
CygnusCloud: provisión de puestos de laboratorio virtuales bajo demanda



Proyecto de Sistemas Informáticos
Curso 2012/2013

Facultad de Informática
Universidad Complutense de Madrid

Luis Barrios Hernández

Adrián Fernández Hernández

Samuel Guayerbas Martín

Dirigido por
José Luis Vázquez-Poletti
José Antonio Martín Hernández

Nosotros, Luis Barrios Hernández, Adrián Fernández Hernández y Samuel Guayerbas Martín, autores del presente documento y del proyecto *CygnusCloud: provisión de puestos de laboratorio virtuales bajo demanda*, autorizamos a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

Luis Barrios Hernández
DNI: 53416459-V

Adrián Fernández Hernández
DNI: 05337637-G

Samuel Guayerbas Martín
DNI: 04217069-L

Agradecimientos

Resumen

Abstract

Índice general

Índice general	I
Índice de figuras	VII
Índice de cuadros	IX
1 Descripción del problema	1
1.1. Introducción	1
1.2. El modelo de gestión de los laboratorios de la Facultad de Informática de la UCM	1
1.2.1. Introducción	1
1.2.2. Requisitos docentes de los laboratorios	2
1.2.3. Servicios prestados a los alumnos	2
1.2.4. Servicios prestados a los profesores	3
1.2.5. Otros servicios	3
1.3. Inconvenientes del modelo presentado	3
1.3.1. Desaprovechamiento de recursos	3
1.3.1.1. Aulas de informática de otros centros	3
1.3.1.2. Personal encargado de las aulas de informática	4
1.3.1.3. Energía consumida	4
1.3.1.4. Licencias de pago	5
1.3.2. Limitaciones del sistema de almacenamiento	5
1.3.3. Sobrecoste de los PCs de las aulas de informática	5
1.3.4. Excesiva rigidez de la configuración de los equipos	7
2 Cloud Computing	9
2.1. Introducción	9
2.2. Definición	9
2.3. Historia	10
2.3.1. Precedentes	10
2.3.2. Aparición de las ideas básicas del <i>Cloud Computing</i>	11
2.3.3. Primeros años de los computadores en la empresa	11
2.3.4. Aparición del modelo <i>peer to peer</i>	11
2.3.5. Desarrollo del <i>Cloud Computing</i>	11
2.4. Características	12
2.5. Modelos de servicio	13
2.5.1. Infraestructura como servicio	13
2.5.2. Plataforma como servicio	14
2.5.3. Software como servicio	14
2.6. Otros modelos de servicio especiales	14
2.6.1. Escritorio como servicio	14
2.6.2. Entorno de pruebas como servicio	15
2.6.3. Identidad como servicio (IDaaS)	15
2.7. Tipos	15
2.7.1. Cloud público	15
2.7.2. Cloud privado	16

2.7.3. Cloud híbrido	16
2.7.4. Cloud comunitario	17
2.8. Ventajas e inconvenientes	17
2.8.1. Ventajas	17
2.8.2. Desventajas	18
2.9. El <i>Cloud Computing</i> como un estándar abierto	18
3 Virtualización	21
3.1. Introducción	21
3.2. Historia	21
3.3. Distintos esquemas de virtualización	21
3.3.1. Virtualización de hardware	22
3.3.2. Virtualización de software	22
3.3.3. Paravirtualización	22
3.3.4. Virtualización completa	23
3.4. Software de virtualización	23
3.5. Xen	23
3.5.1. Dom0, DomU	23
3.5.2. Interrupciones	24
3.5.3. CPU	24
3.5.4. Memoria	24
3.5.5. Dispositivos de entrada/salida	24
3.5.6. Red	24
3.5.7. Dispositivos de bloque	24
3.6. KVM	25
3.6.1. Breve historia de KVM	25
3.6.2. Características de KVM	25
3.6.3. Requisitos Hardware	26
3.6.4. El paquete KVM	26
3.6.5. Ventajas de KVM	27
3.6.6. Desventajas de KVM	27
3.6.7. Limitaciones de KVM	28
3.6.8. Red en KVM	28
3.7. Xen frente a KVM	28
3.7.1. Integración en el kernel	29
3.7.2. Rendimiento	29
3.7.3. Soporte técnico	29
3.7.4. Tipo de virtualización	29
4 Arquitectura del sistema	31
4.1. Introducción	31
4.1.1. Visión general	31
4.1.2. Sobre los diagramas UML de este documento	32
4.2. Objetivos de la arquitectura y restricciones	32
4.2.1. Objetivos	32
4.2.2. Restricciones	32
4.3. Decisiones de diseño	33
4.3.1. Uso de una aplicación web para interactuar con <i>CygnusCloud</i>	33
4.3.2. Descomposición del sistema <i>CygnusCloud</i> en cuatro subsistemas	34
4.3.2.1. Los servidores de máquinas virtuales	34
4.3.2.2. El repositorio de imágenes	34
4.3.2.3. El servidor de <i>cluster</i>	35
4.3.2.4. El servidor web	35
4.3.3. Implementación de una infraestructura <i>ad-hoc</i>	36
4.3.4. Configuración de las redes virtuales en modo NAT	36
4.3.5. Uso del protocolo de escritorio remoto VNC	37

4.3.6. Uso del gestor de bases de datos MariaDB	37
4.3.7. Uso del hipervisor KVM	38
4.3.8. Uso del servidor VNC integrado en KVM	38
4.3.9. Uso del cliente VNC noVNC	39
4.3.10. Uso de la librería de virtualización <i>libvirt</i>	39
4.3.11. Uso de <i>Python 2.7</i> como principal lenguaje de programación	40
4.3.12. Uso de la librería de red <i>twisted</i>	41
4.3.13. Uso del protocolo de transferencia de ficheros <i>FTP</i>	41
4.3.14. Uso del servidor <i>FTP</i> <i>pyftpdlib</i>	42
4.3.15. Almacenamiento de las imágenes de disco en formato comprimido	43
4.3.16. Uso de imágenes <i>copy-on-write</i>	47
4.3.17. Uso de dos discos duros en cada máquina virtual	48
4.3.18. Uso del formato de imagen <i>qcow2</i>	48
4.3.19. Creación de seis familias de máquinas virtuales	49
4.3.20. Uso de los servidores de edición de imágenes	51
4.3.21. Cifrado selectivo del tráfico	52
4.3.22. Uso del <i>framework</i> <i>web2py</i>	52
4.4. Vista lógica	55
4.4.1. Visión general	55
4.4.2. Paquetes y clases significativos de la arquitectura	56
4.4.2.1. <i>ccutils</i>	56
4.4.2.2. <i>network</i>	57
4.4.2.3. <i>ftp</i>	58
4.4.2.4. <i>imageRepository</i>	59
4.4.2.5. <i>virtualMachineServer</i>	60
4.4.2.6. <i>clusterServer</i>	61
4.4.2.7. <i>clusterEndpoint</i>	62
4.4.2.8. <i>clusterConnector</i>	62
4.4.2.9. <i>testing</i>	62
4.4.2.10. <i>webServer</i>	63
4.4.3. El paquete <i>network</i>	63
4.4.3.1. La librería de red <i>twisted</i> : visión general	63
4.4.3.2. La clase <i>Packet</i>	67
4.4.3.3. Envío y recepción de datos	69
4.4.3.4. Conexiones de red	70
4.4.3.5. Hilos de red	74
4.4.3.6. La clase <i>NetworkManager</i>	76
4.4.4. El paquete <i>ftp</i>	77
4.4.4.1. El servidor <i>FTP</i> <i>pyftpdlib</i> : visión general	77
4.4.4.2. El servidor <i>FTP</i> de <i>CygnusCloud</i>	78
4.4.4.3. El cliente <i>FTP</i> de <i>CygnusCloud</i>	79
4.4.5. El paquete <i>imageRepository</i>	80
4.4.5.1. Adición de una capa adicional al servidor <i>FTP</i>	80
4.4.5.2. Funciones soportadas por el repositorio de imágenes	81
4.4.5.3. La conexión de control	81
4.4.5.4. Los <i>slots</i> de transferencia	82
4.4.5.5. Clases principales	82
4.4.5.6. Arranque del repositorio de imágenes	84
4.4.5.7. Interacciones del repositorio de imágenes	84
4.4.5.8. Formatos de paquete	99
4.4.5.9. Distribución de los ficheros	102
4.4.5.10. Esquema de la base de datos	102
4.4.6. El paquete <i>virtualMachineServer</i>	103
4.4.6.1. Redes virtuales	103
4.4.6.2. Interacción con <i>libvirt</i> a bajo nivel	104
4.4.7. El paquete <i>clusterServer</i>	107

4.4.8. El paquete <i>webServer</i>	107
4.4.8.1. Estructura general de la web	108
4.4.8.2. Estructura de direcciones	108
4.4.8.3. Secciones y subsecciones	110
4.4.8.4. La barra de direcciones	112
4.4.8.5. Gestión de usuarios	113
4.4.8.6. Interacción entre páginas	113
4.4.8.7. Flujo de ejecución de arranque de una máquina virtual	113
4.5. Vista de procesos	115
4.6. Vista de despliegue	115
4.7. Vista de implementación	115
5 Resultados	117
6 Manual de usuario	119
6.1. Requisitos mínimos	119
6.1.1. Requisitos <i>hardware</i>	119
6.1.2. Requisitos <i>software</i>	120
6.2. Instrucciones de instalación y configuración en Ubuntu 12.04	120
6.2.1. Pasos comunes	120
6.2.2. Servidores de máquinas virtuales	120
6.2.2.1. Instalación y configuración de KVM	120
6.2.2.2. Instalación del módulo de <i>CygnusCloud</i>	121
6.2.2.3. Configuración del módulo de <i>CygnusCloud</i>	122
6.2.3. Servidores de cluster	123
6.2.3.1. Instalación del módulo de <i>CygnusCloud</i>	123
6.2.3.2. Configuración del módulo de <i>CygnusCloud</i>	123
6.2.4. Servidor web	124
6.2.4.1. Instalación del módulo de <i>CygnusCloud</i>	124
6.2.4.2. Configuración del módulo de <i>CygnusCloud</i>	124
6.3. Creación de imágenes	125
6.3.1. Creación de nuevas imágenes en <i>virt-manager</i>	126
6.3.1.1. Pasos básicos	126
6.3.1.2. Configuración de la máquina virtual	129
6.3.1.3. Instalación de los <i>drivers</i> en Windows	132
6.3.2. Importar máquinas virtuales existentes a <i>virt-manager</i>	134
6.3.3. Uso de <i>samba</i> para configurar las máquinas virtuales	134
6.3.3.1. Prerrequisitos (sólo en Linux)	134
6.3.3.2. Compartición de Windows a Windows	135
6.3.3.3. Compartición de Linux a Linux	136
6.3.3.4. Compartición de Windows a Linux	136
6.3.3.5. Compartición de Linux a Windows	136
6.3.4. Importar máquinas virtuales a <i>CygnusCloud</i>	137
6.4. Uso de la página web	138
6.4.1. Instalación y uso del framework <i>web2py</i>	138
6.4.2. Uso de la web	139
6.4.2.1. Acceso	139
6.4.2.2. Opciones disponibles para los estudiantes	142
6.4.2.3. Opciones disponibles para los administradores	143
Apéndices	147
A Licencia	147
A.1. Código fuente	147
A.1.1. Versión modificada de <i>noVNC</i>	147

A.1.2. Resto de código fuente de <i>CygnusCloud</i>	147
A.2. Documentación	147
B Git: guía de referencia	149
B.1. Git y Subversion	149
B.2. Instalación y configuración básica de Git	150
B.3. Uso del repositorio de Google Code	151
B.4. Uso básico de Git	151
B.4.1. Registrar cambios en el repositorio	151
B.4.1.1. Comprobando el estado de los ficheros	152
B.4.1.2. Incluyendo nuevos ficheros	152
B.4.1.3. Modificando ficheros ya existentes	152
B.4.1.4. Ignorando ficheros y directorios	153
B.4.1.5. Nuestro primer <i>commit</i>	154
B.4.1.6. <i>Commits</i> “rápidos”	154
B.4.1.7. Borrando ficheros del repositorio	154
B.4.1.8. Moviendo y renombrando ficheros	154
B.4.2. Revisar cambios en el repositorio	155
B.4.2.1. Uso de <i>gitk</i>	155
B.4.3. Deshacer cambios	157
B.4.3.1. Cambiar el último <i>commit</i>	157
B.4.3.2. Eliminar un fichero en estado <i>staged</i>	157
B.4.3.3. Deshacer los cambios en un fichero	157
B.4.4. Uso de repositorios remotos	158
B.4.4.1. Incorporar los cambios de un repositorio remoto al repositorio local	158
B.4.4.2. Incorporar los cambios del repositorio local a un repositorio remoto	159
B.4.4.3. Renombrar y eliminar repositorios remotos	159
B.4.5. Uso de etiquetas	159
B.4.5.1. Crear etiquetas	159
B.4.5.2. Compartir etiquetas	160
B.5. Uso de ramas o <i>branches</i>	160
B.5.1. ¿Qué es una rama?	160
B.5.2. Uso básico de ramas	161
B.5.2.1. Resolución de conflictos	163
B.5.3. Gestión de ramas	164
B.5.4. Ramas remotas	164
B.5.4.1. Acceso a ramas remotas	164
B.5.4.2. Crear ramas remotas	165
B.5.4.3. <i>Tracking branches</i>	165
B.5.4.4. Borrar ramas remotas	165
B.5.5. Uso de <i>rebase</i>	165
B.5.5.1. Uso básico	165
B.5.5.2. Un caso interesante	166
C BibTeX: guía de referencia	169
C.1. Configuración de BibTeX	169
C.2. Formato de la base de datos	169
C.2.1. Aspectos básicos	169
C.2.2. Registros y campos bibliográficos	169
C.2.3. Tipos de registros bibliográficos	170
D Sockets en Python	173
D.1. Introducción	173
D.2. Clasificación	173
D.3. Familias de Sockets	173
D.4. Creación de un socket	174

D.4.1. Socket en el servidor	174
D.4.2. Socket en el cliente	175
D.5. Ejemplos	175
D.5.1. Ejemplo 1	175
D.5.2. Ejemplo 2	175
D.6. Bibliografia	178
E Creación de imágenes Windows en Xen	179
F Configuración de KVM	185
G Configuración de una red virtual en modo NAT	187
G.1. ¿Qué queremos hacer?	187
G.2. Creación del bridge	187
G.3. Activar el encaminamiento IP	188
G.4. Configuración de iptables	188
G.5. Configuración de dnsmasq	189
G.6. Redirección del tráfico de un puerto al servidor VNC de una máquina virtual	189
Bibliografía	191

Índice de figuras

2.1. Computación en la nube: un servicio global	9
2.1. El <i>Cloud Computing</i> en la actualidad	12
2.1. Cloud privado “ <i>inHouse</i> ”	16
2.2. Cloud híbrido en el centro de proceso de datos del proveedor	17
3.1. Virtualización hardware	22
3.2. Virtualización software	22
3.3. Paravirtualización	22
3.4. Virtualización completa	23
4.1. Tiempos de compresión y descompresión	44
4.2. Tamaño del fichero comprimido	44
4.3. Uso promedio de CPU	45
4.4. Uso promedio de memoria RAM	45
4.5. Tasa de lectura de disco (promedio)	46
4.6. Tasa de escritura a disco (promedio)	46
4.7. Diagrama de secuencia web2py	53
4.1. Diagrama de paquetes de <i>CygnusCloud</i>	55
4.2. Descomposición del paquete <i>ccutils</i>	56
4.3. Descomposición del paquete <i>network</i>	57
4.4. Descomposición del paquete <i>imageRepository</i>	59
4.5. Descomposición del paquete <i>virtualMachineServer</i>	60
4.6. Descomposición del paquete <i>clusterServer</i>	61
4.7. Descomposición del paquete <i>clusterEndpoint</i>	62
4.8. Representación gráfica del funcionamiento de la librería <i>twisted</i>	64
4.9. Procesamiento de datos entrantes en <i>twisted</i> : diagrama de secuencia	65
4.10. Relaciones entre protocolos, factorías de protocolos y <i>endpoints</i>	66
4.11. Establecimiento de una conexión en <i>twisted</i> : diagrama de secuencia	67
4.12. Interacción con <i>twisted</i> a muy bajo nivel de abstracción: diagrama de clases	69
4.13. Relaciones más relevantes en las que intervienen las clases <i>NetworkConnection</i> , <i>ClientConnection</i> y <i>ServerConnection</i>	71
4.14. Evolución del estado de una conexión de tipo servidor	72
4.15. Evolución del estado de una conexión de tipo cliente	73
4.16. Jerarquía de hilos de red y sus relaciones más importantes	74
4.17. Relaciones más importantes de la clase <i>NetworkManager</i>	76
4.18. Principales clases del servidor FTP <i>pyftpdlib</i>	77
4.19. El servidor FTP de <i>CygnusCloud</i> : diagrama de clases	79
4.20. Principales clases del repositorio de imágenes y sus relaciones	83
4.21. Arranque del repositorio de imágenes: diagrama de secuencia	85
4.22. Registro de un identificador de imagen: diagrama de secuencia	86
4.26. Transferencia de una imagen a una máquina remota	87
4.23. Inicio de la transferencia de una imagen a una máquina remota: secuencia básica	88
4.24. Detección de errores al recibir la petición de transferencia	89
4.25. Detección de errores al habilitar la transferencia	90
4.28. Liberación de una imagen de disco	91

4.27. Descarga de un fichero en exclusividad	92
4.32. Transferencia de una imagen desde una máquina remota	93
4.29. Inicio de la transferencia de una imagen desde una máquina remota: secuencia básica	94
4.30. Detección de errores al recibir la petición de transferencia	95
4.31. Detección de errores al habilitar la transferencia	96
4.33. Tratamiento de las transferencias parciales	97
4.34. Interacciones asociadas al borrado de una imagen de disco	98
4.35. Recopilación del estado del repositorio	99
4.36. Apagado del repositorio de imágenes: diagrama de secuencia	100
4.37. Codificación del estado de una imagen	103
4.38. Fichero de definición de una red virtual configurada en modo NAT	103
4.39. Relaciones de la clase <code>VirtualNetworkManager</code>	104
4.40. Fichero de definición de una máquina <code>Linux</code>	106
4.41. Diagrama de direcciones web2py	110
4.42. Diagrama de niveles de la web.	112
4.43. Secuencia de arranque de una máquina virtual.	115
 6.1. Pantalla inicial de <code>virt-manager</code>	126
6.2. Ubicación del botón <code>create a new virtual machine</code>	126
6.3. Pasos 1 y 2 del asistente de creación de máquinas virtuales	127
6.4. Pasos 3 y 4 del asistente de creación de máquinas virtuales	127
6.5. Paso 5 del asistente de creación de máquinas virtuales	128
6.6. Diálogo de configuración de la máquina virtual	129
6.7. Configuración del procesador de la máquina virtual	130
6.8. Creación de las imágenes de disco	130
6.9. Configuración de la tarjeta de vídeo	131
6.10. Configuración de los periféricos de entrada	132
6.11. Configuración del CD-ROM con los drivers VirtIO	133
6.12. Corrección de los problemas de arranque en instalaciones Windows	133
6.1. Interfaz administrativa de web2py	139
6.2. Página de inicio de <code>CygnusCloud</code>	140
6.3. Página de arranque de máquinas virtuales para estudiantes	141
6.4. Visor de escritorio remoto para una máquina virtual arrancada	142
 B.1. Subversion	149
B.2. Git	149
B.1. Ciclo de vida de un fichero en Git	151
B.2. La GUI <code>gitk</code>	156
B.1. Almacenamiento de los datos en la mayoría de sistemas de control de versiones (CVS, SVN, Bazaar,...)	160
B.2. Almacenamiento de los datos en Git	160
B.3. Organización de los datos tras un <code>commit</code>	161
B.4. Historia tras dos <code>commits</code> más	161
B.5. <i>Branches</i> y <i>commits</i>	162
B.6. Creación de una nueva rama	162
B.7. Situación tras ejecutar <code>git checkout testing</code>	163
B.8. <i>Commit</i> en la rama <code>testing</code>	163
B.9. Situación inicial (antes de utilizar <code>rebase</code>)	166
B.10. Integrando los cambios con <code>merge</code>	166
B.11. Integrando los cambios con <code>rebase</code>	166
B.12. Uso útil de <code>rebase</code> . Situación de partida.	167
B.13. Uso útil de <code>rebase</code> . Resultado obtenido.	167
 E.1. Configuración de RDP en Windows 7 y Vista	182

Índice de cuadros

4.3.1Características de las imágenes <i>vanilla</i> Windows	50
4.3.2Características de las imágenes <i>vanilla</i> Linux	50
4.4.1Características de los distintos tipos de paquete	68
4.4.2Etiquetas de tipo que pueden aparecer en un paquete	69
4.4.4Características principales de los paquetes utilizados en el repositorio de imágenes	101
4.4.5.Tabla relación tipos de usuario y secciones	111
4.4.6.Tabla relación secciones y subsecciones	111
B.4.1Argumentos muy útiles de <code>git log</code>	155
C.2.1Tipos de registros bibliográficos	171

Capítulo 1

Descripción del problema

1.1. Introducción

El uso de las tecnologías de la información y la comunicación (TIC) en el sector educativo no es algo nuevo. Ya en la década de los 60, Plattner y Herron reconocieron que el uso de computadores era extremadamente útil para enseñar a tomar decisiones en situaciones reales y a desempeñar distintos cargos, y también para fomentar la participación. Por ello, no es de extrañar que las TIC hayan adquirido un papel protagonista en el panorama educativo actual.

De esta manera, a medida que las TIC han ido implantándose en la sociedad, su uso en la universidad no ha hecho más que crecer. Además, la puesta en funcionamiento del llamado Espacio Europeo de Educación Superior ha hecho que las sesiones prácticas tengan un importante peso en todos los planes de estudio universitarios. Y dado que en estas sesiones prácticas se hace uso de las TIC, podemos afirmar que los estudiantes pasarán una cantidad de tiempo considerable delante de un ordenador personal durante su estancia en la universidad.

La Universidad Complutense de Madrid (UCM) es la universidad presencial española con mayor número de alumnos[INE]. Para que todos ellos puedan realizar sus trabajos académicos, existe un gran número de aulas de informática y laboratorios repartidos entre sus diversos centros. Naturalmente, la gestión y el mantenimiento de estos espacios es responsabilidad de personal de administración y servicios dependiente, en última instancia, de los Servicios Informáticos de la universidad.

Existen varias alternativas para llevar a cabo la gestión de las aulas de informática. A lo largo de este capítulo, presentaremos la que nos es más conocida y también la que, a causa de sus inconvenientes, ha motivado el desarrollo de *CygnusCloud*: el modelo de gestión de los laboratorios de la Facultad de Informática de la UCM.

A la hora de presentar dicho modelo de gestión, nos centraremos en el *software*, es decir, en la forma en que se instala, configura y se permite utilizar a los estudiantes un conjunto distinto de programas en cada aula de informática. Por ello, en este capítulo no haremos referencia a algunas características también relevantes del modelo como la gestión de altas y bajas de usuarios, el mantenimiento del equipamiento *hardware* del aula y el alojamiento de material docente en la web de la Facultad de Informática.

1.2. El modelo de gestión de los laboratorios de la Facultad de Informática de la UCM

1.2.1. Introducción

Los planes de estudio de la Facultad de Informática de la UCM siempre se han caracterizado por el importante peso asociado a las sesiones prácticas. En el caso de los planes de estudio en extinción, los estudiantes deben pasar más de la mitad de su tiempo de estudio ante un ordenador personal. Además, esta situación se ha mantenido en el caso de los nuevos planes de estudio, implantados en el curso académico 2010/2011.

Así pues, es necesario disponer de espacios en los que los alumnos puedan realizar todas estas actividades prácticas: los *laboratorios*. Para ello, la Facultad de Informática cuenta con 300 ordenadores personales conectados a Internet, repartidos entre 12 laboratorios. En términos generales, podríamos considerar un laboratorio como un aula de informática, si bien algunos disponen de equipamiento específico (como FPGAs, osciloscopios, entrenadores, polímetros, ...).

Con independencia de su equipamiento, del *software* instalado y de las asignaturas que se imparten en los mismos, todos los laboratorios son gestionados por el propio centro.

1.2.2. Requisitos docentes de los laboratorios

En la actualidad, en la Facultad de Informática de la UCM se imparte docencia correspondiente a tres planes de estudio de grado (Grado en Ingeniería de Computadores, Grado en Ingeniería del Software y Grado en Ingeniería Informática), uno de doble grado (Doble Grado en Ingeniería en Informática y Matemáticas), dos ingenierías técnicas (en Informática de Gestión y en Informática de Sistemas) y una ingeniería superior (Ingeniería Informática), junto con múltiples programas de postgrado y doctorado.

Es cierto que la informática es fundamental en todos estos estudios, pero las diferentes asignaturas que se imparten en los laboratorios tienen necesidades muy diferentes. Por ejemplo, una asignatura de introducción a la programación en C++ requerirá un editor y un compilador, mientras que una asignatura de electrónica requerirá osciloscopios, polímetros, resistencias, transistores, ... junto con *software* de simulación de circuitos electrónicos (como PSpice).

Además, en las asignaturas pueden utilizarse herramientas de pago, como MATLAB y *Rational Software Architect*. En la mayoría de casos, los estudiantes no pueden instalar estas aplicaciones en sus equipos por diversos motivos (como el elevadísimo coste de las licencias o la imposibilidad de encontrar la versión instalada en los laboratorios).

De esta manera, la Facultad debe permitir trabajar a los alumnos en los laboratorios fuera del horario de clases no sólo para que utilicen dichas herramientas o el equipamiento específico de estos espacios, sino también para que puedan elaborar los distintos trabajos de clase que se les asignen.

Por último, también es necesario que en los laboratorios puedan tener lugar exámenes de carácter práctico a lo largo de todo el curso académico.

En las siguientes secciones, mostraremos cómo la red de laboratorios satisface todos estos requisitos.

1.2.3. Servicios prestados a los alumnos

Cada alumno matriculado en la Facultad de Informática de la UCM dispone de una cuenta personal que le permite utilizar cualquier puesto de la red de laboratorios. Dicha cuenta tiene asociado un espacio de almacenamiento en disco (100 MB) junto con una cuota de impresión (de 200 hojas por curso académico).

Los servicios de la red de laboratorios que son visibles para los alumnos son los siguientes:

- el control de uso de los puestos de laboratorio. Todos los alumnos deben autenticarse antes de utilizar cualquier puesto, lo que permite detectar y sancionar las actividades indebidas que se realicen en los laboratorios.
- el envío de avisos y notificaciones a los alumnos que utilizan la red de laboratorios.
- el reparto equitativo del tiempo de uso de los puestos de laboratorio en períodos de alta ocupación (restringiendo la duración de las sesiones). Esto garantiza que todos los alumnos puedan utilizar los laboratorios en igualdad de condiciones durante los turnos de práctica libre.
- la consulta de la ocupación en cualquier momento de todos los puestos de la red de laboratorios.

- el almacenamiento de la documentación del material experimental de los laboratorios (como las *datasheets* en las que se recogen las especificaciones de los circuitos integrados que se utilizan en las prácticas).

1.2.4. Servicios prestados a los profesores

La Facultad de Informática de la UCM también ofrece a cada uno de sus profesores una cuenta personal. Esta también les permite utilizar cualquier puesto de la red de laboratorios y dispone de mayores cuotas de almacenamiento y de impresión.

Los profesores también pueden hacer uso de servicios adicionales de apoyo a la docencia. Los más relevantes son los siguientes:

- la restricción de las comunicaciones entre los puestos de laboratorio. Esto permite, entre otras cosas, restringir el acceso a internet durante las clases.
- la recolección de prácticas, exámenes y trabajos realizados por los alumnos. El acceso a los mismos siempre estará limitado al profesor y a su propietario.
- la adquisición, instalación y configuración de todo el *software* que se vaya a utilizar a lo largo del curso académico.

1.2.5. Otros servicios

Junto con los servicios que hemos descrito hasta ahora, que están orientados a los usuarios, existen otros especialmente orientados a facilitar el trabajo del personal técnico, como la monitorización de toda la actividad de los laboratorios en tiempo real y la replicación de configuraciones (que es posible al compartir todos los puestos de un mismo laboratorio la misma configuración *hardware* y *software*).

1.3. Inconvenientes del modelo presentado

1.3.1. Desaprovechamiento de recursos

Uno de los grandes problemas del modelo de gestión que hemos presentado en la sección 1.2 es el desaprovechamiento de recursos.

Los recursos desaprovechados son de naturaleza muy dispar: aulas de informática infrautilizadas que se encuentran en otros centros, PCs de la red de laboratorios, energía, personal, etcétera. Para mostrar cómo se desperdician todos estos recursos, presentaremos algunos ejemplos.

1.3.1.1. Aulas de informática de otros centros

Justo antes de las convocatorias de exámenes y de los períodos vacacionales, los estudiantes deben lidiar con una auténtica “avalancha” de prácticas a entregar en un corto periodo de tiempo. Además, antes de la realización de sus exámenes es bastante frecuente que los alumnos deban revisar las prácticas ya entregadas. Por todo esto, a nadie debe extrañar el elevadísimo uso de los laboratorios en estas épocas del año.

Puesto que la red de laboratorios cuenta con sólo 300 puestos y un número mucho mayor de usuarios (según la memoria de la Facultad del curso 2009/2010 [UCM11], hay más de 2000 alumnos matriculados en este centro), los alumnos deberán competir por los distintos puestos. Ahora bien, no todos los alumnos que utilizan los puestos del mismo laboratorio fuera del horario de clase (horario de práctica libre) tienen por qué estar matriculados en las mismas asignaturas, por lo que necesitarán utilizar *software* (y también posiblemente *hardware*) distinto. Para mostrar cómo se agrava aún más la situación, haremos uso de un ejemplo.

Consideremos la semana anterior a las vacaciones de Navidad. En ella, los alumnos de la asignatura Estructura de Computadores estarán trabajando en una práctica de VHDL de cierta envergadura, mientras que los alumnos de primer curso deberán entregar una práctica de programación en C++.

Supongamos que el simulador y el entorno de desarrollo en VHDL sólo están instalados en los laboratorios 7, 8, 9 y 10. Supongamos también que, en un momento dado, se está impartiendo docencia en los laboratorios 9 y 10. A causa del gran número de alumnos matriculados en la Facultad de Informática, esta situación se da con bastante frecuencia.

Si un alumno desea trabajar en su práctica de VHDL en los laboratorios de la Facultad y fuera del horario de clase (en los denominados turnos de práctica libre), deberá utilizar obligatoriamente los laboratorios 7 y 8. Pero en los turnos de práctica libre estos laboratorios no están reservados en exclusiva para los alumnos de Estructura de Computadores, por lo que los estudiantes de primero también podrán realizar en ellos su práctica de programación.

Si no existen puestos libres en los laboratorios 7 y 8 y un alumno desea continuar con su práctica de VHDL, existen varias alternativas:

- echar de esos laboratorios a los alumnos que no deban utilizar el software específico, es decir, a los alumnos de primero.
- imponer restricciones de tiempo, de modo que los distintos alumnos que utilizan el laboratorio estarán obligados a compartir sus puestos. De esta manera, los alumnos que deseen realizar la práctica en el laboratorio deberán esperar a que haya puestos libres; y los que estén utilizando el laboratorio deberán abandonarlo tras cierto intervalo de tiempo.

Incluso dejando de lado el aumento de la crispación entre los alumnos al que dan lugar ambas alternativas y la situación de auténtico hacinamiento en los laboratorios, esta situación sigue siendo poco deseable: si en un aula de informática de la Facultad de Filología hay puestos libres, estos no podrán utilizarse para completar *cualquiera* de las dos prácticas por el mero hecho de no tener instalado el *software* necesario (ya que las necesidades de los alumnos de esa facultad no justifican la instalación de dicho *software*). En otras palabras: esta situación se puede evitar.

1.3.1.2. Personal encargado de las aulas de informática

Los laboratorios de la Facultad de Informática de la UCM tienen asignado su propio personal de administración y servicios. En períodos de intensa actividad en la Facultad, los técnicos de los laboratorios estarán saturados, mientras que sus compañeros de otros centros pueden tener mucho menos trabajo o incluso estar ociosos.

Esta situación da lugar a mayores tiempos de espera y a un incremento considerable del tiempo de resolución de las incidencias, y mejoraría si los alumnos pudiesen distribuirse de forma más homogénea entre las aulas de informática de todos los centros de la UCM. Nuevamente, esto no es posible al no permitirlo las características técnicas de los PCs y el *software* instalado en las distintas aulas de informática.

1.3.1.3. Energía consumida

Hoy en día, el coste de la energía no es despreciable en absoluto. Prueba de ello es que los centros de proceso de datos de más reciente construcción se caracterizan por la búsqueda de la mayor eficiencia energética posible para así reducir lo máximo posible los costes de explotación.

Ante la actual situación de crisis económica, la Universidad Complutense no puede quedarse atrás en la mejora de la eficiencia energética: el ahorro energético permite reducir el efecto que las sucesivas reducciones del presupuesto tienen sobre otras partidas (como las dedicadas a becas y a investigación). Por ello, es fundamental racionalizar el consumo energético de las aulas de informática.

La primera fuente de consumo innecesario es el sistema de climatización y alumbrado de las aulas de informática infrautilizadas, cuyo uso resulta imprescindible con independencia del número de alumnos que estén trabajando en el aula.

La segunda fuente de consumo es el propio equipamiento informático asignado a las aulas infrautilizadas: los PCs, servidores y también el equipamiento de red. Aún apagando los equipos en desuso, se sigue desperdiciando energía, ya que

- frecuentemente, la red no se utiliza para más que para acceder a internet o a la intranet de la UCM. Estas actividades utilizan muy poco ancho de banda, por lo que el equipamiento de red y los servidores asignados al aula de informática están siendo frecuentemente infrautilizados.
- los equipos en uso están inactivos durante buena parte de su vida útil, y el consumo energético en períodos de inactividad no es despreciable.

Estos consumos pueden reducirse si los alumnos que utilizan estas aulas se agrupan en un único aula de informática, cosa que no permite el modelo actual por motivos que ya hemos expuesto:

- los PCs del aula de informática pueden no ser lo suficientemente potentes como para cubrir las necesidades de todos los estudiantes, y
- los alumnos de distintas titulaciones no tienen por qué utilizar el mismo *software*.

1.3.1.4. Licencias de pago

En las aulas de informática de la UCM se encuentran instaladas aplicaciones de pago. Puesto que las licencias de dichas aplicaciones son muy costosas, la UCM garantiza que todos sus alumnos puedan utilizarlas en las aulas de informática.

Ahora bien, tal y como hemos expuesto en las secciones anteriores, no todos los alumnos utilizan las aulas de informática para lo mismo. Así, a causa de la mera ocupación de ciertos espacios, no es posible utilizar todas las licencias de pago adquiridas por la UCM aún cuando es necesario. También podemos aprovechar el ejemplo que presentamos en la sección 1.3.1.1 para mostrar cómo esto puede ocurrir.

Los alumnos que deseen avanzar con su práctica de VHDL necesitan utilizar un entorno de desarrollo de VHDL y un simulador. Este *software* estará instalado en todos los puestos de los laboratorios 7 y 8, pero no se utilizará en los puestos ocupados por los alumnos de primero. De esta manera, aunque las licencias de estos puestos no se utilizan, no será posible aprovecharlas al no estar instalado el *software* en otras aulas de informática.

Además, puesto que también la adquisición de *software* se realiza a través de la red de laboratorios, resulta imposible compartir las licencias sin utilizar con otros centros, lo que incrementa considerablemente el gasto en *software*. Por ejemplo, si tanto en la Facultad de Ciencias Físicas como en la Facultad de Informática se utiliza el *software* MATLAB, estos dos centros no podrán compartir sus licencias de MATLAB porque las aulas de informática de Físicas no forman parte de la Red de Laboratorios de la Facultad de Informática.

Considerando el desorbitado precio de las aplicaciones propietarias en general, las limitaciones de las versiones de prueba de muchas de ellas y la imposibilidad de encontrar en el mercado las versiones instaladas de algunas de las que se utilizan para realizar prácticas (frecuentemente, por ser demasiado antiguas), esta situación resulta inadmisible.

1.3.2. Limitaciones del sistema de almacenamiento

Toda cuenta de la red de laboratorios de la Facultad de Informática tiene asignado un espacio de almacenamiento en disco accesible desde cualquier puesto.

Ahora bien, la cuota actual asignada a los alumnos (100 MB) es insignificante, y en no pocos casos es sólo suficiente para almacenar el material docente de las distintas asignaturas.

Por otra parte, sólo es posible acceder a ese espacio de almacenamiento desde la propia red de laboratorios, siendo necesario utilizar dispositivos de almacenamiento extraíble o servicios de alojamiento de archivos en la nube (como Dropbox, Microsoft SkyDrive y Google Drive) para transferir datos entre los equipos particulares de los estudiantes y los PCs de la red de laboratorios.

1.3.3. Sobrecoste de los PCs de las aulas de informática

En el modelo de gestión de los laboratorios de la Facultad de Informática de la UCM no existen laboratorios asignados en exclusiva a una asignatura o departamento. Por ello, en un mismo laboratorio se imparte docencia de asignaturas distintas.

Ahora bien, no todas las asignaturas asignadas al mismo laboratorio tienen las mismas necesidades. Por ello, los PCs de cada laboratorio deben ser lo suficientemente potentes para así cumplir los requisitos del *software* que utilizan dichas asignaturas.

Para cuantificar el sobrecoste de los equipos, consideraremos tres asignaturas típicas: una asignatura de programación, una asignatura relacionada con la informática gráfica y una asignatura de diseño de circuitos. Actualmente, estos tipos de asignatura se imparten en los laboratorios de la Facultad de Informática.

- La asignatura de programación requiere un PC con una CPU de gama media-baja, al menos 4 GB de memoria RAM, una tarjeta gráfica integrada y un disco duro de capacidad estándar.
- La asignatura de informática gráfica requiere un PC con una CPU de gama media-alta, al menos 4 GB de memoria RAM, una tarjeta gráfica dedicada de gama media-baja y un disco duro de capacidad estándar.
- La asignatura de diseño de circuitos requiere un PC con una CPU de gama media-alta, 4 GB o más de memoria RAM, una tarjeta gráfica integrada y un disco duro de capacidad estándar.

Para realizar la comparación, fijaremos un proveedor. En nuestro caso, el proveedor será Clevisa Informática, que ha suministrado un gran número de equipos a la UCM.

Los precios y características de los equipos requeridos por las tres asignaturas, a los que llamaremos equipos 1, 2 y 3, a fecha de 25 de octubre de 2012, son los siguientes:

- Equipo 1 (programación): Asustek CM6431, con CPU Intel Core i3 3220, 4 GB de memoria RAM, tarjeta gráfica integrada Intel HD Graphics 3000, grabadora de DVDs y disco duro de 1 TB. Su coste es de 784,30 €, con IVA incluido.
- Equipo 2 (informática gráfica): HP P6-2206ES, con CPU Intel Core i5 2320, 8 GB de memoria RAM, tarjeta gráfica Nvidia Geforce GT 620, grabadora de DVDs y disco duro de 1 TB. Su coste es de 960,16 €, con IVA incluido.
- Equipo 3 (diseño de circuitos): Asustek CM6431, con CPU Intel Core i5 3450, 6 GB de memoria RAM, tarjeta gráfica integrada Intel HD Graphics 3000, grabadora de DVDs y disco duro de 1 TB. Su coste es de 941,48 €, con IVA incluido.

Para impartir las tres asignaturas en el mismo laboratorio, sería necesario adquirir el Equipo 2. Dado que el laboratorio tiene 20 puestos,

- si solamente se impartiera la asignatura de programación, el ahorro sería de

$$20 \cdot (960,16 - 784,30) = 20 \cdot 175,86 = 3517,2 \text{ euros}$$

- si solamente se impartieran las asignaturas de programación y diseño de circuitos, el ahorro sería de

$$20 \cdot (960,16 - 941,48) = 20 \cdot 18,64 = 373,6 \text{ euros}$$

En realidad, el sobrecoste es aún mayor por dos motivos:

- cuanto más potente es el *hardware* de un PC, mayor es su consumo energético. De esta manera, cuando se imparte la asignatura de programación la CPU y la tarjeta gráfica del Equipo 2 consumen más energía de la requerida. Análogamente, cuando se imparte la asignatura de informática gráfica, será la tarjeta gráfica la que eleve el consumo innecesariamente.
- las licencias del sistema operativo Windows incluidas con el equipo no se utilizan (la UCM utiliza sus propias licencias). El coste de estas licencias, llamadas licencias OEM (*Original Equipment Manufacturer*, fabricante de equipamiento original) es de 102,90 € (IVA incluido) para las versiones *Home Premium* y de 142,90 € (IVA incluido) para las versiones *Professional*.

1.3.4. Excesiva rigidez de la configuración de los equipos

De esta manera, en el caso mejor, en el que todos los equipos incorporan la versión más barata, *Home Premium*, el sobrecoste aumenta en

$$20 \cdot 102,90 = 2058 \text{ euros}$$

y en el caso peor, en el que todos los equipos incorporan la versión más cara (*Professional*), el sobrecoste aumenta en

$$20 \cdot 142,90 = 2858 \text{ euros}$$

1.3.4. Excesiva rigidez de la configuración de los equipos

Tal y como hemos descrito en la sección 1.2.4, entre los servicios que presta la red de laboratorios de la Facultad de Informática de la UCM se encuentra la adquisición, instalación y configuración de todo el *software* que se vaya a utilizar a lo largo del curso académico.

Este servicio evita a los docentes la tediosa labor de configuración de todos los equipos de los laboratorios, permitiendo que estos se dediquen en mayor medida a la atención del resto de sus obligaciones.

Pero también tiene un serio inconveniente: la instalación de nuevo *software* y la modificación de la configuración de los equipos de los laboratorios se convierte en una auténtica “carrera de obstáculos” burocrática, que da lugar a ingentes retrasos que afectan muy negativamente al normal desarrollo de las clases en los laboratorios.

Para ilustrar por qué ocurre esto último, describiremos el procedimiento a seguir para fijar la configuración de los equipos de los laboratorios que, en el presente curso académico (2012/2013), es el siguiente:

1. A principio de curso, los profesores escogen de entre todos los programas que figuran en una lista aquellos que necesitarán utilizar. Estos no tienen por qué instalarse en todos los laboratorios: cada profesor sólo puede solicitar la instalación de *software* en los laboratorios que utilizará en sus clases.
2. Los técnicos proceden a realizar la instalación de dicho *software* en las máquinas de prototipado de los distintos laboratorios.
3. Se crean imágenes de disco de las máquinas de prototipado, que se utilizarán para replicar la configuración de las mismas en todos los puestos de laboratorio.
4. En la primera semana de cada cuatrimestre, los profesores verifican la configuración de los laboratorios que utilizarán en sus clases. Durante este periodo de pruebas, los alumnos no podrán utilizarlos.
5. Los laboratorios se abren a todos los alumnos para su uso en turnos de prácticas a partir de la segunda semana de cada cuatrimestre.

Lo primero que llama la atención es el hecho de que *todos* los profesores de *todas* las asignaturas que se imparten en los laboratorios tienen *una* semana para verificar la configuración de los equipos. Teniendo en cuenta el gran número de titulaciones que se imparten en la Facultad de Informática, esta restricción impone un ritmo frenético a la hora de comprobar la correcta configuración de los equipos, lo que facilita que muchos errores se pasen por alto.

Es más, en caso de que se detecte alguna anomalía, esta deberá subsanarse durante esa misma semana, lo cual propicia la aparición de muchos más errores. Así, es bastante habitual que no todo el *software* instalado en los puestos de laboratorio funcione de forma correcta, siendo fácil encontrar aplicaciones en las que no es posible utilizar ni la funcionalidad más básica.

Además, es importante notar que los profesores sólo pueden escoger el *software* a instalar de una lista. Si un programa no aparece en dicha lista, deberán solicitar su instalación al Analista, lo cual da lugar a otra sucesión de trabas burocráticas. En caso de aplicaciones de pago, cuyas licencias suponen un coste importante para la Facultad, esto es bastante razonable. Pero esta situación resulta injustificable cuando se trata de *software* gratuito o que no supondrá ningún sobrecoste

Capítulo 1. Descripción del problema

para la Facultad, y da lugar a retrasos que, de acuerdo a nuestra experiencia, superan fácilmente el mes de duración.

Pero el problema no acaba aquí: si un profesor detecta un error en la configuración de los puestos tras la semana de pruebas, también será necesario iniciar otro lento proceso burocrático para algo tan sumamente simple como volver a ejecutar algunos instaladores en la máquina de prototipado, volver a generar la imagen de disco y clonarla.

En definitiva, el modelo actual da lugar a una sobreacumulación de retrasos y errores en la configuración de los equipos que degrada considerablemente la calidad de la docencia e impide a los alumnos el uso normal de los puestos de laboratorio, obligándoles en no pocos casos a utilizar sus propios equipos durante una parte muy significativa del curso académico.

Capítulo 2

Cloud Computing

2.1. Introducción

El *Cloud Computing* o computación en la nube es el resultado de la evolución de los modelos de cómputo existentes, y permite al cliente abstraerse del *hardware* con el cual va a trabajar y centrarse en los servicios que necesita para realizar dicho trabajo. De esta forma, el cliente solo debe preocuparse por lo que quiere hacer y no por configurar el entorno que necesita.

Así, en este nuevo modelo pueden combinarse muchas clases de recursos en tiempo real para satisfacer las especificaciones o proporcionar los servicios que el cliente necesita. Estos recursos pueden ser de naturaleza muy distinta: tiempo de CPU, espacio de almacenamiento, infraestructura de comunicaciones, etcétera.

Además, el *Cloud Computing* elimina los costes de adquisición del *hardware*, las licencias del software, el mantenimiento de las instalaciones, etcétera.



FIGURA 2.1: Computación en la nube: un servicio global

En este capítulo abordaremos el estudio de *Cloud Computing*, que proporciona la sólida base sobre la que hemos desarrollado *CygnusCloud*. Para ello, comenzaremos definiéndolo detalladamente para después explicar su evolución y finalizar describiendo las características y los diferentes tipos de sistemas en la nube.

2.2. Definición

Según indica el Instituto Nacional de Estándares y Tecnologías de Estados Unidos (NIST) en [Tec11], el *Cloud Computing* es

un modelo que permite acceder a través de una red, de forma adecuada, desde cualquier sitio y bajo demanda a un conjunto compartido de recursos informáticos configurables (como

redes, servidores, almacenamiento, aplicaciones, servicios...), que se pueden proporcionar y utilizar rápidamente y con un mínimo esfuerzo de gestión o interacción con el proveedor de servicios.

En esta definición podemos distinguir ya las cuatro características fundamentales del *Cloud Computing*:

- el autoservicio o administración propia de los servicios ofertados,
- la accesibilidad desde múltiples plataformas,
- la compartición de los recursos entre varios usuarios, y
- la prestación de servicios flexibles y perfectamente ajustables al entorno de trabajo.

A continuación, mostraremos los precedentes y la evolución histórica del *Cloud Computing*. Más adelante, en la sección [2.4](#), hablaremos detalladamente de estas características.

2.3. Historia

2.3.1. Precedentes

Aunque las ideas que hay detrás del *Cloud Computing* pueden parecer novedosas, en realidad están presentes en nuestra sociedad desde hace siglos. Para justificar esto, utilizaremos un ejemplo.

Con independencia de nuestro lugar de procedencia, edad o condición, todos nosotros necesitamos una misma cosa para subsistir: agua. Así, los primeros humanos ya se asentaban en territorios cercanos a ríos y lagos para poder acceder al agua dulce.

Siglos después, el crecimiento de la población hizo necesaria la construcción de pozos para extraer el agua subterránea. Según los registros históricos, los primeros pozos se excavaron en Jericó (Israel) hace unos 7000 años.

La concentración de la población dio lugar a la aparición de los primeros sistemas de almacenamiento y distribución de agua potable. Los primeros que fueron lo suficientemente eficientes y que transportaron agua con la calidad suficiente aparecieron en la antigua Grecia, y los primeros que lograron cubrir grandes distancias, los famosos acueductos, se construyeron durante el Imperio Romano.

Tras el retroceso que experimentaron durante la Edad Media, los sistemas de distribución de agua volvieron a adquirir protagonismo en la Edad Contemporánea. Así, el primer sistema de suministro capaz de abastecer (mediante fuentes) a una ciudad completa (Paisley, Escocia) fue diseñado por John Gibb a principios del siglo XIX.

Hoy en día, todos nosotros disponemos de agua potable purificada abriendo cualquier grifo de nuestra casa. Así, cada vez que abrimos un grifo, estamos obviando

- la procedencia del agua,
- los embalses, pozos o depósitos en los que se almacena,
- la planta de tratamiento en la que se ha purificado,
- el diseño de las redes de distribución y de alcantarillado, y
- la planta donde se depurarán las aguas residuales que generamos en nuestro hogar.

Aunque el sistema de distribución de agua puede llegar a ser muy complejo, para disponer de agua potable sólo debemos preocuparnos por pagar las facturas de la compañía suministradora.

Esta es la misma idea que hay detrás del *Cloud Computing*: los usuarios se preocupan únicamente por utilizar los servicios que han contratado y no por la forma en que se los presta su proveedor.

2.3.2. Aparición de las ideas básicas del *Cloud Computing*

El origen de las ideas básicas que han dado lugar al *Cloud Computing* se remonta a la década de los 60.

Así, en 1961 John McCarthy dio una gran importancia al hecho de que el mundo de la informática debía ser de uso público. Por otra parte, J.C.R Licklider introdujo en 1962 la idea de una “red de ordenadores intergaláctica”, que interconectaría a todo el mundo a lo largo y ancho del planeta y haría posible la compartición de información, datos y programas.

2.3.3. Primeros años de los computadores en la empresa

En la década de los 80, los sistemas informáticos de las empresas operaban de acuerdo a un modelo cliente/servidor: todas las aplicaciones y todos los datos estaban almacenados en enormes computadores, los servidores u ordenadores centrales. Así, si un usuario quería acceder a cierta información o ejecutar un determinado programa, tenía que conectarse al ordenador central a través de un terminal o estación de servicio.

Pero los ordenadores centrales tenían unos recursos limitados, el acceso a los mismos no era inmediato y no soportaban la concurrencia, es decir, dos usuarios no podían acceder a la misma información a la vez. Por ello, los usuarios debían esperar a que el ordenador central atendiese sus peticiones, lo que daba lugar a enormes retardos.

2.3.4. Aparición del modelo *peer to peer*

El gran inconveniente del modelo cliente/servidor es que el servidor se convierte en un cuello de botella, ya que todos los usuarios necesitan acceder al servidor para trabajar. El modelo *peer to peer* (P2P) surgió a partir de la idea de no tener que utilizar el servidor como intermediario a la hora de establecer una comunicación entre dos ordenadores. En este modelo, todos los ordenadores son a clientes y servidores a la vez.

La principal finalidad con las que se desarrolló el modelo P2P fue su uso en internet: los mensajes podían propagarse desde cualquier ordenador, lo que permitía incrementar la velocidad de trasferencia al no producirse las transmisiones desde un único punto.

En la práctica, puesto que el procesamiento de información requiere tiempo y energía y no todos los ordenadores estaban conectados permanentemente, la Red no sigue este modelo de forma estricta: cada usuario sólo puede conectarse a un conjunto de servidores, lo que le proporciona mayor libertad con respecto al modelo cliente/servidor y permite restringir el control y la gestión de los datos a un conjunto de servidores centrales.

2.3.5. Desarrollo del *Cloud Computing*

A finales de la década de los 90 aparecieron las primeras empresas que empezaron a proporcionar servicios a otras a través de la *web* y con un coste considerablemente inferior. Entre todas ellas, destaca *Salesforce.com*, que comenzó a vender aplicaciones a empresas a través de la *web* en 1999.

Tras el estallido de la burbuja .com, *Amazon* pasó a jugar un papel protagonista en el desarrollo del *Cloud Computing* al iniciar la modernización de sus centros de procesos de datos. Para evitar su infrautilización, *Amazon* apostó por la nueva arquitectura *Cloud* y lanzó al mercado el paquete *Amazon Web Services* en julio de 2002. Este incluía almacenamiento, capacidad de cómputo e incluso aprovechaba la inteligencia humana (en el caso del servicio *Amazon Mechanical Turk*).

Cuatro años más tarde, en 2006, *Amazon* lanzó al mercado su producto *Elastic Compute Cloud*, que constituye una infraestructura completa y accesible de *Cloud Computing* y que permite alquilar máquinas virtuales en las que individuos y pequeñas empresas pueden alojar sus propias aplicaciones.

En 2009, *Google* dio el gran salto al lanzar *Google Apps*, un conjunto de aplicaciones diseñadas para ser utilizadas a través de un navegador web.

Por otra parte, las *killer apps* creadas por empresas como *Microsoft* y *Google* también han tenido una gran importancia en el desarrollo del *Cloud Computing*. Un claro ejemplo son los servicios de

correo web, que han desterrado a los clientes de correo electrónico tradicionales en muchos casos y han contribuido a que todos los usuarios estén utilizando tecnologías *Cloud* en su vida diaria.

En la actualidad, resulta evidente que el *Cloud Computing* adquiere cada vez más importancia y que ha venido para quedarse: hoy en día, es posible acceder a estos servicios desde cualquier plataforma y desde casi cualquier lugar del mundo, lo que hace que cada vez más aplicaciones se adapten al *Cloud*.

Además, la mejora de la seguridad permite que las empresas puedan ampliar su infraestructura o externalizarla, lo que se traduce en un aumento de la flexibilidad, en un mayor número de servicios disponibles y en un importante ahorro de coste.

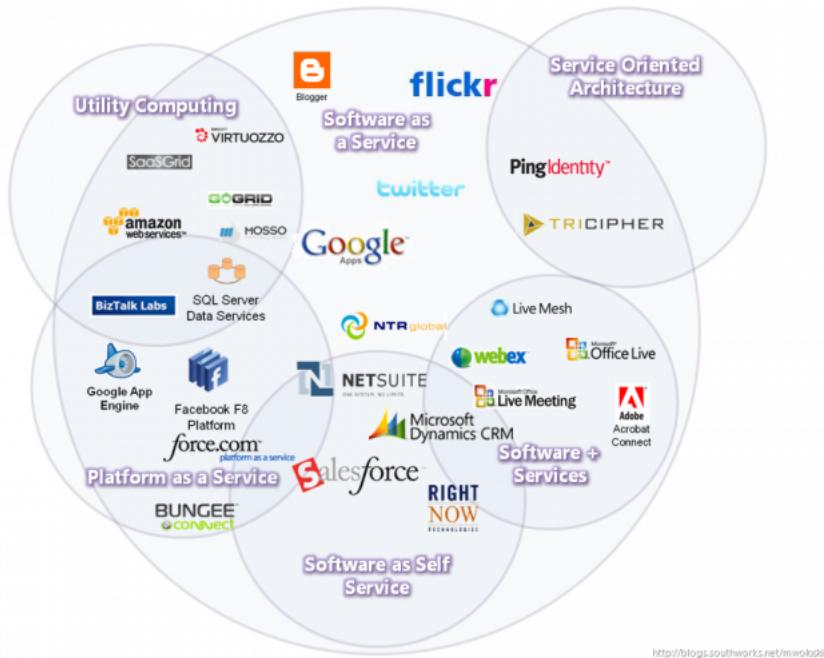


FIGURA 2.1: El *Cloud Computing* en la actualidad

2.4. Características

Las características que han permitido que el *Cloud Computing* sea un servicio en auge en la actualidad son las siguientes:

- **Agilidad.** El usuario puede gestionar de manera sencillas que aplicaciones necesita en su entorno de desarrollo. De esta forma el usuario puede adquirir o borrar sus aplicaciones según las requiera o no en cada momento de manera fácil y rápida. Para explicar mejor este aspecto introduciremos un ejemplo. Imaginemos una determinada empresa que usa una determinada aplicación durante un mes . Una vez finalizado el mes , la empresa decide centrarse en otro aspecto del desarrollo que no requieren de la aplicación y por lo tanto no quieren pagar por ella durante este mes así que decide eliminarla . Si al tercer mes la empresa requiere de nuevo la aplicación puede volver a adquirirla de manera rápida y sencilla.
 - **Reducción de costes.** Gracias a las empresas encargadas en alquilar su *hardware* a los usuarios interesados en ejecutar sus aplicaciones en la nube, estos no necesitarán hacer frente a los gastos iniciales de compra de recursos así como a los gastos de mantenimiento. Esto reduce el coste de desarrollo ya que aunque bien es cierto que es necesario pagar un alquiler por los servidores sobre los que se ejecuta el entorno de la empresa, este es menor a los posibles gastos a corto y largo plazo a los que habría que hacer frente.

- **Accesibilidad.** La independencia entre la entorno de trabajo y la localización del servidor que ejecuta el servicio permite al usuario acceder a sus herramientas de trabajo desde cualquier dispositivo con acceso a la web. De esta forma los cliente pueden trabajar desde su sobremesa, su portátil o incluso desde su dispositivo móvil.
- **Elasticidad y escalabilidad.** Las aplicaciones en la nube son totalmente elásticas en cuanto a su sencillez de implementación y adaptabilidad. Además, son totalmente escalables, es decir, una empresa puede estar utilizando un día solo un 10 % del total de sus recursos y al siguiente poder acceder al 80 % de los mismos con total normalidad y rapidez.
- **Recuperación.** Los proveedores de servicios en la nube proporcionan a los usuarios sistemas de almacenamiento secundario en servidores de forma que en caso de perdida de información se pueda recuperar de manera inmediata .
- **Estabilidad.** Las empresas de gestión de las aplicaciones en la nube aseguran al usuario una fuerte estabilidad en su entorno y permiten un rápido restablecimiento de los servicios que experimenten algún problema.
- **Seguridad.** Quizás esta es la característica que suscita una mayor controversia el *Cloud Computing*. Los clientes de estos servicios pueden tener la sensación de que al almacenar su información en un lugar externo a su propio dispositivo ,esto facilite que terceras personas puedan acceder a la misma.[Cry10, Mil09] Sin embargo, esto no es así en realidad. Las empresas encargadas de ofertar servicios en la nube mantienen estrictos controles de seguridad para evitar que la información de sus clientes pueda ser filtrada al exterior. Para almacenar esta información suelen utilizar centros de datos secundarios especializados en la protección y vigilancia de datos. [Tec11]

2.5. Modelos de servicio

el *Cloud Computing* ofrece sus servicios de acuerdo a tres modelos fundamentales, que se refieren a la forma en que las diferentes empresas clientes gestionan sus servicios contratados en la nube.

Estos modelos no son siempre excluyentes entre sí : existen empresas que mezclan los modelos fundamentales. Por ejemplo, Google App Engine permite al cliente ejecutar aplicaciones web en la infraestructura de Google , convirtiéndose de esta forma en una plataforma como servicio . Además estas aplicaciones web pueden ser utilizadas por otros usuarios prestando se esta forma además un modelo de software como servicio. En la figura ?? podemos ver más ejemplos de empresas que combinan varios de los modelos de servicio.

2.5.1. Infraestructura como servicio

Del termino inglés *infraestructura as a service* (IaaS), es el modelo más básico. En este modelo el proveedor oferta al usuario un conjunto de maquinas virtuales sobre las que trabajar. Por su parte , el cliente estará contratando únicamente la infraestructura tecnológica. Esta dispone de cierta capacidad de procesamiento, de almacenamiento y de comunicación.

Una vez que el cliente la ha contratado y el servidor la ha configurado, el cliente la utilizará para alojar sus aplicaciones. Estas maquinas virtuales pueden ser manejadas por hiper-visores, tales como Xen o KVN, o bien por sistemas como VirtualBox o VMWare.

Gracias a la habilidad de gestión de las maquinas virtuales por parte de los proveedores el cliente puede contratar tantas maquinas virtuales como necesite.

En cualquier caso una vez obtenidas las maquinas virtuales el cliente instalará el sistema operativo que más le interese y las aplicaciones. Por ello, en este modelo el cliente es el responsable del mantenimiento y configuración del entorno.

Puesto que los proveedores no deben preocuparse por otros aspectos como la seguridad o el pago de licencias, únicamente cobran por la cantidad de recursos asignados y consumidos.

Así, el modelo de infraestructura como servicio se limitan a aportar maquinas virtuales que los clientes utilizarán cuando las necesiten.

Algunos servicios de este tipo son *Amazon CloudFormation*, *Rackspace Cloud*, *Terremark*, *Windows Azure Virtual Machines* y *Google Compute Engine*.[\[MCF08\]](#)

2.5.2. Plataforma como servicio

Del término inglés *Platform as a service* (PaaS), los proveedores de este tipo de servicios ofrecen a sus clientes una plataforma completa, la cual incluye tanto el sistema operativo como el conjunto de todas las aplicaciones, la gestión de las bases de datos y el conjunto de servicios web que este necesite. Esta plataforma se instala sobre una infraestructura también proporcionada por el proveedor.

De esta forma, el cliente despliega las aplicaciones que necesite (desarrollos propios o licencias adquiridas) sin preocuparse por los costes de compra y de mantenimiento del *hardware y del software*.

La plataforma como servicio ofrece el equipo base y escala los recursos automáticamente para que el cliente no tenga que asignar dichos recursos manualmente.

Este modelo resulta ventajoso tanto para el cliente como para el proveedor, ya que el primero no necesita costear el precio completo de las licencias de todas las herramientas de desarrollo (algunas de las cuales serán usadas tan poco que costear una licencia completa por ella resultaría un despilfarro) y el segundo puede ofrecer un mismo entorno a un gran número de clientes con necesidades similares.

Algunos ejemplos de plataforma como servicio son *Amazon Elastic Beanstalk*, *Cloud Foundry*, *Heroku*, *Force.com*, *EngineYard*, *Mendix*, *Google App Engine*, *Windows Azure Compute* y *OrangeScape*.

2.5.3. Software como servicio

Del término inglés *Software as a service* (SaaS), este modelo es el que proporciona menos libertad y también menos preocupaciones al cliente. En este caso, el proveedor no solo ofrece la infraestructura y las herramientas, sino que sobre ellas instala el conjunto de aplicaciones que será usadas por el cliente.

Esto no solo libera al cliente de instalar y ejecutar las aplicaciones en sus propia computadora sino que también se encarga del mantenimiento, la recuperación y el soporte de las mismas.

El coste de alquiler de las aplicaciones en un modelo de software como servicio suele estar ligado a una tarifa plana anual[\[Tec12\]](#) o mensual por usuario[\[Cho10\]](#). Este precio dependerá del número de usuarios que vayan a usar la aplicación y puede reajustarse si se añaden o quitan usuarios.

Algunos ejemplos de software como servicio son *Google Apps*, *innkeypos*, *Quickbooks Online*, *Successfactors Bizzx*, *Limelight Video Platform*, *Salesforce.com* y *Microsoft Office 365*.

2.6. Otros modelos de servicio especiales

A parte de los tres modelos de servicio fundamentales, existen otros modelos menos relevantes que han surgido con el fin de resolver problemas muy puntuales. En general, estos modelos secundarios han sido creados al mezclar dos de los modelos fundamentales.

2.6.1. Escritorio como servicio

Del inglés *Desktop as a Service* (DaaS), también conocido como escritorio virtual, en el se ofrece al cliente un escritorio con todas las aplicaciones pertinentemente instaladas para ser ejecutadas a su gusto.

Este escritorio puede ser usado en paralelo por múltiples usuarios sin que las acciones de uno afecte sobre el resto. Así cuando el usuario inicia la sesión su configuración personal se carga en el escritorio, así como toda la información previa que el usuario había dejado guardada en él. De igual forma, cuando el usuario se da de baja en la sesión se almacena la información y la configuración que el usuario hubiera dejado antes de cerrar su sesión.

2.6.2. Entorno de pruebas como servicio

Del inglés Test environment as a service(TEaaS) , este modelo ofrece al cliente un entorno en el que poder realizar sus pruebas sobre un determinado software.

Un ejemplo de entorno de pruebas como servicio viene dado por testersdesk.com.

2.6.3. Identidad como servicio (IDaaS)

El establecimiento y la prueba de la identidad es una función de red central. Un servicio de identidad almacena la información asociada con una entidad digital de forma que pueda ser consultado y gestionado para uso en las transacciones electrónicas. los servicios de identidad tiene como funciones principales las de almacenamiento de datos, motor de búsqueda y control de la integridad de dichos datos.

Ya se trate de la protección del tráfico de red, acceso a recursos privilegiados, o algún otro derecho o privilegio definido, la autorización valida de un objeto en función de su identidad es el elemento central del diseño de una red segura.

De esta forma la identidad como servicio permite gestionar como y en que medida debe ser gestionada los datos enviados y recibidos por un determinado usuario.[[Sos11](#)]

2.7. Tipos

De cara a que grupo de clientes se encuentre destinados los servicios y donde se encuentren los servidores podemos distinguir tres tipo de computación en la nube.

2.7.1. Cloud público

Es el tipo más común y el que mejor representa la idea de computación en la nube. En este tipo, los servicios están destinados al público en general y suelen ser gratuitos o bien ligados a un modelo de “pago por uso”.

Este modelo es el mejor para aquellos negocios donde es necesario:

- controlar los picos de carga.
- dirigir las aplicaciones del modelo de software como servicio.
- dirigir las aplicaciones que son utilizadas por muchos usuarios que de otra manera necesitarían una gran inversión en infraestructuras para sus negocios.
- usar un servicio compartido por un número elevado de usuarios.
- utilizar herramientas para el testeo y desarrollo de una aplicación.
- utilizar un modelo de software como servicio (SaaS) con una buena estrategia de seguridad.
- una capacidad de almacenamiento incremental. La capacidad que va a necesitar no se puede aproximar desde el principio.
- trabajar en un proyecto colaborativo.
- desarrollar una aplicación utilizando un modelo de plataforma como servicio (PaaS). [[HBKH09](#)]

Las principales ventajas que ofrece este tipo son la gran flexibilidad, la seguridad y el ahorro en su infraestructura. Además no requiere grandes inversiones, es escalable y no tiene límite de recursos ni de disponibilidad. Algunos ejemplos de cloud públicos son el de Amazon, el de Microsoft o el de Yahoo.[[Tec11](#), [Gen08](#)].

2.7.2. Cloud privado

Es el segundo tipo de computación en la nube más utilizado. En general se acepta que cualquier centro de datos propiedad de una organización y destinado a su uso exclusivo es considerado cloud privado. Este cloud debe tener cierto nivel de virtualización y autonomía.

