Infraestructura distribuida para la construcción de paquetes Debian

UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

Infraestructura distribuida para la construcción de paquetes Debian

José Luis Sanroma Tato

Abril, 2013

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

Infraestructura distribuida para la construcción de paquetes Debian

Autor: José Luis Sanroma Tato

Director: Dr. Francisco Moya Fernández

Abril, 2013

José Luis Sanroma Tato

Ciudad Real – Spain

E-mail: josel.sanromatato@gmail.com
Web site:

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

Índice general

Índice general VI

Índice de figuras VIII

Índice de listados IX

Listado de acrónimos X

1. Objetivos 1

1.1. Objetivo general . . . 1

1.1.1. Objetivos específicos . . . 1

2. Metodologías y herramientas 2

2.1. Metodología de desarrollo . . . 2

2.1.1. SCRUM . . . 2

2.1.2. Procesos . . . 2

2.2. Herramientas . . . 5

2.2.1. ZeroC ICE . . . 5

2.3. Iteracion 0 . . . 9

2.3.1. Máquinas virtuales . . . 9

2.3.2. Arquitectura mínima . . . 11

2.3.3. Implementación y ejecución del «Hello World» . . . 12

2.4. Iteracion 1 . . . 13

2.4.1. Usar o no pbuilder . . . 13

2.4.2. User Stories . . . 15

2.4.3. Firma de paquetes: The GNU Privacy Guard . . . 18

2.5. Iteracion 2 . . . 20

2.5.1. Prueba para dos paquetes . . . 20

2.5.2. Logger . . . 20

A. Manual de instalación de un Nodo **24**
A.1. Introducción . . . 24
A.1.1. Estructura del documento . . . 24
A.2. ¿Qué es Node-Builder? . . . 24
A.2.1. ¿Por qué Node Builder? . . . 25
A.3. Requisitos . . . 26
A.3.1. Requisitos de hardware . . . 26
A.3.2. Paquetes necesarios . . . 26
A.3.3. Requisitos de ejecución . . . 26
A.3.4. El archivo preseed . . . 27
A.4. Instalación . . . 27
A.4.1. Instalación de máquinas virtuales . . . 27
A.5. Configuración . . . 29
A.5.1. Configuración de máquinas virtuales . . . 29
A.5.2. Configuración de sudo . . . 29
A.5.3. Configuración de repositorios . . . 30
A.5.4. Configuración de Grub . . . 30

B. GNU Free Documentation License **32**
B.0. PREAMBLE . . . 32
B.1. APPLICABILITY AND DEFINITIONS . . . 32
B.2. VERBATIM COPYING . . . 33
B.3. COPYING IN QUANTITY . . . 33
B.4. MODIFICATIONS . . . 33
B.5. COLLECTIONS OF DOCUMENTS . . . 34
B.6. AGGREGATION WITH INDEPENDENT WORKS . . . 34
B.7. TRANSLATION . . . 34
B.8. TERMINATION . . . 35
B.9. FUTURE REVISIONS OF THIS LICENSE . . . 35
B.10. RELICENSING . . . 35

Bibliografía **36**

Índice de figuras

2.1. Scrum Process 3
2.2. Esquema de un sprint en Scrum 4
2.3. Sistema base formado por un computador y varias máquinas virtuales . . . 10
2.4. Arquitectura de un Nodo y del sistema 16
2.5. Firma de claves gpg 19

Índice de listados

2.1. Ejemplo de interfaz Slice 7
2.2. Cliente «HelloWorld» distribuido en Python 12
2.3. Servidor «HelloWorld» distribuido en Python 12
2.4. Interfaz Logger en Slice 21

Listado de acrónimos

GPL	General Public License
ICE	Internet Communication Engine
IP	Internet Protocol
Slice	Specification Language for Ice
SGBD	Sistema de Gestión de Bases de Datos

Objetivos

1

En este capítulo se detallan los objetivos del proyecto, tanto generales como específicos. De esta manera, se determina el alcance del proyecto y los resultados esperados.

1.1. Objetivo general

Disponer de una infraestructura distribuida para la construcción de paquetes en un repositorio Debian. Una infraestructura que, en definitiva, automatice la construcción de paquetes Debian en un repositorio para ayudar a generar los paquetes para el resto de arquitecturas soportadas por el laboratorio ARCO.

1.1.1. Objetivos específicos

En general se pueden definir los siguientes objetivos:

- Diseñar una plataforma que cumpla con las necesidades que existen, para ello se tomará de ejemplo el laboratorio ARCO.
- Debe ser fácil de instalar, de configurar y de mantener. Para ello se empaquetarán las partes del mismo en paquetes Debian.
- La plataforma deberá conocer las arquitecturas a las que debe dar soporte.
- La construcción de paquetes debe realizarse en un sistema limpio con el fin de no ensuciar el sistema con dependencias y paquetes que van quedando instalados para construir los paquetes Debian. Además de esta forma se evitan problemas de seguridad, pruebas, y riesgo de romper el sistema.
- Si un paquete no se puede construir, la plataforma debe ofrecer información suficiente acerca del error para que se le pueda poner remedio e intentar de nuevo su construcción.
- Cuando los paquetes están en el repositorio y un usuario realiza cambios en un programa, al subirlo de nuevo la plataforma deberá hacer el mismo proceso que cuando se sube un paquete nuevo. Además se deberá comprobar que las versiones se corresponden con la más nueva.

Metodologías y herramientas

2

2.1. Metodología de desarrollo

2.1.1. SCRUM

El enfoque de Scrum [PAW13] ha sido desarrollado para manejar el desarollo de procesos. Se trata de un enfoque empírico de aplicando las ideas de proceso industrial la teoría de control para el desarrollo de sistemas que resulta en un enfoque que retoma las ideas de flexibilidad, adaptabilidad y productividad.

La idea principal de Scrum es que el desarrollo del sistema envuelve muchas variables (requisitos, tiempo, recursos y tecnología) que pueden cambiar durante el proceso. Esto hace el proceso de desarrollo impredecible y complejo, requiriendo felxibilidad para ser capaz de responder a los cambios que irán surgiendo.

El proceso Scrum ayuda a mejorar la existencia de prácticas ingenieriles en la organización, ya que implica frecuentes actividades de gestión destinadas a identificar las deficiencias o impedimentos en el desarrollo del proceso, así como las prácticas que se utilizan.

2.1.2. Procesos

El proceso de Scrum está formado por tres partes:

1. Pre-game phase
2. Development phase
3. Postgame phase

Roles y responsabilidades

Se pueden identificar seis roles en SCRUM que tienen distintas tareas y propósitos, estos son: Scrum Master, Product Owner, Scrum Team, Customer, User y Management.

Scrum Master: Es una función de administrador dentro de Scrum. Es el responsable de asegurarse que el proyecto se haga de acuerdo a las prácticas establecidas y de que el proyecto avanza según lo previsto. Se encarga también de liderar al equipo de desa-

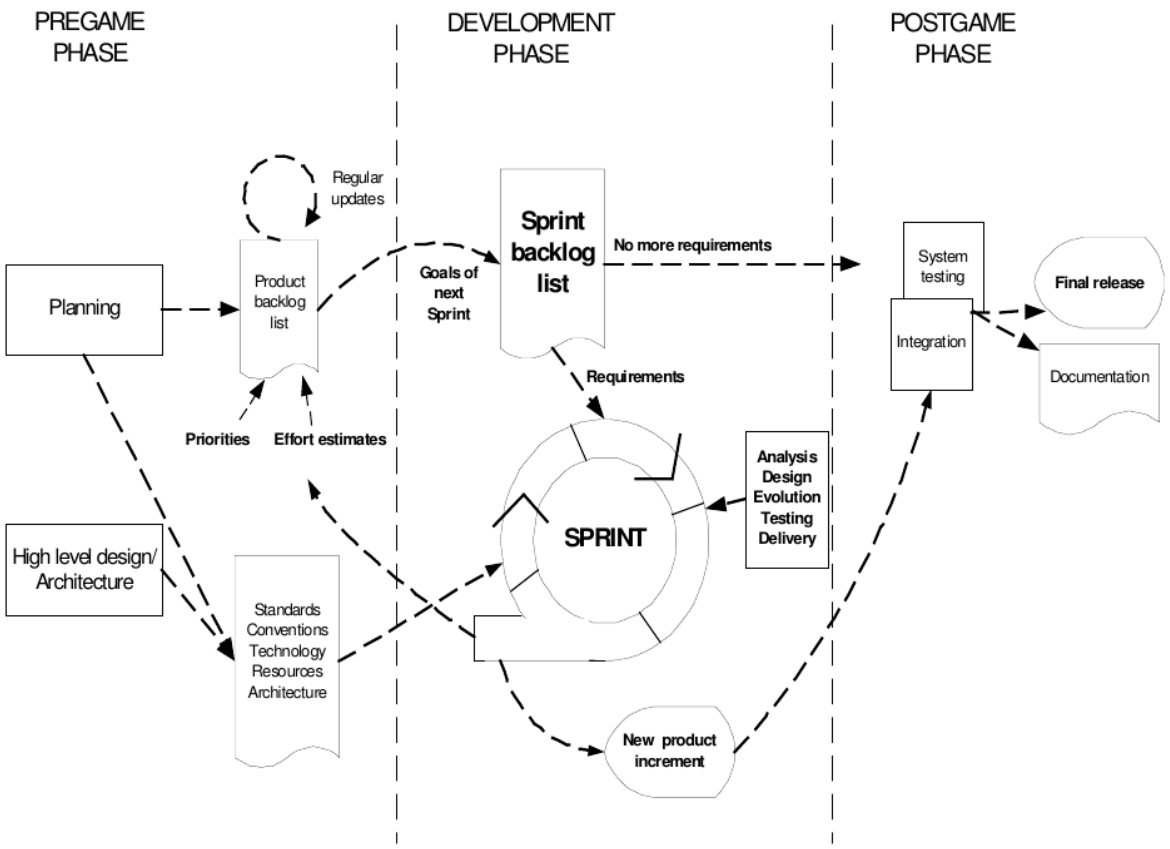


Figura 2.1: Scrum Process

rrollo cuando se encuentra algún obstáculo, y de mantener el equipo de modo que se ocupe de las tareas de la forma más productiva posible.

