Marcobootstrap

Diego Martín Arroyo

Lunes, 11 de mayo de 2015

Resumen

El paquete Marcobootstrap comprende un conjunto de utilidades que permite llevar a cabo la descarga, instalación y actualización de un sistema operativo en varios nodos sin requerir la supervisión de un administrador, incluyendo el descubrimiento de los equipos necesarios para el desarrollo de dichas operaciones. Junto a estas se incluye una serie de aplicaciones de gestión del sistema utilizables por el administrador del sistema.

Índice

Introducción

El uso de un sistema conformado por una cantidad significativa de nodos dificulta su mantenimiento, y en particular la instalación de un conjunto de herramientas iniciales y las posteriores actualizaciones de las mismas. Dicho mantenimiento, si es llevado a cabo de forma "manual" suele propiciar fallos humanos o inconsistencias en el resultado final (nodos que no cuentan con el mismo conjunto de herramientas o con configuraciones diferentes). Por ello, en el desarrollo del sistema se ha apostado por una herramienta que automatice la instalación del sistema operativo reduciendo al máximo el tiempo de atención que el administrador debe prestar a cada nodo, facilitando además la gestión y actualización posterior. Dichas herramientas se apoyan sobre los principios básicos de desarrollo seguidos en todo el sistema, entre los que destaca el descubrimiento automático de equipos.
En el presente documento se detallan las decisiones de diseño llevadas a cabo en el desarrollo de las utilidades **marco-netinst**, **marco-bootstrap**, y **marco-bootstrap-backend** que llevan a cabo este cometido, así como el funcionamiento de las mismas.

Búsqueda de solución

marco-netinst Es la herramienta principal del sistema, que permite realizar instalaciones y actualizaciones de un sistema operativo emulando el funcionamiento de un servidor **PXE**, requiriendo únicamente un conjunto reducido de herramientas preinstaladas en la tarjeta SD para realizar todas las tareas. Se integra con MarcoPolo, por lo que no es necesario realizar ningún tipo de configuración para realizar el proceso de instalación previamente.
La complejidad del problema a resolver es significativa, pues implica la creación de una herramienta capaz de funcionar sin un sistema operativo completo, en un entorno en el cual la funcionalidad disponible es mínima.

Definición de requisitos

- El software no debe requerir configuración por parte del administrador, debe ser capaz de detectar toda la información necesaria a partir de las herramientas provistas.
- El tiempo de atención humana que la ejecución de la herramienta debe ser nulo o mínimo.
- El sistema debe ser capaz de integrarse en el sistema sin ningún tipo de configuración posterior a la instalación.
- Las herramientas creadas deben ser flexibles, a fin de poder adaptarlas a nuevas situaciones.

Evaluación de alternativas

Como se define en la memoria del Trabajo, las opciones típicamente utilizadas por administradores de sistemas no son viables en el sistema a construir debido a las características de la plataforma utilizada: los equipos **Rasbperry Pi** no soportan el protocolo **PXE**, debido a que el mismo no se incluye en el *software* que provee la tarjeta de red (como suele ocurrir en equipos de escritorio, en los que se integra como módulo de la **BIOS**), por lo que es necesario crear una herramienta propia, o buscar alternativas ya existentes creadas para solucionar un problema similar.

Estrategia inicial

El arranque de una placa **Rasbperry Pi** es llevado a cabo principalmente por la GPU de la misma. Al conectarse a la corriente eléctrica, se activa la secuencia de arranque definida en la memoria de solo lectura. Este código busca en la primera partición de la tarjeta SD un *bootloader* y lo carga, activando la memoria

SDRAM y pasando al siguiente estado, en el que se carga el archivo start.elf, que carga el Kernel del sistema operativo según los parámetros definidos en los ficheros `config.txt` (pues no tiene un cargador que indique los parámetros, como ocurre en sistemas de escritorio, en los que existen opciones como GRUB o LILO), `cmdline.txt` y **bcm2835.dtb**.

