Universidad Zaragoza

Proyecto Fin de Carrera
Ingeniería en Informática

Diseño e implementación de un sistema dinámico de gestión de trabajos distribuidos en un entorno de máquinas virtuales.

David Ceresuela Palomera

Director: Javier Celaya

Departamento de Informática e Ingeniería de Sistemas
Escuela de Ingeniería y Arquitectura
Universidad de Zaragoza

Curso 2011/2012
Septiembre 2012

Resumen

A la hora de ejecutar trabajos en un entorno distribuido, la aproximación clásica ha sido bien el uso de un *cluster* de ordenadores o bien el uso de la computación en malla o *grid*. Con la proliferación de entornos *cloud* durante estos últimos años y su facilidad de uso, una nueva opción se abre para la ejecución de este tipo de trabajos. De hecho, la ejecución de trabajos distribuidos es uno de los principales usos dentro del ámbito de los sistemas *cloud*. Sin embargo, la administración de este tipo de sistemas dista de ser sencilla: cuestiones como la puesta en marcha del sistema, el aprovisionamiento de nodos, las modificaciones del sistema y la evolución y actualización del mismo suponen una tarea intensa y pesada.

En vista de lo cual, en este proyecto se ha diseñado una solución capaz de automatizar la administración de sistemas *cloud*, y en particular de un sistema de ejecución de trabajos distribuidos. Para ello se han estudiado entornos clásicos de ejecución de trabajos como TORQUE y entornos de ejecución de trabajos en *cloud* como AppScale. Además, se han estudiado herramientas clásicas de configuración automática de sistemas como Puppet y CFEngine. El objetivo principal de estas herramientas de configuración de sistemas es la gestión del nodo. En este proyecto se ha extendido la funcionalidad de una de estas herramientas — Puppet — añadiéndole la capacidad de gestión de sistemas *cloud*.

Como resultado de este proyecto se presenta una solución capaz de administrar de forma automática sistemas de ejecución de trabajos distribuidos. La validación de esta solución se ha llevado a cabo sobre los entornos de ejecución de trabajos TORQUE y AppScale y también, para mostrar su carácter genérico, sobre una arquitectura de servicios web de tres niveles.

Índice general

Resumen **i**

1 Introducción **1**
1.1 Contexto del proyecto . . . 2
1.2 Objetivos . . . 3
1.3 Trabajos previos . . . 3
1.4 Tecnologías utilizadas . . . 3
1.5 Herramientas utilizadas . . . 4
1.6 Organización de la memoria . . . 4
1.7 Agradecimientos . . . 4

2 Herramientas e infraestructuras utilizadas **5**
2.1 Herramienta de gestión de configuración . . . 5
2.2 Infraestructuras de ejecución de trabajos distribuidos . . . 6
2.2.1 Infraestructura AppScale . . . 6
2.2.2 Infraestructura TORQUE . . . 8

3 Modelado de recursos distribuidos con Puppet **11**
3.1 Configuración de recursos distribuidos . . . 11
3.2 Modelización en Puppet . . . 12
3.2.1 Patrón de diseño del proveedor . . . 12
3.2.2 *Framework* de implementación . . . 14

4 Metodología de diseño e implementación de recursos distribuidos **17**
4.1 Especificación del tipo . . . 17
4.2 Diseño e implementación del proveedor . . . 20

5 Diseño e implementación de recursos distribuidos **23**
5.1 Diseño e implementación de un recurso distribuido para una infraestructura AppScale . . . 23
5.2 Diseño e implementación de un recurso distribuido para una infraestructura TORQUE . . . 25

6 Validación de la solución planteada **27**
6.1 Pruebas comunes a todas las infraestructuras . . . 27
6.2 Pruebas específicas de la infraestructura AppScale . . . 28
6.3 Pruebas específicas de la infraestructura TORQUE . . . 28
6.4 Pruebas específicas de la infraestructura web de tres niveles . . . 28

7 Conclusiones **31**

Bibliografía **33**

A Gestión del proyecto **35**

B Ejemplo de uso **37**

C Lenguaje de Puppet **41**

D AppScale **43**
D.1 Roles y despliegue . . . 43
D.2 Bases de datos . . . 44
D.3 Infraestructuras . . . 44
D.4 Desarrollo y organización . . . 45

E Instalación de tecnologías **47**
E.1 AppScale . . . 47
E.1.1 Instalación . . . 47
E.1.2 Comprobación de la instalación . . . 48
E.1.3 Versiones instaladas . . . 49
E.2 TORQUE . . . 49
E.2.1 Instalación de TORQUE . . . 49
E.2.1.1 Nodo maestro . . . 49
E.2.1.2 Nodos de computación . . . 49
E.2.2 Configuración de TORQUE . . . 50
E.2.2.1 Nodo maestro . . . 50
E.2.2.2 Comprobación de la configuración . . . 50
E.2.2.3 Envío de un trabajo . . . 51
E.2.3 Versiones instaladas . . . 52
E.3 Arquitectura Web . . . 52
E.3.1 Balanceador de carga . . . 52
E.3.1.1 Instalación . . . 52
E.3.1.2 Configuración . . . 52
E.3.1.3 Ejecución . . . 54
E.3.1.4 Versiones instaladas . . . 54
E.3.2 Servidor web . . . 54
E.3.2.1 Instalación . . . 54
E.3.2.2 Ejecución . . . 55
E.3.2.3 Añadiendo soporte para la base de datos . . . 55
E.3.2.4 Versiones instaladas . . . 56
E.3.3 Base de datos . . . 56
E.3.3.1 Instalación . . . 56
E.3.3.2 Configuración . . . 56
E.3.3.3 Ejecución . . . 57
E.3.3.4 Versiones instaladas . . . 57
E.4 Ruby . . . 57
E.4.1 Instalación . . . 58
E.4.2 Problemas . . . 58
E.4.3 Versiones instaladas . . . 59

E.5 Libvirt . . . 59
E.5.1 Instalación . . . 59
E.5.2 Versiones instaladas . . . 59
E.6 MCollective . . . 59
E.6.1 Instalación . . . 60
E.6.2 Configuración . . . 61
E.6.3 Comprobación de la instalación . . . 61
E.6.4 Versiones instaladas . . . 61
E.7 RabbitMQ . . . 61
E.7.1 Instalación de RabbitMQ . . . 61
E.7.2 Instalación de los *plugins* para el soporte de AMQP y STOMP . . . 62
E.7.3 Configuración de la cuenta de MCollective . . . 62
E.7.4 Comprobación de la instalación . . . 63
E.7.5 Versiones instaladas . . . 63
E.8 Puppet . . . 63
E.8.1 Instalación de libopenssl-ruby . . . 63
E.8.2 Creación del entorno de instalación de Puppet . . . 63
E.8.3 Instalación de Facter . . . 63
E.8.4 Instalación de Puppet . . . 64
E.8.5 Configuración de Puppet . . . 64
E.8.6 Comprobación de la instalación . . . 64
E.8.6.1 Comprobación del maestro de Puppet . . . 65
E.8.6.2 Comprobación del agente de Puppet . . . 65
E.8.6.3 Comprobación del sistema Puppet . . . 67
E.8.7 Problemas . . . 67
E.8.8 Versiones instaladas . . . 68
E.9 Neptune . . . 68
E.9.1 Instalación . . . 68
E.9.2 Comprobación de la instalación . . . 69
E.9.3 Versiones instaladas . . . 69
E.10 God . . . 69
E.10.1 Instalación . . . 69
E.10.2 Comprobación de la instalación . . . 69
E.10.3 Versiones instaladas . . . 70

F Código fuente **71**
F.1 generic-module . . . 71
F.1.1 mcollective_leader.rb . . . 71
F.1.2 genericp_helper.rb . . . 72
F.1.3 cloudmonitor.rb . . . 75
F.1.4 cloudinfrastructure.rb . . . 77
F.1.5 genericp_main.rb . . . 78
F.1.6 vm.rb . . . 88
F.1.7 mcollective_client.rb . . . 88
F.1.8 gedit.sh . . . 89
F.1.9 mcollective_files.rb . . . 89
F.1.10 mcollective_cron.rb . . . 90
F.1.11 cloudconstants.rb . . . 91
F.1.12 mac.rb . . . 91

F.1.13 cloudleader.rb . . . 92
F.1.14 cloudssh.rb . . . 94
F.1.15 ssh_copy_id.sh . . . 96
F.2 appscale . . . 98
F.2.1 appscale.rb . . . 98
F.2.2 appscalep.rb . . . 100
F.2.3 appscalep_helper.rb . . . 104
F.2.4 appscale-run-instances.tcl . . . 105
F.2.5 appscale_yaml.rb . . . 106
F.2.6 appscale_functions.rb . . . 109
F.2.7 appscale_parsing.rb . . . 113
F.2.8 appscale-add-keypair.tcl . . . 117
F.3 torque . . . 118
F.3.1 torque.rb . . . 118
F.3.2 torquep_helper.rb . . . 120
F.3.3 torquep.rb . . . 121
F.3.4 torque_parsing.rb . . . 124
F.3.5 torque_functions.rb . . . 126
F.3.6 torque_yaml.rb . . . 135
F.4 web . . . 136
F.4.1 web.rb . . . 136
F.4.2 webp_helper.rb . . . 139
F.4.3 webp.rb . . . 139
F.4.4 web_parsing.rb . . . 143
F.4.5 web_functions.rb . . . 145
F.4.6 web_yaml.rb . . . 152

Índice de tablas

A.1 Número de horas invertidas. 35

E.1 Versiones instaladas de AppScale. 49
E.2 Versión instalada de TORQUE. 52
E.3 Versión instalada de nginx. 54
E.4 Versiones instaladas de Ruby, RubyGems, Sinatra y ActiveRecord. 56
E.5 Versión instalada de MySQL. 57
E.6 Versiones instaladas de Ruby y RubyGems. 59
E.7 Versión instalada de libvirt. 59
E.8 Versiones instaladas de MCollective y libstomp. 61
E.9 Versiones instaladas de RabbitMQ y Erlang. 63
E.10 Versiones instaladas de Puppet y Facter. 68
E.11 Versión instalada de neptune. 69
E.12 Versión instalada de god. 70

Índice de figuras

2.1 Tecnologías usadas por AppScale. 7

4.1 Infraestructura web de tres niveles. 18

5.1 Infraestructura AppScale en despliegue personalizado. 24
5.2 Infraestructura Torque. 26

A.1 Diagrama de Gantt: septiembre a febrero. 35
A.2 Diagrama de Gantt: marzo a agosto. 35

E.1 Arquitectura de MCollective. 60

Capítulo 1
Introducción

La computación en la nube es un nuevo paradigma que pretende transformar la computación en un servicio. Durante estos últimos años la computación en la nube ha ido ganando importancia de manera progresiva, ya que la posibilidad de usar la computación como un servicio permite a los usuarios de una aplicación acceder a ésta a través de un navegador web, una aplicación móvil o un cliente de escritorio, mientras que la lógica de la aplicación y los datos se encuentran en servidores situados en una localización remota. Esta facilidad de acceso a la aplicación sin necesitar de un profundo conocimento de la infraestructura es la que, por ejemplo, brinda a las empresas la posibilidad de ofrecer servicios web sin tener que hacer una gran inversión inicial en infraestructura propia. Las aplicaciones alojadas en la nube tratan de proporcionar al usuario el mismo servicio y rendimiento que las aplicaciones instaladas localmente en su ordenador.

A lo largo de los últimos años las herramientas de gestión de configuración (o herramientas de administración de sistemas) también han experimentado un considerable avance: con entornos cada vez más heterogéneos y complejos la administración de estos sistemas de forma manual ya no es una opción. Entre todo el conjunto de herramientas de gestión de configuración destacan de manera especial Puppet [1] y CFEngine. Puppet es una herramienta basada en un lenguaje declarativo: el usuario especifica qué estado debe alcanzarse y Puppet se encarga de hacerlo. CFEngine, también con un lenguaje declarativo, permite al usuario un control más detallado de cómo se hacen las cosas, mejorando el rendimiento a costa de perder abstracciones de más alto nivel.

Sin embargo, estas herramientas de gestión de la configuración carecen de la funcionalidad requerida para administrar infraestructuras distribuidas. Son capaces de asegurar que cada uno de los nodos se comporta de acuerdo a la configuración que le ha sido asignada pero no son capaces de administrar una infraestructura distribuida como una entidad propia. Si tomamos la administración de un *cloud* como la administración de las máquinas virtuales que forman los nodos del mismo nos damos cuenta de que la administración es puramente *software*. Únicamente tenemos que asegurarnos de que para cada nodo de la infraestructura distribuida hay una máquina virtual que está cumpliendo con su función.

Teniendo en cuenta el considerable avance de la computación en la nube, parece claro que el siguiente paso de las herramientas de gestión de configuración debería ir encaminado a la gestión de la nube. Para demostrar una posible manera en la que esto se podría lograr, en este proyecto se ha tomado una de esas herramientas de gestión de la configuración y se ha extendido su funcionalidad añadiéndole la posibilidad de gestionar infraestructuras distribuidas. La modificación realizada se ha validado usando tres ejemplos de infraestructuras distribuidas: las dos primeras de ejecución de trabajos y la última es una infraestructura web de tres niveles

que demuestra que la solución obtenida no es únicamente válida para las infraestructuras de ejecución de trabajos.