En general los proveedores de clouds privados suelen ofrecer infraestructuras a grandes empresas para que pueda residir en ellas sus centros de datos. Por ello los *clouds* privados son el tipo que debe prestar una mayor atención a la seguridad y mantenimiento de los datos que manejan, lo que hace que los clouds privados resulten algo más costosos que el resto de tipos.

Las principales ventajas que ofrece el cloud privado son aislamiento y exclusividad de su propia infraestructura, así como máximas garantías de seguridad y privacidad en sus sistemas de información. [Tec11, Gen08, Inf10, Haf09].

Un cliente suele usar un cloud privado cuando :

- su negocio se basa en su información y sus aplicaciones. Por lo tanto, el control y la seguridad son primordiales.
- su negocio está relacionado con la gestión de la información de otras empresas.
- su empresa es lo suficientemente grande como para poder mantener un sistema en la nube propio eficiente.[HBKH09]

2.7.3. Cloud híbrido

En este tipo se mezclan las características más importantes del cloud privado y público, de manera que el cliente gestiona y explota en exclusividad su infraestructura pero dispone de acceso al cloud público que mantiene el proveedor en sus instalaciones. El cloud híbrido combina los beneficios del cloud público junto con los del cloud privado . Así por un lado extrae del cloud privado la exclusividad de tener un servidor propio mientras que por el otro extrae del cloud público la posibilidad de aumentar sus recursos obteniéndolos de la nube a modo de ampliación y adaptación de su arquitectura.

Dentro del cloud híbrido existen dos submodelos :

- Cloud Privado "inHouse" : El cliente pone las máquinas y las instalaciones donde situarlas para montar su cloud, es decir, la infraestructura se encuentra físicamente en las instalaciones del cliente. Este modelo requiere una importante inversión por parte del cliente, tanto económica como de recursos, y la seguridad de la información es relativa a la experiencia y capacidad técnica disponible.

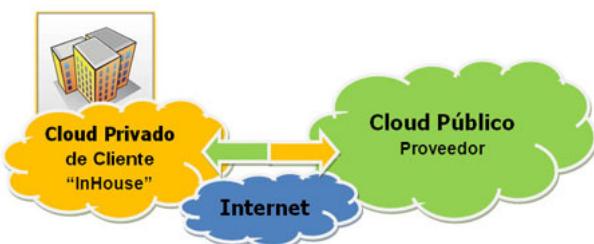


FIGURA 2.1: Cloud privado “inHouse”

- Tanto Cloud Privado como Cloud Público, en *Datacenter* del proveedor : La infraestructura del cliente puede residir en el centro de datos del proveedor o bien puede subcontratarse, lo que hace desaparecer la necesidad de una inversión inicial. Esto permite reducir costes al modelo de pago por uso, y escalar de forma rápida y flexible y sin inversión extra. [Tec11, Ste12, Gen08]

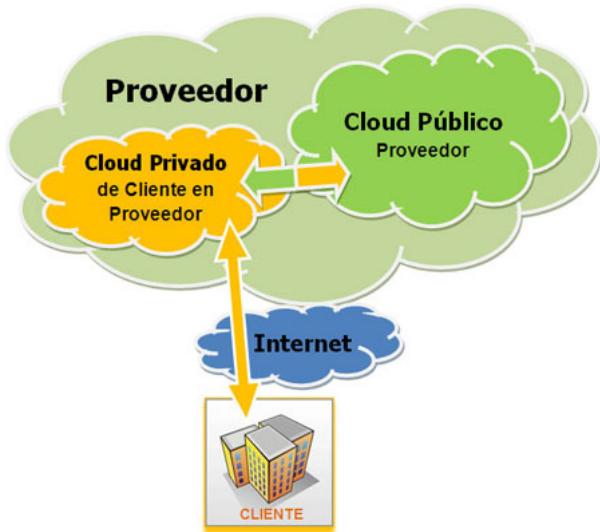


FIGURA 2.2: Cloud híbrido en el centro de proceso de datos del proveedor

Un cliente suele usar un cloud híbrido cuando:

- su compañía sigue una organización privada pero delega ciertas funcionalidades en servicios públicos. Por ejemplo una empresa mantiene una infraestructura privada para el desarrollo de sus aplicaciones pero utiliza servicios públicos para almacenar información.
- tiene una empresa que usa un modelo de software como servicio pero necesita cierta infraestructura privada para almacenar información que necesita unos requisitos de seguridad muy elevados. [HBKH09]

2.7.4. Cloud comunitario

En el caso de este tipo de cloud, dos o más organizaciones forman una alianza y comparten una infraestructura con un marco de seguridad y privacidad común. Esto es posible ya que todas las empresas aliadas tienen unos objetivos comunes y requieren de los mismos recursos y herramientas.

De esta forma, las empresas participantes pueden beneficiarse de una exclusividad casi total reduciendo el coste de compra y de mantenimiento de servidores.

El principal inconveniente de este modelo es la pérdida del nivel de seguridad característico de los clouds privados, ya que este pasa a ser dependiente del anfitrión de la infraestructura. [Tec11, Gen08]

2.8. Ventajas e inconvenientes

Tal y como parece el *Cloud Computing* ofrece a los usuarios una nueva forma de entender. Queda claro que las ventajas que ofrece esta computación hará posible que en un futuro no muy lejano la idea de utilizar un conjunto de servicios no residentes en nuestros ordenadores personales será aceptado como algo común entre los usuarios. Sin embargo, aunque las ventajas que ofrece este sistema con respecto a los anteriores lo convierta en actualmente la mejor opción no hay que olvidar que el *Cloud Computing* cuenta con un reducido conjunto de desventajas. A continuación detallaremos las ventajas y desventajas más relevantes de la computación en la nube:

2.8.1. Ventajas

Entre las ventajas más importantes del *Cloud Computing* podemos mencionar:

- **Accesibilidad**, que suele ser uno de los principales motivos por el cual se acaba eligiendo este sistema frente al resto. El uso del *Cloud Computing* permite que todos los empleados puedan acceder a los datos de la organización no importando el lugar en el que se encuentren de forma fácil y rápida.
- **Reducción de gastos en infraestructura, mantenimiento y servicios**, mediante este tipo de tecnología se reduce los gastos de adquisición y mantenimiento de los servidores sobre los que se ejecutan las aplicaciones.
- **Escalabilidad**, los servidores son capaces de proveer recursos, software y/o datos en función de la demanda necesaria en cada momento, sin que el usuario tenga conocimiento acerca de los servicios que le son proporcionados.
- **Seguridad** a la hora de mantener la integridad de los datos, si bien la cosa depende del cloud que se utilice.
- **Integración probada de servicios**. Las aplicaciones en la nube pueden ser adaptadas con gran rapidez y facilidad al resto de aplicaciones empresariales.[[Sal11](#)]
- **Prestación de servicios a nivel mundial**. Independientemente donde se encuentre la infraestructura, los servicios que presta cierta organización pueden utilizarse en cualquier parte del mundo de manera rápida y sencilla.
- **Actualizaciones automáticas** . En sistemas ajenos al *Cloud Computing* el usuario debe dedicar tiempo y esfuerzo en volver a personalizar e integrar la aplicación cada vez que se realice una actualización de la misma. Con la computación en la nube estas actualizaciones se realizan de manera automática y sin afectar al cliente.

2.8.2. Desventajas

El *Cloud Computing* también tiene sus inconvenientes. Las principales desventajas a destacar son :

- Privacidad. Es más una cuestión de confianza de los usuarios que un problema de la computación en la nube en sí. Al estar almacenados los datos de los usuarios en equipos externos, este teme que otros puedan acceder fácilmente a su información.
- La gran dependencia del proveedor de servicios en la nube. Esta desventaja reside en que el caso de optar por un sistema en la nube no privado el usuario pasa a depender de su proveedor para desarrollar total o parcialmente su actividad. [[Joh08](#)]
- La disponibilidad de los servicios está estrechamente ligada al correcto funcionamiento de la red.
- La disponibilidad de herramientas especializadas podría tardar un largo periodo de tiempo hasta que sea factible su utilización en la nube. Es decir, existen ciertas necesidades que , a día de hoy , la nube no puede cubrir.
- La escalabilidad debe planificarse con cuidado. A medida que más usuarios empiecen a compartir la infraestructura de la nube, los servidores se encontrarán más sobrecargados lo cual puede provocar importantes degradaciones del servicio en el caso de que la organización no haya realizado las previsiones oportunas.

2.9. El *Cloud Computing* como un estándar abierto

Si tenemos en cuenta el desarrollo de la computación en nube hasta la fecha, es evidente que esta tecnología es el resultado de la convergencia de muchas normas diferentes. El crecimiento del *Cloud Computing* ha dado lugar a un importante impulso por parte de la industria para crear estándares abiertos basados en la nube.

En la actualidad, el *Cloud Computing* se rige por los siguientes estandares tecnológicos:

- **Virtualización de la plataforma de recursos.** Consiste fundamentalmente en la abstracción de un sistema hardware completo permitiendo que diversas instancias de sistemas operativos corran sobre él.
- **Arquitectura orientada a servicios.** Permite la creación de sistemas de información altamente escalables que reflejan el negocio de una organización y a su vez brinda una forma bien definida de exposición e invocación de servicios, lo cual facilita la interacción entre diferentes sistemas propios o de terceros.
- **FrameWork orientado a aplicaciones web.** Consiste en un conjunto de conceptos, prácticas y criterios para enfocar un tipo de problemática en las aplicaciones web que sirva como referencia, para enfrentar y resolver nuevos problemas de índole similar.
- **Despliegue de software de código abierto.** Basado en la idea del desarrollo de aplicaciones de código abierto que puedan ser utilizadas y ampliadas por una comunidad de usuarios.
- **Servicios de web estandarizados.** Consiste en el uso de servicios web que sigan los estándares establecidos con la finalidad de facilitar la incorporación de dichos servicios en aplicaciones.
- **Sistemas autónomos.** Se define como un grupo de redes IP que poseen una política de rutas propia e independiente. Un sistema autónomo realiza su propia gestión del tráfico que fluye entre él y los restantes Sistemas Autónomos que forman Internet.
- **Computación Grid .** Permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado.

Estos estandares permiten ajustar los diferentes modelos de negocio que los proveedores de servicios en la nube pueden ofrecer. Para conseguir que los datos puedan ser portables y accesibles universalmente es necesario que estos estandares sean abiertos.

La carrera para crear la primera generación de tecnologías abiertas en la nube que competirán con tecnologías patentadas que ofrecen las empresas como Microsoft (Azure Platform) y VMware (vSphere) ya está en marcha. Algunos ejemplos de ello son:

- Rackspace.com, uno de los grandes proveedores de servicios cloud IaaS, anunció en julio de 2010 que está iniciando un proyecto de código abierto llamada OpenStack. Este software desarrollado se distribuye bajo la licencia Apache 2.0.
- La NASA también ha donado parte de la tecnología Nebula Cloud Platform que desarrolló. [Sos11]

Capítulo 3

Virtualización

3.1. Introducción

En este capítulo veremos como puede ser virtualizado un sistema, que alternativas hay y cuales han sido las seleccionadas para el proyecto.

La virtualización nos permite tener varios sistemas en la misma máquina, aislados unos de otros. Gracias a que esos sistemas pueden estar en un único archivo, podríamos cambiar alguno de ellos a otra máquina o hacer una copia de seguridad de forma rápida y sencilla, solo habría que copiar ese archivo al nuevo servidor o a un dispositivo de almacenamiento. Para conseguir esto, lo que tenemos es un hipervisor que nos permite gestionar esos recursos y asignárselos a las máquinas según los van necesitando. Este termino es una evolución de otro que se usa para referirse al *kernel* de los sistemas operativos, el supervisor. Esta gestión puede ser, en el caso de dispositivos de almacenamiento, la división de ese espacio según sea asignado por el usuario o, en el caso de dispositivos de tratamiento de datos, como puede ser el procesador, asignándole un tiempo determinado de ejecución a cada sistema virtualizado.

3.2. Historia

La virtualización, aunque pueda parecerlo, no es nueva, ya se usaba en la década de los 1960, antes incluso de los ordenadores personal.

IBM, en 1964, desarrolló un sistema operativo que gestionaba los recursos de la máquina, llamado VM (*Virtual Machine*, Máquina Virtual), que permitía a varios usuarios usar una misma máquina a la vez, dando la apariencia de que eran ordenadores personales independientes. Este desarrolló continuó en los dando soporte a los sistemas operativos que ya tenía IBM para que pudiesen ejecutarse sobre VM, como son *PC/DOS*, *MVS* y *DOS/VSE* e incluso el sistema *Unix*.

A raíz de que se popularizó el uso de computadoras personales, en los años 80s, la virtualización quedó relegada. Hasta que en 1999 VMware sacó el primer producto que permitía la virtualización sobre la arquitectura x86, la mas extendida en el mercado de los ordenadores personales. Luego, en 2003 se inició el proyecto de software libre Xen, en la universidad de Cambridge y en 2007 KVM, basado en el kernel de Linux para usarlo como hipervisor.

En 2005, la industria de los procesadores, sobre todo Intel y AMD, empezaron a desarrollar, independientemente, tecnologías en sus procesador que permitían una virtualización más sencilla y eficiente para los programas de virtualización.

3.3. Distintos esquemas de virtualización

El tipo de virtualización depende de como quién gestiona los recursos de la máquina y como se comunica el sistema virtualizado con esos recursos.

3.3.1. Virtualización de hardware

En lo que se basa es en que sobre el hardware de la máquina se simula el hardware mediante máquinas virtuales, para luego poder ejecutar el sistema. El gran inconveniente de este esquema es que las instrucciones de la máquina virtualizada debe traducirse a instrucciones de la máquina *host*, lo que requiere mucho tiempo.



FIGURA 3.1: Virtualización hardware

3.3.2. Virtualización de software

En este esquema se utiliza un software para simular el ordenador, una máquina virtual, este debe ejecutarse sobre un sistema operativo concreto, que es el que tiene acceso al hardware y hace de intermediario entre este y la máquina virtual.



FIGURA 3.2: Virtualización software

En este caso en el sistema *guest*, el hardware de almacenamiento, como puede ser el disco duro o las unidades ópticas, pueden ser virtuales o reales. Si son virtuales, es la máquina virtual la que se encarga de gestionarlo mediante archivos en el sistema *host*, en el caso de que sean reales, la máquina virtual, en colaboración con el sistema *host*, le proporcionará al sistema virtualizado acceso a los mismos. El hardware que no sea posible virtualizar, como pueden ser una cámara o un micrófono, debe usar el hardware real, para ello la máquina virtual, transforma las llamadas del sistema *guest*, en llamadas al sistema *host*.

El mayor inconveniente de este esquema es que, como debe haber un sistema operativo ejecutándose, consumido recursos de la máquina.

3.3.3. Paravirtualización

En este esquema, al contrario que en el anterior, no es necesario de un sistema *host* para gestionar los recursos, sino que lo que se usa es una capa de virtualización, también llamada hipervisor, que es la que gestiona los recursos. Pero no basta con eso, sino que debe haber una comunicación entre el sistema *guest* y el hipervisor.

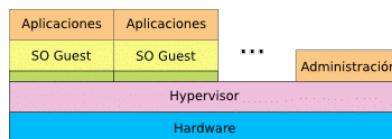


FIGURA 3.3: Paravirtualización

Esa comunicación se debe a que en los procesadores hay varios niveles de privilegios, llamados *ring*, en concreto tienen 4 niveles, de ellos el 0 está reservado para el *kernel* del sistema operativo. Por lo que para que se pueda ejecutar el hipervisor, habría que modificar el *kernel* para que se ejecutase en el nivel 1 y dejar el 0 para el hipervisor. Esto no es ningún problema para sistemas

operativos en los que el código fuente esté disponible y modificarlo, como pueden ser *Linux*, pero en sistemas privativos como *Microsoft Windows* o *Mac OSX*, no es posible hacer este cambio.

Aunque algunos programas de virtualización que usando esta técnica han conseguido ejecutar sistemas operativos sin modificar, para ello lo que hacen es, en tiempo de ejecución, detectar las instrucciones problemáticas, las que requieren de privilegios de nivel 0, y parchearlas, para que no haya problemas en el funcionamiento del sistema operativo, ni que el procesador se queje al intentar ejecutar una instrucción en un nivel inadecuado.

3.3.4. Virtualización completa

Este esquema es parecido al anterior, también existe el hipervisor, pero al contrario que en la paravirtualización, no es necesaria una comunicación. Esto es porque en los procesadores que mayor cuota de mercado tiene (Intel y AMD) se ha añadido dos modos de ejecución, uno de ellos reservado para el hipervisor y el otro mantiene el uso que tenía, es decir, reservando el nivel 0 para el kernel, por ello no es necesario modificar alguna del sistema *guest*. Estos modos son VMX para Intel y SVM para AMD

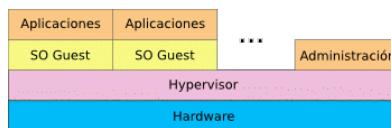


FIGURA 3.4: Virtualización completa

3.4. Software de virtualización

Existen muchos programas de virtualización para todos los tipos como pueden ser VMware en sus distintas versiones, Xen, KVM, VirtualBox... Lo que buscamos en este proyecto es poder tener sistemas bajo demanda a bajo coste, por lo que nos centraremos en evaluar el software libre, *open source, freeware*, con lo queda descartado como opción VMware y aunque VirtualBox tiene una versión como software libre, algunas funciones como el soporte USB, está bajo una licencia propietario, por lo que también lo descartamos.

Aunque siguen existiendo muchas alternativas bajo las condiciones que queremos, nos centramos principalmente en Xen y KVM, por ser los que mayor trayectoria tienen. Debido a que Xen lleva mas años que KVM y estar apoyado por la comunidad, a parte de grandes empresas que apoyan el desarrollo como IBM, Intel, Red Hat u Oracle, es la mejor opción que podíamos escoger.

3.5. Xen

Xen permite una virtualización de paravirtualización y virtualización completa en los procesadores que lo soporten.

Pero no todo en Xen son ventajas, ya que en el caso de que solo se pueda usar paravirtualización en una máquina solo podrán virtualizar los sistemas operativos que soporten el hipervisor de Xen. Además para que el usuario pueda interactuar con Xen debe ejecutar, sobre el hipervisor, un sistema operativo *guest* que tiene más privilegios que los demás sistemas que puedan ejecutarse sobre Xen.

Para saber como funciona Xen hay que como gestiona cada uno de los recursos. Eso es lo que vamos a ver a continuación:

3.5.1. Dom0, DomU.

El nombre con el que se denominan los sistemas *guest* en Xen es dominio (*domain*, en inglés). Existen dos tipos de dominios diferenciados, un es el *dom0*, que es el sistema privilegiado con el que se administra Xen y puede controlar los distintos dominios que se estén ejecutando, solo puede haber uno. El otro es el *domU* es como se denomina a un sistema *guest*.

El nombre de un dominio es como identifica Xen a cada uno de los sistemas, por lo que debe ser único. Puede modificarse en el archivo de configuración con el que se arranca el *guest*.

3.5.2. Interrupciones

Una interrupción se puede ver como una llama de atención del hardware e indica que ha ocurrido algo, un timer ha finalizado su cuenta, se ha pulsado un botón del ratón... Aunque una interrupción requiere de una acción inmediata, lo que hace Xen es interceptarla y planificarla, así cuando el *guest* vuelva a tener acceso al procesador la podrá tratar con el manejador correspondiente. Este permite que cuando llega una interrupción, no haya que espera a que todos los *guest* la procesen, lo que sería muy costoso, por los cambios de dominios necesarios.

3.5.3. CPU

La CPU es el recurso más utilizado, ya que es el que ejecuta los programas. Xen trata a los distintos dominios como un sistema operativo trataría a los programas, es decir, tiene un planificador en el que a cada dominio les deja cierto tiempo para que puedan hacer sus cálculos, también debe garantizar que a todos los dominios les deje usar la CPU.

3.5.4. Memoria

El hipervisor, como gestor de los recursos de la máquina, tiene acceso a toda la memoria, la cual debe alojar toda la memoria usada por los distintos dominios, pero solo la memoria física y la tabla de página, ya que del resto, como la memoria virtual, se encarga cada domU. La memoria virtual permite que los procesos accedan a la memoria como si no hubiese nadie más, es decir, acceso relativo, y luego se le suma la dirección donde empieza su espacio de memoria.

Como la asociación entre memoria física y virtual se hace a través de las tablas de página, que son gestionadas por Xen, le permite tener un control de que dominio accede a que partes de la memoria, ya que puede denegar el acceso a una página si no le pertenece al dominio que se la ha pedido. Este permite aislamiento en la memoria, es decir, un dominio solo puede acceder a la memoria que le ha asignado Xen y le protege de que otro dominio modifique la memoria del primero.

3.5.5. Dispositivos de entrada/salida

Ningún domU tiene gestión sobre los dispositivos, de esto se encarga el hipervisor con ayuda de dom0. Xen maneja los dispositivos de E/S mediante dispositivos virtuales, que son enlaces entre las interfaces de los domUs y el dispositivo en el dom0.

El acceso a los dispositivos se hace mediante un buffer en anillo, así los dominios pueden acceder al dispositivo directamente sin intervención del hipervisor.

3.5.6. Red

Como cualquier otro dispositivo, Xen proporciona una interfaz de red virtual a los dominios, además de algunas funciones para mover paquetes de la interfaz virtual del dominio a una interfaz virtual del dom0. El hipervisor en este caso solo mueve los datos de la interfaz física a la virtual, y de la virtual a la física cuando el dominio lo pida, pero no valida paquetes, eso lo manejan reglas en el dom0.

3.5.7. Dispositivos de bloque

Son dispositivos de almacenamiento en los que solo se puede leer y escribir en bloques de bytes, el tamaño del bloque depende del dispositivo y su configuración, estos pueden discos duros, CD-ROMs o memorias USB.

Xen gestiona los dispositivos de bloque casi como los dispositivos de red. El hipervisor exporta el dispositivo virtual de bloques al domU y proporciona al dom0 la capacidad de mapear la funcionalidad del dispositivo real en el virtual.

3.6. KVM

KVM (Kernel-based Virtual Machine) es un hipervisor tipo 1 que proporciona una solución completa de virtualización para el sistema operativo Linux en arquitecturas x86 que cuenten con las extensiones de virtualización Intel VT o AMD-V.

En realidad, el término más acertado sobre KVM se refiere al nivel del kernel donde se aloja la funcionalidad de virtualización, pero en la práctica es más utilizado para hacer referencia al componente en el que trabaja el usuario.

Usando KVM, se puede ejecutar múltiples máquinas virtuales sin necesidad de modificar las imágenes de linux o de windows. Cada máquina virtual tiene su propio hardware de virtualización privado : su propia tarjeta de red, su disco, su tarjeta gráfica etc.

El componente KVM para el kernel viene incluido en linux desde la versión 2.6.20. Cuando se carga el modulo KVM, Linux se convierte en un hipervisor "bare-metal" (tipo 1) capaz de ejecutar varias máquinas virtuales aisladas. KVM hospeda las máquinas virtuales como procesos, por lo que cada máquina virtual puede beneficiarse de todas las características del kernel de Linux, incluyendo todas aquellas referentes al hardware, la seguridad, el almacenamiento etc.

3.6.1. Breve historia de KVM

KVM se inició como un proyecto Open Source por la empresa israelí Qumranet. Su principal objetivo era la de dar una infraestructura de escritorio virtual (VDI) en entornos windows.

En el año 2007, KVM se integró como un módulo en la rama principal del kernel de Linux, convirtiéndose desde entonces en el hipervisor de virtualización oficial para linux.

En 2008, Red Hat adquiría Qumranet y con RHEL 6 (Red Hat Enterprise Linux) centraba su estrategia en convertir KVM en la mejor solución de virtualización en el mundo Open Source, desechariendo la solución de virtualización con Xen utilizada en RHEL 5.

Poco después IBM también apostó por KVM, contribuyendo en áreas como la gestión de memoria, mejoras del rendimiento y el subsistema de entrada/salida virtual.

Actualmente existen muchos sistemas de virtualización tales como Xen, Bochs, UML, Linux VServer, y coLinux, sin embargo, KVM está adquiriendo una mayor presencia en el mercado.

3.6.2. Características de KVM

Las principales características de KVM son:

- **Seguridad** : KVM se aprovecha de los modelos de seguridad estándar de Linux: SELinux y AppArmor. Estos modelos proporcionan el aislamiento y el control de recursos necesarios. El proyecto SVirt, un esfuerzo de la comunidad para integrar el control de acceso obligatorio (MAC) con KVM, está construido sobre SELinux y proporciona una infraestructura que permite al administrador definir políticas de aislamiento para las máquinas virtuales. Básicamente SVirt se asegura que los recursos de una máquina virtual no puedan ser accedidos por otro proceso o máquina virtual.
- **Gestión de memoria** : La memoria utilizada por una máquina virtual se gestionará de la misma forma que la de otro proceso, podrá ser guardada en disco (swapped) y utilizada en páginas grandes (large pages). El soporte NUMA de Linux, permite también el uso por parte de las máquinas virtuales de grandes cantidades de memoria. Además, KVM soporta las últimas características de virtualización en memoria proporcionadas por fabricantes como EPT (Extended Page Table de Intel) o RVI (Rapid Virtualization Indexing de AMD). La compartición de páginas de memoria se lleva a cabo por medio de la herramienta integrada en linux KSM (Kernel Same-page Merging).
- **Almacenamiento** : KVM utiliza cualquier tipo de almacenamiento soportado por linux para el almacenamiento de imágenes de las máquinas virtuales. Esto incluye discos locales (IDE/SCSI/SATA), NAS (Network Attached Storage) o SAN (iSCSI y Fibre Channel). KVM también soporta el almacenamiento en sistemas de ficheros distribuidos tales como GFS2,

OCFS o ClusterFS.

Las imágenes de disco utilizadas, soportan aprovisionamiento bajo demanda evitando tener que reservar todo el espacio inicialmente. El formato nativo de KVM es QCOW2, el cual permite la realización de compresión y cifrado.

Con respecto al tipo de almacenamiento podemos destacar dos clases que pueden ser gestionadas por KVM:

- **Almacenamiento local** : Es el almacenamiento que se produce al conectar directamente al host algún dispositivo de almacenamiento. Resulta muy útil para entornos de desarrollo, pruebas y pequeños despliegues aunque no se recomienda para entornos con muchas máquinas virtuales o que requieran de una migración en caliente.
- **Almacenamiento en red** : Se refiere a los dispositivos de almacenamiento que se comparten en una red o a partir de protocolos estándares.
- **Migración en vivo** : KVM permite migraciones en caliente (live migrations). Estas migraciones permiten mover una máquina virtual en ejecución entre servidores físicos (hipervisores) sin intervención del servicio. Estas migraciones son transparentes al usuario ya que la máquina permanece encendida, las conexiones de red activas y las aplicaciones en ejecución mientras la máquina se realoca en otro servidor físico.
- **Drivers** : KVM soporta virtualización híbrida. En los sistemas operativos invitados hay que instalar drivers paravirtualizados que permiten utilizar una interfaz de entrada/salida para dispositivos de bloques y dispositivos de red. El hipervisor KVM utiliza el estándar VirtIO. VirtIO es un estándar de drivers paravirtualizados desarrollado por IBM y Red Hat con la ayuda de la comunidad Linux. El objetivo principal de estos drivers es conseguir una mayor interoperabilidad con los invitados.
- **Rendimiento y escalabilidad** : KVM posee los mismos rasgos de rendimiento y escalabilidad que caracterizan a Linux. Por ello soporta máquinas virtuales de hasta 16 CPUs virtuales y 256 GB de Ram, pudiéndose conseguir ratios de hasta 600 máquinas virtuales en un solo servidor físico.

3.6.3. Requisitos Hardware

KVM está diseñado en torno a las características de virtualización de hardware incluidas en los ordenadores AMD (AMD-V) e Intel (VT-x). Es compatible con funciones de virtualización de chipsets y dispositivos PCI, como la E/S de la unidad de memoria (IOMMU) y la E/S del sistema simple de virtualización root (SR-IOV).

Por ello KVM no podrá ejecutarse si el ordenador no puede soportar la virtualización hardware o si la misma no se encuentra activada en la bios.

En cuanto al rendimiento, es necesario tener en cuenta que el servidor deberá ser capaz de afrontar el incremento de memoria RAM que supone la conexión de cada huésped, y tener al menos un núcleo del procesador o un hilo para cada uno.

3.6.4. El paquete KVM

El paquete de KVM contiene el programa qemu-kvm, el cual se encarga de realizar la emulación. Este programa no es más que una modificación de la aplicación QEMU también integrada en linux.

Además de este programa el paquete también incluye una utilidad de monitorización a nivel de depuración(kvm-sat), componentes de firmware, archivos de correlación de teclas, scripts y algunos drivers para realizar la paravirtualización en windows, aunque se desaconseja el uso de estos últimos por estar bastante desactualizados.

Originariamente, el paquete de KVM también contaba con los módulos del kernel de KVM. Sin embargo, en la actualidad estos módulos se encuentran ya incluidos en el kernel y el paquete solo proporciona los componentes de usuario.

Además de las herramientas incluidas en el paquete de KVM es recomendable la instalación y uso de otras herramientas secundarias que facilitan la gestión del sistema. Todas estas herramientas son proporcionadas por los paquetes que llevan su nombre. Algunos de estos paquetes son:

- Libvirt: este paquete incluye la herramienta libvirt que permite la creación, eliminación y gestión en general de las diferentes máquinas virtuales. Además en este paquete también se incluyen herramientas de gestión de redes virtuales y almacenamiento.
- Virt-manager (Virtual Machine Manager): Incluye una herramienta gráfica de administración de los huéspedes de las máquinas virtuales.
- Vm-install: configura una nueva maquina virtual e instala un sistema operativo en la misma.
- Virt-viewer: Incluye una herramienta que permite interactuar con la virtualización de un determinado huésped. Utiliza libvirt y pretende ser un reemplazo para los clientes de VNC y SPICE.

3.6.5. Ventajas de KVM

La virtualización con KVM ofrece las siguientes ventajas:

- Esta diseñada para procesadores x86, centrada en una virtualización total.
- No se modifica el kernel de GNU/Linux.
- Está basada en un módulo que no necesita ninguna clase de parches.
- Contiene soporte para la paravirtualización.
- Funciona en todo tipo de máquinas, servidores, escritorios o laptop actuales.
- Permite una migración en caliente de las máquinas virtuales.
- Permite una administración vía web, gráfica y consola.
- Las máquinas serán reconocidas como procesos, facilitándose así la administración de las mismas.
- Maneja 3 tipos de configuración de red, Bridge, Route y NAT.
- Permite ejecutar múltiples máquinas virtuales, cada una con su propia instancia.

3.6.6. Desventajas de KVM

La virtualización con KVM ofrece también las siguientes desventajas:

- Es un proyecto aun muy joven. la reciente aparición de KVM en el mercado hace que no disponga de toda la experiencia presente en otros modelos de virtualización.
- No hay herramientas muy sofisticadas para la gestión de servidores y de maquinas virtuales.
- Puede mejorar mucho más en áreas como el soporte de redes virtuales, el soporte de almacenamiento virtual, la seguridad, la alta disponibilidad, la tolerancia a fallos, la gestión de energía y el soporte en tiempo real.

3.6.7. Limitaciones de KVM

Aunque las máquinas virtuales se comportan casi como máquinas físicas existen ciertas limitaciones. Estas limitaciones afectan tanto al huésped alojado en la máquina virtual como al servidor que la gestiona. Las principales limitaciones que podemos encontrar en la virtualización con KVM son:

- OverCommits : KVM permite un overcommit tanto en la memoria como en el espacio de disco. Esto permite aumentar el número de usuarios conectados a un determinado servidor. Sin embargo, esto también provoca errores permanentes que pueden acarrear fallos en la conexión de los huéspedes. También es posible un overcommit en la CPU, lo cual provoca un empeoramiento del rendimiento.
- Tiempo de sincronización : Generalmente los usuarios requieren de un soporte adicional para poder utilizar de manera adecuada el tiempo de sincronización. Este soporte adicional puede comportarse de manera contraproducente y empeorar el rendimiento.
- Direcciones Mac : Si no existe una dirección Mac asociada a una tarjeta de red, kvm asignará una por defecto. Esto puede dar lugar a problemas en la red cuando más de una tarjeta de red recibe la misma dirección Mac. Se recomienda usar siempre una dirección Mac única.
- Migración en vivo : La migración en vivo es sólo posible entre los servidores host de máquinas virtuales con la misma CPU. El almacenamiento de la información de un usuario debe ser accesible desde los diferentes servidores, se encuentren o no en la misma CPU.
- Permisos de usuario : Las diferentes herramientas de gestión deben ser autenticadas por libvirt. Por ello para que un usuario pueda invocar a qemu-kvm desde linea de comandos, el usuario debe ser miembro del grupo de KVM.
- Suspensión e hibernación del servidor de máquinas virtuales : No se permite la posibilidad de suspender o hibernar el servidor mientras existan usuarios conectados al mismo.

3.6.8. Red en KVM

Como ya mencionamos en el apartado de ventajas, KVM dispone de tres tipos de configuración de red :

- Bridge : Este tipo de configuración permite crear un puente entre las tarjetas de red física y las virtuales, permitiendo al usuario conectarse a la red remota como si fueran otras máquinas que estuvieran en la red local. La conexión es directa. Por ello habrá que asignar IPs a nuestras máquinas virtuales o tomar la dirección por DHCP si en la red donde está conectado el anfitrión hay un servidor DHCP.
- NAT : Este tipo de configuración permite tener máquinas virtuales dentro de una red virtual, configurando de manera especial su conexión a la tarjeta de red, lo cual ayuda a tener una mayor seguridad en las máquinas virtuales.
- Route : En este modo el switch virtual se conecta a la LAN física del anfitrión, pasando todo el tráfico entre anfitrión y las máquinas virtuales sin el uso de NAT. Todas las máquinas virtuales se encuentran en su propia subred, enrutasadas a través del switch virtual. Desde fuera solo se puede acceder a las máquinas virtuales si se definen reglas de enruteamiento.

3.7. Xen frente a KVM

En este apartado realizaremos una comparación entre Xen y KVM, extrayendo los beneficios y defectos de cada uno de ellos y justificando las razones que nos han llevado a decantarnos por KVM.

3.7.1. Integración en el kernel

KVM es el único hipervisor de los dos que a día de hoy se encuentra completamente integrado en el kernel de linux. Por ello no es necesario realizar ninguna acción especial para poder utilizar todos los servicios que KVM nos ofrece. KVM utiliza linux para todo, desde la gestión de dispositivos y el intercambio de páginas de memoria hasta el uso de algoritmos para optimizar al máximo la gestión de los núcleos y el control de virtualización de todo el sistema.

Sin embargo, Xen todavía no ha sido aceptado completamente en el núcleo de linux, no pudiendo aprovechar las tecnologías disponibles en este sistema operativo, tales como los planificadores CFS, la paginación o la sobre explotación de memoria con KSM.

Además todavía es necesario la instalación de Xen desde cero, con la creación de una máquina virtual especial llamada Dom0, que se encargue de la gestión tanto del sistema como de todo el conjunto de dominios que cuelguen de él.

En resumen, el hecho de que KVM se encuentre integrado completamente en el kernel de linux le confiere una serie de beneficios contra los que Xen no puede enfrentarse y necesita parchear de forma externa.

3.7.2. Rendimiento

Con respecto al rendimiento KVM y Xen ofrecen beneficios en aspectos muy diferentes, lo que hace que en este punto no pueda considerarse uno mejor que el otro en todas las tareas posibles, haciendo que sea más indicado uno u otro dependiendo de en que puntos se va a basar el sistema. Así las máquinas virtuales de Xen presentan un mejor rendimiento con aplicaciones de cálculo que las máquinas virtuales de KVM, mientras que para las pruebas de escritura en disco, el rendimiento de las máquinas virtuales de KVM es mejor que el de las máquinas virtuales de Xen.

Para llegar a estas conclusiones se realizaron pruebas de rendimiento usando los benchmarks de Linpack, el cual se utiliza para conocer el rendimiento del sistema en cuanto a cálculo, e IOZone, el cual se utiliza para conocer la medida de rendimiento del disco en los procesos de lectura y escritura. En el primero se experimentó una perdida del 1% del rendimiento con respecto al anfitrión para el hipervisor de Xen, mientras que las perdidas en el caso de KVM alcanzaron el 36%.

Con respecto a la segunda prueba, el hipervisor de KVM presento solo un 3.6% de perdida de rendimiento para escrituras en disco frente a una perdida del 42.9% experimentada por Xen.

3.7.3. Soporte técnico

Gracias a la integración en el kernel, así como por ser un sistema más actual y en crecimiento, KVM ofrece una mayor atención y soporte técnico a todos los usuarios que lo utilizan. Esto hace que en la actualidad la mayoría de la empresas opten por utilizar KVM frente a Xen. Por extensión, la preferencia de las empresas por el uso de KVM otorgan al usuario una mayor compatibilidad de su aplicación con otras posibles herramientas, las cuales también utilizan este soporte. Por otro lado, la adaptación de KVM al sistema de la gran empresa de mantenimiento de linux Red hat, hace que KVM disponga de una gran cantidad de información de soporte, además de encontrarse actualizado y adaptado a las últimas versiones de linux, algo que Xen debe hacer de manera externa no proporcionando siempre unos resultados óptimos.

3.7.4. Tipo de virtualización

Xen utiliza únicamente paravirtualización. En este tipo de virtualización se introduce una capa de abstracción intermedia entre el hardware y el sistema operativo que actúa como un árbitro que controla el acceso al hardware de las máquinas virtuales. Sin embargo, sólo se encarga de algunas de las instrucciones, el resto es ejecutado directamente por el hardware en nombre de los sistemas. La ventaja principal es que las operaciones no se degradan, y los sistemas se ejecutan a una velocidad cercana a la nativa. El inconveniente es que los núcleos de los sistemas operativos que se desean utilizar necesitan ser adaptados.

Por otro lado en el caso de KVM usa una virtualización completa unida a linux mediante una modificación de la aplicación Qemu. De esta forma KVM dispone de un módulo en el núcleo de linux que permite aprovechar las instrucciones del procesador dedicadas a la virtualización(Intel-VT y AMD-V). Esto permite usar un sistema ligero, rápido y con todos los recursos disponibles. La contrapartida es que KVM trabaja con procesadores i386 y AMD64 y solo aquellos lo suficientemente modernos como para aceptar estas instrucciones.

Capítulo 4

Arquitectura del sistema

4.1. Introducción

En este capítulo presentaremos una visión general de la arquitectura de *CygnusCloud*. Para facilitar su comprensión, hemos intentado que la claridad prevalezca a la hora de exponer los distintos conceptos y decisiones de diseño. Por ello, evitaremos a toda costa realizar descripciones exhaustivas de todas y cada una de las clases y de sus distintos atributos. Si el lector desea conocer con total precisión qué atributos y qué métodos tiene cada clase, cómo se comporta cierto método o cuál es la finalidad de un determinado atributo, le remitimos al código fuente, extensamente comentado.

Por otra parte, en este capítulo también prestaremos especial atención a las decisiones de diseño que hemos tomado ya que, a nuestro parecer, son fundamentales para comprender el funcionamiento del sistema, sus ventajas y sus limitaciones.

4.1.1. Visión general

Los contenidos de este capítulo se agrupan en las siguientes secciones:

1. La presente **introducción**.
2. **Objetivos de la arquitectura y restricciones.** En esta sección expondremos todos los objetivos que se tuvieron al diseñar *CygnusCloud*, y también recopilaremos las restricciones con las que hemos tenido que tratar a lo largo de todo el proceso de diseño.
3. **Decisiones de diseño.** En esta sección justificaremos las decisiones más relevantes que hemos tomado en el proceso de diseño.
4. **Vista lógica.** En esta sección, mostraremos cómo las funciones de cada subsistema se distribuyen entre sus distintos módulos con la ayuda de un diagrama de paquetes. Además, utilizaremos diagramas de clase y diagramas de secuencia para exponer las responsabilidades de cada elemento significativo de la arquitectura y sus relaciones con el resto de componentes de la misma.
5. **Vista de procesos.** En ella mostraremos el funcionamiento de cada subsistema desde el punto de vista de los procesos ligeros o *threads*, y también discutiremos el cometido de los mismos.
6. **Vista de despliegue.** En esta sección mostraremos cómo se distribuyen los distintos subsistemas que forman parte de *CygnusCloud* entre las distintas máquinas que forman parte de la infraestructura utilizada.
7. **Vista de implementación.** Concluiremos el presente capítulo con esta sección, en la que describiremos los componentes que se distribuyen con cada subsistema.

4.1.2. Sobre los diagramas UML de este documento

Para que el aumento del tamaño de los diagramas UML en un lector de ficheros PDF no suponga una pérdida de calidad de los mismos, hemos procurado incrustarlos en este documento como ficheros .pdf.

El comando de *IBM Rational Software Architect 7.5* que los genera usa degradados para mejorar la presentación de los diagramas, haciendo que los elementos que aparecen en la parte superior del diagrama sean más oscuros que los que aparecen en la parte inferior del mismo. En cualquier caso, este efecto *no* tiene asociado ningún tipo de significado en los diagramas.

4.2. Objetivos de la arquitectura y restricciones

En esta sección describiremos todos los objetivos que consideramos a la hora de diseñar la arquitectura, así como las restricciones que hemos tenido en cuenta a lo largo de todo el proceso de diseño.

4.2.1. Objetivos

Con *CygnusCloud* queremos hacer posible que cualquier estudiante de la Universidad Complutense, sea de la titulación que sea, pueda sacar el máximo partido a cualquier aula de informática infrautilizada del campus. Para ello, es necesario que

- el sistema pueda atender a más alumnos utilizando más servidores y más ancho de banda. Por tanto, la *escalabilidad* debe considerarse en el diseño.
- los alumnos y profesores puedan utilizar *CygnusCloud* sin necesidad de instalar ningún *software* adicional en las aulas de informática. De no ser así, no es posible aprovechar los equipos de las aulas que no tienen instalado el *software* adicional.

Por otra parte, dada la actual situación de crisis económica, *CygnusCloud* debe poder implantarse con *coste cero*. Para ello,

- debemos utilizar exclusivamente software gratuito, y
- el *software* debe ejecutarse sobre servidores antiguos. Así, resulta imprescindible que el *software* sea tan eficiente como sea posible.

Asimismo, el sistema sólo podrá utilizarse con garantías cuando detecte y trate un número suficiente de errores. Por ello, *el sistema debe ser robusto, de forma que pueda detectar y trate el mayor número de errores posible*.

Finalmente, dado que el desarrollo de *CygnusCloud* sólo se extiende a lo largo de un curso académico, también queremos que cualquiera que lo desee pueda continuarlo. Por ello, hemos procurado que nuestro diseño también se caracterice por su *simplicidad*.

4.2.2. Restricciones

Tal y como mencionamos en el apartado 4.2.1, *CygnusCloud* debe poder implantarse con coste cero, por lo que debemos construirlo utilizando *software* gratuito. Puesto que los hipervisores deben instalarse en un sistema operativo en concreto, que tiene que ser gratuito, esto ha impuesto la primera restricción importante: la *dependencia de la plataforma Linux*.

Por otra parte, existe una ingente cantidad de distribuciones *Linux*, que utilizan distintos sistemas de gestión de paquetes y distintos procedimientos de configuración. Para garantizar que *CygnusCloud* funciona correctamente en el mayor número posible de distribuciones *Linux*, resulta fundamental utilizar mecanismos estándar para interactuar con los hipervisores y, a ser posible, lenguajes interpretados. Esto ha impuesto una doble restricción:

- el uso obligatorio de la librería *libvirt* para interactuar con los hipervisores y configurar las redes virtuales, y

- el uso de un lenguaje de tipo interpretado, como Java o Python, como principal lenguaje de programación.

Además, la práctica totalidad de los *frameworks* que hemos evaluado al diseñar la web se basan en el modelo vista-controlador o en otros patrones de diseño derivados de él, lo que nos ha obligado a utilizar dicho patrón de diseño.

Asimismo, puesto que la red troncal de la Complutense utiliza la versión 4 del protocolo IP (*Internet Protocol*), *CygnusCloud* debe operar en redes IP versión 4. No obstante, dado que esta versión del protocolo IP ha quedado totalmente obsoleta, hemos decidido facilitar, en la medida de lo posible, la transición a la versión 6 del protocolo IP, de forma que sólo sea necesario introducir pequeños cambios en el código del módulo de comunicaciones y ajustar las longitudes de ciertos campos en los esquemas de las bases de datos.

Finalmente, para reducir el tiempo de implementación de *CygnusCloud* resulta imprescindible que *todos los subsistemas compartan tanta funcionalidad como sea posible*. De esta manera, todos ellos utilizarán el mismo código para realizar, entre otras cosas, operaciones básicas sobre una base de datos y comunicaciones a través de una red. No obstante, esto sólo es posible si *todos los subsistemas se implementan utilizando el mismo lenguaje de programación*, lo que ha restringido considerablemente el número de librerías y *frameworks* que hemos podido considerar.

4.3. Decisiones de diseño

4.3.1. Uso de una aplicación web para interactuar con *CygnusCloud*

En la sección 4.2.1 dijimos, entre otras cosas, que *CygnusCloud* debe poder utilizarse sin necesidad de instalar *software* adicional en las aulas de informática. Para ello, existen dos alternativas: crear una aplicación web o crear una aplicación de escritorio convencional que pueda utilizarse sin necesidad de ser instalada.

El uso de una aplicación de escritorio tiene una gran ventaja: no es necesario utilizar tecnologías web, totalmente desconocidas para nosotros al inicio del proyecto. No obstante, también tiene asociados algunos inconvenientes considerables:

- para utilizar *CygnusCloud* en móviles o tabletas es necesario crear una aplicación específica, muy dependiente de la plataforma del dispositivo.
- puesto que los usuarios no tienen por qué descargarse siempre la aplicación antes de utilizar *CygnusCloud*, es posible que convivan varias versiones de esta, lo que puede dar lugar a conflictos y, sobre todo, a problemas de seguridad. Para evitar todos estos problemas, es necesario crear un sistema de actualizaciones periódicas, por lo que el diseño y la implementación de la aplicación se complican.
- para poder enviar las peticiones al sistema, la aplicación necesita averiguar la dirección IP y el puerto de la máquina que las tratará. Para ello, existen dos alternativas:
 - incrustar estos datos en el código. Esto dificulta el cambio de la IP y el puerto de la máquina que tratará las peticiones.
 - anunciarlos periódicamente a través de la red. Con esto, además de desperdiciar ancho de banda, estamos comprometiendo la seguridad del sistema, ya que cualquier usuario malintencionado podrá recibir estos mensajes.
- aunque el tráfico viaje cifrado, cualquier usuario que utilice un analizador de tráfico podrá averiguar la dirección IP y el puerto de la máquina que trata sus peticiones, lo que facilita mucho la realización de ataques de denegación de servicio.

Por otra parte, al utilizar una aplicación web desaparecen muchos de estos inconvenientes, ya que

- para utilizar *CygnusCloud* en móviles o tabletas sólo es necesario introducir cambios en el *layout* o la disposición de elementos de la página.

- no es necesario descargar ningún programa en los PCs que usan los usuarios. Esto hace el sistema más fácil de usar (basta con utilizar un navegador web) y permite ahorrar ancho de banda.
- las actualizaciones son muy fáciles de aplicar: basta con parar el servidor web, copiar en él los ficheros binarios y reiniciarlo.
- aunque sigue siendo posible realizar ataques de denegación de servicio, estos sólo afectarán al servidor web y no a los servidores de la infraestructura, por lo que esta seguirá procesando las peticiones ya recibidas correctamente.

Finalmente, el uso de una aplicación web tiene un serio inconveniente: debido a nuestra inexperiencia en este campo, es posible que no detectemos algunas alternativas para comprometer la seguridad del sistema. De todas formas, podemos considerar que la red troncal de la UCM es segura, por lo que este tipo de prácticas no serán muy frecuentes.

4.3.2. Descomposición del sistema *CygnusCloud* en cuatro subsistemas

Para atender las peticiones de los usuarios, en la infraestructura de *CygnusCloud* colaboran hasta cuatro subsistemas distintos. En esta sección, mostraremos, de forma general, las responsabilidades de cada uno de ellos. Además, también expondremos los motivos que nos llevaron a hacer esta descomposición.

4.3.2.1. Los servidores de máquinas virtuales

Instanciar una máquina virtual requiere muchos recursos. No basta con asignarle memoria RAM y tiempo de CPU: también es necesario reservar espacio en disco en el que almacenar los datos de los usuarios y los archivos de paginación.

Por muy potente que sea el *hardware* del servidor que alojará las máquinas virtuales, el número máximo de máquinas virtuales estará siempre limitado. Por tanto, la única forma de superar esta limitación y aumentar el número de máquinas virtuales que pueden estar activas a la vez es utilizando varios servidores para albergarlas.

Por otra parte, el arranque y la creación o edición de máquinas virtuales son operaciones que requieren mucha entrada/salida. Para garantizar que el tiempo de respuesta del sistema es adecuado, lo más conveniente es que los nodos que albergan máquinas virtuales se dediquen a ello en exclusiva. De ahora en adelante, llamaremos **servidores de máquinas virtuales** a los nodos que albergan exclusivamente máquinas virtuales.

Es importante notar que, una vez creadas las máquinas virtuales, el servidor VNC que utilizarán los usuarios para interactuar con ellas reside en el propio servidor de máquinas virtuales, por lo que el tráfico VNC circulará exclusivamente entre los PCs de los usuarios y los servidores de máquinas virtuales. Además, el tráfico que los usuarios generan para conectarse a internet se vuela directamente a la red, es decir, no atraviesa ninguna otra máquina de la infraestructura de *CygnusCloud*.

4.3.2.2. El repositorio de imágenes

Los usuarios sólo pueden arrancar, crear y editar máquinas virtuales cuando sus imágenes de disco están desplegadas en un servidor de máquinas virtuales. No obstante, hay imágenes de disco que no deben estar desplegadas en ningún servidor de máquinas virtuales. Este es el caso de

- las imágenes *vanilla*, que sólo tienen el sistema operativo configurado con los programas más básicos instalados, y
- las imágenes que no han terminado de editarse.

Estas imágenes no están completamente configuradas, por lo que ningún alumno las podrá arrancar. Así, su presencia en un servidor de máquinas virtuales supone un desperdicio de espacio en disco.

Además, para simplificar la administración del sistema no resulta conveniente que las imágenes de disco se dispersen entre múltiples máquinas. Si existe una máquina que almacena una copia de todas las imágenes de disco existentes, para modificar todas las copias existentes de una de ellas sólo será necesario acudir a esta máquina, modificar los ficheros correspondientes y volver a desplegar la imagen de disco.

Esta máquina, a la que llamaremos **repositorio de imágenes**, almacenará una copia de todas las imágenes de disco existentes. Es importante notar que la única copia de las imágenes *vanilla* y de las imágenes de disco que se están editando se encuentra en el repositorio de imágenes.

4.3.2.3. El servidor de *cluster*

En un momento dado, varios servidores de máquinas virtuales pueden albergar una nueva instancia de cierta máquina virtual. Ahora bien, no todos ellos tienen por qué encontrarse en las mismas condiciones.

Por ejemplo, si los servidores A y B pueden albergar una nueva instancia de la máquina virtual Debian-LSO y el servidor A alberga más instancias que el servidor B, la experiencia del usuario mejorará si alojamos la nueva instancia la nueva instancia en el servidor B.

Por tanto, necesitamos utilizar una máquina que

- averigüe periódicamente el estado de todos los servidores de máquinas virtuales y del repositorio de imágenes,
 - al atender las peticiones del usuario, compruebe que estas se pueden satisfacer. Esto resulta fundamental para ahorrar ancho de banda.
- Por ejemplo, si el repositorio de imágenes no tiene suficiente espacio en disco para alojar una nueva máquina virtual, no tiene sentido transferir sus imágenes de disco a un servidor de máquinas virtuales y configurar en él la máquina virtual, ya que estas no podrán subirse al repositorio de imágenes.
- realice el balanceado de carga entre todos los servidores de máquinas virtuales.

Además, para facilitar la administración del sistema, esta máquina también se ocupará de arrancar, parar, dar de alta y borrar servidores de máquinas virtuales. De ahora en adelante, llamaremos a esta máquina **servidor de *cluster***.

Como el lector habrá notado ya, este diseño tiene un serio inconveniente: el servidor de *cluster* se convierte un cuello de botella que limita el número máximo de servidores de máquinas virtuales.

Para superar esta limitación, podemos crear varias agrupaciones (*clusters*) de servidores de máquinas virtuales, cada una de las cuales tendrá asociado su propio servidor de *cluster* y su propio repositorio de imágenes. De esta manera, para atender a un número mayor de usuarios bastará con utilizar más *clusters*.

Finalmente, es importante notar que cada *cluster* es completamente autónomo, por lo que no tiene por qué compartir máquinas virtuales con otros *clusters*.

4.3.2.4. El servidor web

Como los servidores de *cluster* son ya un cuello de botella y puede haber varios, la página web de *CygnusCloud* no puede residir en ninguno de ellos. Además, por motivos de eficiencia, los servidores de *cluster* no realizan ningún tipo de control de acceso, por lo que no pueden atender directamente las peticiones de los usuarios.

Así pues, es necesario utilizar una máquina adicional, el **servidor web**, para alojar la página web de *CygnusCloud*. A través de ella,

- se realizará el control de acceso,
- se enviarán las peticiones de los usuarios a un servidor de *cluster*, y
- se administrará el sistema.

4.3.3. Implementación de una infraestructura *ad-hoc*

Para gestionar los servidores de la infraestructura y las máquinas virtuales, es posible utilizar dos soluciones gratuitas y muy populares: *OpenStack* y *OpenNebula*. Estas herramientas permiten gestionar *clouds* de gran tamaño, e implementan toda la funcionalidad que necesitamos para realizar la gestión de los servidores de la infraestructura y de las máquinas virtuales. No obstante, hemos preferido diseñar e implementar una infraestructura *ad-hoc* por las siguientes razones:

- *OpenStack* y *OpenNebula* proporcionan muchas funciones a costa de consumir muchos recursos. Tal y como dijimos en la sección 4.2.1, *CygnusCloud* debe poder implantarse con coste cero, y para ello es imprescindible aprovechar servidores antiguos, para los que estas soluciones pueden ser demasiado pesadas.
- *OpenStack* y *OpenNebula* son excesivamente grandes. Para implementar *CygnusCloud*, sólo es necesario utilizar un pequeño subconjunto de la funcionalidad que ofrecen *OpenStack* y *OpenNebula*. Ahora bien, para fijar dicho subconjunto y familiarizarnos con estas soluciones habríamos necesitado, como mínimo, tres meses. Esto supone casi la tercera parte del tiempo de desarrollo del proyecto, por lo que no es aceptable.
- al implantar una infraestructura *ad-hoc*, podemos descubrir por nosotros mismos qué hace falta para suministrar un servicio del tipo infraestructura como servicio. Así, nos podemos familiarizar con muchas de las tecnologías de virtualización existentes en el mercado. Además, este conocimiento nos permitirá, en un futuro próximo, comprender mejor y más rápidamente el funcionamiento de *OpenStack* y *OpenNebula*.

4.3.4. Configuración de las redes virtuales en modo NAT

Para que los usuarios de *CygnusCloud* puedan utilizar las máquinas virtuales para trabajar, es imprescindible que estas accedan, a través de la red troncal de la Complutense, al Campus Virtual y a Internet.

La forma más sencilla de lograr esto es configurando un bridge para cada máquina virtual. Así, estas se conectarán a la red troncal de la Complutense como si fuesen un equipo físico más.

Tal y como dijimos en la sección 4.2.2, la red troncal de la UCM sigue utilizando la versión 4 del protocolo IP (Internet Protocol). Con esta primera alternativa, cada máquina virtual debe tener una dirección IP única. Y dada la actual escasez de direcciones IP versión 4, esto limita el número de máquinas virtuales que pueden estar activas a la vez: puesto que *CygnusCloud* debe poder implantarse con coste cero,

- no es posible adquirir nuevos (y costosos) bloques de direcciones IP para utilizarlo, y
- para que la red troncal de la UCM funcione con normalidad, *CygnusCloud* no puede monopolizar el uso de las direcciones IP disponibles.

Por ello, la solución que acabamos de presentar no es viable.

La alternativa se basa en el uso de NAT (*Network Address Translation*, traducción de direcciones de red) en los servidores de máquinas virtuales. En este caso,

- sólo los servidores de máquinas virtuales (y no las máquinas virtuales) se conectan a la red troncal de la UCM.
- las máquinas virtuales se conectan a una red virtual, existente sólo en el servidor de máquinas virtuales.
- cuando una máquina virtual envíe un datagrama IP, el servidor reescribirá su cabecera. En la red troncal de la UCM, parecerá que es el servidor de máquinas virtuales el que ha creado dicho datagrama.

- cuando el servidor de máquinas virtuales reciba un datagrama IP, reescribirá su cabecera y lo reenviará a la máquina virtual correspondiente. En la máquina virtual, parecerá que se ha accedido directamente a la red troncal de la UCM.

Como cada servidor de máquinas virtuales puede contener varias máquinas virtuales activas a la vez, el ahorro de direcciones IP con esta alternativa es considerable. Por ello, decidimos utilizarla. No obstante, nos hemos asegurado de que resulte sencillo crear un bridge para cada máquina virtual. Así, los administradores del sistema podrán utilizar una u otra alternativa cuando lo estimen oportuno.

4.3.5. Uso del protocolo de escritorio remoto VNC

Para que los usuarios puedan interactuar con la máquina virtual, es imprescindible utilizar un protocolo de escritorio remoto. Estos permiten que los clientes interactúen con las máquinas virtuales a través de las interfaces gráficas de usuario a las que están acostumbrados. En su momento, consideramos dos alternativas: VNC y RDP.

VNC (*Virtual Network Computing*, computación virtual en red) es un protocolo de escritorio remoto independiente de la plataforma y desarrollado por Olivetti, Oracle Corporation y, posteriormente, por AT&T.

Por su parte, **RDP** (*Remote Desktop Protocol*, protocolo de escritorio remoto) es un protocolo de escritorio remoto desarrollado por Microsoft, utilizado mayoritariamente en sistemas Windows. Existen clientes y servidores RDP para otros sistemas operativos, como *Linux* y *Mac OS X*.

En términos generales, el funcionamiento de ambos protocolos es el siguiente:

- la máquina que comparte su pantalla se denomina **servidor**.
- la máquina en la que trabajan los usuarios se denomina **cliente**.
- cuando el cliente mueve el ratón o pulsa una tecla,
 1. el cliente de escritorio remoto envía los datos correspondientes al servidor de escritorio remoto.
 2. al recibirlas, el servidor de escritorio remoto generará el evento de teclado o ratón correspondiente, que será atendido por el sistema operativo de la máquina remota de la forma habitual.
 3. el servidor de escritorio remoto averigua qué zonas de la pantalla se han actualizado, comprime esta información y se la envía al cliente de escritorio remoto.
 4. a partir de esta información, el cliente de escritorio remoto actualiza la imagen de la pantalla que ve el usuario.

Como *CygnusCloud* debe poder implantarse con coste cero, los clientes y servidores de escritorio remoto que utilicemos deben ser gratuitos.

En términos generales, las implementaciones libres y gratuitas de RDP y VNC tienen un rendimiento bastante similar. Además, VNC está integrado en la práctica totalidad de sistemas de virtualización, por lo que es posible utilizarlo sin necesidad de instalar nada en las máquinas virtuales. Por ello, decidimos utilizar el protocolo VNC.

4.3.6. Uso del gestor de bases de datos MariaDB

Al inicio del proyecto, estábamos mucho más familiarizados con los gestores de bases de datos relacionales que con los no relacionales. Por ello, decidimos utilizar un gestor de bases de datos relacional para manipular todas las bases de datos de la infraestructura.

En la actualidad, los gestores gratuitos de bases de datos relacionales más completos son dos: MySQL y MariaDB.

MySQL es el gestor de bases de datos relacionales de código abierto más extendido, y se utiliza en servicios web con un gran número de usuarios (como *Facebook*, *Wordpress* y *YouTube*) y también en muchos otros sistemas de menor tamaño.

Por otra parte, **MariaDB** es un gestor de bases de datos relacionales cuya popularidad entre las comunidades de *software libre* no ha parado de crecer. En los últimos meses, la mayoría de distribuciones *Linux* han decidido reemplazar MySQL por MariaDB, y organizaciones de gran tamaño como la *Wikimedia Foundation* han decidido migrar sus bases de datos a MariaDB.

Aunque inicialmente MariaDB estaba basada en la versión 5.5 de MySQL, sus nuevas versiones incluyen un gran número de mejoras enfocadas, sobre todo, a conseguir un rendimiento superior al de MySQL. Además, como MariaDB es totalmente compatible a nivel sintáctico y a nivel binario la versión 5.5 de MySQL, resulta muy sencillo reemplazar MySQL por MariaDB.

Tras examinar lo que ha sucedido tras la compra de *Sun Microsystems* por parte de *Oracle* en 2009, nos parece que el desarrollo de MySQL ha sufrido un abandono progresivo, de forma que en la actualidad todo parece indicar que, a medio plazo, *Oracle* va a dejar de dar soporte al proyecto.

Considerando nuestras necesidades, MariaDB y MySQL son totalmente equivalentes. Además, el soporte a medio y a largo plazo de MariaDB parece estar garantizado, cosa que no ocurre en el caso de MySQL. Por ello, decidimos utilizar MariaDB.

4.3.7. Uso del hipervisor KVM

Las dos soluciones de virtualización libres y gratuitas que cuentan con el mejor rendimiento son dos: Xen y KVM. En la sección 3.7 hicimos una comparativa de ambas y explicamos que, por contar con un rendimiento muy similar, mejor documentación y mejor soporte que Xen, optamos por utilizar KVM.

4.3.8. Uso del servidor VNC integrado en KVM

Tal y como dijimos en la sección 4.3.5, para poder utilizar el protocolo de escritorio remoto VNC es necesaria la existencia de un servidor VNC, que envíe periódicamente al cliente VNC las actualizaciones de la pantalla de la máquina virtual.

El uso del servidor VNC integrado en KVM tiene dos grandes ventajas:

- no es necesario instalar ni ejecutar código adicional en la máquina virtual, lo que simplifica la creación y configuración de máquinas virtuales.
- el usuario de la máquina virtual no puede detener por error el servidor VNC. Si el servidor VNC se ejecuta en la máquina virtual y el cliente mata su proceso por error, la máquina virtual quedará inutilizada.
- la gestión de los datos de conexión a las máquinas virtuales es muy sencilla.

Por otra parte, el uso de un servidor VNC que resida en cada máquina virtual tiene una gran ventaja: aporta mucha más flexibilidad. No obstante, en este caso la gestión de los datos de conexión es más compleja, ya que estos se fijan en la máquina virtual y son, *a priori*, desconocidos para los servidores de la infraestructura.

Tras hacer pruebas para evaluar las dos alternativas, concluimos lo siguiente:

- la gran mayoría de servidores VNC libres y gratuitos proporcionan las mismas funciones que el servidor VNC integrado en KVM.
- el rendimiento de ambas es prácticamente idéntico.

Además, como el uso del servidor VNC integrado en KVM simplificaba considerablemente la implementación del sistema, finalmente decidimos utilizarlo.

4.3.9. Uso del cliente VNC noVNC

Tal y como dijimos en la sección [4.3.1](#), CygnusCloud se utilizará a través de una aplicación web. Para que los usuarios puedan utilizar las máquinas virtuales, es necesario disponer de un cliente VNC, que se conecte al servidor VNC correspondiente. Además, este debe integrarse con la aplicación web de forma que, al arrancar una máquina virtual, los usuarios puedan utilizarlo para interactuar con ella.

Inicialmente, consideramos modificar un cliente de escritorio VNC libre y gratuito. Este debía ser multiplataforma y funcionar, como mínimo, en los sistemas operativos *Windows* y *Linux*. En su momento, consideramos tres distintos: TigerVNC, TightVNC y la versión *open source* de RealVNC.

La principal ventaja de esta alternativa es el rendimiento: dado que estos clientes están escritos en C o C++, es posible utilizarlos en equipos poco potentes de forma muy satisfactoria. No obstante, el hecho de que estos clientes estén escritos en C o C++ supone también su mayor inconveniente: es necesario crear una versión distinta para cada sistema operativo, algo que no es trivial, y sólo es posible utilizarlos en PCs.

Por ello, pensamos en una segunda alternativa: el uso de un cliente VNC web. Aunque el rendimiento es menor, podemos utilizar una única versión, que funcionará en cualquier sistema operativo y en cualquier dispositivo, incluyendo PCs, móviles, tabletas, . . . Además, el cliente VNC se integrará mucho mejor con la web de *CygnusCloud*.

Y puesto que los equipos antiguos que actualmente se utilizan en la UCM tienen suficiente capacidad de procesamiento para utilizar un cliente VNC web, nos decantamos por esta segunda alternativa. En su momento, valoramos dos clientes VNC web: el applet Java del proyecto TigerVNC y noVNC.

Aunque en términos generales el rendimiento del applet Java es satisfactorio, su desarrollo ha estado detenido desde el año 2007, y en la actualidad carece de todo tipo de soporte.

Por otra parte, noVNC es un cliente VNC relativamente nuevo, escrito íntegramente en HTML5 y *JavaScript*, que se ha integrado en un gran número de soluciones como *OpenStack*, *OpenNebula*, *CloudSigma*, *Intel MeshCentral*, . . .

Sus principales características son las siguientes:

- para utilizarlo, sólo es necesario tener instalado un navegador web razonablemente reciente.
- se adapta a los tamaños de pantalla de móviles y tabletas, lo que permite utilizarlo en PCs y en estos dispositivos, cada vez más extendidos.
- se distribuye bajo licencia *GPL*, lo que permite modificarlo cuanto sea necesario a la hora de realizar su integración con la web de *CygnusCloud*.
- soporta tráfico cifrado
- ajusta el número de colores y la resolución de la pantalla en función de la potencia del dispositivo que utilizan los usuarios.

Puesto que noVNC tiene características muy interesantes, requiere instalar menos software en los PCs en los que trabajan los usuarios y, sobre todo, cuenta con un buen soporte, decidimos utilizarlo en *CygnusCloud*.