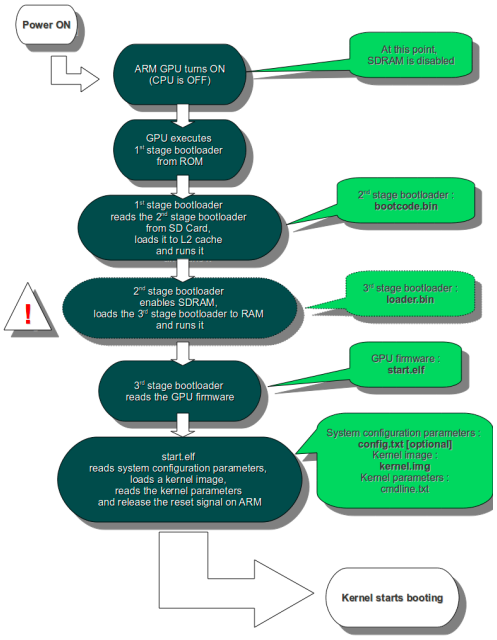

Figura 1: Secuencia de arranque de la Raspberry Pi[1]

Un archivo de configuración `cmdline.txt` sin modificar suele seguir la siguiente estructura:

```
root=/dev/mmcblk0p2 rw rootwait console=ttyAMA0,115200 console=tty1 \
selinux=0 plymouth.enable=0 smsc95xx.turbo_mode=N dwc_otg.lpm_enable=0 \
kgdboc=ttyAMA0,115200 elevator=noop
```

Los parámetros de interés son los siguientes:

- `root`: Partición raíz a utilizar (generalmente la segunda partición)
- `init`: Especifica el archivo a ejecutar en el arranque. Este proceso tendrá por tanto el **PID** 1 y generalmente su ejecutable es el fichero `/sbin/init`[1]. Si no se especifica, como ocurre en el archivo por defecto, su valor es este[2].

Modificando el parámetro `init` es posible ejecutar un código diferente a `init` en el proceso de arranque, que realice la descarga e instalación del sistema operativo, logrando una solución de compromiso entre la instalación manual y el uso de **PXE**.
Debido a la falta de experiencia en el desarrollo de herramientas similares, se han seguido varios enfoques hasta dar con la solución adecuada:

[1] En sistemas donde `systemd` es el proceso inicial, `/sbin/init` es un enlace simbólico al ejecutable de `systemd`

Primer enfoque: modificación del script de inicio

En este primer intento se crea un *script* y un ejecutable programado en C con un conjunto mínimo de utilidades que permitan la descarga del sistema operativo (utilizando herramientas como **cURL**. Estos archivos son instalados en la primera partición de la tarjeta SD, siendo correctamente ejecutados. Sin embargo, no es posible llevar a cabo la instalación debido a la falta de herramientas de configuración de red, y el enfoque presenta varios problemas adicionales: al carecer de un proceso `init`, el sistema es muy inestable, y es difícil realizar cualquier tarea más allá de la gestión de ficheros o tareas de mantenimiento básicas.

Segundo enfoque: Busybox

Busybox es una utilidad que aglutina un conjunto de herramientas UNIX en un único ejecutable de tamaño muy reducido, reemplazando así el uso de varios paquetes en unidades independientes, como **fileutils** o **shellutils**. Las utilidades a las que reemplaza cuentan con mucha menos funcionalidad, pero son capaces de crear un sistema utilizable con únicamente un *kernel* y varios ficheros de configuración[3]. Es la opción más común a la hora de crear sistemas embebidos o para realizar operaciones sin un sistema operativo completo. Utilizando BusyBox es posible configurar la interfaz de red de la Raspberry Pi mediante **DHCP** gracias a la herramienta **udhcpc**, incluida en la distribución de BusyBox estándar, así como a un conjunto de herramientas de utilidad como **wget**.

Incluyendo BusyBox en el código del primer proyecto es posible conseguir acceso a la red, pero el hecho de no utilizar el proceso **init** dificulta la mayor parte de operaciones, por lo que esta opción es descartada.