La primera de estas infraestructuras es AppScale [2], una implementación de código abierto del App Engine de Google [13]. App Engine permite alojar aplicaciones web en la infraestructura que Google posee. Además del alojamiento de aplicaciones web que ofrece App Engine, AppScale también ofrece las APIs de EC2 [3], MapReduce [4] y Neptune [5]. La API de EC2 añade a las aplicaciones la capacidad de interactuar con máquinas alojadas en Amazon EC2 [7]. La API de MapReduce permite escribir aplicaciones que hagan uso del *framework* MapReduce. La última API, Neptune, añade a AppScale la capacidad de usar los nodos de la infraestructura para ejecutar trabajos. Los trabajos más representativos que puede ejecutar son: de entrada, de salida y MPI [1]. El trabajo de entrada sirve para subir ficheros (generalmente el código que se ejecutará) a la infraestructura, el de salida para traer ficheros (generalmente los resultados obtenidos después de la ejecución) y el de MPI para ejecutar un trabajo MPI.

La segunda es TORQUE [28, 25], una infraestructura de ejecución de trabajos distribuidos. Este tipo de infraestructuras están especializadas en la ejecución de grandes cargas de trabajo paralelizable e intensivo en computación. Son por lo tanto idóneas para ser usadas en la computación de altas prestaciones.

La tercera y última infraestructura es la de servicios web en tres capas, así denominada porque tiene tres niveles claramente diferenciados: balanceo o distribución de carga, servidor web y base de datos. El balanceador de carga es el encargado de distribuir las peticiones web a los servidores web que se encuentran en el segundo nivel de la infraestructura. Éstos procesarán las peticiones web y para responder a los clientes puede que tengan que consultar o modificar ciertos datos. Los datos de la aplicación se encuentran en la base de datos, el tercer nivel de la estructura, y por consiguiente, cada vez que uno de los elementos del segundo nivel necesite leer información o modificarla, accederá a este nivel. Para esta infraestructura no se puede elegir un ejemplo que destaque sobre los demás porque es tan común que cualquier página web profesional de hoy en día se sustenta en una infraestructura similar a ésta.

1.1 Contexto del proyecto

Para la realización de este proyecto de fin de carrera se ha hecho uso del laboratorio 1.03b de investigación que el Departamento de Informática e Ingeniería de Sistemas posee en la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza. Los ordenadores que forman este laboratorio poseen procesadores con soporte de virtualización, lo que permite la creación de diversas máquinas virtuales. La creación de los distintos tipos de *cloud* que representan cada una de las infraestructuras distribuidas se ha llevado a cabo a través de máquinas virtuales alojadas en distintos ordenadores del laboratorio.

En este laboratorio se ha comprobado la validez de la extensión introducida en la herramienta de gestión de configuraciones Puppet para administración de infraestructuras distribuidas que se ha desarrollado a lo largo de este proyecto de fin de carrera.

[1] MPI (del inglés *Message Passing Interface*, Interfaz de Paso de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones de una biblioteca de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

1.2 Objetivos

El objetivo de este proyecto es proporcionar una herramienta que facilite la puesta en marcha de infraestructuras distribuidas y su posterior mantenimiento. Las tareas principales en las que se puede dividir este proyecto son:

1. Modelado de un recurso distribuido en la herramienta de gestión de la configuración Puppet, proporcionando facilidades a los usuarios que quieran definir infraestructuras distribuidas.

2. Definición de una metodología de diseño e implementación de un recurso sistema distribuido.

3. Aplicación de dicha metodología en las infraestructuras distribuidas AppScale, TORQUE y servicios web en tres niveles.

4. Validación de la extensión desarrollada en la herramienta de gestión de configuración para la puesta en marcha y el mantenimiento de dichas infraestructuras.

1.3 Trabajos previos

Desde un primer momento se decidió trabajar con la herramienta de gestión de la configuración Puppet para la realización de este proyecto. La otra alternativa posible era CFEngine, pero a diferencia de ésta, Puppet posee un nivel mayor de abstracción que permite un mejor modelado de los recursos de un sistema. Además, el hecho de que Puppet esté programado en Ruby hace que sea más fácil trabajar y realizar abstracciones de alto nivel en él que en CFEngine, que está programado en el lenguaje C.

También se decidió desde el principio trabajar con la infraestructura de ejecución de trabajos que proporciona AppScale. AppScale combina la capacidad de ejecutar trabajos con el alojamiento de aplicaciones web. Esta dualidad la convierte en una infraestructura muy interesante para trabajar con ella.

La infraestructura de ejecución de trabajos TORQUE también se eligió desde el inicio. La infraestructura de servicios web en tres niveles se incluyó para cerciorarnos de que la solución aportada era lo suficientemente genérica para cualquier infraestructura distribuida, y no se centraba únicamente en las de ejecución de trabajos.

1.4 Tecnologías utilizadas

Para la elaboración de este proyecto se ha hecho uso de las siguientes tecnologías: KVM, QEMU, libvirt y virsh para el soporte y la gestión de las máquinas virtuales; Puppet como herramienta de configuración automática; Ruby como lenguaje de programación para la extensión de Puppet; AppScale y TORQUE como infraestructuras de ejecución de trabajos distribuidos en las que validar la extensión; Nginx, WEBrick y MySQL como balanceador de carga, servidor web y base de datos para la infraestructura web de tres niveles en la que se valida la extensión; Shell como lenguaje de programación de los *scripts* de configuración de las máquinas virtuales; Sistema operativo Debian para las máquinas del laboratorio y Ubuntu para las máquinas virtuales; LaTeX [26] para la redacción de esta memoria y Dia para la confección de los diagramas que aparecen en esta memoria.

1.5 Herramientas utilizadas

Una de las herramientas sobre las que se ha basado este proyecto ha sido la virtualización *hardware* o virtualización de plataforma, que permite la simulación de un ordenador completo (llamado huésped) dentro de otro ordenador (llamado anfitrión). A la hora de hacer una virtualización *hardware* hay varias opciones entre las que elegir, siendo las más ampliamente usadas Xen y KVM. La principal diferencia entre ellas es que Xen ofrece paravirtualización mientras que KVM ofrece virtualización nativa.

La virtualización nativa permite hacer una virtualización *hardware* completa de manera eficiente. Para ello, y a diferencia de la paravirtualización, no requiere de ninguna modificación en el sistema operativo de la máquina virtual; pero a cambio necesita un procesador con soporte para virtualización. KVM, que proporciona virtualización *hardware*, está incluido como un módulo del núcleo de Linux desde su versión 2.6.20, así que viene incluido por defecto en cualquier sistema operativo con núcleo Linux.

Como los ordenadores del laboratorio poseen procesadores con extensiones de soporte para virtualización y sistema operativo Debian, se eligió KVM para dar soporte a las máquinas virtuales. Esto significa que se puede usar cualquier sistema operativo para las máquinas virtuales, sin necesidad de hacer ninguna modificación en el mismo.

El resto de herramientas utilizadas se explican con más detalle en sus respectivas secciones.

1.6 Organización de la memoria

El resto de este documento queda organizado de la siguiente manera: el capítulo 2 describe las herramientas e infraestructuras utilizadas; el capítulo 3 indica cómo se ha realizado el modelado de recursos distribuidos con Puppet; el capítulo 4 introduce la metodología de diseño e implementación de recursos distribuidos; el capítulo 5 muestra como aplicar dicha metodología a la creación de recursos distribuidos; el capítulo 6 valida la solución planteada y el capítulo 7 presenta las conclusiones obtenidas e indica las posibilidades de trabajo futuro.

Además consta de una serie de anexos organizados de esta manera: el anexo A muestra la gestión del proyecto; el anexo B muestra un ejemplo de uso; el anexo C amplía el lenguaje declarativo de Puppet; el anexo D profundiza en la infraestructura AppScale; el anexo E muestra el proceso de instalación de algunas de las tecnologías usadas a lo largo de este proyecto y el anexo F contiene el código fuente de la solución planteada.

1.7 Agradecimientos

Quisiera agradecer en primer lugar a Javier Celaya y a Unai Arronategui por la oportunidad que me han dado de hacer este proyecto, por su guía y por su infinita paciencia durante el mismo.

A los amigos de aquí y a los de allá, a los de la carrera y a los de fuera, por hacer esto mucho más llevadero.

A mis padres, por trabajar durante toda su vida para que hoy pueda estar donde estoy y por su apoyo a lo largo de toda la carrera.

Capítulo 2
Herramientas e infraestructuras utilizadas

En este capítulo se realiza un breve análisis de las distintas herramientas e infraestructuras usadas a lo largo del proyecto.

2.1 Herramienta de gestión de configuración

Las herramientas de gestión de configuración tienen como objetivo describir y llevar a un sistema informático a un cierto estado. Normalmente esto suele incluir llevar a una serie de recursos (ficheros, usuarios, paquetes instalados, etc.) al estado deseado. Para cumplir esta misión se apoyan en dos conceptos básicos: iteración y convergencia. Estas herramientas tratan iteración tras iteración de acercar el sistema informático lo máximo posible al estado deseado. Es posible, por tanto, que el estado final no se alcance en una única ejecución, sino que sean necesarias varias ejecuciones.

Aunque esto pueda contrastar con el funcionamiento habitual de los programas (sería sorprendente que sólo se abriera medio editor de texto), no es tan excepcional en este entorno: si tenemos que poner en marcha dos servicios, de los cuales uno de ellos depende del otro, hasta que el primero no esté funcionando no podrá hacerlo el segundo. Una sola ejecución de la herramienta no serviría para poner ambos servicios en marcha, sino que serían necesarias dos iteraciones como mínimo.

Puppet [1] es una herramienta de gestión de configuración que posee un lenguaje declarativo (ver Anexo C) mediante el cual se especifica la configuración de los sistemas. A través de este lenguaje se especifica de forma declarativa los distintos elementos de configuración, que en la terminología de Puppet se llaman recursos. Mediante el uso de este lenguaje se indica en qué estado se quiere mantener el recurso y será tarea de Puppet el encargarse de que así sea. Cada recurso está compuesto de un tipo (el tipo de recurso que estamos gestionando), un título (el nombre del recurso) y una serie de atributos (los valores que especifican el estado del recurso).

A la especificación de forma declarativa de los recursos y del estado que estos deben alcanzar se le denomina manifiesto. En general, un manifiesto contiene la información necesaria para realizar la configuración de un nodo. Un administrador de un sistema informático que use Puppet creará manifiestos en los que especifique los recursos y su estado. Por ejemplo, para crear un fichero el administrador escribirá en un manifiesto algo similar a lo siguiente:

```
file {'testfile':
  path    => '/tmp/testfile',
  ensure  => present,
  mode    => 0640,
  content => "I'm a test file.",
}
```

Cuando se tiene listo el manifiesto a Puppet se le da la orden de aplicarlo. Los pasos que sigue para aplicarlo son:

- Interpretar y compilar la configuración.
- Comunicar la configuración compilada al nodo.
- Aplicar la configuración en el nodo.
- Enviar un informe con los resultados.

Normalmente Puppet se ejecuta de manera periódica mediante un planificador de trabajos (por ejemplo, cron). Cada cierto tiempo contactará con el nodo que debe ser administrado y volverá a repetir los pasos anteriores. Es decir, Puppet está continuamente intentando llevar al nodo al estado especificado en el manifiesto. Si entre una ejecución y otra algo cambiara en el nodo, Puppet se daría cuenta e intentaría llevar al nodo al estado que le corresponde.

Aunque Puppet tiene la capacidad de actuar sobre un nodo distinto al nodo desde el que se aplica el manifiesto, a lo largo de este proyecto las ejecuciones de Puppet siempre serán sobre el nodo que aplica el manifiesto, siempre serán locales. Uno de los motivos para que esto sea así es que al ser un sistema distribuido con potencialmente muchos nodos es más sencillo si cada nodo posee el manifiesto que a él le corresponde en vez de que un único nodo alberge uno o dos manifiestos enormes en los que se especifique el sistema entero. Otro de los motivos es que la tolerancia a fallos es mejor cuanto menos se dependa de un nodo.

Dentro del ecosistema de Puppet hay una herramienta llamada MCollective E.6. Esta herramienta permite interactuar con grupos de máquinas y es especialmente útil cuando se quiere enviar una orden a varias máquinas en paralelo. La manera tradicional de implementar este tipo de comunicación es mediante un bucle de conexiones ssh, sin embargo, MCollective utiliza el patrón de mensajes editor/suscriptor. Hay varias tecnologías que implementan este patrón de mensajes, pero por simplicidad en este proyecto se usa RabbitMQ E.7.

2.2 Infraestructuras de ejecución de trabajos distribuidos

En esta sección se analizan en profundidad las dos infraestructuras de ejecución de trabajos distribuidos usadas en este proyecto: AppScale y TORQUE.

