root. Si está, importa el archivo de copia en MySQL que completará el directorio /u01/my3306/data con los datos. Si no hay un archivo mysql.dmp, no se hace nada adicional al proceso habitual.
- mysql.dmp: el archivo de volcado de la base de datos MySQL de mi blog

Los cambios en el archivo run.sh en comparación con el que viene con la imagen docker MySQL se muestran a continuación:

```
...
DMP_FILE=/root/mysql.dmp
...
if [ "$MYSQL_DATABASE" ]; then
   mysql -uroot -e "CREATE DATABASE IF NOT
EXISTS `$MYSQL_DATABASE`"
   if [ -f "$DMP_FILE" ]; then
      mysql -uroot $MYSQL_DATABASE < $DMP_FILE
   fi
fi
...
```

Ten en cuenta que esto sólo es necesario cuando ejecutas el contenedor por primera vez. La implementación posterior no requerirá de esta distribución de volúmenes ya que la base de datos ha sido configurada durante la primera ejecución. Esto significa que puede comentar esta línea en el archivo yaml tras haber implementado correctamente una vez esta pila:

```
# -
/nfs/common/services/wordpress/db_root:/root
```

/nfs/common/services/wordpress/www_src/html:/usr/src/wordpress

arm64v8/wordpress activa WordPress copiando los contenidos en su directorio /usr/src/wordpress al directorio /var/www/html en el arranque si /var/www/html no tiene contenido. Al rellenar previamente el directorio del host /nfs/common/services/wordpress/www_src /html con el contenido del archivo tar creado anteriormente, arm64v8/wordpress iniciará WordPress con el contenido de mi blog. Esto solo es necesario cuando ejecutes el contenedor por primera vez. Esto significa que puede comentar esta línea en el archivo yaml tras haber implementado correctamente esta pila una vez:

```
# -
/nfs/common/services/wordpress/www_src/html:/usr/src/wordpress
```

/nfs/common/services/wordpress/www_data:/var/www/html

El directorio de host /nfs/common/services/wordpress/www_data es un directorio vacío cuyo contenido se activará mediante

el script arm64v8/wordpress tal y como se ha descrito anteriormente.

¿Por qué no usar docker-componer?

Es posible que te pregunte por qué no utilice docker-compose para ejecutar el archivo yaml, por ejemplo, usando comandos de una sola vez como sugiere la documentación de docker. La razón es que el docker-compose que instalé usando apt-get es la versión 1.8.0, que no entiende la versión 3 del archivo yaml del docker-compose, que es necesario para "docker stack deploy". Intenté compilar la última versión de docker-compose desde la fuente pero no tuve éxito. Esta es la razón por la que no estoy usando docker-componer.

Persistencia del estado usando volúmenes de almacenamiento compartido

El uso de volúmenes bind-mount depende del host. El uso de volúmenes compartidos tiene la ventaja de ser independientes del host. Puede haber un volumen compartido disponible en cualquier host en el que se inicie un contenedor siempre que tenga acceso al back-end del almacenamiento compartido y tenga instalado el plugin (controlador) del volumen adecuado, el cual te permita usar diferentes back-end de almacenamiento, como, por ejemplo: Amazon EC2, GCE, Isilon, ScaleIO, Glusterfs, solo por nombrar unos cuantos. Existes muchos drivers o plugins de volumen disponibles como Flocker, Rex-Ray, etc. Desafortunadamente, no hay binarios para esos complementos disponibles para máquinas ARM64 como ODROID-C2. Afortunadamente, el driver 'local' integrado soporta NFS. Es el driver que yo estoy usando para la implementación de volumen compartido. El archivo yaml para esto es el siguiente:

```
version: '3'
services:
  db:
    image: mrdreambot/arm64-mysql
    volumes:
      - db_data:/u01/my3306/data
#       -
/nfs/common/services/wordpress/db_root:/root
    environment:
      MYSQL_ROOT_PASSWORD: Password456
      MYSQL_DATABASE: wordpress
      MYSQL_USER: wordpressuser
      MYSQL_PASSWORD: Password456
    deploy:
      placement:
        constraints: [node.role == manager]
      replicas: 1
      restart_policy:
        condition: on-failure
  wordpress:
    depends_on:
      - db
    image: arm64v8/wordpress
    volumes:
#       -
/nfs/common/services/wordpress/www_src/html:/u
sr/src/wordpress
      - www_html:/var/www/html
    ports:
      - "80:80"
    environment:
      WORDPRESS_DB_HOST: db:3306
      WORDPRESS_DB_USER: wordpressuser
      WORDPRESS_DB_PASSWORD: Password456
      WORDPRESS_DB_NAME: wordpress
    deploy:
#       placement:
#         constraints: [node.role == manager]
      replicas: 3
      restart_policy:
        condition: on-failure
volumes:
    db_data:
      external:
        name: db_data
    www_html:
      external:
        name: www_html
```

Una vez más, los volúmenes justifican alguna explicación:

/nfs/common/services/wordpress/db_root:/root

Sirve para el mismo propósito que en el caso del volumen bind-mount. Solo es necesario cuando ejecutas la pila por primera vez para iniciar la base de datos MySQL.

/nfs/common/services/wordpress/www_src/html:/usr/src/wordpress

Sirve para el mismo propósito que en el caso del volumen bind-mount. Solo es necesario cuando

ejecutas la pila por primera vez para iniciar el contenido de WordPress.

db_data:/u01/my3306/data

db_data es un volumen compartido creado fuera de la implementación de la pila, lo que significa que es creado antes de que llegue a usarse el archivo yaml. Se utiliza para almacenar el contenido de la base de datos MySQL y no se activa en la creación.

www_html:/var/www/html

www_html es un volumen compartido creado fuera de la implementación de la pila, lo que significa que es creado antes de que llegue a utilizarse el archivo yaml. Se utiliza para almacenar el contenido de WordPress y no se activa en la creación.

Creando los volúmenes compartidos

Probablemente hayas observado una sección en el archivo yaml que dice:

```
volumes:
    db_data:
      external:
        name: db_data
    www_html:
      external:
        name: www_html
```

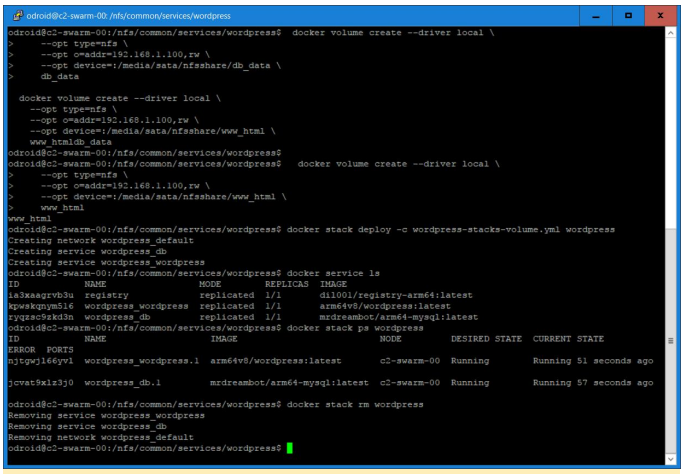
Los volúmenes compartidos db_data y www_html se crean utilizando los siguientes comandos:

```
docker volume create --driver local
    --opt type=nfs
    --opt o=addr=192.168.1.100,rw
    --opt
device=:/media/sata/nfsshare/www_html
    www_html
docker volume create --driver local
    --opt type=nfs
    --opt o=addr=192.168.1.100,rw
    --opt device=:/media/sata/nfsshare/db_data
    db_data
```

Los directorios /media/sata/nfsshare/db_data y /media/sata/nfsshare/www_htm deben existir antes de crear los volúmenes. Mi archivo /etc/exports tiene una entrada:

```
/media/sata/nfsshare
192.168.1.0/255.255.255.0(rw,sync,no_root_squa
sh,no_subtree_check,fsid=0)
```

Para probar que los volúmenes compartidos funcionan, inicialmente implementé solo 1 réplica de mySQL y 1 de WordPress en el gestor de Docker y les permití iniciar los volúmenes compartidos.



Implementación de volumen compartido de WordPress

Luego comenté las 2 líneas de la ubicación del WordPress:

```
#       placement:
#         constraints: [node.role == manager]
```

y los 2 volúmenes bind-mount

```
#       -
/nfs/common/services/wordpress/db_root:/root
#       -
/nfs/common/services/wordpress/www_src/html:/u
sr/src/wordpress
```

Después, quise implementar 3 réplicas de WordPress en varios nodos. Como estamos usando el driver “local”, debemos crear los volúmenes en cada nodo. Tal y como se muestra en la Figura 5, usé “parallel ssh” para crearlos en todos los nodos usando solo 2 comandos. La figura 5 muestra el volumen y la implementación de la pila:

Figura 5: Creando los volúmenes en los nodos

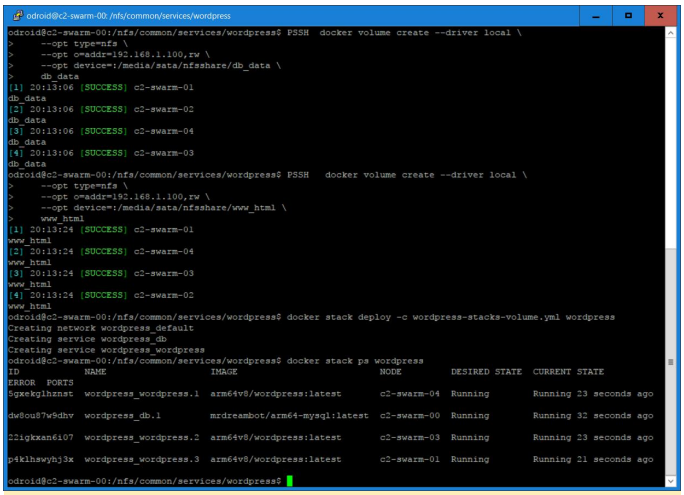


Figura 6: Implementación del volumen compartido de WordPress

Verifiqué que todas las réplicas estuvieran usando los volúmenes compartidos utilizando “docker exec -it” para acceder a los contenedores de WordPress en los nodos en los que se estaban ejecutando y examiné el contenido del directorio /var/www/html para verificar que todo funcionaba correctamente.

En el fondo, ambos planteamientos usan NFS para compartir entre los nodos. Sin embargo, los volúmenes compartidos proporcionan una abstracción independiente del host de mayor nivel que los volúmenes bind-mount. Potencialmente, puedes volver a crear los volúmenes compartidos utilizando backends de almacenamiento distintos de NFS, como AWS EC2 y Glusterfs. Bind-mount, por otro lado, está vinculado a tu sistema de archivos del host, que puede ser difícil de migrar a otro entorno.

Conclusion

Aprendí algo nuevo explorando el uso de la denominada “implementación de pila docker”. Espero que encuentres útil e informativo este artículo. Todavía hay muchas características, como las actualizaciones sucesivas, la Implementación continua/Integración continua (CI / CD), las implementaciones A/B y azul/verde, solo por nombrar unas cuantas, que aún no he explorado con mi clúster ODROID-C2 Swarm. Además, existen otros entornos de trabajo de planificación de servicios como Kubernetes y Openshift que son más frecuentes en el entorno empresarial que en el modo Docker Swarm. Exploraré otros usos del modo Swarm de Docker y alternativas al modo Swarm e informaré de mis hallazgos en el futuro cuando surja la oportunidad.

Juegos Linux: Sistema de Entretenimiento Móvil

October 1, 2017 By Tobias Schaaf Juegos, Linux

Hardkernel ha hecho un gran trabajo con el lanzamiento de un nuevo hardware recientemente. Vi la oportunidad de crear mi propio sistema de entretenimiento móvil usando algunos componentes disponibles en Hardkernel. Este proyecto es bastante sencillo y muy apropiado para principiantes, incluso para niños.

Qué es lo que necesitarás

Este proyecto está basado en el VuShell y los componentes que puedan acoplar dentro de la carcasa. De hecho, hay un poco de espacio en la carcasa, lo cual permite hacer diferentes diseños. Por ahora, me centraré en el diseño que estoy usando, aunque si quieres probar este proyecto, puedes cambiar o agregar componentes como mejor te parezca.

ODROID-VuShell (**http://bit.ly/2b8lk6a**) Como es la carcasa de nuestro proyecto, ¡ésta es absolutamente imprescindible!

ODROID-VU7 Plus (**http://bit.ly/2cmKyuN**) También podrías usar el ODROID-VU7 en su lugar (**http://bit.ly/1NWxgDx**), si quieres ahorrarte unos dólares o usar una pantalla con un consumo de energía ligeramente menor.

ODROID-C1+ (**http://bit.ly/1Upx5yI**) También puede usar un ODROID-C2 o XU4. Desafortunadamente, el C1 y el XU3 no funcionarán, ya que no cuentan con los conectores I2S necesarios.

El C1 + es probablemente tu mejor opción, ya que usa muy poca energía y te permite usar un pack de pilas. La placa alimenta el VU7 a través del conector USB 2.0 OTG, por lo que solo se necesita un conector de alimentación.

Stereo Boom Bonnet (**http://bit.ly/2wbKkyE**) Como queremos que nuestro proyecto tenga sonido, para ser realmente móviles, esto es imprescindible

5V/2A PSU Si usas un ODROID XU4, necesitarás una fuente de alimentación adicional de 5V/4A.

Tarjeta SD con 8 GB de almacenamiento o más También puede usar un módulo eMMC, pero una vez montado todo, ya no podrá acceder al módulo eMMC, haciendo que las modificaicones y/o correcciones sean imposibles sin tener que desmontarlo todo. La tarjeta SD, por otro lado, seguirá siendo accesible con unas pinzas.

Separadores Utilice los separadores de otros productos ODROID que tenía disponibles, pero también se pueden comprar por poco dinero en Amazon (**http://amzn.to/2yj4OG8**).

Teclado, Ratón Tras la configuración inicial, es posible que ya no sean necesarios.

Teniendo en cuenta la lista anterior, el coste debería estar en torno a los 160$ (sin incluir el teclado y el ratón, ni los gastos de envío).

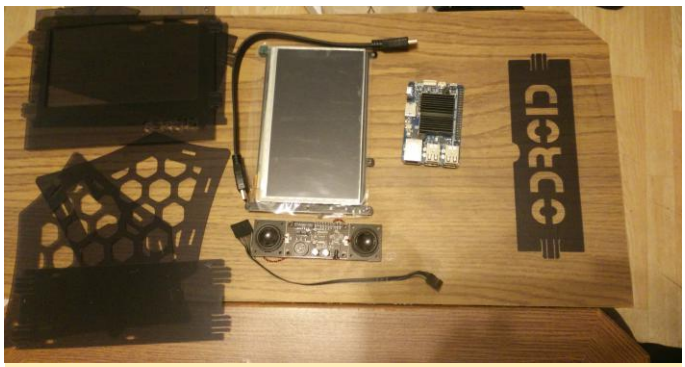


Figura 1: los componentes principales del proyecto listos para el montaje

Hay algunos otros componentes que puede que quieras conseguir, pero éstos dependen de ti completamente:

Gamepad (para una mejor experiencia de juego) Te sugiero un mando inalámbrico XBox 360 con un adaptador de PC inalámbrico, ya que un sólo adaptador admite hasta cuatro mandos, lo que significa que no tendrías que lidiar con ningún cable.

Almacenamiento externo (para almacenar grandes cantidades de datos) Por ejemplo, es posible quieras usar una memoria USB o un disco duro externo para almacenar películas o juegos. Si usas una tarjeta SD de gran tamaño (32 GB o más) no necesariamente lo necesitas, pero probablemente será más fácil cambiarlo que una tarjeta SD si te encuentras con la necesidad de contar con más espacio de almacenamiento.

Módulo WLAN Si deseas conectarse a una red inalámbrica, necesitarás uno de estos.

UPS3 o cualquier otro sistema de pilas o baterias También funcionará con la batería externa de tu teléfono móvil o tablet. De esta forma, puedes hacer que el sistema sea completamente móvil sin que necesites tener un enchufe cerca. Una batería externa decente debería permitir que el todo sistema funcionase entre 3 y 5 horas.

Micro USB-DC Power Bridge Board (**http://bit.ly/2wbWQ1e**) Si usas un ODROID-XU4, esto asegurará que la potencia para la pantalla sea constante.

*IR Remote Controller (**http://bit.ly/1M6UGiR**) o cualquier otro sistema de control remoto por inflarrojos* Los C1 + y C2 vienen con un receptor IR. Si quieres usarlo con Kodi, es algo que también puedes hacer.

Juego de soldadura Esto es recomendados para usuarios avanzados que quieran tener un sonido estéreo más “realista”

Software

Antes de empezar a montar los componentes, debes configurar tu ODROID, instalar el sistema operativo (yo utilice mi propia imagen ODROID GameStation Turbo para la serie ODROID-C1), preparar el boot.ini y, si quieres, instar juegos , películas y lo que quieras en tu placa. Es mejor hacerlo al principio, ya que puede ser difícil hacerlo con posterioridad si no tienes una conexión de red.

Asegúrate de configurar los paramentos para VU7 o VU7 Plus (dependiendo de la pantalla LCD que elijas) en el boot.ini:

```
$ setenv m "1024x600p60hz" # 1024x600

$ # HDMI DVI Mode Configuration
$ # setenv vout_mode "hdmi"
$ setenv vout_mode "dvi"
$ # setenv vout_mode "vga"
```

También puedes configurar el sistema para cargar los módulos necesarios para el Stereo Boom Bonnet.

Abre un terminal y escribe los siguientes comandos:

```
$ su -
$ echo "snd-soc-pcm5102 snd-soc-odroid-dac" >>
/etc/modules
```

Después de esto, puedes copiar los juegos o las películas que quieras, y configurar Emulation Station, Kodi y cualquier otra cosa que desees, o puede hacer esto más tarde una vez que el todo el sistema esté montado.

Sin duda alguna necesitarás como mínimo configurar boot.ini o de lo contrario no verás nada en la pantalla.

Montaje

El montaje es bastante fácil, simplemente sigue los pasos de Hardkernel sobre cómo ensamblar el VuShell (**http://bit.ly/2b8lk6a**) con algunas pequeñas modificaciones.

Figura 2 – Fijando el ODROID-C1+ en la parte posterior del Vu7 Plus

Una vez que coloques la pantalla en el frontal y añadas el primer lateral sobre la placa (Paso 7), es hora de realizar algunas modificaciones. Primero, conecta el Stereo Boom Bonnet a la placa. Para hacerlo, dobla suavemente las partes que sostienen los altavoces hasta que se separen y tengas la placa y el altavoz por separado. Desenchufa el cable de los altavoces. Lo mejor es conectar el cable del Stereo Boom Bonnet antes de montarlo. Consulta la guía de Hardkernel para asegurarte de colocar el cable correctamente (**http://bit.ly/2xuWVjA**).

Retira el tornillo que se añadió en el paso 3 del montaje del VuShell en el lateral del VU7 Plus y reemplázalo con un par de separadores. Coloca el Stereo Boom Bonnet boca abajo sobre los espaciadores. Utiliza el tornillo que retiraste originalmente para fijar el Stereo Boom Bonnet. Usa (4) separadores M3 de 20 mm para elevar el Stereo Boom Bonnet, de modo que el control del volumen se alinee con uno de los agujeros del VuShell, lo cual te permitirá regular el volumen.

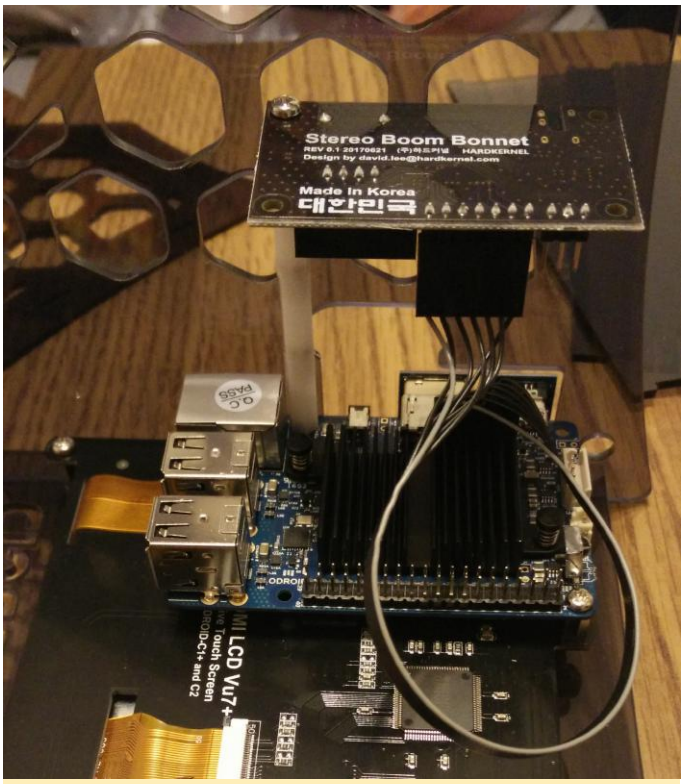


Figura 3: Stereo Boom Bonnet conectado boca abajo con separadores sobre el ODROID-C1 +

Tras acoplar el primer lateral, puede hacer lo mismo con el otro. Ten en cuenta que el orificio superior del C1+ normalmente no está unido a la carcasa, como se puede ver en el paso 5 del montaje del VuShell. Si colocas un separador aquí, no te preocupes si no está atornillado en el zócalo. Funcionará igual de bien sin él. Una vez montada la segunda línea de separadores en el C1 + y Stereo Boom Bonnet, puedes conectar el primer altavoz que viene con el Stereo Boom Bonnet.

Alinea el altavoz con uno de los agujeros de la carcasa VuShell. Yo utilicé cinta adhesiva transparente para sujetar el altavoz a la carcasa en mi primera prueba. Más tarde, usé super-glue para fijarlo mejor a la carcasa. Técnicamente, con un altavoz es suficiente para disponer de un sonido bastante bueno, pero si quieres, puedes fijar el segundo altavoz a otro de los agujeros del mismo lateral.

Si quieres tener un sonido estéreo más realista, necesitarás alargar el cable del segundo altavoz para poder ubicarlo en otro lateral del VuShell. Ten en cuenta que esto requiere algo de soldadura, de modo

que, aunque es bastante fácil, se debe hacer con cuidado, y en el caso de niños, éstos deben estar supervisados por un adulto.

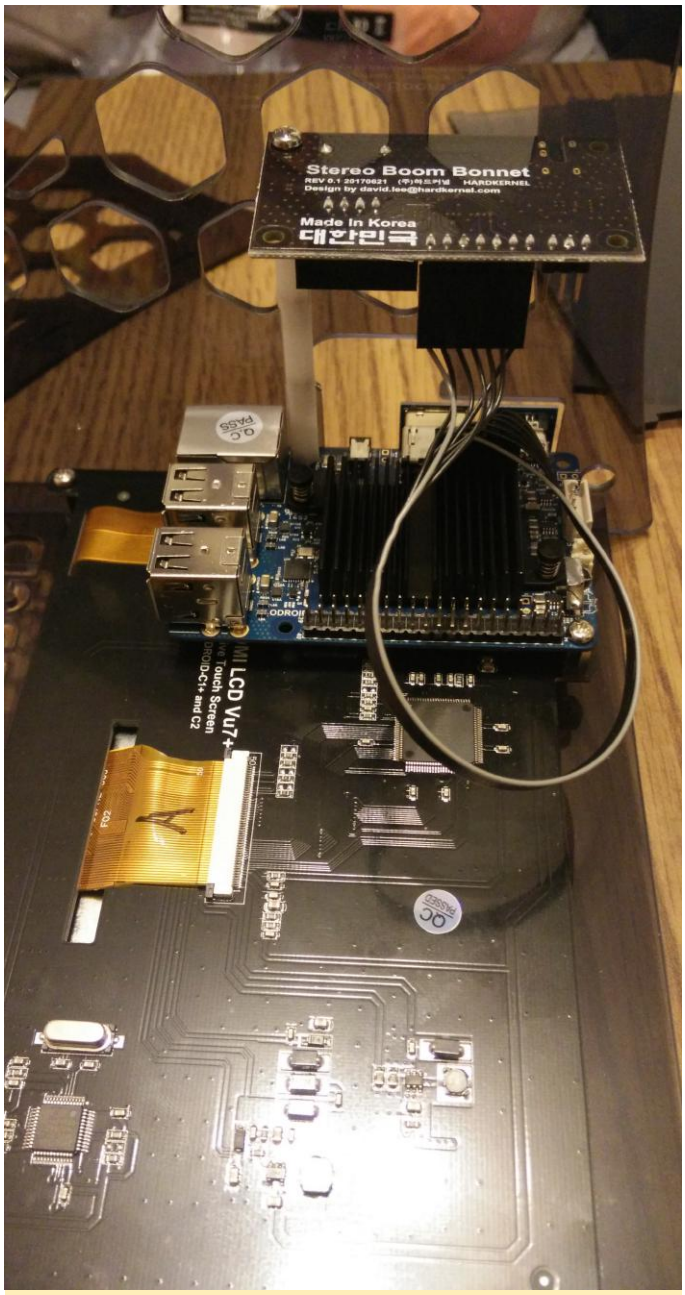


Figura 4 y 5: Me quedé sin separadores de 20 mm y cambié a los de 10 mm. No se veía nada bonito, Por favor usa mejor los de 20mm. No seáis tan vagos como yo.

Incluso con solo un altavoz conectado, el sonido debería ser lo suficientemente bueno para ver películas o jugar. Hice un video donde probé la reproducción del video con ffmpeg **http://bit.ly/2xox1wb**. En este video, subo y bajo el volumen usando el control de volumen, el cual es fácil de alcanzar gracias a los separadores. Después de esto, también probé algunos sonidos antiguos de 8 bits iniciando Cave Story desde EmulationStation (**http://bit.ly/2xlDxGo**). También funcionó perfectamente. Disponer de un único altavoz conectado realmente no representaba un gran problema.