Enfoque exitoso

Antes de detallar la solución definitiva al problema, es necesario detallar las herramientas de terceros valoradas y descartadas:
Raspi-LTSP, PiNET
Estos proyectos[4, 5] están enfocados a la centralización de toda la información en un servidor, incluyendo la carga del sistema operativo, por lo que no constituyen una opción viable.

BerryBoot BerryBoot[6] es un sistema de arranque que posibilita la descarga e instalación de un sistema operativo y la coexistencia de varios en una misma tarjeta SD. Constituye una opción bastante prometedora, que sin embargo depende significativamente de una interfaz gráfica y de un usuario que interaccione con la misma, haciendo que su uso sea inviable para el objetivo deseado sin una intensa reestructuración.

Propuesta de solución

El proyecto **raspbian-ua-netinst**[7] posibilita la instalación de la distribución Raspbian desde un repositorio público, realizando el particionado de la tarjeta SD, la descarga e instalación de los ficheros y el resto de tareas necesarias desde la propia máquina, únicamente copiando en la tarjeta unos 15 *megabytes* de información. Este proyecto ha sido adaptado a las necesidades específicas del sistema, con las siguientes modificaciones:

- Instalación de **ArchLinux ARM** en lugar de **Raspbian**.
- Instalación del sistema operativo completo a partir de un archivo **.tar.gz** en lugar de la descarga de paquetes.
- Nuevo *script* de carga del *software* en la tarjeta SD (en el paquete original se delega a utilidades de terceros).
- Detección del servidor sin configuración previa utilizando **MarcoPolo**.

El sistema se compone de varios componentes, como se detalla en la siguiente sección.

marco-netinst

marco-netinst está compuesto por un conjunto de herramientas que permiten la gestión de los nodos del sistema y la instalación del sistema operativo en los mismos. Para ello se compone de varios módulos:

marco-netinst

Este módulo, basado en **raspbian-ua-netinst** únicamente comprende varios scripts, detallados a continuación:

update.sh

Descarga del repositorio indicado los paquetes que necesita el sistema operativo para arrancar. Dichos paquetes consisten básicamente en código de arranque, *kernels*, bibliotecas y un conjunto de herramientas básicas, siendo la más importante de ellas **BusyBox**. El *script* realiza la descarga en un directorio temporal de los archivos **.deb**.

build.sh

Crea los archivos necesarios para la instalación, que deberán ser copiados a la tarjeta SD. Para ello, realiza la siguiente secuencia de acción:

1. Copia del *kernel*.
2. Creación de los archivos de configuración.
3. Creación de los archivos **.cpio** (uno por cada arquitectura) que aglutina todas las bibliotecas, ejecutables y otros archivos de interés, así como los *scripts* que llevarán a cabo la instalación del sistema operativo.
4. Creación de los parámetros de la línea de comandos.
5. Copia de las utilidades de **MarcoPolo**.
6. Creación de un fichero **.zip**.

deploy.sh

Formatea la tarjeta SD, creando una única partición, donde descomprime el archivo **.zip** creado anteriormente.
Estos tres *scripts* deben ser ejecutados en el orden en el que han sido expuestos, pues dependen de los archivos generados por el resto. En caso de que no se realice la llamada de uno sin su antecesor, este procederá a su ejecución antes de realizar la funcionalidad que le corresponde.

clean.sh

Elimina todos los ficheros temporales y el resultado de las ejecuciones.
La mayoría de los archivos temporales que son descargados se conservan y son reutilizados en posteriores ejecuciones, optimizando el tiempo de instalación en varias máquinas.

marco-bootstrap

Ejecutable en C++ que permite llevar a cabo el descubrimiento del servidor donde se aloja el sistema operativo mediante **MarcoPolo**. Debido a la ausencia del *daemon* de **Marco** en la imagen mínima copiada a la tarjeta SD, se utiliza una implementación básica del protocolo, denominada **marco-minimal**. Esta implementación expone la misma funcionalidad que el **binding** de Marco en C++, pero a más bajo nivel.
Se apoya en la biblioteca **RapidJSON** para la generación e interpretación de cadenas **JSON**, así como en la API de sockets del sistema. Con estas dos herramientas es capaz de implementar toda la funcionalidad básica que requiere para utilizar **MarcoPolo**. La gestión de la compilación se lleva a cabo con la herramienta **Cmake**. Se integra con el script rcS mediante las salidas estándar y de error.