2.2.1 Infraestructura AppScale

AppScale es una implementación de código abierto del App Engine de Google. Al igual que App Engine, AppScale permite alojar aplicaciones web; a diferencia de App Engine, las aplicaciones no serán alojadas en la infraestructura que Google posee, sino que serán alojadas en una infraestructura que el usuario posea. Además de permitir alojar aplicaciones web, AppScale también ofrece las APIs de MapReduce y Neptune. La API de MapReduce permite escribir aplicaciones

que hagan uso del *framework* MapReduce. La API de Neptune añade a App Engine la capacidad de usar los nodos de la infraestructura para ejecutar trabajos. Los trabajos más representativos que puede ejecutar son: de entrada, de salida y MPI, aunque también se pueden ejecutar trabajos de otro tipo.

AppScale trata de imitar de la manera más fiel posible los servicios que ofrece el App Engine de Google, tratatando de alcanzar el mayor grado de compatibilidad con éstos. Como la tecnología que Google usa internamente en App Engine permanece oculta al público, AppScale tiene que hacer uso de otras tecnologías existentes para ofrecer las mismas funciones. En la Figura 2.1 [1] se puede ver toda la pila de tecnologías usadas en AppScale.

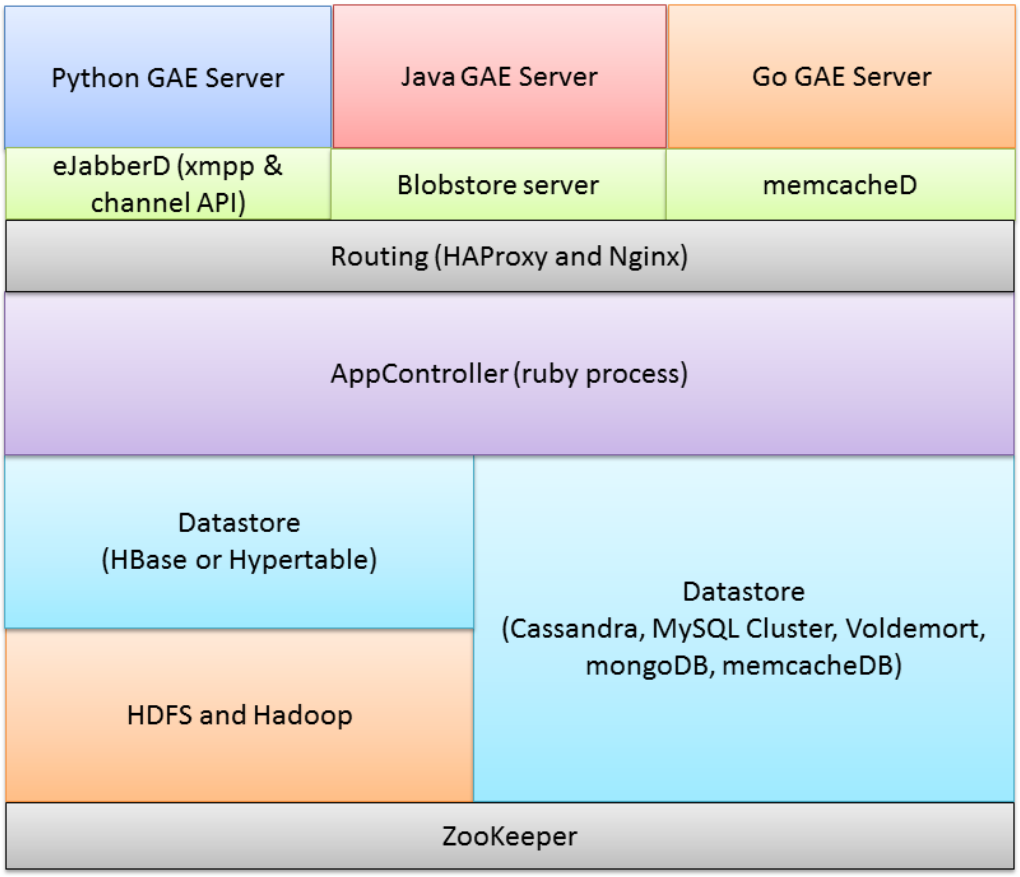

Figura 2.1: Tecnologías usadas por AppScale.

Para permitir el alojamiento de aplicaciones web y la ejecución de trabajos una compleja infraestructura debe ponerse en marcha. Los distintos roles que un nodo puede desempeñar en una infraestructura de este tipo son los siguientes:

`Shadow` : Comprueba el estado en el que se encuentran el resto de nodos.

`Load balancer` : Servicio web que lleva a los usuarios a las aplicaciones.

`AppEngine` : Aloja las aplicaciones web de los usuarios.

`Database` : Ejecuta los servicios necesarios para alojar a la base de datos elegida.

[1] Imagen obtenida de `http://code.google.com/p/appscale/wiki/Supported_AppEngine_APIs`

`Login` : Máquina desde la que se pueden hacer funciones administrativas.

`Memcache` : Proporciona soporte para almacenamiento en caché para las aplicaciones App Engine.

`Zookeeper` : Aloja los metadatos necesarios para hacer transacciones en las bases de datos.

`Open` : No ejecuta nada por defecto, pero está disponible en caso de que sea necesario.

Como muchos de estos roles suele desempeñarlos una máquina, se han creado unos roles agregados para facilitar el despliegue de la infraestructura. El rol `Controller` está compuesto por los roles Shadow, Load Balancer, Database, Login y Zookeeper; el rol `Servers` agrupa a los roles AppEngine, Database y Load Balancer; el rol `Master` está formado por los roles Shadow, Load Balancer y Zookeeper. Una especificación más detallada de los roles puede encontrarse en el Anexo D.

AppScale contempla la posibilidad de dos tipos de despliegue: por defecto y personalizado. En el despliegue por defecto un nodo sólo puede ser o bien `Controller` o bien `Servers`. En el despliegue personalizado el usuario es libre de especificar los roles de manera más precisa.

Una vez puesta en marcha la infraestructura de AppScale, servirá tanto para alojar las aplicaciones web que el usuario despliegue como para ejecutar trabajos. Para desplegar las aplicaciones web hay que hacer uso de las AppScale Tools, un conjunto de herramientas que permiten, entre otras cosas, iniciar y terminar instancias, desplegar aplicaciones y eliminar aplicaciones. Para ejecutar trabajos hay que servirse, además de las AppScale Tools, de la API de Neptune, que no es tan sencilla. En general, la ejecución de trabajos se consigue mediante tres pasos: en el primero se le indica el código fuente que se quiere subir a la infraestructura; en el segundo se le da la orden de ejecutar el trabajo; en el tercero se le piden los resultados de la ejecución. Cada uno de estos pasos debe indicarse, mediante un lenguaje específico de dominio, en un fichero que luego se interpretará con el programa Neptune. En el caso de un trabajo MPI, podemos definir, además del código a ejecutar, el número de máquinas sobre las que ejecutar el código y el número de procesos que se usarán para el trabajo.

2.2.2 Infraestructura TORQUE

La otra infraestructura de ejecución de trabajos distribuidos que se ha elegido ha sido TORQUE, una de las infraestructuras clásicas en lo que a ejecución de trabajos se refiere. TORQUE permite a los usuarios la ejecución de trabajos *batch* sobre un conjunto de nodos. Además de los nodos de computación necesarios para ejecutar los trabajos, una infraestructura de este tipo estará compuesta de un nodo maestro, que es el encargado de planificar la ejecución de los trabajos y de la gestión de los nodos de computación.

Una vez que la infraestructura está en funcionamiento los usuarios pueden mandar sus trabajos al nodo maestro. Éste, valiéndose de un planificador (en su versión más simple es una cola FIFO), decidirá a cuál de los nodos de computación le enviará el trabajo. El nodo de computación que reciba el trabajo será el encargado de ejecutarlo y de enviar los resultados de vuelta al nodo maestro una vez terminada la ejecución.

Además de contar con un planificador básico, TORQUE permite la integración con otros planificadores tanto comerciales (Maui Cluster Scheduler) como de código abierto (Moab Workload Manager) para mejorar la planificación de trabajos y la administración de los nodos del *cluster*. A lo largo de este proyecto se usará, por simplicidad, el planificador más básico.

Para determinar qué usuarios son los que pueden enviar trabajos y qué usuarios son los que pueden realizar tareas administrativas, como por ejemplo cancelar un trabajo, TORQUE permite especificar dos grupos de usuarios: `managers` y `operators`. Para administrar el nodo maestro un usuario deberá pertenecer al primer grupo; para mandar trabajos deberá pertenecer al segundo. Por defecto únicamente el usuario `root` del nodo maestro puede administrarlo.

Capítulo 3
Modelado de recursos distribuidos con Puppet

Las herramientas de gestión de la configuración se han centrado en la gestión de recursos de manera local a un nodo. Por otra parte la automatización existente en la administración de infraestructuras distribuidas es de bajo nivel, no yendo mucho más allá de la gestión de máquinas virtuales. En este capítulo veremos cómo se pueden crear recursos distribuidos en la herramienta de gestión de la configuración Puppet y cómo ésta puede ser usada para automatizar una administración de más alto nivel en infraestructuras distribuidas que tenga en cuenta conceptos como el de disponibilidad o el de dependencia.

3.1 Configuración de recursos distribuidos

En Puppet, la definición clásica de recursos se presupone dentro del ámbito local del nodo. Es decir, para cada nodo especificamos qué recursos debe contener y cuál debe ser el estado de dichos recursos. Dentro de este tipo de recursos se encuentran, entre otros, el recurso usuario y el recurso fichero. Sin embargo, el modelado de un recurso sistema distribuido plantea ciertos desafíos a la definición clásica de recursos de Puppet, ya que ésta no está pensada teniendo en cuenta la problemática asociada a los sistemas distribuidos.

Al modelar un recurso sistema distribuido, deben tenerse en cuenta las características propias de este tipo de recursos, como las dependencias y la disponibilidad. Las dependencias contemplan la necesidad de que un servicio esté funcionando para que otro pueda hacerlo. La gestión de las dependencias es inherente a Puppet, ya que como herramienta de gestión de la configuración trabaja conforme a los conceptos de iteración y convergencia intentando llevar al sistema informático al estado deseado.

La disponibilidad contempla los fallos que se pueden dar en una infraestructura distribuida y en ella están incluidos los fallos de procesos y los fallos de máquinas. Como lo deseable en un sistema distribuido es que el fallo en un nodo no cause el fallo de todo el sistema, a lo largo de los años se han desarrollado distintas soluciones para este problema. La que se usa en este proyecto consiste en la elección de un nodo líder que será el encargado de tratar con los fallos que se den en el sistema. Para determinar quién es el nodo líder se usa un algoritmo de elección de líder, en concreto el algoritmo peleón (en inglés *Bully algorithm*). En este algoritmo todos los nodos tratan periódicamente de convertirse en el líder; si hay un líder, impedirá que otro nodo le quite el liderazgo, y si no lo hay, uno de los restantes nodos se convertirá en líder.

Además de la gestión de las dependencias y la disponibilidad, un recurso sistema distribuido puede presentar elementos comunes con otros recursos sistema distribuido, tales como una moni-

torización básica. La presencia de elementos comunes entre los recursos clásicos de Puppet, por ejemplo un fichero y un usuario, no es tan corriente.

A la hora de definir un recurso sistema distribuido tenemos que presentarlo como un único sistema coherente, es decir, como una única abstracción, y por lo tanto no vale con describir un recurso sistema distribuido como una colección de recursos clásicos de Puppet. Afortunadamente, Puppet proporciona varios mecanismos de extensión. El primer mecanismo de extensión de Puppet consiste en la creación de nuevos tipos de recurso. El segundo es la creación de subcomandos y acciones dentro de Puppet mediante la API *Faces* [27]. Después de analizar esta API a fondo se vio que las opciones que ofrecía no permitían la integración del recurso sistema distribuido dentro del modelo de Puppet. Como lo que interesaba era crear una abstracción del recurso distribuido esta opción se acabó descartando en favor de la creación de un nuevo tipo de recurso y un proveedor asociado, que soluciona el problema de una manera más elegante: la creación de un nuevo tipo de recurso con sus atributos correspondientes para definir los recursos sistema distribuido y la creación de un proveedor que implemente los pasos necesarios para llevar dicho recurso al estado deseado.

3.2 Modelización en Puppet

Puppet puede ser extendido para incluir la definición de nuevos recursos. Para ello hay que proporcinarle como mínimo dos ficheros: uno en el que se define el recurso y otro en el que se define cómo gestionar ese recurso. Al fichero en el que se define el recurso se le llama tipo y al fichero en el que se define cómo gestionarlo se le llama proveedor. Es decir, el tipo se encarga del "qué" y el proveedor se encarga del "cómo".

Para definir un recurso sistema distribuido, al que también llamaremos *cloud*, se han considerado como fundamentales los siguientes atributos:

- Nombre: Para identificar al recurso de manera única.
- Fichero de dominio: Para definir una plantilla de creación de máquinas virtuales especificando sus características *hardware*.
- Conjunto de máquinas físicas: Para indicar qué máquinas físicas pueden ejecutar las máquinas virtuales definidas.