Product Owner: Es el responsable del proyecto, gestión, control. Lo elige el Scrum Master. Participa sobre todo en la estimación de esfuerzo de los elementos.

Scrum Team: Es el equipo del proyecto, tienen la autoridad para decidir sobre las acciones y la organización necesaria para alcanzar el éxito. También está implicado en toma de decisiones en cuanto a estimación del esfuerzo.

Customer: Participa en las tareas relacionadas con el "backlog" para el sistema que está siendo desarrollado o mejorado.

Management: Es responsable de las decisiones finales de las normas que se tienen que seguir en el proyecto. También participa en la redacción de requisitos y objetivos a cumplir.

Prácticas

Scrum no exige ni proporciona ningún método ni prácticas de desarrollo de software. Sin embargo, requiere de ciertas prácticas y manejo para evitar el caos creado por la complejidad y la imprevisibilidad.

Product Backlog Define todo lo que es necesario en el producto final basado en el conocimiento actual que se tiene. El "Product Backlog"define el trabajo que se tiene que hacer en el proyecto. Se compone de una lista con prioridades que se actualizan constantemente. Pueden incluir características, funciones, correciones de errores, mejoras solicitadas, etc. Varias personas pueden participar en la generación del "Product Backlog", como clientes, equipos de desarrollo, etc.

Estimación de esfuerzo Es un proceso interactivo, en el cual las estimaciones son más precisas cuanta más información se tiene. El "Product Owner"junto con el "Scrum Team"son los responsables de realizar la estimación.

Sprint Es el proceso por el cual se van adaptando los cambios al proyecto. El "Scrum Team"se organiza él mismo para producir nuevos ejecutables en cada iteración que dura aproximadamente unos 30 días. Reuniones de planificación de sprints, reuniones diarias y Sprint Backlog son algunas de las actividades que se realizan en los Sprint.

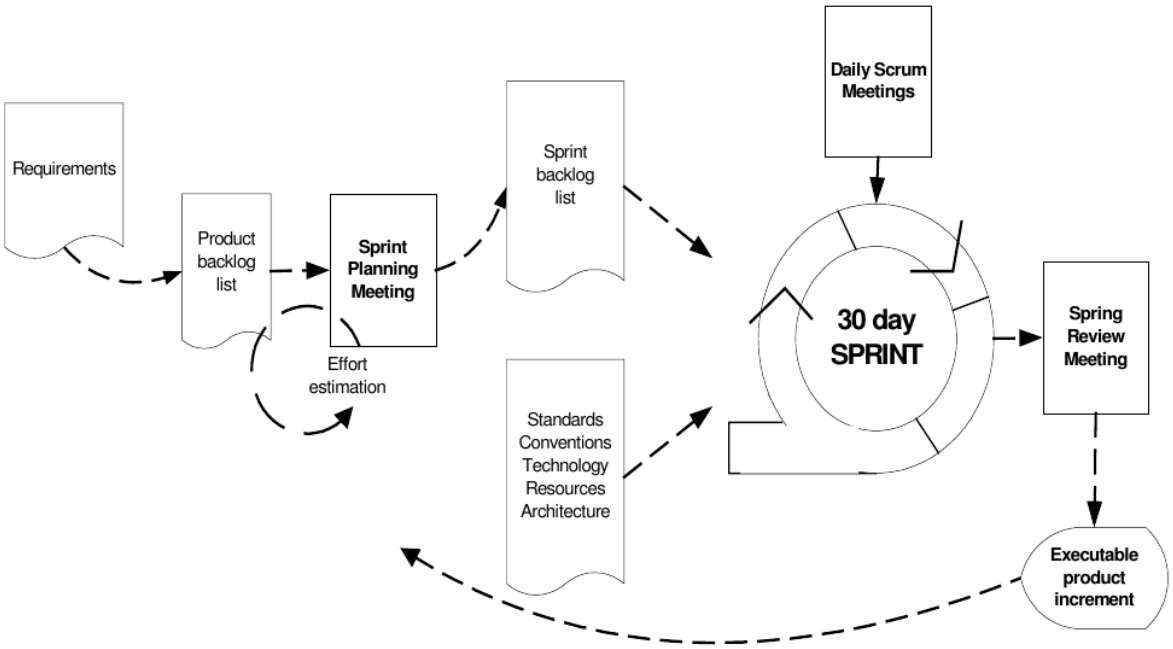


Figura 2.2: Esquema de un sprint en Scrum

Sprint Planning meeting Es organizada por el Scrum Master y tiene dos fases. En la primera de ellas participan el cliente, usuarios, administradores y scrum team, en esta fase se definen los objetivos y la funcionalidad para el siguiente Sprint. La segunda fase se realiza entre el Scrum Master y el Scrum Team, centrándose en como se implementarán las cosas durante el Sprint.

Sprint Backlog Es el punto de partida de cada Sprint, es una lista de elementos del "Product Backlog"seleccionados para ser implementados en el siguiente Sprint. Estos elementos son seleccionados por el "Scrum Teamz "Product Owner.[en] la *Sprint Planning meeting*.

Daily Scrum meeting Son reuniones diarias para mantener un seguimiento. Se responde a preguntas del tipo: Qué se ha hecho desde la anterior reunión y que se va a hacer para la siguiente. También se discuten problemas que han surgido. Duran en torno a 15 minutos. El Scrum Master es quien las dirige.

Sprint Review meeting El último día del Sprint, se reunen el Scrum Team y el Scrum Master y se presentan los resultados a los administradores, clientes, usuarios etc. Los participantes el prototipo entregado. Esta reunión debe dar un nuevo Backlog con nuevos elementos y si se debe o no cambiar de dirección en el desarrollo.

2.2. Herramientas

2.2.1. ZeroC ICE

Internet Communications Engine (Internet Communication Engine (ICE)) [HS10] es un middleware orientado a objetos construido por la empresa ZeroC y con licencia GPL.

Soporta varios lenguajes de programación, lo cual permite una mayor adaptabilidad según las necesitadades del momento. ICE

Clientes y servidores

Los conceptos cliente y servidor varian un poco en lo fundamental, el cliente requiere la funcionalidad que el servidor proporciona. Los servidores ICE necesitan un adaptador de objetos donde poder añadir los sirvientes necesarios para ofrecer el servicio. En cuanto a los clientes, necesitan un proxy al servidor para poder solicitar la funcionalidad requerida por la aplicación.

Adaptador de objetos

El adaptador de objetos actúa como un contenedor del objetos del servidor que son accedidos mediante invocaciones remotas. En cada adaptador de objetos hay unas direcciones

asignadas que se conocen como **endpoints**. En ICE un ejemplo de **endpoint** sería el siguiente:

```
tcp -h 127.0.0.1 -p 9999
```

Este ejemplo es un endpoint `tcp`, o explicado de otra forma, utilizará el protocolo **TPC!** y escuchará en la interfaz con dirección IP 127.0.0.1 y en el puerto 9999.

Objetos y sirvientes

Los **objetos** en ICE son el mismo concepto que los de cualquier lenguaje de programación. Pero con algunas características:

- Una entidad en el espacio de direcciones remoto o local capaz de responder a las peticiones de los clientes.
- Un objeto puede instanciarse en un servidor o en varios servidores.
- Cada objeto tiene una o más interfaces, la cual tiene varias operaciones soportadas por un objeto. Los clientes son los que invocan estas operaciones.
- Una operación tiene cero o más parametros y un valor de retorno. Tanto los parámetros como el valor de retorno son de un tipo específico.
- Cada objeto tiene una identidad única, con esta identidad única, se distingue a un objeto de otros.

Las peticiones de los clientes deben terminar ejecutándo código en el servidor. El componente en el lado del servidor que proporciona el comportamiento que está asociado a la invocación se denomina **sirviente**. Cuando un sirviente se añade a un adaptador, es necesario asignale una **identidad de objeto**, para identificarlo globalmente en el sistema distribuido.