4.3.10. Uso de la librería de virtualización *libvirt*

Tal y como mencionamos en la sección [4.2.2](#), cada distribución *Linux* tiene sus propios procedimientos de configuración. Así, aunque utilicemos el mismo hipervisor, la forma de manipular las máquinas y redes virtuales puede variar de unas distribuciones a otras.

Además, interactuar directamente con el hipervisor no es nada conveniente ya que, para utilizar otro diferente, será necesario introducir muchas modificaciones.

Para resolver estos dos problemas, no manipulamos las máquinas y las redes virtuales directamente: lo hacemos a través de la librería *libvirt*.

libvirt es la librería de virtualización por excelencia. Sus principales características son las siguientes:

- soporta muchos sistemas de virtualización, entre los que están Xen, KVM, VirtualBox y VM-Ware.
- aunque está escrita en C, dispone de *bindings* para muchos otros lenguajes, como Java o Python.
- se distribuye bajo licencia LGPL, por lo que es posible utilizarla incluso desde *software* privativo.
- a través de ella, es posible crear y destruir redes virtuales, y también crear, destruir y migrar (mover de servidor) máquinas virtuales en ejecución.

Tras examinar las características de *libvirt* y evaluar nuestras necesidades, concluimos que esta librería nos proporcionaba todas las funciones que necesitábamos para utilizar máquinas y redes virtuales en *CygnusCloud*.

Por otra parte, al inicio del proyecto no encontramos ninguna librería similar, por lo que la única alternativa al uso de *libvirt* era interactuar directamente con el hipervisor. Así pues, decidimos utilizar *libvirt*.

4.3.11. Uso de *Python 2.7* como principal lenguaje de programación

Tal y como mencionamos en la sección 4.2.2, debido a las distintas características y procedimientos de configuración de las distribuciones *Linux*, es altamente recomendable utilizar lenguajes interpretados para facilitar la portabilidad del sistema. Los lenguajes de este tipo que consideramos al inicio del proyecto fueron esencialmente dos: *Java* y *Python*.

El uso de *Python* tenía las siguientes ventajas:

- el grado de adopción de *Python* no ha hecho más que crecer a lo largo de los últimos años, por lo que aprender a utilizarlo era muy importante para mejorar nuestras perspectivas laborales.
- a diferencia de *Java*, *Python* es *software* libre, por lo que sus restricciones de uso son considerablemente menores que las de *Java*.
- la interacción entre *scripts* y programas escritos en *Python* es claramente mejor en *Python* que en *Java*.
- los programas escritos en *Python* son mucho más breves y más fáciles de leer que los programas escritos en *Java*, lo que facilita la comprensión del código.
- para ejecutar un programa *Python* se utiliza su propio código fuente, lo que facilita enormemente la resolución de los problemas que aparecen al utilizar componentes de terceros.
- *Python* forma parte de la instalación estándar de la inmensa mayoría de distribuciones *GNU/Linux*, lo que simplifica el proceso de instalación de *CygnusCloud*.

Los grandes inconvenientes del uso de *Python* eran principalmente dos:

- era necesario aprender a utilizar un lenguaje totalmente nuevo para nosotros, cosa que no ocurría en el caso de *Java*.
- en general, hay muchas más bibliotecas y *frameworks* escritos en *Java* que escritos en *Python*, lo que limitaba las alternativas que podíamos considerar a lo largo del desarrollo del proyecto.

Ante el gran número de ventajas que tenía el uso de *Python*, estos dos inconvenientes nos parecieron aceptables, por lo que decidimos utilizar *Python* como el principal lenguaje de programación de *CygnusCloud*.

Por otra parte, en la actualidad existen dos familias de versiones de *Python*: las derivadas de la versión 2.7 y las derivadas de la versión 3.0. Ambas familias son incompatibles a nivel sintáctico, aunque es cierto que los cambios que es necesario introducir para utilizar código de la familia 2.7 en la familia 3.0 suelen ser triviales.

Ahora bien, la gran mayoría de librerías escritas en *Python*, entre las que se encuentran los *bindings* de libvirt y los conectores de MySQL, no han hecho aún la transición a la familia 3.0, por lo que no pueden utilizarse con estas versiones. Por ello, no hemos tenido más remedio que utilizar la sintaxis de la familia 2.7.

No obstante, hemos procurado no hacer uso de ninguna de las características de la familia 2.7 que ha desaparecido en la familia 3.0. Así, la conversión del código de *CygnusCloud* a la familia 3.0 resultará bastante sencilla.

4.3.12. Uso de la librería de red *twisted*

Puesto que *CygnusCloud* se compone de sistemas diferentes que residen en máquinas distintas, es imprescindible comunicarlos. Puesto que las máquinas intercambian entre sí un gran volumen de datos, es muy conveniente utilizar un servicio orientado a conexión y fiable: TCP.

Para realizar las comunicaciones *socket* a *socket*, optamos por utilizar la librería de red *twisted*, que se basa en eventos y está íntegramente escrita en *Python*. Las razones que nos llevaron a elegirla fueron las siguientes:

- *twisted* es el estándar *de facto* para comunicar programas escritos en *Python* sin necesidad de manipular *sockets* directamente, y forma parte de la instalación predeterminada de la mayoría de distribuciones *GNU/Linux* (e incluso de *Mac OS X*).
- existen libros, una ingente cantidad de documentación, foros y tutoriales con abundantes explicaciones de uso.
- se trata de una librería de código abierto, distribuida bajo licencia MIT y actualizada periódicamente.
- soporta los protocolos de transporte TCP, UDP y TCP sobre SSL, por lo que proporciona toda la funcionalidad que necesitábamos a lo largo del desarrollo del proyecto.
- soporta IP versión 6. Esto permite utilizar *CygnusCloud* en redes IP versión 4 y, a medida que las organizaciones que lo utilicen hagan la transición, también en redes IP versión 6.

El gran inconveniente de esta librería es su propia documentación. Debido a su enorme extensión, hay partes desactualizadas, partes que contradicen a otras, partes sin apenas documentar... por lo que en la práctica su uso requiere seguir el método de prueba y error.

No obstante, tal y como expusimos en la sección 4.2.2, para facilitar la portabilidad entre distribuciones *GNU/Linux* es imprescindible utilizar mecanismos estandarizados, por lo que nos acabamos decantando por utilizar esta librería.

4.3.13. Uso del protocolo de transferencia de ficheros FTP

En *CygnusCloud* es necesario transferir imágenes de disco entre el repositorio de imágenes y los servidores de máquinas virtuales. Para hacerlo, lo más adecuado es utilizar uno de los múltiples protocolos de transferencia o de compartición de ficheros existentes. En su momento, consideramos los siguientes:

- **NFS (Network File System).** Se trata de un protocolo muy popular en sistemas tipo UNIX. La principal ventaja de este protocolo es que, una vez establecida la conexión con la máquina remota, es posible acceder a los ficheros compartidos como si estuviesen almacenados en el sistema de ficheros de la máquina local.

- **Samba.** Se trata de una reimplementación del protocolo conocido como SMB (*Server Message Block*) o CIFS (*Common Internet File System*), utilizado para compartir ficheros y dispositivos entre máquinas *Windows*. Al igual que en el caso de NFS, tras establecer la conexión los ficheros compartidos son accesibles desde el sistema de ficheros de la máquina local.
- **FTP (File Transfer Protocol).** Es uno de los primeros protocolos de transferencia de ficheros que se diseñaron, y por ello también uno de los más inseguros, eficientes y fáciles de utilizar. La funcionalidad de FTP es muy básica comparada con la de NFS o Samba: sólo permite transferir ficheros entre dos máquinas.
- **FTPS (File Transfer Protocol Secure).** Se trata de una extensión del protocolo FTP que cifra todo el tráfico.
- **SFTP (Secure File Transfer Protocol).** Aunque ambos tengan nombres parecidos, este protocolo no está relacionado con el protocolo FTP. Se trata de una extensión del protocolo SSH (*Secure Shell*) que permite transferir ficheros entre máquinas.
- **HTTP (HyperText Transfer Protocol) o HTTPS (HyperText Transfer Protocol Secure).** Estos protocolos, que son la base de las comunicaciones en internet, también pueden utilizarse para transferir ficheros entre máquinas. La diferencia entre ambos está en el cifrado de los datos: en el caso de HTTP, los datos viajan sin cifrar, a diferencia de lo que ocurre con HTTPS.

Puesto que *CygnusCloud* se utilizará a través de una aplicación web, decidimos utilizar los protocolos que más fácilmente pudiesen integrarse con ella. Por ello, descartamos los protocolos NFS, Samba y SFTP. De esta manera, tuvimos que elegir entre los protocolos FTP, FTPS, HTTP y HTTPS.

Tal y como dijimos en la sección 4.2.1, *CygnusCloud* debe ser eficiente en el uso de los recursos. Desde el punto de vista de la eficiencia, los protocolos FTP y FTPS son mejores que HTTP y HTTPS, ya que envían mucha menos información a través de la red, y requieren mucho menos procesamiento en las máquinas que intercambian ficheros. Por ello, descartamos los protocolos HTTP y HTTPS.

Así pues, al final tuvimos que elegir entre los protocolos FTP y FTPS. Finalmente, decidimos utilizar el protocolo FTP por ser el más eficiente y rápido de los dos.

Aunque es cierto que el protocolo FTP tiene serios problemas de seguridad, no debemos olvidar que *CygnusCloud* se utilizará a través de la red troncal de la Complutense que es, en principio, segura. No obstante, para permitir el uso de *CygnusCloud* en redes menos seguras, hemos decidido adaptar el diseño y la implementación para que sea posible utilizar el protocolo FTPS con poco esfuerzo.

4.3.14. Uso del servidor FTP `pyftpdlib`

Para transferir y recibir imágenes de disco en el repositorio de imágenes, es necesario que este albergue un servidor FTP. En su momento, consideramos dos: `pyftpdlib` y un servidor FTP construido sobre la librería *twisted*.

`pyftpdlib` es un servidor FTP ligero íntegramente escrito en *Python*. Sus principales características son las siguientes:

- es gratuito, y se distribuye bajo licencia MIT.
- es muy ligero, eficiente, estable y escalable.
- funciona con cualquier versión de *Python* entre la 2.4 y la 3.3.
- implementa, entre otras cosas, FTPS (FTP Secure, FTP Seguro), el soporte de IP versión 6, el uso nombres de fichero Unicode, múltiples protocolos de autenticación y el control del ancho de banda utilizado por el servidor FTP.
- dispone de callbacks para detectar múltiples eventos, tales como las conexiones y desconexiones de clientes, el inicio y la finalización de transferencia de archivos,...

4.3.15. Almacenamiento de las imágenes de disco en formato comprimido

- está muy bien documentado.

La alternativa a `pyftpdlib`, construida sobre la librería de red *twisted*, tiene las siguientes características:

- es también gratuita, y también se distribuye bajo licencia MIT.
- el servidor FTP consume muy pocos recursos.
- funciona con cualquier versión de *Python* entre la 2.4 y la 2.7.
- implementa el soporte de IP versión 6 y los protocolos de autenticación más básicos.
- también dispone de *callbacks* para detectar conexiones, desconexiones, errores de autenticación,...

Para decidir entre ambos, los probamos durante algunos días, tras las cuales concluimos lo siguiente:

- como viene siendo habitual en la librería de red *twisted*, la documentación del servidor FTP construido sobre *twisted* es incompleta y en muchos casos incoherente.
- el rendimiento del servidor FTP `pyftpdlib` es superior al del servidor FTP basado en la librería *twisted*.
- el servidor FTP `pyftpdlib` era mucho más fácil de integrar con el repositorio de imágenes que el servidor FTP que utiliza la librería *twisted*.

Además, dado que el servidor `pyftpdlib`

- ofrece muchas más funciones que la otra alternativa, y algunas, como el control del ancho de banda utilizado por el servidor FTP son muy interesantes, y que
- soporta la versión 3 de *Python*, lo que facilita la adaptación del código de *CygnusCloud* para que funcione con las nuevas versiones de *Python*.

decidimos utilizar `pyftpdlib` como servidor FTP del repositorio de imágenes.

4.3.15. Almacenamiento de las imágenes de disco en formato comprimido

Para aprovechar al máximo el espacio en disco disponible en el repositorio y reducir el ancho de banda utilizado para transferir imágenes de disco entre él y los servidores de máquinas virtuales, estas deben almacenarse y transferirse en formato comprimido.

Puesto que *CygnusCloud* debe poder implantarse con coste cero, es necesario utilizar formatos de compresión libres y gratuitos. En su momento, consideramos los formatos 7-zip, zip, tar, lzma, .tar.gz, .tar.bz2, .tar.lzma y .tar.xz.

Para decidir entre ellos, hicimos un pequeño *benchmark*, utilizando todos estos formatos para comprimir una imagen de disco de 4,1 GB. En todas las pruebas,

- utilizamos la misma máquina, con una CPU Intel Core i7 2330QM, 8 GB de memoria RAM DDR3 a 1333 MHz y un disco duro Serial-ATA 2 a 7200 RPM.
- utilizamos la misma versión del kernel Linux, la 3.2.0.
- las mediciones temporales se realizaron con la orden `time`, consideran únicamente el tiempo de CPU, omitiendo los bloqueos provocados por las operaciones de entrada/salida y por los cambios de contexto forzados que provoca el planificador del sistema operativo.

- el uso promedio medio de la CPU, de la RAM y del disco se midieron con las herramientas `iostat` y `pidstat`, pertenecientes al paquete `sysstat`¹. Estas herramientas obtienen la información que muestran del pseudo-sistema de ficheros `/proc`.
- se especificó un nivel de compresión medio.

Los resultados del *benchmark* aparecen en las figuras 4.1, 4.2, 4.3, 4.4, 4.5 y 4.6.

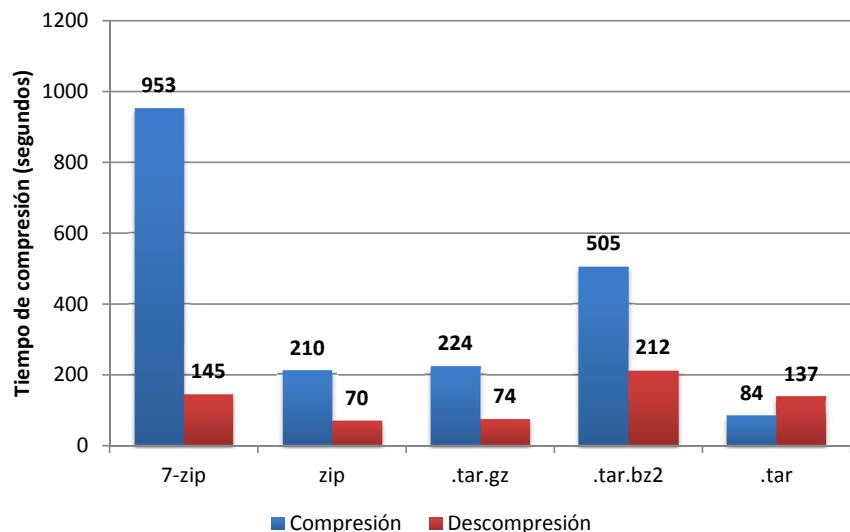


FIGURA 4.1: Tiempos de compresión y descompresión

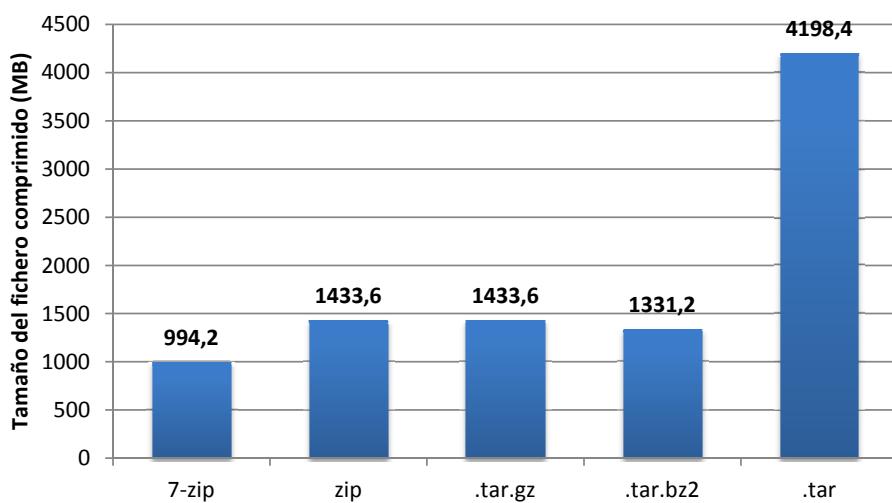


FIGURA 4.2: Tamaño del fichero comprimido

¹El paquete `sysstat` puede descargarse desde <http://sebastien.godard.pagesperso-orange.fr/>

4.3.15. Almacenamiento de las imágenes de disco en formato comprimido

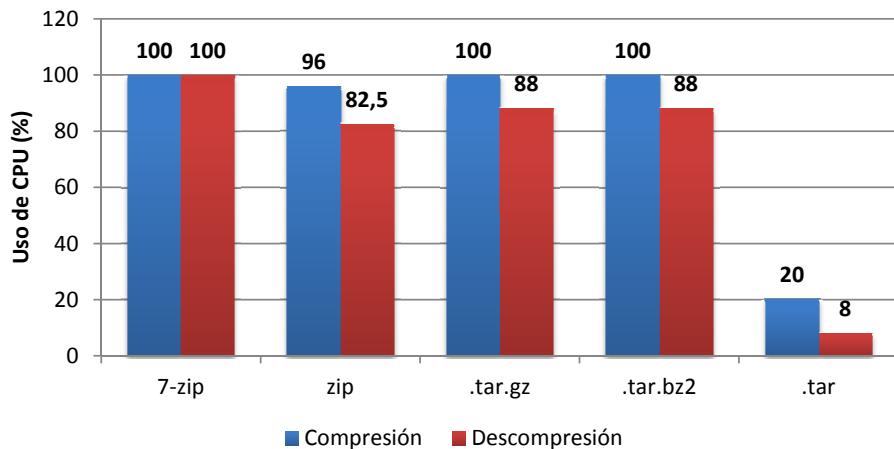


FIGURA 4.3: Uso promedio de CPU

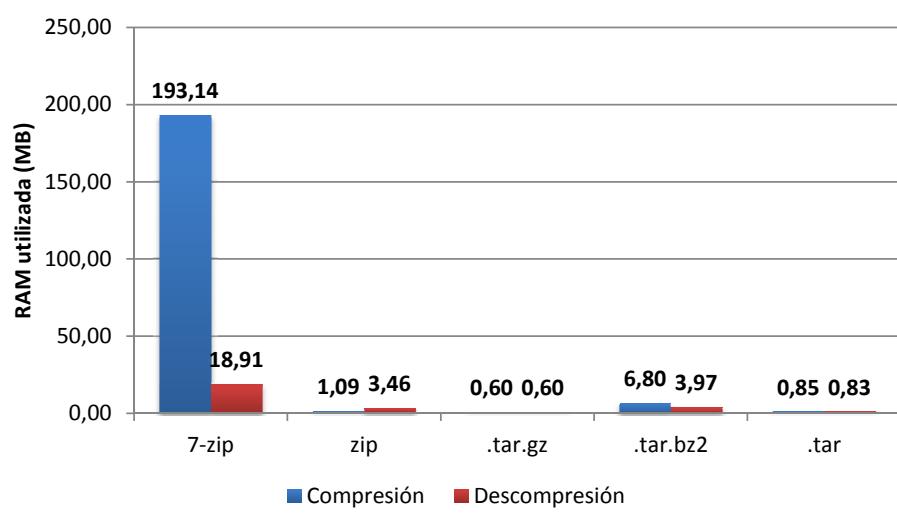


FIGURA 4.4: Uso promedio de memoria RAM

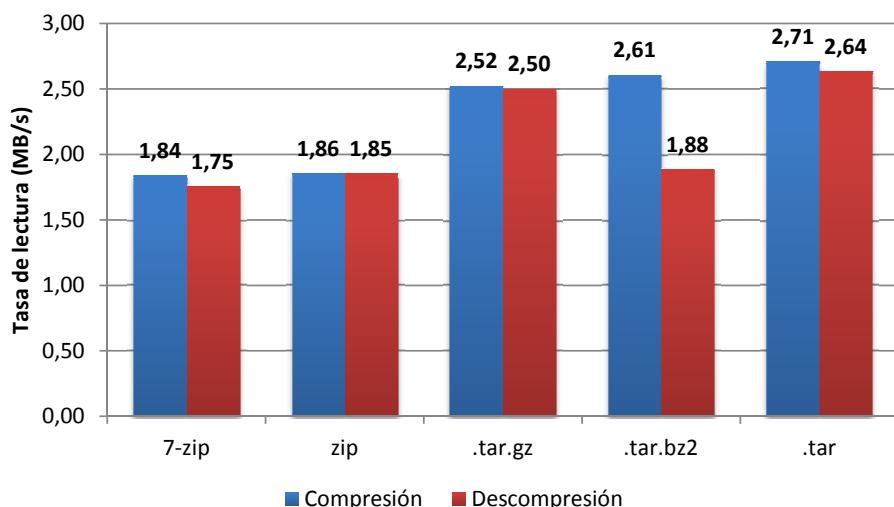


FIGURA 4.5: Tasa de lectura de disco (promedio)

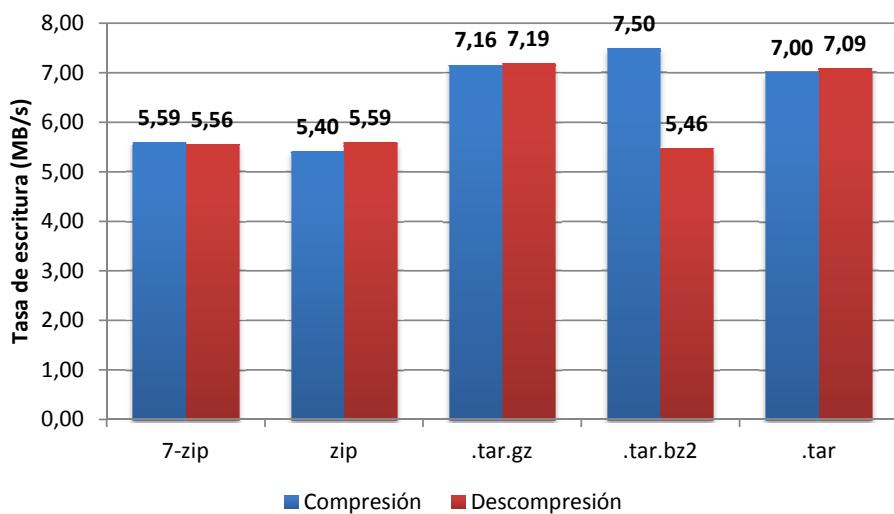


FIGURA 4.6: Tasa de escritura a disco (promedio)

Como el lector habrá observado ya, en las gráficas no aparecen los formatos lzma, tar.lzma y tar.xz. Esto se debe a que

- la compresión y la descompresión de ficheros son mucho más lentas que las del formato 7-zip,
- la tasa de compresión es muy similar a la del formato zip, y
- el uso de CPU y de memoria RAM es superior al del formato zip.

Por ello, los descartamos. Si observamos las gráficas de las figuras 4.1, 4.2, 4.3, 4.4, 4.5 y 4.6, podemos concluir que

- el formato 7-zip consigue la tasa de compresión más alta, aunque lo hace a costa de aumentar muchísimo el tiempo de compresión y extracción, el uso promedio de CPU y el uso promedio de memoria RAM.
- los formatos zip, .tar.gz y .tar.bz2 alcanzan tasas de compresión muy similares. Esto se debe a que los tres usan variantes del algoritmo de Lempel-Ziv.
No obstante, los compresores y descompresores de los dos últimos son algo más lentos que los del formato zip, utilizan más tiempo de CPU y realizan más accesos a disco.
- el formato tar alcanza la tasa de compresión más baja. Por ello, es el que menos recursos usa. Las elevadas tasas de lectura y escritura en disco se deben a que, al comprimir un fichero utilizando el formato tar, se escribe prácticamente una copia del fichero original a disco.

A la vista de estos resultados, descartamos el formato 7-zip por ser demasiado lento y consumir demasiados recursos, y el formato tar por alcanzar tasas de compresión demasiado bajas.

Así pues, tuvimos que elegir entre los formatos zip, tar.gz y .tar.bz2. De todos ellos, escogimos el formato zip por las siguientes razones:

- aunque al comprimir y descomprimir se utiliza más memoria RAM que en el caso del formato .tar.gz, los usos de la CPU y del disco son inferiores, por lo que los procesos de compresión afectarán menos al rendimiento global del servidor.
- es posible añadir ficheros a un fichero zip ya creado sin necesidad de descomprimirlo. En el caso de los formatos .tar.gz y .tar.bz2, esto no es posible.

4.3.16. Uso de imágenes copy-on-write

El sistema operativo y los programas instalados en cada máquina virtual ocupan una cantidad considerable de espacio en disco. Además, el sistema operativo y las utilidades básicas no suelen ser suficientes para trabajar, por lo que frecuentemente es necesario instalar *software* adicional. Esto requiere aún más espacio en disco.

Supongamos que el sistema operativo y los programas que se van a utilizar en la máquina virtual ocupan 12 GB, que el fichero de paginación ocupa 1 GB y los usuarios disponen de 2 GB para almacenar sus datos temporales. En este caso, cada máquina virtual activa requerirá

$$12 + 1 + 2 = 15 \text{ GB de espacio en disco}$$

Como varios usuarios pueden compartir el mismo servidor de máquinas virtuales, es posible que existan varias máquinas virtuales activas de este tipo. Por ejemplo, si existen tres, serán necesarios

$$15 \cdot 3 = 45 \text{ GB de espacio libre en disco}$$

Este comportamiento presenta dos graves inconvenientes:

- las máquinas virtuales activas consumen mucho espacio en disco, lo que limitará el número de máquinas virtuales activas, y también el número de máquinas virtuales diferentes que puede albergar un mismo servidor de máquinas virtuales. Esto último no es nada conveniente para sacar el máximo partido a los servidores de máquinas virtuales.
- el arranque de una máquina virtual será muy lento. Retomando el ejemplo anterior, la máquina sólo se podrá arrancar cuando se copien los 12 GB que ocupan el sistema operativo y los programas. Esto, en un disco duro relativamente moderno, puede llevar más de cinco minutos.

Para superar estas limitaciones, sólo existen dos alternativas:

- limitar el tamaño máximo de las imágenes, lo que limita también la versatilidad de *Cygnus-Cloud*, o

- hacer que todas las máquinas virtuales activas del mismo tipo comparten la instalación del sistema operativo y de los programas. Esto es posible utilizando imágenes *copy-on-write*.

Tras considerar la gran cantidad de espacio en disco que ocupan las instalaciones de *Windows*, la primera alternativa no nos pareció aceptable. Por ello, decidimos utilizar imágenes *copy-on-write* para almacenar el sistema operativo y los programas instalados en las máquinas virtuales.

Así, si en el ejemplo anterior utilizásemos imágenes *copy-on-write*, para alojar las tres máquinas virtuales activas necesitaríamos, aproximadamente,

$$12 + 3 \cdot (1 + 2) = 21 \text{ GB de espacio libre en disco}$$

cantidad muy inferior a la del caso anterior.

4.3.17. Uso de dos discos duros en cada máquina virtual

El uso de imágenes *copy-on-write* nos permite ahorrar una gran cantidad de espacio en disco, pero también tiene un serio inconveniente: las escrituras a disco son mucho más lentas que en el caso de las imágenes normales. Esta diferencia se debe al propio funcionamiento de las escrituras.

Cuando se escribe en una imagen de disco normal, se consulta una cabecera y se actualizan los bytes correspondientes del fichero. Así, las escrituras en este tipo de imágenes son prácticamente igual de rápidas que las escrituras directas a disco.

En cambio, cuando se escribe en una imagen de disco *copy-on-write*,

1. se calculan los cambios que introduce la escritura con respecto a la imagen original.
2. los cambios se escriben a disco, pero no en el fichero de imagen original.

Debido a esto, las escrituras en imágenes *copy-on-write* son más lentas. Asimismo, también serán más lentas las posteriores lecturas de los datos escritos: para obtenerlos, hay que aplicar los cambios sobre el contenido del fichero de imagen original.

Por tanto, si nos limitamos a utilizar imágenes *copy-on-write*, el rendimiento de la máquina virtual se resentirá mucho, ya que las lecturas y escrituras en el fichero de paginación son muy frecuentes, sobre todo en el caso de sistemas operativos *Windows*.

Para superar esta limitación, decidimos utilizar dos imágenes de disco en cada máquina virtual:

- una imagen del tipo *copy-on-write*, para almacenar el sistema operativo y los programas, y
- una imagen normal para almacenar el fichero de paginación y los datos temporales de los usuarios.

Tras realizar esta separación, siguen realizándose escrituras en la imagen *copy-on-write*. No obstante, estas son muy poco frecuentes, por lo que esta solución nos permite ahorrar espacio en disco y garantizar que el rendimiento de la máquina virtual no se resentirá de forma apreciable.

4.3.18. Uso del formato de imagen qcows2

En la sección anterior concluimos que es necesario el uso de dos imágenes de disco en cada máquina virtual: una *copy-on-write*, que almacenará los datos del sistema operativo y de las aplicaciones instaladas, y una imagen normal, que almacenará el fichero de paginación y los datos temporales de los usuarios.

Para crear las imágenes *copy-on-write*, tuvimos que decidir entre dos formatos: *qcows* y *qcows2* (*Qemu Copy-On-Write* y *Qemu Copy-On-Write 2*). Sus principales características son las siguientes:

- tal y como dijimos en la sección 3.6, el hipervisor KVM utiliza los modelos de dispositivo del emulador *Qemu*. Como *qcows* y *qcows2* son los formatos de imagen nativos de *Qemu*, son también los formatos de imagen nativos de KVM.

- pueden utilizarse como imágenes *copy-on-write* o como imágenes de disco normales.
- soportan compresión *zlib*.
- las imágenes *qcow* y *qcow2* crecen dinámicamente a medida que se escriben datos en el disco duro de la máquina virtual. Así, el espacio libre en las imágenes no ocupará espacio en disco en los servidores de máquinas virtuales.

La principal diferencia existente entre los formatos *qcow* y *qcow2* está en las escrituras en modo *copy-on-write*. No obstante, dado que este es un aspecto de muy bajo nivel, no lo discutiremos aquí. Teniendo en cuenta que

- KVM soporta los formatos *qcow* y *qcow2*
- resulta muy sencillo hacer conversiones entre ambos formatos
- el formato *qcow2* es el sucesor natural del formato *qcow*
- con independencia de cuál de los dos formatos utilicemos, el rendimiento es prácticamente idéntico

optamos por utilizar el formato *qcow2* en las imágenes *copy-on-write*.

Por otra parte, en el caso de las imágenes de disco que almacenan el fichero de paginación y los datos temporales de los usuarios consideramos los siguientes formatos:

- *raw*. En realidad, no se trata de un formato de imagen, sino de un fichero que contiene directamente los bytes almacenados en el disco duro de la máquina virtual.
Puesto que para acceder a disco basta con leer o escribir directamente en este fichero, esta es la alternativa que proporciona mejor rendimiento. No obstante, tiene un serio inconveniente: el fichero de la imagen siempre debe tener el mismo tamaño que el disco duro que representa, por lo que el espacio libre en ese disco duro ocupa espacio en disco del servidor de máquinas virtuales.
- *qcow2*. Para que el rendimiento sea adecuado, en estos casos hay que deshabilitar el modo *copy-on-write*, utilizando las imágenes *qcow2* como imágenes normales.
- *vmdk*. Se trata del formato nativo del sistema de virtualización VMWare, también soportado por *Qemu* (y, por tanto, por KVM).

Nuevamente, resulta muy sencillo realizar conversiones entre estos formatos. De todas formas, el hipervisor Xen no soporta el formato *vmdk*, por lo que, de cara a facilitar un hipotético cambio de hipervisor, resulta más conveniente utilizar los formatos *raw* o *qcow2*.

Y puesto que al utilizar el formato *raw* se está desperdiando espacio en disco en muchos casos, decidimos utilizar el formato *qcow2* en las imágenes que almacenan el fichero de paginación y los datos temporales del usuario.

Finalmente, para que el rendimiento de las máquinas virtuales no se resienta en exceso, decidimos deshabilitar la compresión *zlib*. De esta manera, el rendimiento de la máquina virtual no se degradará de forma apreciable con respecto al uso de imágenes *raw*.

4.3.19. Creación de seis familias de máquinas virtuales

Las máquinas virtuales de CygnusCloud pueden utilizarse con fines muy diversos. Por ejemplo, mientras que unas pueden tener instalada una suite ofimática y las herramientas básicas para navegar por internet, otras pueden tener instalado un entorno de desarrollo de desarrollo integrado.

En cualquier caso, todos estos usos no explotan igual la potencia del hardware de la máquina virtual. Retomando el ejemplo anterior,

- en la primera máquina, que tiene instalada la suite ofimática y las herramientas básicas para navegar por internet, el uso de CPU y de memoria RAM será bastante reducido. Además, se utilizará poco espacio en disco.
- en la segunda máquina, que tiene instalado el entorno de desarrollo integrado, se utilizará más memoria RAM y espacio en disco. Además, durante los procesos de compilación, el uso de CPU será muy superior al de la máquina anterior.

Además, los recursos utilizados por el sistema operativo varían de unas familias de sistemas operativos a otras. Por ejemplo,

- los sistemas operativos *Windows* de última generación requieren un mínimo de 9 GB de espacio en disco, al menos 1 GB de memoria RAM y un fichero de paginación que sea, como mínimo, 1,5 veces el tamaño de la memoria RAM, y
- las distribuciones más recientes que utilizan el kernel *Linux* requieren, de media, unos 2 GB de espacio en disco, 512 MB de memoria RAM y un fichero de paginación que sea tan grande como la memoria RAM.

Así, desde el punto de vista del aprovechamiento de los recursos, no resulta adecuado que todas las máquinas virtuales tengan las mismas características. De hacerlo,

- se estarían desperdiando recursos en las máquinas que tienen instalado el software menos exigente en lo que al consumo de recursos se refiere, y
- en muchos casos, no se dispondría de recursos suficientes en las máquinas que tienen instalado software que consume muchos recursos.

Por ello, hemos creado seis familias de máquinas virtuales. Tres de ellas están asociadas a los sistemas operativos *Windows*, y las tres restantes están asociadas a los sistemas operativos *Linux*. Sus características se recogen en los cuadros 4.3.1 y 4.3.2.

Imagen	vCPUs	RAM	Disco		Variante SO
			SO y software	Datos / paginación	
<i>small</i>	1	1 GB	20 GB	2 GB / 2 GB (4 GB en total)	32 bits
<i>medium</i>	2	2 GB	30 GB	4 GB / 4 GB (8 GB en total)	32 bits
<i>big</i>	4	3 GB	40 GB	8 GB / 8 GB (16 GB en total)	64 bits

CUADRO 4.3.1: Características de las imágenes *vanilla Windows*

Imagen	vCPUs	RAM	Disco		Variante SO
			SO y software	Datos / paginación	
<i>small</i>	1	1 GB	5 GB	2 GB / 1 GB (3 GB en total)	64 bits
<i>medium</i>	2	2 GB	10 GB	4 GB / 2 GB (6 GB en total)	64 bits
<i>big</i>	4	3 GB	15 GB	8 GB / 4 GB (12 GB en total)	64 bits

CUADRO 4.3.2: Características de las imágenes *vanilla Linux*

A la hora de fijar las características de las imágenes *Windows*, hemos considerado los siguientes aspectos:

- tras desinstalar los componentes multimedia y las características del sistema operativo que no se usan con frecuencia, una instalación de *Windows 7* ocupa 9 GB (en el caso de una versión de 32 bits) o 12 GB (en el caso de una versión de 64 bits).

- en *Windows*, se recomienda que el tamaño del fichero de paginación sea 1,5 veces el tamaño de la memoria RAM. De acuerdo a nuestra experiencia, en *Windows* se utiliza este fichero de forma intensiva, por lo que hemos decidido que su tamaño sea 2 veces el tamaño de la memoria RAM.
- las instalaciones de programas razonablemente complejos (como *Microsoft Office*) ocupan, en media, unos 2 GB de espacio en disco.
- las versiones de *Windows* de 64 bits utilizan, aproximadamente, el doble de memoria RAM que las de 32 bits. Siempre que no se utilicen las instrucciones de 64 bits de la CPU, cosa que en la actualidad ocurre con muy poca frecuencia, el rendimiento de todas las aplicaciones en ambas versiones es prácticamente el mismo.

Como las familias de imágenes *Windows small* y *medium* no cuentan con 4 GB de RAM, en principio no sacarán partido a un sistema operativo de 64 bits, por lo que en estos casos recomendamos el uso de sistemas operativos de 32 bits.

Por otra parte, aunque la familia de imágenes *Windows big* tampoco cuenta con 4 GB de memoria RAM, sí es posible que, por su complejidad, el software instalado en estas máquinas saque partido a las instrucciones de 64 bits de la CPU, por lo que en este caso recomendamos el uso de un sistema operativo de 64 bits.

De todas formas, estas recomendaciones no tienen por qué cumplirse y, en caso de que se ignoren, el rendimiento de la infraestructura no se verá afectado.

Por otra parte, cuando fijamos las características de las imágenes *Linux* consideramos los siguientes aspectos:

- en la mayoría de distribuciones *Linux*, cuando hay mucho software instalado se suelen utilizar unos 7 GB de espacio en disco (menos si se usa un entorno de escritorio ligero).
- las versiones de 64 bits de *Linux* usan la misma cantidad de RAM y espacio en disco que las de 32 bits. Además, permiten utilizar las instrucciones de 64 bits de los procesadores del servidor. Por ello, recomendamos el uso de versiones de *Linux* de 64 bits.

Nuevamente, el rendimiento de la infraestructura no se verá afectado si se ignora esta recomendación.

Finalmente, como el lector habrá notado ya, todas las máquinas virtuales tienen asignada una generosa cantidad de espacio en disco. Con esto, pretendemos incentivar el uso de las máquinas virtuales con menos CPUs y menos memoria RAM, ya que estos recursos son mucho más escasos que el espacio en disco.

Además, tal y como contamos en la sección 4.3.18, las imágenes de disco que utilizamos crecen dinámicamente a medida que se escribe en el disco duro de la máquina virtual. Por ello, el espacio libre en las imágenes no ocupará espacio en disco en el servidor ni en el repositorio de imágenes, y la existencia de discos duros de gran tamaño infrautilizados no supone un desperdicio de recursos.

4.3.20. Uso de los servidores de edición de imágenes

La creación y edición de imágenes de disco requiere, entre otras cosas,

- el intercambio de ficheros de gran tamaño entre el repositorio de imágenes y un servidor de máquinas virtuales, lo que consume mucho ancho de banda, y
- la compresión y descompresión de ficheros de gran tamaño en el servidor de máquinas virtuales, lo que consume mucho tiempo de CPU y realiza muchos accesos a disco.

Por tanto, si un servidor de máquinas virtuales que se utiliza para editar imágenes alberga otras máquinas virtuales, el rendimiento de estas se degradará cuando

- se compriman y descompriman ficheros, o cuando

- se transfieran ficheros entre el servidor de máquinas virtuales y el repositorio de imágenes, ya que en estos casos el tráfico FTP, el tráfico VNC y el tráfico generado por las propias máquinas virtuales competirán por el ancho de banda del enlace que conecta el servidor de máquinas virtuales a la red de la UCM.

Así, si no tomamos medidas, un reducido número de usuarios que pueden crear o editar imágenes de disco (los profesores y los administradores) pueden perjudicar gravemente a la gran mayoría de usuarios, que se limitan a trabajar con las máquinas virtuales ya configuradas.

Para evitar esto, hemos decidido que los procesos de creación y edición de imágenes sólo tengan lugar en cierto número de servidores de máquinas virtuales, los **servidores de edición de imágenes**.

4.3.21. Cifrado selectivo del tráfico

El tráfico que genera el sistema *CygnusCloud* puede clasificarse en dos grupos:

- tráfico generado por el protocolo de escritorio remoto VNC. Se trata del tráfico mayoritario, y permite a los usuarios manipular sus máquinas virtuales utilizando las interfaces gráficas a las que están acostumbrados.
- tráfico generado por los protocolos de *CygnusCloud*. Este tráfico es minoritario, y se genera al procesar peticiones como la instancia y destrucción de máquinas virtuales, la recopilación de estadísticas dentro de los distintos *clusters*, ...

Considerando que la red de la UCM es ya de por sí segura, no sería necesario cifrar el tráfico. No obstante, para garantizar el correcto funcionamiento del sistema *CygnusCloud*, todas las comunicaciones salvo las conexiones a escritorio remoto están protegidas mediante cifrado SSL. Esto nos permite:

- ahorrar ancho de banda. Como el tráfico mayoritario está sin cifrar, es posible utilizar el ancho de banda para soportar más conexiones a escritorio remoto (y, por tanto, dar servicio a un mayor número de usuarios).
- monitorizar fácilmente las actividades que realizan los usuarios en sus máquinas virtuales.
- aumentar la robustez del sistema frente a ataques desde la propia red de la UCM. Gracias al cifrado de los datos generados por los protocolos, lo único que un atacante puede controlar con facilidad es una única máquina virtual, pero no la infraestructura de *CygnusCloud*.

4.3.22. Uso del framework web2py

Web2py es una plataforma web de código abierto (licencia GPL versión 2) que permite un ágil desarrollo de aplicaciones web seguras, gestionadas por medio de bases de datos. Esta escrita y es programable en python y contiene todos los componentes necesarios para construir aplicaciones completamente funcionales.

Al ser un framework web, web2py ofrece un rígido diseño de cara a la seguridad. Así, es capaz de resolver ciertas vulnerabilidades de forma automática siguiendo unas prácticas bien establecidas. Por ejemplo, web2py gestiona automáticamente el formateo de las entradas y salidas, aplica acciones de encriptado en los ficheros subidos y controla el área de maniobra de los desarrolladores de cara a la seguridad del sistema.

Además web2py ofrece una capa de abstracción de base de datos (DAL por su acrónimo inglés), que genera código SQL de forma transparente al modelo de gestión de datos utilizado (SQLite, MySQL, PostgreSQL, Oracle ...).

Modelo Vista-Controlador

Web2py fuerza al desarrollador a utilizar unas buenas prácticas de ingeniería del software, evitando la repetición de código en la medida de lo posible y estandarizando la manera de hacer las cosas.

Por ello, este framework utiliza un modelo Vista-Controlador que incentiva al desarrollador a separar la forma de representar los datos (la vista) y el flujo de trabajo de la aplicación (el controlador). Este modelo ofrece una estructura modular que permite aislar los posibles errores en una sección de código concreta y trabajar en paralelo de forma independiente, reduciendo así los posibles tiempos de espera en el desarrollo de la web.

El flujo de trabajo típico de una petición web2py viene representado en el diagrama 4.7.

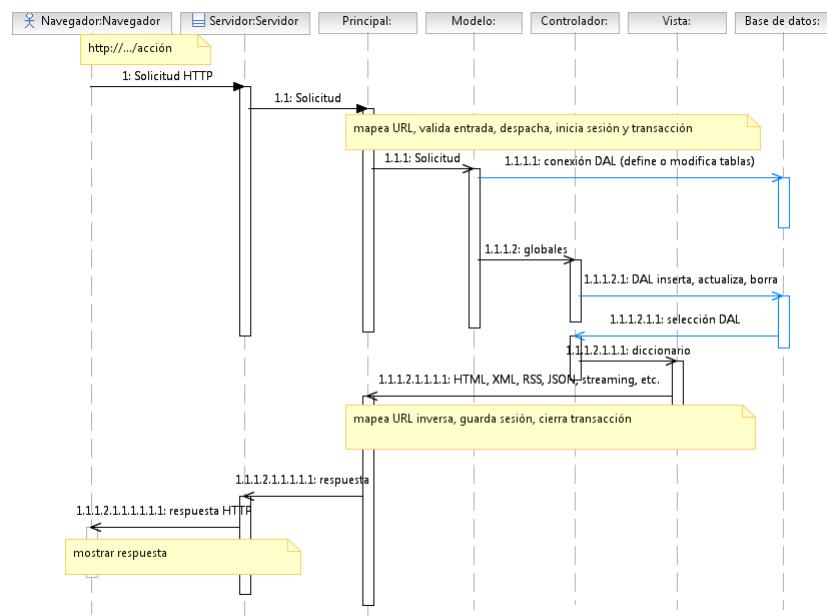


FIGURA 4.7: Diagrama de secuencia web2py

En el diagrama podemos destacar los siguientes elementos:

- El navegador, a partir del cual el usuario se conectará a la web.
- El servidor, puede ser el servidor incluido en web2py o un servidor de terceros, tal como Apache.
- La aplicación principal, encargada de realizar todas las tareas comunes tal y como la gestión de cookies, sesiones, transacciones y enrutamiento.
- Los modelos, vistas y controladores que componen la aplicación del desarrollador.
- La Base de datos que se encargará de almacenar la información utilizada por la web.

Todas las llamadas se encierran en una transacción. Cualquier excepción no contemplada hace que la transacción se cancele.

Con web2py es posible registrar tareas recurrentes para ejecutarse en horas específicas y/o después de determinadas acciones. También es posible alojar varias aplicaciones en una misma instancia. [Pie13]

Ventajas

Las principales razones por las cuales hemos decidido utilizar web2py frente a otros frameworks web basados en python tales como Django o Grok son:

- Ofrece una estructura sencilla que permite a los usuarios aprender sobre el desarrollo web sin comprometer la funcionalidad del sistema. Por esta razón, web2py no requiere instalación ni configuración, no tiene dependencias, y expone la mayor parte de su funcionalidad a través de una interfaz de navegador web.
- Se ha mantenido estable desde el primer día, ofreciendo un diseño de arriba a abajo que asegura una total compatibilidad con respecto a aplicaciones que fueron realizadas utilizando versiones anteriores.
- Ataca de manera pro activa las cuestiones de seguridad más relevantes en las aplicaciones web modernas.
- Ofrece interfaces administrativas para simplificar la creación y gestión de las bases de datos (appAdmin) y la interacción de las diferentes vistas y controladores.
- Es ligero. Todas las librerías y código necesario para desarrollar aplicaciones ocupa en torno a 2 MB.
- Es rápido, ofreciendo una velocidad un 30 % superior que un servidor Apache medio.

Seguridad

Con respecto a la seguridad, web2py se centra en resolver los 10 principales problemas de seguridad definidos por OWASP (Proyecto de Seguridad de Aplicaciones Web Abiertas), una comunidad mundial libre enfocada en la mejora de la seguridad de aplicaciones de software [[OWA11](#)].

En términos generales, estos 10 problemas son:

- Cross Site Scripting (XSS)
- inyecciones SQL
- Ejecución de archivos maliciosos
- Referencia directa a objetos
- Cross Site Request Forgery (CSRF)
- Fugas de información y manejo de errores inapropiado
- Administración de sesiones y autentificación rota
- Almacenamiento criptográfico inseguro
- Comunicaciones inseguras
- Restricciones de acceso URL

4.4. Vista lógica

4.4.1. Visión general

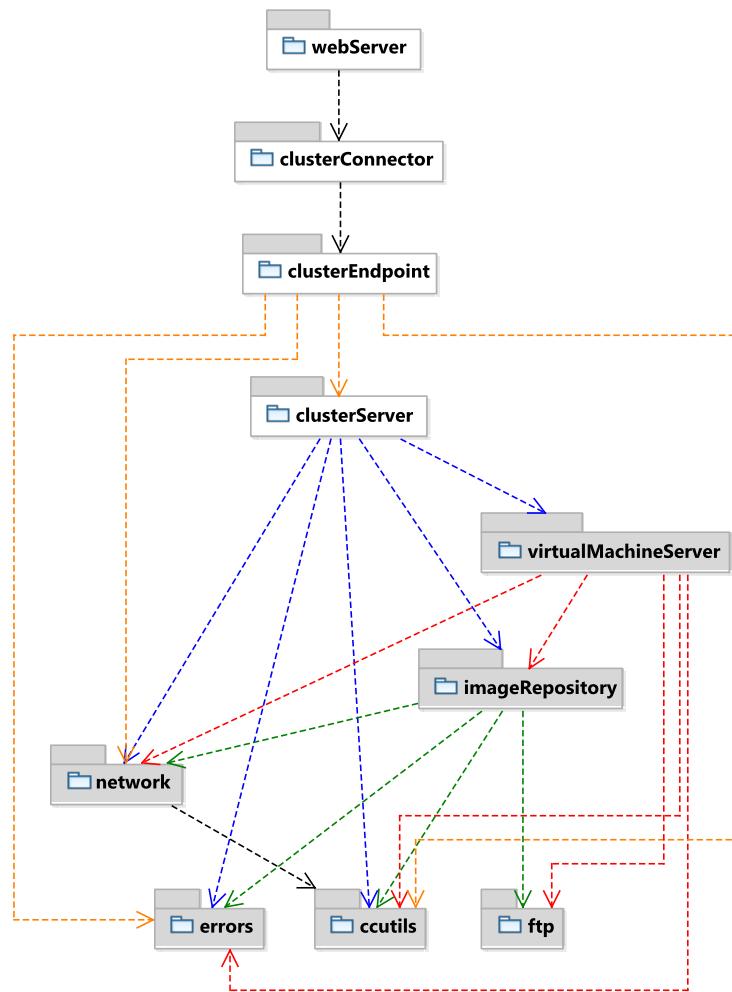


FIGURA 4.1: Diagrama de paquetes de *CygnusCloud*

La funcionalidad del sistema *CygnusCloud* está distribuida en once paquetes:

- el paquete **ccutils** contiene clases que se utilizan en varios subsistemas de *CygnusCloud*.
- el paquete **network** contiene las clases que permiten comunicar un conjunto de máquinas distintas a través de una red. Dichas comunicaciones se realizan utilizando el protocolo TCP a través de la librería de red *twisted*.
- el paquete **virtualMachineServer** contiene las clases que implementan el subsistema servidor de máquinas virtuales.
- el paquete **clusterServer** contiene todas las clases que implementan el subsistema servidor del *cluster*.
- el paquete **imageRepository** contiene todas las clases que implementan el subsistema repositorio de imágenes.

- los paquetes `clusterEndpoint` y `clusterConnector` contienen las clases que permiten comunicar la página web de *CygnusCloud* con un servidor de *cluster*.
- el paquete `errors` contiene un tipo enumerado con todos los códigos de error generados por la infraestructura.
- el paquete `webServer` contiene las vistas y controladores asociados a la página web de *CygnusCloud*.
- el paquete `ftp` contiene el cliente y el servidor FTP utilizados para transferir imágenes de disco entre el repositorio de imágenes y los servidores de máquinas virtuales.
- finalmente, el paquete `testing` contiene clases utilizadas para depurar los sistemas servidor de máquinas virtuales y servidor del *cluster*.

Las relaciones existentes entre estos paquetes aparecen en la figura 4.1. Por claridad,

- las dependencias del paquete `virtualMachineServer` aparecen en rojo,
- las dependencias del paquete `clusterServer` aparecen en azul,
- las dependencias del paquete `imageRepository` aparecen en verde,
- las dependencias del paquete `clusterConnector` aparecen en naranja, y
- el resto de dependencias entre paquetes aparecen en negro.

Finalmente, algunos de los paquetes que acabamos de presentar se descomponen en otros paquetes más pequeños. Para facilitar la visualización de las dependencias principales entre paquetes, hemos decidido omitir dichas descomposiciones. A medida que presentemos el diseño, las mostraremos junto con las relaciones entre clases que justifican las dependencias entre paquetes.

4.4.2. Paquetes y clases significativos de la arquitectura

En esta sección, mostraremos la estructura interna de los paquetes más significativos de la arquitectura. También mostraremos sus clases más relevantes, junto con una breve descripción de sus responsabilidades.

Aunque el paquete `errors` es significativo desde el punto de vista de la arquitectura, sólo contiene un tipo enumerado, en el que se definen los códigos de error. Por ello, lo omitiremos en esta sección.

4.4.2.1. ccutils

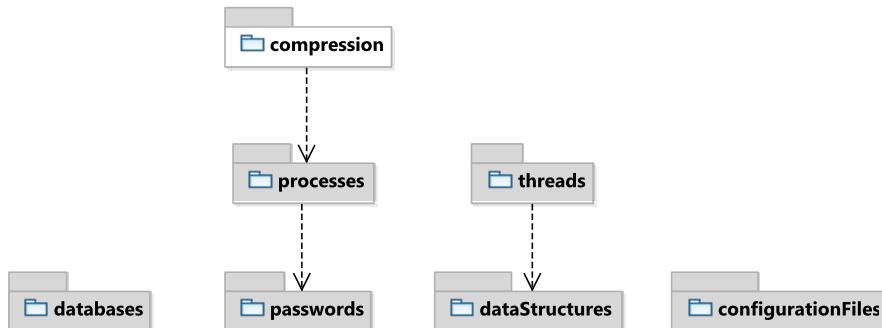


FIGURA 4.2: Descomposición del paquete ccutils

El paquete `ccutils` contiene clases que se utilizan en múltiples subsistemas de *CygnusCloud*. Como puede observarse en el diagrama de paquetes de la figura 4.2, el paquete `ccutils` se descompone en siete subpaquetes:

- el paquete `databases` contiene dos clases que permiten manipular una base de datos MySQL. La primera de ellas, `DBConfigurator`, permite crear, destruir y configurar una base de datos a través del conector no oficial de MySQL `mysqldb`. La segunda, `BasicDBConnector`, utiliza el conector oficial de MySQL para realizar consultas y actualizaciones sobre una base de datos. Por ello, es el antepasado común del resto de clases que manipulan una base de datos.
- el paquete `passwords` contiene una única clase, `RootPasswordHandler`, que permite obtener y recuperar la contraseña del usuario `root`.
- la principal clase del paquete `processes`, `ChildProcessManager`, permite lanzar procesos hijos en *background* y en *foreground*, bien como el usuario actual o bien como `root`.
- el paquete `dataStructures` contiene envoltorios de las principales estructuras de datos de la librería estándar de *Python*. Estos envoltorios permiten utilizar dichas estructuras de datos en entornos multihilo de forma segura.
- el paquete `threads` contiene dos clases: `BasicThread` y `QueueProcessingThread`. La primera es el antepasado común de todos los hilos de la infraestructura. Por otra parte, la segunda se corresponde con un hilo que, como su nombre indica, procesa los elementos de una cola.
- finalmente, las clases del paquete `configurationFiles` se utilizan para procesar los distintos ficheros de configuración. La más relevante de todas, `ConfigurationFileParser`, procesa los ficheros de configuración haciendo uso del módulo `ConfigParser` de la librería estándar de *Python*.

4.4.2.2. network

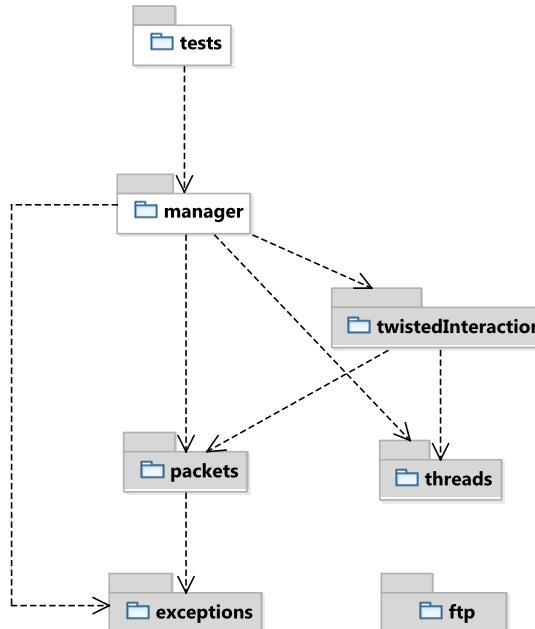


FIGURA 4.3: Descomposición del paquete network

Las clases del paquete `network` interactúan con la librería de red `twisted` para comunicar una máquina con otra u otras conectadas a la misma red. El diagrama de paquetes de la figura 4.3 muestra que este paquete se descompone en ocho subpaquetes:

- el paquete `exceptions`, que contiene todas las clases de excepción que utilizan las clases del paquete `network`.
- el paquete `packets`, que contiene la clase `Packet`. Esta clase se ocupa de la serialización y deserialización de la información que intercambian los equipos.
- el paquete `ftp`. Sus clases más importantes son las siguientes:
 - `FTPClient`, que permite interactuar con el cliente FTP de la librería estándar de *Python* a un elevado nivel de abstracción.
 - `ConfigurableFTPServer`, que ofrece una interfaz de alto nivel para interactuar con el servidor FTP `pyftpdlib`.
 - `FTPCallback`, que define la interfaz que se usará para procesar los eventos generados por el servidor FTP `pyftpdlib`.
- el paquete `threads`. Este contiene las clases `ConnectionMonitoringThread`, `DataProcessingThread` y `TwistedReactorThread`, correspondientes a los distintos tipos de hilo que se utilizan en la implementación de la red.
- el paquete `twistedInteraction`, que contiene las clases
 - `Connection`, `ServerConnection` y `ClientConnection`, asociadas a las conexiones de red,
 - `ConnectionStatus`, asociada al estado de una conexión de red, y
 - `CygnusCloudProtocol` y `CygnusCloudProtocolFactory`, utilizadas para interactuar con la librería de red `twisted` a muy bajo nivel de abstracción.
- el paquete `manager`, que contiene las clases `NetworkManager` y `NetworkCallback`. La primera ofrece una interfaz de alto nivel para manipular conexiones de red, y la segunda define la interfaz que se utiliza para procesar los datos recibidos a través de la red.
- el paquete `tests`, que contiene las pruebas de los módulos de este paquete.
- el paquete `ftp`. Sus principales clases son `FTPClient`, correspondiente a un cliente FTP basado en el de la librería estándar de *Python*, y `ConfigurableFTPServer`, un servidor FTP basado en `pyftpdlib`.

4.4.2.3. `ftp`

Este paquete contiene las clases del cliente y del servidor FTP que utilizan el repositorio de imágenes y los servidores de máquinas virtuales para intercambiar imágenes de disco. Sus principales clases son las siguientes:

- `FTPClient`. Esta clase es un envoltorio del cliente FTP incluido en la biblioteca estándar de *Python*.
- `ConfigurableFTPServer`. Esta clase es un envoltorio del servidor FTP `pyftpdlib`.

Asimismo, el paquete `ftp` incluye un único subpaquete, `pyftpdlibInteraction`. Como su nombre indica, las clases de ese paquete realizan la interacción con el servidor `pyftpdlib` a un bajo nivel de abstracción.

4.4.2.4. imageRepository

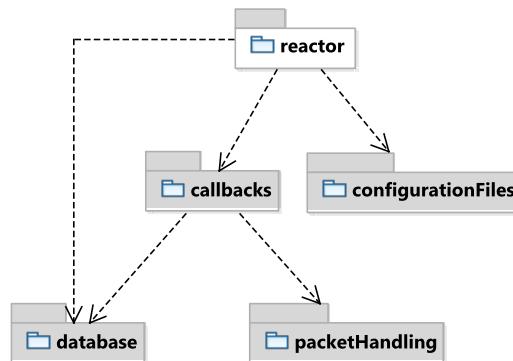


FIGURA 4.4: Descomposición del paquete imageRepository

Este paquete contiene las clases del subsistema repositorio de imágenes. Como puede observarse en el diagrama de paquetes de la figura 4.4, el paquete `imageRepository` se descompone en cinco subpaquetes:

- el paquete `database`. Contiene la clase `ImageRepositoryDBConnector`, que permite manipular la base de datos del repositorio de imágenes.
- el paquete `callbacks`, que contiene las interfaces `FTPServerCallback` y `CommandsCallback`. Estas se utilizarán para procesar los eventos generados por el servidor FTP `pyftpdlib` y para procesar los paquetes recibidos respectivamente.
- el paquete `configurationFiles`. Su única clase, `ImageRepositoryConfigurationFileParser`, parsea el fichero de configuración del demonio del repositorio de imágenes.
- el paquete `packetHandling`. Incluye una única clase, `ImageRepositoryPacketHandler`, permite crear y leer los distintos tipos de paquete que se utilizan en el repositorio de imágenes.
- el paquete `reactor`. Su única clase, `ImageRepositoryReactor`, se ocupa de realizar los procesos de inicialización y apagado del demonio del repositorio de imágenes.

4.4.2.5. virtualMachineServer

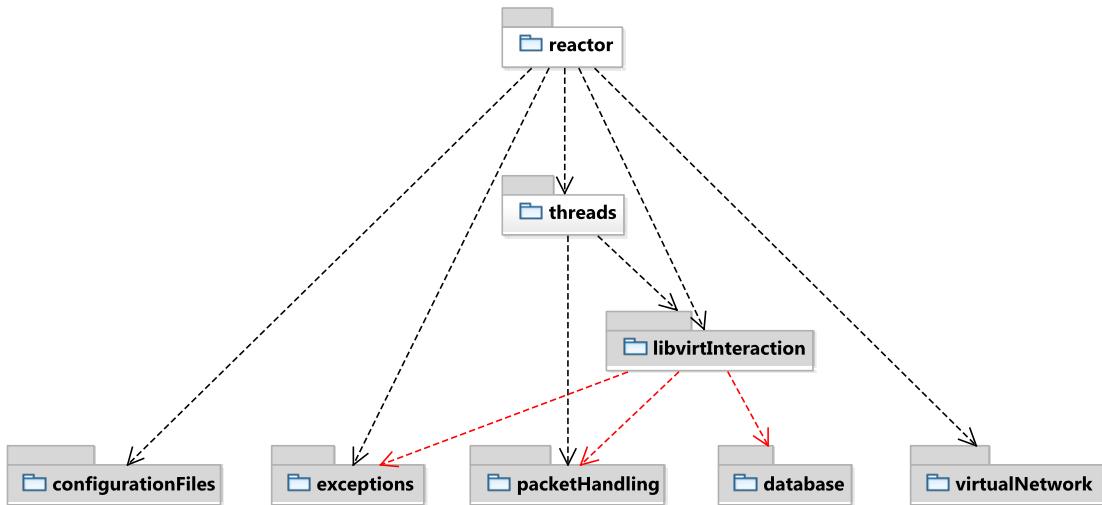


FIGURA 4.5: Descomposición del paquete virtualMachineServer

Este paquete contiene las clases del subsistema servidor de máquinas virtuales. El diagrama de paquetes de la figura 4.5 muestra la descomposición del paquete virtualMachineServer. Por claridad, las dependencias del paquete libvirtInteraction aparecen en rojo, y el resto de dependencias aparecen en negro.

Como puede observarse en el diagrama de paquetes de la figura 4.5, el paquete virtualMachineServer se descompone en siete subpaquetes:

- **configurationFiles**. La única clase que contiene, VMServerConfigurationFileParser, parsea el fichero de configuración del demonio del servidor de máquinas virtuales.
- **database**. Su clase más relevante, VMServerDBConnector, permite manipular la base de datos del servidor de máquinas virtuales.
- **exceptions**. Este paquete contiene la clase VMServerException, la clase de excepción asociada a la mayoría de clases del subsistema servidor de máquinas virtuales.
- **libvirtInteraction**. Este paquete contiene las clases DomainHandler, LibvirtConnector y ConfigurationFileEditor, que se utilizan para crear y destruir dominios interactuando con la librería libvirt.
- **packetHandling**. Este paquete contiene la clase VMServerPacketHandler, que permite crear y leer los distintos tipos de paquete asociados al servidor de máquinas virtuales.
- **reactor**. La principal clase de este paquete, VMServerReactor, procesa los paquetes enviados por el servidor de *cluster*.
- **threads**. Este paquete contiene las clases de hilo CompressionThread y FileTransferThread. Estas están asociadas a la compresión y descompresión de imágenes de disco y al intercambio de imágenes de disco con el repositorio de imágenes respectivamente.
- **virtualNetwork**. La única clase de este paquete, VirtualNetworkManager, dispone de métodos para crear y destruir redes virtuales.

4.4.2.6. clusterServer

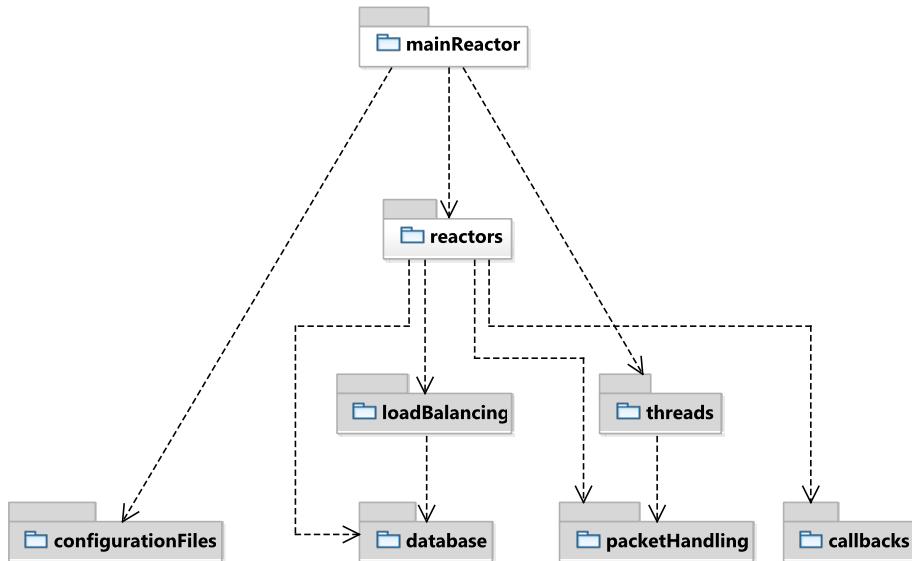


FIGURA 4.6: Descomposición del paquete clusterServer

Tal y como se muestra en el diagrama de paquetes de la figura 4.6, el paquete clusterServer se descompone en ocho subpaquetes:

- **configurationFiles**. Nuevamente, este paquete contiene la clase que procesa el fichero de configuración del demonio del servidor de *cluster*, ClusterServerConfigurationFileParser.
- **callbacks**. En él se definen las interfaces ClusterEndpointCallback, ImageRepositoryCallback y VMServerCallback, que se usarán para procesar los paquetes recibidos desde el servidro web, el repositorio de imágenes y los servidores de máquinas virtuales respectivamente.
- **database**. Su clase más importante, ClusterServerDBConnector, permite manipular la base de datos del servidor de *cluster*.
- **loadBalancing**. Contiene la interfaz LoadBalancer, común a todos los algoritmos de balan- ceado de carga, y la clase PenaltyBasedLoadBalancer, asociada al algoritmo de balanceado de carga basado en penalizaciones.
- **mainReactor**. Contiene la clase ClusterServerMainReactor, el punto de entrada del demo- nio del servidor de *cluster*.
- **packetHandling**. Contiene la clase ClusterServerPacketHandler, que permite manipular los paquetes que intercambian el servidor de *cluster* y el servidor web.
- **reactors**. Contiene las clases EndpointPacketReactor, ImageRepositoryPacketReactor, VMServerPacketReactor y NetworkEventsReactor. Las tres primeras procesan los paquetes recibidos desde el servidor web, el repositorio de imágenes y el servidor de máquinas virtuales respectivamente. La última procesa los eventos de desconexión y reconexión gene- rados por la red.
- **threads**. Su única clase, ClusterStatusMonitoringThread, se corresponde con un hilo que envía periódicamente las actualizaciones de estado a todas las máquinas del *cluster*.