Estos atributos se obtienen mediante la observación de los elementos comunes a todo recurso sistema distribuido y forman, por tanto, el núcleo de un recurso *cloud* genérico. Además de estos atributos cada subtipo de recurso sistema distribuido (AppScale, TORQUE...) puede añadir los que considere necesarios para una completa especificación del recurso.

3.2.1 Patrón de diseño del proveedor

En Puppet, el proveedor es el encargado de llevar al recurso al estado que se le indique en el manifiesto. Típicamente el proveedor posee las funciones necesarias para crear un nuevo recurso y para destruirlo. Para llevar a cabo estas funciones en un recurso de tipo *cloud* el proveedor se apoya en tres grupos de funciones: puesta en marcha, monitorización y parada de un *cloud*. Las funciones de los dos primeros grupos se usan a la hora de crear un nuevo recurso mientras que las del último grupo se usan a la hora de parar un recurso ya existente.

Puppet no contempla la herencia entre tipos o entre proveedores de distintos tipos. Por lo tanto, las funciones anteriormente especificadas no se encuentran dentro de ningún proveedor concreto (no existe, como tal, el proveedor de un recurso *cloud* genérico), sino que se presentan como una clase Ruby con unas funcionalidades básicas que los distintos proveedores de recursos sistema distribuido pueden usar para facilitar el desarrollo de los mismos. Por ejemplo, esta clase tiene una función de inicio a la cual un proveedor le puede pasar como argumento la función de inicio del *cloud* concreto. Cuando se hayan hecho las operaciones de inicio básicas, se procederá a invocar la función de inicio que se pasó como argumento.

Las operaciones de puesta en marcha son las encargadas de poner en funcionamiento el *cloud* especificado en el manifiesto. Las más importantes son:

- Inicio como líder: Función de puesta en marcha que realizará el nodo líder. Ésta es la función más importante dentro de las funciones de inicio del proveedor ya que es la que se encarga de iniciar el *cloud*. A grandes rasgos, los pasos que realiza son:

  1. Comprobación de la existencia del *cloud*: si no existe se creará.
  2. Comprobación del estado del conjunto de máquinas físicas.
  3. Obtención de las direcciones IP de los nodos y los roles que les han sido asignados.
  4. Comprobación del estado de las máquinas virtuales: si están funcionando se monitorizan, mientras que si no están funcionando hay que definir una nueva máquina virtual y ponerla en funcionamiento. Las funciones de monitorización incluyen el envío de un fichero mediante el cual cada nodo se autoadministre la mayor parte posible.
  5. Cuando todas las máquinas virtuales estén funcionando se procede a inicializar el *cloud*.
  6. Operaciones de puesta en marcha particulares dependiendo de cada tipo de *cloud*.

- Inicio como nodo común: Función de puesta en marcha que realizarán los nodos comunes del *cloud*.

- Inicio como nodo externo: Función de puesta en marcha que realizará un nodo no perteneciente al *cloud*.

La monitorización únicamente la llevará a cabo uno de los nodos, al que llamaremos líder, ya que sería redundate que más de un nodo se encargara de comprobar el estado global del sistema. La elección de este nodo líder es necesaria para solucionar los problemas de disponibilidad que puedan aparecer. Como solamente el líder realizará la monitorización, sólo hay una función importante:

- Monitorización como líder: Función de monitorización que realizará el nodo líder del *cloud*.

Para elegir al nodo líder de entre todos los nodos del *cloud* se utiliza el algoritmo peleón. Para ayudar a la implementación de este algoritmo se proporcionan las funciones de elección de líder, de las cuales las más importantes son:

- Lectura y escritura de identificador: Funciones de lectura y escritura del identificador del nodo actual.

- Lectura y escritura de identificador de líder: Funciones de lectura y escritura del identificador del nodo líder.

- Escritura de identificador e identificador de líder remoto: Funciones de escritura del identificador y del identificador del líder en un nodo distinto del actual.

Por último, es posible que en algún momento se desee parar por completo el funcionamiento del *cloud*. Las operaciones de parada más importantes son:

- Apagado de máquinas virtuales: Función de apagado de las máquinas virtuales que forman el *cloud*.
- Borrado de ficheros: Función de eliminación de todos los ficheros internos de gestión.

Aunque no se proporcionan como funciones, hay que tener en cuenta que cada tipo de *cloud* puede tener sus propias funciones de parada. Estas funciones de parada deben realizarse antes de apagar las máquinas virtuales. De forma general, los pasos que hay que hacer a la hora de parar un *cloud* son:

1. Comprobación de la existencia del *cloud*: si existe se procederá a su parada.
2. Parada de las funciones de automantenimiento de los nodos.
3. Operaciones de parada particulares a cada tipo de *cloud*.
4. Apagado de las máquinas virtuales creadas explícitamente para este *cloud*.
5. Eliminación de los ficheros internos de gestión del *cloud*.

3.2.2 *Framework* de implementación

La mejor manera de crear un tipo y un proveedor en Puppet es hacerlo usando el concepto de módulo que ya dispone Puppet. Los módulos deben organizarse siguiendo una estructura predefinida. En esta estructura hay que colocar el código del tipo y del proveedor en la parte correspondiente. Si se crea el tipo `tipo` y el proveedor `proveedor` dentro del módulo `modulo` la estructura sería similar a la siguiente:

```
modulo/
modulo/manifests
modulo/manifests/init.pp
modulo/files
modulo/templates
modulo/lib/facter
modulo/lib/puppet/type
modulo/lib/puppet/type/tipo.rb
modulo/lib/puppet/provider
modulo/lib/puppet/provider/tipo/proveedor.rb
modulo/lib/puppet/parser/functions
```

Al ser el recurso sistema distribuido un recurso genérico, dicha organización es excesiva, puesto que carece de la mayoría de los elementos ahí definidos. Bastará por tanto con la siguiente estructura:

```
generic-cloud/provider
generic-cloud/provider/cloud.rb
```

Dentro del fichero `cloud.rb` se encuentra definida la clase `Cloud` que proporciona las funciones más básicas que ayudan al resto de recursos sistema distribuido a crear sus proveedores. En concreto, las operaciones de puesta en marcha que se proporcionan son `leader_start`, `common_start` y `not_cloud_start`; la función de monitorización que se proporciona es `leader_monitoring`; las funciones de elección de líder que se proporcionan son `get_id`, `set_id`, `get_leader`, `set_leader`, `vm_set_id` y `vm_set_leader`; por último, las funciones de parada de *cloud* que se proporcionan son `shutdown_vms` y `delete_files`. La implementación completa de la clase `Cloud` puede verse en el Anexo F.

Además de proporcionar estas funciones, la clase `Cloud` necesita de otras para realizar su trabajo. En particular, cada proveedor debe implementar dos funciones. La primera de ellas es la función `start_cloud` que se usa para iniciar el *cloud*. La segunda es la función `obtain_vm_data` que se utiliza para obtener datos de las máquinas virtuales que forman el *cloud*.

Para monitorizar el *cloud* concreto es necesario pasar una función como argumento a las funciones `leader_start` y `leader_monitoring`. Esta función debe implementar una monitorización que tenga en cuenta los aspectos particulares de cada tipo de *cloud*.

Capítulo 4

Metodología de diseño e implementación de recursos distribuidos

Una vez que el modelado se ha completado es tiempo de abarcar la metodología que se sigue a la hora de diseñar e implementar nuevos recursos distribuidos. Para ello se usan las facilidades que proporciona la herramienta de gestión de configuración para ampliar su funcionalidad.

4.1 Especificación del tipo

El primer paso que hay que hacer a la hora de crear un nuevo recurso en Puppet es definir el tipo de recurso. Para ello en un fichero habrá que especificar el nombre del nuevo tipo y los argumentos que éste tiene. Durante esta sección y la siguiente se usará el ejemplo de la arquitectura web de tres niveles ya que es la más conocida y sirve para demostrar que el enfoque que se ha tomado es lo suficientemente general y válido también para infraestructuras que nada tienen que ver con la ejecución de trabajos.

Una típica arquitectura de servicios web consta de al menos tres niveles: balanceo de carga, servidores web y base de datos. Cada uno de estos niveles está compuesto por al menos un elemento clave: el balanceador de carga, el servidor web y el servidor de base de datos, respectivamente. El balanceador de carga es el punto de entrada al sistema y el que se encarga, como su nombre indica, de repartir las peticiones de los clientes a los distintos servidores web. Los servidores web se encargan de servir las páginas web a los clientes y para ello, dependiendo de las peticiones que hagan los clientes, podrán leer o almacenar información en la base de datos. Para manipular dicha información los servidores web tendrán que comunicarse con el servidor de base de datos, que es el que hará efectiva la lectura y modificación de la información.

La infraestructura usada como ejemplo consta de un balanceador de carga, dos servidores web y un servidor de bases de datos (Figura 4.1).

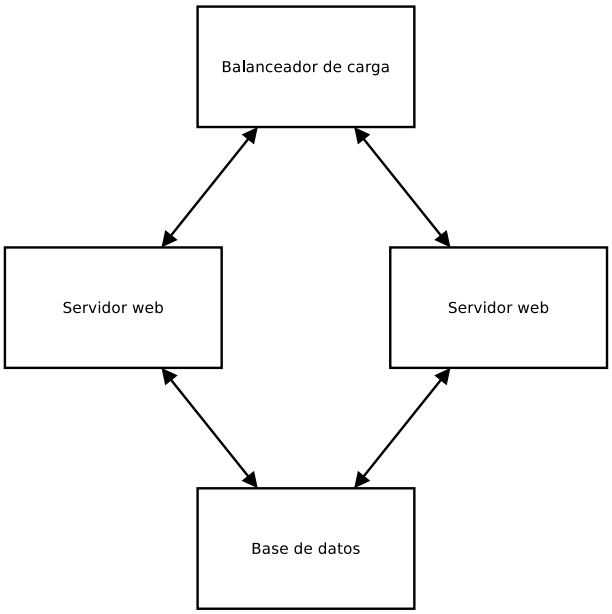

Figura 4.1: Infraestructura web de tres niveles.

La creación del tipo `web` se hace en el fichero `web.rb`, en el lugar apropiado dentro del correspondiente módulo. En el siguiente fragmento de código se indican los aspectos más importantes (la implementación completa puede encontrarse en el Anexo F) del mismo:

```
1  Puppet::Type.newtype(:web) do
2     @doc = "Manages web clouds formed by KVM virtual machines."
3
4
5     ensurable do
6
7        desc "The cloud's ensure field can assume one of the following values:
8     'running': The cloud is running.
9     'stopped': The cloud is stopped.\n"
10
11       newvalue(:stopped) do
12          provider.stop
13       end
14
15       newvalue(:running) do
16          provider.start
17       end
18
19    end
20
21
22    # General parameters
23
24    newparam(:name) do
25       desc "The cloud name"
26       isnamevar
```

```
27      end
28
29      newparam(:vm_domain) do
30         desc "The XML file with the virtual machine domain definition. " +
31              "Libvirt XML format must be used"
32      end
33
34      newproperty(:pool, :array_matching => :all) do
35         desc "The pool of physical machines"
36      end
37
38      # ...
39
40
41      # Web parameters
42
43      newproperty(:balancer, :array_matching => :all) do
44         desc "The balancer node's information"
45      end
46
47      newproperty(:server, :array_matching => :all) do
48         desc "The server nodes' information"
49      end
50
51      newproperty(:database, :array_matching => :all) do
52         desc "The database node's information"
53      end
54
55  end
```

Dentro del tipo `web`, los atributos `name`, `vm_domain` y `pool` se corresponden con los atributos `Nombre`, `Fichero de dominio` y `Conjunto de máquinas físicas` definidos en la sección 3.2. Los atributos `balancer`, `server` y `database` son los atributos propios de la arquitectura web.