Proxy

Para identificar un sirviente bastará con conocer su identidad y el `endpoint` del adaptador de objetos donde se encuentra:

```
Objeto1 -t:tcp -h 127.0.0.1 -p 9999
```

El sirviente con la identidad `Objeto1` es accesible en el endpoint `tcp -h 127.0.0.1 -p 9999`. El valor `-t` quiere decir que el modo de acceso al objeto es **twoway**, o lo que es lo mismo, que el sirviente puede recibir mensajes y se esperarán sus respuestas

Slice

¿Cómo sabe el cliente el tipo de objeto que hay al otro lado? El servidor y el cliente deben compartir información sobre los objetos involucrados en la comunicación, así como sus atributos, métodos, etc. **Specification Language for Ice (Slice)** es el lenguaje para especificar el contrato entre el cliente y el sevidor. Se define la interfaz y las operaciones que serán accesibles mediante un *poxy* a un determinado sirviente. ICE proporciona herramientas como `slice2cpp` o `slice2java`para traducir la interfaz SLICE a C++ u Java respectivamente. ICE también da soporte a: Python, .NET, PHP, Pbjetive-C, Ruby y ActionScript.

```
module Prueba {
  interface Cajero {
    void sacar_dinero(int dinero);
    void ingresar_dinero(int dinero);
  };
};
```

Listado 2.1: Ejemplo de interfaz Slice

El listado 2.1 muestra un ejemplo muy sencillo de SLICE. Este archivo debe ser compartido entre el cliente y el servidor. En el ejemplo se puede ver una interfaz `Prueba.Cajero` que tiene dos métodos `ingresar()` y `sacar()`.

Un objeto que implemente la interfaz `Prueba.Cajero` podrá ser añadido a un adaptador de objetos de un servidor. As u vez, el cliente, puede acceder al sirviente asegurándose así que cumple con la interfaz que se ha definido previamente en el archivo SLICE. Tanto el cliente como el servidor pueden estar implementados en diferentes lenguajes de programación, esto, sin duda alguna, es una ventaja porque se pueden implementar clientes en diferentes lenguajes sin necesidad de hacer cambios en el servidor.

Servicios de ICE

ICE ofrece una serie de servicios, estos servicios suministran la funcionalidad que la mayoría de aplicaciones distribuidas requieren. Estos servicios proporcionan un excelente rendimiento y pueden ser replicados para lograr tolerancia a fallos y escalabilidad.

- **IceGrid**: *IceGrid* es uno de los servicios más importantes de ICE, proporciona una gran cantidad de funcionalidades para un *grid* de ordenadores como activación automática de objetos, balanceo de carga y transparencia de localización. Permite que un cliente se comunique con un objeto remoto sin saber en qué nodo se encuentra ni en qué puerto está escuchando. *IceGrid* incluye dos herramientas de administración para controlar sus características, `icegridadmin` la cual tiene una interfaz por linea de comandos y `icegrid-admin` con una interfaz gráfica. Es importante conocer algunos de los conceptos que maneja *IceGrid*:

Nodo Identifica a un «nodo lógico», puede haber más de un nodo en un computador.

Servidor Identifica mediante un nombre único a un programa que se ejecutará en un nodo.

Adaptador de objetos Tiene datos específicos de un adaptador de objetos utilizado en un servidor ICE como endpoints,

Aplicación Conjunto de servicios, objetos y configuraciones que forman la aplicación distribuida. Se puede almacenar en un fichero XML.

- **IceStorm**: *IceStorm* es el servicio de ICE que proporciona comunicación entre clientes y servidores a través de canales de eventos. En este ámbito es más común utilizar *publicador* y *suscriptor* en lugar de cliente y servidor. El o los publicadores envían datos a un canal mediante invocaciones remotas, que serán enviadas a los suscriptores de dicho canal. Por ejemplo, imagínese que hay nodos que quieren recibir audio de un nodo servidor. El publicador puede publicar el audio en un canal de eventos de tal forma que los clientes que quieran recibir el audio solamente tendrán que suscribirse a ese canal. Cada canal se identifica por su nombre y puede tener múltiples publicadores y suscriptores.
- **Freeze y FreezeScript**: *Freeze* es un servicio de persistencia que permite guardar el estado de los objetos en una base de datos *Berkeley*. *Freeze* puede recuperar automáticamente los objetos de una base de datos sobre la demanda, y automáticamente actualizar la base de datos cuando cambia el estado del objeto. *FreezeScript* es un lenguaje de comandos que permite consultar el contenido de una base de datos *Freeze*. Resulta de mucha utilidad para la depuración de aplicaciones. *FreezeScript* ofrece funciones de transformación que permiten migrar de forma sencilla de una base de datos existente para adaptarse a un nuevo esquema
- **Glacier2**: Es un servicio de firewall que permite la comunicación segura entre clientes y servidores.
- **IcePatch2**: *IcePatch* permite la distribución de actualizaciones de software a los nodos que componen el *grid*. *IcePatch* comprueba automáticamente la versión que tiene cada nodo y envía las actualizaciones disponibles.

Lenguajes de programación

Python Lenguaje de alto nivel, interpretado y orientado a objetos. Se pueden construir prototipos en poco tiempo sin mucho esfuerzo. Ha sido utilizado para implementar algunas pruebas.

C++

Bash

Documentación

GIMP Editor de imágenes u retoque fotográfico. Utilizado en la maquetación de fotografías e ilustraciones.

Inkscape Editor de imágenes vectorial.

Dia Editor de diagramas. Utilizado para construir los diagramas que se han utilizado para el desarrollo del proyecto.

LaTeX Lenguaje de maquetación de textos, utilizado para la elaboración del presente documento.

Sistemas Operativos

Debian GNU/Linux

Hardware

2.3. Iteracion 0

En esta iteración se define el sistema base para todo el proyecto. Al tratarse de un sistema distribuido, y dado que no se dispone de infinitos computadores, se ha obtado por la utilización de máquinas virtuales como medio para la realización de las pruebas y montar el sistema, por lo tanto, se requiere de un estudio de las alternativas que existen para la virtualización.

Además se construirá un sistema base con una máquina virtual de arquitecturas *i386*, *amd64* y *armel* tal y como se muestra en la figura 2.3

Una máquina virtual es un sistema de virtualización por software, es decir, simula un sistema físico (en este caso un computador) con unas características de hardware determinadas. Cuando se ejecuta, proporciona un ambiente de ejecución similar a todos los efecto a un computador físico, con su CPU (puede ser más de una), memoria RAM, tarjeta gráfica, tarjeta de red, sistema de sonido, dispositivos de almacenamiento secundario.

Con esto se consigue ejecutar uno o más computadores dentro de un mismo computador de manera simultanea, y se resuelve así el problema de no tener muchos computadores disponibles donde probar el sistema.

2.3.1. Máquinas virtuales

Se procede al estudio de las distintas alternativas que existen para virtualizar computadores de diferentes arquitecturas. Para ello se consultan varias herramientas basando la búsqueda en los los siguientes requisitos:

- Se requiere de la virtualización de máquinas virtuales de distintas arquitecturas, como mínimo *i386* y *amd64*, aunque también sería deseable poder tener *armel*,

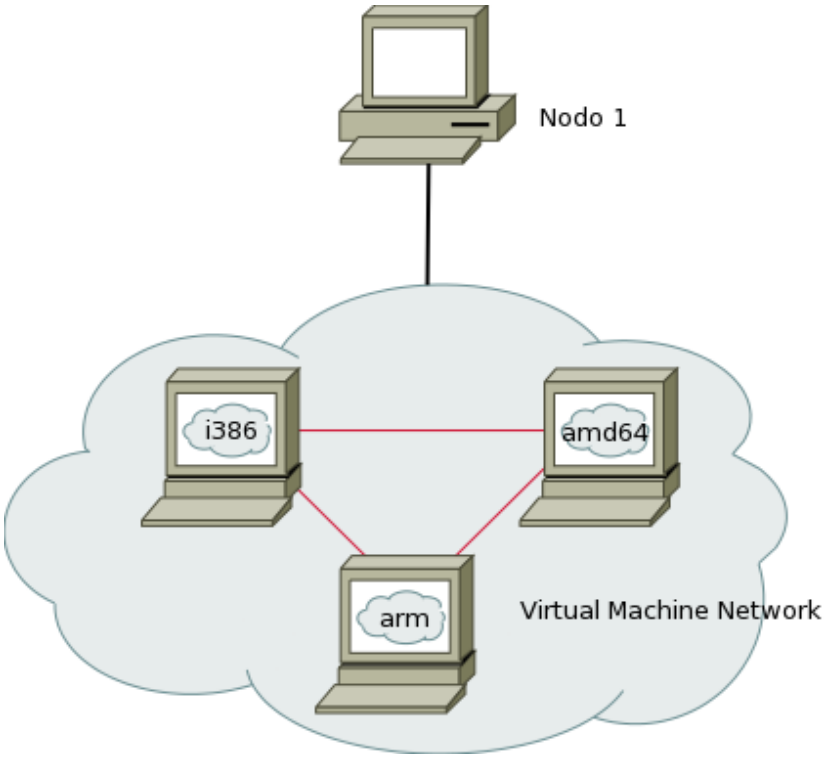


Figura 2.3: Sistema base formado por un computador y varias máquinas virtuales

- Deben poderse realizar las siguientes acciones con las máquinas virtuales:
  - **Parar**: la herramienta debe permitir parar una o varias maquinas virtuales cuando se requiera manteniendo otras encendidas.
  - **Iniciar**: la herramienda debe permitir el inicio de una o varias máquinas virtuales cuando se requiera.
  - **Clonar**: Se debe permitir el clonado de una o varias máquinas virtuales con el fin de poder añadir más nodos al sistemas distribuido.
  - **Reiniciar**: Se debe permitir que una o varias máquinas virtuales se reinicien cuando se requiera.
  - Que sea software libre.
- Además de todo ello sería deseable que las máquinas virtuales se puedan gestionar mediante comandos con el fin de construir algún script que facilite las operaciones.