4.4.2.7. clusterEndpoint

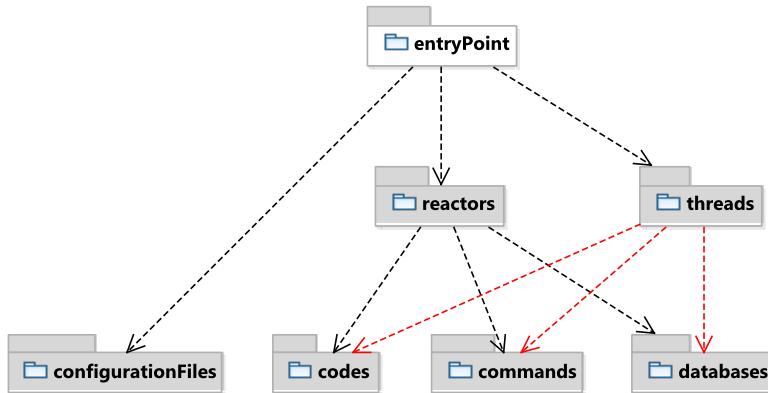


FIGURA 4.7: Descomposición del paquete clusterEndpoint

El diagrama de paquetes de la figura muestra la descomposición del paquete clusterEndpoint. Como se puede apreciar en él, el paquete clusterEndpoint se descompone en siete subpaquetes:

- **codes**. Este paquete contiene la interfaz CodesTranslator, que se utilizará para convertir a *strings* los códigos de error generados por la infraestructura, y la clase SpanishCodesTranslator, que genera esos *strings* en castellano.
- **commands**. Este paquete contiene la clase CommandsHandler, que realiza la serialización y deserialización de los comandos y de las salidas de comandos que usa la página web para interactuar con la infraestructura.
- **configurationFiles**. Este paquete contiene la clase ClusterEndpointConfigurationFileParser, que manipula el fichero de configuración del demonio que conecta la página web con la infraestructura.
- **databases**. Sus principales clases son CommandsDBConnector y ClusterEndpointDBConnector, que permiten manipular las dos bases de datos que utiliza el servidor web.
- **entryPoint**. Su única clase, ClusterEndpointEntryPoint, es el punto de entrada del demonio que conecta la página web con la infraestructura.
- **reactors**. Contiene las clases CommandsProcessor y PacketReactor, que se usan para procesar las órdenes enviadas a través de la página web y los paquetes enviados desde el servidor de cluster respectivamente.
- **threads**. Contiene las clases de hilo CommandsMonitoringThread y DatabaseUpdateThread, que se utilizan para detectar *timeouts* en la ejecución de los comandos que envía la página web y para recopilar periódicamente el estado de la infraestructura respectivamente.

4.4.2.8. clusterConnector

El paquete clusterConnector contiene una única clase, ClusterConnector, que permite manipular las bases de datos del servidor web y enviar peticiones a la infraestructura desde la página web.

4.4.2.9. testing

El paquete testing contiene clases que permiten comprobar el correcto funcionamiento de los demonios del repositorio de imágenes, del servidor de máquinas virtuales y del servidor de cluster. Estas clases son las siguientes:

- `ImageRepositoryTester`. Esta clase interactúa con el demonio del repositorio de imágenes tal y como lo harían el demonio del servidor de máquinas virtuales y el demonio del servidor de *cluster*. Por ello, permite comprobar el correcto funcionamiento de todas las funciones del repositorio de imágenes.
- `VirtualMachineServerTester`. Esta clase interactúa con el demonio del servidor de máquinas virtuales tal y como lo haría el demonio del servidor de *cluster*. Así pues, permite comprobar el correcto funcionamiento de todas las funciones del servidor de máquinas virtuales.
- `ClusterServerTester`. Esta clase interactúa con el demonio del servidor de *cluster* tal y como lo haría el servidor web. Esto permite comprobar el correcto funcionamiento de todas las funciones del servidor de *cluster*.

4.4.2.10. webServer

El paquete `webServer` se encuentra distribuido en un conjunto de paquetes que hacen referencia a los diferentes tipos de elementos necesarios para implementar la funcionalidad de la web. Estos paquetes son:

- **Vistas**: Paquetes con el conjunto de vistas que definen la representación de las páginas
- **Controladores** : Paquete que contiene los módulos python encargados de manejar las diferentes funcionalidades de la web.
- **Modelos** : Incluye dos módulos python encargados de definir dos aspectos importantes de la web. El primer módulo, `menu`, introduce la información sobre la versión , las palabras clave y todos los aspectos generales de la web. El segundo, `db`, se encarga de crear las tablas necesarias que serán utilizadas para almacenar la información manejada por la web.
- **Lenguajes** : Incluye un conjunto de diccionarios que permiten traducir la web a diferentes idiomas.
- **Archivos estáticos** : Este paquete incluye las `css` , `js` e `imagenes` necesarias para definir el aspecto de la web. También incluye el controlador `noVNC`.
- **Módulos** : En este paquete se encuentran todos los módulos python secundarios que serán utilizados por los controladores.

4.4.3. El paquete `network`

Los módulos de este paquete proporcionan una interfaz que permite utilizar la librería de red `twisted` a un alto nivel de abstracción. Puesto que el diseño de esta parte del sistema está íntimamente relacionado con el funcionamiento de la librería de red `twisted`, es fundamental que el lector esté familiarizado con los conceptos básicos de esta librería.

Por ello, comenzaremos mostrando, en líneas generales, cómo funciona la librería de red `twisted`. Posteriormente, describiremos el contenido de este paquete en orden creciente del nivel de abstracción, profundizando más en el funcionamiento de `twisted` cuando sea preciso.

En cualquier caso, dado que `twisted` es una librería con muchas funciones, sólo mostraremos los aspectos de su funcionamiento más relacionados con el diseño del paquete `network`. Si el lector está interesado en ampliar la información que aquí le proporcionamos, le remitimos a la documentación oficial ([[Twi12](#)]) y al libro *Twisted Network Programming Essentials* ([[Fet05](#)]).

4.4.3.1. La librería de red `twisted`: visión general

Tal y como adelantamos en la sección 4.3.12, `twisted` es una librería de red totalmente basada en eventos. Así, todas las tareas que realiza (como la recepción de datos, el establecimiento de conexiones, la detección de desconexiones, ...) son totalmente asíncronas.

Para generar los eventos, *twisted* utiliza el patrón *reactor*, es decir, muestrea periódicamente el estado de los *sockets* y actúa en consecuencia. De esta manera, todas las aplicaciones que utilicen *twisted* seguirán el siguiente esquema de comportamiento:

1. el código de la librería muestrea los *sockets* abiertos por la aplicación para detectar cambios.
2. cuando se detecta un cambio, se genera un evento para informar del cambio al código del cliente.
3. el código del cliente utiliza la información asociada al evento y procesa el cambio.
4. cuando termina, el código del cliente devuelve el control al código de la librería.

El muestreo de los *sockets* y el tratamiento de los cambios tiene lugar en un bucle que se ejecuta permanentemente: el *bucle reactor*. La figura 4.8 resume de forma gráfica las ideas que acabamos de exponer.

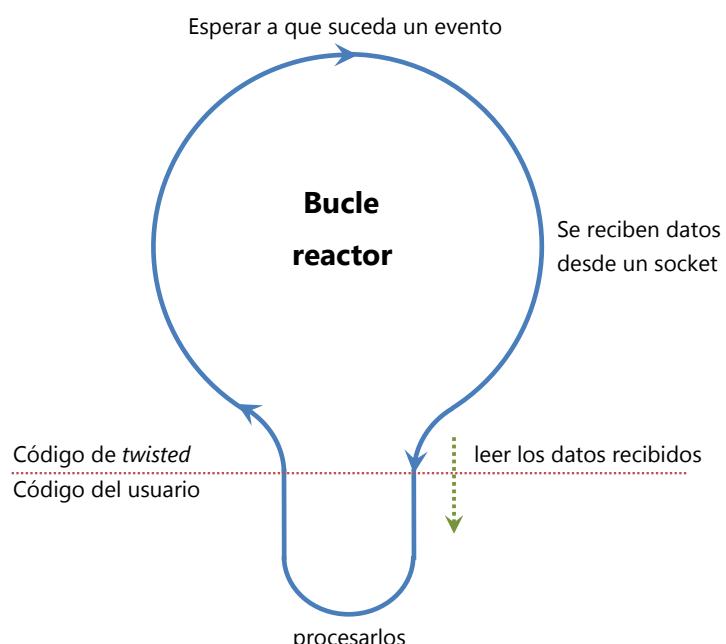


FIGURA 4.8: Representación gráfica del funcionamiento de la librería *twisted*

Protocolos y factorías de protocolos

La forma en que se procesan los datos recibidos a través de la red varía de unas aplicaciones a otras, e incluso puede variar en una misma aplicación. Por ejemplo, en *CygnusCloud* algunos datos recibidos se escriben en una base de datos, otros se reenvían y otros se le muestran directamente al usuario. Por ello, es necesario que el proceso de generación y tratamiento del evento no dependa de los datos concretos que manipula cada aplicación. Para hacer esto posible, *twisted* define los *protocolos* y las *factorías de protocolos*.

Un *protocolo* es una clase cuyos métodos tratan y generan eventos de red, tales como la recepción de un segmento TCP, el envío de un segmento TCP y la detección de desconexión. Por su parte, las *factorías de protocolos* son objetos capaces de instanciar y configurar protocolos. Los protocolos y las factorías de protocolos se manipulan siempre a través de dos interfaces, que se definen en las clases abstractas *Protocol* y *Factory* respectivamente.

El bucle reactor

Su misión principal es muestrear periódicamente todos los *sockets* abiertos e invocar a las rutinas de tratamiento adecuados cuando se produzca un evento. Como ya hemos dicho, las rutinas de tratamiento se definen en instancias de subclases concretas de *Protocol*.

En la librería *twisted* se definen varios bucles reactor, que se diferencian fundamentalmente por la forma de realizar el muestreo periódico. Nosotros utilizamos el que está definido en la clase *PollReactor*, que se limita a leer periódicamente todos los *sockets* para determinar si se han producido cambios.

Para que el funcionamiento de la librería sea el correcto, deben cumplirse dos restricciones:

- aunque en cada aplicación puede haber un número arbitrario de conexiones activas, sólo puede existir *un* único bucle reactor.
- una vez que se detiene el bucle reactor, este *no* puede reiniciarse mientras la aplicación se siga ejecutando. Por ello, sólo se puede salir de este bucle cuando la aplicación no va a utilizar más la red.

Por otra parte, los eventos de red se tratan en el propio bucle del reactor. Para garantizar un tiempo de respuesta adecuado, es imprescindible que los métodos de tratamiento sean tan rápidos como sea posible.

Procesamiento de datos entrantes

El diagrama de secuencia de la figura 4.9 muestra cómo se procesan los eventos asociados a dos conexiones, cuyos protocolos son instancias de las subclases de *Protocol* *Protocol1* y *Protocol2*.

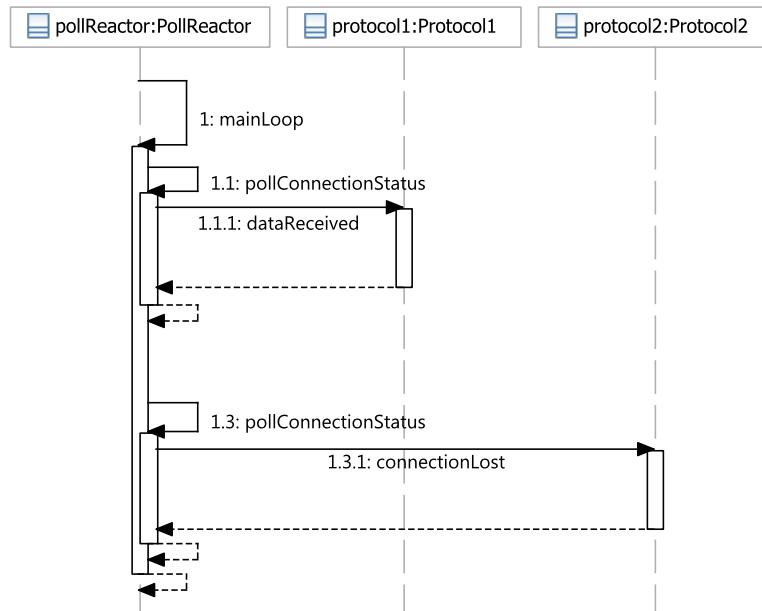


FIGURA 4.9: Procesamiento de datos entrantes en *twisted*: diagrama de secuencia

En ambos casos, se muestrea el *socket*, para posteriormente invocar al método correspondiente del protocolo: el método de procesamiento de los datos entrantes en el primer caso, y el método de procesamiento de una desconexión en el segundo.

Establecimiento de conexiones

Para establecer una conexión, es necesario tener en cuenta

- el protocolo de transporte que hay que utilizar (TCP versión 4, TCP versión 4 sobre SSL,...), y
- el papel que desempeña la máquina en la conexión (cliente o servidor).

Para eliminar en la medida de lo posible los detalles de bajo nivel de nuestro código, hemos utilizado una función que se ha añadido recientemente a la librería *twisted*: los *endpoints*.

Un *endpoint* sirve para configurar uno de los extremos de la conexión. Todos los protocolos de transporte implementados en *twisted* definen dos: uno para la máquina servidor y otro para las máquinas cliente. En el proceso de conexión, el *endpoint*, que es una instancia de una subclase concreta de *Endpoint*,

- interactúa con el reactor para establecer la conexión.
- suministra al reactor la factoría de protocolos. Este la utilizará para instanciar protocolos cuando sea necesario.

Las relaciones existentes entre la clase *Endpoint* y el resto de clases principales de la librería *twisted* aparecen en el diagrama de clases de la figura 4.10.

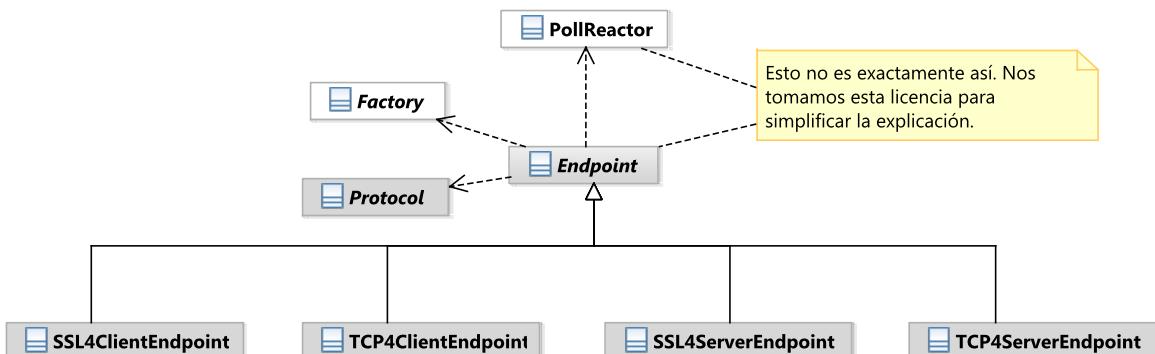


FIGURA 4.10: Relaciones entre protocolos, factorías de protocolos y *endpoints*

Todos los *endpoints* son instancias de alguna subclase concreta la clase *Endpoint*. En el diagrama sólo aparecen las subclases de *endpoint* que utilizamos en el paquete *network*, correspondientes a los protocolos TCP versión 4 y TCP versión 4 sobre SSL. Por otra parte, los *endpoints* no manipulan directamente un *PollReactor*: para ello, utilizan las interfaces que esta clase implementa. Nos hemos tomado esta licencia para simplificar la explicación.

Como ya hemos visto, los protocolos se crean al establecerse la conexión. Además, puesto que siempre están ligados a una conexión, estos objetos se destruyen cuando esta se cierra.

Por otra parte, el diagrama de secuencia de la figura muestra la interacción que tiene lugar al establecer dos conexiones TCP, una de tipo servidor y otra de tipo cliente. En ambos casos, los *endpoints* se limitan, en esencia, a solicitar al reactor al establecimiento de la conexión y a suministrarle la factoría de protocolos que debe utilizar.

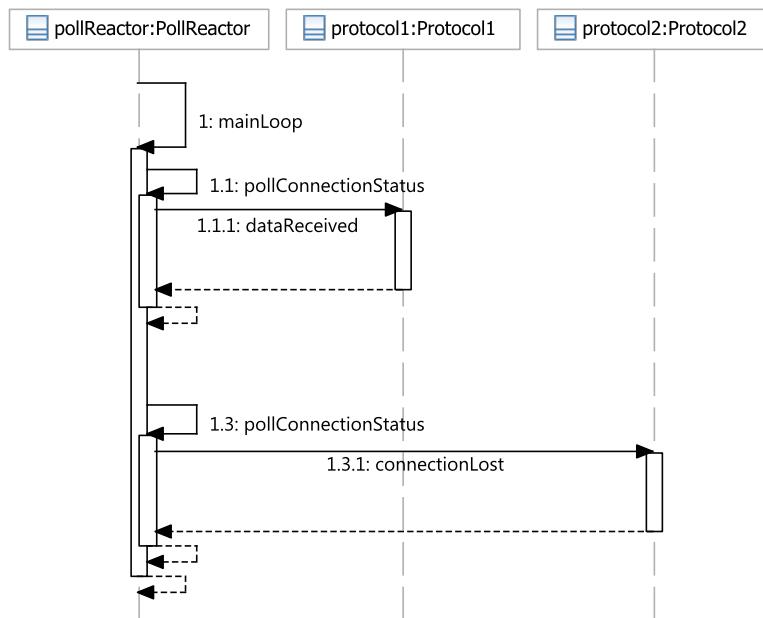


FIGURA 4.11: Establecimiento de una conexión en *twisted*: diagrama de secuencia

Finalmente, los protocolos se usan de forma distinta en función del tipo de conexión:

- en el caso de una conexión de tipo cliente, existirá un único protocolo, que permitirá la comunicación bidireccional con el servidor.
- en el caso de una conexión de tipo servidor, existirán tantos protocolos como clientes. Cada uno de ellos permitirá la comunicación bidireccional entre el servidor y un único cliente.

Envío de datos

Tal y como hemos mencionado, para enviar datos a través de la red se utilizan instancias de subclases concretas de `Protocol`. El reactor nunca interviene a la hora de enviar datos: los `bytes` a enviar se escriben directamente en el `socket` correspondiente.

En principio, la única restricción que impone la librería *twisted* es el tipo de los datos a enviar: sólo es posible enviar *strings* (es decir, secuencias de `bytes`), lo que hace necesario serializar toda la información antes de enviarla y deserializarla al recibirla.

Es importante notar que el número de `bytes` a enviar no está limitado. Por tanto, los segmentos se fragmentarán cuando sea preciso.

4.4.3.2. La clase Packet

Como ya hemos mencionado, el principal objetivo de las clases del paquete `network` es proporcionar una forma de utilizar la librería de red *twisted* a un elevado nivel de abstracción. Para ello, es necesario, como mínimo,

- hacer totalmente transparentes al usuario los procesos de serialización y de deserialización de la información que se envía y recibe, y
- fijar la prioridad de los datos que circulan por la red. Esto es fundamental para que el sistema pueda responder rápidamente ante eventos de suma importancia como la caída de un servidor de máquinas virtuales.

La clase `Packet` cubre estas dos necesidades. Sus instancias representan paquetes de red, y tienen asociada la siguiente información:

- un tipo. Existen dos: uno para transportar datos de gestión de la red y otro para transportar datos del cliente. Los paquetes de gestión de la red son siempre más prioritarios que los que transportan datos del cliente.
- una prioridad. Se trata de un valor entero, positivo en el caso de los paquetes que contienen datos de los usuarios y negativo en el caso de los paquetes de gestión de la red. Cuanto *menor* es la prioridad de un paquete, *más* prioritario será.
- una secuencia de *bytes* con los datos del paquete.

Tipo	Rango de la prioridad	Tamaño máximo (paquete completo)
Datos del cliente	[0, 32767]	64 KB
Gestión de la red	[-32768, -1]	64 KB

CUADRO 4.4.1: Características de los distintos tipos de paquete

El cuadro 4.4.1 recoge las características de los dos tipos de paquete. En ambos casos, el tamaño máximo del paquete completo (es decir, incluyendo la cabecera con el tipo y la prioridad y los datos) no puede exceder los 64 KB. Esto nos permite

- garantizar que no haya usuarios que acaparen el ancho de banda de la red mediante el envío de paquetes de gran tamaño.
- mantener el tiempo de respuesta de la red dentro de unos márgenes razonables. Puesto que los paquetes de pequeño tamaño tardan menos en enviarse y en ser recibidos, los paquetes más prioritarios podrán “adelantar” a los paquetes menos prioritarios que les preceden y llegar a su destino tan rápido como sea posible.
- reducir el tiempo de procesamiento de los paquetes que se van a redirigir.

Serialización y deserialización de los datos

Para hacer la serialización y deserialización de los datos totalmente transparentes a los usuarios, la clase `Packet` dispone de métodos de lectura y escritura de *strings*, valores enteros, valores booleanos y números en punto flotante.

Tal y como mencionamos en la sección 4.4.3.1, la librería `twisted` sólo es capaz de enviar y recibir *strings*, por lo que

- los métodos de escritura convierten a *strings* los valores a escribir en el paquete y los añaden al final de una estructura de datos intermedia, y
- los métodos de lectura extraen *strings* de la estructura de datos intermedia y los convierten en valores del tipo que el usuario quiere leer. Todas las lecturas son destructivas, es decir, cada dato escrito en el paquete sólo puede leerse una vez.

La estructura de datos intermedia es un *string*. Para leer y escribir en ella se usan los métodos de extracción de subcadenas y concatenación definidos en la librería estándar de Python. El formato de los datos serializados viene dado por la siguiente expresión regular:

`(etiqueta de tipo$valor$)*`

Los valores se serializan y deserializan con los métodos definidos en la librería estándar de Python. El cuadro 4.4.2 contiene la codificación de las etiquetas de tipo de los paquetes.

Etiqueta	Tipo de datos
0	entero
1	entero largo
2	string
3	número de punto flotante

CUADRO 4.4.2: Etiquetas de tipo que pueden aparecer en un paquete

También es necesario serializar la cabecera del paquete. Su formato es el siguiente:

tipo, prioridad

Nuevamente, tanto el tipo como la prioridad del paquete se serializan y deserializan utilizando métodos de la librería estándar de Python. El paquete serializado se obtiene concatenando la cabecera y los datos serializados, y su formato viene descrito por esta expresión regular:

tipo, prioridad(etiqueta de tipo\$valor\$)*

Finalmente, los métodos de la clase `Packet` también garantizan que los paquetes siempre están bien formados, es decir, que

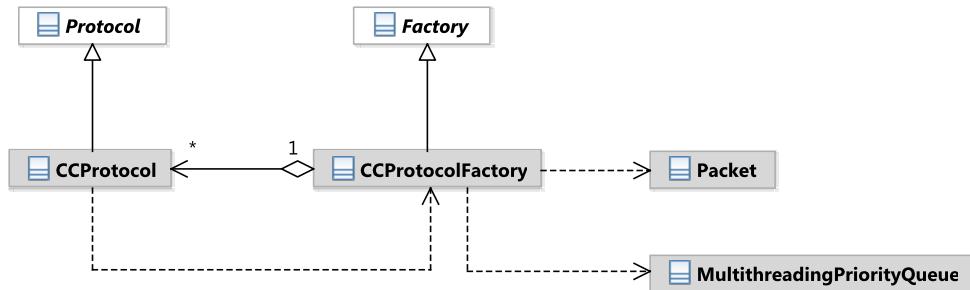
- la prioridad es la adecuada para el tipo de paquete
- no se excede el tamaño máximo del paquete al escribir datos en él
- los datos se leen del paquete en el mismo orden en que se escriben

4.4.3.3. Envío y recepción de datos

Como ya mencionamos en la sección 4.4.3.1, para poder enviar y recibir información utilizando la librería `twisted` es necesario definir dos clases:

- una subclase de `Protocol`, `CygnusCloudProtocol`. Sus instancias procesarán los datos recibidos a través de cada conexión.
- una subclase de `Factory`, `CygnusCloudProtocolFactory`. Sus instancias crearán objetos `CygnusCloudProtocol` cuando se establezcan las conexiones.

Las relaciones entre estas clases, la clase `Packet` y las clases de la librería `twisted` aparecen en el diagrama de clases de la figura 4.12. En dicho diagrama, hemos abreviado `CygnusCloud` utilizando las siglas CC.


 FIGURA 4.12: Interacción con `twisted` a muy bajo nivel de abstracción: diagrama de clases

Lo primero que llama la atención en el diagrama de clases de la figura 4.12 es la dependencia cíclica existente entre las clases `CygnusCloudProtocol` y `CygnusCloudProtocolFactory`. Dicha dependencia está provocada por el funcionamiento de la librería twisted: cada par de máquinas conectadas se comunica siempre a través de una conexión bidireccional. Por ello, si n clientes se conectan a un mismo servidor,

- en los clientes habrá un único objeto `CygnusCloudProtocol`, y
- en el servidor habrá n objetos `CygnusCloudProtocol`, cada uno de los cuales permite al servidor comunicarse con un cliente concreto.

Para simplificar la implementación de la red nos interesa que, en los niveles de mayor nivel de abstracción, sólo sea necesario realizar la distinción entre clientes y servidores a la hora de establecer las conexiones. Lo más conveniente es extender la funcionalidad de la clase `CygnusCloudProtocolFactory` para que, además de instanciar objetos `CygnusCloudProtocol`, sea capaz de:

- mantener una lista de referencias a todos los objetos `CygnusCloudProtocol` asociados a la conexión y actualizarla a medida que los clientes se conecten y desconecten.
- indicar si la conexión se puede utilizar, teniendo en cuenta que una conexión no está lista cuando no tiene asociado ningún objeto `CygnusCloudProtocol`.
- serializar y enviar todos los paquetes del servidor
 - a todos los clientes conectados (envío *multicast*), o
 - a uno de los clientes conectados (envío *unicast*)

según lo que especifique el código cliente. Para ello, se utilizarán las referencias a los objetos `CygnusCloudProtocol`. En caso de que no haya ninguna, los datos a enviar se descartarán.

- procesar los datos recibidos a través de cualquier objeto `CygnusCloudProtocol`, deserializándolos para formar los paquetes y avisando a la capa superior.

La recepción de paquetes es totalmente asíncrona. Por motivos de eficiencia y para respetar el tipo y la prioridad de los paquetes, los objetos `CygnusCloudProtocolFactory` insertan todos los paquetes que reciben en una cola de prioridad, que es una instancia de la clase `MultithreadingPriorityQueue`. La capa superior extraerá de ella los paquetes de acuerdo a su tipo y prioridad.

4.4.3.4. Conexiones de red

Con lo que hemos mostrado hasta ahora, ya podemos interactuar con twisted para enviar y recibir paquetes. Pero antes es necesario establecer la conexión de red y también gestionar los recursos que tiene asociados, entre los están, en principio,

- su estado, es decir, si la conexión se está estableciendo, si ya se puede utilizar, si se está intentando reestablecer, etcétera.
- la cola de paquetes recibidos, de la que hablamos en la última sección.
- la factoría de protocolos, que es, como vimos, una instancia de `CygnusCloudProtocolFactory`.
- una dirección IP, un puerto y el protocolo a utilizar (TCP versión 4 o TCP versión 4 sobre SSL). Esta información sólo se utiliza para establecerla.

Puesto que todos estos recursos están muy relacionados, lo más conveniente es manipularlos de forma conjunta. Esta es la finalidad de la clase `NetworkConnection` y de sus dos subclases concretas: `ClientConnection` y `ServerConnection`.

La clase abstracta `NetworkConnection` define la interfaz que utilizará la capa superior para interactuar con una conexión de red. Sus dos subclases concretas, `ClientConnection` y `ServerConnection`,

se corresponden con los dos tipos de conexiones que es posible establecer: las conexiones de tipo cliente y de tipo servidor respectivamente. Estas dos clases comparten la mayor parte del código, y sólo difieren a la hora de establecer la conexión y al actualizar su estado.

El diagrama de clases de la figura 4.13 recoge las relaciones más relevantes en las que intervienen estas tres clases. A lo largo de esta sección, las explicaremos en detalle.

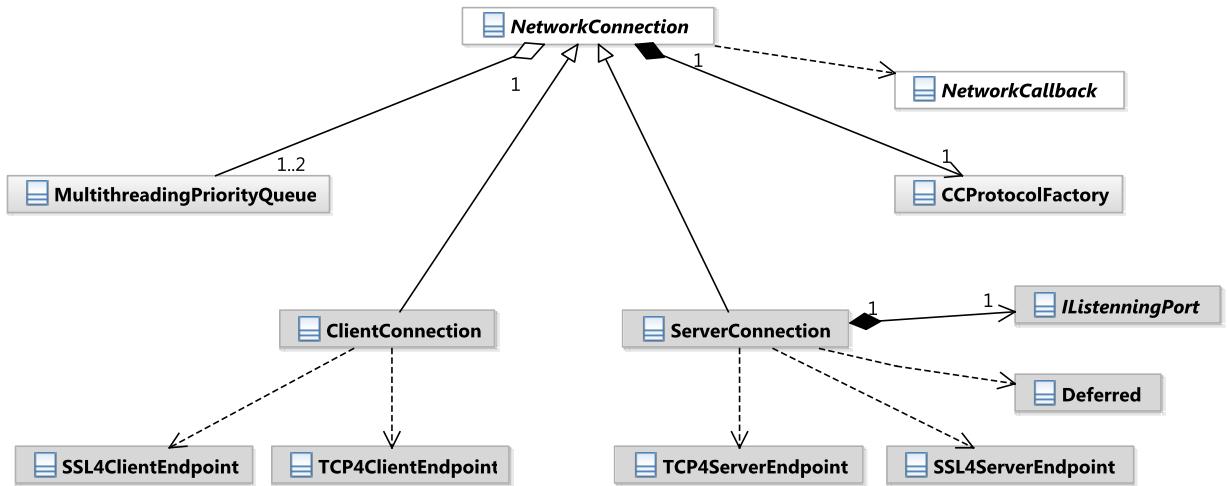


FIGURA 4.13: Relaciones más relevantes en las que intervienen las clases `NetworkConnection`, `ClientConnection` y `ServerConnection`

Establecimiento de una conexión

Para establecer una conexión de tipo cliente, basta con escoger un protocolo y el *endpoint* de tipo cliente que tiene asociado. Como nosotros sólo utilizamos los protocolos TCP versión 4 y TCP versión 4 sobre SSL, estos *endpoints* serán `TCP4ClientEndpoint` y `SSL4ClientEndpoint`.

En la sección 4.4.3.1 mostramos el intercambio de mensajes que tiene lugar durante el proceso de conexión. Por ahora, lo único que nos interesa del mismo es que, cuando el método del *endpoint* correspondiente termina, la conexión ya está lista para ser utilizada.

Por otra parte, las conexiones de tipo servidor se establecen de forma similar, pero utilizando los *endpoints* `TCP4ServerEndpoint` y `SSL4ServerEndpoint`. Pero a diferencia del caso anterior, cuando el método del *endpoint* termina la conexión aún no está lista para ser utilizada: sólo lo estará cuando se conecte el primer cliente. Esto justifica el hecho de que el estado evolucione de forma distinta en conexiones de tipo cliente y en conexiones de tipo servidor.

Finalmente, los métodos de establecimiento de una conexión siempre devuelven un objeto `Deferred`, que permite tratar los errores y, en el caso de conexiones de tipo servidor, obtener algunos de sus recursos asociados (un objeto `IListenningPort`) y cancelar el establecimiento de las mismas.

Envío y recepción de datos

La clase `NetworkConnection` envía paquetes a través de su objeto `CygnusCloudProtocolFactory`, cuyos métodos permiten transmitir el paquete a enviar. No obstante, los paquetes no se envían de forma instantánea, y sólo se pueden enviar uno por uno. Para que la capa superior pueda ignorar esta restricción, cada conexión también tiene asociada una cola de prioridad que contiene los paquetes a enviar.

Esa cola será una instancia de la clase `MultithreadingPriorityQueue`. Los paquetes a enviar siempre se insertan en esta cola, y se transmitirán uno por uno y cuando llegue su turno.

Por otra parte, también es necesario procesar los paquetes que el objeto `CygnusCloudProtocolFactory` deposita en la cola de paquetes recibidos. Puesto que el contenido de los paquetes que contienen

datos del cliente depende del dominio de la aplicación, estos deben procesarse en el código del cliente, al que se invocará utilizando la interfaz definida por la clase abstracta `NetworkCallback`. Naturalmente, estos paquetes se borran tras ser procesados.

Estado de las conexiones

Dependiendo del tipo de conexión, el estado evoluciona de forma distinta. Los diagramas de las figuras 4.14 y 4.15 muestran la evolución del estado de las conexiones de tipo servidor y de tipo cliente respectivamente.

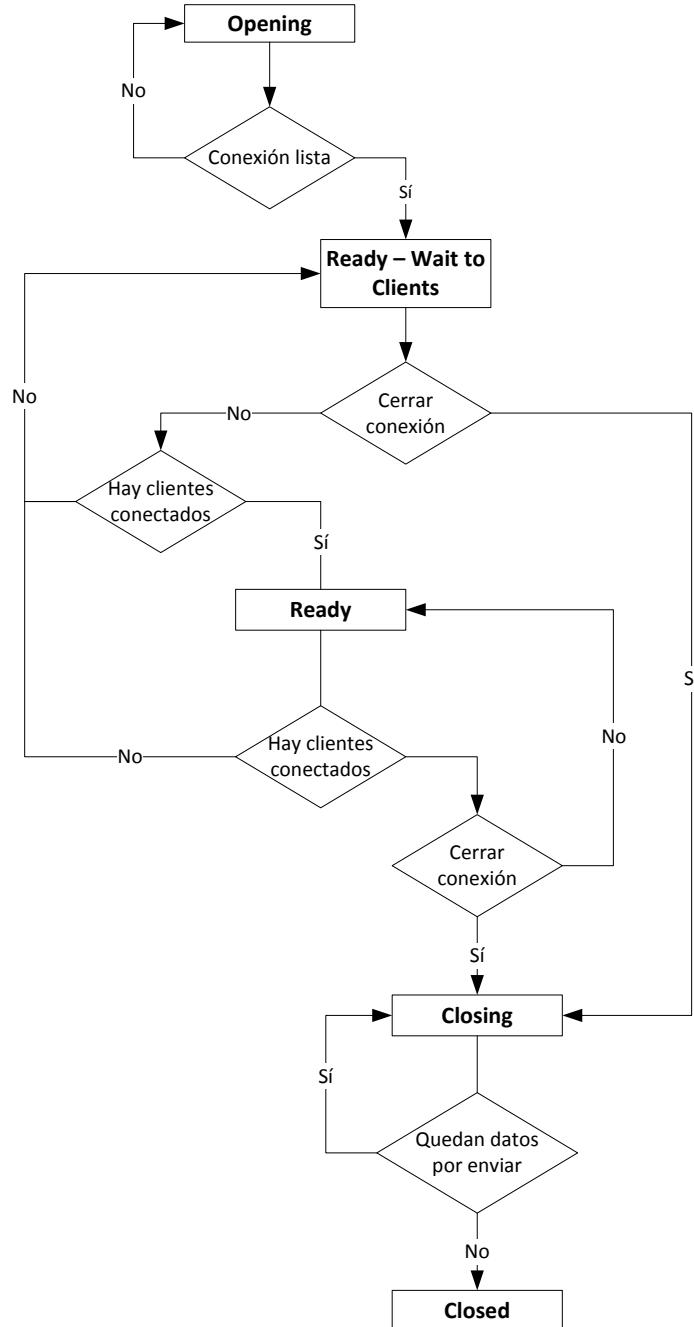


FIGURA 4.14: Evolución del estado de una conexión de tipo servidor

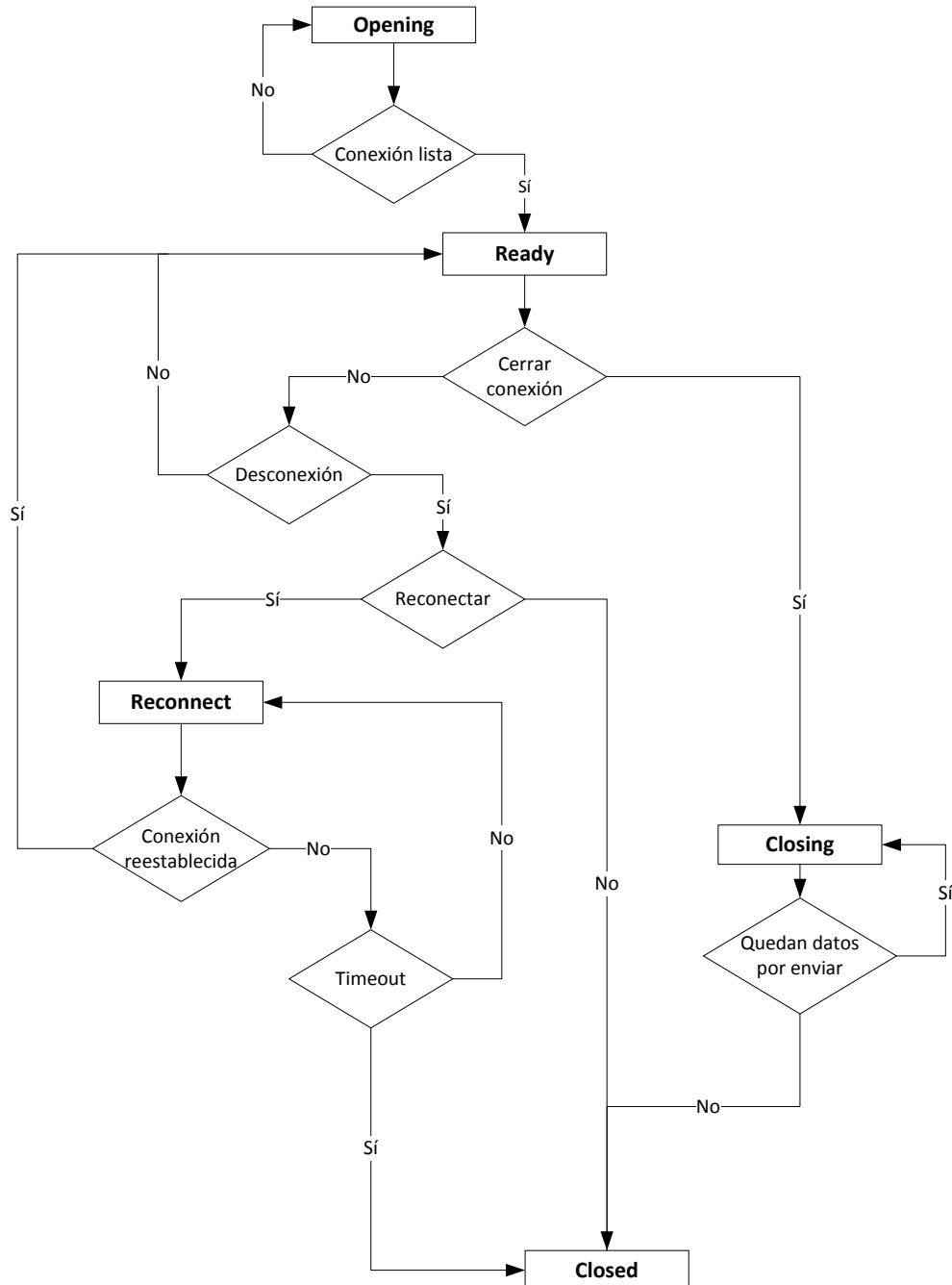


FIGURA 4.15: Evolución del estado de una conexión de tipo cliente

Ambos diagramas comparten los estados Opening, Ready, Closing y Closed. Su significado es el siguiente:

- en el estado Opening, la conexión está siendo establecida. Aún no se dispone de todos sus recursos.
- en el estado Ready la conexión está lista para recibir y transmitir datos. Por lo general, las conexiones permanecen en este estado durante la mayor parte del tiempo.
- en el estado Closing, el cliente ha solicitado el cierre de la conexión. Este estado es imprescindible para garantizar que se envían los datos pendientes. En cualquier caso, es importante

notar que se descartarán todos los paquetes recibidos y todos los nuevos paquetes que se desee enviar.

- en el estado Closed, la conexión está cerrada, y se han liberado todos sus recursos.

Las diferencias están relacionadas con el establecimiento de la conexión y las reconexiones.

Como ya hemos mencionado, una conexión de tipo servidor sólo está lista cuando uno o más clientes se conectan. En el estado Ready – Wait to clients, la conexión de tipo servidor está lista para aceptar conexiones entrantes, pero al no haber clientes conectados no será posible enviar ni recibir datos.

Por otra parte, las reconexiones sólo tienen sentido en conexiones de tipo cliente. Cuando la reconexión está habilitada, en el estado Reconnect se intentará reestablecer la conexión con el servidor utilizando retroceso exponencial binario truncado. Si no se consigue tras quince intentos (suponen unos cinco minutos aproximadamente), se asumirá que la conexión está cerrada.

4.4.3.5. Hilos de red

Utilizando lo que acabamos de contar, podemos crear y manipular conexiones de red a un nivel de abstracción razonablemente elevado. No obstante, hasta ahora hemos omitido, para facilitar la comprensión del diseño, un aspecto fundamental: el rendimiento.

En la librería de red `twisted`, todos los paquetes entrantes se procesarán en el bucle reactor. Mientras tiene lugar el procesamiento del paquete, no se enviará ni se recibirá nada. Esto supone un problema, ya que en ocasiones los subsistemas de *CygnusCloud* tienen tiempos de respuesta muy elevados. Por ejemplo, una petición de arranque de una máquina virtual tardará mucho en procesarse por toda la entrada/salida que conlleva.

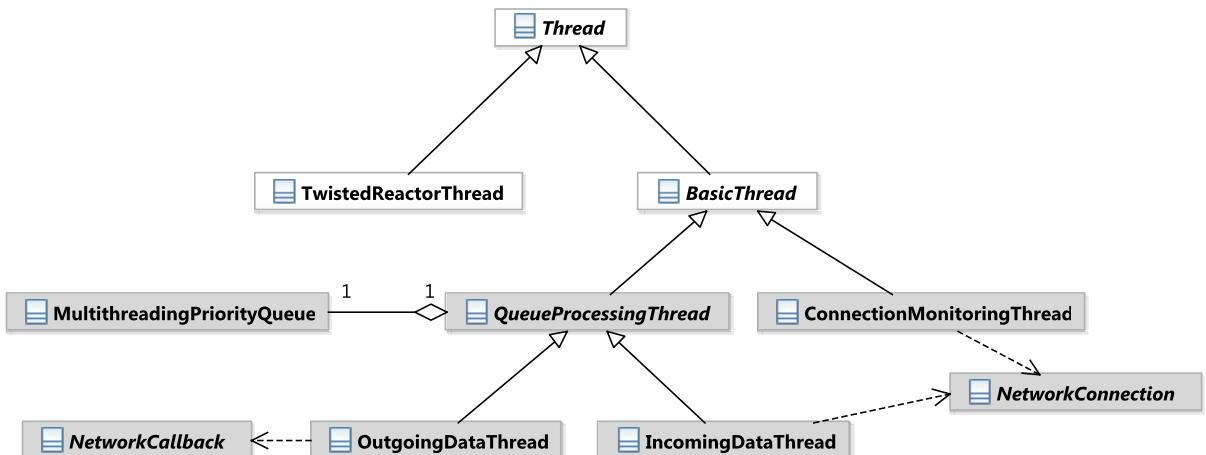


FIGURA 4.16: Jerarquía de hilos de red y sus relaciones más importantes

Así pues, resulta inadmisible que las comunicaciones se bloqueen durante el tiempo que tardan las peticiones en procesarse. La única forma de evitar esta situación es repartir la ejecución de las funciones de red entre varios hilos. Hemos realizado la siguiente descomposición:

- el bucle reactor de `twisted` se ejecutará en un hilo independiente. Esto permitirá a los usuarios utilizar el hilo principal de sus aplicaciones de la forma que estimen oportuna.
- cada conexión de red tendrá asociados un hilo de envío y un hilo de recepción de paquetes. El primero desencolará paquetes de la cola de envío y los transmitirá, y el segundo extraerá paquetes de la cola de recepción e invocará al código del cliente que los procesa.

- por motivos de eficiencia, el estado de *todas* las conexiones de red se actualizará en un hilo independiente.

El diagrama de clases de la figura 4.16 recoge las clases de la jerarquía de hilos de red y sus relaciones más relevantes.

La clase `BasicThread` añade a los hilos de Python la capacidad de detenerse cuando otros se lo solicitan, y la clase `QueueProcessingThread` añade también la capacidad de procesar elementos de una cola mediante un patrón *strategy*.

Por convención, el hilo que ejecutará el bucle reactor, `TwistedReactorThread`, hereda directamente de `Thread`, la clase base de todos los hilos en Python. Por otra parte, la clase `ConnectionMonitoringThread` se corresponde con el hilo que actualizará el estado de todas las conexiones, y las clases `IncomingDataThread` y `OutgoingDataThread` se corresponden con los hilos de recepción y envío de paquetes.

Es importante notar que:

- es el hilo de recepción y no la conexión de red el que indica al cliente la recepción de un nuevo paquete. Para ello, se sigue utilizando la interfaz que define la clase `NetworkCallback`.
- todo el procesamiento de los paquetes entrantes tendrá lugar en un hilo `IncomingDataThread` y no en el hilo del bucle reactor de *twisted*, lo que nos permite garantizar que el tiempo de respuesta de la red será adecuado independientemente del tiempo que tarde en procesarse el paquete.

Control de la concurrencia

Si recapitulamos considerando todo lo que hemos visto, para poder comunicar varias máquinas entre sí es necesario disponer de los siguientes hilos:

- dos hilos, uno de envío y otro de recepción, para cada conexión.
- un hilo para actualizar el estado de las conexiones
- un hilo para ejecutar el bucle reactor

Además, para garantizar el correcto funcionamiento del sistema es necesario utilizar secciones críticas, para lo que necesitamos mecanismos de sincronización y, por tanto, aún más recursos.

Si el número de conexiones de red de cada máquina es reducido, no habrá problemas. Pero existe un tipo de máquina en CygnusCloud que estará conectada a muchas otras: el servidor de *cluster*.

En la sección 4.3.2.3 dijimos que una de sus funciones es realizar el balanceado de carga entre varios servidores de máquinas virtuales. A medida que el número de servidores de máquinas virtuales crece, los recursos asociados a los hilos y a los mecanismos de sincronización también lo harán, reduciendo considerablemente la escalabilidad. Por ejemplo, si el servidor de *cluster* está conectado a 10 servidores de máquinas virtuales, serán necesarios

$$10 \cdot 2 + 1 + 1 = 22 \text{ hilos de red}$$

junto con los mecanismos de sincronización correspondientes. Aunque la CPU de la máquina pueda lidiar con este número de hilos, muchos estarán compitiendo por entrar en las secciones críticas, lo que reduce el rendimiento. Por tanto, debemos reducir el número de hilos de la red.

Lo primero que debemos observar es que, con independencia del número de conexiones de red que haya, todo el tráfico viajará por el mismo medio físico. Por ello, si hacemos que todas las conexiones comparten el mismo hilo de envío (y, por tanto, la misma cola de envío) el rendimiento de la red no se resentirá significativamente.

Además, esto tiene una ventaja adicional: el tráfico prioritario, sea de la conexión que sea, siempre se enviará antes que el tráfico no prioritario. Si se usan varios hilos de envío, el tráfico de las distintas conexiones se mezclará en el bucle reactor, y no será posible garantizar este comportamiento.

Con este cambio, el número de hilos de la red se reduce en casi un 50%, pero aún tenemos margen de mejora. En ocasiones, el código del cliente procesa los paquetes recibidos a través de

varias conexiones de red de forma idéntica. Esto ocurriría si, por ejemplo, el servidor de *cluster* escribiese periódicamente el estado de todos los servidores de máquinas virtuales en un *log*.

En estos casos, podemos reducir el número de hilos y evitar muchos problemas de sincronización haciendo que las conexiones correspondientes comparten el hilo de recepción de paquetes (y, por tanto, también la cola de recepción de paquetes). Esta segunda mejora permite que, en el ejemplo anterior, sólo haya

$$2 + 1 + 1 = 4 \text{ hilos de red}$$

en el servidor de *cluster*, lo que incrementa la escalabilidad.

Para facilitar el uso de la red, lo más conveniente es detectar cuándo es posible realizar esta optimización y aplicarla de forma totalmente transparente para el usuario. Aunque detectar estos casos puede parecer complicado, en realidad no lo es: cuando el cliente utiliza el mismo objeto para procesar los paquetes recibidos a través de varias conexiones (es decir, el mismo *callback*), es posible aplicar la optimización de forma segura.

En cambio, si el cliente utiliza objetos diferentes o varias instancias de una misma clase, no es posible asegurar que estamos ante uno de estos casos, por lo que la optimización no se aplica.

Finalmente, puesto que los hilos y colas de recepción de paquetes se comparten, no podemos limitarnos a destruirlos cuando se cierra la conexión. Por ello, existe un contador de referencias para estos dos objetos, que sólo cuando este llega a cero.

4.4.3.6. La clase NetworkManager

A causa de las optimizaciones que hemos aplicado para reducir el número de hilos de la red, crear conexiones de red no resulta sencillo. Asimismo, tampoco resulta interesante mostrar la implementación de la red a los clientes, ni tampoco permitir que estos puedan crear paquetes libremente.

Para resolver estos problemas, hemos creado la clase *NetworkManager*. Sus objetivos son los siguientes:

- ocultar las optimizaciones que hemos realizado
- garantizar que todos los paquetes que envían los usuarios no comprometen el correcto funcionamiento de la red
- proporcionar una fachada que permite establecer, utilizar y cerrar conexiones nivel a un elevado nivel de abstracción.

Las relaciones más relevantes de la clase *NetworkManager* aparecen en el diagrama de clases de la figura 4.17.

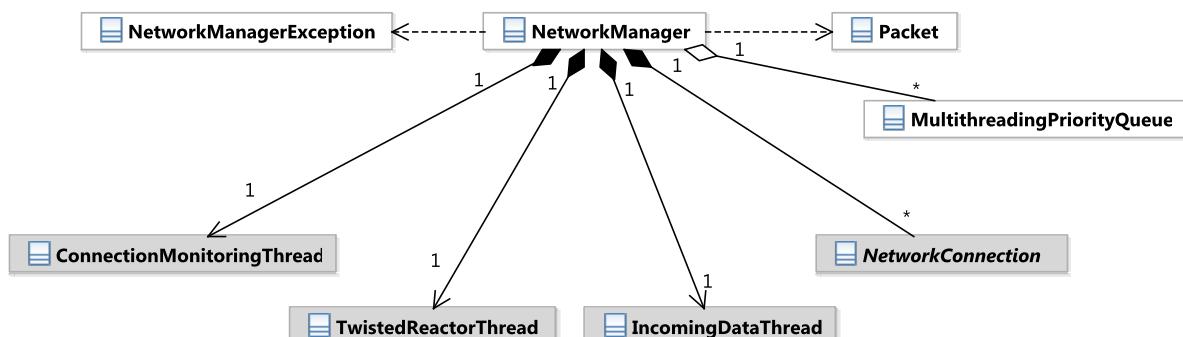


FIGURA 4.17: Relaciones más importantes de la clase *NetworkManager*

En el diagrama anterior no aparecen los hilos de recepción ni las colas de paquetes recibidos porque no están asociados a ningún objeto `NetworkManager`: estos objetos están siempre asociados a una conexión de red, de la que se tomarán en caso de que deban compartirse.

4.4.4. El paquete `ftp`

4.4.4.1. El servidor FTP `pyftpdlib`: visión general

Tal y como dijimos en la sección 4.3.14, `pyftpdlib` es un servidor FTP ligero y escrito íntegramente en *Python*. En primer lugar, mostraremos las clases que hemos utilizado para implementar nuestro servidor FTP. Estas, junto con sus relaciones, aparecen en el diagrama de clases de la figura 4.18.

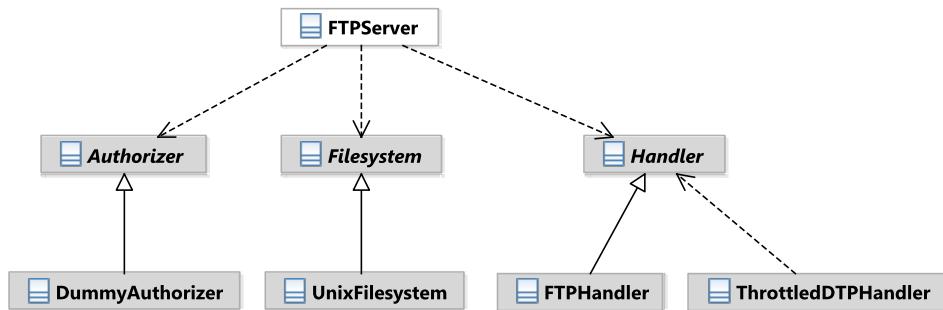


FIGURA 4.18: Principales clases del servidor FTP `pyftpdlib`

En primer lugar, la clase abstracta `Authorizer` define la interfaz que se usará para restringir el acceso al servidor FTP. Nuestro servidor FTP utiliza su subclase concreta `DummyAuthorizer`, que restringe el acceso al servidor FTP utilizando nombres de usuario y sus contraseñas (sin cifrar).

Por otra parte, la clase abstracta `Filesystem` define la interfaz que se usará para manipular los permisos del sistema de ficheros. Nuestro servidor FTP utiliza su subclase concreta `UnixFilesystem`, ya que siempre se ejecutará en Linux, que es un sistema operativo tipo UNIX.

Asimismo, la clase abstracta `Handler` define la interfaz que se usará para procesar los eventos asociados a las transferencias de ficheros. Así,

- su subclase `FTPHandler` define la interfaz que se usará para generar los eventos asociados al servidor FTP (como, por ejemplo, las conexiones y desconexiones de usuarios). Creando una subclase de `FTPHandler`, podemos determinar la forma en que se procesan estos eventos.
- su subclase `ThrottledDTPHandler` permite controlar el ancho de banda consumido por el tráfico FTP.

Finalmente, la clase `FTPServer` implementa el resto del código del servidor FTP.

Configuración de un servidor FTP

Para configurar un servidor FTP basado en `pyftpdlib`, es necesario determinar

- la forma en que se hará la autenticación en el servidor FTP,
- la forma en que se procesarán los eventos asociados a las transferencias FTP y, opcionalmente,
- la porción del ancho de banda total que podrá ser utilizado por el tráfico FTP.

Por tanto, basta con instanciar cuatro clases: la clase `FTPServer`, una subclase concreta de `Authorizer`, una subclase de `FTPHandler` y, opcionalmente, la clase `ThrottledDTPHandler`.

Eventos FTP generados por el servidor FTP `pyftplib`

Como ya hemos mencionado, el servidor FTP `pyftplib` genera eventos para que el código cliente pueda responder a conexiones de clientes, transferencias fallidas,... Los eventos FTP que se generan y su información asociada son los siguientes:

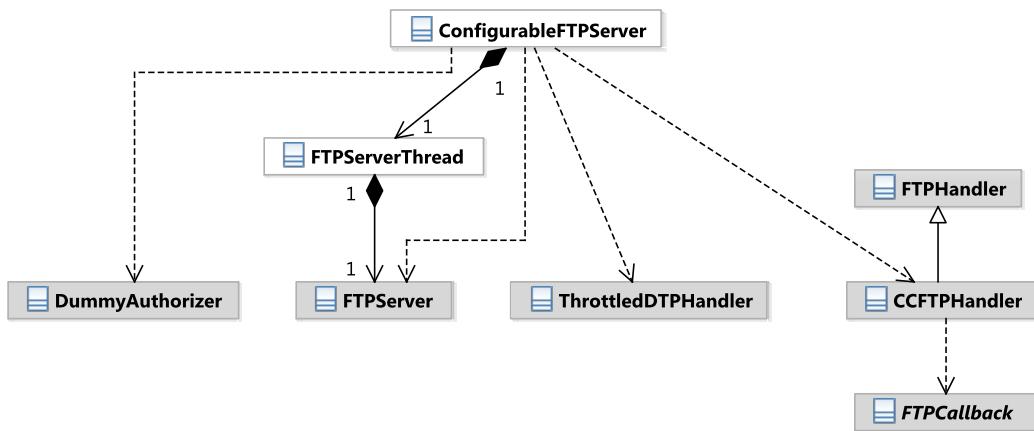
- **desconexión inesperada de un cliente.** No tiene asociado ningún tipo de información.
- **conexión de un cliente.** Tiene asociado el nombre de usuario que ha empleado el cliente para conectarse al servidor.
- **desconexión normal de un cliente.** Tiene asociado el nombre de usuario que ha empleado el cliente para conectarse al servidor.
- **fin de envío de fichero.** Tiene asociado el nombre del fichero que ha terminado de enviarse.
- **fin de recepción de fichero.** Tiene asociado el nombre del fichero que se ha recibido.
- **error en el envío de un fichero.** Tiene asociado el nombre del fichero que ha tratado de enviarse.
- **error en la recepción de un fichero.** Tiene asociado el nombre del fichero que se ha recibido parcialmente.

4.4.4.2. El servidor FTP de *CygnusCloud*

El servidor FTP de *CygnusCloud* utiliza como base el servidor FTP `pyftplib`. Sus principales características son las siguientes:

- permite realizar la gestión de usuarios y contraseñas en tiempo de ejecución. Estos datos siempre se transmitirán sin cifrar.
- permite restringir el número total de conexiones al servidor FTP, y también el número máximo de conexiones por cada dirección IP.
- permite restringir el tráfico FTP a cierta interfaz de red.
- permite configurar la fracción del ancho de banda que usará el tráfico FTP. Para ello, detecta el ancho de banda del enlace por el que viajará el tráfico FTP.
- hace posible que los eventos FTP se atiendan en el código cliente cuando sea oportuno.
- utiliza un hilo dedicado para el servidor FTP.

El diagrama de clases de la figura 4.19 muestra las clases del servidor FTP y sus relaciones con las clases del servidor FTP `pyftplib`. Por claridad, en él hemos omitido las relaciones existentes entre las clases de `pyftplib`. Asimismo, hemos abreviado *CygnusCloud* como CC.


 FIGURA 4.19: El servidor FTP de *CygnusCloud*: diagrama de clases

`ConfigurableFTPServer` es la clase principal del servidor FTP. Su misión es ocultar al código cliente el proceso de configuración del servidor FTP `pyftpdlib`, haciendo posible que este pueda procesar los eventos generados por el servidor FTP.

El procesamiento de dichos eventos se realiza a través de la interfaz definida por la clase abstracta `FTPCallback`. La clase `CygnusCloudFTPHandler`, que hereda de `FTPHandler`, se encargará de invocar a los métodos de un objeto `FTPCallback` a medida que se generen los eventos FTP.

Por otra parte, la clase `ConfigurableFTPServer` utiliza un objeto `DummyAuthorizer` y un objeto `ThrottledDTPHandler` durante el proceso de configuración del servidor FTP `pyftpdlib`, lo que significa que

- la autenticación usará nombres de usuario y contraseñas en texto plano, y
- el ancho de banda utilizado por el tráfico FTP podrá restringirse.

Finalmente, el código del servidor FTP se ejecuta en un hilo independiente, que se corresponde con la clase de hilo `FTPServerThread`. Así, el hilo en el que se haya configurado y arrancado el servidor FTP podrá utilizarse con otros fines.

El código de la clase `FTPServerThread` está tomado de las pruebas unitarias de `pyftpdlib`: por ahora, no se encuentra integrado con el resto de código de `pyftpdlib`.

4.4.4.3. El cliente FTP de *CygnusCloud*

El cliente FTP de *CygnusCloud* no es más que un envoltorio del cliente FTP incluido en la librería estándar de Python, `FTP`. Esto permite que los clientes FTP puedan intercambiar información con un servidor FTP a un mayor nivel de abstracción.

La clase asociada al cliente FTP de *CygnusCloud* es `FTPClient`. Permite, entre otras cosas,

- crear y destruir conexiones con un servidor FTP,
- subir un fichero a un determinado directorio del servidor FTP, y
- descargar un fichero ubicado en cierto directorio del servidor FTP.

Es importante notar que la clase `FTP` de la librería estándar de *Python* *no* soporta FTP seguro. De todos modos, resulta muy sencillo utilizar otro cliente FTP que sí lo soporte: basta con modificar la implementación de los métodos de la clase `FTPClient`.

4.4.5. El paquete `imageRepository`

Tal y como dijimos en la sección 4.3.2.2, el repositorio de imágenes se ocupa de almacenar todas las imágenes de disco asociadas a todas las máquinas virtuales que pueden utilizarse en un *cluster*, es decir,

- las imágenes de disco asociadas a las máquinas virtuales que utilizan los alumnos, y
- las imágenes de disco base, que se usan para crear nuevas máquinas virtuales.

En esta sección, mostraremos detalladamente el diseño del repositorio de imágenes. Para ello, empezaremos discutiendo por qué un servidor FTP no puede utilizarse directamente como repositorio de imágenes. Acto seguido, mostraremos qué funciones realiza el repositorio de imágenes, para después mostrar en qué nos hemos basado para implementarlas.

Posteriormente, mostraremos las funciones que añaden las clases del paquete `imageRepository` al servidor FTP, para continuar mostrando las relaciones entre estas clases y las interacciones entre el repositorio de imágenes y una máquina remota. A continuación, mostraremos la organización de los ficheros en el servidor FTP.

Finalmente, mostraremos los formatos de paquete que emplea el repositorio de imágenes para interactuar con el exterior y el esquema de su base de datos.

4.4.5.1. Adición de una capa adicional al servidor FTP

Puesto que el repositorio de imágenes almacena todas las imágenes de disco que pueden utilizarse en el *cluster*, lo más razonable es que el repositorio de imágenes se construya sobre un servidor FTP, y que todos los servidores de máquinas virtuales se conecten a él como clientes FTP.

Ahora bien, si usamos directamente un servidor FTP para implementar el repositorio de imágenes, tendremos que enfrentarnos con serios problemas. Para ilustrarlos, daremos varios ejemplos.

En primer lugar supongamos que, en un momento dado,

- el *cluster* contiene n servidores de máquinas virtuales, que
- un administrador decide desplegar la misma imagen de disco en todos ellos, y que
- ningún servidor de máquinas virtuales tiene almacenada esa imagen de disco.

Cuando se realiza el despliegue de la imagen, el repositorio de imágenes tendrá que lidiar con n conexiones simultáneas, y debe transferir un fichero de gran tamaño por cada una de ellas. A medida que aumente el número de servidores de máquinas virtuales del *cluster*, tarde o temprano se acabarán generando errores por *timeout*.

Así, a medida que crece el número de servidores de máquinas virtuales del *cluster*,

- la infraestructura se hace más difícil de implementar o administrar, ya que hay que limitar el número de transferencias simultáneas entre el repositorio de imágenes y los servidores de máquinas virtuales, y
- se está desperdiando cada vez más ancho de banda: el que consumen las transferencias de ficheros que fallan.

Para que el sistema sea usable, resulta imprescindible limitar el número máximo de servidores de máquinas virtuales del *cluster*. Así, el número de servidores de máquinas virtuales del *cluster* pasará a estar limitado por el ancho de banda disponible para el tráfico que intercambian estos y el repositorio de imágenes, y no por la capacidad del servidor de *cluster* para atender a todos ellos. Esto compromete la escalabilidad del sistema.

Por otra parte, el uso directo de un servidor FTP en el repositorio de imágenes tiene un inconveniente aún mayor: la aparición de conflictos en la edición de imágenes de disco existentes. Por ejemplo, supongamos que

4.4.5.2. Funciones soportadas por el repositorio de imágenes

- dos profesores A y B imparten la misma asignatura, que tiene asociada la máquina virtual VM_1 , que
- tras utilizar la máquina virtual VM_1 por separado, A y B descubren que varias herramientas están mal configuradas, y que
- A y B deciden resolver por su cuenta el problema.

Para resolver este problema, tanto A como B tendrán que editar las imágenes de disco asociadas a la máquina virtual VM_1 . Pero a la hora de transferirlas de nuevo al repositorio, las modificaciones que ha hecho A acabarán sobreescribiéndose con las que ha hecho B o viceversa. Por tanto, uno de los profesores ha estado perdiendo el tiempo: los cambios que ha introducido se perderán.

En definitiva, el uso directo de un servidor FTP en el repositorio de imágenes presenta serios inconvenientes. Por ello, el repositorio de imágenes no sólo utiliza un servidor FTP, y añade una capa adicional para resolverlos.

4.4.5.2. Funciones soportadas por el repositorio de imágenes

El repositorio de imágenes está construido sobre un servidor FTP, al que añade las siguientes funciones:

- gestión de los identificadores de las imágenes de disco. Todas las imágenes de disco que pueden utilizarse en el *cluster* tienen asociado un identificador único, y el repositorio de imágenes se ocupa de la gestión de estos identificadores.
- borrado de una imagen del servidor que actúa como repositorio de imágenes.
- transferencia de un fichero *en exclusividad* a una máquina remota. Tras realizar una de estas transferencias, el fichero no podrá transferirse a ninguna otra máquina hasta que el repositorio reciba la copia modificada. Esto nos permite evitar la aparición de conflictos como el del último ejemplo de la sección anterior.
- cálculo del espacio en disco total y disponible.
- control del número de transferencias simultáneas, tratando con equidad a todas las máquinas remotas que deseen intercambiar un fichero con el repositorio de imágenes.