Una vez que el tipo está definido, ya se puede ver cuál será el aspecto de un manifiesto para un recurso de tipo web:

```
web {'mycloud':
    balancer  => ["155.210.155.175", "/var/tmp/dceresuela/lucid-lb.img"],
    server    => ["/etc/puppet/modules/web/files/server-ips.txt",
                  "/etc/puppet/modules/web/files/server-imgs.txt"],
    database  => ["155.210.155.177", "/var/tmp/dceresuela/lucid-db.img"],
    vm_domain => "/etc/puppet/modules/web/files/mycloud-template.xml",
    pool      => ["155.210.155.70"],
    ensure    => running,
}
```

4.2 Diseño e implementación del proveedor

Después de haber definido el tipo del recurso, queda definir el proveedor que implementa dicho tipo. En el caso de los recursos distribuidos, y teniendo en cuenta las funciones de arranque y parada especificadas en el tipo (en el bloque `ensurable`, líneas 5 a 19), el proveedor deberá contener obligatoriamente las funciones `start` y `stop`. A continuación se presenta un proveedor con los aspectos más importantes (la implementación completa se encuentra en el Anexo F):

```
1  Puppet::Type.type(:web).provide(:webp) do
2     desc "Manages web clouds formed by KVM virtual machines"
3
4     # Require files
5     # ...
6
7     # Start function. Makes sure the cloud is running.
8     def start
9
10       cloud = Cloud.new(CloudInfrastructure.new(), CloudLeader.new(), resource,
11                         method(:err))
12       puts "Starting cloud %s" % [resource[:name]]
13
14       # Check existence
15       if !exists?
16          # Cloud does not exist => Startup operations
17
18          # Check pool of physical machines
19          puts "Checking pool of physical machines..."
20          pm_up, pm_down = cloud.check_pool()
21          unless pm_down.empty?
22             puts "Some physical machines are down"
23             pm_down.each do |pm|
24                puts " - #{pm}"
25             end
26          end
27
28          # Obtain the virtual machines' IPs
29          puts "Obtaining the virtual machines' IPs..."
30          vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data(cloud.resource)
31
32          # Check whether you are one of the virtual machines
33          puts "Checking whether this machine is part of the cloud..."
34          part_of_cloud = vm_ips.include?(MY_IP)
35          if part_of_cloud
36             puts "#{MY_IP} is part of the cloud"
37
38             # Check if you are the leader
39             if cloud.leader?()
40                cloud.leader_start("web", vm_ips, vm_ip_roles, vm_img_roles,
41                                   pm_up, method(:web_monitor))
42             else
```

```
43                cloud.common_start()
44             end
45          else
46             puts "#{MY_IP} is not part of the cloud"
47             cloud.not_cloud_start("web", vm_ips, vm_ip_roles, vm_img_roles,
48                                   pm_up)
49          end
50
51       else
52
53          # Cloud exists => Monitoring operations
54          puts "Cloud already started"
55
56          # Check if you are the leader
57          if cloud.leader?()
58             cloud.leader_monitoring(method(:web_monitor))
59          else
60             puts "#{MY_IP} is not the leader"      # Nothing to do
61          end
62       end
63
64    end
65
66
67    # Stop function. Makes sure the cloud is not running.
68    def stop
69
70       # ...
71
72    end
73
74
75    def status
76       if File.exists?("/tmp/cloud-#{resource[:name]}")
77          return :running
78       else
79          return :stopped
80       end
81    end
82
83 end
```

Son especialmente importantes dentro de este proveedor las líneas 10, 30 y 40. En la línea 10 creamos un nuevo objeto de la clase `Cloud` (3.2.2). En la línea 30 se llama a la función `obtain_vm_data` para obtener los datos de las máquinas virtuales (3.2.2). En la línea 40 puede verse como el objeto de la clase `Cloud` se usa para llamar al método `leader_start` que realizará las funciones de puesta en marcha como nodo líder del *cloud* (3.2.1). Uno de los argumentos del método en esa llamada es la función `web_monitor` que se usará para monitorizar los aspectos particulares del *cloud* web (3.2.2).

Merece la pena comentar que tanto para la implementación del tipo como para la del proveedor Puppet hace uso de la capacidad de metaprogramación que ofrece Ruby. La implementación de cada uno de ellos se pasa como un bloque de código Ruby a una función de Puppet (`newtype` para el tipo y `provide` para el proveedor) que se encarga internamente de realizar las operaciones necesarias. Es por este motivo que en el código del tipo y del proveedor aparecen variables (como `resource` y `provider`) y funciones (como `newparam` y `desc`) que no pertenecen a la sintaxis del lenguaje Ruby y no se encuentran *a priori* definidas en ninguna parte. Gracias a la ejecución dinámica de Ruby sabemos que estas variables y funciones estarán definidas en el entorno de Puppet en tiempo de ejecución.

Capítulo 5
Diseño e implementación de recursos distribuidos

En este capítulo se muestra cómo aplicar la metodología anteriormente expuesta para diseñar e implentar los recursos distribuidos de las infraestructuras de ejecución de trabajos AppScale y TORQUE.

5.1 Diseño e implementación de un recurso distribuido para una infraestructura AppScale

Una infraestructura AppScale puede ser definida de dos maneras: mediante un despliegue por defecto o uno personalizado. En un despliegue por defecto un nodo es el encargado de controlar la infraestructura y el resto de nodos se encargan de hacer el resto del trabajo. En un despliegue personalizado podemos especificar con mayor grado de precisión qué tipo de trabajo debe hacer cada nodo. Por ejemplo, podemos indicar qué nodos se encargarán de alojar las aplicaciones de los usuarios, qué nodos alojarán la base de datos o qué nodos serán los encargados de ejecutar los trabajos de computación. Para administrar una infraestructura AppScale, sin importar el tipo de despliegue, necesitaremos una cuenta de correo y una contraseña. Este usuario y contraseña son necesarios para poder administrar las aplicaciones alojadas y observar el estado de la infraestructura.

En un despliegue por defecto los posibles roles que puede tomar un nodo son:

`controller` : La máquina que desempeñará el rol de nodo controlador.

`servers` : La lista de máquinas que desempeñarán el rol de nodos de trabajo.

Mientras que los posibles roles que puede desempeñar un nodo en un despliegue personalizado y que resultan interesantes desde nuestro punto de vista son:

`master` : La máquina que desempeñará el rol de nodo maestro.

`appengine` : Los servidores para alojar las aplicaciones.

`database` : Las máquinas que contienen la base de datos.

`login` : La máquina encargada de redirigir a los usuarios a sus servidores. Es también la que se le facilita al administrador de la infraestructura para que realice las tareas administrativas.

`open` : Las máquinas de ejecución de trabajos. También pueden ser usadas como nodos de reserva por si falla algún otro nodo.

Hay multitud de despliegues posibles combinando estos roles, pero será de especial interés para este proyecto el que permite ejecutar trabajos de computación en AppScale. La infraestructura usada como ejemplo consta de un nodo `master`, `appengine`, `database` y `login` y dos nodos `open` (Figura 5.1).

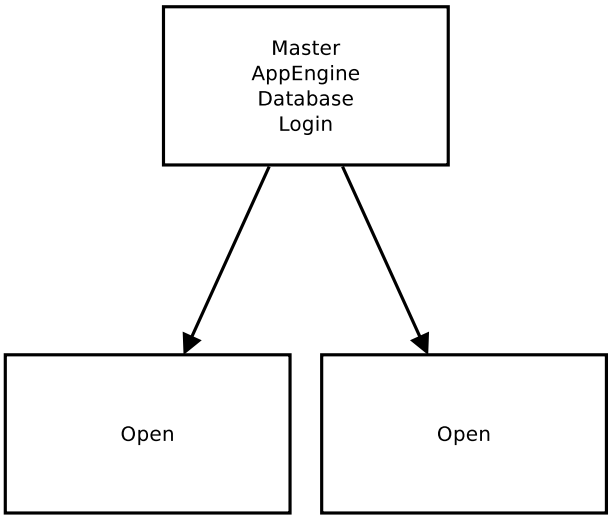

Figura 5.1: Infraestructura AppScale en despliegue personalizado.

Además de los atributos necesarios para los roles de las máquinas, el tipo `appscale` también debe contar con la cuenta de correo y la contraseña para el administrador del *cloud*. Los atributos más importantes del tipo `appscale` (ver Anexo F para implementación completa), en lo que a ejecución de trabajos concierne, son:

```
1  Puppet::Type.newtype(:appscale) do
2     @doc = "Manages AppScale clouds formed by KVM virtual machines."
3
4     # ...
5
6     # AppScale parameters
7
8     newproperty(:master, :array_matching => :all) do
9        desc "The master node"
10    end
11
12    newproperty(:appengine, :array_matching => :all) do
13       desc "The appengine nodes"
14    end
15
16    newproperty(:database, :array_matching => :all) do
17       desc "The database nodes"
18    end
19
20    newproperty(:login, :array_matching => :all) do
```

```
21        desc "The login node"
22     end
23
24     newproperty(:open, :array_matching => :all) do
25        desc "The open nodes"
26     end
27
28
29     newparam(:app_email) do
30        desc "AppScale administrator e-mail"
31        defaultto "david@gmail.com"
32     end
33
34     newparam(:app_password) do
35        desc "AppScale administrator password"
36        defaultto "appscale"
37     end
38
39  end
```

Para lograr un despliegue de este tipo podría utilizarse un manifiesto similar a éste:

```
appscale {'mycloud':
   master    => ["155.210.155.73",
                 "/var/tmp/dceresuela/lucid-tor1.img"],
   appengine => ["155.210.155.73",
                 "/var/tmp/dceresuela/lucid-tor1.img"],
   database  => ["155.210.155.73",
                 "/var/tmp/dceresuela/lucid-tor1.img"],
   login     => ["155.210.155.73",
                 "/var/tmp/dceresuela/lucid-tor1.img"],
   open      => ["/etc/puppet/modules/appscale/files/open-ips.txt",
                 "/etc/puppet/modules/appscale/files/open-imgs.txt"],
   vm_domain => "/etc/puppet/modules/appscale/files/mycloud-template.xml
      ",
   pool => ["155.210.155.70"],
   ensure => running,
}
```

5.2 Diseño e implementación de un recurso distribuido para una infraestructura TORQUE

Una infraestructura TORQUE está formada por un nodo maestro y un conjunto de nodos de computación (Figura 5.2). El nodo maestro es el encargado de recibir los trabajos a ejecutar y de asegurar una correcta planificación para esos trabajos; en su versión más simple, el planificador es una cola FIFO. Los nodos de computación son los encargados de ejecutar los trabajos enviados por el nodo maestro y, una vez terminados, enviarle los resultados de vuelta.

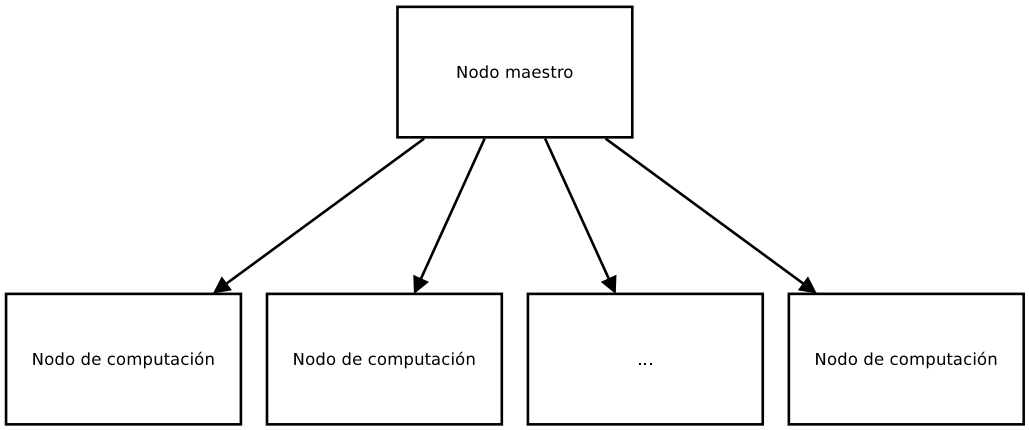

Figura 5.2: Infraestructura Torque.

Los nuevos atributos que aparecen en el tipo `torque` reflejan esta nueva infraestructura:

```
1  Puppet::Type.newtype(:torque) do
2     @doc = "Manages Torque clouds formed by KVM virtual machines."
3
4     # ...
5
6     # Torque parameters
7     newproperty(:head, :array_matching => :all) do
8        desc "The head node's information"
9     end
10
11    newproperty(:compute, :array_matching => :all) do
12       desc "The compute nodes' information"
13    end
14
15 end
```

La sintaxis del manifiesto de puesta en marcha también se modifica para reflejar este cambio. Un ejemplo de un manifiesto para la puesta en marcha de una infraestructura TORQUE sería similar a éste:

```
torque {'mycloud':
   head     => ["155.210.155.73", "/var/tmp/dceresuela/lucid-tor1.img"],
   compute  => ["/etc/puppet/modules/torque/files/compute-ips.txt",
                "/etc/puppet/modules/torque/files/compute-imgs.txt"],
   vm_domain => "/etc/puppet/modules/torque/files/mycloud-template.xml",
   pool       => ["155.210.155.70"],
   ensure     => running,
}
```

Capítulo 6
Validación de la solución planteada

Para validar la solución desarrollada, se ha hecho uso del laboratorio 1.03b que el Departamento de Informática e Ingeniería de Sistemas posee en la Escuela de Ingeniería y Arquitectura de la Universidad de Zaragoza. Los ordenadores de este laboratorio poseen procesadores con soporte para virtualización, lo que hace posible la creación de máquinas virtuales para simular los nodos que forman cada una de las infraestructuras distribuidas.

Antes de empezar con las pruebas hay que configurar el entorno en el que se realizarán. En particular, y dado que las máquinas virtuales necesitan conectarse a internet para la descarga e instalación de paquetes, el uso de un servidor de DNS es bastante recomendable. De este modo, podemos usar direcciones IP públicas (por ejmplo, las de las máquinas del laboratorio que no se estén usando en ese momento) para nuestras máquinas virtuales. El servidor DNS se usa también para hacer la resolución de nombres, tanto normal como inversa, que requieren AppScale y TORQUE para su correcto funcionamiento.