En base a estos criterios se procede al estudio de las siguientes herramientas: VMware [VMw13], VirtualBox [Cor13], QEMU [Bel13], Gnome Boxes, Libvirt [RH11]

VMWare: Software de virtualización disponible para computadores x86. Este programa no dispone de una licencia de software libre, por lo tanto no cumple con uno de los requisitos y se descarta su uso por esta razón.

VirtualBox: Software de virtualización para arquitecturas *x86* y *amd64*, es multiplataforma, lo cual quiere decir que se pueden tener instancias en varios sistemas operativos como *Windows Mac* o *GNU/Linux*. Dispone de gestión mediante comandos a través del comando `VBoxManage` así como de una interfaz gráfica donde poder instalar el sistema operativo deseado. Su licencia es *GPL*, sin embargo un punto en contra es la no virtualización de la arquitectura *arm*. Se pueden realizar todas las operaciones descritas en los requisitos, por lo tanto se convierte en candidato.

QEMU: En inglés Quick EMUlator, es capaz de virtualizar diferentes tipos de CPU, y de este modo se pueden virtualizar diferentes arquitecturas. Dispone de interfaz gráfica y de administración mediante comandos, gracias a esto y con la ayuda del manual, es muy sencillo emular un procesador con arquitectura *arm* y por consiguiente realizar la instalación de una versión de Debian para esa arquitectura. Se pueden realizar todas las acciones descritas en los requisitos, por lo que se convierte en candidato.

gnome-boxes: Gnome boxes: Es una aplicación del escritorio GNOME para acceder a sistemas virtuales ya sea a través del propio computador o remotamente. Pretende tener funcionando rápidamente y de forma fácil máquinas virtuales. Gracias a su sencilla interfaz resulta extremadamente fácil de usar. Está diseñado para alguien que sólo requiera encender o apagar lás máquinas virtuales, utiliza *qemu* como motor para virtualizar pero no se pueden realizar todas las operaciones que sí se pueden realizar con *qemu* o *VirtualBox*.

Libvirt: Es una API para virtualización que puede realizar todas las operaciones descritas en los requisitos además de otras como monitorizar y modificar entre otras. Dispone de una interfaz gráfica fácil de usar donde se puede ver la carga de los sistemas virtualizados. Permite ejecutar órdenes por comando a través del comando `virsh` o `virt`. Tiene licencia *LGPL*, y al igual que *gnome-boxes* usa *qemu*, por lo tanto se convierte en candidato.

Tras este análisis y tras haber probado cada una de las alternativas la opción final ha sido utilizar *Libvirt*. Dispone de un abanico de operaciones más amplio para interactuar con los sitemas virtualizados, además al utilizar *qemu*, permite emular procesadores *arm* y con ello también se puede dar soporte a esta arquitectura, cosa que con *VirtualBox* no se puede hacer. Cumple también con el último requisito, que es que se pueda administrar por comandos para poder así construir scripts que automaticen todo lo posible el proceso.

2.3.2. Arquitectura mínima

Tras el análisis se procede a la construcción de la arquitectura mínima sobre la que se construirá el sistema.

User Stories

- Como usuario quiero una máquina virtual que emule una arquitectura *i386* para poder construir paquetes para esta arquitectura.
- Como usuario quiero una máquina virtual que emule una arquitectura *amd64* para poder construir paquetes para esta arquitectura.
- Implementar un «Hello World» distribuido con el que poder probar que todo está bien.
- Ejecutar el *Hello World* en el sistema distribuido

2.3.3. Implementación y ejecución del «Hello World»

Tras instalar ambas máquinas virtuales con la API libvirt se procede a la ejecución del *Hello World* para demostrar que la arquitectura mínima funciona. Para ello se ha elegido el lenguaje de programación *Python* que permite crear prototipos de forma rápida. El cliente y el servidor del *HelloWorld* se muestran a continuación.

```
import sys
import Ice
Ice.loadSlice('Printer.ice')
import Example

class client (Ice.Application):
  def run(self, argv):
      proxy = self.communicator().stringToProxy(argv[1])
      printer = Example.PrinterPrx.checkedCast(proxy)

      if not printer:
          raise RuntimeError('Invalid proxy')

      printer.write('Hello World!')

      return 0

sys.exit(client().main(sys.argv))
```

Listado 2.2: Cliente «HelloWorld» distribuido en Python

```
import sys
import Ice
Ice.loadSlice('Printer.ice')
import Example

class PrinterI(Example.Printer):
  def write(self, message, current=None):
      print message
      sys.stdout.flush()

class Server(Ice.Application):
  def run(self, argv):
```

```
        broker = self.communicator()
        servant = PrinterI()

        adapter = broker.createObjectAdapter("PrinterAdapter")
        proxy = adapter.add(servant, broker.stringToIdentity("printer1
            "))

        print proxy

        adapter.activate()
        self.shutdownOnInterrupt()
        broker.waitForShutdown()

        return 0

server = Server()
sys.exit(server.main(sys.argv))
```

Listado 2.3: Servidor «HelloWorld» distribuido en Python

Se ha probado con éxito y todo funciona, por lo tanto, se puede continuar con la siguiente iteración.

2.4. Iteracion 1

En esta iteración se propone avanzar en el desarrollo de un nodo del sistema. Tras tener el sistema mínimo instalado que consta de una máquina virtual de cada arquitectura, se ha comprobado que el proyecto puede tomar dos vertientes y que están basadas en el uso o no de *pbuilder* [Uek07], un sistema de construcción automática de paquetes Debian donde el objetivo, es poder generar paquetes Debian sin ensuciar el sistema con las dependencias de los paquetes que se están construyendo. El dilema es, ¿usar pbuilder para construir los paquetes o aprovechar las máquinas virtuales que ya son un entorno limpio?

2.4.1. Usar o no pbuilder

Uno de los requisitos del sistema es que aprovechando los computadores disponibles en el laboratorio ARCO se construirían los paquetes, pero surge un problema al que aún no se le ha dado solución, ¿qué pasaría si un computador se apagaba en medio de la construcción de un paquete?. Este caso sirve de ejemplo ilustrativo para poder plantear la resolución del camino a tomar a partir de ahora, porque, ¿qué pasaría si se necesita construir un paquete muy grande y en mitad de la construcción se apaga el computador? Una de las opciones sería empezar al día siguiente con un computador de la misma arquitectura elegido por el sistema, pero, ¿qué sucedería si la construcción durara más tiempo del que los computadores estén encendidos? Se tendría que resolver eso de algún modo. Una posible solución sería, no apagar ese computador hasta que termine la construcción del paquete, pero entonces se estaría alterando la forma en la que funciona el laboratorio. El edificio donde se encuentre el

entorno de trabajo puede apagar la electricidad por multitud de razones y esto ya supone un problema de por sí, y el cual hay que tener en cuenta.

Con el uso de máquinas virtuales se abren nuevas opciones y nuevas formas de resolver el problema, incluso dando soluciones a otros nuevos problemas. Las "snapshots"son una operación que se puede hacer sobre las máquinas virtuales y que puedes resolver este problema. Una “snapshot” o, traducido al español, una imagen, es la operación por la cual se toma un estado en un instante t cualquiera de la máquina virtual y se congela, pudiendo en cualquier momento volver a ese estado de “congelación” y, empezar o continuar desde ese estado la tarea que se estuviese desarrollando la proxima vez que se encienda el ordenador o la máquina virtual. Esta operación también tiene la ventaja de que se puede revertir y volver a un estado anterior cuando se requiera.

La idea de esto, es que ya que se necesita un entorno limpio para construir los paquetes, aprovechar que al generar el sistema base en la iteración anterior, se tienen máquinas virtuales mínimas muy similares a lo que *pbuilder* para construir los paquetes. Tal y como se describe parece que usar las máquinas virtuales, a priori, no tiene ninguna desventaja, pero si que tiene. Por ejemplo, se necesita almacenamiento en disco extra cada vez que se genera una *snapshot* de la máquina virtual en cuestión, si ocupa 3GB se necesitarían 6GB (3 GB de la snapshot y 3GB de la original). Ahora la pregunta es, ¿es un problema el almacenamiento? La respuesta es que no, ya que hoy en día la relación Euro-Giga es de más o menos 0.039 en discos duros no SSD, y en el futuro bajará cada vez más. De hecho, el computador usado para realizar este proyecto (Dell XPS 420) salió a la venta en el año 2007 y ya tenía un disco duro de 300GB, de los cuales a día de hoy para trabajar no se utilizan ni la mitad.