Montaje Avanzado

Como habrás notado, el cable del segundo altavoz es demasiado corto para llegar al otro extremo del VuShell. Por lo tanto, necesitaba alargar el cable para poder alcanzar el otro lateral de la carcasa. Este proceso es bastante sencillo y probablemente incluso lo puedan hacer los niños, pero siempre con la supervisión de un adulto

Necesitarás equipamiento básico de soldadura. Principalmente, un cable adicional, estaño para soldar y un poco de envoltura de cable termo retráctil (**http://amzn.to/2wH9edI**) si la tienes. Lamentablemente, yo no las tenía. Puede funcionar sin ella, pero es mejor tener la envoltura termo retráctil para proteger los cables una vez se hayan soldado. También necesitarás algo para cortar el cable. Un cortador de cable/alambre funcionará muy bien, como los cables son bastante delgados, posiblemente bastará con unas tijeras o incluso un cuchillo.

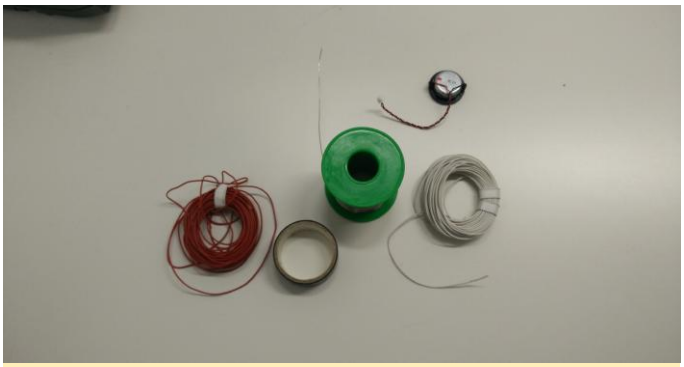


Figura 6 – Equipo de soldadura y un segundo altavoz

Cuando dispongas de todos los elementos, puedes empezar desenrollando los cables cerca del altavoz hasta una longitud de aproximadamente 5 cm (2 pulgadas). Luego corta el cable con el cortador de alambres y deja al descubierto los alambres blancos.

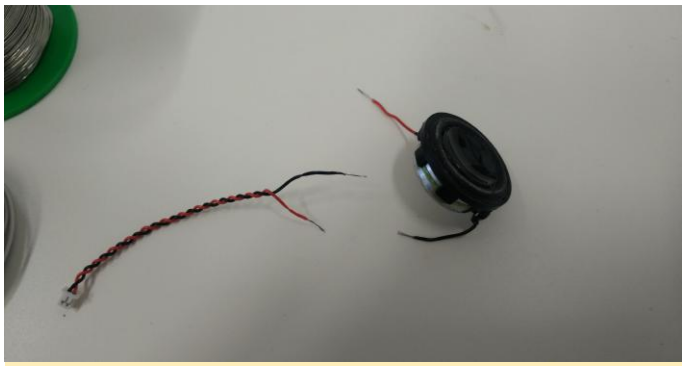


Figura 7: No cortes los cables demasiado cerca de los altavoces, por si tienes que volver a empezar de nuevo. Enrosca los extremos del cable que están al descubierto.

Corta dos alambres más largos de unos 20-25 cm (8-10 pulgadas). Recomiendo encarecidamente usar diferentes colores para los cables para saber así que cable es el que hay que conectar con el otro. Asegúrate de que los dos cables que cortas tengan la misma longitud. También sugiero usar cables finos similares a los de los altavoces. Los míos eran ligeramente más gruesos y se ajustaban perfectamente.

Después de cortar los cables, deja al descubierto los extremos retirando lentamente la funda del cable. Ten cuidado de no cortar el cable durante el proceso. Una vez hecho esto, retuerce los alambres descubiertos para que se mantengan unidos. Luego, puedes aplicar estaño a los extremos, cubriendo los alambres descubiertos de una capa delgada de estaño. Este también sería un buen momento para colocar la envoltura retráctil en los cables ampliados (dos para cada cable). Después, puedes soldar los extremos del cable. Asegúrate de conectar los cables correctos.

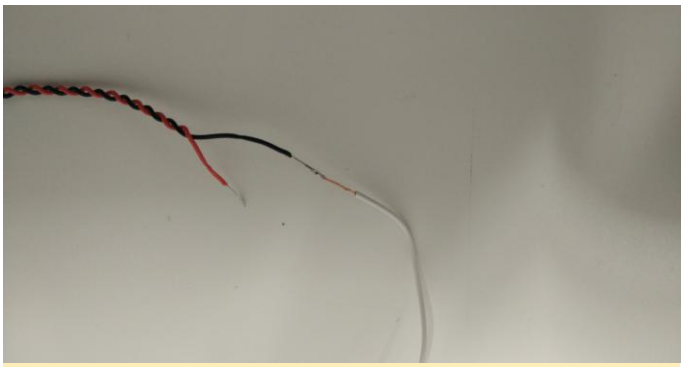
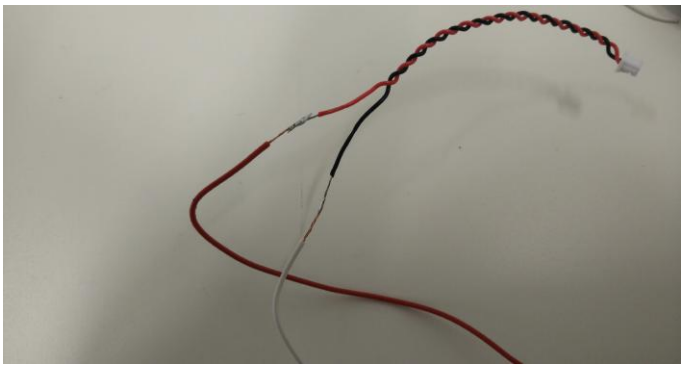


Figura 8 y 9: Une los extremos del cable con los cables de ampliación, uno al lado del otro.



Después de soldar un extremo del cable, puedes conectar el otro extremo al altavoz.

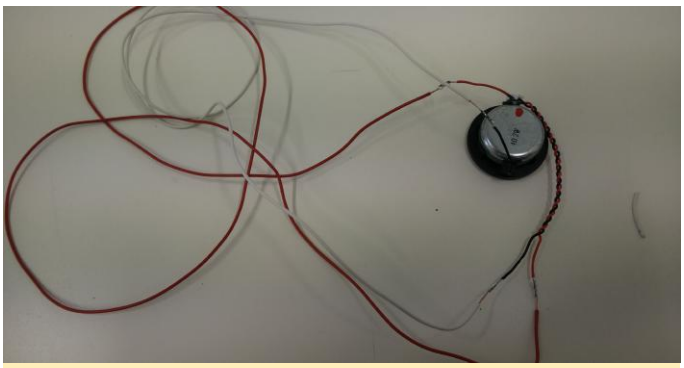


Figura 10: ambos extremos están conectados y el altavoz ahora cuenta con un largo cable con el que poder trabajar.

Al final, retorcí el cable como lo estaba el cable original, así era más fácil de manejar. Esto realmente me llevo un poco de tiempo, pero el resultado fue bueno y me facilito bastante el montaje en el VuShell. Sin embargo, asegúrate de no tensar demasiado los puntos de soldadura cuando retuerzas el cable, o se pueden separar.

Ahora también sería un buen momento para colocar las fundas termorretráctiles en los extremos descubiertos del cable y calentarlas para que sellen los alambres. Intenté hacer lo mismo con cinta aislante, pero los cables eran demasiado finos para

envolverlo correctamente. Una vez que termines de retorcer el cable, debería verse como una versión más larga del cable original, con algunos puntos de soldadura.

Figura 11: Asegúrate de que los cables coincidan en los extremos

Ahora es el momento de juntar toda la unidad y colocar el nuevo altavoz dentro del VuShell. Cuando montes los altavoces, el conector del altavoz en la parte superior es para el canal izquierdo y el conector del altavoz en la parte inferior es para el canal derecho. También puedes usar algunos videos de YouTube para probar si los altavoces izquierdo y derecho están conectados en el orden adecuado. Puede fijar el altavoz izquierdo con pegamento super glue o cinta adhesiva.

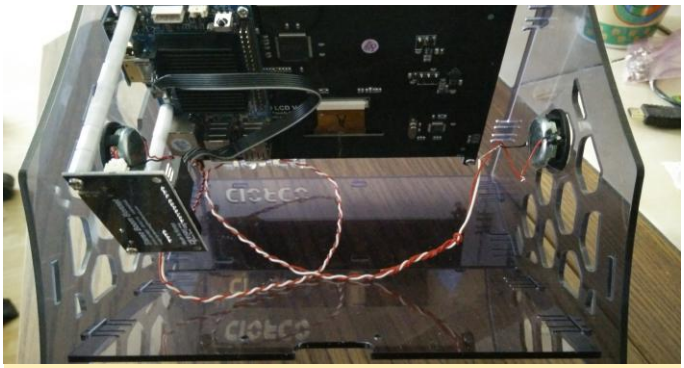


Figura 12: Los altavoces están montados y todavía queda mucho espacio en la carcasa

Después de esto, encendí el dispositivo y lo probé para ver si ambos altavoces funcionaban correctamente desde el principio (http://bit.ly/2xuF4ct). Debido a que hay mucho espacio dentro de la carcasa, puedes añadir componentes adicionales con bastante facilidad. Tal y como muestras las instrucciones de montaje de Hardkernel, hay agujeros para colocar un disco duro o una batería externa en el interior, lo que haría que el dispositivo fuese totalmente móvil.

Figura 13: Esta batería de 12500 mA debería proporcionarte entre 3-5 horas de entretenimiento móvil para jugar, ver películas o escuchar música

También podrías colocar fácilmente un adaptador de PC inalámbrico XBox 360 junto con la batería externa. De esta forma, podrías usar hasta 4 mandos XBox 360 al mismo tiempo sin tener que añadir un nuevo cable. Esto es fabuloso para controlar Kodi o EmulationStation sin la necesidad de un teclado.

Conclusión

Este es un proyecto divertido y fácil. Algunas personas ya están disfrutando de esta pequeña consola, afirmando que están sorprendidos por la idea y la movilidad que tiene gracias a la posibilidad de acoplarle una batería externa. Dado que el VuShell tiene mucho espacio, este proyecto puede tener muchas variantes dependiendo de los accesorios adicionales que quieras usar. Es posible que incluso desee eliminar los altavoces por completo y, en su lugar, simplemente uses el conector para auriculares de Stereo Boom Bonnet, el cual te permitiría jugar a tus juegos en un viaje en tren o en un avión durante varias horas.

Aunque no es la pantalla más grande, es lo suficientemente buena como para tener un par de amigos sentados a tu lado viendo algunas películas en una excursión, o jugar algunos juegos divertidos o competitivos en uno de los muchos emuladores. Algunos preferirán la potencia extra de un XU4 para jugar seriamente a algunos juegos de PSP, Dreamcast o N64, mientras que otros se conformarán con algunos de Nintendo, Super Nintendo, SEGA Genesis u otros clásicos en un C1. Colocarlo en la cocina con Android y un C2 te permitirá escuchar tu música favorita mientras cocinas. Gracias a la pantalla táctil, todo lo que necesitas está al alcance de tus dedos.

En definitiva, las opciones son casi ilimitadas y es muy fácil de hacer. Incluso los niños pueden montar su propia consola. Te invito a intentarlo y a que comentes lo que puedes llegar a hacer con este sistema todo en uno.

Cómo instalar ArchLinux con Full Disk Encryption en un ODROID-C2

October 1, 2017 By @YesDay Linux, ODROID-C2, Tutoriales

La Full Disk Encryption (FDE) protege nuestros datos del acceso no autorizado en el caso de que alguien logre acceder físicamente a los medios de almacenamiento. En este artículo, voy a describir cómo instalar ArchLinux con Full Disk Encryption en el OROID-C2. El método de encriptación es LUKS con el tamaño de clave XTS 512 bit (AES-256).

En pocas palabras, Full Disk Encryption requiere lo siguiente:

- Encriptar una partición y copiar el sistema de archivos root en ella.
- El kernel para incluir el módulo dm_crypt. En nuestro caso, éate ya está incluido por defecto. De modo que, no necesitaremos volver a re-compilar el kernel.
- Los initramfs para incluir el módulo del dm_crypt y el binario cryptsetup. Usamos una herramienta llamada dracut para generar los initramfs necesarios. Dracut soporta la funcionalidad requerida a través de los módulos adicionales crypt y lvm.
- Pasar las **opciones dracut para LUKS** a los initramfs a través de la propiedad bootargs dentro de boot.ini. Por ejemplo, digamos que, en nuestro caso, queremos que los initramfs desbloqueen un volumen LUKS con UUID ae51db2d-0890-4b1b-abc5-8c10f01da353 y carguen el sistema de archivos root del dispositivo mapper/dev/mapper/vg-root. Para pasar estas opciones dracut configuramos lo siguiente:

```
sudo nano /boot/boot.ini
setenv bootargs "rd.luks.uuid=ae51db2d-0890-
4b1b-abc5-8c10f01da353 root=/dev/mapper/vg-
root rootwait < leave the rest as is >"
```

Note

Muchos de los pasos de este documento implican editar archivos de configuración. Para mantener las palabras al mínimo, usamos la notación anterior como una forma concisa de describir los pasos en los

que hay que editar archivos. La notación anterior significa:

- Debes editar el archivo /boot/boot.ini con privilegios root (sudo nano /boot/boot.ini). Nano es el editor de línea de comandos; sin embargo, puedes usar cualquier otro editor.
- Busca la línea que comienza con setenv bootargs y añade o edite las opciones de configuración rd.luks.uuid=ae51db2d-0890-4b1b-abc5-8c10f01da353 root=/dev/mapper/vg-root rootwait. Es posible que algunos archivos mencionados en este documento tengan la correspondiente línea comentada o no presente en absoluto. Si ese es el caso, tendrá que descomentar o añadir la línea en el archivo, respectivamente.
- Deja el resto de la línea después de rootwait como está.

Además, para una configuración sin monitor ni teclado, necesitarás habilitar el desbloqueo remoto mediante SSH como se describe en el artículo "Remotely unlock the LUKS rootfs during boot using Dropbear sshd" en **http://bit.ly/2g6qjDv**. Por último, pero no menos importante, si prefieres utilizar de inmediato esta funcionalidad, simplemente descárgate la imagen de sistema operativo en **http://bit.ly/2xR8LDe**. De cualquier modo, el documento actual te proporcionará más detalles técnicos sobre los componentes básicos y cómo trabajan conjuntamente en un entorno Full Disk Encryption.

Requisitos de hardware

- ODROID-C2
- Una maquina Linux desde donde garbar la imagen del SO e interactuar con el ODROID-C2
- Disco USB con al menos 4GB de capacidad
- Una tarjeta microSD o módulo eMMC con al menos 4GB de capacidad
- (Opcional) Un kit módulo USB-UART para conectarse con la consola serie del ODROID-C2. Consulta la publicación de **http://bit.ly/2fM29BB** para obtener instrucciones sobre cómo conectarte y por qué la consola serie es muy recomendable en este caso.

Grabar la imagen del SO y arrancar ODROID-C2

Graba la imagen del SO en el disco USB siguiendo las instrucciones de **http://bit.ly/2fGKEik** Reemplaza /dev/mmcblk0 en las siguientes instrucciones por el nombre del dispositivo de la tarjeta microSD tal como aparece en tu ordenador. Si están montadas, desmonta las particiones de la tarjeta microSD

```
$ lsblk
$ umount /dev/mmcblk0p1
$ umount /dev/mmcblk0p2
```

Zero al principio de la tarjeta microSD:

```
$ sudo dd if=/dev/zero bs=1M count=8
of=/dev/mmcblk0
$ sync
```

Usando una herramienta como GParted, crea una tabla de particiomes MBR/msdos y dos particiones en la tarjeta microSD:

- Partición ext4 con un tamaño de 128M
- Partición lvm2 ocupando el resto del espacio (no es necesario formatearla todavía)

A continuación, copia el contenido del directorio /boot del disco USB en la primera partición de la tarjeta microSD:

```
$ sudo cp -R /media/user/usb-disk/boot/*
/media/user/micro-sd-card-part1/
```

Crea un enlace simbólico para la ruta de acceso boot.ini codificada de alarm/uboot-odroid-c2 (**http://bit.ly/2xbEdPo**):

```
$ cd /media/user/micro-sd-card-part1
$ sudo ln -s . boot
```

Luego, graba los archivos del gestor de arranque:

```
$ sudo ./sd_fusing.sh /dev/mmcblk0
```

Determina el UUID del disco USB:

```
$ sudo lsblk -o name,uuid,mountpoint
NAME UUID MOUNTPOINT
sdb
└─sdb1 2b53696c-2e8e-4e61-a164-1a7463fd3785
/media/user/usb-disk
```

Ten en cuenta que, si hay UUID duplicados entre las particiones del disco USB y la tarjeta microSD, debes eliminar los duplicados para evitar futuros conflictos:

```
$ sudo tune2fs /dev/sda2 -U $(uuidgen)
```

Configura boot.ini para arrancar desde el disco USB. Para hacerlo, usa el UUID del paso anterior y configura el boot.ini de la tarjeta microSD:

```
$ sudo nano /media/user/micro-sd-card-part1/boot.ini
$ setenv bootargs "root=UUID=2b53696c-2e8e-4e61-a164-1a7463fd3785 rootwait "
```

Desmonta, ejecuta sync varias veces y retira la tarjeta microSD y el disco USB de la maquina Linux. Conecta la tarjeta microSD y el disco USB al ODROID-C2, luego inicia el ODROID-C2 y conéctate a su consola serie. Si necesitas instrucciones sobre cómo conectarte a la consola serie, consulte el artículo de **http://bit.ly/2fM29BB**.

Si todo va bien, deberías arrancar con el disco USB. Ten en cuenta que si root=UUID= 2b53696c-2e8e-4e61-a164-1a7463fd3785 no funciona, intenta root=/dev/sda1, root=/dev/ sdb1 o el nombre del dispositivo que veas en la consola antes de que falle el arranque (p. ej., [14.812393] sd 1:0:0:0: [sda] Attached SCSI removable disk). Si todavía tienes problemas, intenta reiniciar unas cuantas veces y/o reposiciona el disco USB en un puerto USB diferente en el ODROID-C2. No te preocupes sí parece que te está causando problemas, ya que no tendrá que arrancar de nuevo con el disco USB tras el primer arranque exitoso.

A continuación, verifica que el sistema de archivos root esté montado desde el disco USB:

```
$ df -h
```

Cambiar contraseñas

Cambia las contraseñas para el usuario root y alarm. Las credenciales predeterminadas son alarm/alarm y root/root.

```
$ passwd
$ su
$ passwd
```

Instalar los paquetes requeridos

```
$ su
$ pacman -Syu
$ pacman -S --needed sudo python git rsync lvm2 cryptsetup
```

(Opcional) Configura sudo sin contraseña para el usuario alarm:

```
$ echo 'alarm ALL=(ALL) NOPASSWD: ALL' > /etc/sudoers.d/010_alarm-nopasswd
```

Instalar dracut

Instala pacaur (http://bit.ly/2yEjAaY):

```
$ sudo pacman -S --needed base-devel cower
$ mkdir -p ~/.cache/pacaur && cd "$_"
$ cower -d pacaur
$ cd pacaur
$ makepkg -si --noconfirm --needed
```

Instala dracut usando la herramienta pacaur:

```
$ pacaur -S dracut
```

Verifica la instalación de dracut listando los módulos

```
$ dracut --list-modules
```

Si el comando “pacaur -S dracut” informa de un error en el que la arquitectura aarch64 no es compatible con el paquete, sigue estos pasos para configurar el soporte para aarch64:

```
$ cd ~/.cache/pacaur/dracut/
$ nano PKGBUILD # replace `arch=("i686" "x86_64")` with `arch=("aarch64")`
$ makepkg -si --noconfirm --needed
```

Si makepkg informa de un error así como dracut-046.tar ... FAILED (unknown public key 340F12141EA0994D), introduce estos comandos e inténtalo de nuevo:

```
$ gpg --full-gen-key
$ gpg --recv-key 340F12141EA0994D
```

Consulta la documentación de Makepkg para obtener más información en **http://bit.ly/2wuuBe6**.

Si el comando “gpg -full-gen-key” informa del error, Key generation failed: No pinentry, sigue los siguientes pasos para configurar gpg tal y como se describe en **http://bit.ly/2yDAJBye** e inténtalo de nuevo. El gpg-agent necesita saber cómo solicitar el usuario para la contraseña:

```
$ nano ~/.gnupg/gpg-agent.conf
$ pinentry-program /usr/bin/pinentry-curses
$ gpg-connect-agent reloadagent /bye
```

Si makepkg reportas errores sobre dependencias que faltan, actualiza los paquetes e inténtalo nuevamente.

```
$ sudo pacman -Syu
$ pacaur -Syua
```

Prepare los rootfs LUKS

Encripta la segunda partición de la tarjeta microSD (consulta también las opciones recomendadas para LUKS en **http://bit.ly/2yF15D2**):

```
$ sudo cryptsetup -v -y -c aes-xts-plain64 -s
512 -h sha512 -i 5000 --use-random luksFormat
/dev/mmcblk0p2
```

-v = información detallada -y = verifica la contraseña, pregunta dos veces y reporta un error si no coinciden -c = especifica el cifrado utilizado -s = especifica el tamaño de la clave utilizada -h = especifica el hash utilizado -i = número de milisegundos para gastar en el procesamiento de contraseña (si usa algo más que sha1, debe ser mayor que 1000) –use-random = generador de números aleatorios para usar luksFormat = para activar la partición y fijar una contraseña /dev/mmcblk0p2 = la partición a encriptar

Desbloquea el dispositivo LUKS y móntalo en /dev/mapper/lvm:

```
$ sudo cryptsetup luksOpen /dev/mmcblk0p2 lvm
```

Crea el volumen primario, el grupo de volumen y volumen lógico:

```
$ sudo pvcreate /dev/mapper/lvm
$ sudo vgcreate vg /dev/mapper/lvm
$ sudo lvcreate -l 100%FREE -n root vg
```

Crea el sistema de archivos:

```
$ sudo mkfs.ext4 -O ^metadata_csum,^64bit
/dev/mapper/vg-root
```

Monta el nuevo volumen root encriptado (volumen lógico):

```
$ sudo mount /dev/mapper/vg-root /mnt
```

Copia el volumen root existente al nuevo volumen root encriptado. Una instalación de 1.5 GB se completa en aproximadamente 6 minutos con una microSD de gama media:

```
$ sudo rsync -av
--exclude=/boot
--exclude=/mnt
--exclude=/proc
--exclude=/dev
--exclude=/sys
--exclude=/tmp
--exclude=/run
--exclude=/media
--exclude=/var/log
--exclude=/var/cache/pacman/pkg
--exclude=/usr/src/linux-headers*
--exclude=/home/*/.gvfs
--exclude=/home/*/.local/share/Trash
/ /mnt
```

Si las claves SSH del host están vacías, elimínalas para que se regeneren la próxima vez que se inicie sshd. Esto evitará el problema de la pérdida de memoria tal y como se describe en **http://bit.ly/2xQxGqe**.

```
$ sudo rm /mnt/etc/ssh/ssh_host*key*
```

Crea algunos directorios y monta la partición de arranque:

```
$ sudo mkdir -p /mnt/boot /mnt/mnt /mnt/proc
/mnt/dev /mnt/sys /mnt/tmp
$ sudo mount -t ext4 /dev/mmcblk0p1 /mnt/boot
```

Registra el volumen encriptado en crypttab

```
$ sudo bash -c 'echo lvm UUID=$(cryptsetup
luksUUID /dev/mmcblk0p2) none luks>>
/mnt/etc/crypttab'
```

Configura fstab:

```
$ sudo nano /mnt/etc/fstab
$ /dev/mapper/vg-root / ext4 errors=remount-
```

```
ro,noatime,discard 0 1
$ /dev/mmcblk0p1 /boot ext4 noatime,discard 0
2
Next, generate a new initramfs using dracut.
The following commands will add the dracut
modules crypt and lvm to the initramfs. These
modules will prompt for LUKS password during
boot and unlock the LUKS volume. Note that the
order of the modules is important:

$ sudo dracut --force --hostonly -a "crypt
lvm" /mnt/boot/initramfs-linux.img
```

A continuación, determina el UUID de LUKS:

```
$ sudo cryptsetup luksUUID /dev/mmcblk0p2
470cc9eb-f36b-40a2-98d8-7fce3285bb89
```

Configura las opciones root dracut y rd.luks.uuid en bootargs. Estas desbloquearán el volumen LUKS y cargará los rootfs durante el arranque:

```
$ sudo nano /mnt/boot/boot.ini
$ setenv bootargs "rd.luks.uuid=470cc9eb-f36b-
40a2-98d8-7fce3285bb89 root=/dev/mapper/vg-
root rootwait "
```

Ten en cuenta que en el paso anterior, NO se elimina el resto de bootargs, básicamente reemplaza root = UUID = 2b53696c-2e8e-4e61-a164-1a7463fd3785 por rd.luks.uuid=470cc9eb-f36b-40a2-98d8-7fce3285bb89 root=/dev/mapper/vg-root y no toques el resto de bootargs. Luego, desmonta y reinicia los rootfs de LUKS:

```
$ sudo umount /mnt/boot
$ sudo umount /mnt
$ sudo reboot
```

Si todo va bien, se te pedirá que introduzcas la contraseña de LUKS durante el arranque. A continuación, verifica los rootfs de LUKS:

```
df -h
output
Filesystem Size Used Avail Use% Mounted on
devtmpfs 714M 0 714M 0% /dev
tmpfs 859M 0 859M 0% /dev/shm
tmpfs 859M 8.3M 851M 1% /run
tmpfs 859M 0 859M 0% /sys/fs/cgroup
/dev/mapper/vg-root 1.7G 1.4G 256M 85% /
tmpfs 859M 0 859M 0% /tmp
/dev/mmcblk0p1 120M 26M 86M 23% /boot
tmpfs 172M 0 172M 0% /run/user/1000
```

A continuación, desbloquea de forma remota los rootfs de LUKS durante el arranque usando Dropbear sshd. Reemplace 10.0.0.100 en las siguientes instrucciones por la dirección IP asignada al ODROID-C2 por tu servidor DHCP local. Usa la herramienta fing para encontrar la dirección IP asignada (por ejemplo, sudo fing 10.0.0.1/24). Luego, asegúrate de que el demonio SSH se está ejecutando:

```
$ sudo systemctl status sshd
$ journalctl -u sshd -n 100
```

Si los comandos anteriores informan que sshd falla con un error de asignación de memoria, introduce los siguientes comandos:

```
$ sudo rm /etc/ssh/ssh_host*key*
$ sudo systemctl start sshd
```

Consulta el artículo de **http://bit.ly/2xQxGqe** para obtener más información sobre las pérdidas de memoria en sshd.