Naturalmente, el repositorio de imágenes también soporta la transferencia y la recepción de ficheros. Para realizar estas operaciones, delega en el servidor FTP.

4.4.5.3. La conexión de control

El enlace que conecta el repositorio de imágenes a la red troncal de la UCM será *full duplex*. Para aprovechar su ancho de banda al máximo, debemos utilizar todo el ancho de banda de bajada y todo el ancho de banda de subida que podamos, por lo que debemos intentar, en la medida de lo posible, realizar varias transferencias de subida y de descarga de ficheros en paralelo.

Por otra parte, el ancho de banda del enlace está limitado. Si permitimos que los servidores se conecten sin control al servidor FTP, podrán generarse errores por *timeout* que supondrán, tal y como dijimos en la sección 4.4.5.1, un desperdicio del ancho de banda del enlace.

Así pues, debemos restringir el número máximo de transferencias simultáneas entre el servidor FTP y las máquinas remotas, y también debemos realizar las transferencias de subida y de descarga en paralelo cuando sea posible. Para ello, las máquinas remotas no se conectarán directamente al servidor FTP, y dialogarán previamente con el repositorio de imágenes a través de una conexión de control adicional.

4.4.5.4. Los *slots* de transferencia

Como dijimos en la sección 4.4.5.3, para no desperdiciar el ancho de banda del enlace resulta imprescindible restringir el número de transferencias simultáneas entre el repositorio de imágenes y las máquinas remotas. Para ello, hemos utilizado un mecanismo basado en *slots*.

Cada *slot* estará asociado a una transferencia activa. Por ejemplo, si en un instante dado existen n transferencias activas en total, se estarán utilizando n *slots*. El número máximo de *slots* de transferencia se configura durante el arranque del repositorio de imágenes.

Así, cuando se recibe una nueva petición de transferencia,

- si hay un *slot* libre, se marcará como ocupado y se iniciará la transferencia, y
- si no hay ningún *slot* libre, la petición correspondiente se encolará.

Tras finalizar una transferencia, se desencolará e iniciará la siguiente petición. Esto nos permite limitar el número máximo de transferencias activas entre el repositorio de imágenes y el resto de máquinas del *cluster*.

4.4.5.5. Clases principales

El diagrama de clases de la figura 4.20 muestra las principales clases del repositorio de imágenes y sus relaciones con otras clases. Por claridad,

- `ImageRepository` aparece abreviado como `IR`,
- las relaciones de la clase `FTPServerCallback` aparecen en azul,
- las relaciones de la clase `CommandsCallback` aparecen en rojo,
- hemos omitido las relaciones existentes entre las clases que no forman parte del paquete `imageRepository`, y
- no hemos indicado los paquetes a los que pertenecen las clases que no están definidas en el paquete `imageRepository`.

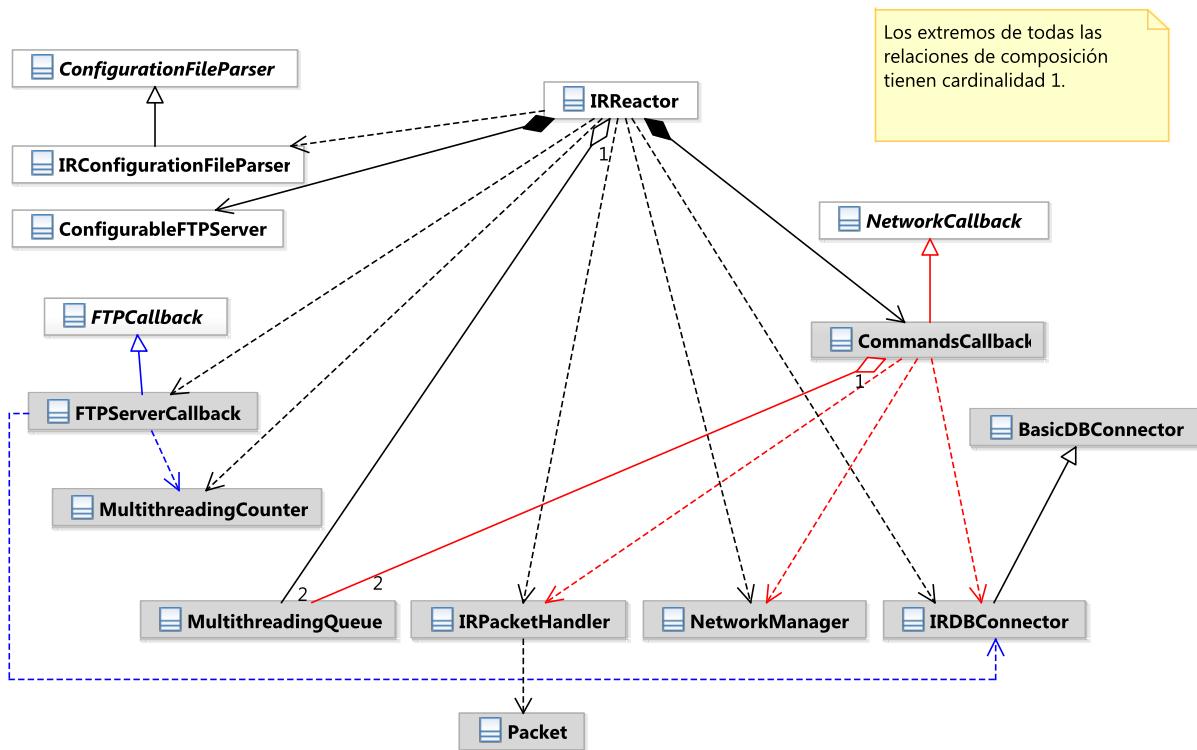


FIGURA 4.20: Principales clases del repositorio de imágenes y sus relaciones

Las principales clases que aparecen en el diagrama y sus responsabilidades son las siguientes:

- **ImageRepositoryReactor** es la clase principal del repositorio de imágenes. Además de ocuparse de los procesos de arranque y apagado del demonio del servidor de imágenes, también se encarga de asignación de *slots* a las transferencias encoladas.
- la clase **CommandsCallback** procesará todos los paquetes recibidos a través de la conexión de control. Análogamente, la clase **FTPServerCallback** procesará todos los eventos generados por el servidor FTP.
- la clase **ImageRepositoryPacketHandler** dispone de métodos para crear y leer los paquetes que se intercambian a través de la conexión de control.
- la clase **ImageRepositoryDBConnector** dispone de métodos para manipular la base de datos del repositorio de imágenes.
- la clase **ImageRepositoryConfigurationFileParser** procesa el fichero de configuración del demonio del repositorio de imágenes.

Por otra parte,

- las clases **ConfigurationFileParser**, **BasicDBConnector**, **MultithreadingQueue** y **MultithreadingCounter** forman parte del paquete **ccutils**,
- las clases **Packet**, **NetworkManager** y **NetworkCallback** forman parte del paquete **network**,
- las clases **FTPCallback** y **ConfigurableFTPServer** forman parte del paquete **FTP**.

Más adelante, justificaremos el uso de estas clases, así como la cardinalidad de las relaciones de agregación y composición.

4.4.5.6. Arranque del repositorio de imágenes

El proceso de arranque del demonio del repositorio de imágenes aparece en el diagrama de secuencia de la figura 4.21. En él, `ImageRepository` aparece abreviado como `IR`.

Como puede observarse en el diagrama, desde el punto de entrada del demonio del repositorio de imágenes se invoca a varios métodos de la clase `ImageRepositoryReactor`. Los pasos que se siguen son los siguientes:

1. se crea la base de datos si no existe, y se registra un usuario que disponga de todos los permisos sobre ella.
2. se parsea el fichero de configuración. El reactor del repositorio de imágenes utilizará esa información durante el proceso de inicialización.
3. se crean las dos colas de transferencias y el contador de *slots*.

Aunque sería posible utilizar una única cola de transferencias y extraer de ella peticiones de subida y de bajada, por eficiencia hemos utilizado dos colas: una contiene las peticiones de subida, y otra las peticiones de bajada. Esto explica la cardinalidad de las relaciones de agregación del diagrama de clases de la figura 4.20.

Por otra parte, el contador de *slots* indica cuántos *slots* de transferencia están ocupados en un momento dado.

4. se establece la conexión con la base de datos del repositorio de imágenes.
5. se inicializan el gestor de red y el gestor de paquetes (una instancia de `ImageRepositoryPacketHandler`). Como dijimos en la sección 4.4.5.5, esta clase manipula los formatos de paquete asociados al repositorio de imágenes. Así, si es necesario modificar alguno de ellos basta con modificar el código de esta clase.
6. se configura y utiliza la conexión de control. Para ello, es necesario disponer de un objeto *callback* que procese los paquetes recibidos por la conexión de control. En el caso del repositorio de imágenes, se utiliza como objeto *callback* una instancia de la clase `CommandsCallback`.
7. se configura e inicia el servidor FTP. Es importante notar que la contraseña se genera aleatoriamente en cada arranque.
8. se empieza a asignar *slots* a las peticiones de transferencia encoladas.

Tras esto, el repositorio de imágenes está totalmente operativo.

4.4.5.7. Interacciones del repositorio de imágenes

En esta sección, mostraremos cómo interactúa el repositorio de imágenes con una máquina remota. Todas estas interacciones tienen lugar intercambiando paquetes a través de la conexión de control. Además, las que requieran la transferencia de un fichero también involucrarán al servidor FTP.

Por claridad, en esta sección hemos preferido centrarnos en la secuencia de acciones que se realizan en cada caso, y no en los formatos de los tipos de paquete que se utilizan. Más adelante, en la sección 4.4.5.8, los describiremos con todo lujo de detalles.

Registro de un identificador de imagen

Como dijimos en la sección 4.4.5.2, el repositorio se ocupa de la gestión de los identificadores de las imágenes de disco de la infraestructura. Por ello, cuando se cree una nueva imagen, será necesario solicitar el identificador de la misma al repositorio.

El diagrama de secuencia de la figura 4.22 muestra la interacción que tiene lugar para reservar dicho identificador. Nuevamente, hemos abreviado `ImageRepository` como `IR`.

4.4.5.7. Interacciones del repositorio de imágenes

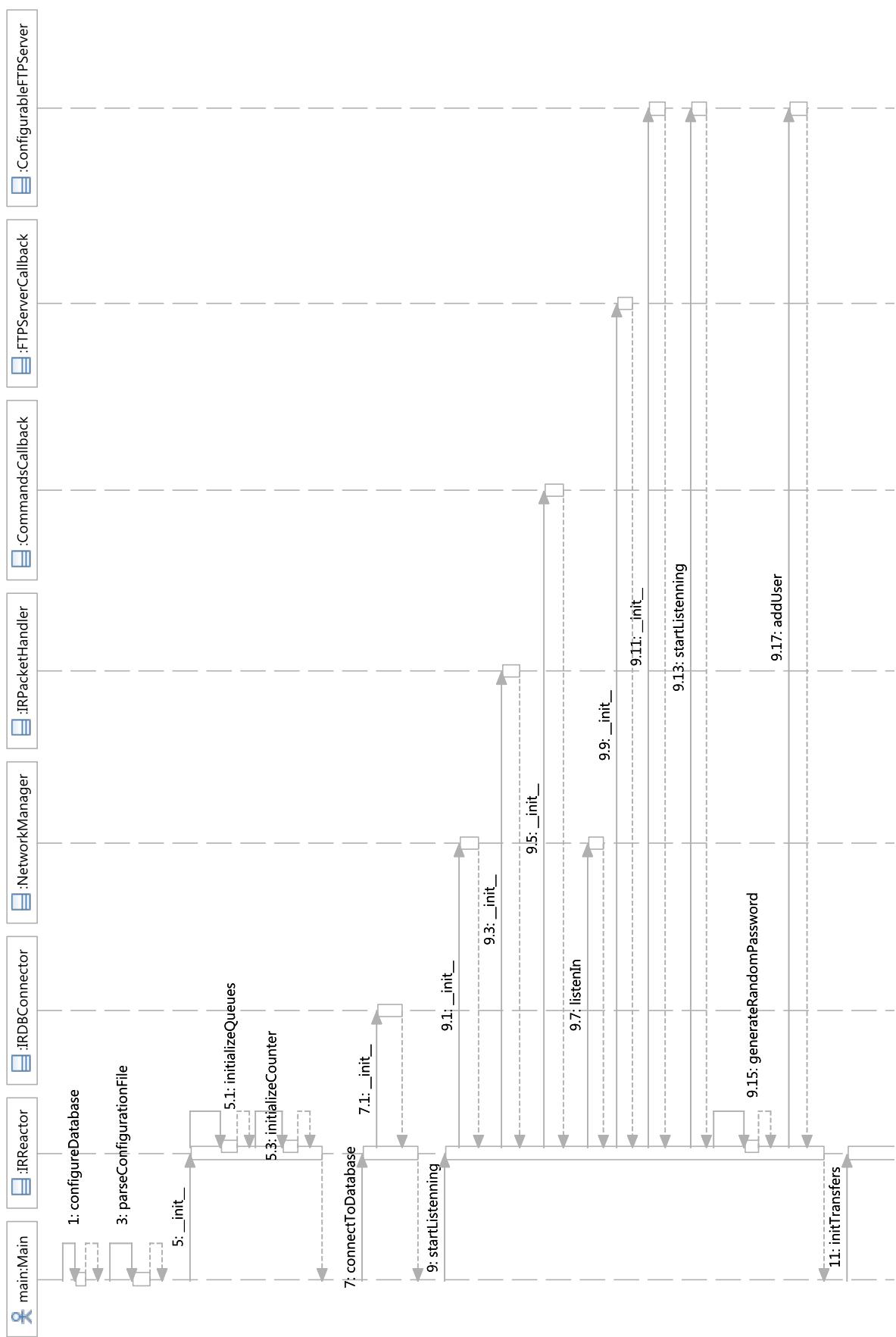


FIGURA 4.21: Arranque del repositorio de imágenes: diagrama de secuencia

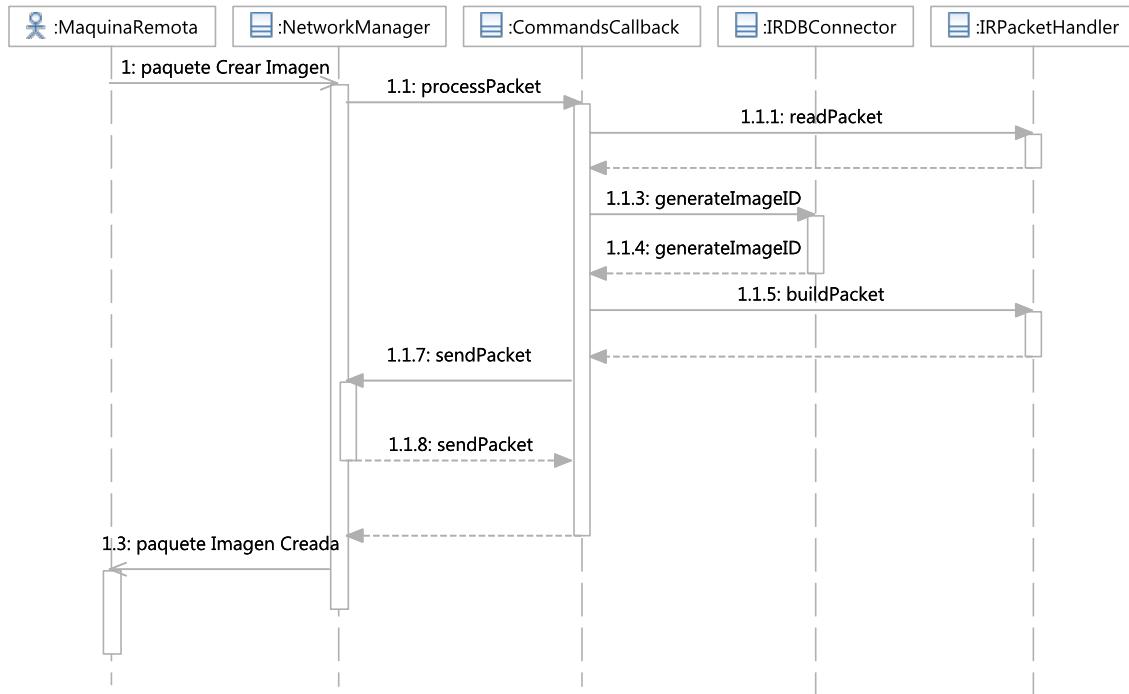


FIGURA 4.22: Registro de un identificador de imagen: diagrama de secuencia

Como puede observarse en el diagrama, se siguen los siguientes pasos:

1. a través de la conexión de control, la máquina remota envía un paquete del tipo Crear Imagen al repositorio de imágenes.
2. la red invoca al método de procesamiento de paquetes del objeto CommandsCallback del repositorio de imágenes.
3. el objeto CommandsCallback lee el contenido del paquete a través de un objeto ImageRepositoryPacketHandler. Acto seguido, solicita a un objeto ImageRepositoryDBConnector la generación de un nuevo identificador de imagen. El objeto ImageRepositoryDBConnector generará el identificador único utilizando el contenido de las tablas de la base de datos del repositorio de imágenes.
4. el objeto CommandsCallback construye el paquete con la respuesta, que es del tipo Imagen Creada. Este paquete se crea a través de un objeto ImageRepositoryPacketHandler, y contiene el identificador de imagen que se ha generado.
5. el objeto CommandsHandler envía ese paquete a la máquina remota. Para ello, utiliza un objeto NetworkManager.

En todas las interacciones, el repositorio de imágenes utiliza del modo de envío *unicast* de la conexión de control para enviar las respuestas. Por ello, sólo el cliente que envió la petición correspondiente (y no todos) recibirá la respuesta. Esto nos permite ahorrar mucho ancho de banda.

Descarga de imágenes de disco

En la sección 4.3.15 dijimos que, para ahorrar ancho de banda y espacio en disco, las máquinas de la infraestructura no intercambian directamente ficheros de imagen, sino ficheros zip que contienen ficheros de imagen.

Por ello, el repositorio de imágenes sólo intercambia ficheros zip con las máquinas remotas. En esta sección, mostraremos todas las posibles interacciones que tienen lugar cuando una máquina remota intenta descargar uno de estos ficheros comprimidos.

4.4.5.7. Interacciones del repositorio de imágenes

El diagrama de secuencia de la figura 4.23 muestra la interacción básica de inicio de la transferencia, en la que no aparecen errores.

Los pasos que se siguen son los siguientes:

1. el objeto `CommandsCallback` recibe el paquete del tipo Petición FTP `RETR`, y lee su contenido mediante un objeto `ImageRepositoryPacketHandler`. Estos paquetes contienen el identificador único del fichero comprimido.
2. el objeto `CommandsCallback` comprueba que el fichero existe y que no ha sido concedido en exclusividad a otra máquina. Como no se producen errores, confirma a la máquina remota la recepción de su petición mediante un paquete del tipo Petición FTP `RETR Recibida`.
3. cuando un *slot* de transferencia queda libre y la petición llega a la cabecera de la cola, desde el método `initTransfers()`
 - a) se comprueba nuevamente que el fichero puede transferirse. En la interacción del diagrama de secuencia de la figura 4.23 no se detecta ningún error.
 - b) se reserva un *slot* de transferencia
 - c) se envía un paquete del tipo Inicio transferencia FTP `RETR` a la máquina remota. Este contiene todo la la información que la máquina remota necesita para descargar el fichero comprimido.

Para aprovechar al máximo el ancho de banda del enlace, las transferencias se desencolan procurando que, cuando sea posible, se realicen transferencias de subida y descarga en paralelo. Así, las transferencias no siempre se inician en estricto orden de recepción. Por claridad, hemos preferido no reflejar esto en el diagrama.

Por otra parte, no se podrá realizar la transferencia cuando

- el fichero a descargar no existe, o
- este ha sido concedido en exclusividad a otra máquina.

Estos errores pueden detectarse tanto al procesar el paquete Petición FTP `RETR` como al habilitar la transferencia. Los diagramas de secuencia de las figuras 4.24 y 4.25 muestran cómo interactúan el repositorio y la máquina remota al detectar estos errores.

Como puede observarse en los diagramas de secuencia de las figuras 4.24 y 4.25, al detectar un error el repositorio de imágenes envía un paquete de error a la máquina remota, y esta no realiza la transferencia. Todos los paquetes de error contienen, además del identificador de las imágenes de disco del fichero comprimido, un código que indica la causa del error.

Finalmente, cuando no se producen errores se realizará una transferencia FTP `RETR` entre la máquina remota y el servidor FTP. El diagrama de secuencia de la figura 4.26 muestra la interacción que tiene lugar en este caso.

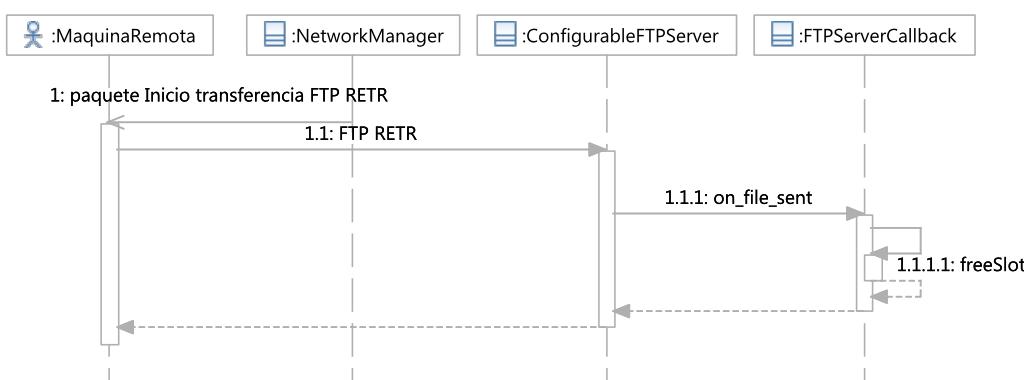


FIGURA 4.26: Transferencia de una imagen a una máquina remota

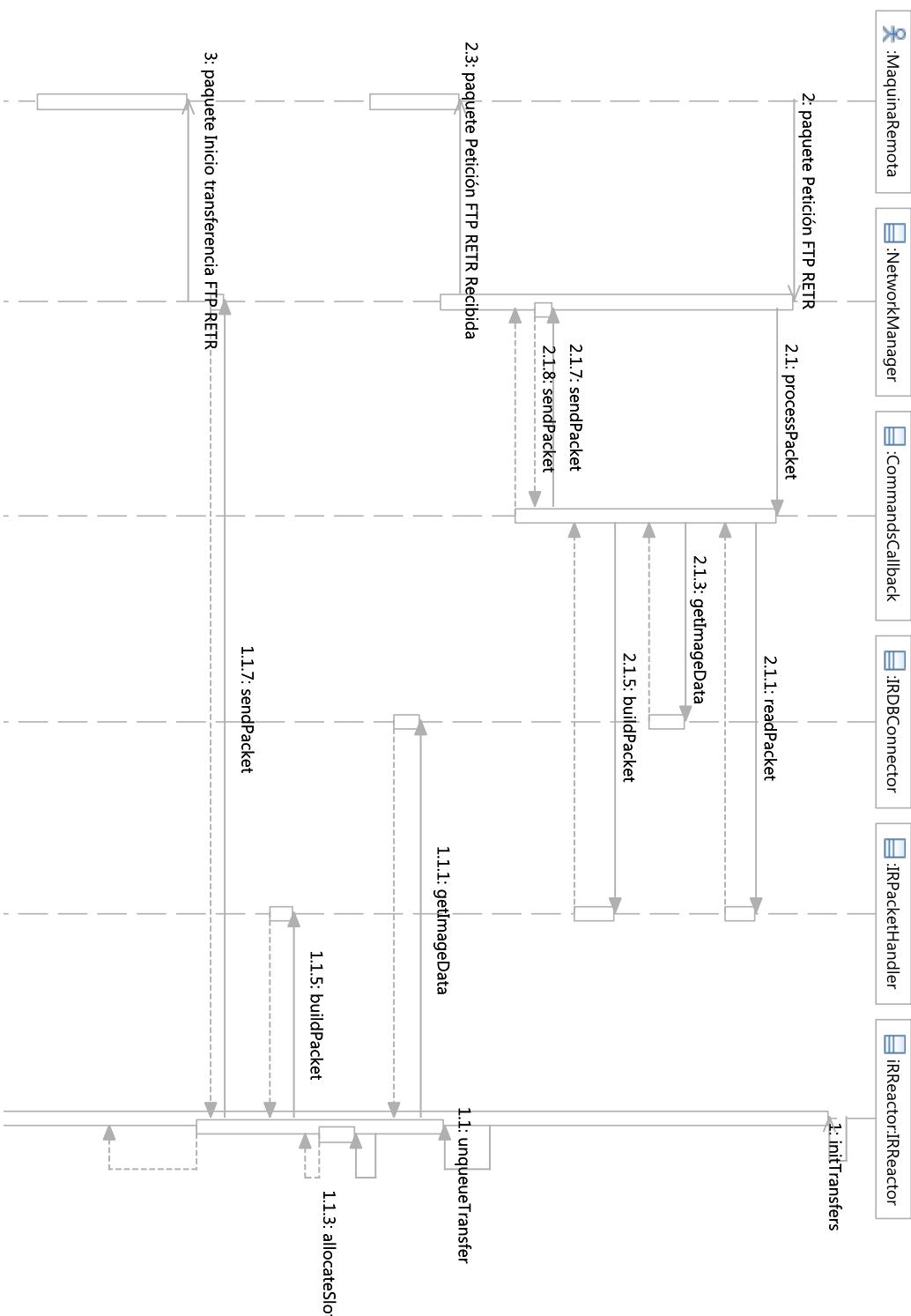


FIGURA 4.23: Inicio de la transferencia de una imagen a una máquina remota: secuencia básica

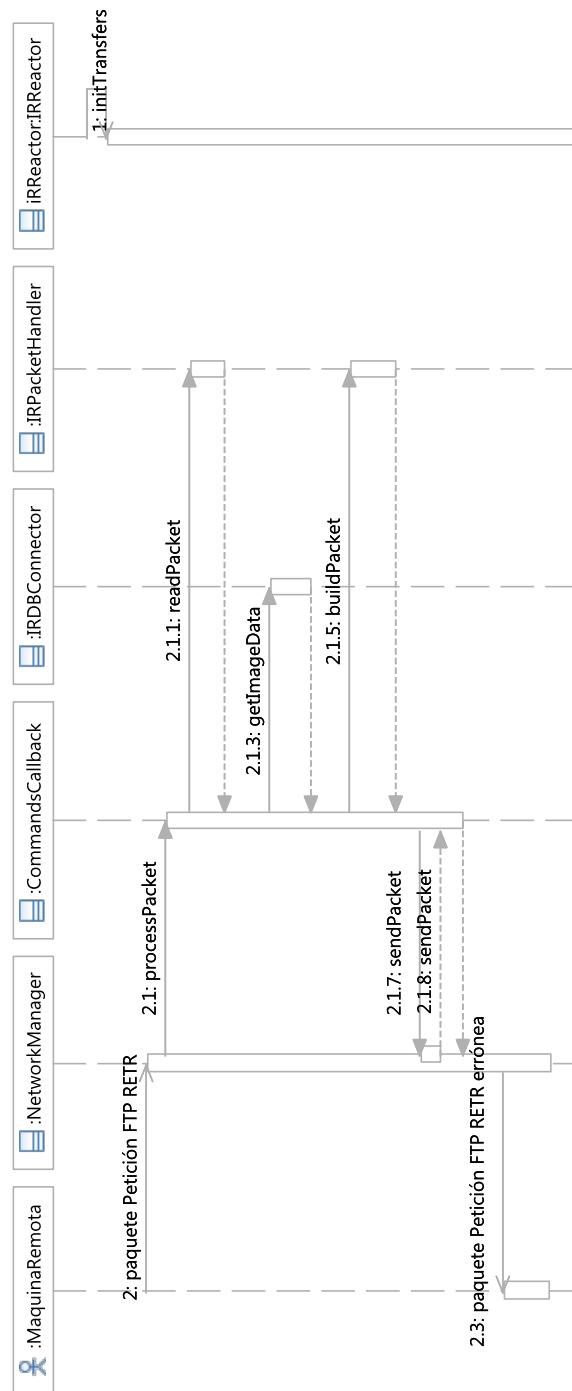


FIGURA 4.24: Detección de errores al recibir la petición de transferencia

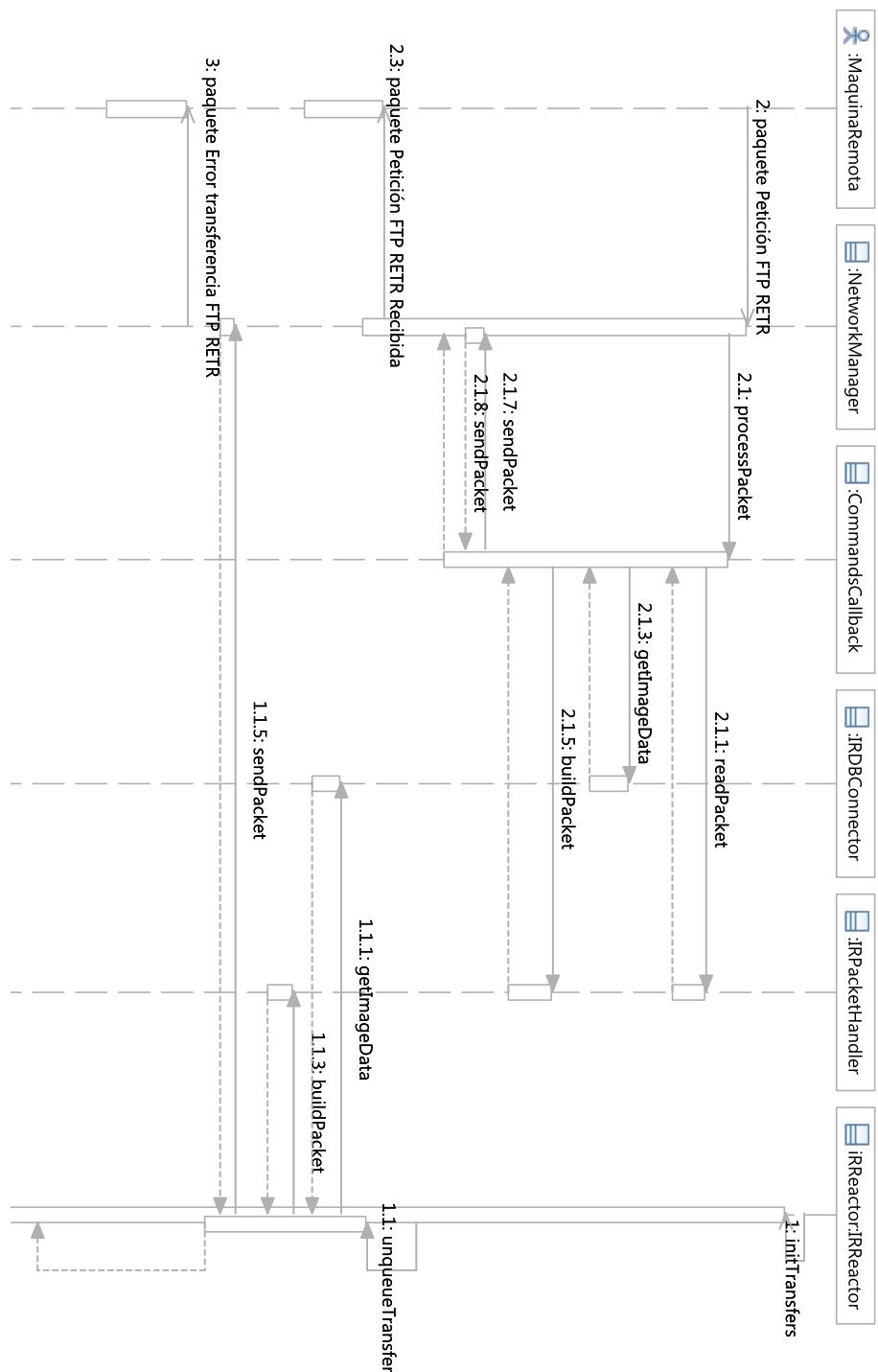


FIGURA 4.25: Detección de errores al habilitar la transferencia

Tras finalizar la transferencia, se invocará al método `on_file_sent()` del objeto `FTPServerCallback`, en el que se liberará el *slot* asignado a la transferencia. Cuando se libera el *slot*, se iniciará la siguiente transferencia encolada (si la hay).

Uso de imágenes de disco en exclusividad

Tal y como dijimos en la sección 4.4.5.2, el repositorio de imágenes puede transferir un fichero comprimido a una máquina remota en exclusividad. Esto significa que, mientras que esa máquina remota no lo libere, bien subiendo una versión modificada del fichero o bien pidiendo al repositorio que lo libere, ninguna otra máquina remota podrá descargarlo.

En esta sección, mostraremos cómo una máquina remota descarga un fichero en exclusividad, y también cómo lo puede liberar sin volver a subirlo. En la siguiente sección, mostraremos cómo puede liberarlo subiendo una copia modificada del mismo.

El diagrama de secuencia de la figura 4.27 muestra cómo el repositorio de imágenes recibe una petición de descarga en exclusividad.

Como se puede observar en el diagrama de secuencia, la petición se procesa casi igual que una petición de descarga “normal”. No obstante, a diferencia del caso anterior,

- el flag *modify*, que forma parte todos los paquetes Petición FTP `RETR`, está activado, y
- al procesar la petición, el repositorio de imágenes registra en la base de datos que la imagen se va a descargar en exclusividad.

La descarga de la imagen y el tratamiento de los errores son idénticos a los de una transferencia de descarga “normal”.

Por otra parte, el diagrama de secuencia de la figura 4.28 muestra cómo una máquina remota libera un fichero que tiene en exclusividad.

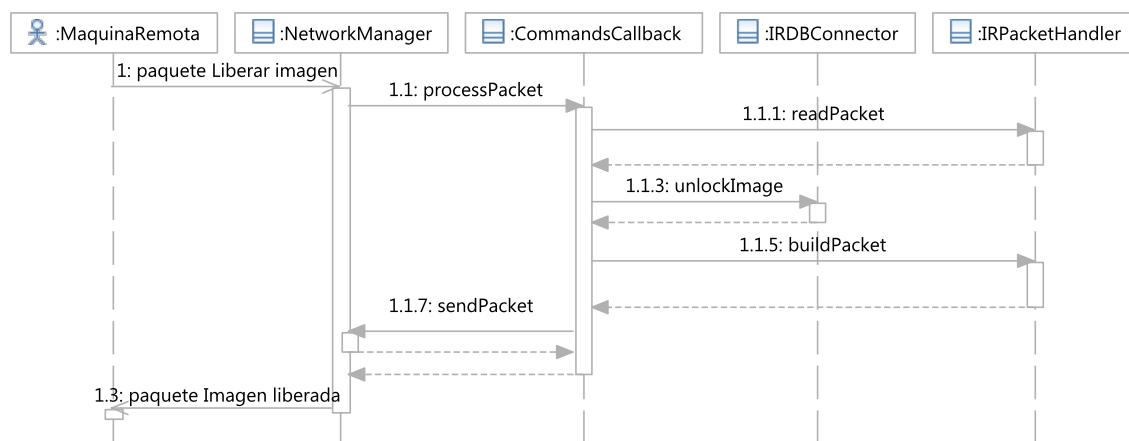


FIGURA 4.28: Liberación de una imagen de disco

En este caso,

1. la máquina remota envía un paquete del tipo *Liberar imagen* al repositorio de imágenes. Este contiene el identificador del fichero comprimido.
2. al procesar el paquete, desde el objeto `CommandsCallback` se modifica el estado de la imagen en la base de datos del repositorio de imágenes.
3. finalmente, el objeto `CommandsCallback` envía un paquete del tipo *Imagen liberada* a la máquina remota.

Es importante notar que, cuando se intenta liberar un fichero que no está asignado en exclusividad a una máquina, la petición no tendrá efecto. Por ello, en este caso no se generarán errores.

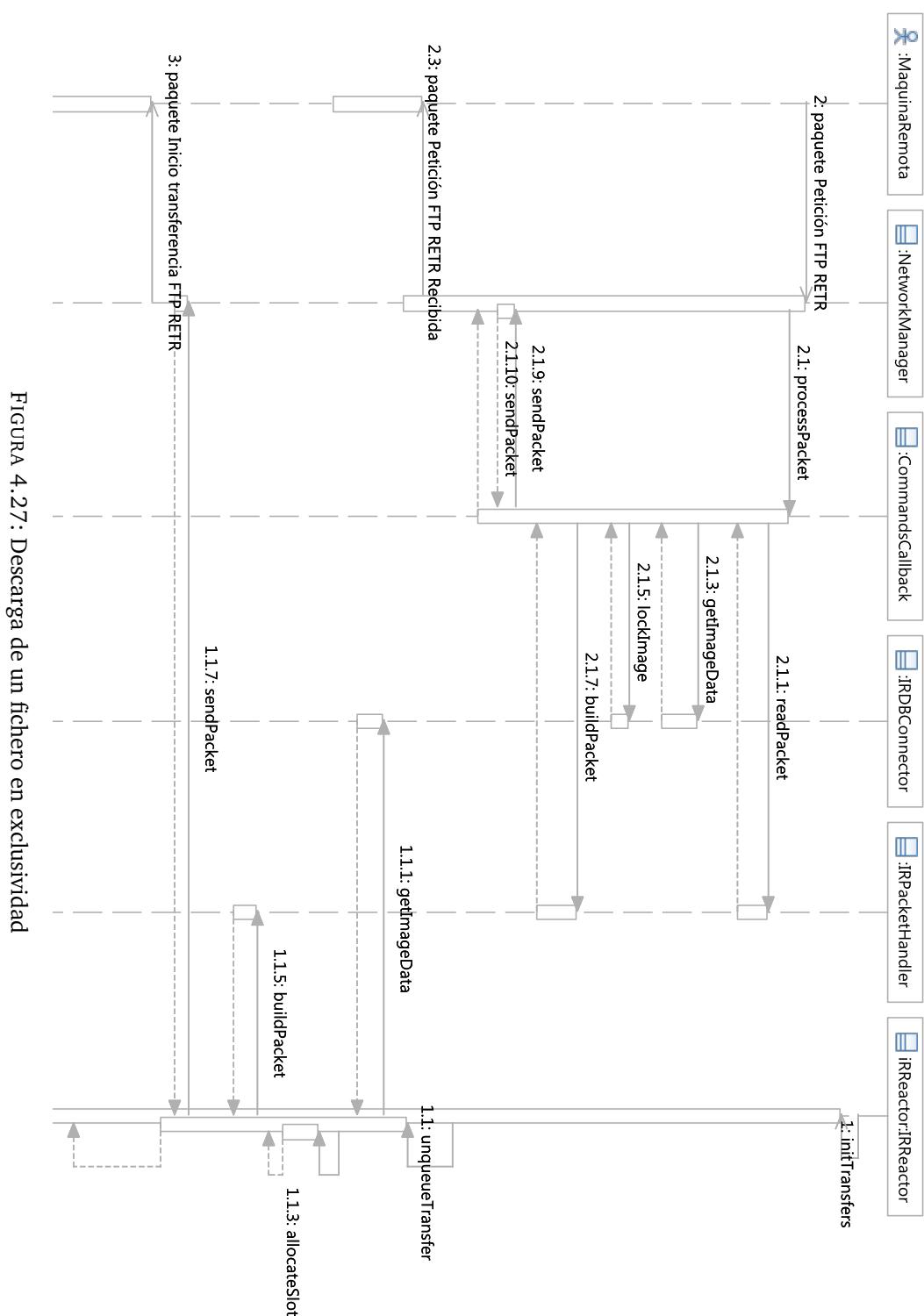


FIGURA 4.27: Descarga de un fichero en exclusividad

Subida de imágenes de disco

Las interacciones que tienen lugar para transferir una imagen de disco desde una máquina remota al repositorio de imágenes son prácticamente idénticas a las interacciones de descarga. Por ello, en este apartado las comentaremos de forma breve.

El diagrama de secuencia de la figura 4.29 muestra la interacción básica de inicio de transferencia. En ella, no se produce ningún error. Como puede observarse en el diagrama, el repositorio de imágenes y la máquina remota intercambian los paquetes

- Petición FTP STOR, que contiene el identificador único del fichero comprimido a subir,
- Petición FTP STOR Recibida, e
- Inicio Transferencia FTP STOR que, nuevamente, contiene toda la información que necesita la máquina remota para conectarse al servidor FTP e iniciar la transferencia.

Además, también debemos recordar que, para aprovechar el ancho de banda del enlace, las transferencias FTP STOR y FTP RETR no tienen por qué desencolarse en el mismo orden en que se reciben. Por claridad, tampoco hemos reflejado esto en el diagrama de secuencia de la figura 4.29.

Por otra parte, en este caso sólo se producirán errores cuando el identificador de la imagen a subir no exista. Si una máquina tiene el fichero comprimido en exclusividad, sólo ella podrá descargarlo y sólo ella podrá subirlo. Así, en las transferencias de subida no es necesario detectar si otra máquina tiene la imagen en exclusividad o no.

Nuevamente, los errores pueden detectarse tanto al recibir la petición de transferencia como al habilitar la subida del fichero. Los diagramas de secuencia de las figuras 4.30 y 4.31 contienen las interacciones correspondientes.

Como puede observarse en los diagramas de secuencia de las figuras 4.30 y 4.31, cuando se suben ficheros al repositorio de imágenes se usan los paquetes de error Petición FTP STOR Errónea y Error en transferencia FTP STOR. Nuevamente, estos paquetes contienen el identificador asociado a las imágenes de disco y un código de descripción del error.

Finalmente, el diagrama de secuencia de la figura 4.32 muestra cómo interactúan el repositorio de imágenes durante la subida de un fichero con imágenes de disco.

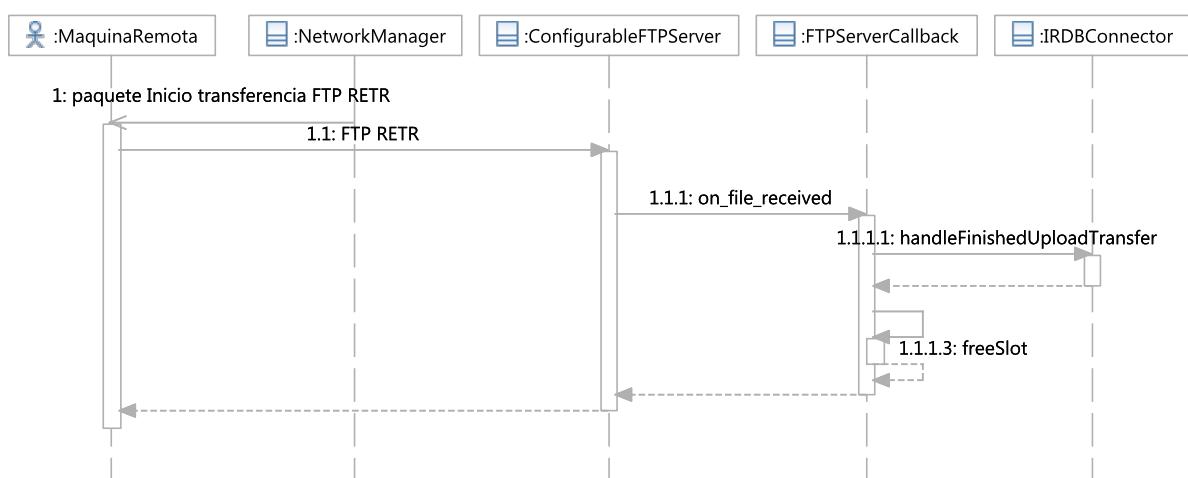


FIGURA 4.32: Transferencia de una imagen desde una máquina remota

Como se puede observar en este último diagrama, cuando se invoca a la función `on_file_received()` del objeto `FTPServerCallback`, se invoca a un método del conector de la base de datos, que registra la ruta del fichero comprimido y modifica el estado de la imagen.

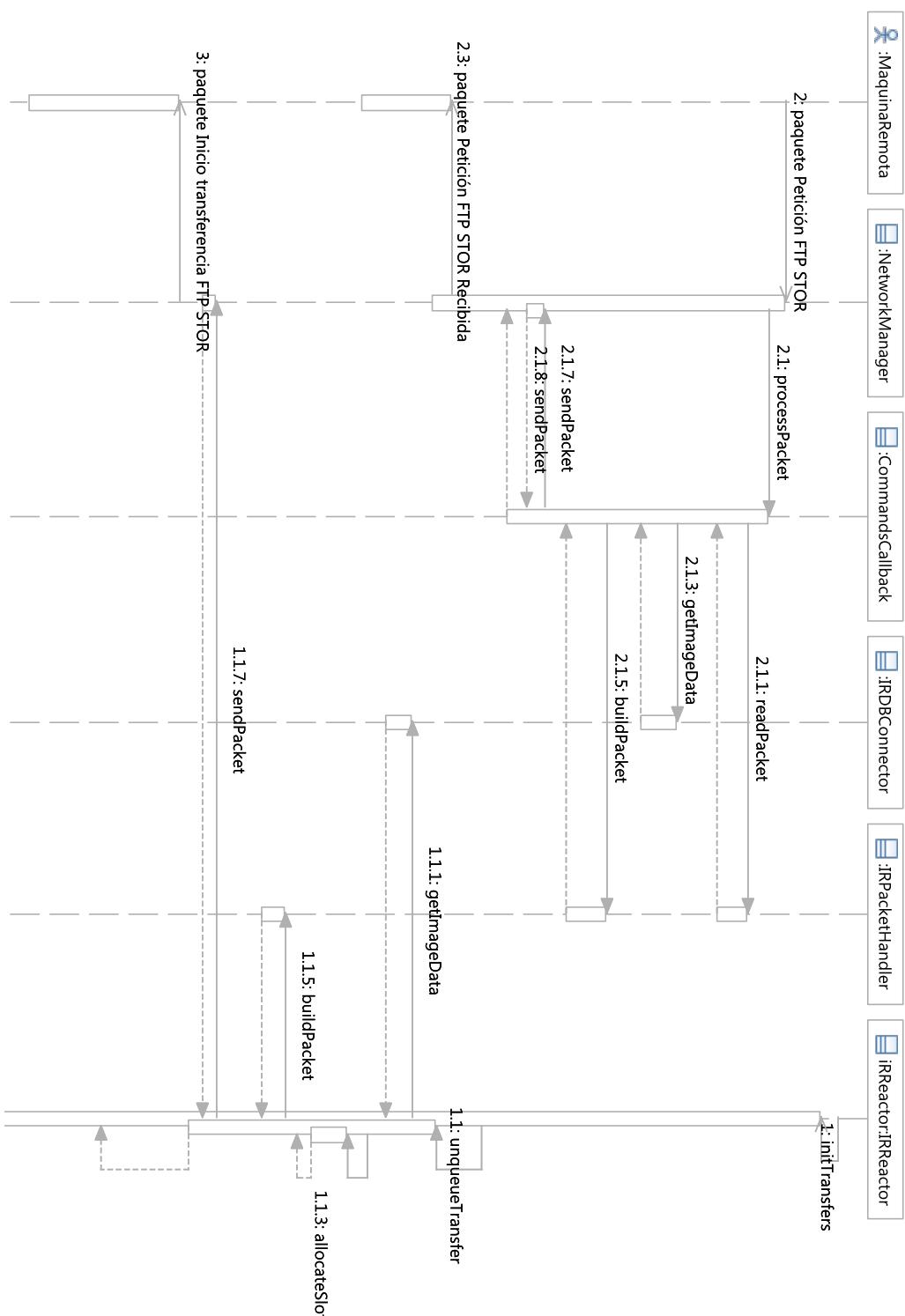


FIGURA 4.29: Inicio de la transferencia de una imagen desde una máquina remota: secuencia básica

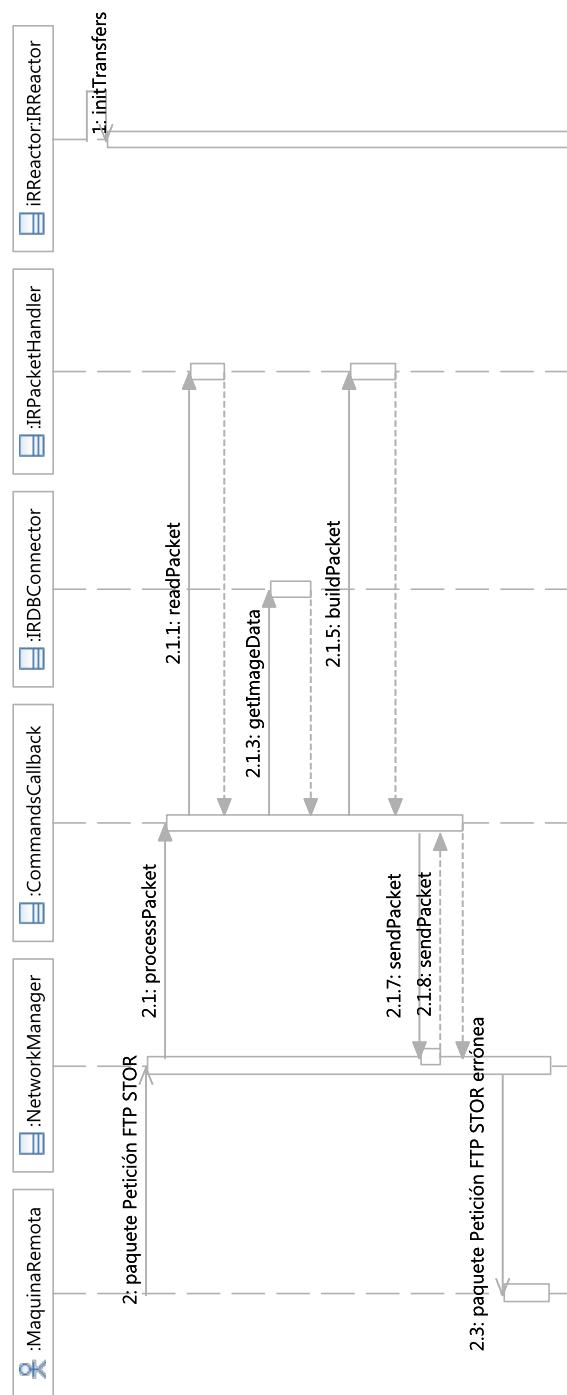


FIGURA 4.30: Deteción de errores al recibir la petición de transferencia

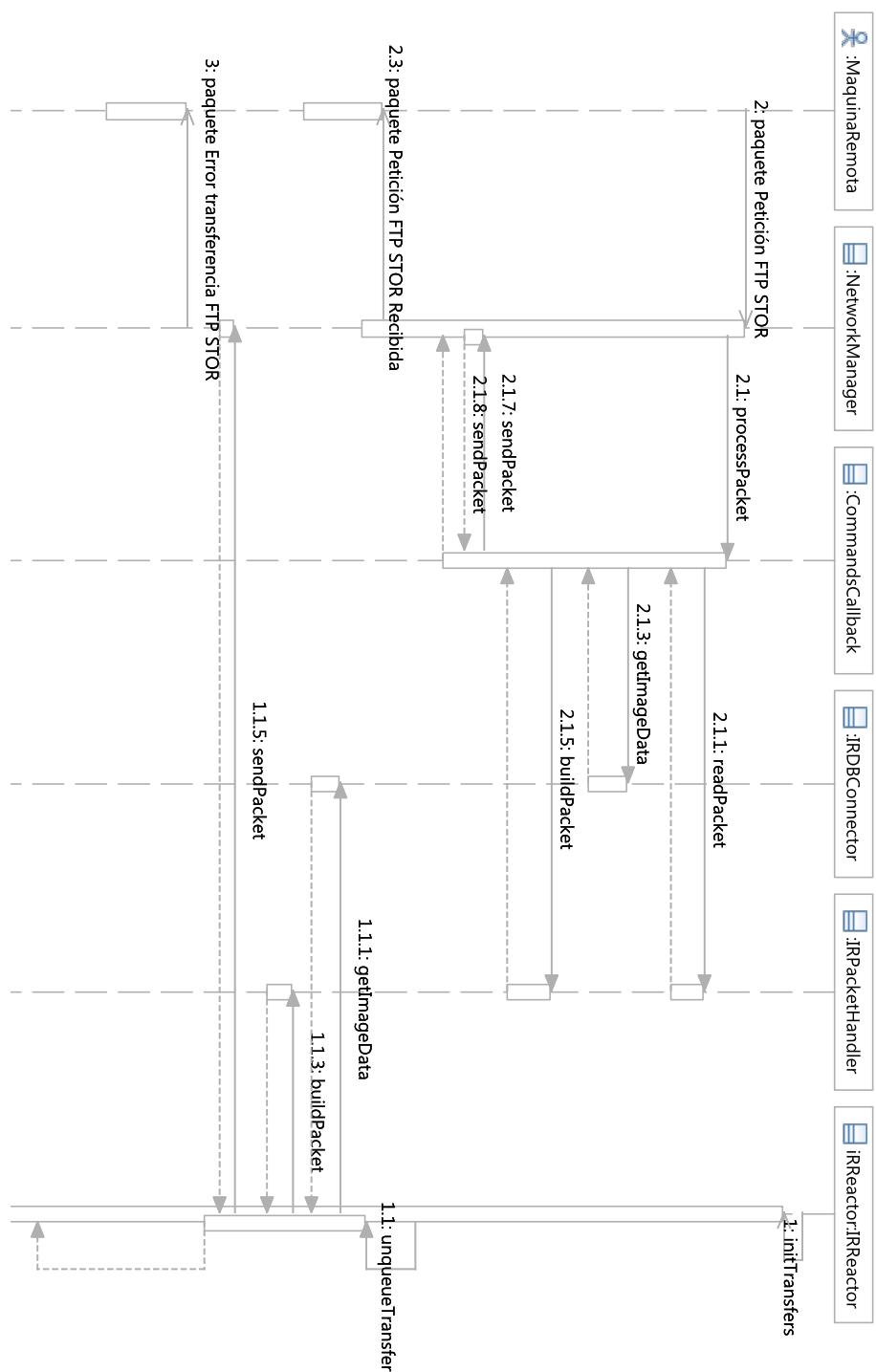


FIGURA 4.31: Detección de errores al habilitar la transferencia

Transferencias parciales

En las dos secciones anteriores, hemos mostrado las interacciones que tienen lugar cuando una máquina remota intercambia un fichero comprimido con el repositorio de imágenes. Por claridad, en ambas hemos omitido el tratamiento de un tipo de error adicional: las transferencias parciales.

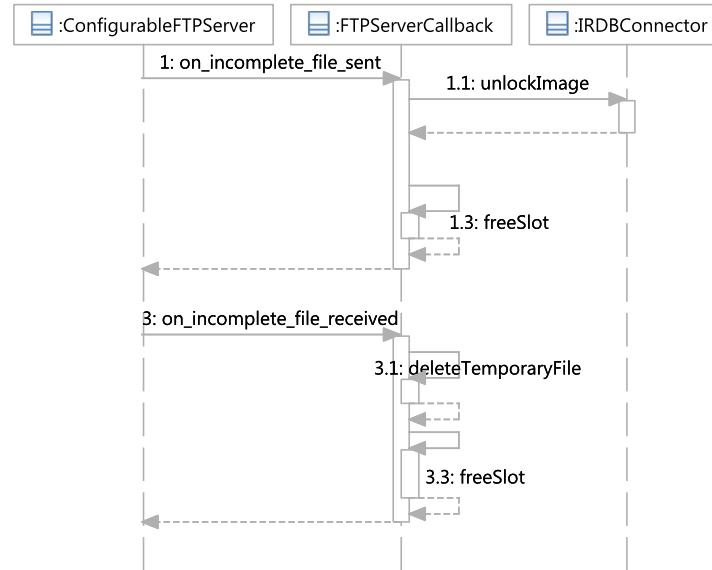


FIGURA 4.33: Tratamiento de las transferencias parciales

El diagrama de secuencia de la figura 4.33 muestra cómo se procesan los eventos de descarga parcial y de subida parcial en el repositorio de imágenes. Como se puede observar en él,

- cuando una transferencia de descarga falla, se libera el fichero correspondiente si estaba asignado en exclusividad a la máquina remota, y
- cuando una transferencia de subida falla, se borran los datos recibidos.

En ambos casos, se libera el *slot* de la transferencia. Además, no es necesario informar a la máquina remota, ya que el cliente FTP que esta utiliza detectará el error.

Borrado de una imagen de disco

El diagrama de secuencia de la figura 4.34 muestra las interacciones que tienen lugar al borrar una imagen de disco y al detectar un error en el borrado de una imagen de disco.

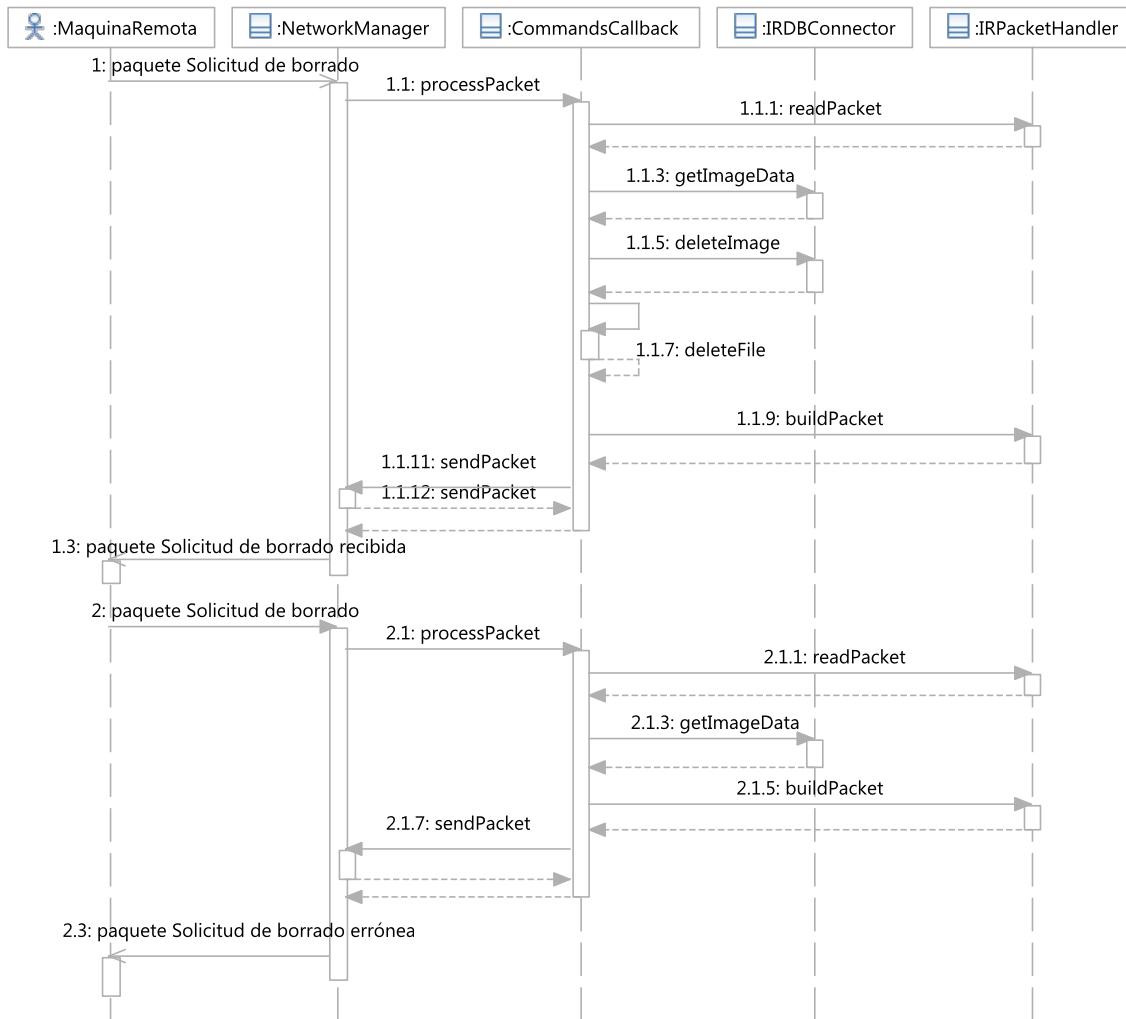


FIGURA 4.34: Interacciones asociadas al borrado de una imagen de disco

Los paquetes del tipo Solicitud de borrado contienen el identificador del fichero a borrar. Tras leer su contenido, el objeto CommandsCallback

1. comprueba que la imagen existe y que ninguna máquina lo tiene en exclusividad. Si alguna de estas condiciones no se cumple, generará y enviará un paquete del tipo Solicitud de borrado errónea, que contiene, junto con el identificador del fichero, un código con la descripción del error.
2. borra el fichero del disco y de la base de datos.
3. envía un paquete del tipo Solicitud de borrado recibida a la máquina remota.

Solicitudes de estado

El diagrama de secuencia de la figura 4.35 muestra cómo se procesan los paquetes de solicitud de estado. Estos permiten que una máquina remota conozca el espacio en disco total y el espacio en disco disponible.

Con esta información podemos garantizar que, siempre que se suba un fichero al repositorio de imágenes, habrá suficiente espacio en disco. Así, no se desperdiciará el ancho de banda que requieren las transferencias de subida que requieren demasiado espacio en disco y que, por tanto, fallarán irremediablemente.

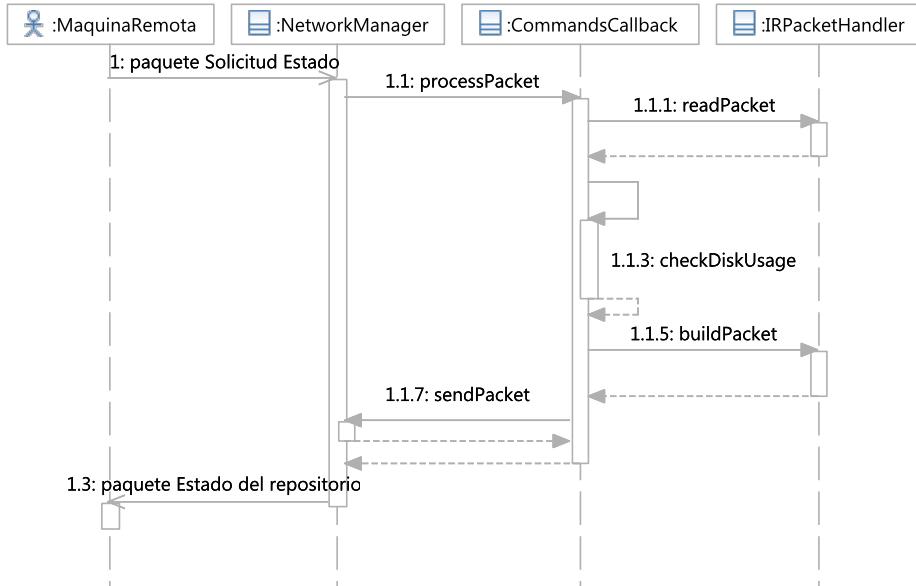


FIGURA 4.35: Recopilación del estado del repositorio

Como puede observarse en el diagrama de la figura 4.35, para averiguar uso de disco del repositorio,

1. la máquina remota envía un paquete del tipo Solicitud de estado.
2. para procesarlo, el objeto CommandsCallback averigua el espacio en disco total y el espacio en disco disponible (en kilobytes),
3. con estos valores, el objeto CommandsCallback construye un paquete del tipo Estado del repositorio y lo envía a la máquina remota.

Apagado del repositorio de imágenes

El diagrama de secuencia de la figura 4.36 muestra el proceso de apagado del demonio del repositorio de imágenes. Como puede observarse en el diagrama de secuencia, el apagado del repositorio es bastante sencillo:

1. cuando se recibe el paquete de apagado, el objeto CommandsCallback modifica una variable compartida con el método `initTransfers()`. Como consecuencia de ello, el método `initTransfers()` termina.
2. el hilo principal invoca al método `stopListening()` del reactor del repositorio.
3. el servidor FTP finaliza su ejecución.
4. se cierran todas las conexiones de red.

Tras esto, el demonio del repositorio de imágenes habrá terminado de ejecutarse.

4.4.5.8. Formatos de paquete

En esta sección, mostraremos detalladamente las características de los tipos de paquete asociados al repositorio de imágenes. El cuadro 4.4.4 muestra los códigos, prioridades y las constantes del tipo enumerado `PACKET_T` (definido en el paquete `imageRepository.packetHandling`) asociados a cada clase de paquete.

Como dijimos en la sección 4.4.3.2, cuanto menor es la valor numérico de la prioridad de un paquete, más prioritario es.