Otra de las ventajas que tiene utilizar este sistema es que se pueden generar paquetes por medio de máquinas virtuales utilizando la emulación de algunos procesadores y que permitiría construir paquetes para más arquitecturas de las disponibles. A priori no sería necesario, pero de esta forma se generaliza la solución al problema dando soporte a más arquitecturas. Esto también puede resultar ventajoso porque la máquina emulada, puede resultar en ocasiones más rápida que la propia máquina real. Se puede utilizar de ejemplo el famoso *Raspberry Pi*, en el cual la construcción de un paquete tardaría mucho más que en una máquina virtual emulando ese procesador en un PC de escritorio.

Pbuilder es un sistema ya probado, y que funciona, pero tiene la desventaja de que no se pueden construir paquetes para otras arquitecturas diferentes de las del computador que tengamos en posesión, por ejemplo, dado un computador con *amd64*, solamente podría generar paquetes para esa arquitectura o, como mucho *i386*.

Inicialmente se construyó un entorno de pruebas mínimo con una máquina virtual de cada arquitectura, y ese entorno de pruebas ha pasado a ser el sistema para seguir adelante ya que se ha observado una ventaja importante: el hecho de poder congelar la construcción de

paquetes.

Ventajas y desventajas de pbuilder y máquinas virtuales

Un resumen de las ventajas e inconvenientes de cada opción son:

- Pbuilder
  - Ventajas:
    - Ya está hecho, y funciona.
    - No hace falta crear nada adicional salvo configurarlo.
    - Menor consumo que las máquinas virtuales en cuanto a memoria principal y secundaria.
  - Inconvenientes:
    - Está acotado, pues solamente se pueden construir paquetes para la arquitectura del computador que lo ejecuta.
    - Si un computador se apaga se tiene que empezar la construcción del paquete la próxima vez que se encienda el computador por lo que se presenta el riesgo de que no se pueda construir el paquete.
- Máquinas virtuales
  - Ventajas:
    - Con una instalación mínima se tiene un entorno muy similar al que se obtiene con *pbuilder*
    - Se puede parar la construcción del paquete y continuarla cuando se desee gracias a las *snapshots*.
    - Da soporte a más arquitecturas gracias a la emulación a través de *qemu* de diferentes arquitecturas de procesadores.
  - Inconvenientes:
    - Mayor consumo de memoria principal y de memoria secundaria.
    - Tarda más tiempo en la construcción de paquetes.

Por todo ello, y viendo que el tiempo que tarde en construir los paquetes no es un problema, y que tampoco lo es el espacio en disco duro, se elige las máquinas virtuales y se cambia el entorno de pruebas por el entorno final donde se desarrollará el proyecto.

2.4.2. User Stories

Una vez descrito las dos propuestas y escogida una de ellas se procede a la implementación de la iteración.

Se parte de un entorno donde se tienen dos máquinas virtuales, una con arquitectura *amd64* y otra con arquitectura *i386* conectadas al computador que las ejecutas. A su vez ese compu-

tador está conectado a la red del entorno de trabajo, donde hay otros computadores que también son un nodo del sistema distribuido tal y como se muestra en la figura 2.4

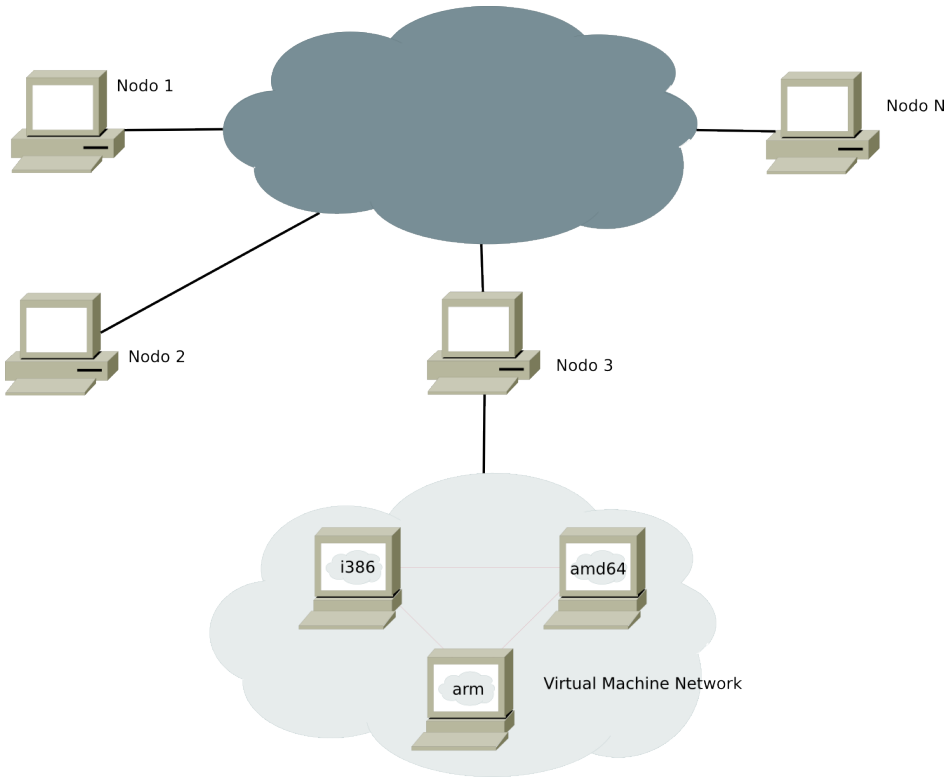


Figura 2.4: Arquitectura de un Nodo y del sistema

La implementación de esta iteración consiste en la construcción de un paquete para cada arquitectura, por lo tanto las *User Stories* son:

- Dado un paquete del repositorio quiero construir el paquete para que esté disponible para las dos arquitecturas ejecutando un comando de la siguiente forma: **comando <nombre_de_paquete>**
- Como usuario, quiero poder salvar el estado de la máquina virtual por si algo va mal que me pueda quedar como estaba. Crear, destruir y revertir un estado.
- Quiero que mi máquina virtual se actualice, y si todo va bien usar esa máquina virtual en el futuro para tenerla actualizada en todo momento.
- Como usuario quiero estar informado del proceso, que me muestre mensajes simples como [OK] cuando se finalicen tareas.
- Generar una firma para poder firmar los paquetes que se construyan en las máquinas virtuales.
- Como usuario quiero que el paquete se firme para poder tener un paquete construido y firmado.

Para realizar estas "user stories" se utilizará el paquete *gentoo*, un paquete muy sencillo que aparece en los ejemplos de la *Debian maint-guide* y que se utilizará para realizar las pruebas.

Como se verá más adelante se requerirá en algunos comandos el uso de privilegios de administrador, por lo que las máquinas virtuales se han configurado para usar `sudo` sin que pida la contraseña. Para conectar cada una de las máquinas virtuales se hace a través del protocolo *SSH* y *sshpass* para pasar el usuario y la contraseña como argumentos a través de la siguiente orden:

```
sshpass -p 'arco' ssh -o StrictHostKeyChecking=no user@host <nombre\_de\_paquete>
```

La orden `sshpass -p 'arco'` indica que la contraseña será 'arco'. El argumento `-o` de *SSH* sirve para indicarle una opción, que en este caso es `StrictHostKeyChecking=no`, y que sirve para controlar los logins en host donde no se conocen las claves o estas han cambiado.

Para la actualización del sistema se requiere de los típicos comandos

```
sudo apt-get update && sudo apt-get upgrade
```

Ahora que el sistema está actualizado es el momento de crear la "snapshot" con

```
sudo virsh snapshot-delete <nombre_dominio> <nombre_snapshot>
```

Cuando sistema está actualizado y con la "snapshot" hecha, una de las primeras cosas que hay que hacer para construir un paquete Debian que ya está en el repositorio es ejecutar el comando

```
apt-get source <nombre_de_paquete>
```

para conseguir las fuentes. Esto creará los archivos y directorios necesarios para la construcción del paquete, por lo que hay que estar dentro del directorio con nombre `nombre_del_paquete-version`

```
cd nombre\_de\_paquete-*
```

Un paquete Debian necesita cumplir unas dependencias de construcción, estas dependencias de construcción están indicadas en el archivo `debian/rules` en la línea *Build-Depends*. Los paquetes que figuran en esa línea necesitan estar instalados en el sistema para construir el paquete, ya que se depende de ellos para su construcción. En el caso del paquete *gentoo* las dependencias son: `debhelper (>= 9), dh-autoreconf, libgtk2.0-dev,`

`libglib2.0-dev`. Para solventar este problema *apt-get* tiene una opcion "build-dep" que instala las dependencias necesarias para la construcción del paquete. Este comando hay que ejecutarlo como usuario `root`, por lo tanto se usa la orden `sudo` que permite ejecutar comandos como administrador.

```
sudo apt-get build-dep <nombre_del_paquete>
```

Como último paso dentro de la máquina virtual hay que ejecutar `debuild` para construir el paquete

```
debuild -us -uc
```

Para terminar el proceso hay que revertir la "snapshot" para dejar la máquina virtual actualizada y preparada para realizar la siguiente construcción.

```
sudo virsh snapshot-revert <nombre_dominio> <nombre_snapshot>
```

Para automatizar todo este proceso se ha realizado un script que se ejecutará en cada uno de los nodos cuando se requiera para construir los paquetes. Este script se encuentra en `src/`.