Instalar y configurar Dropbear

Instala el módulo crypt-ssh de dracut:

```
$ pacaur -S dracut-crypt-ssh-git
```

Desde tu maquina Linux, copia la clave pública SSH al archivo appconf/dracut-crypt-ssh/authorized_keys en el servidor remoto ODROID-C2:

```
$ cat ~/.ssh/*.pub | ssh alarm@10.0.0.100
'umask 077; mkdir -p appconf/dracut-crypt-ssh;
touch appconf/dracut-crypt-
ssh/authorized_keys; cat >>appconf/dracut-
crypt-ssh/authorized_keys'
```

Despues, configura el módulo crypt-ssh:

```
$ sudo nano /etc/dracut.conf.d/crypt-ssh.conf
$ dropbear_acl="/home/alarm/appconf/dracut-
crypt-ssh/authorized_keys"
```

Genera un nuevo initramfs usando dracut. Los siguientes comandos añadirán la red de módulos dracut y crypt-ssh a initramfs. Ten en cuenta que el orden de los módulos es importante:

```
$ sudo dracut --force --hostonly -a "network
crypt lvm crypt-ssh" /boot/initramfs-linux.img
```

Habilita el acceso a la red durante el arranque añadiendo las opciones ip y rd.neednet de dracut a bootargs:

```
$ sudo nano /boot/boot.ini
setenv bootargs "rd.neednet=1
ip=10.0.0.100::10.0.0.1:255.255.255.0:archlinu
x-luks-host:eth0:off rd.luks.uuid=ae51db2d-
0890-4b1b-abc5-8c10f01da353
root=/dev/mapper/vg-root rootwait "
```

Si prefieres DHCP en lugar de una ip estática, simplemente reemplaza con ip=dhcp. Consulta la documentación de red de dracut en http://bit.ly/2g6XCXky y las opciones dracut en **http://bit.ly/2yUBFT6** para obtener más opciones (man dracut.cmdline). Reinicia para que se inicie Dropbear, permitiendo el desbloqueo remoto:

```
$ sudo reboot
```

Desde tu maquina Linux, conéctate al servidor remoto Dropbear SSH que se ejecuta en el ODROID-C2:

```
$ ssh -p 222 root@10.0.0.100
```

Desbloquea el volumen (te pedirá la contraseña y enviarla a la consola):

```
$ console_auth
Passphrase:
```

Si el desbloqueo del dispositivo tuvo éxito, initramfs se limpiará por sí mismo y Dropbear finalizará por si sólo y tu conexión.

También puede escribir “console_peek” que muestra lo que hay en la consola. También existe el comando unlock, pero encontramos un problema cuando realizamos las pruebas, tal y como se describe en **http://bit.ly/2fHB2nw**.

En algunos casos se requiere alimentar la entrada de datos automáticamente con el comando interactivo console_auth. Desde tu maquina Linux, desbloquea el volumen:

```
$ ssh -p 222 root@10.0.0.100 console_auth <
password-file
```

or

```
$ gpg2 --decrypt password-file.gpg | ssh -p
222 root@10.0.0.100 console_auth
For additional security, you might want to
only allow the execution of the command
console_auth and nothing else. To achieve
this, you need to configure the SSH key with
restricting options in the authorized_keys
file. From your Linux box, copy the public SSH
key, with restricting options, to the
appconf/dracut-crypt-ssh/authorized_keys file
on the remote ODROID-C2 server:
$ (printf 'command="console_auth",no-agent-
forwarding,no-port-forwarding,no-pty,no-X11-
forwarding ' && cat ~/.ssh/*.pub) | ssh
alarm@10.0.0.100 'umask 077; mkdir -p
appconf/dracut-crypt-ssh; touch
appconf/dracut-crypt-ssh/authorized_keys; cat
>appconf/dracut-crypt-ssh/authorized_keys'
```

Consulta la documentación de Dropbear para obtener una lista completa de las opciones restringentes. Antes de continuar, podría ser una buena idea crear una copia de initramfs:

```
$ sudo cp /boot/initramfs-linux.img
/boot/initramfs-linux.img-`date +%y%m%d-
%H%M%S`
```

En una configuración sin monitor y teclado, examina cuidadosamente las opciones restringentes para evitar bloquearte.

Finalmente, genera un nuevo initramfs usando dracut:

```
$ sudo dracut --force --hostonly -a "network
crypt lvm crypt-ssh" /boot/initramfs-linux.img
```

En este caso, puede desbloquear el volumen de forma interactiva simplemente escribiendo el siguiente comando:

```
$ ssh -p 222 root@10.0.0.100
```

Ten en cuenta que, al escribir el comando anterior, el comando console_auth es activado automáticamente en el servidor remoto e inmediatamente solicita la

contraseña, como si simplemente se escribiera ssh -p 222 root@10.0.0.100 console_auth. Mientras escribes la contraseña, se mostrará en la pantalla en texto plano. Por lo tanto, debe evitar desbloquear interactivamente cuando el acceso está restringido al comando console_auth. Cuando presiones enter, se desconectará sin importar si la contraseña era correcta o no. Mientras que con el inicio de sesión no restringido (ver http://bit.ly/2hHAGl0), solo se desconectaría si la contraseña es correcta, lo que significa que te aparecerán comentarios sobre si el desbloqueo ha tenido exito o no. Por otro lado, para desbloquear el volumen usando un archivo de contraseña, desde tu maquina Linux escriba el siguiente comando:

```
$ ssh -p 222 root@10.0.0.100 < password-file
```

o

```
$ gpg2 --decrypt password-file.gpg | ssh -p
222 root@10.0.0.100
```

Para comentarios, preguntas o sugerencias, visita la publicación original del blog en **http://bit.ly/2xMQE3I**.

Referencias

ArchLinux dm-crypt/Encrypting an entire system (**http://bit.ly/2xPaybR**)

Cómo instalar Debian con Full Disk Encryption en ODROID-C2 (**http://bit.ly/2g6JtcF**)

Módulo LCD I2C: Utilizando la LCD serie 1602 16×2 TWI

October 1, 2017 By Miltiadis Melissas ODROID-C2, Mecaniqueo

Después de haber hecho tantos proyectos de IoT con mi ODROID-C2 como el detector sismógrafo (**http://bit.ly/2uWqas0**), el sistema de conservación y notificación para bodegas (**http://bit.ly/2wch3Vb**), el notificador mecánico de Gmail (**http://bit.ly/2wch3Vb**) y muchos otros, estaba pensando en añadir una pantalla LCD de bajo consumo y coste para mostrar cualquier información valiosa de todas estas creaciones electrónicas teniendo en cuenta la portabilidad y la legibilidad. La pantalla del módulo LCD serie I2C TWI 1602 16×2 para Arduino JD es la solución ideal para visualizar datos técnicos y mucha más información

Esta pantalla módulo LCD se comunica con un ODROID-C2 utilizando el protocolo I2C con solo 4 cables. El protocolo I2C es un bus informático serie, multi-maestro, multi-esclavo, mono-terminal con intercambio de paquetes, inventado por Philips Semiconductor (ahora NXP Semiconductors). Normalmente se utiliza para conectar ICs periféricos de baja velocidad a procesadores y microcontroladores en comunicaciones a corta distancia, dentro de la placa (**http://bit.ly/2qGiYP4**). En las siguientes líneas, vamos a describir cómo poder materializar físicamente y en forma de programa esta conexión. El lenguaje utilizado es Python 2.7, y el programa se puede implementar fácilmente en otros proyectos como un módulo con pequeñas modificaciones.

Hardware

Necesitarás todos los accesorios habituales del ODROID-C2:

- ODROID-C2
- Tarjeta MicroSD con el último Ubuntu 16.04 proporcionado por HardKernel (**http://bit.ly/2rDOCfn**)
- La librería WiringPi para controlar los GPIO de un ODROID-C2 que ejecuta Ubuntu 16.04 (las instrucciones de Hardkernel sobre cómo instalar la librería las puedes encontrar en **http://bit.ly/1NsrlU9**)

- Teclado
- Pantalla
- Cable HDMI
- El teclado, la pantalla y el cable HDMI son opcionales ya que puedes acceder a tu ODROID-C2 desde tu ordenador de escritorio a través de SSH
- Alimentación por Micro USB o, mejor aún, la fuente de alimentación proporcionada por Hardkernel (**http://bit.ly/1X0bgdt**)
- Opcional: Batería externa con UBEC (3A max, 5V), si deseas que el dispositivo funcione de forma autónoma (consulta la Figura 1). Hardkernel ofrece una solución mejor con el UPS3 diseñado específicamente para ODROID-C2. Puede comprar el UPS3 desde su tienda en este enlace: **http://bit.ly/2l1rE25**. El UPS3 es una muy buena opción, ya que ofrece al detector la posibilidad de trabajar de forma autónoma con mayor estabilidad y duración
- Cable Ethernet o Dongle USB Wifi
- El kit Tinkering C para Ubuntu, que se puede comprar en Hardkernel (**http://bit.ly/1NsrlU9**)
- Pantalla modulo LCD Serie I2C TWI 1602 16×2 para Arduino JD, que se puede encontrar en varios sitios, como eBay

Para el cableado, sigue el esquema de la figura 1. Hay dos cables que son muy importantes para la comunicación: el SDA que proporciona los datos en serie de I2C y el SCL que proporciona el registro del tiempo en serie del I2C. El SDA está en el Pin 3 en la pantalla LCD I2C y está conectado al Pin 3 GPIO del ODROID-C2. El SCL está en el Pin 4 y está conectado al Pin 5 GPIO del ODROID-C2. A modo de referencia, echa un vistazo al esquema de la Figura 1 y al excelente diseño de 40 pines de Hardkernel para el ODROID-C2 (**http://bit.ly/2aXAlmt**). Esto te ayudará a colocar el cableado correctamente. Ahora que tenemos nuestro hardware listo, veamos cómo podemos establecer una comunicación entre el ODROID-C2 y la pantalla LCD serie I2C utilizando el protocolo I2C. El Pin 2 GPIO proporciona la potencia VCC, +5V, para la pantalla LCD y el Pin 39 GPIO es, por supuesto, la toma a tierra, GND.

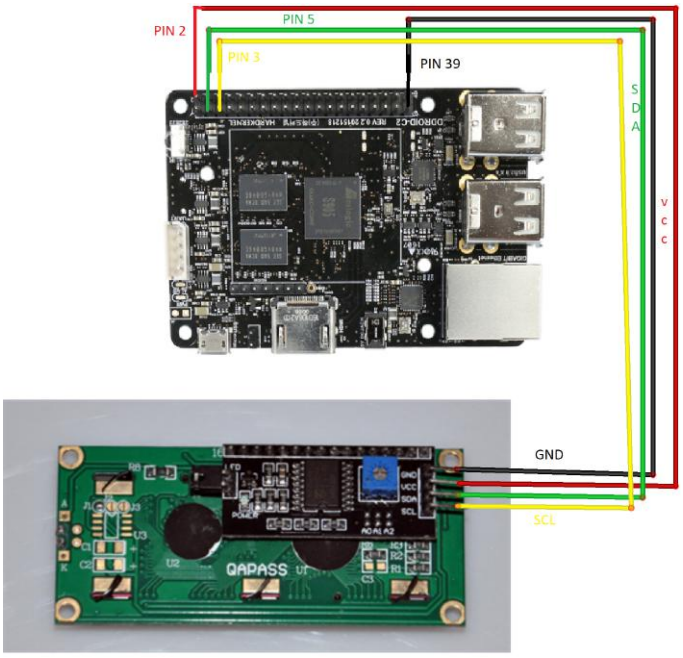


Figura 1: Diagrama del cableado

Iomunicción I2C

Vamos a establecer una conexión entre ODROID-C2 y la pantalla LCD en serie utilizando el protocolo I2C. Los pasos que vamos a seguir son casi idénticos a los detallados en nuestro anterior artículo bajo el título "Detector sismógrafo de terremotos: Midiendo la aceleración sísmica utilizando el ODROID-C2", publicado en la edición de julio de ODROID Magazine (**http://bit.ly/2uWqas0**). En ese artículo, describimos todos los pasos necesarios para establecer la comunicación entre el ODROID-C2 y el acelerómetro MMA7455, que también usa I2C. Repetiremos el mismo procedimiento con el fin de garantizar la coherencia y la integridad del artículo.

Todos los comandos se introducen en una ventana de terminal o a través de SSH. Primero, deberás actualizar ODROID-C2 para asegurarte de que tienes instalados los últimos paquetes:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get dist-upgrade
```

Luego, necesitarás reiniciar el ODROID-C2:

```
$ sudo reboot
```

Tendrás que instalar SMBus y I2C-Tools, ya que la pantalla del módulo LCD utiliza este protocolo para

comunicarse con el ODROID-C2. El Bus de administración del sistema, o SMBus, es un simple bus mono-terminal de dos hilos para comunicaciones livianas. Suele encontrarse con frecuencia en la placas base de los ordenadores para comunicarse con la fuente de alimentación (http://bit.ly/2rAWhuU).

Una vez que hayas iniciado sesión en tu ODROID-C2 desde la línea de comandos, ejecuta el siguiente comando para instalar Python-SMBus y I2C-Tools:

```
$ sudo apt-get install pythonsmbus
```

Configura ODROID-C2 para cargar el driver I2C:

```
$ modprobe ami-i2c
```

Configurar el ODROID-C2 para iniciar I2C automáticamente en el arranque editando /etc/modules:

```
$ sudo nano /etc/modules
```

Usa las teclas de cursor para moverte a la última línea y añade una nueva línea con el siguiente texto:

```
$ i2c-dev
```

Presiona intro, luego añade:

```
$ aml_i2c
```

Guarda tus cambios y salte del editor nano. Para evitar tener que ejecutar las herramientas I2C como root, agrega el usuario "ODROID" al grupo I2C:

```
$ sudo adduser Odroid 12c
```

Después reinicia el ODROID-C2:

```
$ sudo reboot
```

Una vez que hayas reiniciado tu ODROID-C2, tendrás soporte para I2C. Puedes verificar si hay dispositivos I2C conectados con el siguiente comando:

```
$ sudo i2cdetect -y -r 1
```

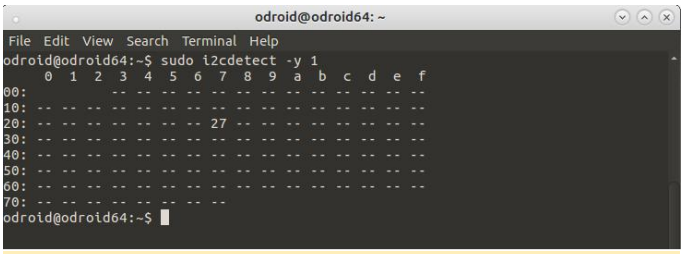


Figura 2: Dispositivos I2C detectados utilizando i2cdetect

Si aparece '27' en la línea 20 de la columna 7, significa que la pantalla LCD se está comunicando con el ODROID-C2 y funciona correctamente. Puedes encontrar más detalles en http://bit.ly/2qCQM1s.

Software Pyton

Vamos a presentar el código por fragmentos, como lo hacemos siempre, para que nuestros lectores lo entiendan mejor. El código de aquí esta modificado ligeramente con respecto al código fuente (http://bit.ly/2w2a957) que ha sido adaptado a las necesidades de este proyecto. El código está en Python y lo que principalmente hace es establecer una conexión entre el ODROID-C2 y la pantalla LCD abriendo una conexión I2C que permite visualizar 16 caracteres en dos líneas. Puede descargar el código desde aquí (http://bit.ly/2vzSMqd) y ejecutarlo para obtener resultados inmediatos, o si simplemente no deseas volver a escribir todo el código. Primero, importa los módulos necesarios:

```
import smbus
import time

# Define device parameters
I2C_ADDR = 0x27 # I2C device address, if any
error,
 # change this address to 0x3f
LCD_WIDTH = 16 # Maximum characters per line

# Define device constants
LCD_CHR = 1 # Mode - Sending dataLCD_CMD = 0 #
Mode - Sending command

LCD_LINE_1 = 0x80 # LCD RAM address for the
1st line
LCD_LINE_2 = 0xC0 # LCD RAM address for the
2nd line
LCD_LINE_3 = 0x94 # LCD RAM address for the
3rd line
LCD_LINE_4 = 0xD4 # LCD RAM address for the
```

```
4th line

LCD_BACKLIGHT = 0x08 # On
ENABLE = 0b00000100 # Enable bit

# Timing constants
E_PULSE = 0.0005
E_DELAY = 0.0005

#Open I2C interface
bus = smbus.SMBus(1) # Open I2C interface for
ODROID-C2

# Initialise display
def lcd_init():
 lcd_byte(0x33,LCD_CMD) # 110011 Initialise
 lcd_byte(0x32,LCD_CMD) # 110010 Initialise
 lcd_byte(0x06,LCD_CMD) # 000110 Cursor move
direction
 lcd_byte(0x0C,LCD_CMD) # 001100 Display
On,Cursor Off, Blink Off
 lcd_byte(0x28,LCD_CMD) # 101000 Data length,
number of lines, font size
 lcd_byte(0x01,LCD_CMD) # 000001 Clear display
 time.sleep(E_DELAY)

# Send byte to data pins
 # (#bits = the data, #mode = 1 for data or 0
for command)
def lcd_byte(bits, mode):
 bits_high = mode | (bits & 0xF0) |
LCD_BACKLIGHT
 bits_low = mode | ((bits<<4) & 0xF0) |
LCD_BACKLIGHT

 bus.write_byte(I2C_ADDR, bits_high) # High
bits
 lcd_toggle_enable(bits_high)

 bus.write_byte(I2C_ADDR, bits_low) # Low bits
 lcd_toggle_enable(bits_low)

# Toggle enable
def lcd_toggle_enable(bits):
 time.sleep(E_DELAY)
 bus.write_byte(I2C_ADDR, (bits | ENABLE))
 time.sleep(E_PULSE)
 bus.write_byte(I2C_ADDR,(bits & ~ENABLE))
 time.sleep(E_DELAY)

# Send string to display
def lcd_string(message,line):
 message = message.ljust(LCD_WIDTH," ")

 lcd_byte(line, LCD_CMD)

 for i in range(LCD_WIDTH):
 lcd_byte(ord(message[i]),LCD_CHR)

# Main program block, # Initialize display
def main():
 lcd_init()

# Send text to I2C TWI 1602 16x2 Serial LCD
Module Display
while True:

 lcd_string("***ODROID-C2***",LCD_LINE_1)
 lcd_string("ODROID-magazine ",LCD_LINE_2)

 time.sleep(3)

 lcd_string("***HardKernel***",LCD_LINE_1)
 lcd_string("*hardkernel.com*",LCD_LINE_2)

 time.sleep(3)

# Handling keyboard interrupts and exception
utility
if __name__ == '__main__':

 try:
 main()
 except KeyboardInterrupt:
 pass
 finally:
 lcd_byte(0x01, LCD_CMD)
```

Ejecutando el código

El código anterior se puede escribir en cualquier editor de texto. Sin embargo, es más fácil hacerlo con un IDE de Python, como Python IDLE. El Python IDLE es accesible desde el escritorio Mate (Aplicación -> Programación -> IDLE). Tan pronto como escribamos el programa, podemos guardarlo con cualquier nombre y finalmente, ejecutarlo como se muestra en la Figura 3:

```
$ sudo python lcd16x2i2c.py
```

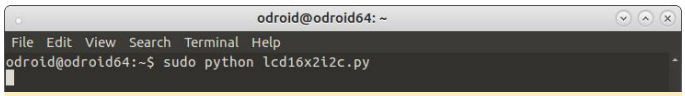


Figura 3: Resultado del programa Python

Los mensajes se presentan en el módulo LCD de forma secuencial, 2 líneas de cada vez.

Figura 4 – Pantalla LCD mostrando un mensaje de dos lineas

Conclusión

La aplicación “Drive I2C LCD screen with ODROID-C2” puede implementarse en cualquier otro proyecto con pequeñas modificaciones en un módulo Python. La única parte del código que debes modificar para cambiar las líneas de caracteres que se muestran en la pantalla LCD es la siguiente:

```
# Send text to I2C TWI 1602 16x2 Serial LCD
Module Display
while True:

 lcd_string("***ODROID-C2***",LCD_LINE_1)
 lcd_string("ODROID-magazine ",LCD_LINE_2)

 time.sleep(3)

 lcd_string("***HardKernel***",LCD_LINE_1)
 lcd_string("**hardkernel.com",LCD_LINE_2)