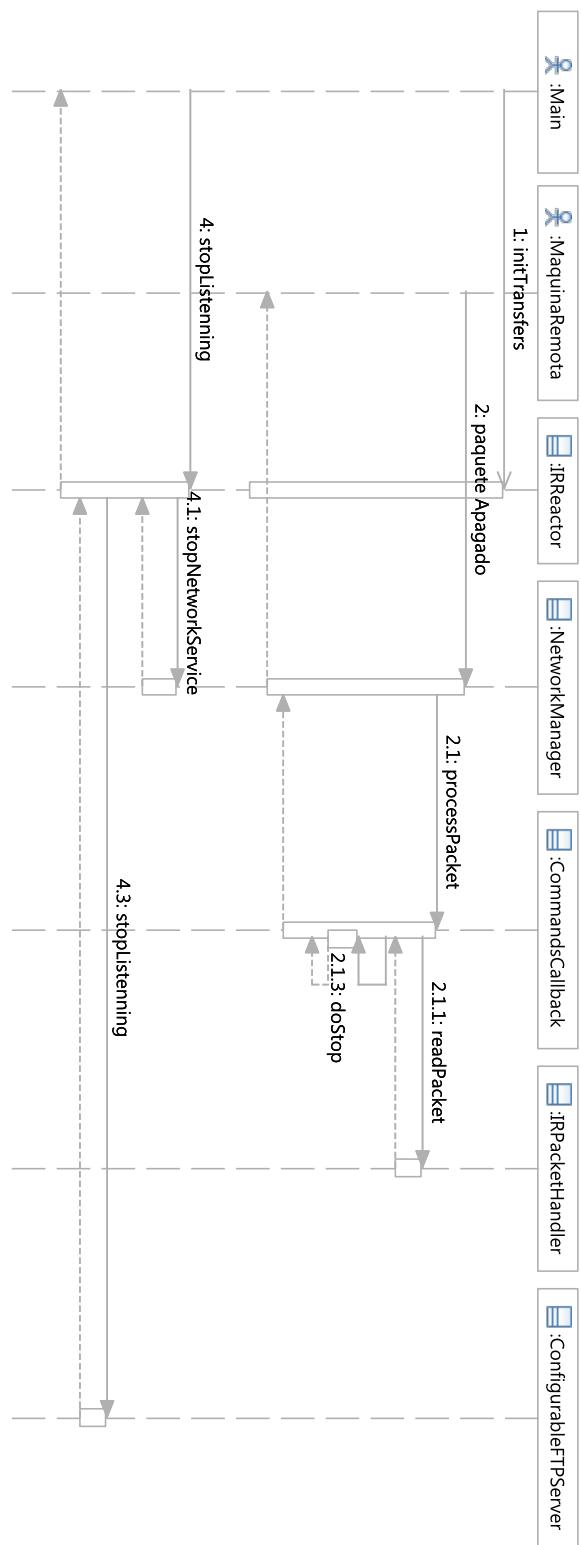


FIGURA 4.36: Apagado del repositorio de imágenes: diagrama de secuencia

Tipo de paquete	Código	Prioridad	Constante (tipo enumerado)
Apagado	1	1	HALT
Crear imagen	2	5	ADD_IMAGE
Imagen creada	3	5	ADDED_IMAGE_ID
Solicitud FTP RETR	4	5	RETR_REQUEST
Solicitud FTP RETR recibida	5	5	RETR_REQUEST_RECV
Solicitud FTP RETR errónea	6	4	RETR_REQUEST_ERROR
Inicio de transferencia FTP RETR	7	3	RETR_START
Error en la transferencia FTP RETR	8	4	RETR_ERROR
Solicitud FTP STOR	9	5	STOR_REQUEST
Solicitud FTP STOR recibida	10	5	STOR_REQUEST_RECV
Solicitud FTP STOR errónea	11	4	STOR_REQUEST_ERROR
Inicio de transferencia FTP STOR	12	3	STOR_START
Error en la transferencia FTP STOR	13	4	STOR_ERROR
Solicitud de borrado	14	2	DELETE_REQUEST
Solicitud de borrado recibida	15	5	DELETE_REQUEST_RECV
Solicitud de borrado errónea	16	4	DELETE_REQUEST_ERROR
Solicitud de estado	17	5	STATUS_REQUEST
Estado del repositorio	18	5	STATUS_DATA
Liberar imagen	19	2	CANCEL_EDITION
Imagen liberada	20	5	IMAGE_EDITION_CANCELLED

CUADRO 4.4.4: Características principales de los paquetes utilizados en el repositorio de imágenes

El contenido de los distintos tipos de paquete es el siguiente:

- los paquetes del tipo **Apagado** no contienen información adicional.
- los paquetes del tipo **Crear imagen** tampoco contienen información adicional.
- los paquetes del tipo **Imagen creada** contienen un nuevo identificador de imagen generado por el repositorio de imágenes.
- los paquetes del tipo **Solicitud FTP RETR** contienen el identificador único del fichero a transferir, y un *flag* `modify` que indica si desea descargar el fichero en exclusividad o no.
- los paquetes del tipo **Solicitud FTP STOR** contienen el identificador único del fichero a transferir.
- los paquetes de los tipos **Solicitud FTP RETR recibida** y **Solicitud FTP STOR recibida** contienen el identificador único del fichero a descargar
- los paquetes de los tipos **Solicitud FTP RETR errónea**, **Solicitud FTP STOR errónea**, **Error en la transferencia FTP RETR** y **Error en la transferencia FTP STOR** contienen el identificador único del fichero a descargar y un código de descripción del error.
- los paquetes del tipo **Inicio de transferencia FTP RETR** e **Inicio de transferencia FTP STOR** contienen todo lo que necesita una máquina remota para subir o descargar una imagen desde el servidor FTP del repositorio de imágenes, es decir
 - el nombre de usuario y la contraseña del servidor FTP
 - el identificador del fichero a transferir
 - el puerto en el que escucha el servidor FTP,
 - el directorio del servidor donde se encuentra el fichero a transferir y
 - el nombre del fichero a transferir.

- los paquetes del tipo **Solicitud de borrado** contienen el identificador de la imagen a borrar.
- los paquetes del tipo **Solicitud de borrado recibida** contienen el identificador único de la imagen borrada.
- los paquetes del tipo **Solicitud de borrado errónea** contienen el identificador único del fichero a borrar (un valor entero) y un código de descripción del error.
- los paquetes del tipo **Solicitud de estado** no contienen información adicional
- los paquetes del tipo **Estado del repositorio** contienen el espacio en disco total y el espacio en disco utilizado en el repositorio de imágenes.
- los paquetes del tipo **Liberar imagen** e **Imagen liberada** contienen el identificador único del fichero a liberar.

Finalmente, los *flags*, los identificadores de imagen, los puertos y los códigos de error son valores enteros. El resto de datos que transportan los paquetes del repositorio de imágenes son *strings*.

4.4.5.9. Distribución de los ficheros

Como hemos visto en las interacciones correspondientes de la sección 4.4.5.7, cuando una transferencia FTP finaliza se llamará al método `on_file_sent()` o al método `on_file_received()` del objeto `FTPServerCallback`. Estos métodos reciben, como único argumento, la ruta del fichero cuya transferencia acaba de finalizar.

Es importante notar que, tras finalizar una transferencia, debemos actualizar el estado de las imágenes de disco correspondientes en la base de datos. Por ello es necesario que, a partir del nombre del fichero comprimido, sea posible obtener el identificador de las imágenes de disco.

Así, los nombres de los ficheros comprimidos no pueden fijarse arbitrariamente, y deben seguir cierto convenio. Por simplicidad, los nombres de todos los ficheros comprimidos que se almacenan en el repositorio de imágenes son de la forma

```
<identificador de las imágenes>.zip
```

Por ejemplo, el fichero comprimido cuyas imágenes de disco tienen asociado el identificador 1 se llamará `1.zip`.

Por otra parte, resulta interesante que, cuando una máquina remota se conecte al servidor FTP, sólo pueda leer o escribir un único fichero comprimido. Por ello, en el directorio raíz del repositorio de imágenes existen múltiples subdirectorios. Cada uno de ellos contiene un único fichero comprimido, y su nombre también se fija a partir del identificador de las imágenes de disco almacenadas en dicho fichero comprimido.

Por ejemplo, la ruta del fichero comprimido `1.zip`, que contiene las imágenes de disco asociadas al identificador 1, será la siguiente:

```
<directorío raíz servidor FTP>/1/1.zip
```

4.4.5.10. Esquema de la base de datos

La base de datos del repositorio de imágenes contiene una única tabla, `Image`, cuyas columnas son las siguientes:

- `imageID`. Este valor entero es la clave primaria de la tabla, y es el identificador único de las imágenes de disco que hemos mencionado en múltiples ocasiones.
Las imágenes de disco de cada máquina virtual comparten el mismo identificador único, y están almacenadas en el mismo fichero comprimido.
- `compressedFilePath`. Se trata de un *string* de hasta 100 caracteres de longitud, que almacena la ruta absoluta del fichero comprimido correspondiente en el repositorio de imágenes.

- `imageStatus`. Este valor ocupa un *byte*, y codifica los tres posibles estados de las imágenes de disco, que aparecen en el cuadro 4.37.

Para evitar errores, en el código fuente no hemos manipulado estos valores directamente, sino a través del tipo enumerado `IMAGE_STATUS_T`, definido en el paquete `imageRepository.database`.

Código	Constante (tipo enumerado)	Descripción
0	<code>NOT_RECEIVED</code>	El identificador de las imágenes de disco está reservado, pero no tiene asociado ningún fichero comprimido en el repositorio de imágenes.
1	<code>READY</code>	El repositorio de imágenes dispone del fichero comprimido. Además, ninguna máquina lo ha descargado en exclusividad.
2	<code>EDITION</code>	El repositorio de imágenes dispone del fichero comprimido. En este caso, una máquina lo ha descargado en exclusividad.

FIGURA 4.37: Codificación del estado de una imagen

4.4.6. El paquete virtualMachineServer

Como dijimos en la sección 4.3, los servidores de máquinas virtuales albergan

- las máquinas virtuales que utilizan los usuarios,
- las redes virtuales a través de las cuales las máquinas virtuales se conectan a la red troncal de la UCM, y
- las máquinas virtuales cuya configuración está siendo editada por un profesor o por un administrador.

En esta sección, mostraremos en detalle su diseño.

4.4.6.1. Redes virtuales

Creación de una red virtual en modo NAT

En todos los servidores de máquinas virtuales existe una red virtual configurada en modo NAT. A través de ella, las máquinas virtuales pueden acceder a la red troncal de la UCM.

Para crear las redes virtuales, ejecutamos las órdenes `virsh net-define` y `virsh net-create` durante el arranque del servidor de máquinas virtuales.

La orden `virsh net-define` recibe como argumento la ruta de un fichero `.xml` en el que se define la configuración de la red virtual. La figura 4.38 muestra el contenido de un fichero de configuración asociado a una red virtual que opera en modo NAT.

```

<network>
  <name>default</name>
  <bridge name="virbr0" />
  <forward/>
  <ip address="192.168.122.1" netmask="255.255.255.0">
    <dhcp>
      <range start="192.168.122.2" end="192.168.122.254" />
    </dhcp>
  </ip>
</network>

```

FIGURA 4.38: Fichero de definición de una red virtual configurada en modo NAT

Como podemos observar, para configurar la red virtual basta con

- indicar el nombre de la red virtual.
- indicar el nombre de la interfaz virtual asociada al encaminador predeterminado. En el caso de la figura 4.38, esta se llamará `virbr0`.
- especificar la dirección IP y la máscara de red del encaminador predeterminado como atributos del elemento `ip`.
- indicar el rango de direcciones que asignará el servidor DHCP. Para ello, se utiliza el elemento `dhcp`.

Cuando se ejecuta la orden `virsh net-create`, se configuran convenientemente las redirecciones de paquetes en el *kernel* (a través de la orden `iptables`) y se lanza el proceso servidor DHCP (`dnsmasq`). A partir de este momento, toda máquina virtual que se conecte a la puerta de enlace podrá acceder a la red virtual y, por tanto, a la red troncal de la UCM.

Finalmente, las redes virtuales se destruyen ejecutando las órdenes `virsh net-destroy` y `virsh net-undefine`. La primera elimina las redirecciones de paquetes y mata el servidor DHCP, y la segunda borra la configuración de la red virtual.

Nótese que es posible destruir una red virtual manteniendo su configuración. En estos casos, es posible reactivar directamente la red virtual ejecutando la orden `virsh net-create`.

La clase `VirtualNetworkManager`

La clase `VirtualNetworkManager` define e implementa una interfaz que permite manipular redes virtuales a un elevado nivel de abstracción. El diagrama de clases de la figura 4.39 muestra sus dependencias.

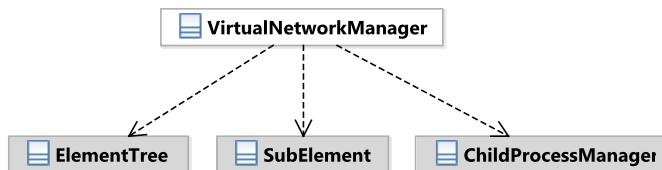


FIGURA 4.39: Relaciones de la clase `VirtualNetworkManager`

Las clases `ElementTree` y `SubElement` forman parte de la librería estándar de *Python*, y se utilizan para generar el fichero `.xml` con la configuración de la red.

Por otra parte, la clase `ChildProcessManager` dispone de métodos que permiten ejecutar comandos en *foreground* y *background*. La clase `VirtualNetworkManager` los utiliza para ejecutar los comandos `virsh net-define`, `virsh net-create`, `virsh net-undefine` y `virsh net-destroy`.

4.4.6.2. Interacción con `libvirt` a bajo nivel

En esta sección, mostraremos cómo se interactúa a bajo nivel con la librería `libvirt` para crear y destruir máquinas virtuales.

Para comenzar, explicaremos qué recursos son necesarios para arrancar una máquina virtual. Posteriormente, presentaremos el formato de los ficheros de configuración de máquinas virtuales y, finalmente, mostraremos la forma en que se interactúa con `libvirt` desde *Python*.

Recursos asignados a una máquina virtual

Toda máquina virtual debe tener asociados, como mínimo,

- una o más CPUs virtuales, todas pertenecientes a la misma arquitectura,

- una memoria principal, y
- una o más imágenes de disco, en las que se almacenarán el *software* instalado y los datos de la máquina virtual.

Aunque estos recursos son suficientes para que la máquina virtual arranque, no permiten interactuar con ella, por lo que esta será totalmente inútil. Por tanto, también es necesario

- utilizar un servidor VNC, que se usará para interactuar con la máquina virtual, o
- utilizar una conexión de red que permita interactuar con la máquina vía SSH (*Secure Shell*) o telnet.

Puesto que en la mayoría de casos los usuarios utilizarán interfaces gráficas de usuario para interactuar con la máquina virtual, el uso de un servidor VNC es imprescindible. Además, los usuarios también necesitarán acceder a la red troncal de la UCM o a internet para poder extraer datos de la máquina virtual, por lo que es fundamental que esta esté conectada a una red virtual.

Finalmente, la máquina virtual también debe tener asignada una dirección MAC, que usará para enviar y recibir tráfico a través de la red virtual, un nombre (que debe ser único para cada máquina virtual alojada en el mismo servidor) y un UUID (*Universally Unique Identifier*), que utilizará libvirt para identificar la máquina virtual de forma única.

Formato de los ficheros de definición

Para crear una máquina virtual a través de libvirt es necesario reservar previamente todos los recursos de los que hemos hablado en la sección anterior e incluirlos en un fichero de definición. En esta sección, nos centraremos en el formato de estos ficheros.

```

<domain type='kvm'>
    <name>Squeeze_AMD64</name>
    <uuid>34ed2109-cd6d-6048-d47c-55bea73e39fd</uuid>
    <memory>1048576</memory>
    <currentMemory>1048576</currentMemory>
    <vcpu>1</vcpu>
    <os>
        <type arch='x86_64' machine='pc-1.0'>hvm</type>
        <boot dev='hd' />
    </os>
    <features>
        <acpi /> <apic /> <pae />
    </features>
    <clock offset='utc' />
    <on_poweroff>destroy</on_poweroff>
    <on_reboot>restart</on_reboot>
    <on_crash>restart</on_crash>
    <devices>
        <emulator>/usr/bin/kvm</emulator>
        <disk type='file' device='disk'>
            <driver name='qemu' type='qcow2' cache='writeback' io='native' />
            <source file='/home/luis/SqueezeAMD64.qcow2' />
            <target dev='vda' bus='virtio' />
        </disk>
        <interface type='network'>
            <source network='default' /> <mac address='52:54:00:8a:56:41' />
        </interface>
        <input type='tablet' bus='usb' />
        <input type='mouse' bus='ps2' />
        <graphics type='vnc' port='15010' autoport='no' passwd='CCRules!'>
            <keymap>es</keymap>
            <listen type='address' address='192.168.0.5' />
        </graphics>
        <video>
            <model type='cirrus' vram='9216' heads='1' />
        </video>
    </devices>
</domain>

```

FIGURA 4.40: Fichero de definición de una máquina *Linux*

La figura 4.40 muestra el contenido de un fichero de definición válido para máquinas virtuales *Linux*. Las características más relevantes de este fichero son las siguientes:

- el atributo *type* del elemento *domain* indica la solución de virtualización en la que se utilizará la máquina virtual. En nuestro caso, toma el valor *kvm*, ya que utilizamos el hipervisor KVM. Es importante notar que las funciones que se pueden especificar en los ficheros de definición varían en función del sistema de virtualización que estemos usando.
- los elementos *name*, *uuid*, *vcpu* y *memory* especifican el nombre, el UUID, el número de CPUs virtuales y el tamaño de la memoria de la máquina virtual.
- el elemento *os* define, esencialmente, la arquitectura del sistema operativo de la máquina virtual y la secuencia de arranque de la misma. En el caso del fichero de definición anterior, el sistema operativo se corresponderá con la arquitectura *x86_64*, y la máquina virtual arrancará directamente del primer disco duro.

- el elemento `features` permite activar las características ACPI (*Advanced Configuration and Power Interface*), APIC (*Advance Programmable Interrupt Controller*) y PAE (*Physical Address Extension*) de la máquina virtual.

Para lo que nos ocupa, podemos decir que la práctica totalidad de las CPUs x86 y de los sistemas operativos existentes en el mercado soportan estas características, por lo que es mejor dejarlas activadas. En caso de que se detecte alguna incompatibilidad, el sistema operativo de la máquina virtual podrá deshabilitar estas características.

- el elemento `clock` permite configurar el desfase del reloj. Puede tomar dos valores: `localtime` y `utc`, que difieren en la forma en que se almacena la hora en la máquina virtual.

Sin entrar en detalles, basta con tener en cuenta que el elemento `clock` debe tomar el valor `localtime` en sistemas *Windows*, y `utc` en sistemas tipo *UNIX*.

- los elementos `on_poweroff`, `on_reboot` y `on_crash` determinan cómo actuar cuando la máquina virtual se apaga, se reinicia o falla respectivamente. En el fichero de definición de la figura 4.40, la máquina virtual se destruirá al apagarse, y se reiniciará en el resto de casos.
- el elemento `devices` permite definir la configuración del subsistema de entrada/salida de la máquina virtual. Destacan sus elementos

- `emulator`, que especifica el hipervisor a utilizar. En nuestro caso, utilizamos `kvm`.
- `disk`, que especifica la configuración de un disco duro. La máquina virtual asociada al fichero de definición de la figura 4.40 tiene un único disco duro, cuyos datos se extraen de una imagen de disco `qcow2`.

Para obtener el mejor rendimiento posible, este disco duro se conecta al bus *VirtIO* que, como sabemos, proporciona un rendimiento muy similar al de un disco duro real.

- `network`, que especifica la configuración de una tarjeta de red. La máquina virtual asociada al fichero de definición de la figura 4.40 se conectará a la red virtual `default`, utilizando para ello la dirección MAC `52:54:00:8a:56:41`.
- `input`, que especifica un dispositivo de entrada. Habitualmente, los ficheros de definición especifican dos: un ratón PS/2 y una tableta gráfica USB que permiten el correcto movimiento del cursor cuando se utiliza el protocolo *VNC*.
- `graphics`, que especifica la configuración del servidor *VNC* asociado a la máquina virtual. Entre otras cosas, es necesario especificar el puerto (`port`), la contraseña (`passwd`), la distribución del teclado (`keymap`) y la dirección IP (atributo `address` del elemento `listen`) en la que escuchará el servidor *VNC*.

Finalmente, los ficheros de definición de máquinas virtuales *Windows* son muy similares. Tan sólo es necesario dar el valor `localtime` al elemento `clock`: el resto de elementos se configuran como ya hemos visto.

Interacción con libvirt desde código Python

La clase `libvirtConnector`

4.4.7. El paquete `clusterServer`

4.4.8. El paquete `webServer`

En este paquete se trata todo lo relacionado con la gestión de la web de CygnusCloud. Así en este paquete se incluyen todos los módulos encargados de gestionar las funcionalidades de interacción con los usuarios a través de la web y los ficheros html y css necesarios para la implementación de las páginas.

4.4.8.1. Estructura general de la web

Para llevar a cabo el desarrollo de la web ha sido necesario la utilización de un framework web que nos facilite algunos aspectos y estructurarse el sistema de forma correcta. Para ello, hemos optado por la utilización de Web2Py cuyas principales características y funcionamiento explicamos en la sección [4.3.22](#).

Como ya mencionamos en esa sección, web2py estructura las aplicaciones siguiendo un modelo vista-controlador. Por ello en CygnusCloud ha sido necesario estructurar el código de la web en un grupo de control y un grupo de vista. A parte de ambos grupos también se han añadido más grupos con ficheros estáticos, css, módulos importados y en general todos los ficheros necesarios para dar la funcionalidad y el aspecto requeridos en la web.

Con todo esto, la web de CygnusCloud puede subdividirse en los siguientes grupos según su finalidad y tipo:

- **Vistas:** Este grupo incluye todos los ficheros de formato html encargados de establecer el aspecto de los componentes, su posicionamiento en las páginas y su comportamiento. Entendemos por comportamiento de un componente, su capacidad para realizar ciertas acciones que mejoren la interacción con los usuarios. Según las restricciones de web2py será necesario disponer de un fichero de vista por cada sección disponible en la web. Cada uno de estos ficheros de vista reciben los componentes creados en el controlador asociado, ajustan su aspecto al que se mostrará en la web, y los posicionan en el lugar deseado.
- **Controladores:** Este grupo incluye el conjunto de módulos python que aportan todas las funcionalidades que serán gestionadas por la web. CygnusCloud dispone de un módulo para cada uno de los tipos de usuarios que pueden acceder a la web (alumnos, administradores y estudiantes), un módulo para las páginas de acceso público, un módulo para la página que contiene el cliente VNC y un módulo appAdmin creado automáticamente por web2py para la gestión de las bases de datos. Dentro de cada módulo encontramos una función asociada a cada una de las secciones que forman la web. Hablaremos más en profundidad sobre este tema en la sección [4.4.8.3](#).
- **Modelos:** Este grupo incluye dos módulos python encargados de la creación de las bases de datos y las características generales de la aplicación. Así, el módulo “db.py” crea las tablas de la base de datos encargada de la gestión de la información en la web. El módulo “menu.py” define las características generales de la aplicación web, tales como su logo, su autor, sus tags de búsqueda... Este par de módulos son creados y exigidos por web2py.
- **Lenguajes:** Este grupo incluye los diccionarios de traducción de las páginas de la web. Estos diccionarios deben ser rellenados por el desarrollador y son usados internamente por web2py para traducir todas las palabras de la web que aparezcan en los diccionarios.
- **Archivos estáticos:** Este grupo incluye los ficheros css y js encargados de definir la apariencia de la web, las imágenes y en general cualquier fichero que defina algún componente externo que forma parte de la web.
- **Módulos:** En este apartado se incluyen todos los módulos python secundarios utilizados por los controladores. Así, en este grupo se encuentra el conector que permite la interacción entre el servidor web y el servidor de cluster.

4.4.8.2. Estructura de direcciones

Con el fin de poder gestionar la web de la forma más modular posible y aprovechando el flujo de direcciones usadas por web2py vamos a estructurar los diferentes apartados de la web en forma de árbol de direcciones tal que cada uno de sus nodos podrá realizar saltos a cualquier otro nodo del árbol según las acciones realizadas por los usuarios. Así el elemento principal será la aplicación en sí (/CygnusCloud). A partir de esta dirección comenzarán a surgir las direcciones asociadas a las páginas manteniendo la estructura de secciones y subsecciones correspondientemente. De esta forma, las diferentes URLs sobre las que trabajará la web son:

- “/CygnusCloud/login” : Es la página de inicio de sesión donde el usuario introduce su nombre y contraseña para iniciar la sesión.
- “/CygnusCloud/about” : Es la página en la que se habla acerca de CygnusCloud y su finalidad.
- “/CygnusCloud/student/runVM” : Es la página de arranque de máquinas virtuales para los usuarios.
- “/CygnusCloud/administrator/runVM/run”: Es la página de arranque de máquinas virtuales para los administradores.
- “/CygnusCloud/administrator/servers/add_servers” : Es la página encargada de añadir nuevos servidores y borrar algún servidor existente.
- “/CygnusCloud/administrator/servers/remove_servers” : Es la página encargada de eliminar servidores de máquinas virtuales existentes.
- “/CygnusCloud/administrator/users/remove” : Esta es la página encargada de eliminar a usuarios previamente creados.
- “/CygnusCloud/administrator/users/add” : Es la página encargada de crear nuevos usuarios con las especificaciones determinadas por el administrador.
- “/CygnusCloud/administrator/users/associate_subjects” : Es la página encargada de establecer las relaciones entre un usuario previamente creado y un grupo de asignatura.
- “/CygnusCloud/administrator/subjects/add” : Esta es la página encargada de crear nuevos grupos de asignaturas con la información introducida por el administrador.
- “/CygnusCloud/administrator/subjects/remove” : Esta es la página encargada de eliminar algún grupo de asignatura existente.
- “/CygnusCloud/vncClient/VNCPage” : Página que contiene el escritorio de trabajo ejecutado por noVNC.

La distribución de estas páginas con respecto a su estructura en el flujo de direcciones viene representado en el diagrama de la figura 4.41.

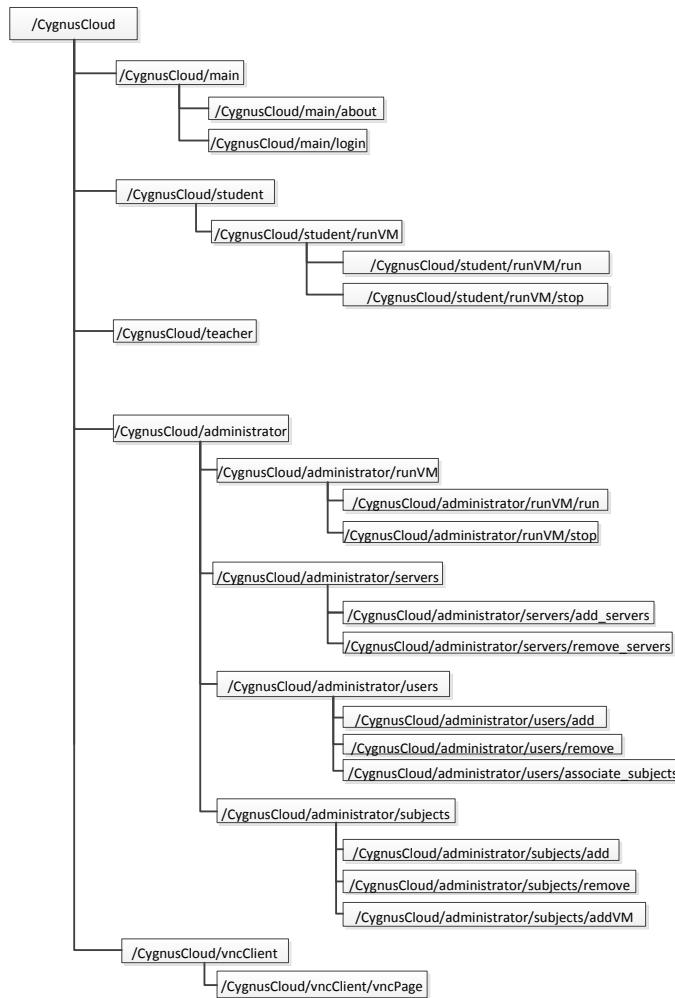


FIGURA 4.41: Diagrama de direcciones web2py.

4.4.8.3. Secciones y subsecciones

Para poder hacer frente al elevado número de páginas que ofrece CygnusCloud, hemos optado por estructurar estas páginas en un sistema de 3 niveles, que permite filtrarlas teniendo en cuenta los privilegios de acceso y la finalidad de cada página.

El primer nivel se centra en los privilegios de acceso. CygnusCloud agrupa a los usuarios registrados en 3 tipos según los privilegios de los que dispongan. Estos privilegios dan acceso a unas u otras páginas. Podemos encontrar los siguientes tipos de usuarios:

- Alumnos : En este tipo se incluyen todos los alumnos que cursan alguna de las asignaturas registradas en la web. Este tipo de usuarios es el más restringido, dándole únicamente acceso a la página de arranque de máquinas virtuales , asociadas a las asignaturas matriculadas por el alumno, y a la página de visualización de máquinas arrancadas.
- Profesores : Este tipo incluye a todos los profesores que imparten alguna de las asignaturas registradas en la web. Dispone de unos privilegios de acceso mayores que el caso del alumno, pudiendo acceder, además de las páginas de arranque de máquinas virtuales y gestión de máquinas arrancadas, a páginas de creación, edición y borrado de máquinas virtuales.

- Administradores: Este tipo incluye a todos los técnicos y administradores encargados de la gestión total de la página. Es el tipo más privilegiado, teniendo acceso a las páginas de :
 - Arranque de todas máquinas virtuales.
 - Detención de máquinas virtuales arrancadas independientemente del usuarios que las arrancó.
 - Gestión de usuarios.
 - Gestión de servidores de máquinas virtuales.
 - Gestión de grupos de asignaturas, así como de las máquinas virtuales asociadas a cada grupo.

Además de estos tipos de usuarios, CygnusCloud también ofrece acceso a ciertas páginas públicas. Estas páginas podrán ser vistas por cualquier usuarios que se conecte a la web de CygnusCloud, aunque no se encuentre registrado.

Como ya hemos dicho, cada uno de estos tipos del primer nivel concretan el conjunto de páginas al cual se quiere acceder.

En el segundo nivel las páginas se encuentran agrupadas por secciones. Entendemos como una sección, el conjunto de páginas que abordan un aspecto común para un tipo de usuario concreto. Cada tipo de usuario tendrá acceso a unas secciones u otras. Algunos ejemplos de secciones son la gestión de usuarios, el arranque de máquinas virtuales o el inicio de sesión. En la tabla 4.4.5 puede verse el conjunto de secciones asociadas a cada tipo de usuario.

Tipo de usuario	Secciones
Main	Login,About
Student	RunVM
Teacher	
Administrator	RunVM, Servers, Users, Subjects

CUADRO 4.4.5: Tabla relación tipos de usuario y secciones.

Por último, en el tercer nivel, las páginas se encuentran definidas en subsecciones. Cada subsección es una página que contiene las funcionalidades relacionadas con un aspecto concreto y más limitado que el de la sección que la incluye. Existirán por tanto, tantas subsecciones como páginas creadas en la web. En la tabla 4.4.6 pueden verse las diferentes subsecciones asociadas a cada sección.

Sección	Subsección
Login	
About	
RunVM	Run, Stop
Servers	Add_servers, Remove_servers
Users	Add, Remove, Associate_subjects
Subjects	Add, Remove, AddVM

CUADRO 4.4.6: Tabla relación secciones y subsecciones

La barra de secciones permanente en la web muestra la distribución de las páginas en secciones y subsecciones.

De cara a la implementación, es necesario ajustar esta estructura para que siga el modelo definido por web2py. Web2py gestiona sus direcciones con respecto a la estructura de los módulos python que actúan como controladores. El conjunto de controladores presentes en la web coincide con los tipos de usuarios definidos en el primer nivel. Así disponemos de 5 controladores:

- Uno para las páginas de acceso público.

- Uno para las páginas de los alumnos.
- Uno para las páginas de los profesores.
- Uno para las páginas de los administradores.
- Uno para la página que mantendrá el cliente VNC.

Este último controlador no corresponde a ningún tipo concreto de usuario pero se define a parte ya que su uso es común para todos.

Dentro de cada controlador es necesario definir una función python para cada una de las secciones del nivel 2 asociadas a este tipo de usuario. Así todo el código presente en los controladores debe pertenecer a una función, bien una función que defina una determinada sección o bien una función auxiliar utilizada por alguna de las funciones principales.

Por último cada una de las funciones python contendrán un variable que les indique cual de las subsecciones del tercer nivel deben controlar en cada momento.

Con respecto a las vistas, el diseño de los niveles se simplifica bastante, siendo necesario tener un fichero html para cada una de las secciones del segundo nivel. El nombre de estos ficheros html deben seguir la siguiente estructura:

“Nombre del controlador”/”nombre de la función”.html

A modo de resumen, el esquema 4.42 muestra como CygnusCloud estructura sus páginas en estos 3 niveles. Como podemos observar, las rutas de direcciones de las que hablamos en la sección 4.4.8.2 coinciden con la estructura en los 3 niveles.

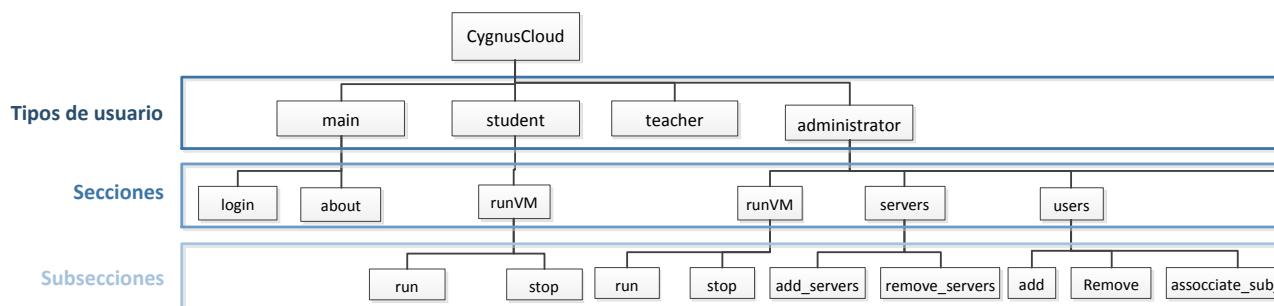


FIGURA 4.42: Diagrama de niveles de la web.

4.4.8.4. La barra de direcciones

Con el fin de permitir a los usuarios acceder a las diferentes páginas de forma directa sin necesidad de utilizar redirecciones a páginas intermedias, la web dispone de una barra de direcciones permanente que permite al usuario redirigirse a cualquier página dentro de su rango de acción. De esta forma dependiendo del tipo de usuarios dispondremos de las siguientes barras:

- barra de direcciones para páginas de acceso público. En esta barra aparecen las páginas de inicio de sesión y de “Acerca de”, a las cuales pueden acceder libremente cualquier usuario que entre en la web.
- barra de direcciones para alumnos. Esta barra incluye la página de arranque de máquinas virtuales y está restringida a usuarios de tipo alumno registrados en la web.
- barra de direcciones para administradores. Esta barra contiene todas las páginas de gestión de servidores, asignaturas y usuarios. Además contiene la página de arranque de máquinas virtuales. Su acceso está restringido a usuarios de tipo administrador.

4.4.8.5. Gestión de usuarios

Para llevar a cabo la gestión de usuarios hemos optado por utilizar una de las librerías que Web2py nos ofrece. Esta librería, de nombre Auth, contiene todas las funcionalidades necesarias con respecto a la gestión de usuarios. Auth contiene funciones para gestionar la creación de usuarios, los inicios de sesión y las restricciones de acceso a diferentes páginas. Además esta librería ofrece también funciones que devuelven los formularios de inicio de sesión y cierre de la misma totalmente implementados para ser directamente incorporados en nuestras páginas. [dP13]

4.4.8.6. Interacción entre páginas

Aunque bien es cierto que cada página debe ser controlada y representada por un controlador y una vista particulares, es posible que requiera de cierta información proveniente de la página a partir de la cual se accedió a ella.

Por esta razón, nuestra web debe ser capaz de trasmitir argumentos entre las diferentes páginas. Como ya mencionamos en el apartado 4.4.8.3, las diferentes secciones que componen la web son interpretadas por parte del controlador como simples funciones python sin argumentos. Aunque a simple vista lo más lógico pueda parecer que la forma correcta de trasmitir información entre páginas sea definiendo las funciones python con tantos argumentos como datos de entrada deba recibir esto no sucede así. Las razones por las cuales el envío y recepción de datos entre páginas no se dé de esta forma son básicamente dos:

1. La redirección de una página a otra no puede darse llamando a la función del controlador directamente. Cuando el desarrollador quiera realizar una redirección, deberá llamar a una función `redirect` con el nombre del controlador que la contiene, el nombre de la función de la sección correspondiente, los argumentos necesarios y el conjunto de variables. Una vez llamada a la función, es Web2py el encargado de llamar a la función del controlador que corresponda y pasar los argumentos y variables.
2. El número de argumentos no siempre es el mismo. Dependiendo de la página a partir de la cual pueda realizarse la redirección, es posible que la función de la sección correspondiente reciba más o menos argumentos por lo que estos no pueden definirse como argumentos directos en la definición de la sección.

Por lo tanto la forma correcta de trasmitir información entre las diferentes páginas será utilizando la función `redirect`. Como podemos observar en la definición dada, esta función recibe además del nombre del controlador y la sección, un campo `argumentos` y un campo `variables`. Ambos campos permiten trasmitir información entre dos páginas con unas características particulares para cada uno de ellos.

En el caso de trasmitir información por el campo de argumentos, esta formará parte de la ruta de direcciones de la página destino. Así todos los argumentos que pasemos por este campo vendrán acoplados a la dirección de la página junto al nombre de la aplicación, el nombre del controlador y el nombre de la sección. Los valores en este campo son trasmitidos como una lista y son accedidos desde la página destino por medio de la llamada a `request.args[i]` siendo `i` la posición de la lista donde se encuentra el elemento deseado.

El segundo campo nos permite transferir la información como variables y no como argumentos. En este caso el nombre de la variable no formará parte de la dirección destino. Este campo recibe un diccionario con el nombre que se quiere dar a cada una de estas variables y su contenido y será accedido desde la página destino por medio de la llamada a `request.vars.x` donde `x` es el nombre que se le dió a la variable concreta a la que se quiere acceder.

En el caso de CygnusCloud, el campo de argumentos se utiliza para retransmitir el nombre de la subsección que debe ejecutarse dentro de la sección correspondiente. El campo de variables se utiliza para trasmitir el resto de información útil como por ejemplo los resultados de las búsquedas.

4.4.8.7. Flujo de ejecución de arranque de una máquina virtual

Con la finalidad de dar una última explicación sobre el funcionamiento interno de la web, vamos a exemplificar de forma detallada cada una de las acciones que debe realizar el sistema para atender

la petición de arranque de una máquina virtual por parte de un alumno que se conecte a nuestra web.

Una vez el alumno haya pulsado el vínculo que le da acceso a la página de CygnusCloud:

- Se ejecuta la función `login` dentro del controlador `Main`. Esta función crea el formulario de inicio de sesión (utilizando la utilidad `auth` que se encarga de la gestión de usuarios) y lo devuelve.
- Justamente después se ejecuta la vista asociada a esta función (“`main/login.html`”), la cual recibe el formulario creado por el controlador, le aplica el aspecto correspondiente y lo coloca en la página.
- Una vez el usuario ha introducido su nombre, contraseña y ha pulsado el botón arrancar, el controlador envía una consulta a la base de datos para comprobar que el usuario existe y que su contraseña es correcta. Tras esto se mira el tipo de usuario y redirecciona a la página que sea necesaria. En nuestro caso, al ser un alumno el que inicia la sesión, redirecciona a la sección `runVM` del controlador `student` con el argumento `run`.
- Una vez en la función `runVM` del controlador `student`, la cual solo es accesible para alumnos registrados, se evalúa el argumento de entrada para comprobar que subsección debe ejecutarse. Tras esto, se extrae la lista de asignaturas asociadas a este alumno concreto. Para ello, se envía una petición a la base de datos preguntando por estos valores. Para cada una de las asignaturas encontradas se crea una tabla, cuyas filas corresponden a las máquinas virtuales que pueden arrancarse en esta asignatura. La información de las máquinas virtuales asociadas a cada asignatura se extrae también a partir de consultas a la base de datos. Cada tabla creada se añade al formulario que es devuelto por la función.
- A continuación se ejecuta la vista “`student/runVM.html`” la cual recibe el formulario, ajusta su aspecto y lo coloca en la página. Además, esta vista, define un script que permite mostrar y ocultar la descripción de las diferentes máquinas virtuales según se encuentren seleccionadas o no.
- Una vez el usuario ha seleccionado alguna máquina virtual y ha pulsado el botón de arrancar, el controlador extrae el identificador de la máquina virtual y pide al conector que arranque dicha máquina. El conector devuelve los parámetros de conexión, los cuales son enviados como variable en la redirección a la página `vncPage`, la cual es creada en una nueva pestaña.
- Llegados al controlador `vncClient` y a su sección única `vncPage`, se extrae la variable con la información de conexión y se devuelve como resultado de la función.
- La vista asociada a este controlador (“`vncClient/vncPage.html`”) recibe los datos de conexión y ejecuta `noVNC`, el cual se encarga de mostrar el escritorio de trabajo por pantalla.

La figura 4.43 muestra un diagrama de secuencia con todo este proceso.

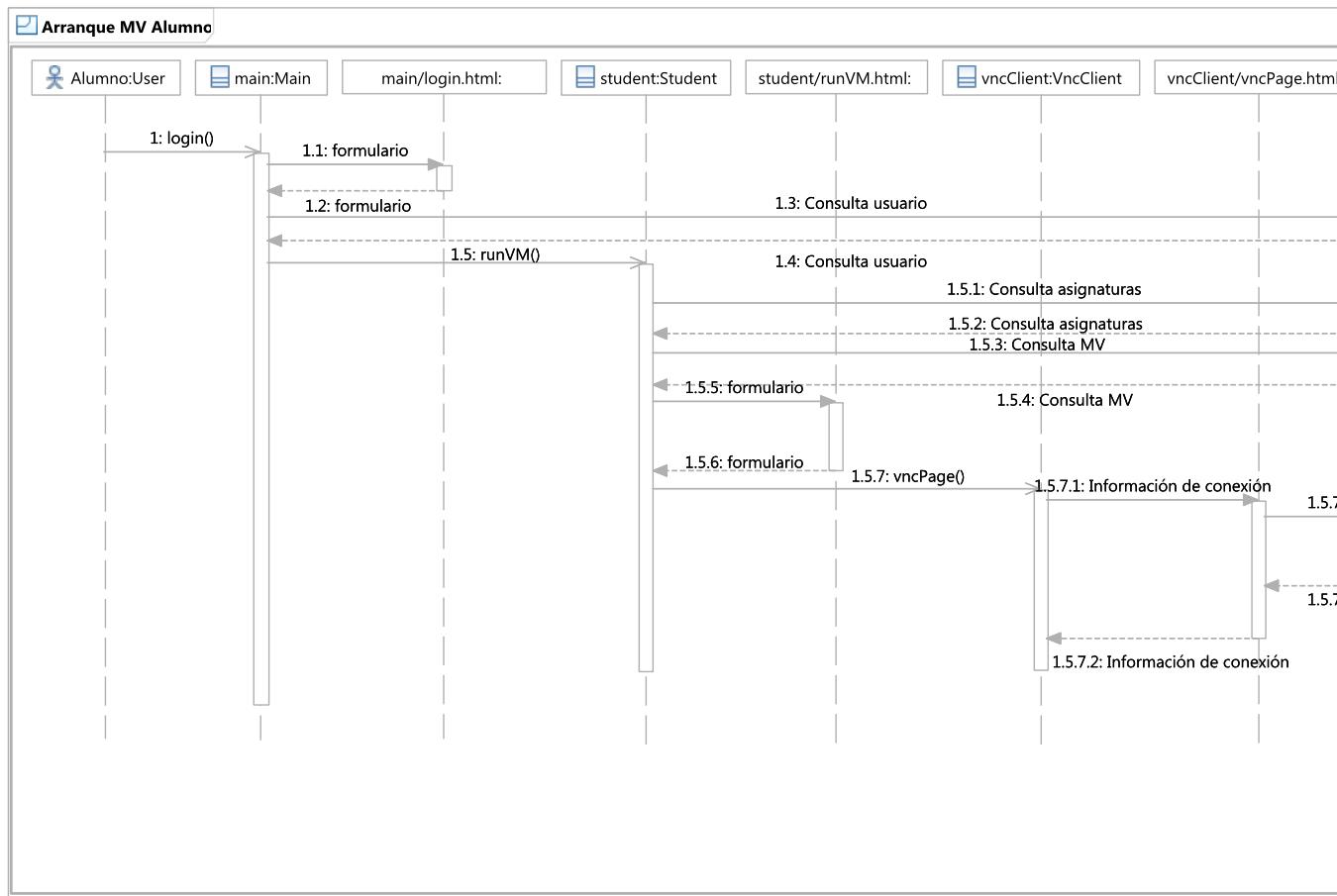


FIGURA 4.43: Secuencia de arranque de una máquina virtual.

4.5. Vista de procesos

4.6. Vista de despliegue

4.7. Vista de implementación

Capítulo 5

Resultados

Capítulo 6

Manual de usuario

6.1. Requisitos mínimos

Cada máquina de la infraestructura de *CygnusCloud* desempeña uno de estos papeles:

- **servidor de máquinas virtuales.** Estos equipos alojan las máquinas virtuales activas de los usuarios.
- **servidor de cluster.** Estas máquinas realizan el balanceado de carga entre varios servidores de máquinas virtuales.
- **servidor web.** Estas máquinas proporcionan una interfaz web que permite a los usuarios enviar peticiones a los servidores de *cluster*.

En esta sección, empezaremos mostrando los requisitos mínimos que debe cumplir el *hardware* de cada una de estas máquinas, para posteriormente mostrar los requisitos *software*.

6.1.1. Requisitos *hardware*

El *hardware* de los servidores de *cluster* y de los servidores web debe cumplir los siguientes requisitos mínimos:

- CPU Intel Pentium 4 a 2,4 GHz
- un mínimo de 512 *megabytes* de memoria RAM. Se recomienda 1 GB para un funcionamiento óptimo.
- un *gigabyte* de espacio en disco
- tarjeta de red *fast ethernet*.

Por otra parte, los requisitos mínimos que debe cumplir el *hardware* de los servidores de máquinas virtuales son los siguientes:

- CPU Intel Core 2 Duo E4400
- un mínimo de 2 *gigabytes* de memoria RAM. Se recomienda 4 GB para un funcionamiento óptimo.
- un mínimo de 30 *gigabytes* de espacio en disco si sólo se desea alojar máquinas virtuales GNU/Linux. Si también se pretende alojar máquinas virtuales Windows, se requiere un mínimo de 60 *gigabytes* de espacio en disco.
- tarjeta de red *fast ethernet*.

Finalmente, todas las máquinas de la infraestructura deben estar conectadas a una misma red de área local, que deberá soportar, como mínimo, una tasa de 100 *megabits* por segundo.

6.1.2. Requisitos software

Todas las máquinas de la infraestructura deben tener instalado un sistema operativo GNU/Linux. En principio, el funcionamiento debería ser correcto en cualquier máquina con una distribución de la misma época o posterior a Ubuntu 12.04 LTS, aunque por falta de tiempo sólo podemos garantizar que el funcionamiento es correcto en Ubuntu 12.04 LTS.

En todas las máquinas de la infraestructura, debe estar instalados, además del sistema operativo,

- una versión de la rama 2.7 del intérprete de Python, igual o superior a la 2.7.3.
- la librería de red *twisted*, versión 12.1.0 o superior.
- el gestor de bases de datos MySQL, versión 5.5.29 o superior.
- el conector de MySQL MySQLdb, versión 1.2.3 o superior.

Importante

- a causa de las limitaciones impuestas por libvirt y por la librería *twisted*, *no* es posible utilizar un intérprete de Python de la rama 3.x.
- la última versión de la librería *twisted*, 13.0.1, provoca errores cuando se hacen pruebas en local utilizando el adaptador *loopback*. Por ello, le recomendamos el uso del paquete que forma parte de su distribución GNU/Linux.

Por otra parte, en los servidores de máquinas virtuales debe estar instalado el siguiente *software* adicional:

- un *kernel* de la versión 3.2.0 o superior.
- el hipervisor de KVM correspondiente a la versión del *kernel* que se encuentra instalada.
- el sistema de virtualización QEMU, versión 2.0 o superior
- la librería libvirt, versión 0.9.8 o superior

6.2. Instrucciones de instalación y configuración en Ubuntu 12.04

6.2.1. Pasos comunes

Con independencia del papel que desempeñe la máquina en la infraestructura, siempre es necesario instalar la librería *twisted*, el gestor de bases de datos MySQL y el conector MySQLdb. Para ello, debemos ejecutar la orden

```
sudo apt-get install python-twisted mysql-server python-mysqldb
```

6.2.2. Servidores de máquinas virtuales

6.2.2.1. Instalación y configuración de KVM

Una vez instalados los paquetes comunes, debemos instalar el hipervisor KVM, la librería *libvirt* y el sistema de virtualización QEMU. Para ello, ejecutamos la orden

```
sudo apt-get install qemu-kvm libvirt-bin ubuntu-vm-builder bridge-utils
```

Tras instalar estos paquetes, es necesario añadir el usuario que vamos a utilizar a los grupos *libvirt* y *kvm*. Para ello, ejecutamos las órdenes

```
sudo addgroup <nombre de nuestro usuario> libvirt
```

y

6.2.2.2. Instalación del módulo de *CygnusCloud*

```
sudo addgroup <nombre de nuestro usuario> kvm
```

Además, debemos configurar libvirt para que nos permita crear máquinas y redes virtuales sin necesidad de ser *root*. Para ello, descomentamos las siguientes líneas del fichero */etc/libvirt/libvirtd.conf*:

```
■ unix_sock_group = "libvirtd"  
■ unix_sock_ro_perms = "0777"  
■ unix_sock_rw_perms = "0770"  
■ auth_unix_ro = "none"  
■ auth_unix_rw = "none"
```

Finalmente, reiniciamos libvirt mediante el comando

```
sudo service libvirtd restart
```

y reiniciamos nuestra sesión. A partir de este momento, podremos crear máquinas virtuales con nuestro usuario.

6.2.2.2. Instalación del módulo de *CygnusCloud*

Los pasos a seguir son los siguientes:

1. descomprimimos el fichero *VirtualMachineServer.tar.gz* en el directorio de instalación. Para ello, ejecutamos la orden

```
tar xvzf VirtualMachineServer.tar.gz -C <ruta del directorio de instalación>
```

2. tomamos posesión de todos los ficheros extraídos. Para ello, ejecutamos la orden

```
sudo chown -R <nuestro usuario>:<nuestro usuario> <ruta del directorio de instalación>
```

3. ejecutamos el *script* *build.py*, ubicado en la raíz del directorio de instalación. Para ello, utilizamos el comando

```
python build.py
```

o el comando

```
sh build.py
```

El *script* comprobará que se satisfacen todas las dependencias e informará de los errores que detecte.

4. si se satisfacen las dependencias, el *script* generará el fichero *virtualMachineServer.sh*. Este fichero es el *script* de arranque del módulo servidor de máquinas virtuales.

6.2.2.3. Configuración del módulo de *CygnusCloud*

Si intenta ejecutar el módulo sin haber rellenado previamente el fichero de configuración, se producirán errores.

En el caso del servidor de máquinas virtuales, el fichero de configuración, `VMServer_settings.conf`, también se encuentra en la raíz del directorio de instalación. Su contenido inicial es el siguiente:

```
# Virtual machine server configuration file
# Version 1.5
# File format:
# - if nothing appears after =, the value will be the empty string
# - multiline string values start with " and end with "
# - the lines starting with # are line comments. Inline comments ARE NOT #
# supported. #
# Delete the following line after editing this file
uninitializedFile = Yes
# Database configuration
mysqlRootsPassword =
databaseUserName = cygnuscloud
databasePassword = cygnuscloud
# Virtual network setup
createVirtualNetworkAsRoot = No
vnName = ccnet
gatewayIP = 192.168.77.1
netMask = 255.255.255.0
dhcpStartIP = 192.168.77.2
dhcpEndIP = 192.168.77.254
# VNC server settings
vncNetworkInterface = lo
passwordLength = 46
# Server settings
listeningPort = 15800
configFilePath = /home/luis/VirtualMachineServer/configuraciones/
sourceImagePath = /home/luis/VirtualMachineServer/imagenes/
executionImagePath = /home/luis/VirtualMachineServer/imagenes_en_ejecucion/
```

Tal y como indica el encabezado del fichero,

- todas las líneas que empiezan por almohadilla (#) son comentarios.
- todos los valores que aparecen a la derecha del símbolo = son cadenas de caracteres. No pueden contener espacios. Si no aparece nada a la derecha del símbolo =, el valor será la cadena vacía.
- es posible que una misma cadena de caracteres ocupe varias líneas, siempre que se delimita con comillas dobles (""). En cualquier caso, estas cadenas *tampoco* pueden contener espacios.

La primera línea del fichero indica que este no ha sido modificado, y debe borrarse tras terminar de editarlo. Tras ella, aparecen cuatro grupos de parámetros:

- el primer grupo especifica la configuración de la base de datos del servidor de máquinas virtuales. Además de la contraseña de *root*, imprescindible para poderla crear, hay que suministrar el nombre y la contraseña del usuario que se utilizará para manipularla.
- el segundo grupo especifica la configuración de la red virtual: su nombre, la IP de la puerta de enlace, la máscara y las direcciones inicial y final del rango que manipulará el servidor DHCP.

La primera línea de este grupo indica si se debe crear la red virtual como *root* o no. Esto no es necesario en Ubuntu 12.04, pero sí en distros derivadas de RedHat.

- el tercer grupo especifica la configuración del servidor VNC: la interfaz en la que escuchará y la longitud de la contraseña.
- el cuarto grupo especifica el puerto en el que el servidor escuchará, y también
 - la ruta del directorio en el que están los ficheros .xml de definición de las máquinas virtuales (configFilePath).
 - la ruta del directorio donde están las imágenes de disco de las máquinas que se pueden arrancar (sourceImagePath)
 - la ruta de directorio donde están las imágenes de disco de las máquinas activas (executionImagePath)

Todos los directorios deben estar en un sistema de ficheros tipo UNIX. Además, el usuario actual debe tener permisos de lectura en los dos primeros, y de lectura y escritura en el último.

Importante

- Si tiene configurado un *firewall* en su equipo o red, asegúrese de que el puerto de escucha que ha escogido y los puertos TCP del 15000 al 15507 están abiertos.
 - Puesto que la funcionalidad de modificación de imágenes está en desarrollo, las pruebas, por defecto, sólo pueden hacerse en local. Por ello, el servidor VNC debe escuchar en el adaptador *loopback*, y el servidor de máquinas virtuales, en el puerto 15800.
- Esta limitación puede superarse registrando manualmente los servidores de máquinas virtuales en el sistema, cosa que enseñaremos a hacer en la siguiente sección.

6.2.3. Servidores de cluster

6.2.3.1. Instalación del módulo de *CygnusCloud*

La instalación del módulo es totalmente análoga a la de la sección 6.2.2.2, aunque en este caso

- el fichero .tar.gz se llama ClusterServer.tar.gz
- el script build.py genera el fichero clusterServer.sh.

6.2.3.2. Configuración del módulo de *CygnusCloud*

Nuevamente, es necesario configurar el módulo para poder utilizarlo. Para ello, se utiliza el fichero ClusterServer_settings.conf, también ubicado en la raíz del directorio de instalación. Su contenido inicial es el siguiente:

```
# Cluster server configuration file
# Version 2.0
# File format:
# - if nothing appears after =, the value will be the empty string
# - multiline string values start with " and end with "
# - the lines starting with # are line comments. Inline comments ARE NOT #
supported.
# Delete the following line after editing this file
uninitializedFile = Yes
# Database configuration
mysqlRootsPassword =
dbUser = cygnuscloud
dbPassword = cygnuscloud
# Listenning port
listenningPort = 9000
# Virtual machine boot timeout (in seconds)
vmBootTimeout = 30
```

El formato de este fichero es el mismo que el de la sección [6.2.2.3](#). En este caso, distinguimos tres grupos:

- en el primero se configura la base de datos, introduciendo nuevamente la contraseña de *root*, el nombre de usuario y la contraseña a utilizar para manipularla.
- en el segundo se indica el puerto en el que el servidor de *cluster* escuchará.
- en el tercero se indica (en segundos) el *timeout* considerado al arrancar máquinas virtuales. Cuando se supere ese tiempo, el proceso de arranque se abortará, y se asumirá que ha fallado.

Finalmente, también es necesario borrar la primera línea del fichero tras editar este fichero de configuración.

Importante

- Si su equipo o red de área local utilizan un *firewall*, asegúrese de que, además de los puertos en los que escuchan los servidores de máquinas virtuales, el puerto TCP 9000 está abierto.

6.2.4. Servidor web

6.2.4.1. Instalación del módulo de *CygnusCloud*

Al igual que en los casos anteriores, la instalación del servidor web es análoga a la de la sección [6.2.2.2](#), aunque en este caso

- el fichero `.tar.gz` se llama `WebServer.tar.gz`, y
- el *script* `build.py` genera dos ficheros, `webServerEndpoint.sh` y `webServer.sh`

Para arrancar el servidor web, hay que ejecutar cada uno de ellos en un proceso independiente. El primero, `webServerEndpoint.sh`, lanza el demonio que comunica la aplicación `web2py` y el servidor de *cluster*, y el segundo lanza el servidor web que aloja la aplicación `web2py`.

6.2.4.2. Configuración del módulo de *CygnusCloud*

En este caso será necesario editar dos ficheros de configuración.

El primero, `Endpoint_settings.conf`, contiene los datos de configuración del proceso que comunica la aplicación `web2py` y el servidor de *cluster*. Este fichero se encuentra en la raíz del directorio de instalación y su contenido inicial es el siguiente:

```

#
# Cluster server endpoint configuration file
# Version 1.3
#
# File format:
# - if nothing appears after =, the value will be the empty string
# - multiline string values start with " and end with "
# - the lines starting with # are line comments. Inline comments ARE NOT #
supported.
#
# Delete the following line after editing this file
uninitializedFile = Yes
# Database configuration parameters
mysqlRootsPassword =
statusdbSQLFilePath = ../../database/SystemStatusDB.sql
commandsdbSQLFilePath = ../../database/CommandsDB.sql
websiteUser = website
websiteUserPassword = CygnusCloud
endpointUser = connector_user
endpointUserPassword = CygnusCloud
# Network parameters
clusterServerIP = 127.0.0.1
clusterServerListenningPort = 9000
statusDBUpdateInterval = 2

```

En este fichero, , distinguimos dos grupos:

- en el primero se configura la base de datos, introduciendo la contraseña de *root*, la ruta de los ficheros con la estructura de la base de datos y el nombre y contraseña para los usuarios encargados de gestionar ambos servidores.
- en el segundo se indica el puerto y la ip para la conexión con el servidor de cluster, así como el intervalo de actualización de la información del servidor de cluster accesible al servidor web (en segundos).

Por otra parte, el fichero *userInputConstants.py* contiene los datos de configuración del servidor web. Nuevamente, este fichero se encuentra en la raíz del directorio de instalación y su contenido por defecto es el siguiente:

```

#
# Web server configuration file
# Version 1.2
#
# Database configuration parameters
rootPassword =
#Network parameters
serverIp = "127.0.0.1"
serverPort = 9000

```

En este fichero, debemos introducir la contraseña del usuario *root* de MySQL y la IP y el puerto en los que estará disponible el endPoint.

6.3. Creación de imágenes

Para poder crear máquinas virtuales, es necesario disponer previamente de algunas imágenes de disco en formato *qcow2*. En esta sección mostraremos cómo crearlas de forma sencilla utilizando *virt-manager* e importarlas en *CygnusCloud*. Naturalmente, también es posible importar a *CygnusCloud* imágenes creadas manualmente. No obstante, por simplicidad, en este manual *no* mostraremos cómo crear una imagen manualmente.

6.3.1. Creación de nuevas imágenes en virt-manager

6.3.1.1. Pasos básicos

Antes de utilizar `virt-manager`, es necesario instalarlo. Asumiendo que utilizamos Ubuntu 12.04, basta con ejecutar la orden

```
sudo apt-get install virt-manager
```

Acto seguido, lo ejecutamos. Aparecerá la pantalla inicial de la figura 6.1.

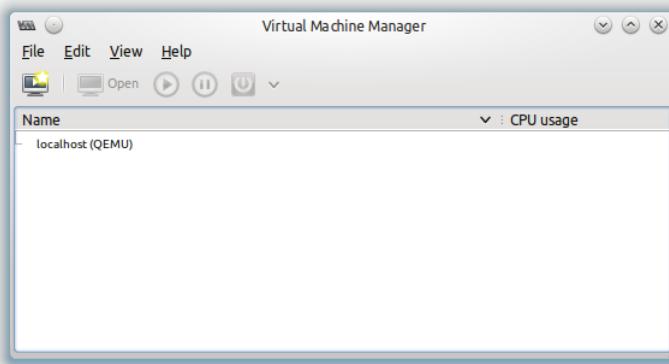


FIGURA 6.1: Pantalla inicial de `virt-manager`

Para empezar a crear una máquina, hacemos clic sobre el botón `Create a new virtual machine`. Aparecerá la primera ventana de la figura 6.3. En todos los casos, avanzamos al paso siguiente haciendo clic sobre `Forward`.

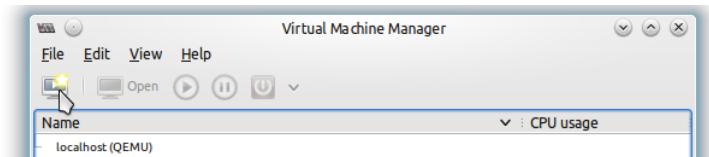


FIGURA 6.2: Ubicación del botón `create a new virtual machine`

Los pasos a seguir son los siguientes:

1. Introducimos el nombre de la máquina virtual.
2. Aparecerá la segunda ventana de la figura 6.3, en la que tendremos que seleccionar
 - la ubicación del medio de instalación del sistema operativo
 - el tipo del sistema operativo (Windows, Linux,...) y su versión
3. Aparecerá la primera ventana de la figura 6.4, en la que tendremos que introducir el tamaño de la RAM y el número de CPUs virtuales de la máquina.
4. Desmarcamos la casilla `Enable storage for this virtual machine`.
5. En la ventana final del asistente, que aparece en la figura 6.5, marcamos la casilla `Customize configuration before install`, y hacemos clic en `Finish`.

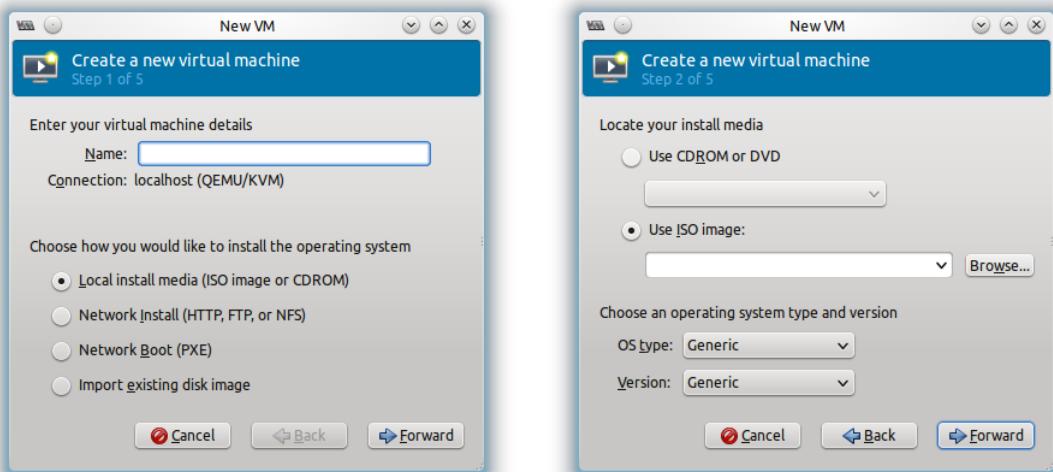


FIGURA 6.3: Pasos 1 y 2 del asistente de creación de máquinas virtuales

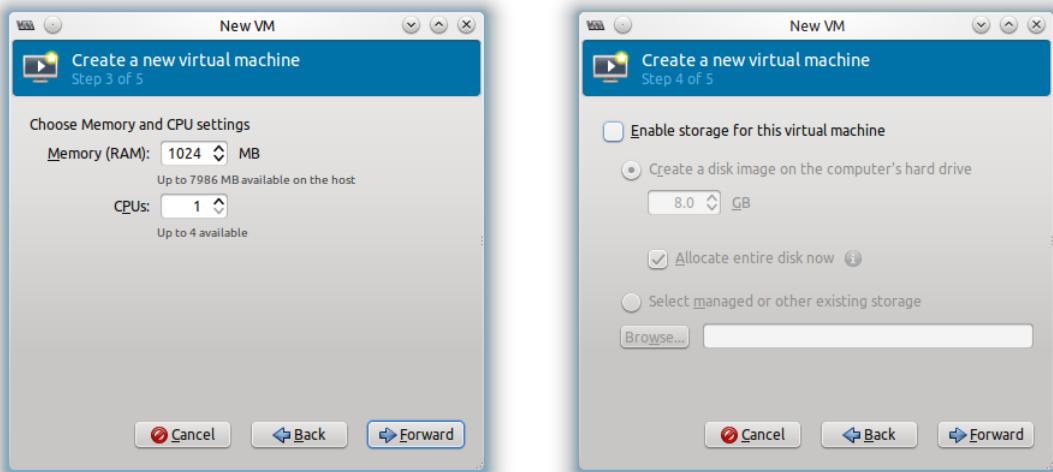


FIGURA 6.4: Pasos 3 y 4 del asistente de creación de máquinas virtuales

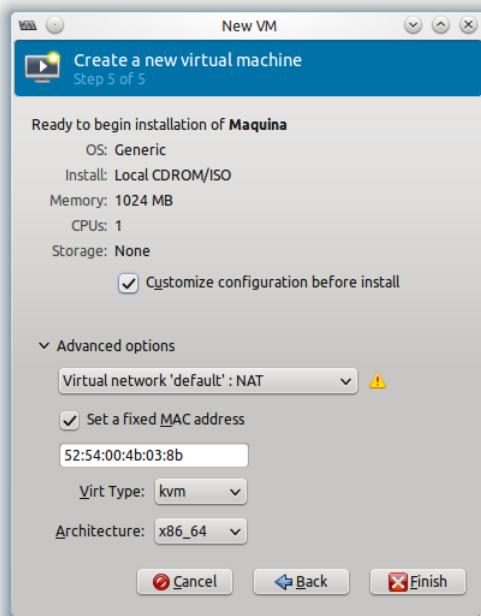


FIGURA 6.5: Paso 5 del asistente de creación de máquinas virtuales

6.3.1.2. Configuración de la máquina virtual

Con los pasos de la sección 6.3.1.1, `virt-manager` ya ha generado casi todo el fichero de definición de la máquina. No obstante, todavía debemos configurar

- las características de la CPU de la máquina virtual
- los discos duros virtuales
- los periféricos de entrada
- la tarjeta de vídeo

Tras salir del asistente, aparecerá la ventana de la figura 6.6.