marco-bootstrap-backend

Interfaz de gestión de los del sistema. Dicha herramienta está diseñada para el administrador de la red, y mediante la misma es posible programar operaciones de actualización en varios nodos, el reinicio de máquinas o la preparación de imágenes de instalación.
La herramienta se apoya en MarcoPolo para la detección de los diferentes nodos y está construida sobre los *frameworks* **Tornado** y **Django**.

Funcionamiento básico

La interfaz se basa en la estructura de la herramienta MarcoDeployer: un conjunto de pestañas con funcionalidad relacionada.

Vista principal

En la vista principal el administrador puede visualizar las diferentes imágenes disponibles para su uso, los nodos sobre los que puede realizar operaciones, y un panel desde el que puede programar acciones sobre los nodos activos.

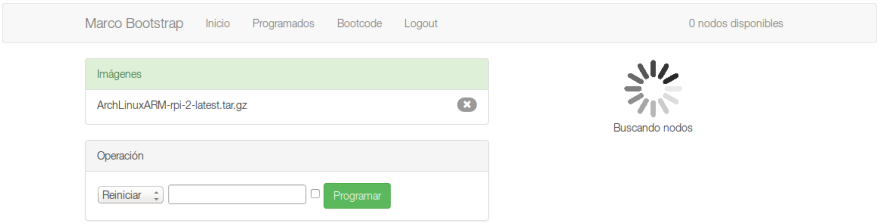

Figura 2: Vista de la interfaz inicial.

El administrador puede desde esta interfaz programar operaciones y enviarlas a un conjunto de nodos. Todas las operaciones pueden ser programadas para ser ejecutadas en un momento futuro.

Operaciones programadas

En ocasiones es necesario conocer las operaciones programadas para un nodo dado y contar con la posibilidad de cancelar las mismas. En la pestaña "Programados.[es] posible observar todos los eventos programados y los

nodos sobre los que se aplicarán. Las operaciones se presentan en orden cronológico.

Cancelación

Todas las operaciones cuentan con un identificador único en el sistema. Utilizándolo es posible cancelar operaciones futuras.

Persistencia

Es necesario mantener una serie de datos de forma persistente que sirvan para conocer las operaciones a realizar y aquellas ya ejecutadas. La herramienta utiliza el gestor relacional de objetos (*Object Relational Manager*) del *framework* Django, que abstrae la interacción con una base de datos, además de gestionar la seguridad de las operaciones ejecutadas en la misma (en especial la protección contra el ataque conocido como "Inyección SQL", mitigado de forma automática por el gestor). La abstracción permite intercambiar la capa permanente de forma sencilla.
Debido a la modularidad del **framework**, es posible integrar el ORM en Tornado con poco esfuerzo, realizando únicamente una serie de ajustes previos en la configuración de la aplicación.
Como gestor de la base de datos se utiliza SQLite, un gestor que implementa un subconjunto del estándar SQL y que se caracteriza por su simplicidad y eficiencia. Es ideal para su uso en sistemas embebidos como las Raspberry Pi. Su simplicidad es tal que el gestor no se incluye como una pieza independiente sobre la que una serie de aplicaciones se conectan (tal es el caso de gestores de bases de datos como MySQL u Oracle Database), sino que se incluye dentro de la aplicación que hace uso de una base de datos.
En la base de datos existe una única tabla, que almacena el tiempo en el que se ha programado una operación, los nodos sobre la que se ha realizado, el tipo de operación y cualquier otro parámetro adicional.

marco-bootstrap-backend-slave

Los esclavos son los encargados de llevar a cabo las operaciones programadas, así como la cancelación de las mismas en caso de que sea necesario.

Reinicio

El reinicio del sistema se realiza mediante la orden **shutdown** proporcionada por el sistema operativo. Se ha dispuesto una espera de dos minutos previa al reinicio a fin de que los usuarios puedan guardar su trabajo de forma segura.