 time.sleep(3)
```

No dude en hacer cambios en el código y en añadir funciones adicionales a cualquier otro proyecto que puedas crear.

GamODROID-C0: Una Consola de Juegos Retro portátil Basada en ODROID

October 1, 2017 By Julien Tiphaine Juegos, ODROID-C0, Mecaniqueo

Este artículo va de otra consola de juegos portátil casera a modo de continuación de la primera que desarrollé (**http://bit.ly/2yFj4th**). En mi primer desarrollo, utilicé un ODROID-W (clon de pi) y una carcasa GameBoy nueva. Para este nuevo proyecto, deseaba algo más potente para ejecutar juegos de N64, Dreamcast y PSX, y también algún que otro juego nativo de Linux. No hay muchas opciones de bajo consumo con suficiente CPU + GPU para esto, así que elegí un ODROID-C0. Además, en lugar de usar y transformar una carcasa existente, utilicé una impresión 3D diseñada por mí con un tamaño y unas dimensiones muy optimizadas. Quisiera dar las gracias a la comunidad ODROID (forum.ODROID.com) y en particular a @meveric por su distribución debian y los paquetes optimizados para ODROID.

Componentes

Aquí tienes una lista de todos los componentes que utilice en este desarrollo:

Componentes principales:

- ODROID-C0
- Módulo eMMC de 8GB
- MicroSD XC de 128 Gb (SanDisk Ultra, XC I, clase 10)
- Pantalla TFT NTSC/PAL de 3.5” (**http://bit.ly/2yUyXgd**)
- Una Placa PCB prototipo de 4x6cm

Componentes para el Audio:

- Amplificador de audio estéreo 2.8W clase D
- 2 altavoces PSP 2000/3000
- Una tarjeta de sonido USB barata con un pequeño cable USB

Componentes para la Batería:

- 2 baterías de LiPo: Keeppower 16650 3.7v 2500mA con protección
- 2 conectores MOLEX, 50079-8100
- 2 receptáculos MOLEX, 51021-0200

Componentes para los Controles:

- 12 Pulsadores táctiles de 8 mm (**http://bit.ly/2xN8qDW**)
- 4 Botones Interruptores táctiles de 6 mm (**http://bit.ly/2xNmlcU**)
- 2 Sticks analógicos PSP 1000
- 1 Multiplexor analógico MC14051BCL

Componentes para la Refrigeración

- 2 Disipadores de calor de cobre PS3 GPU
- 4 disipadores de calor de cobre de 15x15mm
- Un poco de relleno térmico de 1 mm
- Un poco de Pasta Térmica de Silicio

Otros Componentes Electrónicos:

- Un led azul de 3 mm
- Unos cuantos alambres de un viejo cable plano IDE
- Algunos cables de conexión de placa
- 3 Resistencias

Componentes para Adornar:

- Plantillas de esmalte de uñas para los colores (negro, amarillo, rojo, verde, azul)
- Papel de lija 200, 600 y 1200
- XTC 3D (**http://bit.ly/2fG3l5O**)
- Pintura en espray satinada blanca

Consumo de energía previsto

Las fuentes principales de consumo de energía son el ODROID-C0, la pantalla y el sistema de audio (tarjeta de sonido y amplificador de audio). Antes de empezar, decidí medir el consumo de estos 3 componentes:

- ODROID-C0: 200-400 mAh dependiendo del uso de la CPU y de la GPU
- Sistema de audio: 310 mAh
- Pantalla: 420 mAh

Hace un total de 1130 mAh a 5v, de modo que son 5650mA/hora. Las baterías que usé son de (al menos) 3.7v x 5000 mA con un total de 18500 mA. La consola debería durar más de 3h en todos los casos.

¿Por qué usar una pantalla con tan poca resolución?

Existen varios motivos para ello: baja potencia, 60 FPS, cableado sencillo y tiene poca definición aparentando un televisor antiguo, lo que hace que el suavizado de bordes por hardware sea muy bueno.

¿Por qué usar baterias en forma de cilindro? Se trata de una cuestión de optimización del espacio en relación a la capacidad que quería. Usar una batería plana más clásica me habría obligado a ampliar la profundidad de la carcasa en más de 2 cm, aunque esa fue mi primera intención.

¿Por qué usar una placa prototipo para montar los componentes adicionales?

El objetivo era montar con facilidad todos los componentes como si fueran una única placa base, y realmente puedo decir que resulto muy útil.

¿Por qué es necesario un multiplexor analógico?

El ODROID-C0 solo proporciona 2 entradas analógicas, y una ya se usa para informar del nivel de batería. Por lo tanto, solo había 1 entrada analógica disponible para un total de 4 ejes analógicos (2 sticks con 2 direcciones cada uno). La única forma de leer los 4 ejes analógicos con una sola entrada analógica era con un multiplexor. Afortunadamente, el ODROID-C0 tiene suficientes pins digitales para usar 2 de ellos y así poder alternar los canales analógicos.

¿Por qué usar un módulo eMMC en lugar de una microSD?

El módulo eMMC es mucho más rápido que una microSD. Permite que la consola se inicie en unos pocos segundos incluso con Xorg, un gestor de ventanas, y Emulation Station con muchos juegos. Utilizo el eMMC para el sistema operativo y la microSD para los juegos y las vistas previas de los videos.

Carcasa impresa en 3D

La carcasa de la consola se ha modelado con Freecad. La diseñé específicamente para este proyecto, con un tamaño muy concreto para la placa base y todos los componentes. Fue mi primer modelo 3D y la primera impresión 3D, así que puede tener errores. No obstante, los archivos Freecad están disponibles en GitHub (http://bit.ly/2fGJWRU) y los archivos STL están distribuidos libremente en Thingverse en (http://bit.ly/2xW9FAh).

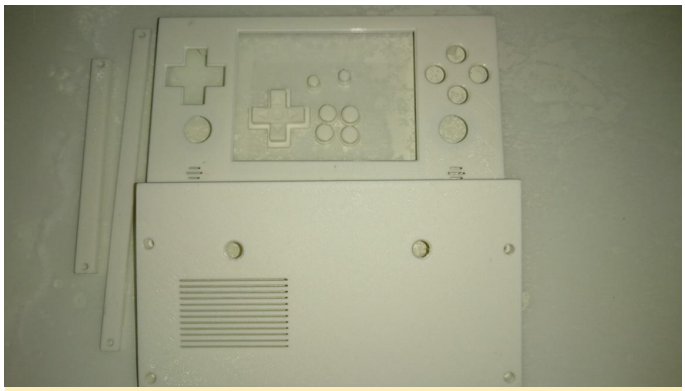


Figura 1 – Vista interna del frontal de la carcasa. Los puntos negros son marcas para hacer agujeros para la inclinación

Figuras 2 y 3 – Vista interna posterior de la carcasa. Se puede ver el espacio de las baterías en la parte inferior y algunas ranuras para la disipación térmica de la CPU + GPU.

La carcasa es muy similar a una Nintendo DS. Puede que no sea tan evidente, pero usar las dimensiones de una consola tan conocida me permitió encontrar fundas de protección buenas y baratas. Como podrás ver en las fotos de más adelante, utilicé una funda NDS para proteger mi GamODROID-C0, que encontré por unos pocos euros.

Para conseguir un buen acabado, primero utilicé el papel de lija 600 y 1200 en todas las piezas. Luego, usé un producto llamado XTC-3D. Es asombroso y ofrece un aspecto brillante, pero todavía no tenía el acabado que perseguía. Utilicé de nuevo el papel de lija 1200 antes de usar una pintura satinada blanca. Finalmente, esto me dio el acabado que ves en las fotos.

Para las piezas más pequeñas como botones y el D-pad, utilicé un poco de esmalte de uñas. Es muy barato y realmente proporciona un acabado genial. Para finalizar barnicé los botones y el D-pad con un barniz de uñas transparente para proteger los colores, ya que éstos son muy utilizados en la consola.

Figura 4 – Montaje del hardware

Mi objetivo era desarrollar una placa base de una única pieza para hacerla más robusta y fácil de colocar dentro de la carcasa. También desarrollé pequeñas placas para los botones Inicio/selección y el D-pad.

Montar la pantalla

El sistema es aproximadamente el mismo que el que use para mi consola Retroboy (http://bit.ly/2yFj4th).

Sin embargo, existen algunas diferencias en lo que respecta al conector: V-in y la salida compuesta han sido invertidas esta vez. La figura 5 muestra la pantalla original, tal como se encuentra en el sitio web de Adafruit.

Figura 5: Pantalla de 3,5”

Primero retiré el conector blanco, conecté el V-in directamente a la salida del regulador de voltaje y añadí dos cables para llevar la alimentación a través de uno de los pins ODROID 5V, como se muestra en la Figura 6.

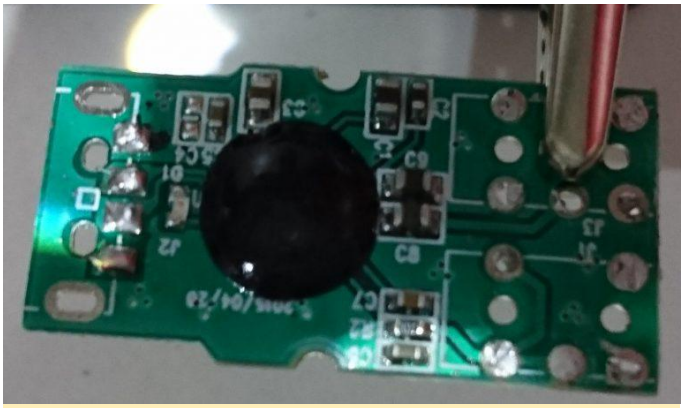
Figura 6 – Primer plano del cableado 5V de ODROID

Tarjeta de sonido

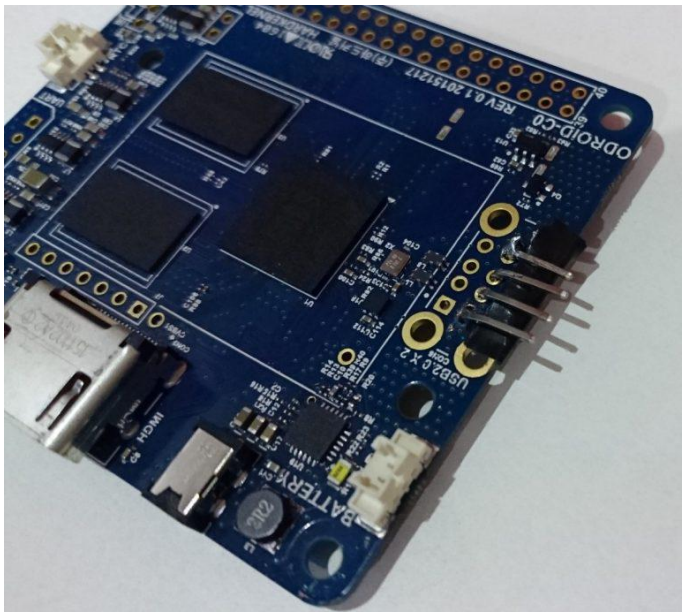
Elegí una tarjeta de sonido USB barata que tuviera un cable entre la placa y el conector USB. Esto era importante porque así resultaba más fácil retirar la soldadura.

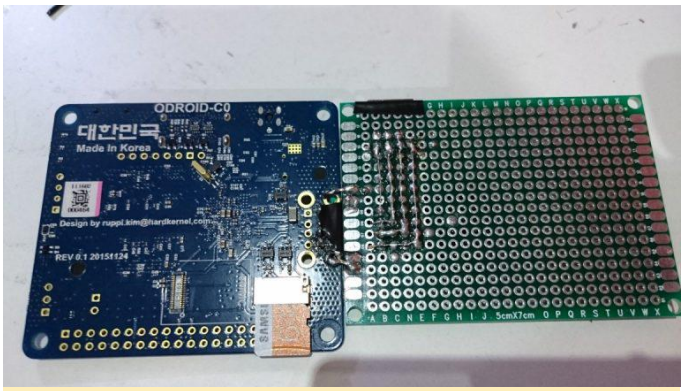
Figura 7 – Tarjeta de sonido USB

Empecé a desmontar los cables, los conectores y volví a perforar los agujeros. Preparé la placa ODROID añadiendo los pins al primer conector USB, tal y como se muestra en las Figuras 8 y 9. Finalmente, soldé la tarjeta de sonido directamente a los pins, como se muestra en la Figura 10.

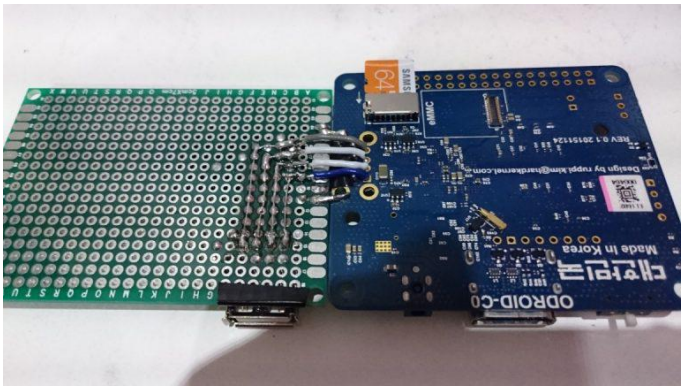


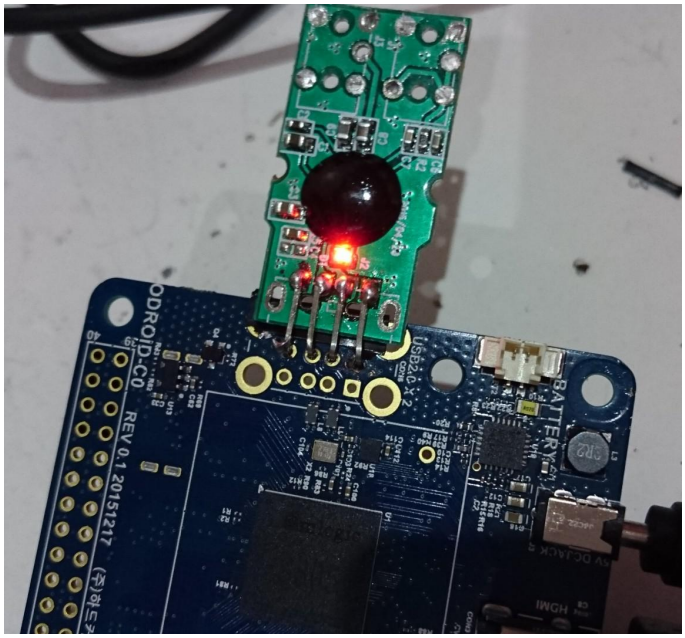
Figuras 8, 9 y 10 – Modificaciones en la tarjeta de sonido





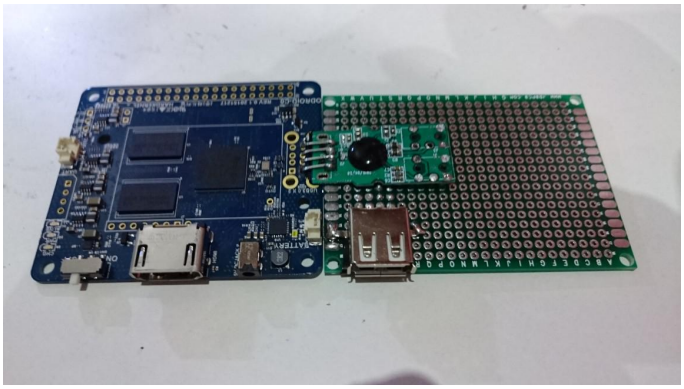
Figuras 11, 12 y 13 – Fijando la placa de ampliación al ODROID





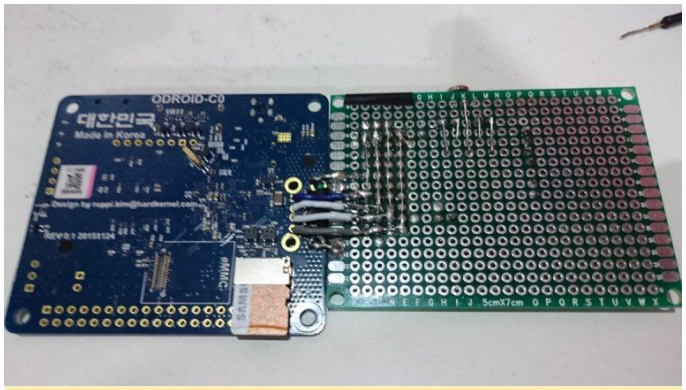
Placa de ampliación con puerto USB

Puse la placa de ampliación justo debajo de la tarjeta de sonido USB. Primero soldé un conector USB, luego lo conecté al segundo conector USB del ODROID a través de la placa de ampliación. Ten en cuenta que también soldé la placa de ampliación a la placa base ODROID para hacer que todo fuese más robusto.

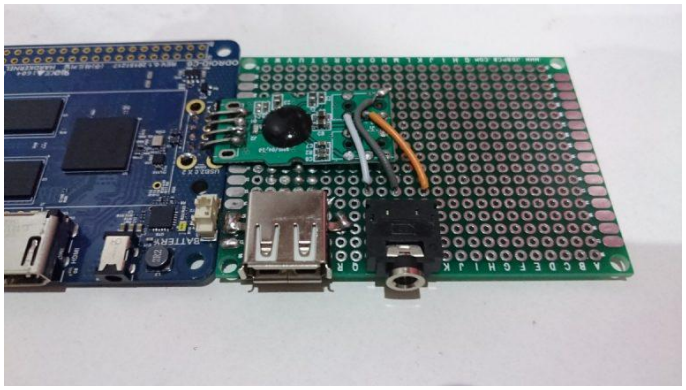


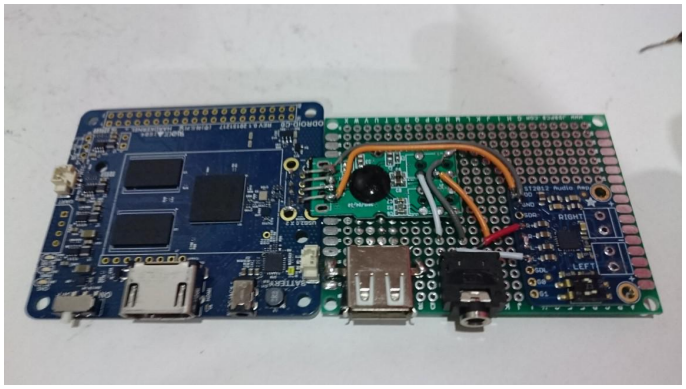
Completando el sistema de audio en la placa de ampliación

Tener una tarjeta de sonido con salida analógica puede dar un buen resultado, pero es mejor una toma de audio 3.5 y un buen amplificador para montar los altavoces. Ese fue precisamente el siguiente paso: cablear y soldar los componentes en la placa de ampliación.



Figuras 14, 15 y 16 – Cableando y soldando los componentes en la placa de ampliación





Cableado del multiplexor analógico

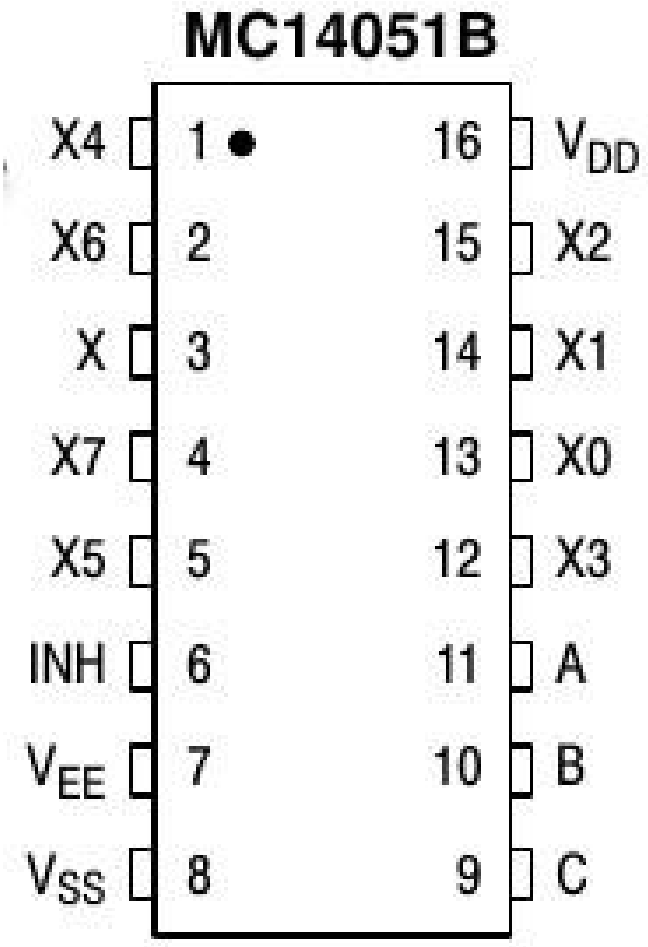
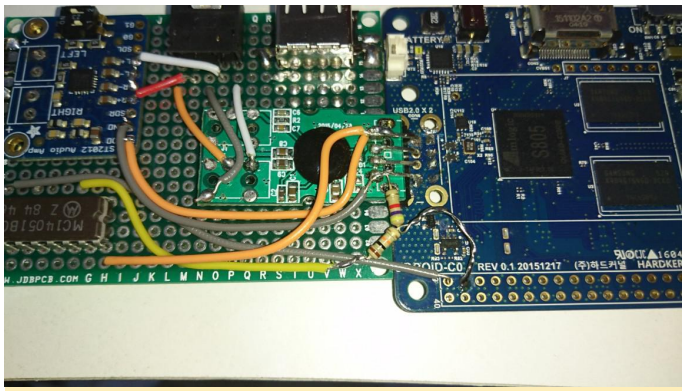
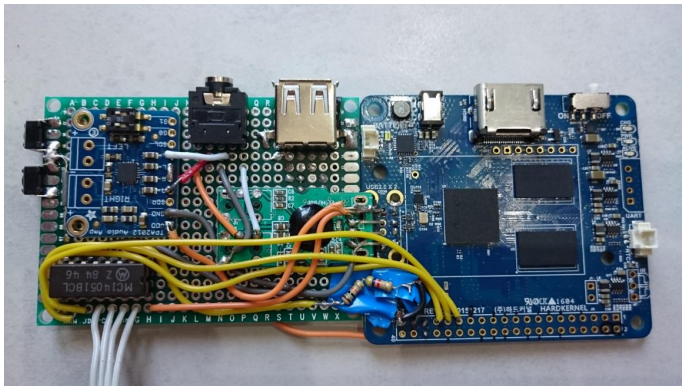


Figura 17 – Diagrama de soldadura para la placa de ampliación

La soldadura de esta pequeña pieza supuso añadir muchos cables y terminó cubriendo casi toda la placa de ampliación. Tuve que usar lo siguiente: Vdd (Vin), Vss (tierra), x (salida analógica), x0, x1, x2, x3 (entradas analógicas), A, B (interruptores digitales). C no era necesario ya que 2 interruptores eran suficientes para alternar entre las primeras 4 salidas. Vee y INH fueron conectados a tierra. Ten presente que hice una separación de voltaje entre x (salida) y la entrada analógica del ODROID. Esto se debe a que los sticks analógicos PSP y MC14051B funcionan en 5V, mientras que la entrada analógica del ODROID-C0 acepta un máximo de 1.8v.



Figuras 18 y 19 – Primer plano del cableado del multiplexor analógico



Botones del volumen

Es posible que hayas visto en la foto anterior que hay 2 botones pulsadores en uno de los bordes de la placa de ampliación. Los conecté a los pins GPIO para controlar el volumen del audio, tal y como se muestra en la Figura 20..

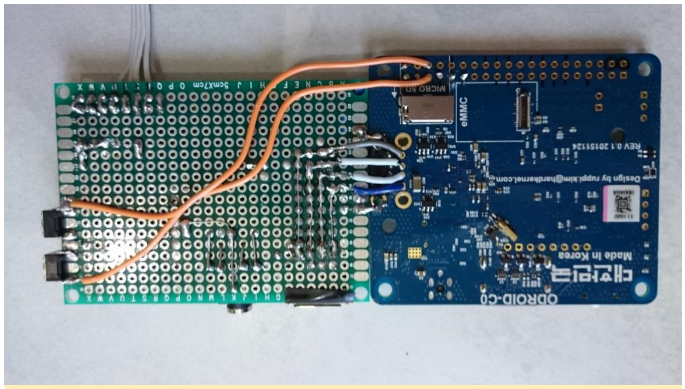


Figura 20 – Cableado de los botones de volumen

Botones de inicio/selección

Utilicé botones pulsadores para los botones de inicio y selección. Los monté en una pequeña placa adicional junto con un led azul para monitorizar la batería.

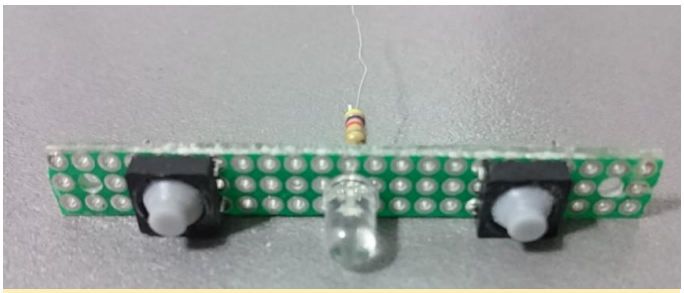


Figura 21: Cableado de los botones de inició y selección

Baterías

Como he indicado anteriormente, he utilizado un par de baterías LiPo cilíndricas con protección. Las conecté en paralelo para conseguir 5000 mA. Tuve que soldar unos cables directamente a las baterías y añadir un conector Molex para poder conectar las dos baterías al conector LiPo del ODROID-C0.

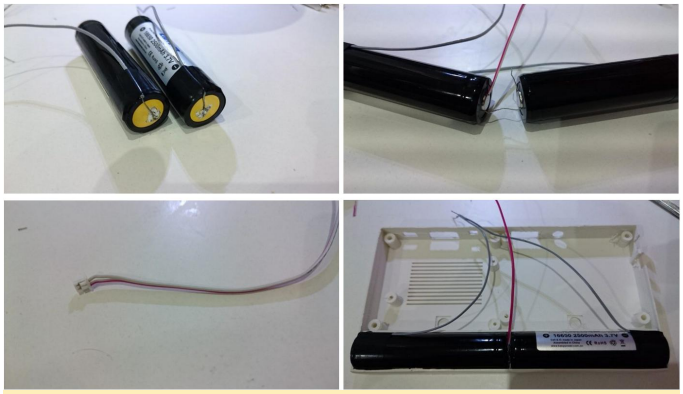
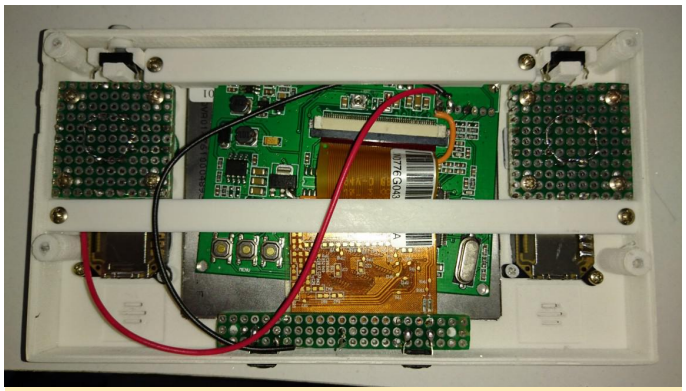
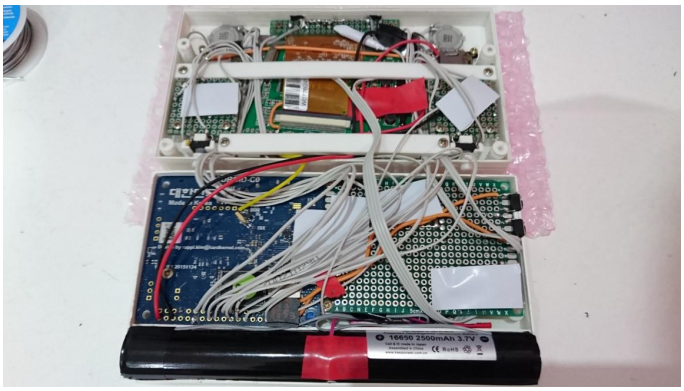


Figura 22 – Detalles del cableado de las baterías

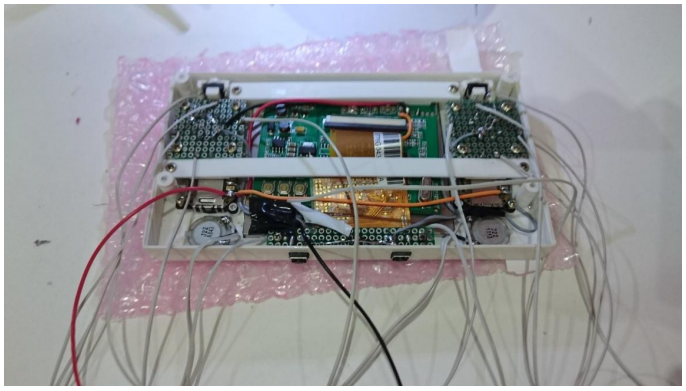
Montaje de los componentes

Llegado a este punto, ya tenía todo lo relacionado con el hardware. Comencé a montar en la parte frontal de la carcasa la pantalla, los sticks analógicos, el D-pad, las placas a-b-x-y y los botones L1 + R1. La pantalla no está pegada, sino que se mantiene con dos barrotes transversales. Como puedes ver en la Figura 23, estas varillas me permitieron colocar y reconducir todos los cables.





Figuras 23 y 24: Pasos del montaje final de los componentes dentro de la carcasa



El siguiente paso a realizar en la parte frontal de la carcasa es añadir los altavoces, los botones de inicio/selección y cablear todo con una puesta a tierra común. Los pasos finales antes del cierre son añadir un disipador de calor, colocar los botones L2 + R2 y la placa base en la parte posterior de la carcasa, luego soldar todo a GPIO. Ten en cuenta también el cable amarillo que es la salida compuesta del ODROID que va a la entrada 1 de la pantalla.

Figuras 25 y 26: Todos los componentes colocados dentro de la carcasa antes de cerrarla

Figuras 27, 28 y 29 – Exterior de la carcasa tras finalizar el montaje

Software

Desarrollé un script que configura el 80% del sistema, incluida una copia de los archivos de configuración específicos. El otro 20% es para la ROMs y los ajustes

personales. Si alguien quiere hacer lo mismo, el script es bastante fácil de adaptar y volver a ejecutar.

Antes de empezar a comentar el script de instalación, aquí te dejo los pasos de instalación que lleve a cabo:

- Implementación de la imagen mínima Debian Jessie de @meveric en el módulo eMMC (**http://bit.ly/2yF2PML**)
- Se crearon dos particiones en la microSD de 128GB: 4 Gb para guardar los estados y el resto para las ROMs, las cuales estarán montadas en /mnt/states y /mnt/ressources). Hice 2 particiones porque tenía la intención de crear un sistema de solo lectura a excepción de los estados, pero finalmente mantuve un sistema completo de lectura/escritura.
- Se creó una carpeta GameODROID en /root y se copia el script de instalación y sus dependencias.

Script de Instalacíon

El script de instalación y todas las dependencias las puedes encontrar en GitHub en **http://bit.ly/2fGJWRU**. Está organizado con funciones dedicadas para cada paso.

La primera función crea puntos de montaje personalizados, copia el fstab personalizado y activa tmpfs:

```
function fstab
{
 echo "fstab and filesystem"

 mkdir -p /mnt/states
 mkdir -p /mnt/ressources

 cp /root/GameODROID/fstab /etc/fstab

 sed -i "s/#RAMLOCK=yes/RAMLOCK=yes/"
/etc/default/tmpfs
 sed -i "s/#RAMSHM=yes/RAMSHM=yes/"
/etc/default/tmpfs
}
```

El archivo fstab personalizado permite cambiar las opciones de montaje para optimizar la velocidad (noatime, discard) y usar una pequeña partición tmpfs para /var/log:

```
tmpfs /var/log tmpfs
nodev,nosuid,noatime,size=20M 0 0
```

```
Tras esta primera función, el sistema se
reinicia, luego se actualiza y se vuelve a
reinicia:

function uptodate
```

```
{
 echo "update"
 apt-get update
 apt-get upgrade
 apt-get dist-upgrade
}
```

El paso final de esta fase es instalar todos los paquetes base necesarios (paquetes de funciones). No tiene nada de especial excepto dos cosas:

- evilwm : Tuve que usar un gestor de ventanas porque algunos juegos nativos no pueden detectar la resolución de pantalla nativa sin él. Descubrí que evilwm era muy buen candidato para la consola, ya que es muy ligero y casi invisible con las configuraciones por defecto.
- Antimicro-ODROID : es un software muy bueno que no conocía. Me permite asignar cualquier evento de teclado y ratón al joypad.
- Paquete de Python evdev: utilizado para configurar la entrada reicast
- Utilicé un archivo de configuración xorg específico para ODROID C1/C0 proporcionado por (**http://bit.ly/2xaSonP**)

Juegos

Esta parte corresponde a las funciones “emulators”, “emulators_glupen64_meveric” y “nativegames”. Excepto para los juegos de Dreamcast para los que utilicé Reicase, el resto de emuladores son parte de Retroarch:

- pcsx-rearmed (PSX)
- fbalpha (CPS2)
- gambatte (Gameboy color)
- gpsp (Gameboy advance)
- mednafen-pce-fast (Pc-Engine + Cdrom)
- nestopia (Nes)
- picodrive (Sega 32X, SegaCD)
- pocketnes (Snes)
- genesis-plus-gx (GameGear, Genesis, MasterSystem)

- mednafen-ngp (Neogeo pocket color)

Para los juegos nativos, seleccioné aquellos que fueran divertidos para jugar con un gamepad y funcionaran correctamente en el ODROID-C0 con una pantalla pequeña:

- hurrican
- hcraft
- frogatto
- SuperMario War
- astromenace
- neverball
- shmupacabra
- aquaria
- Revolt
- Open JK3
- openjazz
- supertuxkart
- mars
- puzzlemoppet
- opentyrian
- pushover

Lanzador de juegos

Esto corresponde a la función "userinterface". Inicialmente, quería usar el modo Attract. Lamentablemente, la implementación de GLES en ODROID-C0/C1 no parece incluir las funciones glBlendEquationSeparateOES () y glBlendFuncSeparateOES (), que son obligatorias para compilar libFSML, que a su vez es obligatorio para compilar el modo Attract. De modo que, utilicé la última versión de Emulation Station con soporte de vista previa de video. Como quería cambiar la pantalla de inicio por defecto por una personalizada, tuve que reemplazar el archivo "splash_svg.cpp" en "EmulationStation/data/converted". Este archivo es una simple matriz C que contiene los bytes de un archivo SVG. Al margen de la configuración clásica de los sistemas, creé una específica que incluye dos scripts para cambiar la pantalla: pantalla interna o HDMI (consulta los scripts composite.sh y hdmi.sh).

Herramientas específicas

Esto corresponde a la función "localtools". Ésta se utilza principalmente para manejar el gamepad GPIO personalizado. Tuve que escribir un pequeño programa en C que crea un gamepad mediante el sondeo GPIO y la entrada de Linux para generar eventos. Utilicé el sondeo en lugar del IRQ porque el SoC no tiene suficiente IRQ para manejar todos los botones. Llamé a esta herramienta gpio_joypad y el código fuente está en GitHub en **http://bit.ly/2xaTdgp**. También maneja el multiplexor analógico para recuperar los valores de izquierda y derecha del sticks analógico.

Archivo de configuración de arranque

Esto corresponde a la función "bootini". Esta función consiste en copiar un archivo boot.ini personalizado en la partición de arranque. Los cambios importantes que hice son:

- Mantener súnicamente dos modos de video: cvbs480 (activado por defecto) y vga (comentado)
- Cec y vpu desactivados
- Argumentos del kernel modificados:
  - "cvbsmode=480cvbs" para lograr una resolución NTSC de 60Hz en lugar de 50Hz PAL
  - "max_freq=1824" para aumentar la frecuencia del reloj del SoC (necesario para los emuladores N64 y Dreamcast)
  - "quiet loglevel=3 rd.systemd.show_status=false udev.log-priority=3" para que el arranque sea lo más silencioso posible

Inicialmente, quería mostrar la pantalla de inicio durante el proceso de arranque. Está bien explicado cómo hacerlo en el wiki de ODROID, pero desafortunadamente sólo funciona para resoluciones 720p.

Lanzar todo en el inicio

Esto corresponde a la función "startup". El inicio automático de X y Emulationstation en el arranque consistia en un servicio tty1 personalizado en systemd que lanza agetty con autologin, un perfil BASH que inicia X cuando la variable tty = tty1 y finalmente un xinitrc que inicia el gestor de ventanas y Emulation Station.

/etc/systemd/system/getty@tty1.service.d/override.conf

```
[Service]

ExecStart=

ExecStart=-/sbin/agetty --autologin root --noclear %I $TERM
The bash /root/.profile :
# ~/.profile: executed by Bourne-compatible login shells.

if [ "$BASH" ]; then
 if [ -f ~/.bashrc ]; then
 . ~/.bashrc
 fi
fi

if [ "$(tty)" = "/dev/tty1" ] ; then
 /usr/local/bin/battery.sh &
 /usr/local/bin/gpio-joypad &
 startx -- -nocursor 2>&1 &
fi

mesg n
```

/root/.xinitrc

```
# a WM is needed some software are correctly sized in full screen
# e.g : emulationstation, rvgl
evilwm & pid=$!

emulationstation.sh &

# this allows not to shutdown X when emulation is killed
# We want that because we have to kill it after gamelaunch
# else it does not reappear on screen (SDL_Createwindow() does never end)
wait $pid
Note that the bash profile start the joypad driver (gpio_joypad) and the battery monitoring script (battery.sh) before starting X.
The battery monitoring script is not very accurate, but I dit not found any way to make a better monitoring to switch on the led on low battery or when charging:
#!/bin/bash

PIN=75
GPIO=/sys/class/gpio
ACCESS=$GPIO/gpio$PIN
LOWBAT=780
CHARGING=1020

if [ ! -d $ACCESS ] ; then
 echo $PIN > $GPIO/export
 echo out > $ACCESS/direction
 echo 0 > $ACCESS/value
fi

while true
do
 ADCVAL=$(cat /sys/class/saradc/saradc_ch0)
# echo "value : $ADCVAL"
 # charging
 if [ $ADCVAL -gt $CHARGING ]; then
 echo 1 > $ACCESS/value
 else
 # low bat
 if [ $ADCVAL -lt $LOWBAT ]; then
 echo 1 > $ACCESS/value
 sleep 1
 echo 0 > $ACCESS/value
 else
 echo 0 > $ACCESS/value
 fi
 fi

 sleep 2
done
```

Finalizar y limpiar

Esto corresponde a la función “optimize_system”. En esta función, el mensaje de inicio de sesión BASH está oculto (para que el proceso de arranque sea lo más silencioso posible) y se limpie la caché de los paquetes (apt-get clean). También se utilizan dos archivos de configuración. El journald.conf personalizado está aquí para escribir registros logs en la ram en lugar de en el disco para mejorar rendimiento:

```
[Journal]
Storage=volatile
I also created a specific alsa configuration file to add latency and buffers, so most sound stutering are avoided for n64 and dreamcast
```

```
games:
pcm.!default {
 type plug
 slave.pcm "softvol"
 ttable.0.1 0.8
 ttable.1.0 0.8
}
pcm.dmixer {
 type dmix
 ipc_key 1024
 slave {
 pcm "hw:1,0"
 period_time 0
 period_size 2048
 buffer_size 65536
 rate 44100
 }
 bindings {
 0 0
 1 1
 }
}
pcm.dsnooper {
 type dsnoop
 ipc_key 1024
 slave {
 pcm "hw:1,0"
 channels 2
 period_time 0
 period_size 2048
 buffer_size 65536
 rate 44100
 }
 bindings {
 0 0
 1 1
 }
}
pcm.softvol {
 type softvol
 slave { pcm "dmixer" }
 control {
 name "Master"
 card 1
 }
}
ctl.!default {
 type hw
 card 1
}
ctl.softvol {
 type hw
 card 1
}
ctl.dmixer {
 type hw
 card 1
}
```

Configuración general de Retroarch

Al margen de cambiar los botones y la ruta, tuve que adaptar algunos parámetros de videos de retroarch (root /.config/retroarch/retroarch.cfg) para optimizar el rendimiento y adaptar mejor al hardware:

```
video_refresh_rate = "59.950001"
video_monitor_index = "0"
video_fullscreen_x = "720"
video_fullscreen_y = "480"
video_vsync = "true"
video_threaded = "true"
video_force_aspect = "true"
```

Configuración específica del núcleo

También hice algunos ajustes en uno de los núcleos de emulador:

Permitiendo 6 botones para SegaCD y 32X:

```
picodrive_input1 = "6 button pad"
Changing glupen64 parameters to optimize
rendering on the ODROID SoC:
glupen64-cpucore = "dynamic_recompiler"
glupen64-rspmode = "HLE"
glupen64-43screensize = "320x240"
glupen64-BilinearMode = "standard"
Allowing PSX analog joypad support:
pcsx_rearmed_pad1type = "analog"
```

Para el emulador Dreamcast, utilicé reicast-joyconfig (**http://bit.ly/2fLE1yH**) para generar la configuración del gamepad y copié el archivo resultante en /root/.config/reicast/joy.conf. También cambié la resolución de toda la pantalla para adaptarla a la pantalla CVBS:

```
[x11]
fullscreen = 1
height = 480
width = 720
```

Asignaciones de teclado y ratón para juegos nativos

Algunos juegos nativos funcionan muy bien, pero requieren de un ratón o un teclado para algunas teclas especiales como Esc, Enter, Space, Shift y las teclas de flechas. Para asignar estas teclas a la consola de gamepad, yo utilicé antimicro. Es un programa sencillo que te permite asignar cualquier tecla del ratón y del teclado a cualquier botón de gamepad.

Buscador de información de videos

Emulation Station tiene un buscador integrado para localizar la información de juegos e imágenes, pero no para videos. Además, si las vistas previas del video son compatibles dependiendo de los temas elegidos, se reproducen a través de VLC, que no permite aceleración en el Soc del ODROID-C0/C1. La consecuencia es que 320×240@30 FPS en h.264 es el tamaño más grande al que se puede jugar. Escribí y usé un script personalizado disponible en GitHub en http://bit.ly/2fGFSkU, que analiza la carpeta game de Emulation Station y rastrea videos desde www.gamesdatabase.org.

Lecciones aprendidas

- No hay forma de controlar correctamente la batería en un ODROID-C0
- Con solo una GPU Mali 450, incluso aumentando la frecuencia del reloj (overclock), aún sigue siendo lento para muchos juegos de N64 y Dreamcast
- Hay algunos fallos que parecen estar relacionados con el driver gráfico, como que Emulation Station no se sale correctamente, y hurrican a veces no se inicia con la resolución correcta
- No es posible utilizar un driver joypad apropiado basado en interrupciones, ya que no hay suficientes IRQ disponibles en el SoC.
- Se hace necesario disponer de un gestor de ventanas, de lo contrario, la pantalla completa no está disponible para los juegos y Emulation Station
- Reicast parece emular ruido de GDRom, realmente lo encuentro algo molesto

Figura 30 – Emulador de Sega ejecutándose en el GamODROID-C0

Puedes ver el GamODROID-C0 en acción en https://youtu.be/3hxYhH7AFYU. Para comentarios, preguntas y sugerencias, visita el post original del blog en http://bit.ly/2khNDTz.

Desarrollo de Android: Proveedor de contenido de Android

October 1, 2017 By Nanik Tolaram Android

Al igual que ocurre con cualquier otro sistema operativo, Android necesita internamente tener un almacenamiento de persistencia para almacenar información del sistema. Estos datos deben estar en un almacenamiento persistente, ya que siempre necesitará hacer referencia a éstos después de cada reinicio para poner el dispositivo en un estado específico. La información del usuario y del dispositivo, como el brillo de la pantalla, el volumen, las cuentas, el calendario, etc., necesitan almacenarse en algún lugar. Android usa lo que se conoce como Proveedor de Contenido. Básicamente, es un mecanismo persistente respaldado por SQLite, o más conocido simplemente como base de datos. La mayoría de los datos se almacenan internamente en varias bases de datos SQLite. En este artículo vamos a echar un vistazo a algunos de los proveedores de contenido que utiliza internamente el sistema operativo.

En este artículo analizaremos algunas de las bases de datos que el sistema operativo utiliza internamente. Un buen punto de partida para obtener información sobre los Proveedores de contenido es dirigirse al sitio web para desarrolladores Android de Google (**http://bit.ly/2hkvljq**).

Qué y donde

Los proveedores de contenido son simplemente aplicaciones corrientes de Android que tienen la función de servir y procesar solicitudes de base de datos desde un cliente. Los datos de los proveedores de contenido internos se almacenan en la carpeta /data/data tal y como se muestra en la Figura 1. Nos interesan las aplicaciones que tienen el siguiente formato de paquetes:

```
com.android.providers.< app_name >
```

La figura 1 muestra los proveedores de contenido internos de Android que hay disponibles.

```
u0_a0     u0_a0     4096 1970-07-21 20:28:00 com.android.providers.blockednumber
u0_a4     u0_a4     4096 1970-07-21 20:29:00 com.android.providers.calendar
u0_a0     u0_a0     4096 1970-07-21 20:29:00 com.android.providers.contacts
u0_a9     u0_a9     4096 2017-08-28 22:20:00 com.android.providers.downloads
u0_a9     u0_a9     4096 1970-07-21 20:29:00 com.android.providers.downloads.ui
u0_a9     u0_a9     4096 1970-07-21 20:29:00 com.android.providers.media
system    system    4096 1970-07-21 20:29:00 com.android.providers.settings
radio     radio     4096 1970-07-21 20:28:00 com.android.providers.telephony
u0_a0     u0_a0     4096 2017-08-28 21:07:00 com.android.providers.userdictionary
```

Figura 1 – Proveedores de contenido de paquetes dentro de la carpeta /data/data

Tenemos, por ejemplo, el servicio DownloadManager proporcionado por Android SDK. Este servicio permite que las apps descarguen archivos de forma asíncrona. Internamente, el entorno de trabajo utiliza este proveedor de contenido para mantener información persistente sobre el estado de un archivo que se va a descargar. El siguiente esquema SQL muestra la declaración que se utiliza internamente para mantener la información del archivo descargado.

```
CREATE TABLE android_metadata (locale TEXT);

CREATE TABLE downloads(_id INTEGER PRIMARY KEY
AUTOINCREMENT,uri TEXT, method INTEGER, entity
TEXT, no_integrity BOOLEAN, hint TEXT,
otaupdate BOOLEAN, _data TEXT, mimetype TEXT,
destination INTEGER, no_system BOOLEAN,
visibility INTEGER, control INTEGER, status
INTEGER, numfailed INTEGER, lastmod BIGINT,
notificationpackage TEXT, notificationclass
TEXT, notificationextras TEXT, cookiedata
TEXT, useragent TEXT, referer TEXT,
total_bytes INTEGER, current_bytes INTEGER,
etag TEXT, uid INTEGER, otheruid INTEGER,
title TEXT, description TEXT, scanned BOOLEAN,
is_public_api INTEGER NOT NULL DEFAULT 0,
allow_roaming INTEGER NOT NULL DEFAULT 0,
allowed_network_types INTEGER NOT NULL DEFAULT
0, is_visible_in_downloads_ui INTEGER NOT NULL
DEFAULT 1, bypass_recommended_size_limit
INTEGER NOT NULL DEFAULT 0, mediaprovider_uri
TEXT, deleted BOOLEAN NOT NULL DEFAULT 0,
errorMsg TEXT, allow_metered INTEGER NOT NULL
DEFAULT 1, allow_write BOOLEAN NOT NULL
DEFAULT 0, flags INTEGER NOT NULL DEFAULT 0);

CREATE TABLE request_headers(id INTEGER
PRIMARY KEY AUTOINCREMENT,download_id INTEGER
NOT NULL,header TEXT NOT NULL,value TEXT NOT
NULL);

CREATE TABLE android_metadata (locale TEXT);

CREATE TABLE downloads(_id INTEGER PRIMARY KEY
AUTOINCREMENT,uri TEXT, method INTEGER, entity
TEXT, no_integrity BOOLEAN, hint TEXT,
otaupdate BOOLEAN, _data TEXT, mimetype TEXT,
destination INTEGER, no_system BOOLEAN,
visibility INTEGER, control INTEGER, status
INTEGER, numfailed INTEGER, lastmod BIGINT,
notificationpackage TEXT, notificationclass
TEXT, notificationextras TEXT, cookiedata
TEXT, useragent TEXT, referer TEXT,
total_bytes INTEGER, current_bytes INTEGER,
etag TEXT, uid INTEGER, otheruid INTEGER,
title TEXT, description TEXT, scanned BOOLEAN,
is_public_api INTEGER NOT NULL DEFAULT 0,
allow_roaming INTEGER NOT NULL DEFAULT 0,
allowed_network_types INTEGER NOT NULL DEFAULT
0, is_visible_in_downloads_ui INTEGER NOT NULL
DEFAULT 1, bypass_recommended_size_limit
INTEGER NOT NULL DEFAULT 0, mediaprovider_uri
TEXT, deleted BOOLEAN NOT NULL DEFAULT 0,
errorMsg TEXT, allow_metered INTEGER NOT NULL
DEFAULT 1, allow_write BOOLEAN NOT NULL
DEFAULT 0, flags INTEGER NOT NULL DEFAULT 0);

CREATE TABLE request_headers(id INTEGER
PRIMARY KEY AUTOINCREMENT,download_id INTEGER
NOT NULL,header TEXT NOT NULL,value TEXT NOT
NULL);
```

El siguiente bloque de código muestra un ejemplo de información almacenada del archivo descargado:

```
1|<a
href="https://www.gstatic.com/android/config_u
pdate/08202014-
metadata.txt">https://www.gstatic.com/android/
config_update/08202014-
metadata.txt</a>|0|||||/data/user/0/com.androi
d.providers.downloads/cache/08202014-
metadata.txt|text/plain|2||2||200|0||com.googl
e.android.configupdater||||||0|||||08202014-
metadata.txt|||1|1|-1|0|0||0||1|0|0

2|http://www.gstatic.com/android/config_update
/07252017-sms-
blacklist.metadata.txt|0|||||/data/user/0/com.
android.providers.downloads/cache/07252017-
sms-
blacklist.metadata.txt|text/plain|2||2||200|0|
|com.google.android.configupdater||||||385|385
```

```
||||07252017-sms-
blacklist.metadata.”txt|||1|1|-1|0|0||0||1|0|0
```

Declaración del proveedor de contenido

Los proveedores de contenido proporcionados por el sistema operativo, que no todos están disponibles para una aplicación de usuario, generalmente tienen la siguiente declaración en su AndroidManifest.xml:

Listado 1: Proveedor de contenido de la configuración

```
< manifest
xmlns:android="http://schemas.android.com/apk/
res/android"
package="com.android.providers.settings"
coreApp="true"
android:sharedUserId="android.uid.system">
    < application
android:allowClearUserData="false"
android:label="@string/app_label"
android:process="system"
android:backupAgent="SettingsBackupAgent"
android:killAfterRestore="false"
android:icon="@mipmap/ic_launcher_settings"
android:defaultToDeviceProtectedStorage="true"
android:directBootAware="true" >
        < provider
android:name="SettingsProvider"
android:authorities="settings"
android:multiprocess="false"
android:exported="true"
android:singleUser="true"
android:initOrder="100" />
    < /application >
< /manifest >
```

El Listado 1 es el AndroidManifest.xml para la aplicación de Configuración que se almacena en el paquete com.android.providers.settings. Se puede ver otro ejemplo en el Listado 2 que muestra la declaración para el Proveedor de Contactos utilizado para la información de contactos almacenados:

Listado 2: Proveedor de contenido de contactos

```
< manifest
xmlns:android="http://schemas.android.com/apk/
res/android"
package="com.android.providers.contacts"
android:sharedUserId="android.uid.shared"
android:sharedUserLabel="@string/sharedUserLab
el" >
    < uses-permission
android:name="android.permission.BIND_DIRECTOR
Y_SEARCH" />
    < uses-permission
android:name="android.permission.GET_ACCOUNTS"
/>
     .....
     .....
     .....
    < permission
android:name="android.permission.SEND_CALL_LOG
_CHANGE" android:label="Broadcast that a
change happened to the call log."
android:protectionLevel="signature|system" />