6.1 Pruebas comunes a todas las infraestructuras

En todas y cada una de las infraestructuras se han realizado las siguientes pruebas para comprobar el correcto funcionamiento del proveedor distribuido:

- Apagado de una máquina virtual que había empezado encendida y no era líder: el líder provee una nueva máquina virtual que sustituye a la que ha fallado.
- Apagado de una máquina virtual que no había empezado encendida y no era líder: el líder provee una nueva máquina virtual que sustituye a la que ha fallado.
- Apagado de una máquina virtual que había empezado encendida y era líder: se elige a un nuevo líder el cual provee una nueva máquina virtual que sustituye a la que ha fallado.
- Puesta en marcha de la infraestructura desde una máquina que no pertenece al *cloud*: se informa de qué máquina perteneciente al *cloud* se ha iniciado para continuar la puesta en marcha desde ella.
- Puesta en marcha de la infraestructura desde una máquina que pertenece al *cloud*: se crea el *cloud*.

Estas pruebas tienen como objetivo simular los fallos que se podrían encontrar en una ejecución en un entorno de producción para demostrar la tolerancia a fallos del sistema.

6.2 Pruebas específicas de la infraestructura AppScale

Para probar la infraestructura AppScale se han usado tres máquinas virtuales alojadas en dos máquinas físicas. Una de las máquinas virtuales actúa como nodo maestro y las otras dos actúan como nodos abiertos. Además de las pruebas comunes a todas las infraestructuras se han realizado una serie de pruebas particulares para comprobar el provedor de la infraestructura AppScale:

- Parada del proceso controlador: el proceso que lo está monitorizando (god E.10) se da cuenta y crea un nuevo proceso controlador.
- Parada del proceso que monitoriza al controlador: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.

6.3 Pruebas específicas de la infraestructura TORQUE

Para probar la infraestructura TORQUE se han usado cuatro máquinas virtuales alojadas en dos máquinas físicas. Una de las máquinas virtuales actúa como nodo maestro y las otras tres actúan como nodos de computación. Además de las pruebas comunes, para comprobar el provedor de la infraestructura TORQUE se han realizado las siguientes pruebas:

- Parada del proceso de autenticación (trqauthd) en el nodo maestro: el proceso que lo está monitorizando (god) se da cuenta y crea un nuevo proceso de autenticación.
- Parada del proceso servidor (pbs_server) en el nodo maestro: el proceso que lo está monitorizando (god) se da cuenta y crea un nuevo proceso servidor.
- Parada del proceso planificador (pbs_sched) en el nodo maestro: el proceso que lo está monitorizando (god) se da cuenta y crea un nuevo proceso planificador.
- Parada del proceso de ejecución de trabajos (pbs_mom) en un nodo de computación: el proceso que lo está monitorizando (god) se da cuenta y crea un nuevo proceso de ejecución de trabajos.
- Parada del proceso que monitoriza al proceso de autenticación en el nodo maestro: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.
- Parada del proceso que monitoriza al proceso servidor en el nodo maestro: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.
- Parada del proceso que monitoriza al proceso planificador en el nodo maestro: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.
- Parada del proceso que monitoriza al proceso de ejecución de trabajos en un nodo de computación: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.

6.4 Pruebas específicas de la infraestructura web de tres niveles

Para probar la infraestructura web se han usado cuatro máquinas virtuales repartidas entre dos máquinas físicas. Una máquina virtual actúa como balanceador de carga, dos actúan como servidores web y la última actúa como base de datos. Las pruebas que se han realizado para comprobar el correcto funcinamiento del proveedor de la infraestructura web han sido:

- Parada del proceso balanceador de carga: el proceso que lo está monitorizando (puppet) se da cuenta y crea un nuevo proceso balanceador de carga.
- Parada del proceso servidor web: el proceso que lo está monitorizando (puppet) se da cuenta y crea un nuevo proceso servidor web.
- Parada del proceso base de datos: el proceso que lo está monitorizando (god) se da cuenta y crea un nuevo proceso base de datos.
- Parada del proceso que monitoriza al proceso balanceador de carga: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.
- Parada del proceso que monitoriza al proceso servidor web: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.
- Parada del proceso que monitoriza al base de datos: el proveedor (puppet) se da cuenta y crea un nuevo proceso de monitorización.

Capítulo 7
Conclusiones

En este proyecto fin de carrera se ha creado un modelo de recursos distribuidos para facilitar la puesta en marcha y la automatización de distintas infraestructuras distribuidas.

Para comprobar la validez del modelo creado se ha extendido la herramienta de gestión de configuración Puppet añadiéndole el recurso sistema distribuido o *cloud*. A la hora de definir el recurso sistema distribuido se han tenido en cuenta las características propias de este tipo de recursos, como la disponibilidad, las prestaciones y las dependencias y los conceptos de iteración y convergencia que las herramientas de configuración proveen.

Posteriormente se ha comprobado la extensión realizada haciendo que la herramienta de gestión de configuración Puppet sea capaz de administrar infraestructuras distribuidas de ejecución de trabajos como AppScale y TORQUE. Sin embargo no se han descuidado el resto de aspectos de estas infraestructuras. En concreto, en la infraestructura AppScale se soportan los dos tipos de despliegue y la totalidad de los roles posibles, siendo por lo tanto capaz de gestionar al completo una infraestructura de este tipo, y no sólo la parte de ejecución de trabajos.

Además, no se ha comprobado la validez del modelo exclusivamente con infraestructuras de ejecución de trabajos, sino que se ha constatado que también es válido para infraestructuras más generales como por ejemplo la de servicios web en tres niveles. Se puede decir, por consiguiente, que se ha ido más allá del mero cumplimiento de los objetivos del proyecto.

En cuanto al trabajo futuro, al ser este un primer prototipo, se encuentra abierto a multiples ampliaciones. El paso más evidente sería integrar este modelo dentro de la herramienta Puppet. Sería especialmente interesante modificar la gramática de descripción de recursos para diferenciar entre los recursos locales y los distribuidos. Posteriormente, se deberían incorporar las funcionalidades básicas de gestión de recursos distribuidos. De esta manera los futuros administradores de este tipo de infraestructuras contarían de manera automática con un conjunto de funciones que ayudan en la administración de sistemas distribuidos en Puppet.

La validez del trabajo realizado se ha comprobado únicamente sobre máquinas virtuales con un sistema operativo con núcleo Linux. Son de sobra conocidas las diferencias que existen entre las distintas distribuciones de Linux, por no hablar ya de las diferencias entre sistemas operativos completamente distintos. Realizar proveedores que sean válidos para cualquier sistema operativo es el segundo paso que debería realizarse. Además, estas máquinas virtuales se alojan en ordenadores de un laboratorio en la EINA; utilizar máquinas virtuales alojadas en proveedores como Amazon o Rackspace supondría un broche de oro.

Por último, y a modo de valoración personal, este proyecto me ha permitido aprender una gran cantidad de nuevas tecnologías; conocer una herramienta de gestión de configuración como Puppet, que es de las que más relevancia está tomando; estudiar las infraestructuras de AppScale, TORQUE y de servicios web e introducirme en un nuevo lenguaje de programación como Ruby que aporta conceptos muy interesantes y distintos a los lenguajes estudiados a lo largo de la carrera. Aunque en ocasiones la documentación fuese más escasa de lo deseado, la oportunidad de aprender nuevos conceptos y tecnologías es algo que personalmente valoro muy positivamente de este proyecto.

Bibliografía

[1] Puppet labs: The leading open source data center automation solution. http://www.puppetlabs.com/.

[2] AppScale: An open-source implementation of the Google AppEngine (GAE) cloud computing interface. http://appscale.cs.ucsb.edu/, 2011.

[3] AppScale: EC2 API Documentation. http://code.google.com/p/appscale/wiki/EC2_API_Documentation, 2011.

[4] AppScale: MapReduce API Documentation. http://code.google.com/p/appscale/wiki/MapReduce_API_Documentation, 2011.

[5] AppScale: Neptune API Documentation. http://www.neptune-lang.org/, 2011.

[6] Amazon DynamoDB. http://aws.amazon.com/dynamodb/, 2012.

[7] Amazon: Elastic Compute Cloud. http://aws.amazon.com/ec2/, 2012.

[8] BerkeleyDB. http://www.oracle.com/us/products/database/berkeley-db/overview/index.html, 2012.

[9] BigTable. http://research.google.com/archive/bigtable.html, 2012.

[10] Cassandra. http://cassandra.apache.org/, 2012.

[11] euca2ools. http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3, 2012.

[12] Eucalyptus. http://www.eucalyptus.com/, 2012.

[13] Google: App Engine. https://developers.google.com/appengine/, 2012.

[14] Hadoop. http://hadoop.apache.org/, 2012.

[15] HBase. http://hbase.apache.org/, 2012.

[16] Hypertable. http://hypertable.com/, 2012.

[17] MemcacheD. http://memcached.org/, 2012.

[18] MemcacheDB. http://memcachedb.org/, 2012.

[19] MongoDB. http://www.mongodb.org/, 2012.

[20] MySQL Cluster. http://www.mysql.com/products/cluster/, 2012.

[21] Redis. http://redis.io/, 2012.

[22] Voldemort. `http://project-voldemort.com/voldemort/`, 2012.

[23] Chris Bunch, Navraj Chohan, Chandra Krintz, and Khawaja Shams. Neptune: a domain specific language for deploying hpc software on cloud platforms. In *Proceedings of the 2nd international workshop on Scientific cloud computing*, ScienceCloud '11, pages 59–68, New York, NY, USA, 2011. ACM.

[24] Navraj Chohan, Chris Bunch, Sydney Pang, Chandra Krintz, Nagy Mostafa, Sunil Soman, and Richard Wolski. Appscale: Scalable and open appengine application development and deployment. In *CloudComp*, pages 57–70, 2009.

[25] Adaptative Computing. TORQUE Resource Manager. `http://www.adaptivecomputing.com/products/open-source/torque/`, 2012.

[26] LaTeX project team. *LaTeX documentation.* `http://www.latex-project.org/guides/`.

[27] Puppet Labs. Puppet Faces API. `http://puppetlabs.com/faces/`, 2012.

[28] Garrick Staples. TORQUE resource manager. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, SC '06, New York, NY, USA, 2006. ACM.

[29] David Thomas, Chad Fowler, and Andrew Hunt. *Programming Ruby. The Pragmatic Programmer's Guide.* Pragmatic Programmers, 2004.

[30] J. Turnbull and J. McCune. *Pro Puppet.* Pro to Expert Series. Apress, 2011.

Anexo A
Gestión del proyecto

La realización de este proyecto comenzó a finales de septiembre de 2011 y se ha prolongado hasta agosto de 2012. A lo largo del mismo se han atravesado las fases de documentación, desarrollo, pruebas y redacción de la memoria. A continuación se muestra una tabla con las distintas fases y el coste en horas de cada una de ellas:

Fase	Fecha de inicio	Fecha de finalización	Número de horas
Documentación	29/09/2011	08/12/2011	148
Desarrollo	13/12/2011	30/08/2012	385
Pruebas	02/05/2012	30/08/2012	36
Memoria	06/03/2012	30/08/2012	75
Número total de horas	29/09/2011	30/08/2012	644

Tabla A.1: Número de horas invertidas.

Para ver el desarrollo del proyecto a lo largo de los meses se incluye a continuación un diagrama de Gantt:

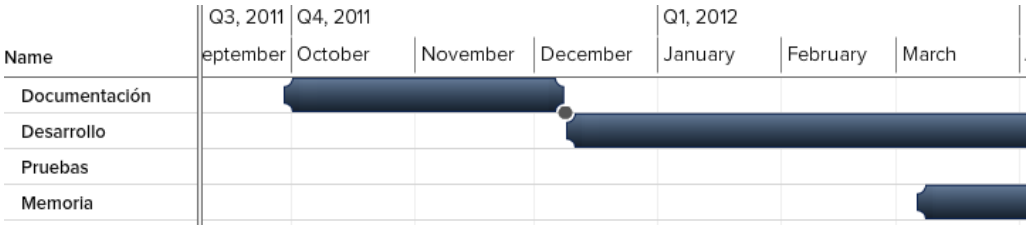

Figura A.1: Diagrama de Gantt: septiembre a febrero.

Figura A.2: Diagrama de Gantt: marzo a agosto.

Como se puede observar, la fase del proyecto que más tiempo ha tomado ha sido la de

desarrollo. En esta fase hubo tres cuestiones que tomaron bastante tiempo.

La primera de ellas fue la falta de documentación en AppScale. A la hora de averiguar cómo funcionaba Neptune (el lenguaje usado para enviar los trabajos MPI), ni la documentación que se podía encontrar en la página web, ni la documentación a la que se podía acceder una vez instalado eran suficientes. Después de enviar un correo electrónico preguntando por la información necesaria para poder trabajar con Neptune se creó la página web que ahora alberga la información relativa a los distintos trabajos que existen en Neptune.

La segunda de ellas fue un fallo en la parte de ejecución de los trabajos MPI. A la hora de copiar al nodo correspondiente el archivo que debía ejecutarse, no se tenían en cuenta los permisos de ejecución y, por lo tanto, se intentaba ejecutar un archivo sin los permisos necesarios. Encontrar y solucionar este bug consumió una buena parte de tiempo.