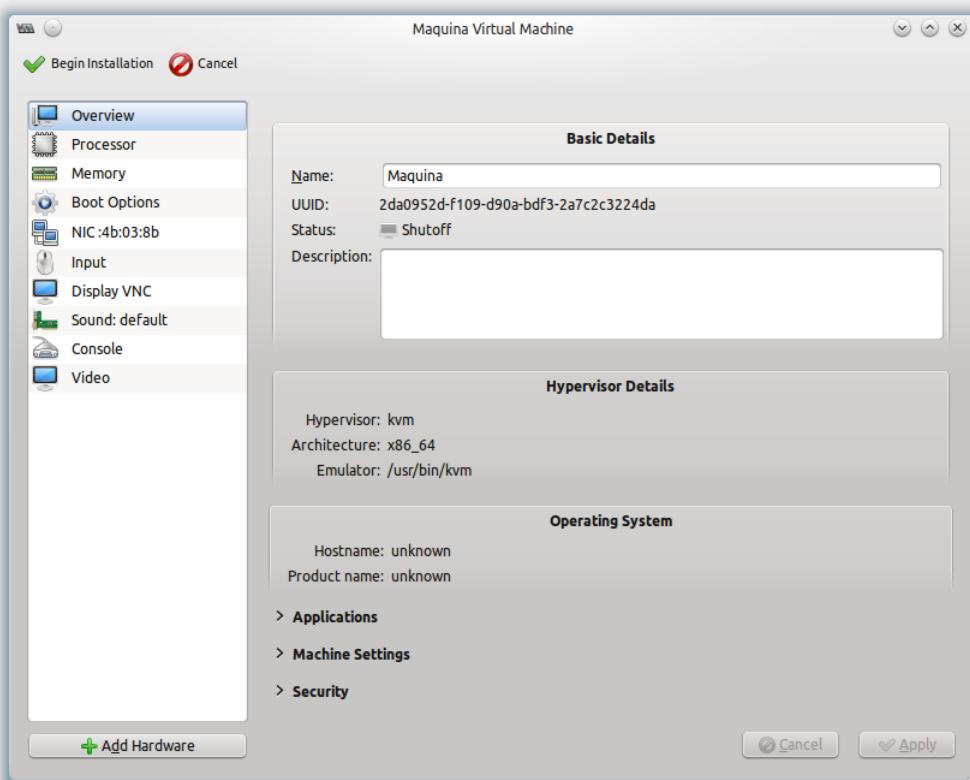


FIGURA 6.6: Diálogo de configuración de la máquina virtual

Empezaremos configurando la CPU de la máquina. Para ello, hacemos clic sobre `Processor`. Tras expandir el apartado `Configuration`, seleccionamos `Core2Duo` como modelo. Todo debería quedar configurado como en la figura 6.7.

A continuación, crearemos las imágenes de disco `qcow2` que utilizará la máquina. Tenemos que crear dos:

- una que contendrá el sistema operativo y los programas instalados
- otra que contendrá los datos del usuario y el fichero de paginación

Para crearlas, hacemos clic sobre el botón `Add Hardware` y seleccionamos `Storage`. El diálogo que aparece se muestra en la figura 6.8.

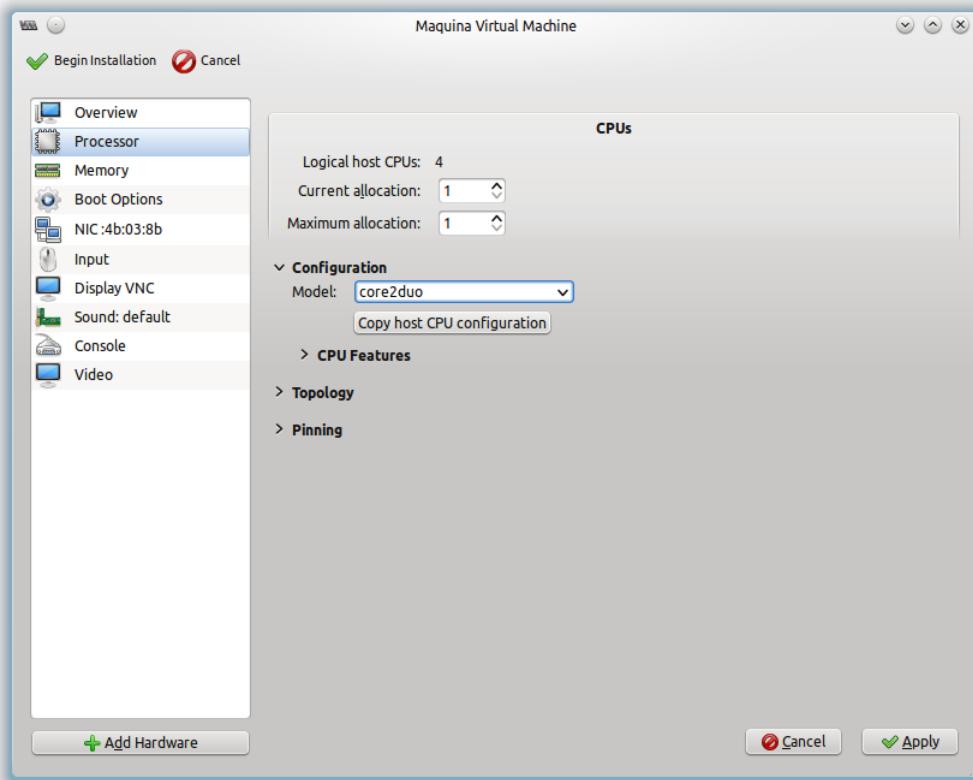


FIGURA 6.7: Configuración del procesador de la máquina virtual

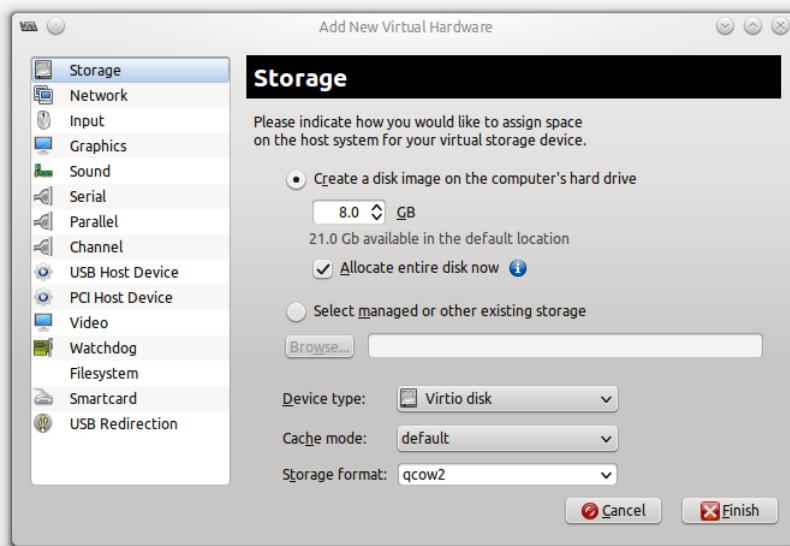


FIGURA 6.8: Creación de las imágenes de disco

A la hora de especificar el tamaño, hay que tener en cuenta que

6.3.1.2. Configuración de la máquina virtual

- la imagen del SO debe tener un tamaño adecuado para lo que queremos instalar en ella. Para un sistema GNU/Linux, basta con 10 GB, pero un sistema Windows requiere un mínimo de 25 GB.
- en GNU/Linux, la partición swap debe tener el mismo tamaño que la RAM. En Windows, se recomienda que el fichero de paginación sea dos veces más grande que la RAM.

Para crear las dos imágenes de disco, debemos seleccionar Virtio disk en Device type y qcow2 en Storage format.

Por otra parte, para configurar la tarjeta de vídeo hacemos clic sobre Video, y en Model seleccionamos vga. Esto se muestra en la figura 6.9.

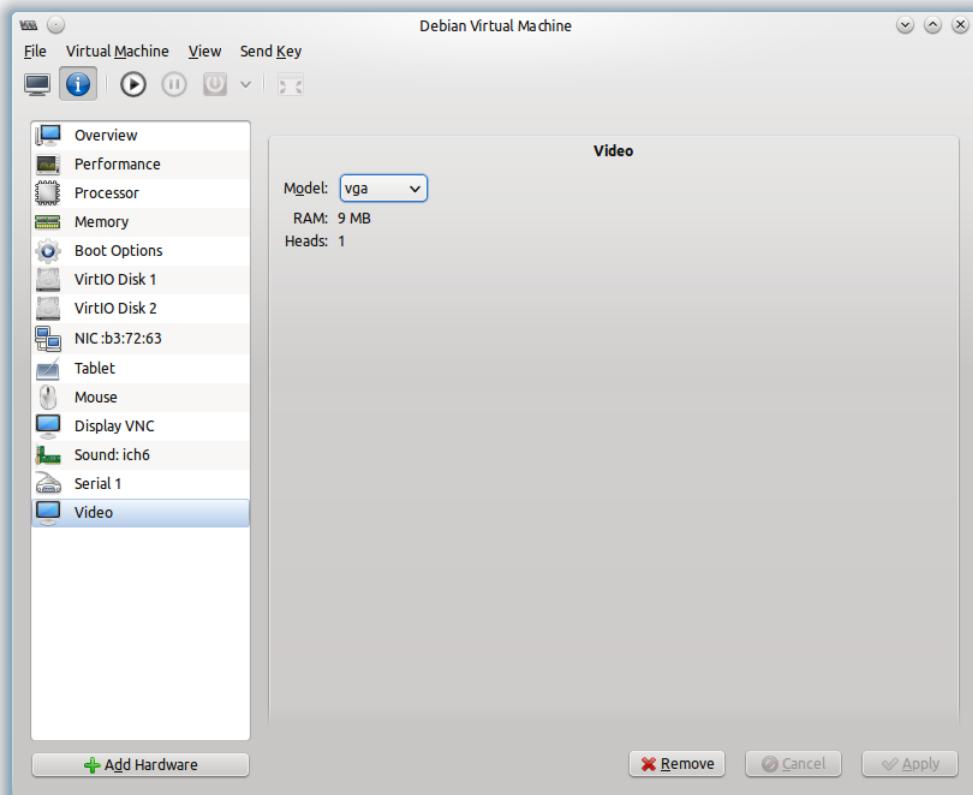


FIGURA 6.9: Configuración de la tarjeta de vídeo

Importante

Debido a un *bug* en el cargador de arranque, todas las distribuciones derivadas de Debian GNU/Linux 6 (*squeeze*) no pueden arrancar con la tarjeta de vídeo *vmvga*. En estos casos, es necesario seleccionar la tarjeta de vídeo *vga*.

Lo último que nos falta es añadir una tableta gráfica para que el servidor VNC siga adecuadamente los movimientos del ratón. Para ello, hacemos clic sobre Add Hardware, seleccionamos Input, seleccionamos EvTouch USB Graphics Tablet y hacemos clic sobre Finish.

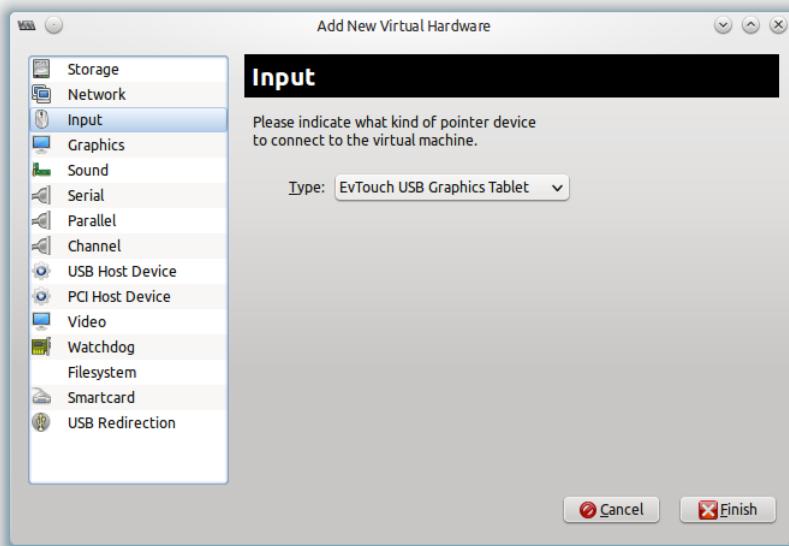


FIGURA 6.10: Configuración de los periféricos de entrada

Para empezar la instalación del sistema operativo, hacemos clic sobre *Begin installation*.

Importante

Para poder instalar un sistema operativo Windows, hay que suministrar al instalador los drivers VirtIO. Mostraremos cómo hacer esto en la siguiente sección.

6.3.1.3. Instalación de los *drivers* en Windows

Para que el instalador pueda acceder a los discos duros virtuales, hay que suministrarle los controladores VirtIO. Los pasos a seguir son estos:

1. Descargamos la imagen ISO correspondiente a la última versión desde <http://alt.fedoraproject.org/pub/alt/virtio-win/latest/images/bin/>.
2. A la hora de configurar la máquina, creamos otra unidad CD-ROM IDE. Para ello, basta con hacer clic sobre *Add hardware*, seleccionar *Storage*
 - a) marcar la casilla *Select managed or other existing storage*, e introducir la ruta de la imagen ISO que hemos descargado.
 - b) seleccionar *IDE cdrom* en *Device type*.

En ocasiones, *virt-manager* selecciona la nueva unidad de CD-ROM (y no la que tiene el medio de instalación de Windows) para arrancar. Si eso ocurre, basta con volver a abrir el diálogo de configuración de la máquina (hacemos clic derecho sobre la máquina en la ventana principal de *virt-manager*, seleccionamos *Open* y accedemos al menú *View > Details* de la nueva ventana que aparece), seleccionar *Boot options*, marcar la casilla *Enable boot menu* y seleccionar la unidad correcta al arrancar la máquina.

6.3.1.3. Instalación de los drivers en Windows

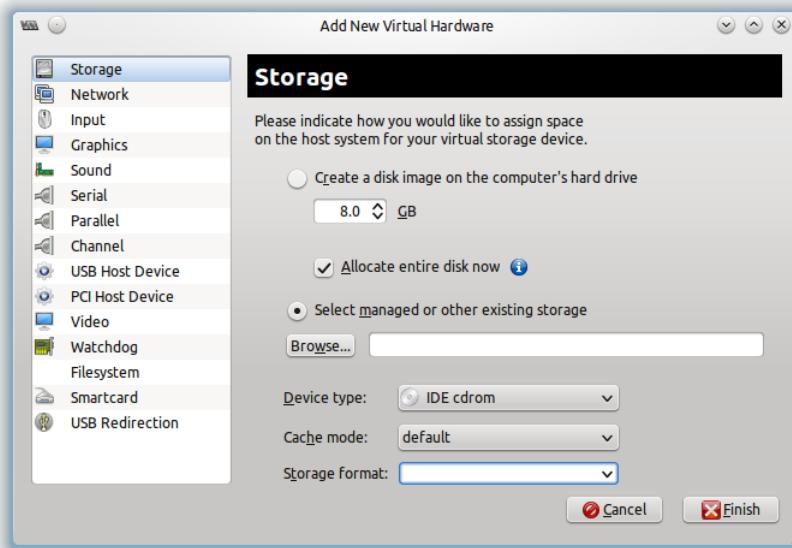


FIGURA 6.11: Configuración del CD-ROM con los drivers VirtIO

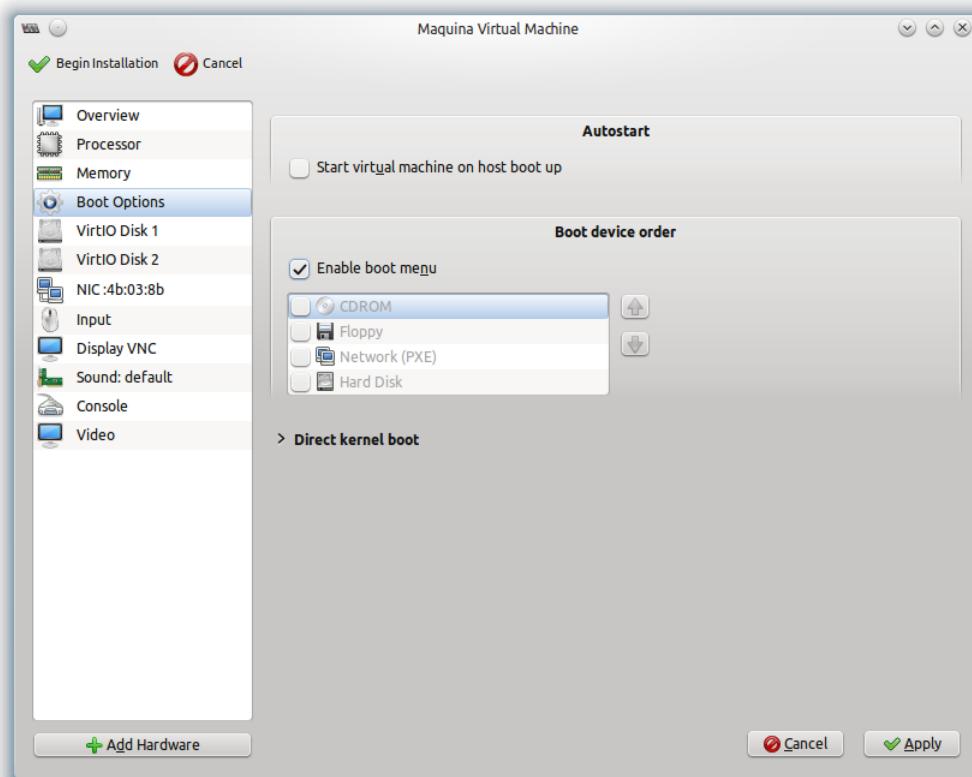


FIGURA 6.12: Corrección de los problemas de arranque en instalaciones Windows

6.3.2. Importar máquinas virtuales existentes a virt-manager

Siempre que una máquina virtual tenga instalado un sistema GNU/Linux y utilice una única imagen de disco podrá utilizarse en *CygnusCloud*. Los pasos a seguir son los siguientes:

1. Convertir las imágenes de disco a un formato soportado por QEMU: raw, vmdk, qcow2 o qcows2. Por ejemplo, en el caso de una imagen de VirtualBox, habría que ejecutar el comando

```
VBoxManage clonehd <fichero fuente>.vdi <fichero destino>.vmdk --format VMDK
```

2. Convertir las imágenes al formato qcows2. Para ello, ejecutaríamos la orden

```
qemu-img convert <fichero fuente>.vmdk -O qcows2 <fichero destino>.qcows2
```

3. En lugar de crear una nueva imagen para el sistema operativo, en el diálogo se importa la existente seleccionando la casilla **Select managed or other existing storage** e indicando su ruta.
4. Creamos una imagen de disco para almacenar los datos del usuario y el fichero de paginación.
5. Movemos el fichero de paginación y los datos del usuario a la imagen correspondiente.

Importante: este procedimiento puede no funcionar en sistemas operativos Windows.

6.3.3. Uso de samba para configurar las máquinas virtuales

Para configurar el entorno de la máquina virtual, no sólo es posible descargar *software* de internet: también es posible, utilizando *samba*, acceder desde la máquina virtual a los recursos de la máquina anfitrión o de cualquier otro PC conectado a la misma red de área local que en la máquina anfitrión. En esta sección mostraremos cómo instalar y configurar samba para que esto sea posible. Distinguiremos cuatro casos

- El PC y la máquina virtual tienen instalado un sistema operativo Windows. Llamaremos a este caso **compartición Windows a Windows**.
- El PC tiene instalado un sistema operativo Windows, y la máquina virtual tiene instalado un sistema operativo Linux. Llamaremos a este caso **compartición Windows a Linux**.
- El PC tiene instalado un sistema operativo Linux, y la máquina virtual tiene instalado un sistema operativo Windows. Llamaremos a este caso **compartición Linux a Windows**.
- El PC tiene instalado un sistema operativo Linux, y la máquina virtual tiene instalado un sistema operativo Linux. Llamaremos a este caso **compartición Linux a Linux**.

Antes de empezar, mostraremos qué hace falta instalar en máquinas Linux (con distros derivadas de Debian o Ubuntu). En Windows, no es necesario instalar nada.

6.3.3.1. Prerrequisitos (sólo en Linux)

Máquina virtual

Si la máquina virtual utiliza un sistema operativo Linux, habrá que instalar todo lo necesario para que se convierta en un cliente Samba. Para ello, basta con ejecutar la orden

```
sudo apt-get install smbclient smbfs
```

Con esto, es suficiente.

PC

Si el PC tiene instalado un sistema operativo Linux, habrá que instalar todo lo necesario para que se convierta en un servidor Samba. Para ello, ejecutaremos la orden

```
sudo apt-get install samba
```

Después, editamos el fichero `/etc/samba/smb.conf`, descomentando las líneas

```
security = user
```

y

```
workgroup = WORKGROUP
```

Para terminar, definimos una contraseña para nuestro usuario. Samba no nos obliga a utilizar nuestro propio usuario: podemos crear a propósito un usuario para samba. De todas formas, lo más sencillo es utilizar el que ya tenemos. El comando que tenemos que escribir es

```
sudo smbpasswd -a <nuestro nombre de usuario>
```

Esta orden nos pedirá la nueva contraseña.

Importante: si omitimos este paso, no será posible montar ninguna carpeta compartida. Este error es muy difícil de ver.

6.3.3.2. Compartición de Windows a Windows

Configuración del PC

Para compartir una carpeta en Windows, basta con seguir estos pasos:

1. Hacemos clic derecho sobre la carpeta que queremos compartir y seleccionamos Propiedades.
2. Vamos a la pestaña Compartir y hacemos clic sobre el botón Uso compartido avanzado
3. Marcamos la casilla Compartir esta carpeta. Por defecto, sólo se permite leer de ella. En la parte superior del diálogo, aparecerá un nombre de la forma

`\nombre-del-pc\nombre-del-recurso`

Recordad el nombre del recurso: os hará falta más adelante.

4. Si queremos escribir en la carpeta desde otra máquina, basta con hacer clic sobre Permisos y marcar la casilla Control total.

Configuración de la máquina virtual

Para conectarnos a la carpeta compartida desde la máquina virtual, seguimos estos pasos:

1. Hacemos clic sobre Equipo, y seleccionamos Conectar a unidad de red.
2. En el diálogo que aparece, metemos este nombre para la máquina:

`\ip-del-pc\nombre-del-recurso`

3. Confirmamos los cambios. A partir de este momento, la carpeta compartida aparecerá montada en Equipo como una unidad de red.

6.3.3.3. Compartición de Linux a Linux

Configuración del PC

Para compartir la carpeta, seguimos los siguientes pasos:

1. Añadimos una entrada como la siguiente *al final* del fichero `/etc/samba/smb.conf`:

```
[nombre-del-recurso]
comment = <comentario>
read only = yes | no
guest ok = yes | no
browsable = yes
force user = <nuestro usuario>
path = <ruta de la carpeta compartida>
```

donde

- `nombre-del-recurso` es el nombre que queremos asignar a la carpeta compartida (como `home`, `mis_datos`,...). Por seguridad, *no* metáis aquí espacios ni caracteres especiales.
- `comment` contiene una descripción de la carpeta compartida. Si está vacío, no aparece nada a la derecha del símbolo `=`.
- `read only` fija los permisos de lectura y escritura. Cuando toma el valor `yes`, no se podrá escribir en la carpeta desde la máquina virtual. Cuando toma el valor `no`, sí.
- `guest ok` determina el comportamiento para los usuarios no autenticados. Cuando toma el valor `yes`, cualquiera podrá leer de la carpeta. Cuando toma el valor `no`, sólo los usuarios que se autentiquen podrán leer la carpeta.

2. Reiniciamos el servidor Samba. Para ello, ejecutamos la orden

```
sudo /etc/init.d/smbd restart
```

Configuración de la máquina virtual

Para montar la carpeta compartida en la máquina virtual, seguimos los siguientes pasos:

1. Creamos el punto de montaje. Por ejemplo, si queremos montar la carpeta en `/mnt/smbshare`, ejecutaremos el comando

```
sudo mkdir /mnt/smbshare
```

2. Montamos la carpeta utilizando el comando

```
sudo mount -t cifs -o username=<nombre de usuario>,password=<contraseña>
//ip-del-pc/nombre-del-recurso <punto de montaje>
```

3. Para desconectarnos de la carpeta compartida, basta con ejecutar el comando

```
sudo umount --force <punto de montaje>
```

6.3.3.4. Compartición de Windows a Linux

Para configurar el PC, seguimos las instrucciones de la sección 6.3.3.2. Para configurar la máquina virtual, seguimos las instrucciones de la sección 6.3.3.3.

6.3.3.5. Compartición de Linux a Windows

Para configurar el PC, seguimos las instrucciones de la sección 6.3.3.3. Para configurar la máquina virtual, seguimos las instrucciones de la sección 6.3.3.2.

6.3.4. Importar máquinas virtuales a *CygnusCloud*

Una vez preparadas las máquinas virtuales, tenemos que borrar las unidades de CD-ROM que hemos utilizado para realizar la instalación del sistema operativo, y también la tarjeta de sonido. Los pasos a seguir son los siguientes:

1. accedemos al diálogo de configuración de la máquina. Para ello, hacemos clic derecho sobre la máquina en la ventana principal de *virt-manager*, seleccionamos Open y accedemos al menú View > Details.
2. hacemos clic derecho sobre las unidades de CD-ROM y seleccionamos Remove hardware.
3. hacemos lo mismo con la tarjeta de sonido, que aparece en la sección sound: ich6.

Una vez hecho esto,

1. copiamos el fichero de definición de la máquina, ubicado en /etc/libvirt/qemu, al directorio configFilePath que especificamos en el fichero de configuración.
 2. copiamos las imágenes de disco de la máquina al directorio sourceImagePath que especificamos en el fichero de configuración.
 3. registramos la máquina virtual en el servidor de máquinas virtuales. Para ello, editamos el fichero bin/database/VMServerDB.sql, añadiendo al final las líneas
- ```
INSERT IGNORE INTO VirtualMachine VALUES (<id de la máquina>, '<Nombre de la máquina>', '<ruta imagen datos>', '<ruta imagen SO>', '<ruta fichero definición>');
```

Las tres rutas son relativas. Por ejemplo, si

- sourceImagePath es /home/luis
- la imagen del sistema operativo se llama OS.qcow2, y está en /home/luis/Debian
- la imagen con los datos del usuario y el fichero de paginación se llama Data.qcow2 y está en /home/luis/Debian
- configFilePath es /home/luis
- el fichero de definición se llama debian.xml y está en /home/luis/Debian
- hay ya dos máquinas virtuales, con IDs 1 y 2
- la máquina se llamará Debian

la línea a añadir sería

```
INSERT IGNORE INTO VirtualMachine VALUES (3, 'Debian', 'Debian/Data.qcow2', 'OS.qcow2', 'Debian/Debian.xml');
```

4. registramos la máquina virtual en el servidor de *cluster*. Asumiendo que la colocaremos en el servidor 1, editamos el fichero bin/database/VMServerDB.sql, añadiendo la siguiente línea:

```
INSERT IGNORE INTO ImageOnServer VALUES (1, 3);
```

5. reiniciamos la infraestructura. A partir de este momento, la nueva máquina podrá utilizarse.

**Importante:** este procedimiento es sólo provisional, y se automatizará completamente en la versión 2.0.

## 6.4. Uso de la página web

### 6.4.1. Instalación y uso del framework web2py

El servidor web se encuentra desarrollado utilizando el framework web2py. Para poder utilizar este *framework* no es necesario realizar ningún tipo de instalación adicional a la indicada en la sección [6.2.4](#). El fichero `.tar.gz` ya incluye la última versión estable de web2py (2.4.5) y la aplicación web de *CygnusCloud* integrada. No obstante, si se quiere descargar una versión diferente será necesario seguir los siguientes pasos:

1. Nos conectamos a <http://www.web2py.com/init/default/download> y descargamos la versión deseada. Es posible descargar versiones para Linux, Windows o Mac OS, aunque para *CygnusCloud* sólo puede usarse la versión de Linux. También es posible elegir tres posibles tipos de descargas de cara a el uso de web2py que vayamos a hacer. En nuestro caso, usaremos la opción For Normal Users.
2. descomprimimos el fichero descargado y lo copiamos al directorio de instalación que queramos.
3. copiamos el directorio `web2py/applications/CygnusCloud` del *tarball* que aportamos al directorio de instalación de web2py.
4. copiamos los directorios `bin` y `certificates`, el script `build.py` y los ficheros de configuración al directorio donde se haya descomprimido la versión de web2py (que contendrá un directorio con el nombre web2py).
5. realizamos la instalación tal y como se menciona en la sección [6.2.4](#).

**Importante**

el *tarball* que hemos publicado ya incluye una versión de web2py estable y configurada. Sólo tendrán que seguir este procedimiento aquellos usuarios que quieran utilizar una versión diferente del *framework* web2py.

Web2py ofrece una interfaz para el desarrollo de aplicaciones web accesible desde cualquier navegador. Tras realizar la instalación y ejecutar el script `webServer.sh` podremos utilizarla siguiendo estos pasos:

1. introducimos la URL <http://127.0.0.1:8000/> en un navegador web.
2. Se cargará una página con el título Welcome. Hacemos clic sobre el botón Administrative Interface, situado a la derecha de esta ventana.
3. En la siguiente página, escribiremos la contraseña de administrador (`cygnuscloud`) y pulsaremos el botón de inicio de sesión.
4. Llegados a este punto nos aparece un escritorio como el que se muestra en [6.1](#), con el nombre de las aplicaciones que podemos gestionar desde web2py. Sobre *CygnusCloud*, seleccionamos la opción Manage > editar.
5. se cargará una página con el título Editar aplicación “*CygnusCloud*”. En esta página podemos ver todo el código fuente de la web de *CygnusCloud* organizado en secciones según su finalidad: modelos, controladores, vistas, lenguajes, ficheros estáticos, módulos secundarios, ficheros privados y *plugins*.
6. A la izquierda de cada fichero hay un botón editar que nos permite modificar el fichero de código correspondiente. Igualmente si queremos visualizar el contenido del fichero sin editararlo podemos pulsar directamente sobre su nombre.

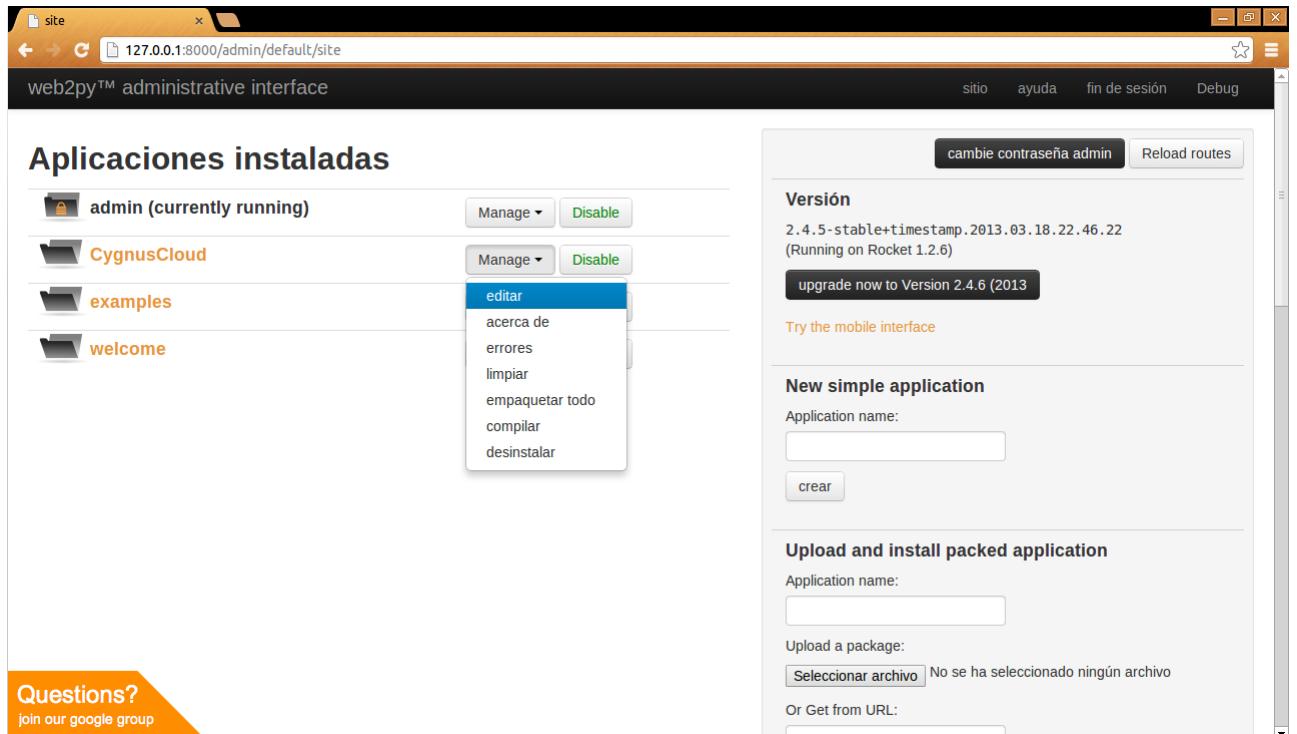


FIGURA 6.1: Interfaz administrativa de web2py

### 6.4.2. Uso de la web

#### 6.4.2.1. Acceso

Tras instalar el *tarball* y ejecutar los *scripts* de arranque del servidor web y del endpoint, podemos acceder a la página de inicio de sesión de *CygnusCloud* introduciendo la URL <http://127.0.0.1:8000/CygnusCloud/main/login> en cualquier navegador web. La figura 6.2 muestra una captura de pantalla de esta página.

En ella, podemos introducir el correo electrónico y la contraseña del usuario que va a iniciar sesión. También podemos acceder desde el menú superior a la página Acerca de con información general del proyecto.

Por defecto, existen dos usuarios de prueba registrados en el sistema:

- un estudiante con correo electrónico Student1@ucm.es y contraseña 1234, y
- un administrador con correo electrónico Admin1@ucm.es y contraseña 1234.

Independientemente del tipo de usuario que ha iniciado sesión, en la parte superior derecha aparece un vínculo Salir que nos permite cerrar la sesión actual y regresar a la página de inicio de sesión.

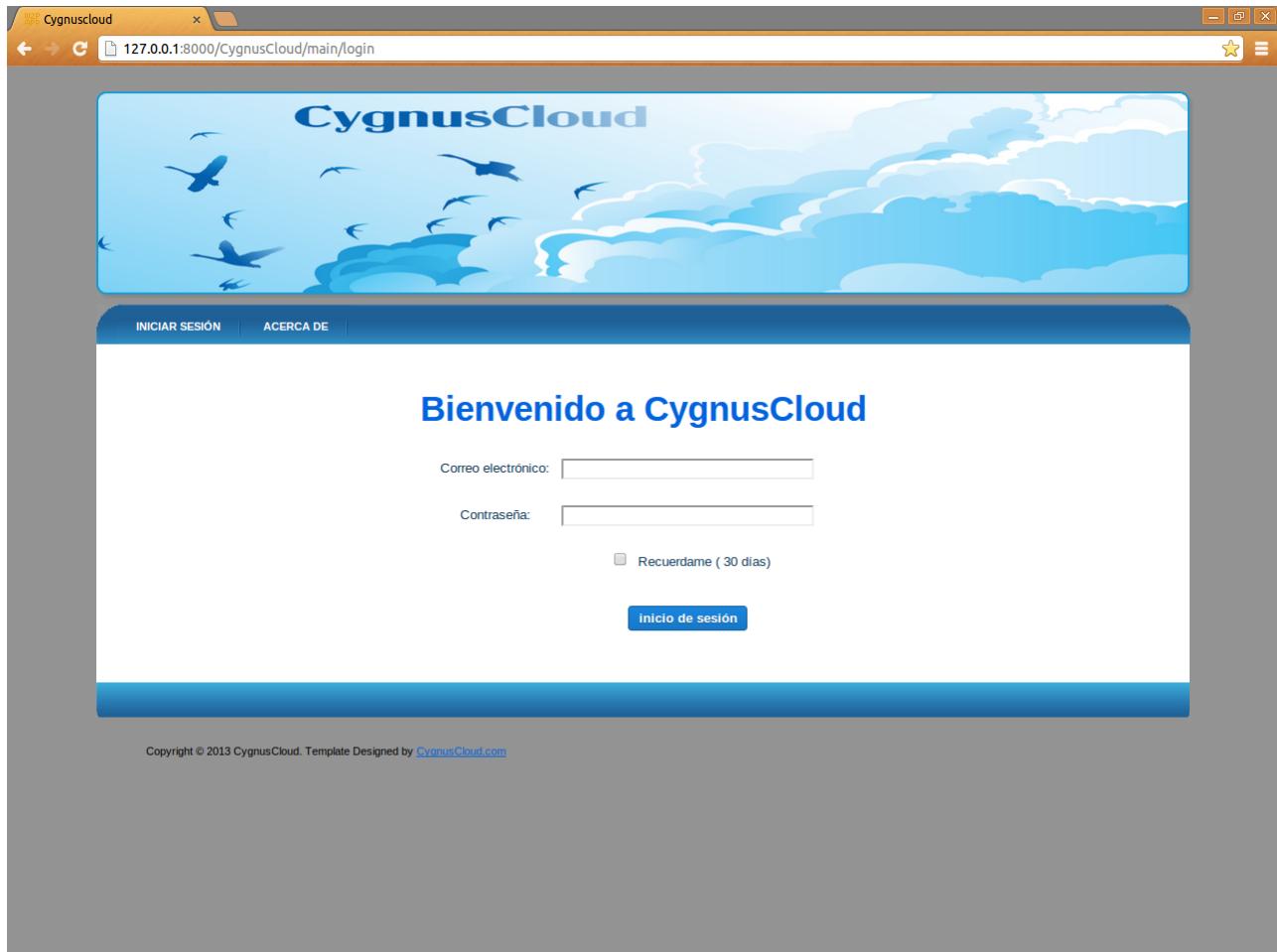


FIGURA 6.2: Página de inicio de *CygnusCloud*



FIGURA 6.3: Página de arranque de máquinas virtuales para estudiantes

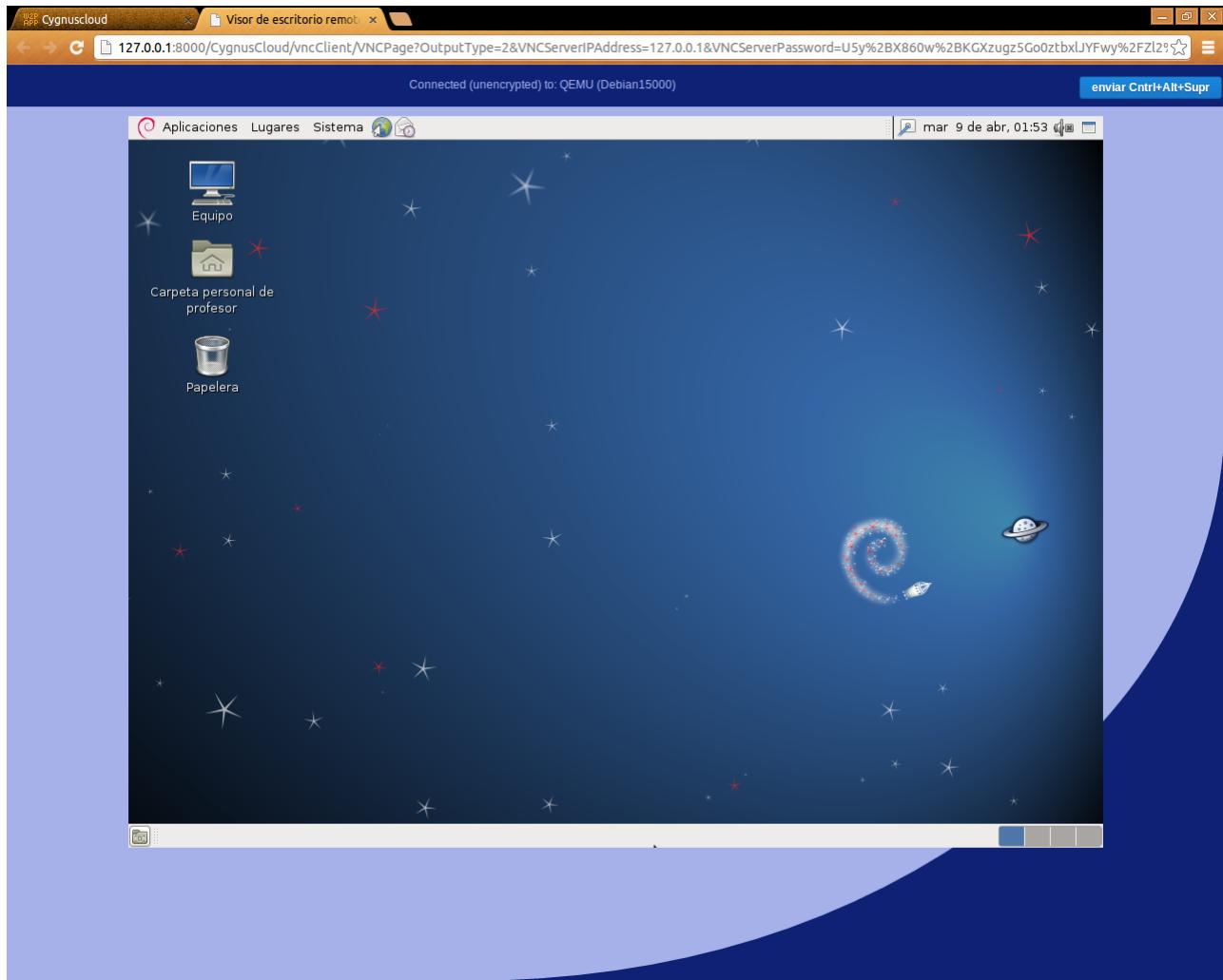


FIGURA 6.4: Visor de escritorio remoto para una máquina virtual arrancada

### 6.4.2.2. Opciones disponibles para los estudiantes

Tras iniciar sesión como un estudiante, la web nos muestra una página como la de la figura 6.3, en la que aparecen con las asignaturas matriculadas y la lista de máquinas virtuales disponibles para cada asignatura.

Para cada máquina virtual, se muestra una casilla de selección que nos permite ver la descripción de la máquina virtual y habilitar su arranque.

Si se hace clic sobre el botón **Arrancar** de alguna máquina virtual, se abrirá una nueva pestaña como la que se muestra en la figura 6.4. Esa pestaña permite trabajar con la máquina mediante el protocolo de escritorio remoto VNC.

#### Importante

- para poder arrancar una máquina virtual, un administrador debe arrancar previamente al menos uno de los servidores de máquinas virtuales en los que se encuentra.
- para que el rendimiento del visor de escritorio remoto sea adecuado, es imprescindible utilizar la última versión de *Internet Explorer*, *Safari*, *Mozilla Firefox* o *Google Chrome*.

##### 6.4.2.3. Opciones disponibles para los administradores

Al acceder como administrador, la web nos muestra una página de arranque de máquinas virtuales bastante similar a la anterior pero con un cuadro de búsqueda adicional. Existen varios cuadros de búsquedas distribuidos a lo largo de las páginas del administrador que nos permiten filtrar la información que debe mostrarse por pantalla. Esto resulta útil ya que el administrador tendrá acceso a absolutamente toda la información gestionada por la web.

En la parte superior de las páginas del administrador hay una barra de menú que nos permite acceder a las diferentes secciones dentro de su alcance. Si desplazamos el puntero del ratón por dicha barra surgen los diferentes submenús con las páginas disponibles. Explicamos brevemente el modo de uso de cada página

- La página Máquinas virtuales > Arrancar permite buscar y arrancar las diferentes máquinas virtuales registradas en algún servidor. Para ello, podemos introducir el nombre, o parte del mismo, de alguna de las asignaturas registradas en el sistema y pulsar el botón Buscar. Una vez hecho esto, aparece la lista de máquinas virtuales asociadas a dicho criterio de búsqueda.
- La página Máquinas virtuales > Detener nos permite buscar una máquina virtual de forma similar al caso anterior y darla de baja.
- La página Administrar servidores > Añadir nos permite añadir un nuevo servidor de máquinas virtuales al sistema, indicando el nombre, la ip y el puerto asociado a dicho servidor.
- La página Administrar servidores > Eliminar servidor nos permite análogamente dar de baja un servidor registrado o arrancarlo. Para ello cuenta con un menú desplegable donde se encuentran el nombre de todos los servidores registrados hasta el momento. Una vez seleccionado el servidor deseado y pulsado el botón Buscar servidor aparecerá la información de dicho servidor junto a los botones correspondientes.
- La página Administrar usuarios > Eliminar permite eliminar un usuario registrado en la web. Para ello, existe un cuadro de búsqueda que nos permite indicar parte del correo electrónico del usuario/s a buscar, así como el tipo de usuario del que se trata.
- La página Administrar usuarios > Añadir permite añadir una nuevo usuario indicando su correo electrónico, su contraseña y el tipo de usuario.
- La página Administrar usuarios > Asociar asignaturas , nos permite buscar los usuarios registrados en la web por medio del cuadro de búsqueda. Una vez realizada la búsqueda, el cuadro para añadir usuarios nos permite seleccionar alguno de los usuarios encontrados en la búsqueda y asociarle una determinada asignatura registrada en el sistema.
- La página Administrar asignaturas > Añadir nos permite añadir una nueva asignatura indicando su código, su nombre, el año en el que se cursa, el curso y el grupo de clase asociado a dicha asignatura.
- La página Administrar asignaturas > Eliminar dentro del menú nos permite buscar las asignaturas que cumplen los criterios de código, nombre, año o grupo introducidos. Una vez mostradas las posibles asignaturas, el sistema nos permite seleccionar y eliminar la que deseemos.



## Apéndices



## Apéndice A

# Licencia

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria como el código, los contenidos audiovisuales incluso si incluyen imágenes de los autores, la documentación y/o el prototipo desarrollado.

En el resto de casos, se aplicarán a cada tipo de contenido los términos que aparecen a continuación.

### A.1. Código fuente

#### A.1.1. Versión modificada de noVNC

Copyright © 2012, Joel Martin (<https://github.com/kanaka>)

Copyright © 2013, Luis Barrios Hernández, Adrián Fernández Hernández, Samuel Guayerbas Martín

This Source Code Form is subject to the terms of the Mozilla Public License, version 2.0. If a copy of the MPL was not distributed with this file, you can obtain one at

<http://mozilla.org/MPL/2.0/>

#### A.1.2. Resto de código fuente de *CygnusCloud*

Copyright © 2013, Luis Barrios Hernández, Adrián Fernández Hernández, Samuel Guayerbas Martín

The source code is licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

### A.2. Documentación

Toda la documentación del proyecto se distribuye bajo la siguiente licencia siempre que no se especifique lo contrario:



**Usted es libre de:**

- copiar, distribuir y comunicar públicamente la obra
- crear obras derivadas

**Bajo las condiciones siguientes:**

**Reconocimiento.** Debe reconocer los créditos de la obra de la manera especificada por el autor o el licenciador (pero no de una manera que sugiera que tiene su apoyo o apoyan el uso que hace de su obra).

**No comercial.** No puede utilizar esta obra para fines comerciales.

**Compartir bajo la misma licencia.** Si altera o transforma esta obra, o genera una obra derivada, sólo puede distribuir la obra generada bajo una licencia idéntica a ésta.

**Para obtener más información,** visite

[http://creativecommons.org/licenses/by-nc-sa/3.0/deed.es\\_ES](http://creativecommons.org/licenses/by-nc-sa/3.0/deed.es_ES)

## Apéndice B

# Git: guía de referencia

### B.1. Git y Subversion

Hasta ahora, estamos acostumbrados a trabajar con algo como lo de la figura B.1. Existe un único repositorio central, al que se van haciendo *commits* cuando toca. Lo más normal es trabajar en un directorio separado, pegar los ficheros modificados en el directorio donde está la copia del repositorio y hacer el *commit*.



FIGURA B.1: Subversion

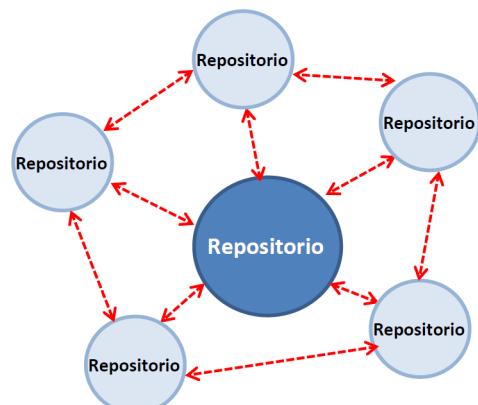


FIGURA B.2: Git

Con Git la situación es bastante diferente, tal y como podéis ver en la figura B.2. Git es un sistema de control de versiones distribuido, por lo que no existe el gran repositorio central: todos los repositorios tienen la misma importancia.

La forma de trabajar en Git es bastante distinta: cada desarrollador tendrá su repositorio local, al que irá haciendo *commits* cuando quiera. Es posible tomar cosas de los repositorios de otros (repositorios remotos), y también es posible subir cosas a esos repositorios. Así, el repositorio de Google Code será un repositorio más.

## B.2. Instalación y configuración básica de Git

Para empezar a utilizar Git, es obvio que hay que tener el binario instalado en nuestro sistema. Para ello, abrid un terminal y escribid

```
$ sudo apt-get install git gitk
```

Una vez hecho esto, vamos a fijar algunas cosas importantes (como el nombre de usuario) y otras que no lo son tanto, pero que facilitan el uso de Git.

Empezaremos dejando de lado el *Editor de la Bestia* y utilizando otro algo más agradable. Para ello, añadid la línea

```
export EDITOR=nano
```

al final del fichero `~/.bashrc`.

Lo siguiente que haremos es introducir nuestro nombre de usuario y dirección de correo electrónico. Esto no hace falta para conectarse a Google Code: simplemente se utilizará para firmar los *commits*. Para ello, abrid un terminal y escribid

```
$ git config --global user.name "<Nombre de usuario>"
$ git config --global user.email "<Correo electrónico>"
```

El *flag* `--global` indica que utilizaremos estos ajustes en *todos* los repositorios. Si queréis utilizarlos solamente en algunos, id al directorio del repositorio y ejecutad estas órdenes sin el *flag* `--global`.

Finalmente, cambiaremos un poco los colores predeterminados para facilitar la lectura de los mensajes de Git. Para ello, añadid este texto al fichero `~/.gitconfig`.

```
[color]
 ui = true
[color "branch"]
 current = yellow reverse
 local = yellow
 remote = green
[color "diff"]
 meta = yellow bold
 frag = magenta bold
 old = red bold
 new = green bold
 whitespace = red reverse
[color "status"]
 added = yellow
 changed = green
 untracked = cyan
[core]
 whitespace=fix,-indent-with-non-tab,trailing-space,cr-at-eol
```

### B.3. Uso del repositorio de Google Code

Como ya sabréis, los repositorios de Google Code son públicos, por lo que cualquiera puede leer de ellos. Empezaremos obteniendo una copia local de nuestro repositorio. Para ello, ejecutad la orden

```
$ git clone https://code.google.com/p/cygnus-cloud/
```

En el directorio de trabajo aparecerá el directorio `cygnus-cloud` con todo el contenido del repositorio.

Pero aún no hemos terminado: para hacer un *commit* es necesario autenticarse. Para ello, añadid la línea

```
machine code.google.com login <correo gmail> password <contraseña google code>
```

al fichero `~/.netrc`. Con esto, ya podréis utilizar el repositorio de Google Code.

### B.4. Uso básico de Git

#### B.4.1. Registrar cambios en el repositorio

Ahora que tenemos un repositorio, ha llegado el momento de utilizarlo. Cada uno de los ficheros del directorio de trabajo puede estar en dos estados:

- *tracked* (registrados). Se trata de ficheros que ya estaban en el repositorio. A su vez, pueden estar en tres estados: *unmodified* (actualizados en el repositorio), *modified* (sin actualizar en el repositorio) o *staged* (ficheros modificados *que hay que subir* al repositorio).
- *untracked* (no registrados). En este grupo están el resto de ficheros.

El ciclo de vida de un fichero en Git aparece en la figura B.1.

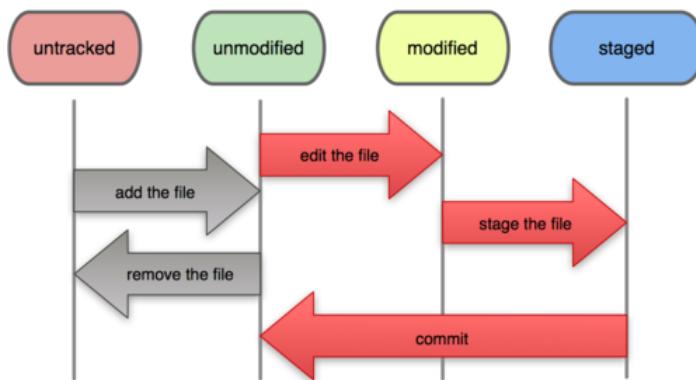


FIGURA B.1: Ciclo de vida de un fichero en Git

Cuando se clona el repositorio, todas los ficheros estarán en estado *tracked* y *unmodified* (aún no se ha hecho nada con ellos). A medida que se editan, pasan a estar en estado *modified*. Finalmente, pasan a estado *staged* y se hace un *commit* para reflejar los cambios en el repositorio.

Todos los ficheros que se subirán al repositorio al hacer un *commit* se encuentran en la *staging area*.

#### B.4.1.1. Comprobando el estado de los ficheros

Para determinar el estado de los ficheros se utiliza la orden `git status`. Justo después de clonar el repositorio, su salida es

```
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
nothing to commit (working directory clean)
```

Como era de esperar, no ha habido cambios con respecto al repositorio. Es más, no existe ningún fichero nuevo, ya que en ese caso habría aparecido en la salida. Además, el comando nos indica que estamos utilizando la rama (*branch*) `master`, que es la rama por defecto. Si añadimos algo, como el fichero `README.txt`, la salida del comando `git status` ya es diferente.

```
luis@luis-laptop:~/cygnus-cloud$ touch README.txt
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
#
README.txt
nothing added to commit but untracked files present (use "git add" to track)
```

#### B.4.1.2. Incluyendo nuevos ficheros

Para evitar errores, Git *no* incluirá ficheros en estado *untracked* a no ser que se le pida explícitamente. Para ello, se utiliza la orden `git add`.

```
luis@luis-laptop:~/cygnus-cloud$ git add README.txt
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
#
new file: README.txt
#
```

Al ejecutar nuevamente la orden `git status`, podemos comprobar que el fichero `README.txt` está en estado *tracked* y *staged*.

En general, la sintaxis de la orden `git add` es

```
git add <ruta>
```

Si `<ruta>` es un directorio, se añaden recursivamente todos los ficheros. Si es un fichero, se añade únicamente ese fichero.

#### B.4.1.3. Modificando ficheros ya existentes

Supongamos que, tras hacer el *commit*, modificamos el fichero `README.txt`. Tras ejecutar la orden `git status`, obtendríamos algo como esto:

```
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
#
modified: README.txt
#
```

¿Qué ocurre? El fichero README.txt ha sido modificado. Para indicar que queremos que los cambios sean permanentes, hay que conseguir que el fichero pase a estado *staged*. ¿Cómo? Volviendo a utilizar la orden `git add`. Ahora, la salida será

```
luis@luis-laptop:~/cygnus-cloud$ git add README.txt
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
#
modified: README.txt
#
```

Supongamos que introducimos una modificación de última hora. Si volvemos a ejecutar la orden `git status`, obtenemos algo como

```
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
#
modified: README.txt
#
```

y volvemos a estar como al principio. ¿Por qué? Porque los datos que se añadirán al repositorio al hacer el *commit* se fijan en el momento en que se ejecuta la orden `git add`. Si tras ejecutar esta orden se realiza otra modificación, será necesario volver a ejecutar el comando `git add` antes de hacer el *commit* si queremos que los nuevos cambios se reflejen en el repositorio.

#### B.4.1.4. Ignorando ficheros y directorios

Existen ficheros irrelevantes que no tienen que estar en el repositorio. Es el caso de los ejecutables, ficheros temporales, *logs*, ficheros auxiliares...

Para excluirlos, basta con crear el fichero `.gitignore` en el directorio de trabajo y escribir en él unas cuantas expresiones regulares. El contenido de uno de estos ficheros podría ser este:

```
luis@luis-laptop:~/cygnus-cloud$ cat .gitignore
*~
*.class
*. [oa]
```

Con esto, estamos ignorando los ficheros temporales, los ficheros `.class` (*bytecode* de la JVM) y los ficheros de código objeto (con extensión `.o` o `.a`).

Las reglas para generar estos ficheros son las siguientes:

- Las líneas en blanco y las que empiezan por almohadilla (#) son ignoradas.
- Las expresiones regulares deben ser patrones glob. Tenéis más información [aquí](#).
- Los patrones que terminan por / hacen referencia a directorios.
- Los patrones que comienzan por admiración (!) están negados.

#### B.4.1.5. Nuestro primer *commit*

Una vez que tenemos claro qué queremos subir al repositorio, ha llegado el momento de hacer nuestro primer *commit*. Es importante notar que todo lo que esté en estado *unstaged* (cambios no confirmados) *no* se subirá al repositorio.

Para hacer el *commit*, basta con ejecutar la orden `git commit`. Al hacerlo, se abrirá el editor que fijamos en la sección B.2. Bastará con escribir un mensaje, guardar el fichero y cerrar el editor. La salida de la orden será algo como

```
luis@luis-laptop:~/cygnus-cloud$ git commit
[master c6733d0] + Subo Readme.txt
1 file changed, 1 insertion(+), 1 deletion(-)
```

Nos está indicando que hemos añadido nuestro fichero a la rama `master`, la suma SHA-1 del *commit*, el número de ficheros modificados y también el número de líneas de código añadidas y eliminadas.

También es posible hacer un *commit* usando el flag `-m "Mensaje"`.

Ahora, al examinar el contenido del repositorio remoto en Google Code, os daréis cuenta de que... ¡no habrá pasado nada! Con lo que ya sabéis, deberíais ser capaces de explicar qué ha pasado. De todas formas, volveremos a discutir esto más adelante.

#### B.4.1.6. *Commits* “rápidos”

Aunque es muy útil saber exactamente qué vamos a subir al repositorio, a veces lo único que hace falta es subir todo lo que haya en el directorio de trabajo y que no deba ignorarse.

Es posible saltarnos el uso de `git add` y hacer directamente el *commit* con todo proporcionándole el flag `-a` al comando `git commit`.

#### B.4.1.7. Borrando ficheros del repositorio

Para eliminar un fichero del repositorio, basta con utilizar la orden `git rm`. Esta orden también eliminará el fichero del directorio de trabajo.

Por seguridad, si el fichero vuelve a crearse y a registrarse, no se borrará al hacer un *commit*. Para forzar el borrado en estos casos, es necesario proporcionar el flag `-f` a la orden `git rm`.

Por otra parte, resulta interesante eliminar un fichero del repositorio y mantenerlo en el directorio de trabajo. Para conseguir esto, basta con utilizar la orden

```
$ git rm --cached <fichero>
```

La orden `git rm` puede recibir como argumento rutas de ficheros, directorios y patrones glob que hagan referencia a ficheros. En caso de utilizar patrones glob, es importante notar que todos los asteriscos deben ir precedidos por \ (para evitar la expansión de nombres que ya utiliza Git).

#### B.4.1.8. Moviendo y renombrando ficheros

En principio, si se renombra uno de los ficheros del repositorio, no habrá metadatos que se lo indiquen a Git. Para evitar problemas, es preferible utilizar la orden `git mv`. Su sintaxis es

```
$ git mv <ruta fichero fuente> <ruta fichero destino>
```

Naturalmente, esto también puede hacerse a mano. La orden anterior equivale a

```
$ mv <ruta fichero fuente> <ruta fichero destino>
$ git rm <ruta fichero fuente>
$ git add <ruta fichero destino>
```

### B.4.2. Revisar cambios en el repositorio

Ahora que hemos hecho nuestros primeros *commits*, ha llegado el momento de ver qué hemos hecho. Para ello, utilizaremos la orden `git log`. Al ejecutarla en nuestro repositorio, obtenemos esta salida:

```
luis@luis-laptop:~/cygnus-cloud$ git log
commit bdd9360b437b8a38cf866879f731ae568309aefd
Author: Luis Barrios <luisbarrioshdez@gmail.com>
Date: Fri Sep 14 19:49:57 2012 +0200
 Plantillas de la documentación revisadas
```

Como podéis ver, aparecen las *checksums* SHA-1, el nombre y el correo del autor, la fecha y el mensaje de cada *commit*.

Evidentemente, es posible alterar este comportamiento cambiando los argumentos de la orden. Algunos muy útiles aparecen en el cuadro B.4.1.

| Flag                                                       | Descripción de la salida                                                                                                                                                       |
|------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>-p</code>                                            | Muestra los cambios introducidos en los <i>commits</i> (vía <code>diff</code> )                                                                                                |
| <code>--stat</code>                                        | Muestra abreviadamente los cambios introducidos en los <i>commits</i>                                                                                                          |
| <code>-n</code>                                            | Restringe la salida a los últimos <i>n</i> <i>commits</i>                                                                                                                      |
| <code>--relative-date</code>                               | Usa fechas relativas (i.e. 10 days ago)                                                                                                                                        |
| <code>--since=&lt;fecha&gt;, --after=&lt;fecha&gt;</code>  | Muestra los <i>commits</i> realizados desde la fecha <code>&lt;fecha&gt;</code> .<br>Las fechas pueden ser específicas ("2012-09-14")<br>o relativas (2.years.1.week.10.hours) |
| <code>--until=&lt;fecha&gt;, --before=&lt;fecha&gt;</code> | Muestra los <i>commits</i> realizados hasta la fecha <code>&lt;fecha&gt;</code> .<br>Las fechas pueden ser específicas ("2012-09-14")<br>o relativas (3.weeks.10.hours)        |
| <code>--author</code>                                      | Muestra los <i>commits</i> con la entrada <code>author</code> especificada.                                                                                                    |
| <code>--committer</code>                                   | Muestra los <i>commits</i> con la entrada <code>committer</code> especificada.                                                                                                 |

CUADRO B.4.1: Argumentos muy útiles de `git log`

#### B.4.2.1. Uso de `gitk`

También es posible examinar los cambios que se han producido en el repositorio en una interfaz gráfica usando la orden `gitk`, que admite prácticamente las mismas opciones que `git log`. La figura B.2 recoge una captura de pantalla.

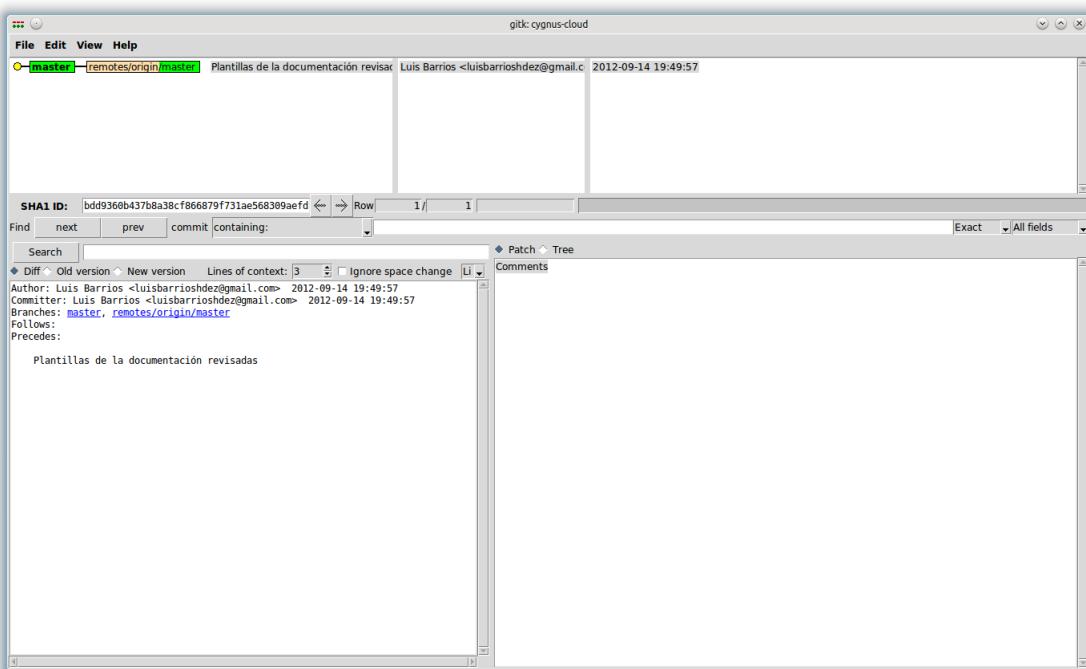


FIGURA B.2: La GUI gitk

### B.4.3. Deshacer cambios

**Importante:** no todos los procedimientos que se describen en esta sección son reversibles. ¡Tened cuidado con lo que hacéis!

#### B.4.3.1. Cambiar el último *commit*

Como ya sabréis, no todos los ficheros que se encuentran en el directorio de trabajo se subirán al repositorio al hacer un *commit*: sólo se subirán aquellos ficheros que se encuentren en estado *staged*. Se dice que todos estos ficheros forman parte de la *staging area*.

Para añadir más ficheros al último *commit* o cambiar el mensaje, se utiliza la orden

```
git commit --amend
```

#### B.4.3.2. Eliminar un fichero en estado *staged*

Para dejar las cosas claras, utilizaré un ejemplo. Empezaremos añadiendo el fichero `basura.txt` al directorio de trabajo.

```
luis@luis-laptop:~/cygnus-cloud$ touch basura.txt
```

Ahora, añadiremos a lo loco todos los ficheros a la *staging area*.

```
luis@luis-laptop:~/cygnus-cloud$ git add .
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Changes to be committed:
(use "git reset HEAD <file>..." to unstage)
#
new file: basura.txt
#
```

Como podréis suponer, no queremos incluir el fichero `basura.txt` en nuestro *commit*. Para eliminarlo de la *staging area*, la propia orden `git status` nos está indicando qué hacer.

```
luis@luis-laptop:~/cygnus-cloud$ git reset HEAD basura.txt
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Untracked files:
(use "git add <file>..." to include in what will be committed)
#
basura.txt
nothing added to commit but untracked files present (use "git add" to track)
```

#### B.4.3.3. Deshacer los cambios en un fichero

Aquí también utilizaré un ejemplo. Empezaremos escribiendo algo en nuestro fichero `en_blanco.txt`.

```
luis@luis-laptop:~/cygnus-cloud$ echo "foo" > en_blanco.txt
```

Como podréis suponer, queremos dejar el fichero `en_blanco.txt` vacío, tal y como estaba en nuestro último *commit*. La orden `git status` también nos indica cómo hacerlo.

```
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git checkout -- <file>..." to discard changes in working directory)
#
modified: en_blanco.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

Ejecutando la orden `git checkout`, la cosa queda así:

```
luis@luis-laptop:~/cygnus-cloud$ git checkout -- en_blanco.txt
luis@luis-laptop:~/cygnus-cloud$ git status
On branch master
nothing to commit (working directory clean)
```

### B.4.4. Uso de repositorios remotos

Como podréis suponer, un repositorio remoto no es más que un repositorio git alojado en otra máquina. Para mostrar los repositorios remotos que estamos utilizando, es posible utilizar la orden `git remote`.