        < provider
android:name="ContactsProvider2"
android:authorities="contacts;com.android.cont
acts" android:label="@string/provider_label"
…... ……. />
                .....
                .....
                .....
        < /provider >
```

La siguiente tabla muestra algunos de los proveedores de contenido que existen dentro de la versión 7.1.2 de Android:

Descripción	Nombre del Paquete	Localización de la Fuente
CalendarProvider	com.android.providers.calendar	
ContactsProvider	com.android.providers.contacts	
DownloadProvider	com.android.providers.downloads com.android.providers.downloads.ui	
MediaProvider	com.android.providers.media	
SettingsProvider	com.android.providers.settings	frameworks/base/packages/SettingsProvider/src/com/android/providers/settings/SettingsProvider.java
TelephonyProvider	com.android.providers.telephony	packages/providers/TelephonyProvider/src/com/android/providers/telephony/TelephonyProvider.java
UserDictionaryProvider	com.android.providers.userdictionary	
BlockedNumberCall	com.android.providers.blockednumber	
PartnerbookmarksProvider		packages/providers/PartnerBookmarksProvider/src/com/android/providers/p

		artnerbookmarks/PartnerBookmarksProvider.java
EmailProvider		packages/apps/Email/provider_src/com/android/email/provider/EmailProvider.java
LauncherProvider		packages/apps/Launcher3/src/com/android/launcher3/LauncherProvider.java
CellBroadcastReceiver		packages/apps/CellBroadcastReceiver/src/com/android/cellbroadcastreceiver/CellBroadcastContentProvider.java
WearPackageIconProvider.java		packages/apps/PackageInstaller/src/com/android/packageinstaller/wear/WearPackageIconProvider.java
GalleryProvider		packages/apps/Gallery2/src/com/android/gallery3d/provider/GalleryProvider.java
DeskClock		packages/apps/DeskClock/src/com/android/deskclock/provider/ClockProvider.java
SearchRecentSuggestionsProvider		frameworks/base/core/java/android/content/SearchRecentSuggestionsProvider.java
RecentsProvider		frameworks/base/packages/DocumentsUI/src/com/android/documentsui/RecentsProvider.java
MtpDocumentsProvider		frameworks/base/packages/MtpDocumentsProvider/src/com/android/mtp/MtpDocumentsProvider.java
ExternalStorageProvider		frameworks/base/packages/ExternalStorageProvider/src/com/android/externalstorage/ExternalStorageProvider.java
BugreportStorageProvider		frameworks/base/packages/Shell/src/com/android/shell/BugreportStorageProvider.java

Programación en Paralelo ODROID-MC1: Primeros Pasos

October 1, 2017 By Andy Yuen ODROID-MC1

Esta guía no pretende enseñarte a escribir programas en paralelo sobre el ODROID-MC1. Está dirigida a proporcionarte un entorno acondicionado para experimentar con MPJ Express, una implementación de referencia de la API mpiJava 1.2. Se proporciona un programa paralelo en MPJ Express que genera imágenes Mandelbrot para que se ejecute en cualquier máquina o clúster que tenga instalado el SDK de Java: ARM o INTEL. En el caso de que exista mucho interés por conocer la programación MPJ Express, podemos escribir un tutorial para una futura edición de la revista.

¿Por qué la programación en paralelo?

La programación o la computación en paralelo es un modo de procesamiento en el cual muchos cálculos independientes se llevan a cabo de forma simultánea, operando bajo la premisa de que los grandes problemas a menudo se pueden dividir en pequeños, y que luego se resuelven al mismo tiempo. En resumen, lo que persigue es tipo de programación es:

- Aumentar la velocidad general,
- Procesar gran cantidad de datos,
- Resolver problemas en tiempo real y
- Solucionar problemas a su debido tiempo

¿Por qué ahora?

Mucha gente debate si la **Ley de Moore** todavía se mantiene. La ley de Moore viene a decir que el número de transistores en un denso circuito integrado se duplica aproximadamente cada dos años (algunos dicen que 18 meses). La ley de Moore debe su nombre a Gordon E. Moore, cofundador de INTEL y Fairchild Semiconductor. Es este continuo avance de la tecnología de circuitos integrados el que nos ha llevado a pasar del PC original de 4,77 megahertz a los actuales procesadores multi-gigahertz. La arquitectura de los procesadores

también ha cambiado mucho con los múltiples hilos de ejecución, la ejecución fuera de orden, el almacenamiento en caché, etc. Suponiendo que la ley de Moore se siga aplicando, nos enfrentamos a grandes problemas para mejorar el rendimiento de la CPU:

> **El Muro de la Potencia**
> Potencia = C * Vdd2 * Frecuencia

No podemos seguir escalando la cantidad de transistores y la frecuencia sin reducir el Vdd (voltaje de alimentación). El escalado del voltaje ya se ha estancado.

> **El Muro de la Complejidad**
> Depurar y verificar los grandes núcleos OOO (Out-Of-Order) es costoso (100 ingenieros durante 3-5 años). Las cachés son más fáciles de diseñar, pero solo pueden ayudar hasta cierto punto.

Como ejemplo del problema de la potencia (frecuencia), tenemos que:

- E5640 Xeon (4 núcleos @ 2.66 GHz) tiene una sobre carga de potencia de 95 vatios.
- L5630 Xeon (4 núcleos @ 2.13 GHz) tiene una sobre carga de potencia de 40 vatios.

Esto implica un aumento del 137% de potencia eléctrica para un aumento del 24% de la potencia de la CPU. A este ritmo, no se va a poder escalar. Entra en escena el diseño de los multinúcleos. Un procesador multinúcleo ejecuta el multiprocesamiento en un único paquete físico. En lugar de aumentar la frecuencia para lograr un mayor rendimiento, se colocan más núcleos en un procesador para que los programas se puedan ejecutar en paralelo y así aumentar el rendimiento. En la actualidad, todos los procesadores INTEL son multinúcleo. Incluso los procesadores utilizados en los teléfonos móviles son todos procesadores con múltiples núcleos.

Limitaciones en las mejoras de rendimiento

¿Cuánta mejora puedo esperar de mi aplicación si logro que se ejecute en un procesador multinúcleo? La respuesta es que depende. Es posible que tu aplicación no tenga ninguna mejora de rendimiento si no se ha diseñado específicamente para aprovechar el potencial de múltiples núcleos. Incluso si lo hace, todavía depende de la propia naturaleza del programa y del algoritmo que utilice. **La ley de Amdahl** establece que si P es la proporción de un programa que se puede hacer en paralelo, y (1-P) es la proporción de que no se puede poner en paralelo, entonces la aceleración máxima que se puede alcanzar utilizando N procesadores es:

> 1/[(1-P) + (P/N)]

La aceleración en relación al número de núcleos o procesadores con valores específicos de P se muestra en el siguiente gráfico.

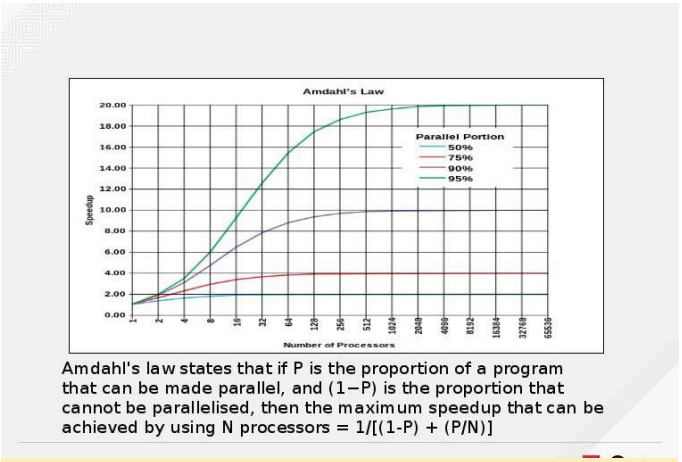


Figura 1 – Ley de Amdahl

Esto te puede dar una cierta idea de cuánto rendimiento puedes llegar a alcanzar a la hora de escribir tu programa pudiendo aprovechar la programación en paralelo en lugar de tener expectativas pocos realistas

W¿Por qué hacer programación paralela en Java?

Algunas de las ventajas de escribir programas java en paralelo son:

- Se escribe una vez, se ejecuta en cualquier lugar,
- Colectivo muy extenso de desarrolladores de software,
- Programación orientada a objetos (OO),
- Verificación del código en tiempo de ejecución y tiempo de compilación,
- Recogida automática de basura (información no válida),

- Soporte para múltiples hilos de ejecución (multi-threading) en el lenguaje y
- Amplia colección de librerías

Java soporta múltiples hilos de ejecución desde sus inicios, entonces, ¿qué hay de nuevo? El multi-hilo Java utiliza el modelo de memoria compartida, lo que significa que no se puede escalar para usar varias máquinas.

Un modelo de memoria distribuida hace referencia a un sistema informático multiprocesador, como un ODROID-MC1, en el que cada procesador tiene su propia memoria privada. Las tareas informáticas solo pueden incidir en datos locales, y si se requieren datos remotos, las tareas informáticas debe comunicarse con uno o más procesadores remotos. En cambio, un multiprocesador de memoria compartida ofrece un único espacio de memoria que es usado por todos los procesadores. Los procesadores no tienen que estar pendientes de dónde residen los datos, excepto que pueden tener penalizaciones de rendimiento, y que deben evitar condiciones de competición (Procesador más rápido).

La Librería MPJ Express Paso de Mensajes

MPJ Express es una implementación de referencia de la API mpiJava 1.2, que es el equivalente de Java de la **especificación MPI 1.1**. Permite a los desarrolladores de aplicaciones escribir y ejecutar aplicaciones en paralelo para procesadores multinucleo y clústeres que utilicen una configuración multinúcleo (modelo de memoria compartida) o una configuración de clúster (modelo de memoria distribuida) respectivamente. Este último también es compatible con una técnica híbrida para ejecutar programas en paralelo en un clúster de máquinas multinúcleo, como el ODROID-MC1. Todas las dependencias de software ya han sido instaladas en la imagen para tarjetas SD que proporciono **Mi proyecto mpj-example en Github** también ha sido clonado y compilado. Se ha copiado un archivo jar resultante y un archivo dependiente en el directorio ~/mpj_rundir donde puede probar en modo clúster o multicore. Toda la documentación sobre de MPJ Express la puedes encontrar en el directorio $MPJ_HOME/doc

Generación de Fractal usando MPJ Express

El proyecto mpj_example es un generador Mandelbrot. Las imágenes de conjunto Mandelbrot se obtienen muestreando números complejos y determinando para cada número si el resultado tiende hacia el infinito cuando se ejecuta la reiteración de una operación matemática particular. Las partes imaginarias y reales de cada número se convierten en coordenadas de imagen para un píxel de color según la rapidez con la que la secuencia diverge, o en nada en absoluto. Mi programa paralelo MPJ Express asigna cada núcleo disponible para que calcule un trozo vertical de la imagen del conjunto Mandelbrot a la vez. En consecuencia, cuantos más núcleos haya disponibles, más trabajo se puede realizar en paralelo. Las imágenes de Mandelbrot en coordenadas específicas se muestran en las siguientes imágenes.

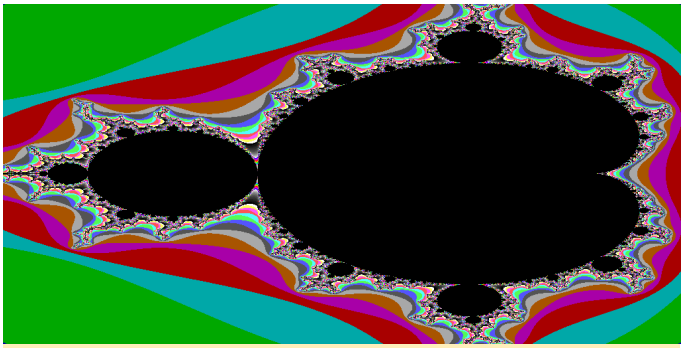


Figura 2.1 – mandelbrot1: (-0.5, 0.0)

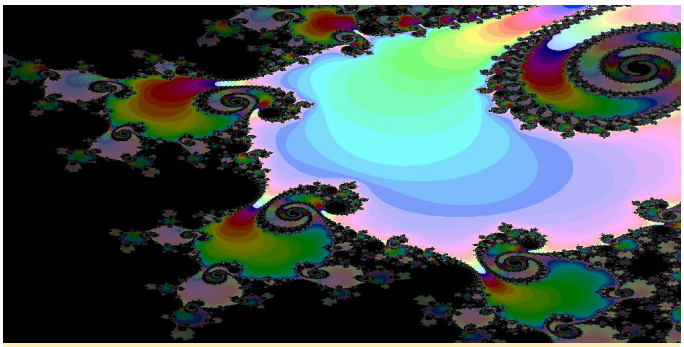


Figura 2.2 – mandelbrot2: (-0.7615134027775, 0.0794865972225)

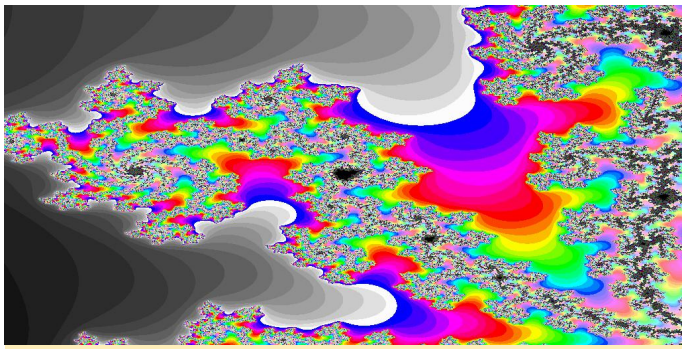


Figura 2.3 – mandelbrot3: (0.1015, -.633)

Estas imágenes de Mandelbrot son generadas usando los siguientes comandos en una única máquina, el nodo maestro, utilizando una configuración multinúcleo. Desde el prompt del nodo maestro, introduce los siguientes comandos:

```
cd ~/mpj_rundir

mpjrun.sh -np 1 -cp ./mpj-example-1.0.jar com.kardinia.mpj.ColourMandelbrot -.5 0 2
colourMap.txt mandelbrot1.png

mpjrun.sh -np 1 -cp ./mpj-example-1.0.jar com.kardinia.mpj.ColourMandelbrot
-0.7615134027775 0.0794865972225 0.0032285925920 colourMap.txt mandelbrot2.png

mpjrun.sh -np 1 -cp ./mpj-example-1.0.jar com.kardinia.mpj.ColourMandelbrot 0.1015 -.633
0.01 colourMap.txt mandelbrot3.png
```

Figura 3: Comandos mandelbrot

Puedes volver a ejecutar el comando anterior con valores -np entre 1 y 8 inclusive para ver la diferencia de rendimiento al variar la cantidad de núcleos utilizados para la generación de Mandelbrot. Recuerda que el XU4 tiene 4 núcleos pequeños A7 y 4 grandes núcleos A15.

Los parámetros que aparecen después de com.kardinia.mpj.ColourMandelbrot son:

- parámetro 1: inicio de coordenada x
- parámetro 2: inicio de coordenada y
- parámetro 3: tamaño del paso
- parámetro 4: mapa de color para asignar el número de iteraciones a un color en particular
- parámetro 5: nombre de archivo para guardar el mandelbrot generado

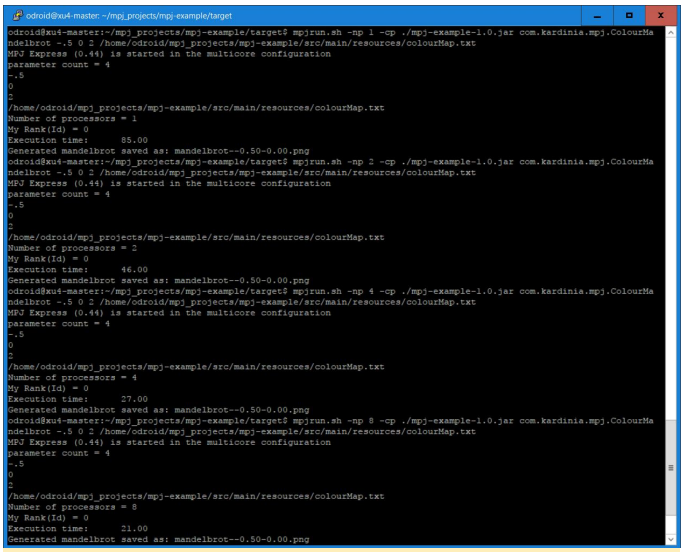


Figura 4 – Esta es una captura de pantalla de la ejecución en modo multinúcleo.

Para ejecutar el generador de Mandelbrot en modo clúster, sigue las siguientes instrucciones: Se necesita un archivo de texto llamado “máquines” que contenga los nombres de host de cada nodo de tu clúster ODROID-MC1 en líneas independientes. El archivo que se encuentra en ~/mpj_rundir contiene las siguientes líneas:

```
xu4-master
xu4-node1
xu4-node2
xu4-node3
```

Para iniciar el demonio MPJ en cada nodo, ejecuta el siguiente comando una vez desde el nodo maestro para iniciar un demonio MPJ en cada nodo:

```
$ mpjboot machines
```

Luego envía los siguientes comandos desde el nodo maestro:

```
mpjrun.sh -np 4 -dev hybdev -cp ./mpj-example-1.0.jar
com.kardinia.mpj.ColourMandelbrot -.5 0 2 colourMap.txt mandelbrot1.png

mpjrun.sh -np 4 -dev hybdev -cp ./mpj-example-1.0.jar
com.kardinia.mpj.ColourMandelbrot -0.7615134027775 0.0794865972225
0.0032285925920 colourMap.txt mandelbrot2.png

mpjrun.sh -np 4 -dev hybdev -cp ./mpj-example-1.0.jar
com.kardinia.mpj.ColourMandelbrot 0.1015 -.633 0.01 colourMap.txt
mandelbrot3.png
```

Figura 5: Comandos del nodo maestro

Una vez más, puedes variar el número que aparece después de -np entre 4 y 32, ya que hay un total de 32 núcleos en tu clúster ODROID-MC1. La siguiente

captura de pantalla muestra la ejecución de los comandos anteriores en el modo clúster.

Figura 6 – clusterRun.png

Cuando hayas terminado de experimentar con el modo de clúster, introduce el siguiente comando en el sistema maestro para finalizar todos los demonios MPJ iniciados anteriormente:

```
$ cd ~/mpj_rundir
$ mpjhalt machines
```

Rendimiento en el ODROID-MC1

El rendimiento de la ejecución del generador de Mandelbrot en ODROID-MC1 tanto en modo multinúcleo como en modo clúster se resume en las siguientes gráficas. Para hacer una comparación, también lo ejecuté en una máquina virtual con 4 núcleos asignados a ésta en una vieja máquina INTEL I7 quad core. Aquí tienes una captura de pantalla del funcionamiento del generador en la máquina virtual.

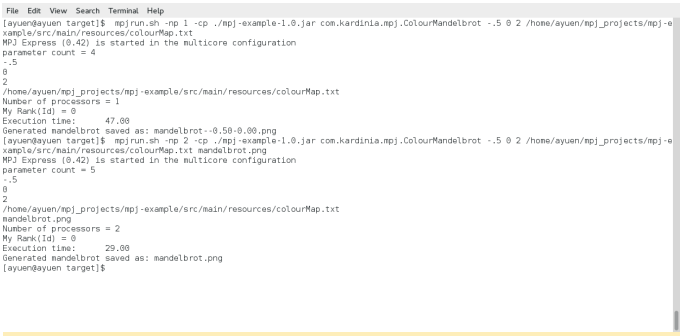


Figura 7 – mpjIntelMulticore.png

El rendimiento de la ejecución en INTEL también se muestra en la misma gráfica. El eje vertical es el tiempo en segundos que se necesita para generar el Mandelbrot en la coordenada -0.5, 0.0. El eje horizontal es la cantidad de núcleos utilizados.

Figura 8 – executionTime.png

Representar gráficamente los datos de un modo diferente nos proporciona el coeficiente de incremento del rendimiento a medida que aumenta el número de núcleos.

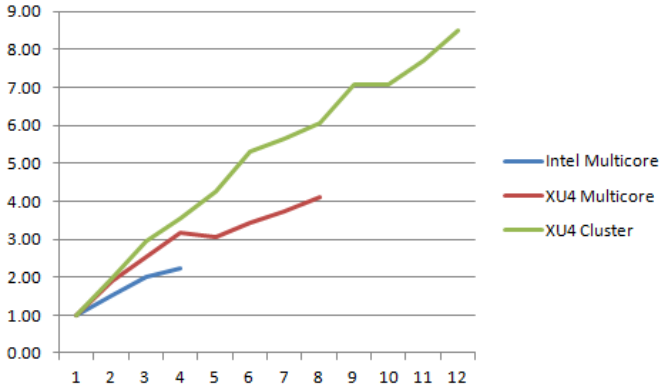


Figura 9 – performanceIncrease.png

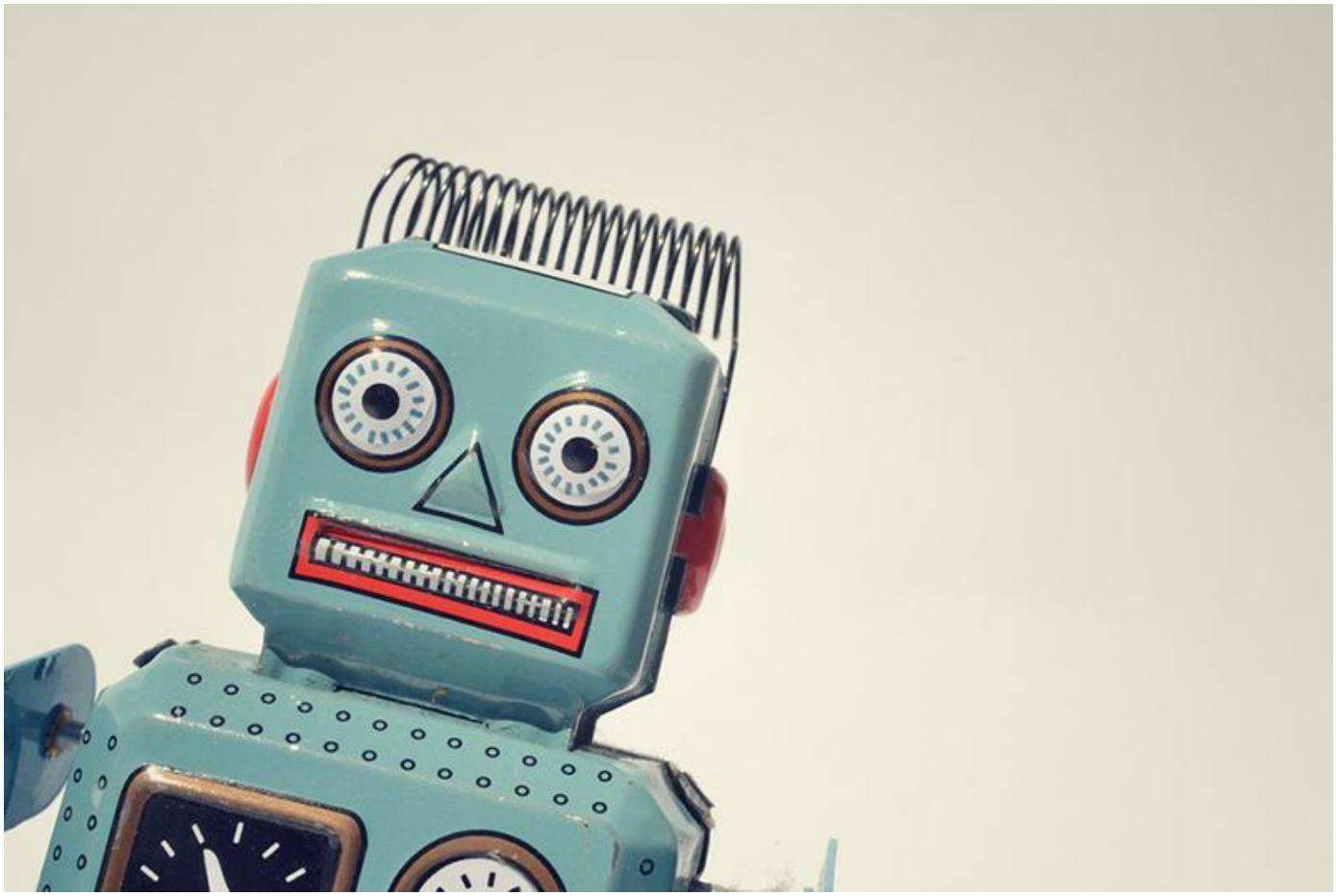
Observé que cuando un nodo estaba usando los 4 núcleos grandes o los 8 núcleos, la corriente utilizada estaba entre 2.0 y 2.5 amperios. Mi barata fuente de alimentación no era capaz de suministrar suficiente corriente cuando los 4 XU4s en el ODROID-MC1 ejecutaban todos los núcleos al 100%. Esta es la razón por la que solo medí el rendimiento de la configuración del clúster hasta 12 núcleos. Otra cuestión interesante que observé fue que en el modo multinúcleo en un solo XU4, el mayor incremento de rendimiento se produjo cuando se utilizaron los 4 núcleos grandes. Añadir los pequeños núcleos apenas mejoró el rendimiento. Incluso en el modo clúster, el aumento de rendimiento se redujo a medida que la cantidad de núcleos aumentaba debido a la ley de Amdahl, ya que el sistema maestro tenía que dedicar la misma cantidad de tiempo combinando las imágenes parciales generadas en una imagen completa y disponía de un tiempo limitado para transferir las imágenes parciales a través de la red.

Conclusión

Espero que mis dos guías de inicio en ODROID Magazine te hayan dado algunas ideas sobre cómo utilizar tu ODROID-MC1 como un clúster swarm Docker y también como un clúster de computación para la programación en paralelo. Lo que puedes hacer con esto sólo está limitado por tu imaginación. Avísanos si está realmente estás interesado en información sobre cómo usar MPJ Express. Podemos crear tutoriales adicionales. Mientras tanto, disfruta y sigue explorando las posibilidades de tu ODROID-MC1.

Home Assistant: Scripts para Personalizar el Sistema

October 1, 2017 By Adrian Popa Tutoriales



En este artículo, profundizaremos en la personalización de Home Assistant, creando nuestros propios scripts para recopilar datos de sensores remotos y otros dispositivos de control. También veremos varias formas de comunicarnos con los sensores remotos.

Obtener datos de temperatura en remoto

Supongamos que tiene este problema: tienes varios sensores de temperatura, como el DS1820 en tu casa, conectados a varios ODROID y deseas enviar los datos a Home Assistant, que se ejecuta en un único dispositivo. Necesitas decidirte por un protocolo de transporte de datos y escribir un script para recopilar las lecturas de temperatura y pasarlas a Home Assistant.

Vamos a analizar algunas técnicas:

- Sondeo por HTTP
- Conexión a través de la API de Home Assistant
- Conexión a través de MQTT

Sondeo por HTTP

Si estás acostumbrado a desarrollar web, probablemente habrás usado la denominada CGI (Interfaz Común de Comunicaciones), la forma más antigua de generar contenido dinámico utilizando un servidor web (**http://bit.ly/2jNVkjT**). Básicamente, carga un script en el servidor, independientemente del lenguaje, que es requerido por el servidor web y que a su vez sirve el resultado del script al cliente. Obviamente, primero necesitas instalar un servidor HTTP en tu host remoto y activar el soporte CGI. Usaremos Apache 2.4:

```
$ sudo apt-get install apache2
$ sudo a2enmod cgi
```

La configuración por defecto asigna /cgi-bin/URL a /usr/lib/cgi-bin en tu sistema de archivos. Cualquier script ejecutable que coloques en esta ubicación

puede ser requerido por el servidor web. Vamos a suponer que puedes obtener los datos de temperatura del host remoto con estos comandos shell:

```
$ cat /sys/devices/w1_bus_master1/28-05168661eaff/w1_slave
c6 01 4b 46 7f ff 0c 10 bd : crc=bd YES
c6 01 4b 46 7f ff 0c 10 bd t=28375
```

En el resultado anterior, la primera línea valida la lectura del valor (si coincide con el CRC) y la segunda línea devuelve el valor en mili-celsius. Vamos a crear dos scripts (no olvide marcarlos como ejecutables) para ilustrar el código en dos lenguajes diferentes: BASH y Python. Los archivos se almacenarán en /usr/lib/cgi-bin/temperature.sh y /usr/lib/cgi-bin/temperature.py.