Actualización

Cada esclavo cuenta con una imagen del conjunto de herramientas necesarias para realizar el proceso de actualización, similares a las utilizadas en el proceso de instalación del sistema operativo. Dicha imagen es instalada en la primera partición (sobreescribiendo todos los archivos allí existentes de mismo nombre) y sus contenidos son leídos por la placa al realizarse el reinicio que se programa tras la extracción. Con el objetivo de garantizar la estabilidad del sistema, los ficheros son sobreescritos en el momento posterior al reinicio, y no en el momento en el que la operación es programada.

Programación

Los diferentes eventos se programan en el futuro utilizando el bucle de eventos de **Tornado**.

Seguridad

La autenticación del administrador se realiza mediante una combinación de usuario y contraseña (almacenada como *hash* SHA de 256 bits). Todas las comunicaciones (exceptuando la descarga de una imagen de sistema operativo) se realizan mediante el protocolo HTTPS, y en el caso de la programación y cancelación de operaciones la identificación de cada una de las partes se realiza mediante certificados SSL tanto en cliente como en servidor.

Instalación

El conjunto de herramientas que realizan la instalación del sistema operativo están basadas en **Debian**, y por tanto utilizan el gestor de arranque **init**. Para realizar la instalación por tanto se delegará la ejecución de los *scripts* que realicen las funciones a dicho gestor. Por ello, se define el script `/etc/rcS`, que es ejecutado en cualquier nivel de ejecución (*runlevel*)[2].
El script lleva a cabo la siguiente secuencia de tareas:

1. Define una serie de variables de utilidad.
2. Crea de directorios necesarios dentro de la tarjeta SD.
3. "Instala" todas las utilidades de **BusyBox** (permite el acceso a las mismas) y define las rutas de búsqueda de ficheros mediante la variable `$PATH`.
4. Monta de los pseudosistemas de ficheros `/proc` y `/sys`.
5. Establece mecanismos de redirección de una copia de las salidas estándar y de error (**stdout** y **stderr**) a un fichero de log para su posterior análisis.
6. Determina el tipo de *hardware* sobre el que se está ejecutando el *script*, a fin de instalar versión del sistema operativo apropiada.
7. Copia los ficheros de arranque contenidos en la tarjeta SD.
8. Carga un *script* con parámetros adicionales definidos por el usuario.
9. Configura de la interfaz de red **eth0** mediante **DHCP**.
10. Determina de la hora mediante **NTP**.
11. Carga módulos del *kernel* si son necesarios.
12. Particiona la tarjeta SD según el siguiente esquema:
    - Partición 1: 128 megabytes, formato FAT32.
    - Partición 2: Resto de la tarjeta SD, formato ext4.
13. Busca de servidores que alojen el sistema operativo mediante **MarcoPolo**.
14. Descarga el sistema operativo como archivo `.tar.gz`.
15. Descomprime el sistema operativo en la partición número 2 (donde la mayoría de ficheros se almacenan. La estructura del fichero `.tar.gz` define el lugar donde cada fichero se debe alojar, y los permisos de acceso a los ficheros son conservados en el proceso de extracción, por lo que con la extracción se consigue además la estructuración del sistema y la gestión de permisos).

[2] apt-browse.org/browse/ubuntu/trusty/main/all/sysv-rc/2.88dsf-41ubuntu6/file/etc/rcS.d/README

16. Mueve los ficheros de arranque a la partición 1 (*kernel*, parámetros de arranque, etcétera).
17. Ejecuta los *scripts* de post-instalación.
18. Almacena el fichero de log.
19. Limpia archivos temporales.
20. Desmonta los sistemas de ficheros y procede al reinicio del sistema.

Una vez ejecutado el script el nodo está preparado para integrarse en el sistema. El tiempo total de instalación es de unos 5 minutos, según el fichero de log, dependiendo del estado de la red. En pruebas realizadas, el tiempo de descarga supone aproximadamente la mitad del tiempo total de instalación, si bien mejora significativamente cuando se cuenta con un servidor local que aloje el sistema operativo.