La tercera cuestión fue un fallo en el planificador de TORQUE. El planificador más básico que usa TORQUE, una cola FIFO, simplemente no funciona en la versión 4.0.2 y los trabajos nunca se llegaban a ejecutar. Aunque actualizar a la versión 4.0.3 solucionó el problema, no es un tipo de fallo que uno espera encontrar, y el hecho de ser una tecnología con la que nunca antes había trabajado no ayudó mucho en este aspecto.

Anexo B
Ejemplo de uso

A continuación se mostrará un ejemplo de cómo poner en marcha y parar una infraestructura web de tres niveles. Por simplicidad, todos los nodos que van a formar parte de la infraestructura ya están encendidos y listos. En primer lugar, tenemos el manifiesto que describe el recurso:

```
web {'mycloud':
   balancer  => ["155.210.155.175", "/var/tmp/dceresuela/lucid-lb.img"],
   server    => ["/etc/puppet/modules/web/files/server-ips.txt", "/etc/
      puppet/modules/web/files/server-imgs.txt"],
   database  => ["155.210.155.177", "/var/tmp/dceresuela/lucid-db.img"],
   vm_domain => "/etc/puppet/modules/web/files/mycloud-template.xml",
   pool      => ["155.210.155.70"],
   ensure    => running,
}
```

Al aplicar el manifiesto de puesta en marcha en uno de los nodos que forman la infraestructura ocurre lo siguiente:

```
root@lucid-web:/etc/puppet/modules/web/manifests# puppet apply init-web.
   pp
Starting cloud mycloud
Checking pool of physical machines...
Obtaining the virtual machines' IPs...
Obtaining virtual machines' data
Checking whether this machine is part of the cloud...
155.210.155.73 is part of the cloud
155.210.155.73 is not the leader
155.210.155.73 will be the leader
Creating /root/cloud/ssh directory...
Deleting previous keys...
Generating key...
Evaluating agent and adding identity...
Identity added: /root/cloud/ssh/id_rsa (/root/cloud/ssh/id_rsa)
notice: /Stage[main]//Web[mycloud]/ensure: created
notice: Finished catalog run in 0.39 seconds
```

En esta primera ejecución, el nodo se ha dado cuenta de que el recurso no existe y que no había ningún líder. Después se ha dado cuenta de que él debe ser el nodo líder. Ejecutamos el mismo comando una vez más y ahora ocurre esto:

```
root@lucid-web:/etc/puppet/modules/web/manifests# puppet apply init-web.
   pp
Starting cloud mycloud
Checking pool of physical machines...
Obtaining the virtual machines' IPs...
Obtaining virtual machines' data
Checking whether this machine is part of the cloud...
155.210.155.73 is part of the cloud
155.210.155.73 is the leader
Checking whether virtual machines are alive...
Monitoring 155.210.155.175...
[CloudMonitor] 155.210.155.175 is up
Sending ssh key to 155.210.155.175
password is not empty, using ssh_copy_id.sh shell script
ssh key sent
[CloudMonitor] MCollective is running on 155.210.155.175
155.210.155.175 will perform the roles: [:balancer]
[Web monitor] Monitoring load balancer
[Web monitor] Monitored load balancer
...Monitored
Monitoring 155.210.155.73...
[CloudMonitor] 155.210.155.73 is up
Sending ssh key to 155.210.155.73
password is not empty, using ssh_copy_id.sh shell script
ssh key sent
MCollective is not running on 155.210.155.73
[CloudMonitor] MCollective is running now on 155.210.155.73
155.210.155.73 will perform the roles: [:server]
[Web monitor] Monitoring web server
[Web monitor] Monitored web server
...Monitored
Monitoring 155.210.155.178...
[CloudMonitor] 155.210.155.178 is up
Sending ssh key to 155.210.155.178
password is not empty, using ssh_copy_id.sh shell script
ssh key sent
MCollective is not running on 155.210.155.178
[CloudMonitor] MCollective is running now on 155.210.155.178
Sending path and content via MCollective Files client

 * [ ============================================================> ] 3 /
     3


Finished processing 3 / 3 hosts in 114.95 ms
155.210.155.178 will perform the roles: [:server]
[Web monitor] Monitoring web server
[Web monitor] Monitored web server
...Monitored
Monitoring 155.210.155.177...
[CloudMonitor] 155.210.155.177 is up
Sending ssh key to 155.210.155.177
password is not empty, using ssh_copy_id.sh shell script
ssh key sent
MCollective is not running on 155.210.155.177
```

```
[CloudMonitor] MCollective is running now on 155.210.155.177
Sending path and content via MCollective Files client

 * [ ============================================================> ] 4 /
     4


Finished processing 4 / 4 hosts in 177.19 ms
155.210.155.177 will perform the roles: [:database]
[Web monitor] Monitoring database
[Web monitor] God is not running in 155.210.155.177
[Web monitor] Starting monitoring database on 155.210.155.177
[Web monitor] Successfully started to monitor database on
   155.210.155.177
[Web monitor] Monitored database
...Monitored
Copying important files to all virtual machines
Starting the cloud
Starting a web cloud
Starting nginx on load balancers
Starting ruby web3 on web servers
Starting mysql on database servers
====================
== Cloud started ==
====================
notice: /Stage[main]//Web[mycloud]/ensure: created
notice: Finished catalog run in 76.41 seconds
```

El nodo líder ha puesto en marcha la infraestructura web de tres niveles correctamente.

A la hora de parar un recurso de tipo *cloud* también necesitamos un manifiesto que, en este caso, sera éste:

```
web {'mycloud':
   balancer => ["155.210.155.175", "/var/tmp/dceresuela/lucid-lb.img"],
   server   => ["/etc/puppet/modules/web/files/server-ips.txt", "/etc/
      puppet/modules/web/files/server-imgs.txt"],
   database => ["155.210.155.177", "/var/tmp/dceresuela/lucid-db.img"],
   pool     => ["155.210.155.70"],
   ensure   => stopped,
}
```

Y para detener la infraestructura simplemente tenemos que aplicar dicho manifiesto de parada:

```
root@lucid-web:/etc/puppet/modules/web/manifests# puppet apply stop-web.
   pp
Stopping cloud mycloud
It is a web cloud
Obtaining virtual machines' data
Stopping nginx on load balancers
Stopping ruby web3 on web servers
Stopping mysql on database servers
Copying ssh key to physical machine
password is empty, using ssh-copy-id command
```

```
scp: /tmp/defined-domains: No such file or directory
No /tmp/defined-domains file found in 155.210.155.70
Sending path and string via MCollective Cron client

 * [ ============================================================> ] 4 /
     4


Finished processing 4 / 4 hosts in 111.56 ms
Deleting cloud files on all machines...
Sending path via MCollective Files client

 * [ ============================================================> ] 4 /
     4


Finished processing 4 / 4 hosts in 94.15 ms
Sending path via MCollective Files client

 * [ ============================================================> ] 4 /
     4


Finished processing 4 / 4 hosts in 105.65 ms
Sending path via MCollective Files client

 * [ ============================================================> ] 4 /
     4


Finished processing 4 / 4 hosts in 90.45 ms
Sending path via MCollective Files client

 * [ ============================================================> ] 4 /
     4


Finished processing 4 / 4 hosts in 86.66 ms
Disconnecting MCollective client...
File /tmp/defined-domains does not exist
====================
== Cloud stopped ==
====================
notice: /Stage[main]//Web[mycloud]/ensure: ensure changed 'present' to '
   stopped'
notice: Finished catalog run in 15.38 seconds
```

Anexo C
Lenguaje de Puppet

Puppet es una herramienta de gestión de configuración basada en un lenguaje declarativo. Este lenguaje declarativo permite expresar de una manera clara la configuración deseada en un sistema. Para expresar dicha configuración un usuario indicará en un fichero, llamado manifiesto, cuál es el estado que deben alcanzar los recursos de ese sistema. Si, por ejemplo, un usario quisiera especificar cuál debe ser el contenido de un fichero, podría utilizar algo similar a lo siguiente:

```
file {'testfile':
  path    => '/tmp/testfile',
  ensure  => present,
  mode    => 0640,
  content => "I'm a test file.",
}
```

Entre los recursos de Puppet se encuentran, además del fichero, otros que pueden ser familiares, como usuario, grupo, paquete, servicio... Por ejemplo, un recurso paquete puede definirse de una forma similar a:

```
package {'vim':
  ensure  => present,
}
```

Además de la especificación de recursos, Puppet permite la agrupación de varios recursos en una entidad de nivel superior. A esta entidad en la terminología de Puppet se le denomina clase [1] y resulta útil cuando sobre un nodo se quieren aplicar varios recursos, por ejemplo un usuario y el grupo de este ususario:

```
class usergroup {

  user { 'david':
    ensure => present,
    comment => "My user",
    gid => "david",
    shell => "/bin/bash",
    require => Group["david"],
  }

  group { 'david':
```

[1] Aunque el nombre sea el mismo este concepto no es idéntico al usado en la programación orientada a objetos. En Puppet una clase debe verse como una colección de recursos.

```
    ensure => present,
  }

}
```

Asimismo el lenguaje declarativo de Puppet presenta elementos comunes a otros lenguajes de programación, como las variables y las estructuras de control, para mejorar la lógica y la flexibilidad del lenguaje. Estos conceptos pueden verse aplicados en el siguiente manifiesto en el que las variables `$operatingsystem` y `$puppetserver` estarían predefinidas:

```
class sudo {

  package { sudo:
    ensure => present,
  }

  if $operatingsystem == "Ubuntu" {
      package { "sudo-ldap":
      ensure => present,
      require => Package["sudo"],
    }
  }

  file { "/etc/sudoers":
    owner => "root",
    group => "root",
    mode => 0440,
    source => "puppet://$puppetserver/modules/sudo/etc/sudoers",
    require => Package["sudo"],
  }
}
```

Anexo D
AppScale

En este anexo se realiza un análisis más detallado de algunos aspectos de AppScale tales como la instalación de AppScale, los distintos tipos de roles que puede tomar un nodo y los distintos tipos de despliegue que podemos especificar, las bases de datos e infraestructuras que permite usar y el desarrollo y organización del *software*.

D.1 Roles y despliegue

A continuación se explica de manera más detallada los distintos roles que puede desempeñar un nodo dentro de la arquitectura AppScale.

En primer lugar tenemos los roles simples:

`Shadow` : Comprueba el estado en el que se encuentran el resto de nodos y se asegura de que están ejecutando los servicios que deben.

`Load balancer` : Servicio web que lleva a los usuarios a las aplicaciones. Posee también una página en la que informa del estado de todas las máquinas desplegadas.

`AppEngine` : Versión modificada de los SDKs de Google App Engine. Además de alojar las aplicaciones web añaden la capacidad de almacenar y recuperar datos de bases de datos que soporten la API de Google Datastore.

`Database` : Ejecuta los servicios necesarios para alojar a la base de datos elegida.

`Login` : La máquina principal que lleva a los usuarios a las aplicaciones App Engine. Difiere del Load Balancer en que esta es la única máquina desde la que se pueden hacer funciones administrativas. Puede haber muchas máquinas que hagan la función de Load Balancer pero sólo habrá una que haga función de Login.

`Memcache` : Proporciona soporte para almacenamiento en caché para las aplicaciones App Engine.

`Zookeeper` : Aloja los metadatos necesarios para hacer transacciones en las bases de datos.

`Open` : No ejecuta nada por defecto, pero está disponible en caso de que sea necesario. Estas máquinas son las utilizadas para ejecutar trabajos MPI.

Además de los roles simples también se proporcionan unos roles agregados que agrupan a varios de los roles simples y que facilitan la descripción de la arquitectura:

`Controller` : Shadow, Load Balancer, Database, Login y Zookeeper.

`Servers` : App Engine, Database y Load Balancer.

`Master` : Shadow, Load Balancer y Zookeeper.

Estos roles pueden usarse en dos tipos de despliegue: por defecto y personalizado. En un despliegue por defecto los nodos únicamente pueden ser de tipo `Controller` y `Servers`. En un despliegue personalizado los nodos pueden ser de cualquier tipo del resto de roles.

D.2 Bases de datos

AppScale no obliga a trabajar con una base de datos determinada, sino que es capaz de trabajar con varias bases de datos. Cada una de estas bases de datos implementa la interfaz AppScale DB, para comunicarse con las aplicaciones del App Engine. Las bases de datos que actualmente soporta AppScale son:

`Cassandra` [10]: Una base de datos híbrida entre BigTable [9] y Dynamo [6]. Es una base de datos de nodos idénticos (*peer-to-peer*) con una alta escalabilidad y rendimiento.

`HBase` [15]: Una implementación de BigTable realizada en Java. Se ejecuta sobre Hadoop [14].

`Hypertable` [16]: Otra implementación de BigTable, esta vez realizada en C++. Al igual que `HBase` se ejecuta sobre Hadoop. Tiene como objetivo el alto rendimiento.

`MongoDB` [19]: Una base de datos centrada en el almacén de documentos. Se comporta bien en operaciones de adición (*append*).