```
#!/bin/bash

filename='/sys/devices/w1_bus_master1/28-05168661eaff/w1_slave'
valid=0

echo "Content-Type: text/plain"
echo

# read line by line, parse each line
while read -r line
do
    if [[ $line =~ crc=.*YES ]]; then
        # the CRC is valid. Continue processing
        valid=1
        continue
    fi
    if [[ "$valid" == "1" ]] && [[ $line =~ t=[0-9]+ ]]; then
        # extract the temperature value
        rawtemperature=`echo "$line" | cut -d "=" -f 2`
        # convert to degrees celsius and keep 1 digit of accuracy
        echo "scale=1;$rawtemperature/1000" | bc
    fi
#read line by line from $filename
done < "$filename"
```

```
#!/bin/bash


filename='/sys/devices/w1_bus_master1/28-05168661eaff/w1_slave'
valid=0

echo "Content-Type: text/plain"
echo

# read line by line, parse each line
while read -r line
do

  if [[ $line =~ crc=.*YES ]]; then
    # the CRC is valid. Continue processing
    valid=1
    continue
  fi
  if [[ "$valid" == "1" ]] && [[ $line =~ t=[0-9]+ ]]; then
    # extract the temperature value
    rawtemperature=`echo "$line" | cut -d "=" -f 2`
    # convert to degrees celsius and keep 1 digit of accuracy
    echo "scale=1;$rawtemperature/1000" | bc
  fi
#read line by line from $filename
done < "$filename"
```

Figura 1a Hay dos formas de leer la misma temperatura, esta en bash

```
#!/usr/bin/python
import re

filename = '/sys/devices/w1_bus_master1/28-05168661eaff/w1_slave'
valid = False

print "Content-Type: text/plain"
print ""

# execute the command and parse each line of output
with open(filename) as f:
    for line in f:

        if re.search('crc=.*YES', line):
            # the CRC is valid. Continue processing
            valid = True
            continue

        if valid and re.search('t=[0-9]+', line):
            # extract the temperature value
            temperature = re.search('t=([0-9]+)', line)
            # convert to degrees celsius and keep 1 digit of accuracy
            output = "%.1f" % (float(temperature.group(1))/1000.0)
            print output
```

```python
#!/usr/bin/python
import re

filename = '/sys/devices/w1_bus_master1/28-05168661eaff/w1_slave'
valid = False

print "Content-Type: text/plain"
print ""

# execute the command and parse each line of output
with open(filename) as f:
    for line in f:

        if re.search('crc=.*YES', line):
            # the CRC is valid. Continue processing
            valid = True
            continue

        if valid and re.search('t=[0-9]+', line):
            # extract the temperature value
            temperature = re.search('t=([0-9]+)', line)
            # convert to degrees celsius and keep 1 digit of accuracy
            output = "%.1f" % (float(temperature.group(1))/1000.0)
            print output

```

Figura 1b – Y esta en Python

Analicemos un poco los scripts. **Ambos scripts** empiezan con una línea shebang que le indica al llamador qué intérprete usar para ejecutar el script (línea 1). A continuación, definimos dos variables que apuntan al archivo que va a leer (línea 4) y una variable para recordar si la lectura es válida o no (línea 5). En las líneas 7 y 8, grabamos los encabezados HTTP. El script CGI tiene que devolver los encabezados HTTP en las primeras líneas, separados por una línea en blanco del resto del resultado. El servidor web necesita al menos el encabezado Content-Type para procesar la solicitud. Si omites esto, recibirás un error de HTTP 500. En la línea 11 empezamos a leer las líneas del archivo para analizarlas. Buscamos un CRC válido con una expresión regular en la línea 14 y si es correcto, fijamos valid = true. En la línea 19, si el CRC es true y la línea contiene una temperatura, extraemos la temperatura en bruto (línea 21) y la convertimos a Celsius, con un digito de precisión (línea 23), y la mostramos como resultado estándar. Para acceder a los datos, puede usar cualquier cliente HTTP, como wget, tal y como se muestra en la Figura 2.

```
adrianp@spica:~$ bash /usr/lib/cgi-bin/temperature.sh
Content-Type: text/plain

28.6
adrianp@spica:~$ python /usr/lib/cgi-bin/temperature.py
Content-Type: text/plain

28.7
adrianp@spica:~$ wget -q -O - http://127.0.0.1/cgi-bin/temperature.sh
28.6
adrianp@spica:~$ wget -q -O - http://127.0.0.1/cgi-bin/temperature.py
28.7
adrianp@spica:~$
```

Figura 2 – Extrayendo los datos del host remoto

Puede haber ligeras diferencias en el resultado que se devuelva debido a los diferentes métodos de redondeo utilizados, o por variaciones en el periodo de tiempo en el que se realiza la consulta, lo cual puede provocar que los datos del sensor fluctúen un poco.

Por razones de seguridad, puedes activar la autenticación básica HTTP en la configuración de tu servidor. Necesitarás SSL/HTTPS con certificados válidos para protegerte de cualquiera que quiera analizar tu tráfico, aunque esta cuestión está fuera del alcance de este artículo. Puedes leer más sobre este tema **aquí** y **aquí**.

Para añadir el sensor a Home Assistant podemos usar el sensor REST dentro de **configuration.yaml** :

```
sensor:
  ...
  - platform: rest
    resource: http://192.168.1.13/cgi-bin/temperature.sh
    name: Temperature REST Bash
    unit_of_measurement: C
  - platform: rest
    resource: http://192.168.1.13/cgi-bin/temperature.py
    name: Temperature REST Python
   unit_of_measurement: C
```

Puedes conseguir el código **aquí**.

Pros de este método:

- Es fácil de implementar si estás familiarizado con el desarrollo web.
- En Home Assistant se reinician los nuevos datos que se sondean

Contras de este método:

- El uso de un servidor web lo expone a posibles vulnerabilidades.
- El servidor web puede usar muchos recursos en comparación con lo que realmente necesitas hacer.

Conexión a través de la API de HA

Una técnica diferente en la que no interviene un servidor web es mover los datos del sensor a Home Assistant desde el sistema remoto. Podemos usar una **Plantilla Sensor** para almacenar y presentar los datos. Para hacer esto, puedes hacer que el script de

la Figura 3 sea requerido periódicamente por cron en el sistema remoto

```
#!/bin/bash

filename='/sys/devices/w1_bus_master1/28-05168661eaff/w1_slave'
homeassistantip='192.168.1.9'
haport=8123
api_password='odroid'
sensor_name='sensor.temperature_via_api'
valid=0

# read line by line, parse each line
while read -r line
do

    if [[ $line =~ crc=.*YES ]]; then
        # the CRC is valid. Continue processing
        valid=1
        continue
    fi
    if [[ "$valid" == "1" ]] && [[ $line =~ t=[0-9]+ ]]; then
        # extract the temperature value
        rawtemperature=`echo "$line" | cut -d "=" -f 2`
        # convert to degrees celsius and keep 1 digit of accuracy
        temperature=`echo "scale=1;$rawtemperature/1000" | bc`
        # push the data to the Home Assistant entity via the API
        curl -X POST -H "x-ha-access: $api_password" -H "Content-Type: application/json"
        --data "{"state": "$temperature"}" http://$homeassistantip:$haport/api/states/$sensor_name
    fi
#read line by line from $filename
done < "$filename"
```

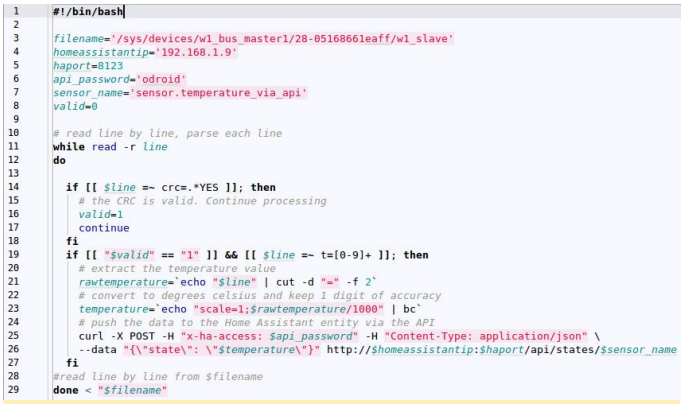


Figura 3: Enviando datos a través de la API de HA

Como puede ver, el código es similar al ejemplo anterior, excepto que en la línea 25 utiliza la **API REST de Home Assistant** para enviar la lectura de la temperatura. La API REST requiere que envíes la clave de la API de Home Assistant dentro de un encabezado HTTP, y los datos que quieras cambiar deben estar en una carga JSON en la solicitud POST. La URL que envías es tu instancia de Home Assistant /api/states/sensor.name. Para activar esto y enviar los datos cada 5 minutos, añade la siguiente entrada cron:

```
$ crontab -e
*/5 * * * * /bin/bash /path/to/script/temperature-HA-API.sh > /dev/null 2>&1
```

La configuración de Home Assistant debería parecerse a esto:

```
sensor:
…
- platform: template
   sensors:
     temperature_via_api:
       value_template: '{{ states.sensor.temperature_via_api.state }}'
       friendly_name: Temperature via API
       unit_of_measurement: C
```

La plantilla sensor generalmente se utiliza para extraer los datos de otras entidades de Home Assistant y, en este caso, la utilizamos para extraer datos de sí mismo. Este truco te evita que se eliminen datos de estado tras una actualización externa. Antes de configurar la temperatura, el estado del sensor estará en blanco. Una vez que cron ejecute el script

por primera vez, obtendrás datos de temperatura. Puedes descargar el código desde **here**

Pros de este método:

- Permite controlar cuando se envían los datos
- El uso de recursos es muy bajo.

Contras de este método:

- Tu script necesita tener claramente tu contraseña secreta de Home Assistant
- Cuando se reinicia Home Assistant, el sensor no tendrá ningún valor hasta la primera actualización

Conexión a través de MQTT

El protocolo MQTT es un protocolo de máquina a máquina diseñado para la eficiencia (y para entornos de baja potencia) y ya ha sido tratado en **anteriores artículos de ODROID Magazine**. El modo de funcionamiento parte de un servidor central llamado broker que transmite mensajes a clientes que se suscriben a un tema común. Piensa en un tema como si fuera algo así como un canal IRC donde los clientes se conectan y se envían mensajes específicos.

Home Assistant tiene un **MQTT Broker**, integrado, pero en mis pruebas lo encontré poco fiable, así que usé un broker dedicado llamado Mosquitto. Se puede instalar en el mismo sistema que Home Assistant o en un sistema diferente. Para instalarlo, sigue estos pasos:

```
$ sudo apt-get install mosquitto mosquitto-clients
$ sudo systemctl enable mosquitto
```

La versión 3.11 de MQTT soporta autenticación, de modo que, debes configurar un nombre de usuario y una contraseña que serán compartidos por el bróker y los clientes y opcionalmente, **Encriptación SSL**. En mi configuración, utilicé la autenticación usuario-contraseña y añadí el usuario 'ODROID'

```
$ sudo mosquitto_passwd -c /etc/mosquitto/passwd ODROID
$ sudo vi /etc/mosquitto/conf.d/default.conf
allow_anonymous false
password_file /etc/mosquitto/passwd
```

Puedes activar el soporte genérico para MQTT en Home Assistant añadiendo una plataforma MQTT en configuration.yaml (recuerda que el parámetro mqtt_password se define en secrets.yaml):

```
mqtt:
  broker: 127.0.0.1
  port: 1883
  client_id: home-assistant
  keepalive: 60
  username: ODROID
  password: !secret mqtt_password
```

Para enviar datos de temperatura a Home Assistant, nuestro script necesitará la librería Python Paho-MQTT. Para analizar los datos de configuración necesitaremos también la librería python-yaml:

```
$ sudo apt-get install python-pip python-yaml
$ sudo pip install paho-mqtt
```

El script se ejecuta como un demonio, realizando lecturas periódicas de temperatura en segundo plano y enviando los cambios a través de MQTT. El código que lee la temperatura en sí (línea 40) es el mismo que aparecen en la Figura 1b y no se muestra en la Figura 4 para simplificar. El único cambio es que, en lugar de mostrar la temperatura, la devuelve como una cadena.

El código comienza importando algunos módulos auxiliares, definiendo las funciones para analizar la configuración YAML en un diccionario. La lectura de la temperatura y la ejecución empiezan en la línea 57. Se define y se activa un nuevo objeto cliente MQTT con los detalles necesarios para acceder al bróker MQTT. En la línea 61, hay un subproceso en segundo plano iniciado por la llamada loop_start () que asegura que el cliente permanezca conectado al bróker MQTT. Sin él, la conexión expiraría tras un tiempo y tendrías que volver a conectarte manualmente. Tienes más información sobre la API MQTT en Python disponible **aquí**. En la línea 65, aparece un bucle que lee los datos de temperatura, los compara con la última lectura de temperatura y, si hay un cambio, publica un mensaje MQTT al bróker con la nueva temperatura. Después, el código está inactivo durante un tiempo antes de la próxima lectura. Cuando se publican datos en el bróker (en la

línea 71), debes especificar el tema MQTT, el valor que se envía y también si estos datos deben ser persistentes o no. Los datos persistentes son muy ventajosos, ya que puedes obtener la última lectura de temperatura desde MQTT cuando inicias Home Assistant y leer la temperatura desde el principio. Puedes conseguir el código completo **aquí**.

```
#!/usr/bin/python
import paho.mqtt.client as mqtt
import re
import time
import sys
import yaml

# Prerequisites:
# * pip: sudo apt-get install python-pip
# * paho-mqtt: pip install paho-mqtt
# * python-yaml: sudo apt-get install python-yaml

# Configuration file goes in /etc/temperature-mqtt-agent.yaml and should contain your mqtt broker details

# For startup copy temperature-mqtt-agent.service to /etc/systemd/system/
# Startup is done via systemd with
#  sudo systemctl enable temperature-mqtt-agent
#  sudo systemctl start temperature-mqtt-agent

filename = '/sys/devices/w1_bus_master1/28-05168661eaff/w1_slave'
valid = False
oldValue = 0

""" Parse and load the configuration file to get MQTT credentials """

conf = {}

def parseConfig():
    global conf
    with open("/etc/temperature-mqtt-agent.yaml", 'r') as stream:
        try:
            conf = yaml.load(stream)
        except yaml.YAMLError as exc:
            print(exc)
            print("Unable to parse configuration file /etc/temperature-mqtt-agent.yaml")
            sys.exit(1)

""" Read temperature from sysfs and return it as a string """

    def readTemperature():
        with open(filename) as f:
            for line in f:
                if re.search('crc=.*YES', line):
                    # the CRC is valid. Continue processing
                    valid = True
                    continue
                if valid and re.search('t=[0-9]+', line):
                    # extract the temperature value
                    temperature = re.search('t=([0-9]+)', line)
                    # convert to degrees celsius and keep 1 digit of accuracy
                    output = "%.1f" % (float(temperature.group(1)) / 1000.0)
                    # print("Temperature is "+str(output))
                    return output

""" Initialize the MQTT object and connect to the server """
parseConfig()
client = mqtt.Client()
if conf['mqttUser'] and conf['mqttPass']:

client.username_pw_set(username=conf['mqttUser'], password=conf['mqttPass'])
    client.connect(conf['mqttServer'], conf['mqttPort'], 60)
    client.loop_start()

""" Do an infinite loop reading temperatures and sending them via MQTT """

    while (True):
        newValue = readTemperature()
        # publish the output value via MQTT if the value has changed
        if oldValue != newValue:
            print("Temperature changed from %f to %f" % (float(oldValue), float(newValue)))
```

```
            sys.stdout.flush()
            client.publish(conf['mqttTopic'],
newValue, 0, conf['mqttPersistent'])
            oldValue = newValue
        # sleep for a while
        # print("Sleeping...")
        time.sleep(conf['sleep'])
```

Figura 4: Enviando datos de temperatura a través de MQTT

El script también necesitará un archivo de configuración donde guardar las credenciales MQTT, ubicado en /etc/temperature-mqtt-agent.yaml:

```
mqttServer: 192.168.1.9
mqttPort: 1883
mqttUser: ODROID
mqttPass: ODROID
mqttTopic: ha/kids_room/temperature
mqttPersistent: True
sleep: 10
```

También hay un script de arranque systemd para que inicie tu script con cada arranque. Cópialo en /etc/systemd/system:

```
$ cat /etc/systemd/system/temperature-mqtt-
agent.service
[Unit]
Description=Temperature MQTT Agent
After=network.target
[Service]
ExecStart=/usr/local/bin/temperature-mqtt-
agent.py
Type=simple
Restart=always
RestartSec=5
[Install]
WantedBy=multi-user.target
```

Para que se active al arrancar, ejecuta los siguientes comandos:

```
$ sudo systemctl enable temperature-mqtt-
agent.service
$ sudo systemctl start temperature-mqtt-
agent.service
```

En lo que respecta a Home Assistant, necesitamos definir un **sendor MQTT** con la siguiente configuración:

```
sensor:
...
 - platform: mqtt
    state_topic: 'ha/kids_room/temperature'
    name: 'Temperature via MQTT'
    unit_of_measurement: C
```

Pros de este método:

- El uso de recursos es bajo
- API estándar de bajo coste operativo diseñada para la comunicación máquina a máquina.

Contras de este método:

- El sistema remoto necesita tener claramente la contraseña de MQTT
- Cuando se reinicia Home Assistant, el sensor no tendrá ningún valor hasta la primera actualización a menos que se use la opción de persistencia de MQTT

Ahora que has visto varios ejemplos de cómo obtener datos en Home Assistant, deberá elegir cual es el mejor para tu configuración. De ahora en adelante yo usare MQTT porque, aunque parezca más difícil al principio, permite un mejor escalado con tareas más complejas.

Controlar una Smart TV con un componente personalizado

Aquí tenemos un nuevo problema que queremos resolver. Queremos obtener el número de canal actual, el nombre del programa y el estado de TV de un televisor Samsung con **firmware SamyGO**. El televisor revela esta información a través de una API REST que se puede instalar en el televisor desde **aquí**. La API devuelve información en formato JSON sobre el estado actual del televisor. Se puede inyectar códigos de control remoto y también se puede enviar de vuelta capturas de pantalla con lo que aparece actualmente en pantalla. La llamada y los resultados de la información actual se ven así:

```
$ wget -O - "http://tv-ip:1080/cgi-bin/samygo-web-api.cgi?challenge=oyd4uIz5WWAkWPo5MzfxBFraI05C3FDorSPE7xiMLCVAQ40a&action=CHANNELINFO"
```

{“source”:”TV (0)”, “pvr_status”:”NONE”, “powerstate”:”Normal”, “tv_mode”:”Cable (1)”, “volume”:”9”, “channel_number”:”45”, “channel_name”:”Nat Geo HD”, “program_name”:”Disaster planet”, “resolution”:”1920×1080”, “error”:false}

En teoría, podríamos configurar los sensores REST para hacer la consulta anterior y utilizar la plantilla para conservar solo la información deseada, algo así como esto:

```
sensor:
...
  - platform: rest
  resource: http://tv-ip:1080/cgi-bin/samygo-web-api.cgi?challenge=oyd4uIz5WWAkWPo5MzfxBFraI05C3FDorSPE7xiMLCVAQ40a&action=CHANNELINFO
  method: GET
  value_template: '{{ value_json.channel_name }}'
  name: TV Channel Name
```

Pero el problema es que, para obtener toda la información de diferentes sensores, debes hacer la misma consulta, descartar una gran cantidad de datos y mantener solo lo que necesitas para ese sensor en particular. Esto es ineficaz y en este caso, no funcionará porque, para obtener y presentar esta información, la API web que se ejecuta en el televisor inyecta varias librerías en los procesos que se ejecutan en el televisor para captar algunas llamadas de funciones y obtener los datos **aquí**. La fase de la inyección es muy crítica, y hacer varias inyecciones al mismo tiempo podría causar que se bloquee el proceso, lo cual bloquearía a su vez el TV. Esta es la razón por la cual la API web realiza las consultas de forma progresiva y no responde a una consulta antes de que se complete la anterior, aunque esto genere tiempos de espera.

Lo que se necesita en este caso es que el componente del sensor almacene todos los datos JSON y tenga una plantilla de sensores para extraer los datos necesarios y presentarlo. Para hacer esto, necesitamos un componente personalizado, que deriva del sensor REST y que actúa justamente como el sensor REST, pero cuando recibe datos JSON, los almacena como atributos de la entidad en lugar de descartarlos.

Los componentes personalizados se encuentran en el directorio ~ homeassistant/.homeassistant/custom_components y mantienen la estructura de los componentes habituales (lo que significa que nuestro sensor estará en el subdirectorio del sensor). Se cargarán en el Inicio de Home Assistant antes de que se analice la configuración. La Figura 5 muestra las diferencias entre el sensor REST y el nuevo sensor JsonRest personalizado.

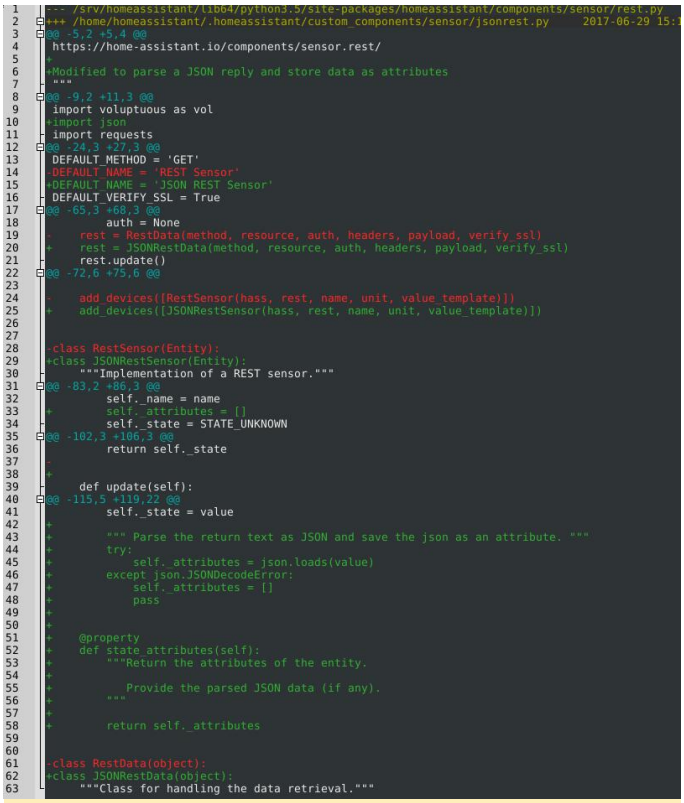


Figura 5: cambios para almacenar y presentar los atributos

Para entender los cambios realizados, deberías seguir la guía de componentes personalizada http://bit.ly/2fvc1PT. El código hace algunos cambios en el nombre de las clases del módulo para evitar enfrentamientos con el componente REST, y activa y gestiona una lista de atributos que se analizan desde la entrada JSON. Estos se mostrarán como atributos en la vista de estados. El nombre del nuevo componente es JsonRest, igual que el nombre del archivo.

Para instalar el componente JsonRest, puede seguir estos pasos:

```
mkdir -p
~homeassistant/.homeassistant/custom_component
s/sensor/
wget -O
~homeassistant/.homeassistant/custom_component
s/sensor/jsonrest.py
https://raw.githubusercontent.com/mad-
ady/home-assistant-
customizations/master/custom_components/sensor
/jsonrest.py
```

TPara configurar el nuevo componente, una vez que se almacene en el directorio custom_components/sensor, podemos usar esta configuración para sondear el televisor cada 5 minutos:

```
sensor:
…
 - platform: jsonrest
    resource: http://tv-ip:1080/cgi-
bin/samygo-web-api.cgi?
challenge=oyd4uIz5WWAkWPo5MzfxBFraI05C3FDorSPE
7xiMLCVAQ40a&action=CHANNELINFO
    method: GET
    name: TV Living ChannelInfo
    scan_interval: '00:05'
 - platform: template
    sensors:
     tv_living_powerstate:
        value_template: '{{
states.sensor.tv_living_channelinfo.attributes
.power_state }}'
        friendly_name: TV Living Power
     tv_living_channel_number:
        value_template: '{{
states.sensor.tv_living_channelinfo.attributes
.channel_number }}'
        friendly_name: TV Living Channel
Number
      tv_living_channel_name:
        value_template: '{{
states.sensor.tv_living_channelinfo.attributes
.channel_name }}'
        friendly_name: TV Living Channel Name
      tv_living_program_name:
        value_template: '{{
states.sensor.tv_living_channelinfo.attributes
.program_name }}'
        friendly_name: TV Living Program Name
```

Ahora solo el componente JsonRest será el que sondee el televisor para obtener la información, y la plantilla sensores extraerán los datos necesarios de los atributos, reduciendo la carga en el televisor.

Puesto que la API web del TV permite tomar capturas de pantalla, vamos añadir esta función también a Home Assistant (para no perder de vista lo que los niños ven el TV). La API devuelve una imagen JPEG cada vez que preguntes con el parámetro URL action=SNAPSHOT. Puedes usar un a **Componente de Cámara IP genérica**:

```
camera 2:
  platform: generic
  name: TV Living Image
  still_image_url: http://tv-ip:1080/cgi-
bin/samygo-web-api.cgi?
challenge=oyd4uIz5WWAkWPo5MzfxBFraI05C3FDorSPE
7xiMLCVAQ40a&action=SNAPSHOT
```

La API web TV también te permite enviar acciones de control remoto, que se pueden configurar a través del **Componente de commandos Restful**:

```
rest_command:
  tv_living_power_on:
    url: !secret samygo_tv_living_power_on
  tv_living_power_off:
    url: !secret samygo_tv_living_power_off
```

Tras agrupar un poco, el resultado final lo puedes ver **aquí**. Tienes un enlace a la configuración disponible **aquí**, y un ejemplo para el archivo secrets **aquí**. Puedes encontrar el código y la configuración en la **pçagina de GitHub.**

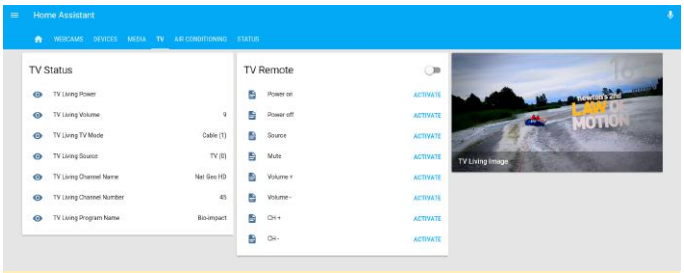
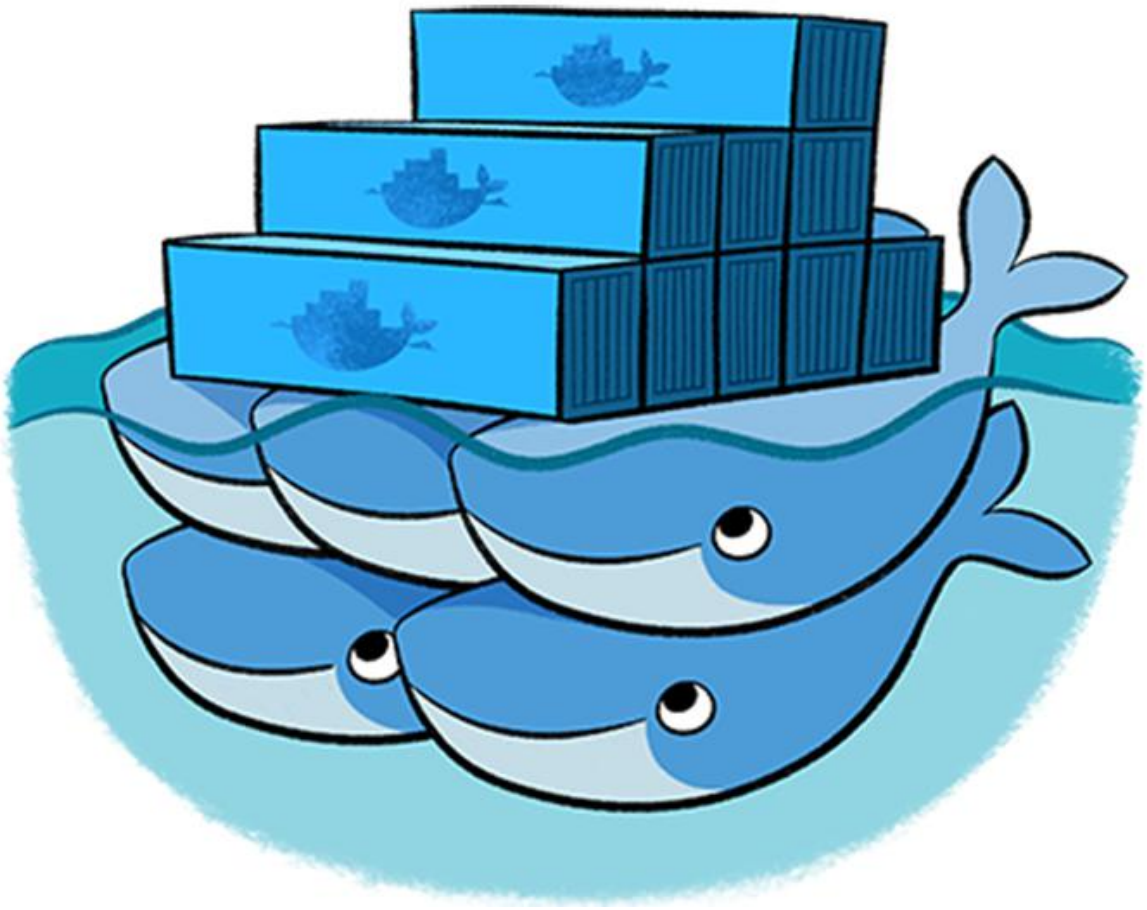


Figura 6 – Echando un ojo a la TV

ODROID-MC1 Docker Swarm: Guía de Inicio

October 1, 2017 By Andy Yuen Docker, ODROID-MC1, Tutoriales



El personal de Hardkernel ha montado una gran instalación informática en forma de clúster para probar la estabilidad de Kernel 4.9. El cluster consistía en 200 ODROID-XU4 (es decir, un total neto de 1600 núcleos de CPU y 400 GB de RAM), tal y como se muestra en la Figura 1.

Figura 1 – Clúster xu4

La experiencia resultante con este ejercicio les llevó a la idea de desarrollar un potente clúster personal a un precio razonable, de donde nació el ODROID-MC1. ODROID-MC1 significa My Cluster One. Consta de 4 unidades apilables, cada una con un ordenador de plaza reducida (SBC) específicamente diseñado y basado en el procesador Octa-core Samsung Exynos 5422. Es compatible con la serie SBC ODROID-XU4 y está montado en una carcasa de aluminio. Estas carcasas (que también incorporan un disipador de calor integrado) están apiladas con un ventilador fijado en la parte trasera que garantizase sufienciente refrigeración.

La placa de circuitos ODROID-MC1 es una versión recortada de la utilizada en el almacenamiento conectado en red (NAS) ODROID-HC1 (Home Cloud One), con el adaptador SATA suprimido. La placa de circuito ODROID-HC1, a su vez, es un ODROID-XU4 rediseñado con el conector HDMI, el conector eMMC, el hub USB 3.0, el botón de encendido y el interruptor deslizante eliminados.

Las principales características del ODROID-MC1 son:

- CPUs Samsung Exynos 5422 Cortex-A15 2Ghz y Cortex-A7 Octa
- 2Gbyte LPDDR3 RAM PoP apilada
- Puerto Ethernet Gigabit
- Puerto USB 2.0 Host
- Ranura para tarjeta microSD UHS-1 para el soporte de arranque
- Imágenes SO de servidor Linux basadas en el moderno Kernel 4.9 LTS

El ODROID-MC1 viene montado y listo para usarse como un clúster personal para el aprendizaje, así como para hacer trabajos útiles. En la Parte 1 de esta serie del ODROID-MC1, describiré cómo usarlo como un clúster Docker Swarm. En la Parte 2, describiré cómo desarrollar programas en paralelo para que se ejecuten en el clúster ODROID-MC1

Para configurar el clúster MC1, necesitas lo siguiente, además del propio hardware MC1:

- 1 switch Gigabit con al menos 5 puertos
- 5 cables Ethernet
- 4 tarjetas SD (de al menos 8 GB de capacidad)
- 4 adaptadores de corriente para los ordenadores MC1

Configuración del sistema operativo en cada ordenador del clúster

La parte más tediosa de la configuración del clúster ODROID-MC1 es instalar el Sistema Operativo (SO) y los paquetes de software necesarios para ejecutar y administrar el Swarm Docker en cada nodo informático Para facilitar el proceso, puede descargar una imagen para tarjetaSD con casi todo listo en https://oph.mdrjr.net/MrDreamBot/. Digo “casi”, porque todavía tienes que realizar unas cuantas cosas para que todo funcione correctamente”. La tarjeta SD tiene los logins ‘root’ y ‘odroid’ ya configurados. La contraseña para ambos es “odroid”:

El swarm que estamos montando consta de 1 nodo maestro y 3 nodos de trabajo. A modo de ejemplo, supongamos que utilizan los siguientes nombres de host y direcciones IP. Por supuesto, puedes cambiarlos para que se adaptaen a tu entorno. Todos los nodos en el swam deben tener una dirección IP estática como la siguiente:

```
xu4-master – 192.168.1.80
xu4-node1 – 192.168.1.81
xu4-node1 – 192.168.1.82
xu4-node1 – 192.168.1.83
```

Para empezar el proceso de configuración, necesitas conectar tu PC y un nodo ODROID-MC1 a un switch Gigabit y que éste tenga conexión a tu router de casa (para acceder a Internet). La imagen está configurada para usar la dirección IP asignada dinámicamente usando DHCP de tu router. Debes concertarte usando SSH para configurar cada nodo y usar una dirección IP estática en su lugar. Hay otros parámetros de configuración que también necesitas cambiar. Se supone que tiene algunos conocimientos de línea de comandos de Linux para realizar los siguientes pasos: Escribe la imagen del sistema operativo en tu tarjeta SD: copia la imagen de la tarjeta SD en las 4 tarjetas SD clase 10 de 8GB. Si usas tarjetas SD de mayor capacidad, debe cambiar el tamaño del sistema de archivos en cada tarjeta SD para usar todo el espacio de la tarjeta SD. La forma más fácil de hacerlo es montar la tarjeta SD en una máquina Linux y usar gparted (www.gparted.com) para cambiar el tamaño. Este es el método que utilice para mis tarjetas SD. Inserta una tarjeta SD en uno de los ordenadores MC1. Inicia una sesión SSH desde tu PC al nodo ODROID-MC1 como root. Su dirección IP la puedes encontrar en tu router. Omite el siguiente paso si estás configurando el nodo maestro. Cambia el nombre de host editando el archivo /etc/hostname, modifica xu4-master por xu4-nodeX donde X es 1, 2 o 3 dependiendo del nodo de trabajo que estés configurando. Configura una dirección IP estática editando /etc/network/interfaces, eliminando el "#" delante de la sección resaltada y reemplazando la dirección IP 192.168.1.80 con la dirección IP (en la subred de tu red doméstica) que deseas que se le asigne al nodo que estás configurando. Actualiza el archivo /etc/hosts de modo que cada entrada de nodo ODROID-MC1 tenga el nombre y la dirección IP correctos. Haz algunas pruebas con los cambios: reinicia el nodo para ver si puedes conectarte por SSH usando la nueva dirección IP que le asignaste. Si es así, has configurado con éxito ese nodo. De lo contrario, verifica los cambios que he descrito anteriormente y asegúrate de que no haya errores tipográficos. Configure el siguiente nodo de trabajo: repite estos los pasos hasta que termines de configurar todos los nodos.

Figura 3 – interfaces.png

Para los usuarios experimentados de Linux, hay una forma alternativa de hacer todo anterior, montar cada tarjeta SD en tu sistema Linux y editar los correspondientes archivos directamente en la tarjeta SD. Tras configurar tu clúster, conéctate por ssh al xu4-master como usuario "odroid", contraseña "odroid". Desde el maestro, entrar por SSH a todos los nodos de trabajo sin usar la contraseña, ya que los nodos del clúster se han configurado con una autenticación basada en claves. Haz lo mismo con "root" utilizando el comando "sudo -s" o utilizando SSH para establecer una conexión como usuario root en el nodo xu4-master, luego usa SSH para conectarse a todos los nodos de trabajo.

Configuración del modo Swarm de Docker

Un nodo es un host Docker que participa en un Swarm. Un nodo gestor (maestro) es aquel donde envías una definición de servicio y éste programa el servicio para que se ejecute como tareas en los nodos de trabajo. Los nodos de trabajo reciben y ejecutan tareas programadas por un nodo gestor. Un nodo gestor, por defecto, también es un nodo de trabajo a menos que esté expresamente configurado para no ejecutar tareas. Es posible configurar múltiples nodos maestros y de trabajo en un swarm para proporcionar Alta Disponibilidad (HA).

Para iniciar el modo swarm, introduce el siguiente comando en el nodo gestor:

```
$ docker swarm init --advertise-addr
192.168.1.80
```

que devolverá:

```
swarm initialized: current node
(8jw6y313hmt3vfa1fme1dinro) is now a manager
```

Ejecuta el siguiente comando para añadir un nodo trabajador a este swarm en cada nodo:

```
$ docker swarm join --token SWMTKN-1-
1q385ckmw7owbj2zfno04dmidb62iqg2devd7yvae5wvuo
hc11-at5g1ad4f24fck4cutsqhnw06
192.168.1.80:2377
```

Para que los otros nodos se unan al clúster, envía el anterior comando "docker swarm join" en cada nodo. Esto se puede hacer usando parallel-ssh usando el comando una vez desde el gestor y ejecutandolo en cada nodo. La Figura 4 muestra una captura de pantalla del funcionamiento del comando "docker ps" usando parallel-ssh.

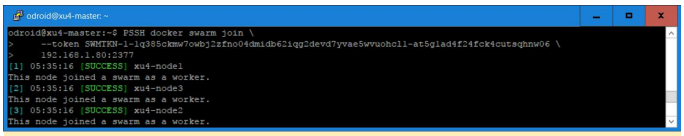


Figura 4 – pssh.png

Ahora tenemos un Swarm Docker en funcionamiento.

Probando el Swarm

Para ayudar a visualizar lo que está sucediendo en el swarm, podemos usar la imagen **Docker Swarm Visualizer** image (visualizer-arm). Para implementarla como un servicio, envía el siguiente comando desde el prompt del gestor:

```
$ docker service create --name=dsv --
publish=8080:8080/tcp --
constraint=node.role==manager --
mount=type=bind,src=/var/run/docker.sock,dst=/
var/run/docker.sock alexellis2/visualizer-arm
```

Ten en cuenta que ODROID-XU4 está basado en ARMv7, es decir, es un sistema de 32 bits, a diferencia del ODROID-C2 que está basado en ARMv8 de 64 bits. Por consiguiente, las imágenes Docker utilizadas en los siguientes comandos son diferentes de las utilizadas en mis ejemplos de Docker con el ODROID-C2.

Apunta tu navegador al gestor visitando http://192.168.1.80:8080, o puedes apuntar tu navegador a cualquiera de los nodos del swarm. Observa los cambios reportados por el visualizador al utilizar el servicio httpd usando mi imagen mdreambot httpd busybox de 32 bits en http://dockr.ly/2wWPCNP. Mi imagen se inicia con el comando:

```
$ docker service create --replicas 3 --name
httpd -p 80:80 mdreambot/arm32-busybox-httpd
```

En la Figura 5 aparece un Docker Swarm Visualizer que muestra los nodos en los que se ejecutan las réplicas del servicio, poniendo de manifiesto el modelo de servicio declarativo utilizado por el modo swarm.

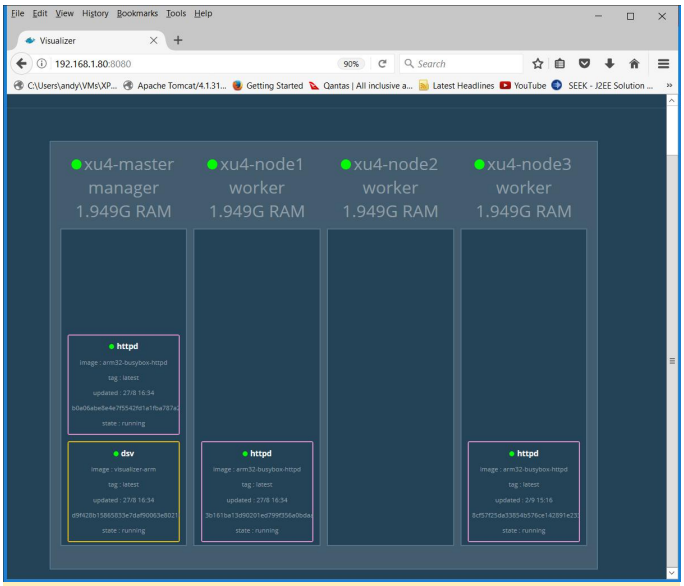


Figura 5: Docker Swarm Visualizer muestra los nodos en los que se ejecutan las réplicas del servicio

Utiliza el siguiente comando curl para probar la función de balanceo de carga de swarm docker:

```
$ curl http://192.168.1.80/cgi-bin/lbtest
```

La Figura 6 es una captura de pantalla del resultado de los comandos curl, que confirma que cada solicitud ha sido redirigida a un nodo diferente.

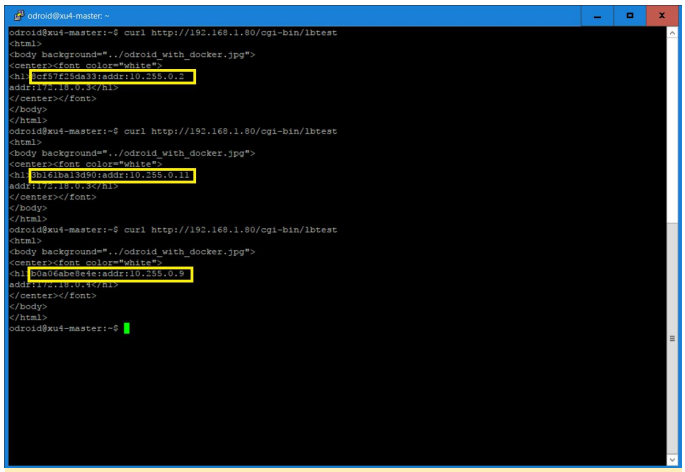


Figura 6: Docker realiza un balanceo de carga automático

Para hacer una prueba de auto-regeneración, detuve el contenedor httpd que se ejecutaba en el xu4-master, otro contenedor httpd fue ejecutado en otro nodo para reemplazar el que acababa de detener, como se puede ver en la siguiente captura de pantalla. Esto es debido a que cuando iniciamos el servicio, especificamos “replica = 3”, de modo que swarm Docker mantendrá el número deseado de réplicas. Esto se conoce como reconciliación de estado deseado.

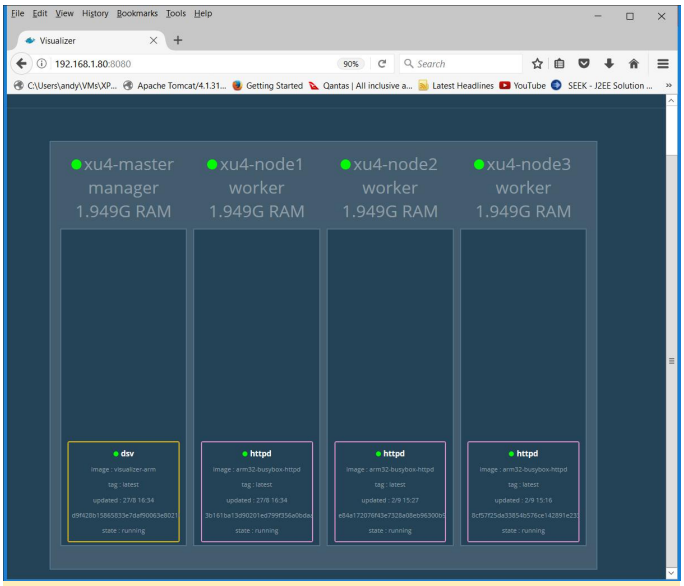


Figura 7: Docker soporta la reconciliación del estado de auto-regeneración

Conclusión

TEl modo swarm de Docker es ahora completamente funcional en tu clúster ODROID-MC1. Tomate la libertad de experimentar con él. Espero que esta guía logre su objetivo consiguiendo que ejecutes swarm de docker en el ODROID-MC1. Para obtener más información sobre el modo swarm de Docker, consulta mis otros artículos de ODROID Magazine sobre el tema.

Conociendo un ODROIDian: Brian Kim, Ingeniero de Hardkernel

October 1, 2017 By Brian Kim Conociendo un ODROIDian

FFigura 1: Brian Kim en California, Estados Unidos.

Figura 2 – Familia de Brian Kim: Hermana menor, madre, padre, hermana mayor y sobrinos

Por favor, háblanos un poco sobre ti. Tengo 36 años y vivo en Seúl, Corea del Sur. Soy el ingeniero de investigaciones de Hardkernel Co., Ltd. Mi trabajo principal para Hardkernel es el de mantener el software de código abierto de u-boot, Kernel de Linux, WiringPi2-Python y Buildroot. Modifico el software de código abierto y realizo algunas tareas rutinas de soporte para ODROID. Hardkernel brinda soporte técnico en los foros de ODROID en http://forum.odroid.com, y mi jefe me asigna los problemas de software reportados en los foros. Aunque a veces es estresante especialmente cuando se trata de un problema complejo o no es un problema de software, es divertido tener discusiones técnicas con los usuarios de ODROID. No solo tengo un grado en Información y Comunicación de la Universidad de Youngsan (Corea del Sur), sino también una licenciatura en Ingeniería de Comunicaciones Móviles de la Universidad Sungkyunkwan (Corea del Sur). No era un buen estudiante cuando estaba en el instituto, pero sí que estudié mucho en la universidad y logré obtener una nota media de 4.5/4.5 en un semestre. Aprendí conocimientos generales de Ciencias de la Computación en la universidad y la escuela de posgrado. Estudié CMT (Concurrence Multipath Transfer) usando SCTP en la licenciatura y escribí un artículo al respecto. Comencé mi carrera profesional como desarrollador de software. Mi primer proyecto serio fue el software del sistema de información médica que usaba Delphi en 2005. Nuestro equipo diseñó una base de datos médica asociada a una base de datos de información personal, escribimos el código fuente Object Pascal para el software. Disfruté desarrollando software usando varios lenguajes de

programación y librerías como Visual Basic, MFC, Win32, Qt, PHP, ASP, JSP, C, C ++ y Java.

Figura 3 – Uno de los proyectos con el que se entretenía Brian, Solo como depurador

Incluso después de graduarme en la universidad, seguía teniendo ganas de aprender. De modo que, me mudé a Seúl para recibir un curso profesional de sistemas embebidos en una academia privada. Aprendí el sistema embebido, la arquitectura ARM, el kernel de Linux, la programación de redes y los RTOS en este curso, desde 2005 hasta 2006. Aprendí mucho sobre la filosofía Unix durante este tiempo. Tras finalizar el curso, desarrollé un software POS (Point Of Sale) y decodificador IP como trabajo a tiempo parcial en 2007. Me uní a WIZnet en 2008, siendo éste mi primer trabajo a tiempo completo.

Figura 4: primer producto comercial de Brian, W5300E01-ARM

WIZnet es una fabulosa compañía que diseña chips de red embebidos con pila TCP/IP cableada. Mi primer trabajo en WIZnet fue desarrollar un controlador de red Linux para el conjunto de chips WIZnet. Trabajé mucho y terminé el proyecto en tres meses. Después de esto, desarrollé la placa embebida ARM que incluía el chip WIZnet llamada W5300E01-ARM, que fue mi primer producto comercial. El controlador de red modificado que desarrollé está incluido en el código fuente del kernel Linux estandar. Además de eso, desarrollé un módulo de puerta de enlace de serie a Ethernet y brindaba soporte técnico. Participaba en un grupo de trabajo de análisis de software de código abierto todos los sábados en 2007 (Kernel Linux) y 2011 (Xen Hypervisor). Nuestro grupo de trabajo analizó el código fuente en detalle hasta conocerlo completamente, lo cual nos apasionaba. Tras finalizar el estudio en aproximadamente un año, escribimos artículos y libros sobre lo que aprendimos. El código fuente del software de código abierto es mi libro de texto, y los desarrolladores de software de código abierto son mis maestros incluso hoy en día.

Figura 5: el primer ordenador de Brian, un IBM XT

¿Cómo empezaste con los ordenadores? Cuando tenía 8 años, empecé con los ordenadores con un IBM XT. Aunque era el ordenador de mi primo, solía usarlo para jugar a juegos de DOS. Recuerdo los viejos dichos de la gente por aquel entonces: “con 640 KB es suficiente”. Cuando tenía 10 años, mi padre me regaló por mi cumpleaños un PC 386, y empecé con las comunicaciones de PC con un simple módem de 2400 bps.

¿A quién admiras en el mundo de la tecnología? Linus Torvalds, desde que creó el kernel Linux y Git. Linux y Git han cambiado el mundo del software.

¿Cómo decidiste trabajar en Hardkernel? El factor más importante fue lo que iba a hacer para Hardkernel. Las responsabilidades del puesto de trabajo me parecieron interesantes.

¿Cómo usas tus ODROID en casa? Disfruto haciendo cosas interesantes con los ODROIDs. Algunos de mis proyectos se pueden encontrar en ODROID Magazine, como Ambilight, cámara de vista posterior y ODROID Arcade Box. En la oficina de Corea del Sur, utilizamos los ODROIDs como servidores privados y comederos de peces automáticos. El sistema de minería de criptomonedas con 200 dispositivos ODROID-XU4 también fue un proyecto interesante. He creado un interruptor de luz por voz con el ODROID-C2 y Google Cloud Pech API para casa.

¿Cuál es tu ODROID favorito y por qué? El ODROID-C2 es mi favorito, porque soy uno de los principales desarrolladores de ODROID-C2 y porque tiene una arquitectura ARM de 64 bits. Aunque trabajo con todos los ODROIDS que están actualmente en venta (ODROID-C1 +, ODROID-C2 y ODROID-XU4), me uní a la empresa después de que se desarrollasen el XU4 y C1+.

¿Qué innovaciones se verán en futuros productos de Hardkernel? Pienso que mantener nuestra posición actual en el mercado de los SBC como dispositivos de alto rendimiento e intentar abrirnos camino en el mercado de servidores de gama baja es bueno para la supervivencia.

¿Qué aficiones e intereses tienes aparte de los ordenadores? Disfruto viajando, con los juegos de ordenador, el snowboard, el wakeboard, buceando y haciendo triatlones (natación, ciclismo y maratón). Realicé un curso olímpico de Triatlón el año pasado.

Este año he ido a Busan desde Seúl en bicicleta durante las vacaciones de verano. La distancia es de aproximadamente 325 millas (523 KM). Mi próximo reto es una maratón completa el próximo mes.

Figura 6. Los hobbies de Brian son el snowboard, el buceo, el triatlón, viajar y montar en bicicleta

¿Qué consejo le daría a alguien que quiere aprender más sobre programación? Leer un buen código fuente del software de código abierto. Escribir tantos códigos de alta calidad como puedas.