2.4.3. Firma de paquetes: The GNU Privacy Guard

El GnuPG es el la implementación libre del estándar OpenPGP definido en el **RFC4880**. *GnuPG* permite encriptar y firmar tus datos y comunicaciones, cuenta con un sistema de gestión de claves, así como módulos de acceso para todo tipo de directorios de claves. Gnupg es conocido por *GPG*, y es una herramienta para la línea de comandos para integrarla fácilmente con el resto de aplicaciones.

El funcionamiento consiste en que cada usuario tiene una clave privada y una clave pública. La clave privada es la que se utiliza para desencriptar aquello que te envían encriptado con la clave pública. La clave privada solamente la debe conocer el propietario, si alguien más la conociese, podría desencriptar lo que te envían y sería un problema de seguridad.

Lo que se pretende conseguir al firmar los paquetes es que se tenga la seguridad de que esos paquetes no están dañados o alterados cuando pasan por las numerosas fases de construcción. De este modo, si un paquete no tuviese firma o tuviese una de la que no se tiene su clave, se considera como no confiable.

APT utiliza *GPG* para validar los paquetes .deb descargados y asegurarse de que no han sido alterados.

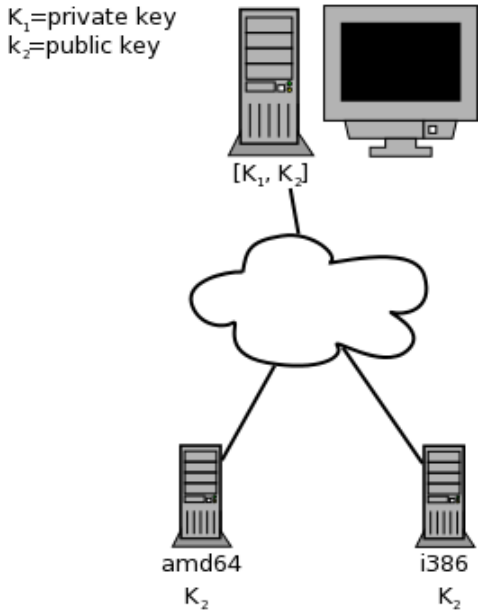


Figura 2.5: Firma de claves gpg

Generación de un nuevo par de claves

Una vez explicado en que consiste el método de firmado, se procede a la creación de claves en la parte del computador que tiene las máquinas virtuales.

```
gpg --gen-key
```

Y se elige la primera opción `RSA and RSA (default)`, cuando pregunte por el tamaño, éste se deja por defecto a 2048, en cuanto a la duración también se deja por defecto a 0. Se siguen los pasos poniendo lo que pida el programa (Nombre, e-mail y comentario).

Una vez se generan las claves, es bueno generar el certificado de revocación, que en el caso de perder la clave o de haber comprometido su seguridad, el certificado de revocación se haría público para notificar al resto de usuarios que la clave pública no debe ser usada nunca más.

```
gpg --output DD123456.asc --gen-revoke 0xDD123456
```

Ahora que se tienen las claves es hora de exportarla

```
gpg --output ARCO.gpg --export e-mail
```

Tras esto es necesario copiar la clave a las máquinas virtuales, y una vez que las tengamos allí se importará en cada una de ellas:

```
gpg --import ARCO.gpg
```

Cuando se contruye un paquete y se quiere firmar, basta con utilizar la orden `debuild`, pero esto solamente funciona si el nombre y los datos coinciden con los del mantenedor del paquete. Como en este caso el mantenedor no va a construir el paquete, si no que va a ser una máquina que no tiene nada que ver con el mantenedor, se debe contruir el paquete utilizando el argumento `-k` seguido de la clave. Por lo tanto el comando quedaría `debuild -kE65B0539`

2.5. Iteracion 2

Tras lo visto en las secciones anteriores 2.3 y 2.4 donde se construía la arquitectura base y se probaba la construcción de dos paquetes, es momento de añadir más funcionalidad al nodo que hay de prueba. Para dar el siguiente paso en esta iteración se proponen dos cosas. La primera es enviar al nodo más de un paquete desde otros dos nodos distintos. Y la segunda es crear un sistema de "logz enviara un nodo dos paquetes desde nodos distintos para que se construyan

2.5.1. Prueba para dos paquetes

Ya que se ha probado que es sistema funciona para un nodo lo ideal sería que utilizando el middleware ICE se enviase más de un paquete y que se construyesen sin problemas. Con ello se demostraría que se puede construir más de un paquete, y lo más importante, que estas "peticiones de construcción"se pueden hacer desde otros computadores distintos al que está ejecutando el nodo.

2.5.2. Logger

Uno de los requisitos enumerados al principio es que se debe tener un "log"donde se pueda consultar que es lo que ha pasado o cuales han sido las últimas acciones realizadas con el fin de informar de algún error, de algún acierto o seguir las acciones que se van realizando. En términos marinos existe el cuaderno de bitácora, que es un libro donde se apuntan los datos de lo acontecido para poder consultar todo lo que ha pasado durante un periodo de tiempo. En informática se utiliza la palabra "log", el cual es un registro oficial de eventos que se hace durante un periodo de tiempo en particular. La información que se suele almacenar en los "logsresponde a preguntas del tipo *quién, qué, dónde, por qué y cuando*. La idea es que de una forma rápida el usuario pueda saber **qué** ha fallado, **quién** hizo esa operación, **dónde** se realizó, **por qué** se produjo el fallo y **cuando** sucedió. Ojo, un fichero de este tipo también ayuda a comprobar que algo funciona correctamente.

En Sistemas Operativos como GNU/Linux existen unos ficheros con extensión `.log` donde se almacena este tipo de información. Normalmente cada aplicación tiene un fichero de

"logza que sería inmanejable tener uno gigantesco para todo el sistema. En este tipo de ficheros la información está organizada respondiendo a esas preguntas formuladas en el párrafo anterior. Por ejemplo, tómese el archivo `/var/user.log` para observar lo que sucede:

```
Apr 26 09:53:49 baran NetworkManager[3077]: <info>   prefix 24 (255.255.255.0)
Apr 26 16:03:55 baran dhclient: DHCPACK from 161.67.106.1
Apr 26 16:03:56 baran NetworkManager[3077]: <info> Clearing nscd hosts cache.
Apr 26 16:58:20 baran sensord: Chip: coretemp-isa-0000
Apr 26 16:58:20 baran sensord: Adapter: ISA adapter
Apr 26 16:58:20 baran sensord:   Core 0: 32.0 C
Apr 26 16:58:20 baran sensord:   Core 1: 30.0 C
Apr 26 16:58:20 baran sensord:   Core 2: 26.0 C
Apr 26 16:58:20 baran sensord:   Core 3: 27.0 C
```

Como se puede observar a primera vista, se muestra una hora, un día del año y un mes, en este caso es el 26 de Abril en varias horas a lo largo del día. `baran` es el nombre del computador, y lo que hay a la derecha es la información que se ofrece. En algunas ocasiones, cuando se quiere resaltar algo suele utilizarse etiquetas como `INFO` para referirse a información `WARNING` para avisar de algo importante y `ERROR` para mostrar un mensaje de error.

Todo esto viene muy bien para hacerse una idea de lo que el fichero de "log"debería tener en el sistema distribuido, ya que podría asemejarse al mostrado arriba por ser una alternativa que es conocida, está probada y se usa en los Sistemas Operativos y en muchas más aplicaciones como los SGBD. Una vez conocido como son los ficheros actuales, es bueno preguntarse cómo debería ser el de este proyecto. Lo que lleva a pensar que interesa que de información acerca de los errores, si ha salido todo bien y por ejemplo que nodo construye cada paquete. Todo ello acompañado de la fecha y hora de cada evento. Se detallará todo esto más adelante cuando se elaboren las üser stories"porque ahora hay que estudiar como se puede hacer el "logger".

Se plantean varias alternativas para dar solución al problema de implementar el "Logger". La primera alternativa es leer la referencia de los lenguajes de programación que se van a utilizar para ver si se dispone de algún "loggerza implementado. *C++* al contrario que *Python*, no tiene un "loggerïmplementado, lo que nos lleva a la segunda opción.