Actualización

El administrador puede programar actualizaciones utilizando la herramienta de backend y enviar notificaciones a los diferentes nodos del sistema. Estos realizarán la actualización en el siguiente reinicio del sistema, o si lo desea el administrador, de forma programada o inmediata.
El administrador únicamente debe subir a la interfaz web una imagen del sistema operativo a replicar, que incluirá todos los paquetes necesarios. Los nodos reemplazarán su sistema operativo por este mediante un proceso similar al de instalación.
Para realizar la creación de la imagen existe la herramienta `createimage.sh`, que requiere como parámetro un descriptor de dispositivo (una tarjeta SD o un archivo .img) que contenga los archivos a copiar.
Los nodos cuentan con un servicio (vinculado a MarcoPolo) que escucha peticiones (verificadas por un certificado SSL) de actualización (o cualquier otra funcionalidad que pueda ser añadida, apostando por la versatilidad en el desarrollo de la plataforma). Al recibir una petición válida (el certificado enviado pertenece a un servidor de despliegue), comienza el siguiente proceso:

- Comprueban los parámetros de la petición, y actúan en consecuencia.
- En el caso de que se solicite una actualización, se leen los parámetros de la petición, que incluyen el momento de reinicio y el nombre del fichero a solicitar al servidor.
- Copia a la partición 1 (arranque) los ficheros kernel.img, paquetes Debian necesarios, Busybox y el script /etc/init/rcS, que en caso de ser necesario, deberá ser modificado para atender a las características propias de cada consulta.
- Programa un reinicio del sistema en el momento definido por el administrador (inmediatamente, en un momento determinado o en el momento en el que se produzca un reinicio originado por otra causa).
- Al reiniciarse se realiza una secuencia de pasos similar a la del proceso de instalación.

A fin de no eliminar la información almacenada por los usuarios, o cualquier otro tipo de datos que deban sobrevivir a este tipo de actualizaciones, se definirán particiones que no serán alteradas.

Tiempo

El tiempo de interacción que el administrador debe emplear por cada máquina es de aproximadamente un minuto, el tiempo necesario para conectar una tarjeta SD, ejecutar el script de carga de los paquetes de instalación, y la desconexión de la misma.
El tiempo de instalación es variable, dependiente principalmente de la velocidad de descarga del sistema operativo y el tiempo de descompresión y copia de este, si bien suele ser inferior a 10 minutos (y este tiempo es paralelo para todas las máquinas, suponiendo únicamente una mayor carga para el servidor).

Evaluación

Se han realizado las siguientes evaluaciones de uso:

- Entrevista con el Administrador de la instalación el día 13 de mayo, donde se le presenta la solución propuesta y los resultados obtenidos. Según su criterio considera que el enfoque es el adecuado.

Referencias

[1] SG60, "What is the boot sequence?." `http://raspberrypi.stackexchange.com/questions/10442/what-is-the-boot-sequence`, nov 2013. Consultado: 2015-05-11.

[2] L. K. Organization, "Kernel parameters." `https://www.kernel.org/doc/Documentation/kernel-parameters.txt`.

[3] D. Vlasenko, "About BusyBox." `http://www.busybox.net/about.html`, mar 2015. Consultado: 2015-05-11.

[4] A. Mulholland, "RaspberryPi-LTSP." `https://github.com/gbaman/RaspberryPi-LTSP`, apr 2015. Consultado: 2015-05-11.

[5] A. Mulholland, "PiNet." `http://pinet.org.uk/`, apr 2015. Consultado: 2015-05-11.

[6] F. Bos, "BerryBoot v2.0 - bootloader / universal operating system installer." `http://www.berryboot.com/`, mar 2015. Consultado: 2015-05-11.

[7] Rasbpian, "raspbian-ua-netinst." `https://github.com/debian-pi/raspbian-ua-netinst`, May 2015.

[8] eLinux.org, "Rpiconfig - elinux.org." http://elinux.org/RPiconfig. Consultado: 2015-05-11.