```
$ git remote
origin
```

El *flag* `-v` muestra también la URL de cada repositorio remoto.

```
$ git remote -v
origin https://code.google.com/p/cygnus-cloud/ (fetch)
origin https://code.google.com/p/cygnus-cloud/ (push)
```

En principio, no es necesario añadir repositorios remotos para empezar a trabajar. Para hacerlo, basta con utilizar la orden

```
git remote add [nombre-del-repositorio] [url]
```

#### B.4.4.1. Incorporar los cambios de un repositorio remoto al repositorio local

Para añadir al directorio de trabajo todos los cambios que se han producido en un repositorio remoto, basta con utilizar la orden

```
git fetch [nombre-del-repositorio]
```

Cuando se clona un repositorio, el repositorio clonado pasa a llamarse `origin`. Como hemos clonado el repositorio de Google Code, `git fetch origin` añadirá todos los cambios del repositorio de Google Code al repositorio local.

**Importante:** la orden `git fetch` *no* mezcla el contenido del repositorio local y del repositorio remoto. Esa mezcla *debe* hacerse a mano.

El uso de `git fetch` “a pelo” es bastante engorroso. La situación más habitual es esta:

- en el repositorio remoto existen varias ramas o *branches*, y
- nos interesa el contenido de *una* de esas ramas. Por ejemplo, una rama con código experimental no es interesante, mientras que la rama `master`, que contiene el código estable, sí lo es.

#### B.4.4.2. Incorporar los cambios del repositorio local a un repositorio remoto

---

En estos casos, lo que se hace es utilizar la orden

```
git pull [nombre-del-repositorio] [nombre-de-la-rama]
```

Esta orden toma los datos de la rama especificada y trata de mezclarlos con lo que hay en el directorio de trabajo. Si hay conflictos, tendrás que resolverlos a mano.

#### B.4.4.2. Incorporar los cambios del repositorio local a un repositorio remoto

En esta sección os enseñaré (al fin) a subir cosas al repositorio en Google Code. El comando que hay que utilizar es

```
git push [nombre-del-repositorio] [nombre-de-la-rama]
```

Por ejemplo, si queremos subir los cambios en la rama master a Google Code, basta con ejecutar `git push origin master`.

Como podrás suponer, solamente es posible subir cosas a un repositorio remoto cuando se dispone de la última versión de sus datos. Esto ya ocurre en otros sistemas de control de versiones, como SVN (recordad que antes de hacer un *commit* era necesario hacer un *checkout*).

#### B.4.4.3. Renombrar y eliminar repositorios remotos

Para cambiar el nombre de un repositorio remoto, basta con utilizar la orden

```
git remote rename [nombre-del-repositorio] [nuevo-nombre-del-repositorio]
```

Para eliminar un repositorio remoto, basta con utilizar la orden

```
git remote rm [nombre-del-repositorio]
```

### B.4.5. Uso de etiquetas

Las etiquetas son sólo formas de marcar puntos importantes de la historia. Lo más habitual es que se correspondan con versiones.

#### B.4.5.1. Crear etiquetas

En Git existen dos clases de etiquetas: ligeras (*lightweight*) y anotadas (*annotated*). Las primeras no son más que el *checksum* de un *commit*, mientras que las segundas almacenan más información, como un mensaje y una fecha.

Para crear una etiqueta anotada, basta con utilizar la orden `git tag -a`.

```
$ git tag -a PlantillasV3 -m 'Plantillas de la documentación'
$ git tag
PlantillasV3
```

Para visualizar el contenido asociado a una etiqueta anotada, se utiliza la orden `git show`.

```
$ git show PlantillasV3
tag PlantillasV3
Tagger: Luis Barrios <luisbarriosshdez@gmail.com>
Date: Mon Sep 24 18:17:22 2012 +0200
Plantillas de la documentación
```

Para crear una etiqueta ligera, basta con no proporcionar ningún *flag* (es decir, sólo el nombre de la etiqueta) a la orden `git tag`.

Con lo que hemos visto hasta ahora, sólo es posible asociar etiquetas al último *commit*. Para asociar las etiquetas a un *commit* anterior, basta con proporcionar el *checksum* o una parte del *checksum* de ese *commit* como último argumento de la orden `git tag`.

#### B.4.5.2. Compartir etiquetas

De forma predeterminada, el comando `git push` *no* transfiere las etiquetas a los repositorios remotos, por lo que habrá que hacerlo de forma explícita. Para ello, basta con utilizar la orden

```
git push [nombre-del-repositorio] [nombre-de-la-etiqueta]
```

Para transferir todas las etiquetas definidas al repositorio remoto, es posible utilizar la orden

```
git push --tags
```

para no hacerlo etiqueta a etiqueta.

### B.5. Uso de ramas o *branches*

#### B.5.1. ¿Qué es una rama?

Para entender exactamente qué es una rama o *branch* y cómo se utilizan en Git hay que tener en cuenta cómo se almacenan los datos.

En la mayoría de sistemas de control de versiones, los datos se almacenan como un conjunto de ficheros y de cambios realizados sobre los mismos a lo largo del tiempo. Esto se muestra en la figura B.1.

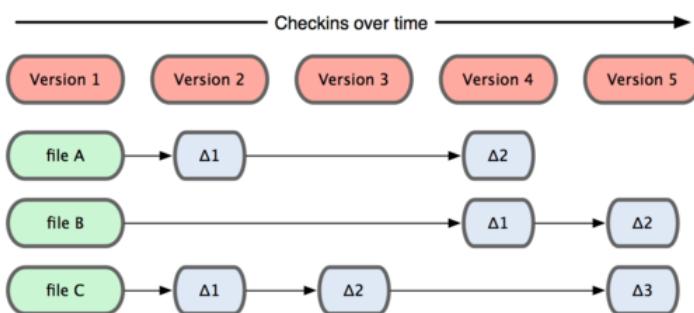


FIGURA B.1: Almacenamiento de los datos en la mayoría de sistemas de control de versiones (CVS, SVN, Bazaar,...)

En cambio, Git almacena los datos como un conjunto de instantáneas de un mismo “sistema de ficheros”. Cada vez que se realiza un *commit*, Git almacena el “estado” de esos ficheros en una instantánea junto con un puntero a la misma. Por eficiencia, si un fichero no ha cambiado, solamente se almacena un enlace a la versión que ya estaba almacenada. Esto aparece en la figura B.2.

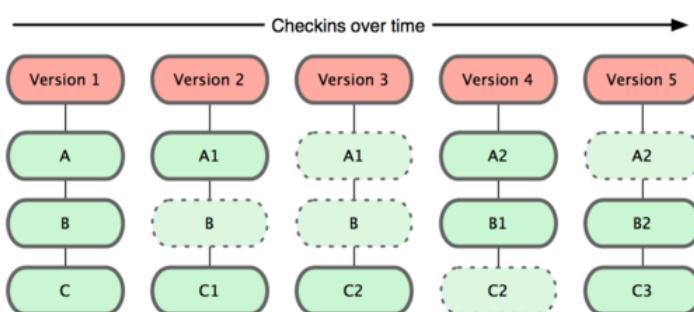


FIGURA B.2: Almacenamiento de los datos en Git

Como ya supondrás, al hacer un *commit* no solamente se almacena un puntero a la instantánea: también se almacenan metadatos (como el autor y el mensaje) y uno o más punteros a los *commits* padre. Así,

- el primer *commit* no tendrá ningún puntero a su padre.
- los *commits* “normales” tienen un puntero a su padre.
- los *commits* que resultan de mezclar dos o más ramas tienen varios punteros.

Para dejar las cosas claras, utilizaré un ejemplo. Supongamos que ejecutamos las órdenes

```
$ git add README test.rb LICENSE
$ git commit -m 'initial commit of my project'
```

Tras el *commit*, los datos del repositorio se organizan de acuerdo a la figura B.3. Los *blobs* no son más que las versiones de los distintos ficheros.

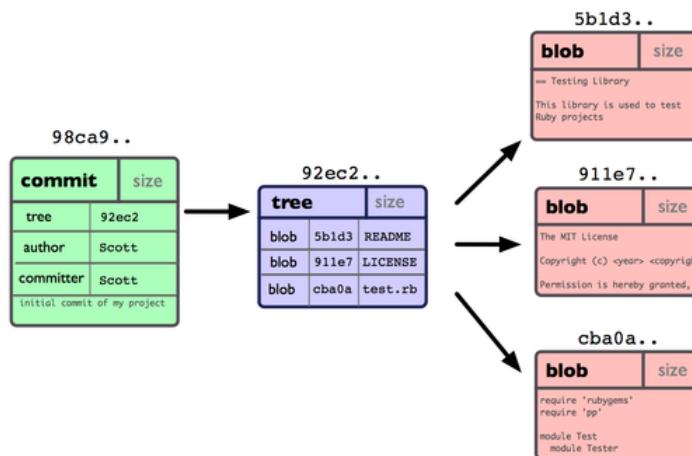


FIGURA B.3: Organización de los datos tras un *commit*

Si introducimos cambios y hacemos otro *commit*, este almacenará un puntero a su *commit* padre. Tras dos *commits* más, la historia sería como la de la figura B.4.

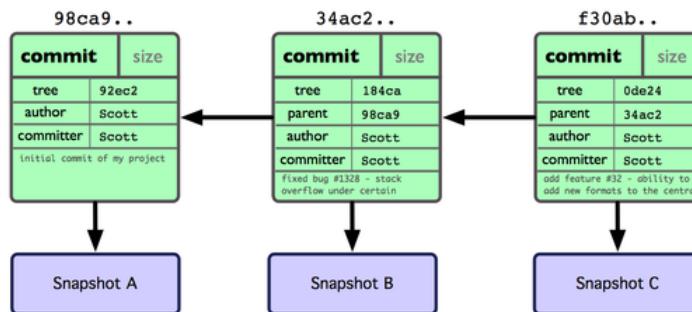


FIGURA B.4: Historia tras dos *commits* más

Tras este rollo, viene lo importante. ¿Qué es una rama? Un puntero a un *commit*. La rama por defecto, que ha aparecido ya en todos nuestros *commits*, se llama `master`.

Cada vez que realizamos un *commit*, se parte de una rama principal que apunta a nuestro último *commit*. Tras cada *commit*, este puntero avanzará automáticamente. Esto se muestra en la figura B.5.

## B.5.2. Uso básico de ramas

Para crear una rama, se utiliza el comando `git branch`. Si ejecutamos el comando

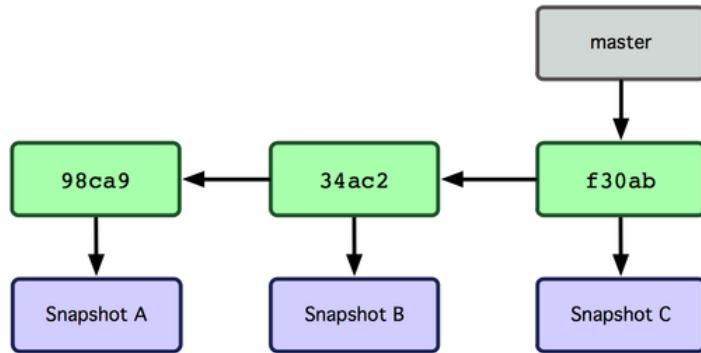


FIGURA B.5: *Branches* y *commits*

```
$ git branch testing
```

¿qué es lo que pasará? Se creará un nuevo puntero al último *commit*. Esto aparece en la figura B.6.

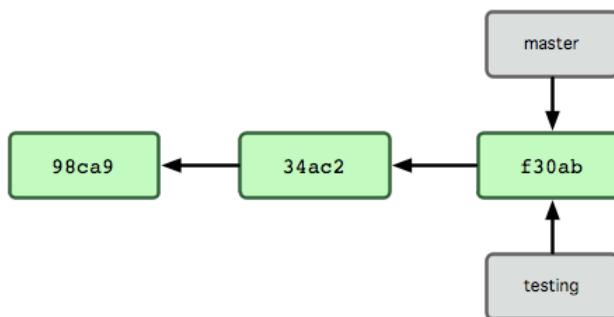


FIGURA B.6: Creación de una nueva rama

¿Cómo averigua Git la rama que hay que utilizar? Manteniendo un puntero especial, `HEAD`, que apunta a la rama *local* en la que se está trabajando.

El comando `git branch` *solo* crea una rama, pero *no* cambia de rama. Para utilizar una rama ya existente, se utiliza el comando `git checkout`. Como podréis suponer, si ejecutamos la orden

```
$ git checkout testing
```

la situación será la de la figura B.7.

¿Y para qué sirve esto? Si hacemos otro *commit*,

```
$ echo "foo" >> LICENSE
$ git commit -a -m 'Cambio en LICENSE'
```

la situación pasa a ser la de la figura B.8.

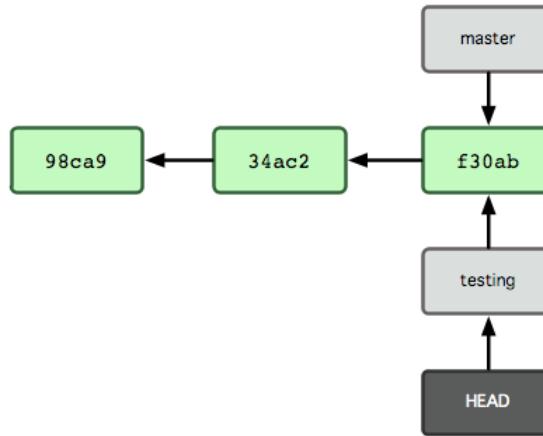
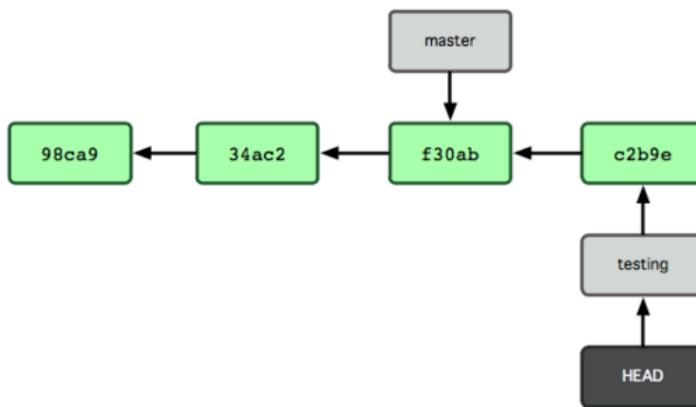
Fijaos bien: mientras que la rama `testing` ha avanzado, `master` sigue apuntando al mismo *commit*. Si ahora ejecutamos

```
$ git checkout master
```

se habrán deshecho todos los cambios. Es decir, cada rama tiene asociada su propia historia.

**Importante:** para evitar problemas, es muy recomendable cambiar de rama *después* de realizar un *commit*.

Las ramas son muy útiles para hacer cambios de forma controlada, pero tarde o temprano esos cambios deberán incorporarse a la rama `master`. Para incorporar los cambios de una rama a otra se utiliza la orden `git merge`. Así, los comandos


 FIGURA B.7: Situación tras ejecutar `git checkout testing`

 FIGURA B.8: *Commit* en la rama *testing*

```

 $ git checkout master
 $ git merge testing

```

incorporan todos los cambios de la rama *testing* a la rama *master*. Dado que el último *commit* de *master* no tiene por qué ser el parent del último *commit* de *testing*<sup>1</sup>, Git deberá trabajar algo más. En cualquier caso, los conflictos que aparezcan deberán resolverse a mano.

Finalmente, las ramas se borran proporcionando el flag `-d` a la orden `git branch`. Así, el comando

```

 $ git branch -d testing

```

eliminará la rama *testing*.

### B.5.2.1. Resolución de conflictos

En algunos casos, mezclar ramas no es trivial. Si se ha modificado el mismo fragmento del mismo fichero en las dos ramas que se están mezclando, aparecerá un conflicto. Por ejemplo, si tenemos dos ficheros `foo.xml` en la raíz del directorio, el último *commit* de *master* no es el ancestro del de *testing* y su contenido es

```

<foo>
 foo
</foo>

```

<sup>1</sup> Cuando esto sucede, Git se limita a mover el puntero de la rama a la que se incorporan los cambios (Fast forward)

y

```
<foo>
 moo
</foo>
```

se producirá un conflicto.

Cuando aparece un conflicto, no se crea el *commit* de la mezcla: la mezcla se pausa hasta que se resuelve manualmente el conflicto.

Resolver conflictos es más sencillo de lo que puede parecer en un principio:

1. Se modifican los fragmentos correspondientes en un editor (escogiendo una alternativa y quitando las líneas <<<<<, ===== y >>>>>).
2. Se ejecuta el comando `git add` sobre los dos ficheros para marcar el conflicto como resuelto.
3. Se realiza un *commit*.

### B.5.3. Gestión de ramas

El comando `git branch` hace algo más que crear y borrar ramas. Si se ejecuta sin argumentos, lista las ramas existentes. El nombre de la rama en la que se está trabajando va precedido por el carácter `*`.

```
$ git branch
* master
 testing
```

Para determinar cuál es el último *commit* realizado en cada rama, es posible utilizar el comando `git branch -v`.

```
$ git branch -v
* master 20aaaf1c Otro commit más
 testing 81f2925 Commit de prueba
```

También es útil saber qué ramas se han mezclado con la actual y cuáles no. Para ello, se utilizan los comandos `git branch --merged` y `git branch --no-merged`.

```
$ git branch --merged
* master
 testing
$ git branch --no-merged
 foo
```

### B.5.4. Ramas remotas

Las ramas remotas son ramas locales que no se pueden modificar. Las modificaciones en estas ramas se realizan intercambiando datos a través de redes, y sus nombres son de la forma

`(repositorio remoto)/(rama)`

Así, la rama `master` del repositorio de Google Code es `origin/master`.

#### B.5.4.1. Acceso a ramas remotas

Para incorporar los cambios en las ramas remotas al directorio de trabajo, basta con ejecutar la orden

```
$ git fetch origin
```

Con esto, descargaréis todos los cambios en el repositorio de Google Code a vuestro directorio de trabajo.

##### B.5.4.2. Crear ramas remotas

Para compartir una rama, hay que registrarlas en el repositorio remoto de Google Code. Las ramas locales no se sincronizan automáticamente con las remotas, por lo que hay que registrarlas manualmente.

Para crear una rama remota, basta con ejecutar la orden

```
$ git push (repositorio remoto) (rama)
```

Es importante notar que, tras leer los datos de una rama remota, no se dispone de una copia local editable. Por ejemplo, tras ejecutar las órdenes

```
$ git push origin foo
$ git fetch origin
```

no existirá la nueva rama `foo`: sólo existirá el puntero `origin/foo`, que *no* podrá modificarse. La copia editable se obtiene mezclando esta rama con la actual, es decir, ejecutando la orden

```
$ git merge origin/foo
```

##### B.5.4.3. Tracking branches

Las *tracking branches* son ramas locales creadas a partir de una rama remota, es decir, ramas locales que tienen una relación directa con una rama remota. Así, al trabajar con una de estas ramas, Git sabe dónde subir los cambios al ejecutar la orden `git push`, y de dónde obtener datos al ejecutar la orden `git pull`.

Al clonar un repositorio, siempre se crea una *tracking branch*, `master`, relacionada con la rama `origin/master`. Esa es la razón por la que `git push` y `git pull` funcionan sin argumentos.

Naturalmente, estas ramas también pueden crearse a mano, mediante la orden

```
$ git checkout --track (repositorio remoto)/(rama)
```

##### B.5.4.4. Borrar ramas remotas

Para borrar una rama remota, basta con utilizar la orden

```
$ git push (repositorio remoto) :(rama)
```

#### B.5.5. Uso de rebase

En Git, existen dos formas de integrar los cambios de una rama en otra: `merge` y `rebase`. En esta sección, presentaremos el uso básico de la segunda.

##### B.5.5.1. Uso básico

Supongamos la situación de la figura B.9, en la que existen dos *commits* en dos ramas diferentes. La forma más sencilla de integrar los cambios de las dos ramas es utilizando el comando `merge`. La situación pasaría a ser la de la figura B.10.

Pero existe una alternativa: podemos tomar los cambios del *commit* C3 y aplicarlos sobre el *commit* C4. Esto se llama *rebasing*. Así, el comando

```
$ git rebase master
```

tomará todos los cambios introducidos en los *commits* de `experiment` y los aplicará sobre `master`. La situación pasará a ser la de la figura B.11.

Si bien el resultado final es el mismo, al usar `rebase` en este caso la historia pasa a ser lineal y los *commits* cambian.

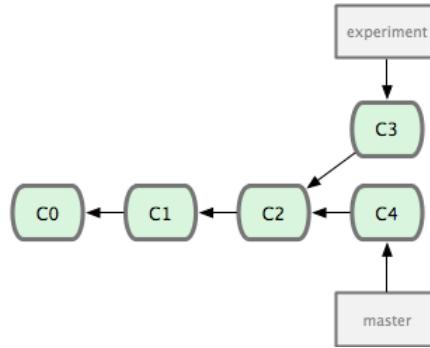


FIGURA B.9: Situación inicial (antes de utilizar `rebase`)

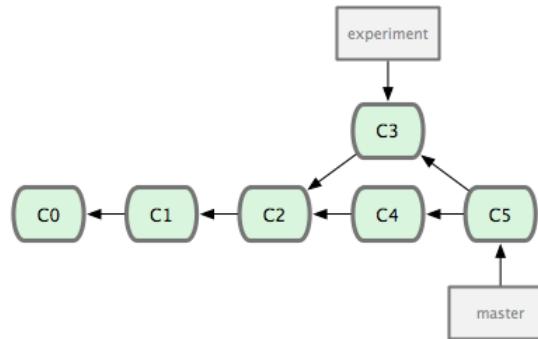


FIGURA B.10: Integrando los cambios con `merge`

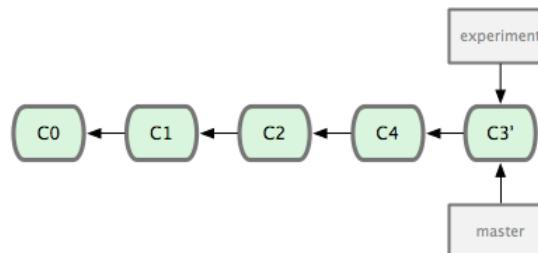


FIGURA B.11: Integrando los cambios con `rebase`

#### B.5.5.2. Un caso interesante

Supongamos la situación de la figura B.12.

Para mezclar las ramas `client` y `master`, dejando de lado los cambios en la rama `server`, se utiliza la opción `--onto` del comando `git rebase`. Así, el comando

```
$ git rebase --onto master server client
```

indicará a Git que hay que

1. tomar los cambios de la rama `client`
2. determinar qué cambios hay que introducir en el ancestro común de `client` y `server`
3. aplicar esos cambios en la rama `master`

El resultado, que aparece en la figura B.13, es bastante interesante.

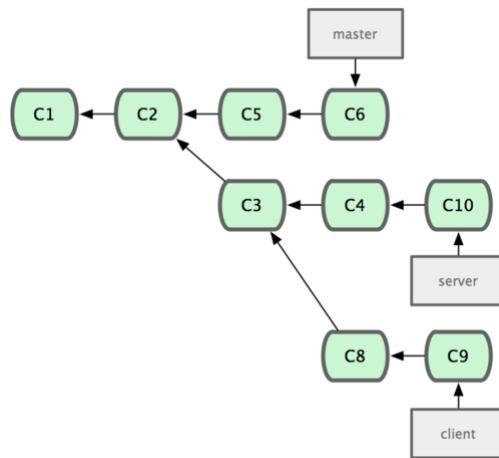


FIGURA B.12: Uso útil de rebase. Situación de partida.

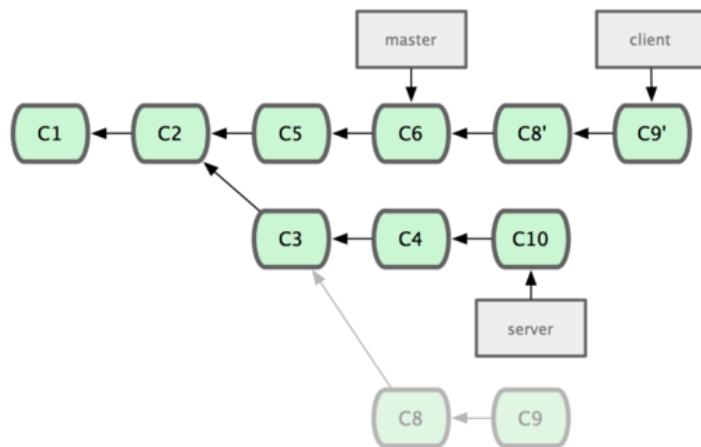


FIGURA B.13: Uso útil de rebase. Resultado obtenido.

Por otra parte hay que tener claro que, al utilizar rebase, se borran los *commits* existentes y se crean nuevos *commits* que son parecidos, pero *no* exactamente iguales. Para evitar que se mezclen esos *commits* “antiguos” con los “nuevos” y evitar el gran desastre, hay que tener clara una cosa: **jamás hay que utilizar rebase para modificar commits que se encuentran registrados en un repositorio remoto.**



## Apéndice C

# BibTeX: guía de referencia

### C.1. Configuración de BibTeX

El documento maestro ya está configurado para incluir las referencias. Para evitar conflictos (y, de paso, para tener ya clasificadas las referencias), cada capítulo de la memoria utilizará una base de datos bibliográfica distinta.

Esta base de datos no es más que un fichero de texto con extensión `.bib`, que podrá modificarse con cualquier editor.

### C.2. Formato de la base de datos

#### C.2.1. Aspectos básicos

En el caso de los ficheros `.bib`,

1. Los espacios en blanco, tabuladores y saltos de línea no son más que delimitadores de palabras.
2. No se distingue entre mayúsculas y minúsculas.
3. Es posible utilizar caracteres acentuados (gracias a la configuración del documento maestro).

#### C.2.2. Registros y campos bibliográficos

En toda base de datos bibliográfica, hay que distinguir entre:

- **Campos.** Son datos básicos aislados (como la fecha de publicación, el nombre de un autor,...).
- **Registros.** Un registro es un conjunto de campos que describe una referencia bibliográfica.

En los ficheros `.bib`, los registros son de la forma

```
@TipoRegistro{clave,
 Campo1,
 Campo2,
 ...
 CampoN
}
```

donde:

- `TipoRegistro` identifica a un tipo concreto de registro (artículo, libro,...).

- La clave identifica al registro, y es siempre su primer elemento. Como podréis suponer, en la misma base de datos *no* podrá haber dos registros con la misma clave, y tampoco deberían usarse bases de datos que comparten claves en el mismo documento. Por convenio, las claves serán de la forma

<Tres iniciales apellido primer autor>Año

Algunos ejemplos de clave son U1175, Knu85. Si existen varias entradas con el mismo autor y el mismo año, se distinguirán añadiendo como sufijo una letra (a para la primera obra, b para la segunda, y así sucesivamente).

- Los campos pueden aparecer en cualquier orden. Para escribirlos, podemos usar uno de estos formatos:

```
Campo = { Contenido }
Campo = " Contenido "
```

En cualquier caso,

- Si no se especifica el nombre, el contenido se descartará.
- Si un campo aparece más de una vez, se tomará el contenido correspondiente a su primera aparición.
- Los delimitadores de contenido deben aparecer siempre que el contenido del campo no sea un número.

Podemos distinguir tres tipos de campos:

- Campos **obligatorios**. Deben aparecer obligatoriamente en determinados tipos de registro.
- Campos **opcionales**. Si aparecen, el estilo los incluye.
- Campos **ignorados**. El estilo los ignorará.

### C.2.3. Tipos de registros bibliográficos

Se recogen en el cuadro [C.2.1](#).

Registro	Descripción	Campos obligatorios	Campos opcionales
<b>article</b>	artículo publicado en una revista	author, title, journal, year	volumen, number, pages, month
<b>book</b>	libro normal	author/editor, title, publisher, year	volume/number, series, address, edition, month
<b>booklet</b>	folleto	title	author, howpublished, address, month, year
<b>inbook</b>	parte de un libro	author/editor, title (título del libro, no del capítulo), chapter/pages, publisher, year	volume/number, series, type, address, edition, month
<b>incollection</b>	parte de un libro con su propio título	author, title, booktitle, Publisher, year	crossref, editor, volume/number, type, chapter, pages, address, edition, month
<b>inproceedings</b>	actas	author, title, booktitle, year	crossref, editor, volume/number, series, pages, address, month, organization, publisher
<b>manual</b>	documentación técnica	title	author, organization, address, edition, month, year
<b>masterthesis</b>	projeto fin de carrera	author, title, school, year	type, address, month
<b>phdthesis</b>	tesis doctoral	author, title, school, year	type, address, month
<b>proceedings</b>	libro de actas	title, year	booktitle, editor, volume/number, series, address, month, organization, publisher
<b>techreport</b>	informe técnico	author title, institution, year	type, number, address, month
<b>unpublished</b>	documento no publicado	author, title, note	month, year
<b>misc</b>	resto de casos	No tiene	author, title, howpublished, month, year

CUADRO C.2.1: Tipos de registros bibliográficos



## Apéndice D

# *Sockets en Python*

### D.1. Introducción

Los sockets son un concepto abstracto con el que se designa al punto final de una conexión. Los programas utilizan sockets para comunicarse con otros programas, que pueden estar situados en computadoras distintas.

Un socket queda definido por la dirección IP de la máquina, el puerto en el que escucha, y el protocolo que utiliza. Los tipos y funciones necesarios para trabajar con sockets se encuentran en Python en el módulo `socket`, como no podría ser de otra forma.

### D.2. Clasificación

Los sockets se clasifican en sockets de flujo (`socket.SOCK_STREAM`) o sockets de datagramas (`socket.SOCK_DGRAM`) dependiendo de si el servicio utiliza TCP, que es orientado a conexión y fiable, o UDP, respectivamente.

Los sockets también se pueden clasificar según la familia. Tenemos sockets `UNIX` (`socket.AF_UNIX`) que se crearon antes de la concepción de las redes y se basan en ficheros, sockets `socket.AF_INET` que son los que nos interesan, sockets `socket.AF_INET6` para IPv6, etc.

### D.3. Familias de Sockets

Dependiendo del sistema y las opciones de construcción, existen varias familias de sockets.

Las direcciones de sockets pueden representarse como siguen :

- Las cadenas de String simples son utilizadas para familias del tipo `AF_UNIX`
- Una par de la forma `(host,port)`. Se utilizan para las familias de tipo `AF_INET` donde “host” es una cadena que representa un nombre de host en la notación de dominio de Internet como `‘daring.cwi.nl’` o una dirección IPv4 como `.50.200.5 ’100’`, y “port” es un entero.
- En el caso de la familia de tipo `AF_INET6`, se utiliza una tupla de cuatro elementos `(host, port, flowinfo, scopeid)`, donde `flowInfo` y `scopeid` representan los miembros `sin6_flowinfo` y `sin6_scope_id` del registro `sockaddr_in6`. Estos parámetros pueden omitirse para compatibilidad con versiones anteriores. Sin embargo, la omisión de `ScopeId` puede causar problemas en la manipulación de ámbito de las direcciones IPv6.
- Los sockets de la familia `AF_NETLINK` se representan como pares `(pid, grupos)`.
- El soporte de `TIPC` en linux solo está disponible bajo al familia `AF_TIPC`. `TIPC` es un protocolo abierto, sin IP y basado en red, diseñado para ser usado en ambientes con computación en cluster. Las direcciones son representadas por una tupla, y sus campos dependen del tipo de dirección. Generalmente suele tener el siguiente aspecto `(addr_type, v1, v2, v3 [, scope])` donde

- `addr_type` es `TIPC_ADDR_NAMESEQ`, `TIPC_ADDR_NAME`, o `TIPC_ADDR_ID`.
  - Si `addr_type` es `TIPC_ADDR_NAME`, entonces `v1` es el tipo de servidor, `v2` es el identificador de puerto, y `v3` suele ser 0.
  - Si `addr_type` es `TIPC_ADDR_NAMESEQ`, entonces `v1` es el tipo de servidor, `v2` es el número menor del puerto, y `v3` es el mayor número de puerto.
  - Si `addr_type` es `TIPC_ADDR_ID`, entonces `v1` es el nodo, `v2` es la referencia, y `v3` suele ser 0.
- `scope` es `TIPC_ZONE_SCOPE`, `TIPC_CLUSTER_SCOPE`, o `TIPC_NODE_SCOPE`.
- Algunas otras familias como `AF_BLUETOOTH` y `AF_PACKET` utilizan representaciones específicas.

## D.4. Creación de un socket

Para crear un socket se utiliza el constructor `socket.socket()` que puede tomar como parámetros opcionales la familia, el tipo y el protocolo. Por defecto se utiliza la familia `AF_INET` y el tipo `SOCK_STREAM`.

### D.4.1. Socket en el servidor

Lo primero que tenemos que hacer es crear un objeto socket para el servidor

```
socket_s = socket.socket()
```

Tenemos ahora que indicar en qué puerto se va a mantener a la escucha nuestro servidor utilizando el método `bind`. Para sockets IP, como es nuestro caso, el argumento de `bind` es una tupla que contiene el host y el puerto. El host se puede dejar vacío, indicando al método que puede utilizar cualquier nombre que esté disponible.

```
socket_s.bind(("localhost", 9999))
```

Por último utilizamos `listen` para hacer que el socket acepte conexiones y `accept` para comenzar a escuchar. `listen` requiere de un parámetro que indica el número de conexiones máximas que queremos aceptar; evidentemente, este valor debe ser al menos 1. `accept` se mantiene a la espera de conexiones entrantes, bloqueando la ejecución hasta que llega un mensaje.

Cuando llega un mensaje, `accept` desbloquea la ejecución, devolviendo un objeto socket que representa la conexión del cliente y una tupla que contiene el host y puerto de dicha conexión.

```
socket_s.listen(10)
socket_c, (host_c, puerto_c) = socket_s.accept()
```

Una vez que tenemos este objeto socket podemos comunicarnos con el cliente a través suyo, mediante los métodos `recv` y `send` (o `recvfrom` y `sendfrom` en UDP) que permiten recibir o enviar mensajes respectivamente. El método `send` toma como parámetros los datos a enviar, mientras que el método `recv` toma como parámetro el número máximo de bytes a aceptar.

```
recibido = socket_c.recv(1024)
print "Recibido: ", recibido
socket_c.send(recibido)
```

Una vez que hemos terminado de trabajar con el socket, lo cerramos con el método `close`.

### D.4.2. Socket en el cliente

Crear un cliente es aún más sencillo. Solo tenemos que crear el objeto socket, utilizar el método connect para conectarnos al servidor y utilizar los métodos send y recv que vimos anteriormente. El argumento de connect es una tupla con host y puerto, exactamente igual que bind.

```
socket_c = socket.socket()
socket_c.connect(("localhost", 9999))
socket_c.send("hola")
```

## D.5. Ejemplos

### D.5.1. Ejemplo 1

En este ejemplo el cliente manda al servidor cualquier mensaje que escriba el usuario y el servidor no hace más que repetir el mensaje recibido. La ejecución termina cuando el usuario escribe “quit”. Este sería el código del script servidor:

```
import socket
s = socket.socket()
s.bind(("localhost", 9999))
s.listen(1)
sc, addr = s.accept()
while True:
 recibido = sc.recv(1024)
 if recibido == "quit":
 break
 print "Recibido:", recibido
 sc.send(recibido)
print "adios"
sc.close()
s.close()
```

Y a continuación tenemos el del script cliente:

```
import socket

s = socket.socket()
s.connect(("localhost", 9999))
while True:
 mensaje = raw_input("> ")
 s.send(mensaje)
 if mensaje == "quit":
 break
print "adios"
s.close()
```

### D.5.2. Ejemplo 2

```
python socket chat example
licence: GPL v3
#server import socket import threading import time

SIZE = 4
soc = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
soc.bind(('127.0.0.1',5432))
```

```
soc.listen(5)

class CThread(threading.Thread):
 def __init__(self,c):
 threading.Thread.__init__(self)
 self.conn = c
 self.stopIt=False

 def mrecv(self):
 data = self.conn.recv(SIZE)
 self.conn.send('OK')
 msg = self.conn.recv(int(data))
 return msg

 def run(self):
 while not self.stopIt:
 msg = self.mrecv()
 print 'recieved-> ',msg

def setConn(con1,con2):
 dict={}
 state = con1.recv(9)
 con2.recv(9)
 if state =='WILL RECV':
 dict['send'] = con1 # server will send data to reciever
 dict['recv'] = con2
 else:
 dict['recv'] = con1 # server will recieve data from sender
 dict['send'] = con2
 return dict

def msend(conn,msg):
 if len(msg)<=999 and len(msg)>0:
 conn.send(str(len(msg)))
 if conn.recv(2) == 'OK':
 conn.send(msg)
 else:
 conn.send(str(999))
 if conn.recv(2) == 'OK':
 conn.send(msg[:999])
 msend(conn,msg[1000:]) # calling recursive

(c1,a1) = soc.accept()
(c2,a2) = soc.accept()
dict = setConn(c1,c2)
thr = CThread(dict['recv'])
thr.start()
try:
 while 1:
 msend(dict['send'],raw_input())
except:
 print 'closing'

thr.stopIt=True
msend(dict['send'],'bye!!!')# for stoping the thread
thr.conn.close()
```

```

soc.close()

#client
import socket
import threading
SIZE =4
class client(threading.Thread):
 def __init__(self,c):
 threading.Thread.__init__(self)
 self.conn = c
 self.stopIt = False

 def mrecv(self):
 data = self.conn.recv(SIZE)
 self.conn.send('OK')
 return self.conn.recv(int(data))

 def run(self):
 while not self.stopIt:
 msg = self.mrecv()
 print 'recieved-> ',msg

soc1 = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
soc1.connect(('127.0.0.1',5432)) soc1.send('WILL SEND')
telling server we will send data from here
soc2 = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
soc2.connect(('127.0.0.1',5432))
soc2.send('WILL RECV') # telling server we will receive data from here

def msend(conn,msg):
 if len(msg)<=999 and len(msg)>0:
 conn.send(str(len(msg)))
 if conn.recv(2) == 'OK':
 conn.send(msg)
 else:
 conn.send(str(999))
 if conn.recv(2) == 'OK':
 conn.send(msg[:999])
 msend(conn,msg[1000:]) # calling recursive

thr = client(soc2)
thr.start()
try:
 while 1:
 msend(soc1,raw_input())
except:
 print 'closing'

thr.stopIt=True
msend(soc1,'bye!!') # for stopping the thread
thr.conn.close()
soc1.close()
soc2.close()

```

## D.6. Bibliografía

1. <http://docs.python.org/3.2/library/socket.html>
2. <http://docs.python.org/3.1/howto/sockets.html>
3. <http://ankurs.com/2009/07/a-simple-python-socket-chat-example/>

## Apéndice E

# Creación de imágenes *Windows* en Xen

Los paquetes que imprescindiblemente tienen que estar instalados son:

- `xen-hypervisor-amd64` y sus dependencias.
- `blktap-utils` y `blktap-kms` para que la E/S a imágenes de disco sea eficiente.
- `bridge-utils` para poder configurar el *bridge* de Xen.

**Importante:** os recomiendo que hagáis una copia de seguridad de todos los ficheros que modifiquéis (o que os apuntéis cómo revertir los cambios) para no cargaros vuestra instalación. Lo mejor es que os creáis una imagen de disco de la instalación base para que así podáis recuperarlos de desastres en menos de 5 minutos. Crear una imagen con *Clonezilla Live* es algo trivial, y a la larga compensa muchísimo.

Bueno, ha llegado el momento de ponernos manos a la obra:

1. Elegid la interfaz de red a la que queréis redirigir el tráfico de los adaptadores de red virtuales. Si sólo tenéis una, os podéis saltar este paso, pero si tenéis dos (por ejemplo, *ethernet* y *WiFi*) tenéis que elegir *una* de ellas. Yo os recomiendo utilizar la interfaz *ethernet* y desactivar la inalámbrica de la forma habitual (es decir, usando el botón; podéis usar comandos a pelo si queréis).
2. Desactivad `network-manager` (si no lo habéis hecho ya) para que podamos configurar la red a placer. Para desactivarlo, escribid en un terminal

```
sudo service network-manager stop
```

Tras hacer esto, la dirección IP y el servidor DNS habrán volado, y no tendréis acceso a internet. Tranquilos, es normal.

3. Modificad el fichero `/etc/network/interfaces` para configurar la red. Suponiendo que queremos configurar *un bridge*, deberíamos añadir las líneas

```
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
auto xenbr0
iface xenbr0 inet dhcp
 bridge_ports eth0
```

Si tenéis dudas sobre el formato de este fichero, usad las órdenes `man 5 interfaces` y `man 5 bridge-utils-interfaces` para obtener más información.

4. Levantad el *bridge* y la interfaz `eth0` mediante los comandos

```
sudo ifup eth0
```

```
sudo ifup xenbr0
```

5. Permitid que todo tipo de tráfico pase a través del *bridge*. Para ello, usad el comando

```
sudo iptables -I FORWARD -m physdev --physdev-is-bridged -j ACCEPT
```

6. Parchear la instalación chapucera de *qemu* que trae Ubuntu: *qemu* espera encontrar los mapas de teclado en */usr/share/qemu*, pero están en */usr/share/qemu-linaro*. Lo arreglamos creando un vulgar enlace simbólico:

```
sudo ln -s /usr/share/qemu-linaro /usr/share/qemu
```

7. Cread el fichero de configuración de *domU* que vamos a crear. Aunque recomiendan colocarlo en */etc/xen*, yo siempre almaceno las imágenes de disco y los ficheros de configuración en el mismo directorio (y lejos de */etc/xen*). Así, es extremadamente sencillo transferir máquinas virtuales ya configuradas entre distintas máquinas. En mi caso, se llamará *Windows.cfg*.

8. Cread la imagen de disco. Para empezar, la crearemos como un fichero *.img*, si bien luego la comprimiremos en formato *vhd* (para ahorrar espacio). La orden a utilizar es

```
dd if=/dev/zero of=Windows.img bs=1 count=1 seek=30G
```

9. Fijad el contenido del fichero *Windows.cfg*. En mi caso, es el siguiente:

```
Habilitar virtualización completa
kernel = '/usr/lib/xen-4.1/boot/hvmloader'
builder = 'hvm'
Configurar RAM, vcpus y shadow memory
memory = 3072
vcpus = 2
Regla: 2 KB por MB de RAM más 1 MB por vcpu
shadow_memory = '8'
Device model (procesa E/S del domU)
device_model = '/usr/lib/xen-4.1/bin/qemu-dm'
Hostname
name = 'Windows'
Almacenamiento: disco duro y CD-ROM
¡¡NO UTILIZAR ESPACIOS dentro de los strings!!
disk = ['tap2:tapdisk:aio:/media/Datos/xen-images/Windows/Windows.img,hda,w',
'phy:/dev/sr0,hdc:cdrom,r']
Bridge a utilizar
vif = ['bridge=xenbr0']
Secuencia de arranque: CD-ROM, disco duro
boot='dc'
Habilitar ACPI
acpi = 1
apic = 1
Configurar VNC
vnc=1
vncviewer=0
sdl=0
usbdevice='tablet'
Tratamiento de eventos
on_poweroff = 'destroy'
on_reboot = 'restart'
on_crash = 'restart'
```

10. Instalar un cliente de escritorio remoto. El que más me gusta es KRDC, aunque vale cualquiera.
11. Terminar la instalación de `blktap-kms`. Para ello,

- a) Configurad DKMS. Así, cada vez que actualicéis el kernel, el módulo `blktap` no se dejará de cargar. Para ello, utilizad el comando

```
sudo dkms autoinstall -k $(uname -r)
```

- b) Cargad el módulo `blktap`. Tendréis que hacer esto en cada arranque.

```
modprobe blktap
```

12. Arrancar la máquina virtual de la forma habitual, y arrancar el cliente de escritorio remoto. Conectaos a `localhost` (no hace falta meter contraseña). Si lo habéis hecho todo bien, veréis cómo aparece el BIOS emulado y, después, el instalador de Windows 7.

A la hora de finalizar la instalación,

- el nombre del equipo deberá ser de la forma

`Xen-Win[XP|Vista|7][Pro|Ent]-[x86_32|x86_64]`

- cread el usuario `VM-Admin`, con contraseña `CygnusCloud`.
- habilidad las actualizaciones automáticas.

**Observación:** en cada reinicio, el domU se destruirá, y tendréis que volver a crearlo (si no recuerdo mal, esto pasa dos veces). Este comportamiento es *normal*, y persistirá hasta que instaléis los drivers de paravirtualización.

13. Instalad los drivers GPL para habilitar la paravirtualización. Los instaladores son estos:

Sistema operativo	Arquitectura	Fichero (de <a href="http://meadowcourt.org/downloads">meadowcourt.org/downloads</a> )
Windows XP	x86_32	<code>gplpv_XP_0.11.0.357.msi</code>
Windows Vista Windows 7 Windows Server 2008 Windows 8	x86_32	<code>gplpv_Vista2008x32_0.11.0.357.msi</code>
Windows Vista Windows 7 Windows Server 2008 Windows 8	x86_64	<code>gplpv_Vista2008x64_0.11.0.357.msi</code>

Hay varias alternativas para pasar el instalador a la máquina virtual. Yo os recomiendo la más simple: copiadlos en un pendrive y enchufadlo al domU añadiendo lo que corresponda a la línea `disk=` del fichero de configuración. En mi caso, lo que hay que añadir es

```
'phy:/dev/sdc,hdb,w'
```

Para saber qué dispositivo es el pendrive, podéis montarlo con el *shell* y después utilizar el comando `mount`. En mi caso, la salida es

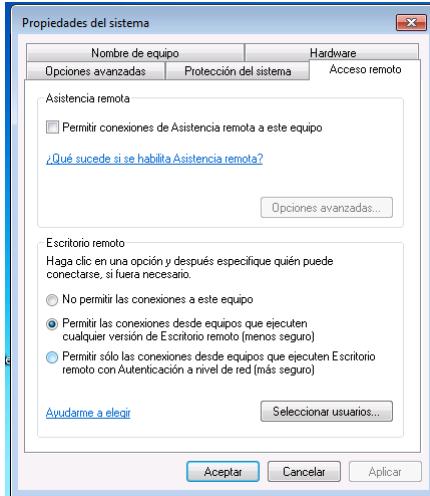


FIGURA E.1: Configuración de RDP en Windows 7 y Vista

```

/dev/sda5 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)

tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)

none on /run/shm type tmpfs (rw,nosuid,nodev)
/dev/sda6 on /home type ext4 (rw)
/dev/sda3 on /media/Datos type fuseblk
(rw,nosuid,nodev,allow_other,default_permissions,blksize=4096)
/dev/sda2 on /media/Windows type fuseblk
(rw,nosuid,nodev,allow_other,default_permissions,blksize=4096)
xenfs on /proc/xen type xenfs (rw)
/dev/sdc1 on /media/Luis type fuseblk
(rw,nosuid,nodev,allow_other,default_permissions,blksize=4096)

```

El 1 que aparece es el número de partición: el pendrive sólo tiene una, por lo que es la primera.

En el siguiente reinicio, notaréis que vuestra máquina virtual es “ligeramente” más rápida.

14. Configurad RDP. Este protocolo de escritorio remoto es el que mejor funciona con máquinas Windows. Los pasos a seguir (en Windows Vista/7) son:
- Haced clic derecho sobre Equipo > Propiedades.
  - Haced clic sobre Configuración Avanzada del Sistema, y luego en la pestaña Acceso remoto. Dejad las cosas tal y como están en la figura E.1.
  - Cerrad sesión e intentad conectaros al domU usando RDP. Como nombre de host, usad el que habéis especificado en la instalación (en mi caso, Xen-Win7Pro-x64) y... voilà.
  - Os recomiendo que uséis el tema Windows 7 Basic: la apariencia de los otros es bastante horrible.

**Importante:** vamos a aprovechar estas imágenes como imágenes *vanilla*. Por eso, *no* cambiéis nada más.

- e) Desactivad VNC. Para ello, cambiad la línea `vnc=1` del fichero de configuración por `vnc=0`.

**Nota:** para apagar el domU, es necesario utilizar el comando `shutdown /n`.

15. Quitar todos los dispositivos de almacenamiento salvo el disco duro. Para ello, modificad la línea `disk=` del fichero de configuración.
16. Configurad el arranque para que sólo se utilice el disco duro. Para ello, modificad la línea `boot='dc'` del fichero de configuración para que sea `boot='c'`.
17. Comprimid la imagen de disco y borrar la antigua (el fichero `.img`). Para ello, utilizad la orden

```
qemu-img convert Windows.img -O qcow2 Windows.qcow2
```

18. Modificad el fichero de configuración para que utilice la nueva imagen. Para ello, sustituid

```
'tap2:tapdisk:aio:/media/Datos/xen-images/Windows/Windows.img,hda,w'
```

por

```
'tap2:tapdisk:qcow:/media/Datos/xen-images/Windows/Windows.qcow,hda,w'
```

Con esto, hemos terminado.



## Apéndice F

# Configuración de KVM

Los pasos a seguir son estos:

1. Nos aseguraremos de que tenemos una CPU con extensiones de virtualización. Si no es el caso, KVM no funcionará. Para ello, usad la orden

```
egrep '(vmx|svm)' --color=always /proc/cpuinfo flags
```

En mi máquina, la salida es esta:

```
luis@luis-desktop:~$ egrep '(vmx|svm)' --color=always
/proc/cpuinfo
flags : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr
pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss
ht tm pbe syscall nx lm constant_tsc arch_perfmon pebs bts
rep_good nopl aperfmpf perfmon dtes64 monitor ds_cpl vmx smx est
tm2 ssse3 cx16 xtpr pdcm sse4_1 xsave lahf_lm dtherm tpr_shadow
vnmi flexpriority
```

Saldrán tantos bloques como *cores* tenga integrados vuestra CPU. En el caso de CPUs Intel, el flag es vmx; en el caso de CPUs AMD, el flag es svm.

2. Instalamos los paquetes básicos

```
sudo apt-get install ubuntu-virt-server python-vm-builder kvm-ipxe
```

3. Añadimos a *root* (y para no estar usando sudo todo el rato, también a nuestro usuario) a los grupos libvirt y kvm. Para ello, utilizaremos el comando adduser, que tiene esta sintaxis básica:

```
adduser <nombre de usuario> <grupo>
```

En mi caso, tendría que ejecutar estas órdenes:

```
sudo adduser root libvirtd
sudo adduser luis libvirtd
sudo adduser root kvm
sudo adduser luis kvm
```

Reiniciad sesión para que los cambios tengan efecto.

4. Comprobamos que KVM funciona, mediante el comando

```
virsh -c qemu:///system list
```

La salida será de la forma:

```
luis@luis-desktop:~$ virsh -c qemu:///system
list
Id Name State

```

Lógicamente, todavía no hemos creado ninguna máquina virtual, por lo que la lista es vacía.

5. Configuramos el bridge, editando el fichero `/etc/network/interfaces` para que quede de esta forma:

```
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
auto kvmbr0
iface kvmbr0 inet dhcp
 bridge_ports eth0
```

6. Detenemos `network-manager` y reiniciamos el demonio de red.

```
sudo service network-manager stop
sudo /etc/init.d/networking restart
```

7. Activamos las interfaces que acabamos de definir

```
sudo ifconfig
```

8. Reiniciamos. Esto es necesario para evitar errores de permisos en `/dev/kvm`.

9. Modificad los ficheros `.xml` del repositorio según el tipo de dominio que vayáis a crear (Windows o GNU/Linux).

10. Arrancad la máquina virtual

```
virsh define <ruta del fichero .xml>
virsh start <nombre de la VM>
```

11. Iniciad un cliente de escritorio remoto, conectándoos a `localhost`. Por defecto, no será necesario introducir un nombre de usuario ni una contraseña.

## Apéndice G

# Configuración de una red virtual en modo NAT

### G.1. ¿Qué queremos hacer?

Al usar NAT, expondremos al exterior una única dirección IP para el servidor de máquinas virtuales. El tráfico de las distintas máquinas virtuales se redirigirá a puertos asociados a esa IP.

Para configurar NAT, hacen falta tres cosas:

- una puerta de enlace (*gateway*), que conectará nuestras máquinas virtuales al exterior.
- una receta de iptables, que determinará cómo redirigir el tráfico destinado a la IP del servidor a las distintas máquinas virtuales.
- un servidor DHCP, que asignará IPs a las máquinas virtuales y les proporcionará un servidor DNS.

Nosotros lo configuraremos todo en ese orden.

**Importante:** antes de continuar, deshabilitad network-manager para poder trastear a gusto con la configuración de la red.

### G.2. Creación del bridge

Para crear y configurar el bridge, utilizaremos el comando `brctl`.

1. Creamos el *bridge*

```
brctl addbr kvmbr0
```

2. Desactivamos STP (*Spanning Tree Protocol*, os deberá sonar de Redes). En nuestra red virtual no habrá ciclos, por lo que no debemos preocuparnos por ellos.

```
brctl stp kvmbr0 off
```

3. Configuramos el *forward delay* del bridge a 0 segundos (¿para qué queremos más?).

```
brctl setfd kvmbr0 0
```

Una vez creado el bridge, debemos registrararlo como interfaz de red. Arbitrariamente, le asignaremos la dirección IP 192.168.77.1.

Como podréis suponer, la primera máquina tendrá la IP 192.168.77.2, la segunda, 192.168.77.3, y así sucesivamente, por lo que la máscara de red debe ser 255.255.255.0.

La orden que configura todo esto es

```
ifconfig kvmbr0 192.168.77.1 netmask 255.255.255.0 up
```

Antes de hacer todo esto, deberíamos comprobar si el *bridge* existe o no. Esto es bastante sencillo:

```
brctl show | grep "kvmbr0" > /dev/null 2> /dev/null
```

Lo único que nos interesa saber es si esta orden falla o no, no la salida de grep.

### G.3. Activar el encaminamiento IP

El siguiente paso es activar el encaminamiento (o *forwarding*) IP, desactivado por defecto en todas las distros de escritorio.

El comando a utilizar es

```
echo 1 | dd of=/proc/sys/net/ipv4/ip_forward > /dev/null
```

De manera predeterminada, este ajuste se pierde tras reiniciar. Si queréis hacerlo permanente, descomentad la línea

```
#net.ipv4.ip_forward=1
```

del fichero `/etc/sysctl.conf`.

### G.4. Configuración de iptables

Antes de nada, es altamente recomendable que guardéis vuestra receta actual para evitar problemas. Podéis hacerlo mediante el comando

```
iptables-save > ruta_del_fichero
```

Para recuperar los ajustes, usad el comando

```
iptables-restore < ruta_del_fichero
```

Ejecutad estas órdenes<sup>1</sup>:

```
iptables -t nat -A POSTROUTING -s 192.168.77.0/255.255.255.0 -j MASQUERADE
iptables -t filter -A INPUT -i kvmbr0 -p tcp -m tcp --dport 67 -j ACCEPT
iptables -t filter -A INPUT -i kvmbr0 -p udp -m udp --dport 67 -j ACCEPT
iptables -t filter -A INPUT -i kvmbr0 -p tcp -m tcp --dport 53 -j ACCEPT
iptables -t filter -A INPUT -i kvmbr0 -p udp -m udp --dport 53 -j ACCEPT
iptables -t filter -A FORWARD -i kvmbr0 -o kvmbr0 -j ACCEPT
iptables -t filter -A FORWARD -s 192.168.77.0/255.255.255.0 -i kvmbr0 -j ACCEPT
iptables -t filter -A FORWARD -d 192.168.77.0/255.255.255.0 -o kvmbr0 -m state
--state RELATED,ESTABLISHED -j ACCEPT
iptables -t filter -A FORWARD -o kvmbr0 -j REJECT --reject-with
icmp-port-unreachable
iptables -t filter -A FORWARD -i kvmbr0 -j REJECT --reject-with
icmp-port-unreachable
```

Para entender estas líneas, hay que tener en cuenta que:

- En el caso de la tabla nat, PREROUTING indica que los datagramas entrantes se alterarán, POSTROUTING indica que los datagramas salientes se alterarán, y OUTPUT indica que los datagramas que se generan en esta máquina también se alterarán.

<sup>1</sup>De forma predeterminada, `iptables` trabaja con la tabla filter. Por eso, el argumento `-t filter` se puede omitir.

- Como podréis suponer, ACCEPT indica que hay que dejar pasar el datagrama.
- MASQUERADE es la dirección IP del servidor de máquinas virtuales.
- En el caso de la tabla filter, INPUT hace referencia a paquetes destinados a esta máquina, FORWARD hace referencia a paquetes que se enrutarán a través de esta máquina y OUTPUT hace referencia a paquetes generados en esta máquina.
- Las primera línea -A INPUT tiene este significado
  - si el datagrama va destinado a la interfaz kvmbr0, es decir, a alguna máquina virtual,
  - lleva datos del protocolo TCP, y
  - está destinado al puerto TCP 67

lo aceptamos. Las otras significan lo mismo que esta.

Los puertos 67 y 53 son los utilizados por los servicios DHCP y DNS.

- Como habréis visto, no hemos conectado el *bridge* a nada. La siguiente línea permite que los datagramas generados por las máquinas virtuales pasen a través del *bridge*.
- La siguiente línea redirige los paquetes procedentes de la red (es decir, del exterior) al *bridge*.
- La siguiente línea redirige los paquetes procedentes del *bridge* (es decir, generados por las máquinas virtuales) al exterior. RELATED y ESTABLISHED hacen referencia al estado de la conexión. Si queréis más detalles, consultad el fichero `man iptables`.
- Las dos últimas líneas descartan los datagramas asociados a errores ICMP.

Para comprobar que la configuración es correcta, usad los comandos

```
iptables -t nat -L -n
```

```
iptables -t filter -L -n
```

## G.5. Configuración de dnsmasq

Para configurar el servidor DNS, ejecutad esta orden:

```
dnsmasq --strict-order --except-interface=lo
--except-interface=eth0 --except-interface=vnet0
--interface=kvmbr0 --listen-address=192.168.77.1
--bind-interfaces --dhcp-range=192.168.77.2,192.168.77.254
--conf-file="" --dhcp-leasefile=/home/luis/kvmbr0.leases
--dhcp-no-override
```

La línea resaltada es la que nos interesa: en ese fichero, figurarán todas las asignaciones de IPs que realiza el servidor DHCP.

## G.6. Redirección del tráfico de un puerto al servidor VNC de una máquina virtual

Por ahora, el servidor VNC lo ponen KVM y Qemu, por lo que reside en el servidor de máquinas virtuales.

Esto significa que, por ahora, no es necesario redirigir el tráfico de un puerto del servidor a una máquina virtual. No obstante, os pongo aquí las dos líneas que lo hacen por si más adelante es necesario hacerlo.

```
iptables -t nat -I PREROUTING -p <tcp|udp> --dport <puerto
servidor> -j DNAT --to-destination <IP VM>:<Puerto VM> iptables
-A FORWARD -i eth0 -o kvmbr0 -p <tcp|udp> --dport <Puerto VM> -j
ACCEPT
```

El primer comando nos permite garantizar que no habrá problemas al crear y utilizar conexiones en las máquinas virtuales. El segundo es el que redirige todo el tráfico de cierto puerto del servidor a cierto puerto de la máquina virtual.

# Bibliografía

- [Car04] Martin Carnoy. ITC in Education: Possibilities and Challenges. In *Apertura del curso académico*. Universitat Oberta de Catalunya, October 2004. Also available at [www.uoc.edu/inaugural04/eng/carnoy1004.pdf](http://www.uoc.edu/inaugural04/eng/carnoy1004.pdf).
- [Cho10] Timothy Chou. *Introduction to Cloud Computing: Business & Technology*. Active Book Press, January 2010.
- [Cry10] Cryptoclarity.com. Encrypted Storage and Key Management for the cloud. *Cryptoclarity.com*, August 2010.
- [dP13] Massimo di Pierro. *Access Control*, 5 edition, 2013.
- [Ene09] ENERGY STAR Program Requirements for Computers. Technical report, US Environmental Protection Agency, 2009. Also available at [http://www.energystar.gov/index.cfm?c=computers.pr\\_crit\\_computers](http://www.energystar.gov/index.cfm?c=computers.pr_crit_computers).
- [evi10] Condorchem evitech. Historia antigua del tratamiento del agua potable. *Blog de ingeniería aplicada al medio ambiente*, January 2010. Also available at <http://blog.condorchem.com/historia-antigua-del-tratamiento-del-agua-potable/>.
- [Fet05] Abe Fettig. *Twisted network programming essentials*. O'Reilly Media, 2005.
- [Gen08] Frank Gens. Defining “Cloud services” and “Cloud computing”. *IDC*, September 2008. Also available at <http://blogs.idc.com/ie/?p=190>.
- [Gil06] George Gilder. Las fábricas de información. *Wired*, October 2006.
- [Gmb] Alternate GmbH. Lista de precios a 25/10/2012. Consulted at <http://www.alternate.es>.
- [GMS<sup>+</sup>04] Fernando Ginés, José M. Mirones, Eduardo Sánchez, David Soria, Rafael Ruiz, Teresa Hortalá, and José M. Mendías. Laboratorios Docentes de la Facultad de Informática de la UCM: un modelo para la gestión integrada de aulas informáticas. In *Primera Jornada Campus Virtual UCM*, pages 146–153. Universidad Complutense de Madrid, November 2004. Also available at [eprints.ucm.es/5501/1/I\\_JORNADA\\_CAMPUS.pdf](http://eprints.ucm.es/5501/1/I_JORNADA_CAMPUS.pdf).
- [Haf09] Gordon Haff. Just don't call them private clouds. *CNET News*, January 2009. Also available at [http://news.cnet.com/8301-13556\\_3-10150841-61.html](http://news.cnet.com/8301-13556_3-10150841-61.html).
- [HBKH09] Judith Hurwitz, Robin Bloor, Marcia Kaufman, and Fern Halper. *Cloud computing for dummies*. Making Everything Easy!, November 2009.
- [INE] INE. Alumnos matriculados por universidad (2011-2012). <http://www.ine.es/jaxi/tabla.do?type=pcaxis&path=/t13/p405/a2010-2011/10/&file=02011.px>.
- [Inf] Clevisa Informática. Lista de precios a 25/10/2012. Consulted at <http://www.clevisa.com>.
- [Inf10] InformationWeek. There's No Such Thing As A Private Cloud. *InformationWeek*, June 2010.

- [Joh08] Bobbie Johnson. Cloud computing is a trap, warns GNU founder Richard Stallman. *The Guardian*, September 2008. Also available at <http://www.guardian.co.uk/technology/2008/sep/29/cloud.computing.richard.stallman>.
- [KVM] KVM Official documentation. Available at <http://www.linux-kvm.org/page/Documents>.
- [Lau93] Mark Laubach, editor. *Meeting report from the IP over ATM working group of the IETF*, July 1993.
- [Liba] Libvirt Language Bindings. Available at <http://libvirt.org/bindings.html>.
- [Libb] Libvirt Supported System. Available at <http://libvirt.org/drivers.html#hypervisor>.
- [MCF08] PAUL MCFEDRIES. The Cloud Is The Computer. *IEEE SPECTRUM*, August 2008. Also available at <http://spectrum.ieee.org/computing/hardware/the-cloud-is-the-computer>.
- [MI] Joel Martin and Hiroshi Ichikawa. noVNC and websockify documentation. Downloadable from noVNC's website: <http://kanaka.github.com/noVNC/>.
- [Mil08] Michael Miller. *Cloud Computing: Web-Based Applications That Change the Way You Work and Collaborate Online*. QUE, August 2008.
- [Mil09] Elinor Mills. Cloud computing security forecast: Clear skies. *CNET News*, January 2009. Also available at [http://news.cnet.com/8301-1009\\_3-10150569-83.html](http://news.cnet.com/8301-1009_3-10150569-83.html).
- [OWA11] OWASP. OWASP: The Open Application Security Project, 2011.
- [Pet] David Peticolas. An introduction to Twisted. Consulted on January, 2013. Available at [http://krondo.com/?page\\_id=1327](http://krondo.com/?page_id=1327).
- [PH62] John W. Plattner and Lowell W. Herron. Simulation: Its Use in Employee Selection and Training. *American Management Association Bulletin*, 20, 1962.
- [Pie13] Massimo Di Pierro. *web2py, Complete Reference Manual.*, 5 edition, 2013.
- [RFM11] Patrick S. Ryan, Sarah Falvey, and Ronak Merchant. Regulation of the cloud in India. *Journal of Internet Law*, October 2011.
- [Sal11] Salesforce. ¿Qué es Cloud Computing? *ITNews*, January 2011. Also available at <http://www.itnews.ec/marco/000035.aspx>.
- [Sos11] Barrie Sosinsky. *Cloud Computing Bible*. John Wiley & Sons, January 2011.
- [Ste12] Alan Stevens. When hybrid clouds are a mixed blessing. *The Register*, August 2012. Also available at [http://www.theregister.co.uk/2011/06/29/hybrid\\_cloud](http://www.theregister.co.uk/2011/06/29/hybrid_cloud).
- [Tec11] National Institute of Science Technology. The NIST definition of Cloud Computing. *Special Publication 800- 145, National Institute of Standards and Technology*, July 2011. Also available at <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>.
- [Tec12] Intrinsic Technology. HVD: the cloud's silver lining. *A white paper*, August 2012. Also available at [http://www.intrinsictechnology.co.uk/FileUploads/HVD\\_Whitepaper.pdf](http://www.intrinsictechnology.co.uk/FileUploads/HVD_Whitepaper.pdf).
- [Twi12] Twisted Matrix Labs. *Twisted 12.3.0 core documentation*, 12.3.0 edition, December 2012. Available at <http://twistedmatrix.com/documents/current/core/howto/index.html>.

- [UCM11] Facultad de Informática UCM. *Memoria de la Facultad de Informática, Curso 2009/2010*. Universidad Complutense de Madrid, 2011. Also available at [http://www.fdi.ucm.es/Futuros\\_Alumnos/memoria\\_curso\\_2009-2010.pdf](http://www.fdi.ucm.es/Futuros_Alumnos/memoria_curso_2009-2010.pdf).
- [Wik] Wikipedia. Original equipment manufacturer. Consulted on 2012/10/25 at [http://en.wikipedia.org/wiki/Original\\_equipment\\_manufacturer](http://en.wikipedia.org/wiki/Original_equipment_manufacturer).