La segunda es implementar un sistema desde cero programando lo que se necesite. Esta opción, aunque válida requiere realizar un trabajo extra. La tercera opción es utilizar algo externo al lenguaje de programación, en este caso el middleware ICE tiene un "logger"que se puede utilizar con el fin de ofrecer los mensajes deseados, además es personalizable, por lo que si se requiere añadir nuevas etiquetas o alguna otra característica se puede añadir sin problemas. La interface Logger de ICE es como se aprecia en el siguiente fragmento de código:

```
module Ice {
```

```
    local interface Logger {
        void print(string message);
        void trace(string category, string message);
        void warning(string message);
        void error(string message);
        Logger cloneWithPrefix(string prefix);
    };
};
```

Listado 2.4: Interfaz Logger en Slice

Como se observa este fichero `slice` produce mensajes de traza, avisos o mensajes de error, así que este puede ser un buen punto de partida.

ANEXOS

Manual de instalación de un Nodo

A.1. Introducción

El presente documento describe los pasos necesarios para la instalación de un nodo en el sistema **distdeb**, un sistema distribuido para construcción de paquetes Debian.

Tanto el presente documento como el proyecto en sí está en constante desarrollo por lo tanto, el lector debe saber que puede haber cambios tanto en el manual como el código fuente.

La instalación de un Nodo dentro del sistema distdeb, se hará en el futuro mediante un paquete Debian.

A.1.1. Estructura del documento

Este manual se divide en 4 secciones:

- ¿Qué es Node-Builder?
- Requisitos
- Instalación
- Configuración

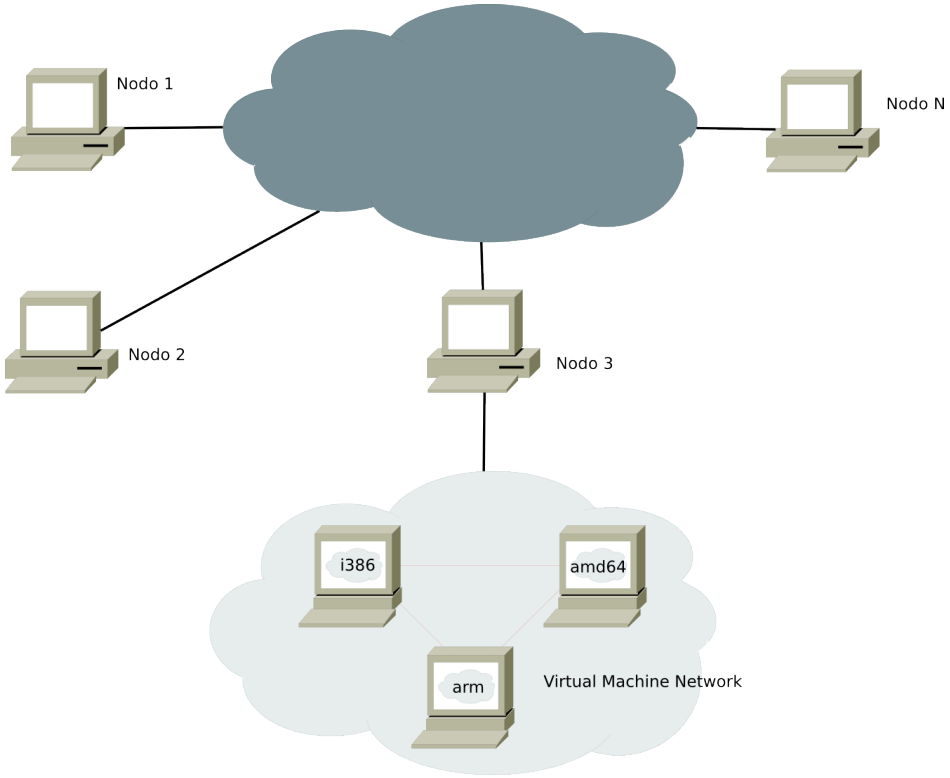
Cada uno de los capítulos describe las acciones necesarias y explicando con detalle todos los pasos a seguir para configurar un nodo que contiene a dos máquinas virtuales de arquitecturas *i386* y *amd64*

A.2. ¿Qué es Node-Builder?

Node-Builder es la parte del sistema distribuido para contruir paquetes debian que se encarga de realizar la recontrucción de paquetes. Ayuda a distdeb y a los desarrolladores a generar los paquetes para las distintas arquitecturas para las cuales se quieren contruir los paquetes ya existentes y una vez construidos, subirlos al repositorio (Esta opción se implementará en posteriores versiones de desarrollo).

Node-Builder está formado por un computador que a su vez aloja una o varias máquinas

virtuales donde se realiza la construcción de paquetes, comprobación de paquetes una vez construido, congelación de estado de máquina virtual y muchas más.



A.2.1. ¿Por qué Node Builder?

Mientras se desarrollaban las primeras iteraciones de **distdeb** y tras realizar algunas pruebas, se llegó a la conclusión de que era más ventajoso utilizar máquinas virtuales en lugar de *pbuilder*. Con las máquinas virtuales se tienen sistemas limpios donde realizar la construcción.

Al utilizar máquinas virtuales se tiene una opción que puede resolver el problema que se tiene cuando un paquete tarda mucho tiempo en construirse y el computador se tiene que apagar. Esto se soluciona con la utilización de "snapshots". Una "snapshot" o, traducido al español, una imagen, es la operación por la cual se toma un estado en un instante *t* cualquiera de la máquina virtual y se congela, pudiendo en cualquier momento ir a ese estado de "congelación" y, empezar o continuar desde ese estado la tarea que se estuviese desarrollando la proxima vez que se encienda el ordenador o la máquina virtual. Esta operación también tiene la ventaja de que se puede revertir y volver a un estado anterior cuando se requiera.

Una de las ventajas que tiene utilizar este sistema es que se pueden generar paquetes utilizando la emulación de algunos procesadores y que permitiría construir paquetes para más arquitecturas que si se usase pbuilder. A priori no sería necesario, pero de esta forma se generaliza la solución al problema dando soporte a más arquitecturas. Otra ventaja es que la máquina emulada, puede resultar en ocasiones más rápida que la propia máquina

real. Se puede utilizar de ejemplo el famoso *Raspberry Pi*, en el cual la construcción de un paquete tardaría mucho más que en una máquina virtual emulando ese procesador en un PC de escritorio.

A.3. Requisitos

A.3.1. Requisitos de hardware

- Conexión a internet de banda ancha.
- Al menos 2GB de RAM.

A.3.2. Paquetes necesarios

Para la instalación y configuración se necesitan los siguientes paquetes:

- libvirt
- virtinst
- virt-viewer
- virt-manager

Todos ellos hacen referencia a la API libvirt por la cual se pueden gestionar máquinas virtuales con comandos y de forma casi automática.

Es una buena idea que se añada el usuario con el que se van a ejecutar los comandos al grupo libvirt, para no tener que recurrir a utilizar privilegios de administrador.

```
# adduser usuario libvirt
# adduser usuarios kvm
```

A.3.3. Requisitos de ejecución

Dada la naturaleza de libvirt, es necesario añadir lo siguiente a tu .bashrc para no tener que ejecutarlo continuamente cada vez que se inicia el computador

```
export LIBVIRT_DEFAULT_URI=qemu:///system
```

Se necesita crear una red[1] para interactuar con las máquinas virtuales y un directorio[2] donde se alojarán las máquinas virtuales. Este directorio debe existir en el sistema donde se realice la instalación. Además deberás introducir la ruta **absoluta** en el archivo src/xml/pool-distdeb.xml entre las etiquetas `<path>`, tal y como se muestra en el siguiente ejemplo:

[1] Formato XML para redes con libvirt
[2] Formato XML para almacenamiento con libvirt

```
<pool type="dir">
  <name>distdeb</name>
  <target>
    <path>/home/usuario/virtualMachines</path>
  </target>
</pool>
```

Ejecuta ahora el script *create_stuff.sh* para que se configuren tanto la red como el directorio donde se instalarán las máquinas virtuales.

A.3.4. El archivo preseed

Preseed[3] ofrece un mecanismo para responder a las preguntas que se realizan durante la instalación sin que tener que introducir las respuestas manualmente mientras se ejecuta la instalación. Gracias a esto es posible automatizar completamente la mayoría de instalaciones e incluso algunas características no están disponibles durante la instalación normal, como puede ser la instalación de algunos paquetes o la configuración de sudo por defecto.

El fichero pressed.cfg ya ha sido configurado de forma óptima para la instalación del sistema.

A.4. Instalación

A.4.1. Instalación de máquinas virtuales

Para proceder a instalar las máquinas virtuales se ha creado un script que automatiza esta tarea, está en src/install_vm.sh. Este script genera automáticamente una instalación mínima del sistema Operativo Debian GNU/Linux rama testing de arquitectura i386 y amd64.

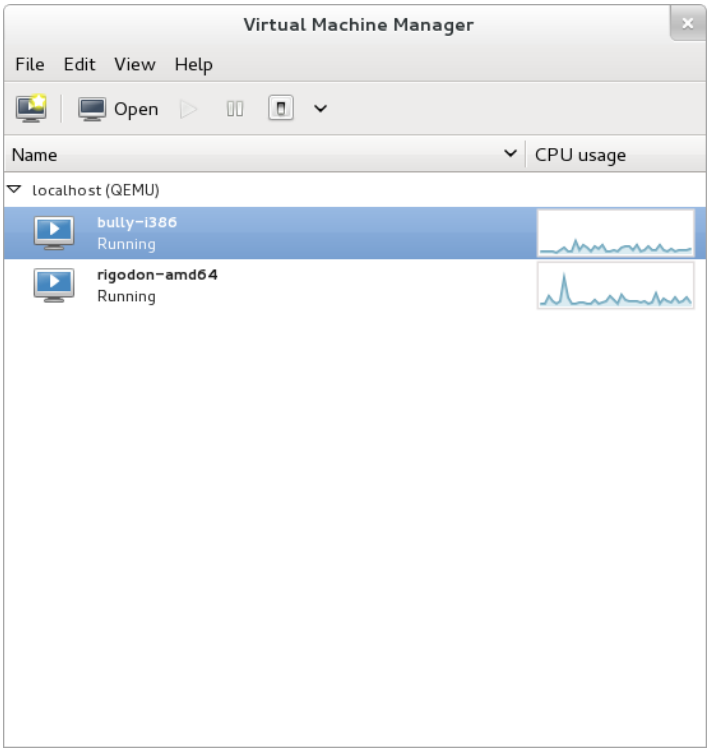
Si solamente se desa instalar **una** de las dos arquitecturas (i386 o amd64, basta con pasarla como parámetro al script y automáticamente la generará:

```
sh install_vm.sh <arquitectura>
```

¡Importante! Si se detectase algún error durante la instalación es recomendable ejecutar la instalación por separado de ambas máquinas virtuales.

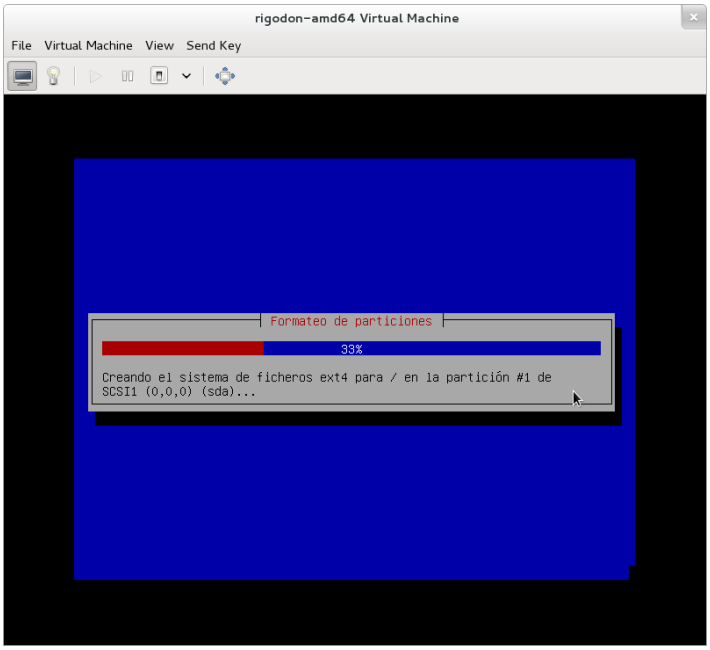
Al ejecutar el script, se realiza una descarga de internet y luego aparece la ventana del virt-manager donde se puede seguir el proceso de la instalación. Esta visión no se podrá ver mñas que unos gráficos con la carga de trabajo que tiene cada una de las máquinas virtuales, por lo que no aporta nada a simple vista.

[3] Más información en su página web



Cabe la posibilidad de ver con más detalle el proceso, seleccionando una máquina virtual y pinchando sobre el botón “open” del menú de virt-manager.

Lo que se verá es la típica instalación de debian pero donde el usuario final no tiene que hacer nada porque todo ya se ha configurado automaticamente.



Cuando todo el proceso de instalar una o ambas máquinas virtuales ha finalizado, las máquinas se apagan y se puede continuar con su configuración.

A.5. Configuración

Esta es la sección referida a la configuración de las máquinas virtuales. Si solamente se ha instalado una máquina virtual, no tiene por qué utilizar *clusterssh*, pase directamente al siguiente punto, configuración de *sudo* A.5.2.

A.5.1. Configuración de máquinas virtuales

Por defecto la mayoría de los paquetes están listos e instalados en las máquinas virtuales, incluso se tiene creado un usuario por defecto que puede utilizar el comando `sudo`. Este usuario tiene por nombre `arco` y por contraseña `arco`. Además, se necesita instalar algún paquete más en el computador anfitrión, ya que la conexión entre el anfitrión y las máquinas virtuales se realizad por *SSH*.

```
# aptitude install sshpass clusterssh
```

El paquete *clusterssh* permite ejecutar el mismo comando en n máquinas conectadas por *SSH*, viene muy bien para no tener que repetir tareas en ambas máquinas virtuales. Con el uso de *cssh* se hará todo de una sola vez. Se verá más adelante su uso en la subsección A.5.2

A.5.2. Configuración de sudo

Si solamente tiene una máquina virtual no utilice *clusterssh*, puede utilizar *ssh* o meterse normalmente en la máquina virtual para realizar la configuración.

Una vez que *clusterssh* está instalado en el sistema anfitrión, lo que hay que hacer ahora es configurar las máquinas virtuales para que no soliciten la contraseña cada vez que se utilice la orden `sudo`. Por lo tanto, mediante el siguiente comando se procede a conectarse a las dos máquinas virtuales instaladas previamente.

```
cssh arco@IP1 arco@IP2
```

Donde `IP1` y `IP2` son las *IP* correspondientes a cada una de las máquinas virtuales, por defecto 192.168.122.11 y 192.168.122.12.

```
sudo nano /etc/sudoers
```

Y se añade al archivo para que no pida la contraseña cada que vez que se usa el comando `sudo` la línea **%arco ALL=NOPASSWD:ALL**:

```
# See the man page for details on how to write a sudoers file.
#
Defaults        env_reset
Defaults        mail_badpass
```

```
Defaults        secure_path="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/s

# Host alias specification

# User alias specification

# Cmnd alias specification

# User privilege specification
root    ALL=(ALL:ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
%arco   ALL=NOPASSWD:ALL
# See sudoers(5) for more information on "#include" directives:

#includedir /etc/sudoers.d
```

A.5.3. Configuración de repositorios

Por defecto las instalación de Debian marca unos repos con el nombre en clave de la distribución testing (actualmente *wheezy* y proximamente *Jessie*).

Se procede a cambiar los repositorios para que apunten a "testing" y evitar así problemas con las nomenclaturas, de esta forma, las máquinas virtuales siempre serán rama "testing" de Debian.

```
sudo nano /etc/apt/sources.list
```

Y se sustituye *wheezy* por testing, quedando el archivo de la siguiente forma:

```
deb http://ftp.es.debian.org/debian/ testing main
deb-src http://ftp.es.debian.org/debian/ testing main

deb http://security.debian.org/ testing/updates main
deb-src http://security.debian.org/ testing/updates main
```

A.5.4. Configuración de Grub

Este paso es opcional y sirve para desactivar el tiempo de espera que tiene Grub para iniciar las máquinas virtuales, ya que "libvirt" no sabe a priori cuando una máquina virtual

está lista para realizar las conexiones ssh.

Por ello, se edita el fichero */boot/grub/grub.cfg* cambiando el `tiemout` de 5 a 0.

Una vez se tiene todo esto, ya no es necesario realizar ninguna tarea más en la máquina virtual, ya que serán los scripts los que se encarguen de realizar todas las tareas.

GNU Free Documentation License

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "publisher" means any person or entity that distributes copies of the Document to the public.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

- A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
- B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
- C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
- D. Preserve all the copyright notices of the Document.
- E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
- F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
- G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
- H. Include an unaltered copy of this License.

- I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
- J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
- K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
- L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
- M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
- N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
- O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version. 5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

5. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

6. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

7. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English

version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

8. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

9. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License “or any later version” applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

10. RELICENSING

“Massive Multiauthor Collaboration Site” (or “MMC Site”) means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A “Massive Multiauthor Collaboration” (or “MMC”) contained in the site means any set of copyrightable works thus published on the MMC site.

“CC-BY-SA” means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

“Incorporate” means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is “eligible for relicensing” if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

Bibliografía

[Bel13] Fabrice Bellard. Qemu Emulator User Documentation, 2013.

[Cor13] Oracle Corporation. Oracle VM Virtual Box User Manual, 2013.

[HS10] M. Henning y M. Spruiell. *Distributed Programming with Ice (Version 3.4.0)*. ZeroC Inc, 2010.

[PAW13] Jussi Ronkainen Pekka Abrahamsson, Outi Salo y Juhani Warsta. Agile Software Development Methods: Review and analysis, 2013.

[RH11] Inc. Red Hat. The virtualization API. `http://libvirt.org/index.html`, Septiembre 2011.

[Uek07] Junichi Uekawa. Pbuilder User's Manual, May 27, 2007.

[VMw13] Inc. VMware. VMware Debian 6 Documentation, 2013.

Este documento fue editado y tipografiado con LaTeX
empleando la clase **arco-pfc** que se puede encontrar en:
`https://bitbucket.org/arco_group/arco-pfc`