`MemcacheDB` [18]: Una base de datos clave-valor distribuida diseñada para ser persistente. Se basa en MemcacheD [17], una memoria cache distribuida, y usa la misma API que ella. La persistencia la obtiene a través de BerkeleyDB [8].

`MySQL Cluster` [20]: Es usada como una base de datos clave-valor dentro de AppScale.

`Voldemort` [22]: Es una base de datos clave-valor de nodos idénticos (*peer-to-peer*). Se basa en Dynamo y gestiona la persistencia a través de BerkeleyDB

`Redis` [21]: Base de datos clave-valor con durabilidad opcional.

La elección de la base de datos a la hora de iniciar AppScale se realiza en el comando `appscale-run-instances`. Los valores que acepta el comando son: `cassandra`, `hbase`, `hypertable`, `mongodb`, `memcachedb`, `mysql` y `voldemort`. La base de datos que usa por defecto es `Cassandra`, así que si queremos elegir otra, deberemos especificarlo claramente. Por ejemplo, en el caso de que eligiésemos `HBase` como base de datos habría que usar el comando `appscale-run-instances` de la siguiente manera:

```
appscale-run-instances --ips ips.yaml --table hbase
```

D.3 Infraestructuras

AppScale puede ser ejecutada sobre diversos tipos de infraestructuras públicas, privadas o híbridas. Dentro de las infraestructuras públicas AppScale soporta tanto Amazon EC2 [7] como otras infraestructuras públicas administradas mediante Eucalyptus [12], o de manera más general,

cualquier infraestructura que sea compatible con las euca2ools [11]. Dentro de las infraestructuras privadas soporta tanto un *cluster* de máquinas con Xen o KVM como infraestructuras privadas gestionadas mediante Eucalyptus. Una infraestructura híbrida puede estar formada por una combinación de infraestructura privada e infraestructura pública. Los casos que se pueden dar son:

- Cloud privado - Cloud público
- Clod privado X - Cloud privado Y
- Clod público (zona X) - Cloud público (zona Y)
- Clod público (empresa X) - Cloud público (empresa Y)

D.4 Desarrollo y organización

AppScale es una tecnología que actualmente se encuentra en un proceso activo de desarrollo. Cuando se empezó este proyecto la versión estable era la 1.5 y hasta ahora se ha seguido trabajando en esa versión; más concretamente en la versión 1.5.1. Otras partes de AppScale como Neptune, el lenguaje usado para la ejecución de trabajos MPI, han progresado desde la versión 0.1.1 hasta la versión 0.2.2 sufriendo severas modificaciones a lo largo del proceso.

Además de los cambios que está experimentando en el *software*, AppScale está realizando cambios en su organización interna. Al empezar este proyecto el código de AppScale se encontraba alojado en Launchpad (`https://launchpad.net/appscale/`) y la información en su página de Google (`http://code.google.com/p/appscale/`). Ahora se encuentran realizando la transición hacia Github (`https://github.com/AppScale/appscale`) y aunque todo el código puede encontrarse allí, la documentación sigue estando en su página de Google.

La complejidad del problema que trata de resolver, el amplio abanico de tecnologías utilizadas y los frecuentes cambios que sufre hacen que sea una tecnología complicada para trabajar con ella. La falta de documentación en alguna de las áreas no ayuda en este aspecto.

Anexo E
Instalación de tecnologías

E.1 AppScale

E.1.1 Instalación

Antes de instalar AppScale es muy recomendable actualizar el sistema operativo. Para ello añadimos las siguientes líneas al archivo `/etc/apt/sources.list`:

```
deb http://archive.ubuntu.com/ubuntu  lucid main restricted universe
    multiverse
deb http://archive.ubuntu.com/ubuntu  lucid-updates main restricted
    universe
deb http://security.ubuntu.com/ubuntu lucid-security main restricted
    universe
```

Y actualizamos el sistema operativo:

```
_: apt-get update
_: apt-get -y upgrade
```

Para gestionar una infraestructura AppScale es necesario instalar una serie de programas agrupados bajo el nombre de appscale-tools. Para ello, se debe descargar el archivo *tarball* de `http://code.google.com/p/appscale/downloads/list`. Una vez descargado, se instala:

```
_: tar xzvf appscale-tools.tar.gz
_: cd appscale-tools
_: sudo bash debian/appscale_build.sh
...
AppScale tools installation completed successfully!
```

Sin olvidarse de añadir el directorio de las appscale-tools a nuestro *path*:

```
_: export PATH=${PATH}:/usr/local/appscale-tools/bin
```

Una vez instaladas las appscale-tools instalaremos AppScale. Como lo que vamos a hacer va a ser clonar un repositorio, tenemos que asegurarnos de tener las herramientas necesarias, en este caso `git`:

```
_: apt-get -y install git-core
```

y cuando esté instalado clonamos el repositorio:

```
_: cd /root/
_: git clone git://github.com/AppScale/appscale.git
```

Accedemos a la carpeta recién creada y ejecutamos el *script* de instalación. La instalación tarda aproximadamente una hora:

```
_: cd appscale
_: bash debian/appscale_build.sh
...
AppScale installation completed successfully!
```

Nota: Si no quieres instalar todo AppScale puedes comentar las partes que no quieras en el *script* de `appscale_install.sh`. Por ejemplo, si no quieres que se instale la base de datos Voldemort, puedes comentar las líneas que hacen referencia a ella:

```
...
all)
# scratch install of appscale including post script.
installappscaleprofile
. /etc/profile.d/appscale.sh
installgems
postinstallgems
installsetuptools
installhaproxy
postinstallhaproxy
...
installcassandra
postinstallcassandra
###installvoldemort        # No queremos Voldemort
###postinstallvoldemort    # No queremos Voldemort
installhbase
postinstallhbase
...
```

E.1.2 Comprobación de la instalación

Para comprobar que AppScale se ha instalado correctamente lanzaremos una instancia. En primer lugar creamos un archivo llamado `ips.yaml` con el siguiente contenido (sustituye la dirección IP por la de tu máquina):

```
---
:controller: 155.210.155.73
```

A continuación lanza la instancia:

```
_: appscale-run-instances --ips ips.yaml
```

Si en tu navegador web vas a la dirección `http://155.210.155.73` (o la que corresponda en tu caso); allí deberías ver la página de inicio de sesión de AppScale.

E.1.3 Versiones instaladas

Software	Versión
Ubuntu	10.04
AppScale	1.5.1
appscale-tools	1.5

Tabla E.1: Versiones instaladas de AppScale.

E.2 TORQUE

E.2.1 Instalación de TORQUE

E.2.1.1 Nodo maestro

Primero instalaremos `libssl-dev`:

```
_: apt-get install libssl-dev
```

Una vez hecho esto debemos seguir las instrucciones del enlace `http://www.adaptivecomputing.com/resources/docs/torque/4-0-1/help.htm#topics/1-installConfig/installing.htm`, que básicamente son:

```
_: tar -xzvf torque-4.0.3.tar.gz
_: cd torque-4.0.3/
_: echo '/usr/local/lib' > /etc/ld.so.conf.d/torque.conf
_: ldconfig
_: ./configure
_: make
_: make install
```

E.2.1.2 Nodos de computación

Para crear los paquetes necesarios para los nodos de computación, haz lo siguiente:

```
_: cd torque-4.0.3/
_: make packages
```

Ahora se pueden instalar estos paquetes en los nodos de computación usando los *scripts* creados dentro del directorio `torque-4.0.3`:

```
_: ./torque-package-mom-linux-x86_64.sh --install
```

No hay que olvidarse de incluir `/usr/local/lib` en la lista de directorios para buscar las librerías enlazadas dinámicamente:

```
_: echo '/usr/local/lib' > /etc/ld.so.conf.d/torque.conf
_: ldconfig
```

Para habilitar TORQUE como un servicio se debe hacer lo siguiente:

```
_: cd torque-4.0.3
_: cp contrib/init.d/debian.pbs_mom /etc/init.d/pbs_mom
_: update-rc.d pbs_mom defaults
```

E.2.2 Configuración de TORQUE

E.2.2.1 Nodo maestro

Añade los nodos de computación en el fichero `/var/spool/torque/server_priv/nodes`. Por ejemplo:

```
## This is the TORQUE server "nodes" file.
##
## To add a node, enter its hostname, optional processor count (np=),
## and optional feature names.
##
## Example:
##    host01 np=8 featureA featureB
##    host02 np=8 featureA featureB
##
## for more information, please visit:
##
## http://www.clusterresources.com/torquedocs/nodeconfig.shtml

### Nodes
lucid-tor2
```

Posteriormente copia el fichero `debian.trqauthd` al directorio `/etc/init.d`:

```
_: cd torque-4.0.3/contrib/init.d
_: cp debian.trqauthd /etc/init.d
_: cd /etc/init.d
_: mv debian.trqauthd trqauthd
```

Puedes empezar el demonio con:

```
_: /etc/init.d/trqauthd start
_: /usr/bin/service trqauthd start
```

y puedes pararlo con:

```
_: /etc/init.d/trqauthd stop
_: /usr/bin/service trqauthd stop
```

Inicializa `serverdb`:

```
_: cd torque-4.0.3/
_: ./torque.setup <user>   # Asegurate de que el usuario <user> existe en cada
                           # nodo, maestro o de computacion.
```

E.2.2.2 Comprobación de la configuración

Reiniciamos `pbs_server` en el nodo maestro:

```
_: qterm -t quick
_: pbs_server
```

Y `pbs_mom` en los nodos de computación:

```
_: pbs_mom
```

Después de esperar durante un tiempo prudencial, el comando `pbsnodes -a` proporcionará una lista de nodos libres:

```
_: pbsnodes -a
lucid-tor2
     state = free
     np = 1
     ntype = cluster
     status = rectime=1338548162,varattr=,jobs=,state=free,netload=2359153,
              gres=,loadave=0.03,ncpus=2,physmem=1023292kb,availmem=1457904kb,
              totmem=1521972kb,idletime=3003,nusers=0,nsessions=0,
              uname=Linux lucid-tor2 2.6.32-33-server
              #70-Ubuntu SMP Thu Jul 7 22:28:30 UTC 2011 x86_64,opsys=linux
     mom_service_port = 15002
     mom_manager_port = 15003
     gpus = 0
```

Nota: Si el nodo aparece como `offline`, y únicamente `offline` (i.e. no como `down, offline`) se puede liberar con:

```
_: pbsnodes -c lucid-tor2
```

Para ver el estado de las colas podemos ejecutar `qstat -q`:

```
_: qstat -q

server: lucid-tor1

Queue            Memory CPU Time Walltime Node  Run Que Lm  State
---------------- ------ -------- -------- ----  --- --- --  -----
batch              --      --       --      --    0   0 --   E R
                                                ----- -----
                                                    0     0
```

E.2.2.3 Envío de un trabajo

Para comprobar que TORQUE es capaz de ejecutar trabajos podemos enviarle un trabajo sencillo. Para ello ejecutaremos el comando `echo "sleep 30"| qsub` desde el usuario creado al final de la sección E.2.2.1. Si el trabajo fue correctamente recibido deberá aparecer al usar el comando `qstat`:

```
root@lucid-tor1:~# su - david
david@lucid-tor1:~$ echo "sleep 30" | qsub
0.lucid-tor1
david@lucid-tor1:~$ qstat
Job id                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
0.lucid-tor1              STDIN            david                  0 Q batch
```

E.2.3 Versiones instaladas

Software	Versión
Ubuntu	10.04
TORQUE	4.0.3

Tabla E.2: Versión instalada de TORQUE.

E.3 Arquitectura Web

E.3.1 Balanceador de carga

Usaremos nginx como balanceador de carga, y no como servidor web, que es la manera más habitual de verlo en funcionamiento.

E.3.1.1 Instalación

Para instalar nginx, haz lo siguiente:

```
_: apt-get update
_: apt-get upgrade
_: apt-get install nginx
```

E.3.1.2 Configuración

Para configurar nginx debemos añadir la parte de balanceo de carga al fichero de configuración `/etc/nginx/nginx.conf`. Como este fichero no es excesivamente grande, se muestra en su totalidad con la parte modificada resaltada:

```
user www-data;
worker_processes  1;

error_log  /var/log/nginx/error.log;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
    # multi_accept on;
}

http {
    include       /etc/nginx/mime.types;

    access_log        /var/log/nginx/access.log;

    sendfile        on;
    #tcp_nopush     on;

    #keepalive_timeout  0;
    keepalive_timeout  65;
    tcp_nodelay        on;

    gzip  on;
    gzip_disable "MSIE [1-6]\.(?!.*SV1)";

    include /etc/nginx/conf.d/*.conf;
    include /etc/nginx/sites-enabled/*;


    ### Modified
    upstream web_servers {
      server 155.210.155.73:4567;
      server 155.210.155.178:4567;
    }

    server {
      listen 155.210.155.175:80;
      location / {
        proxy_pass http://web_servers;
      }
    }
    ### End of modification


}
```

E.3.1.3 Ejecución

Para iniciar nginx haremos uso del *script* localizado en `/etc/init.d`. Dicho *script* puede ser usado tanto para iniciarlo:

```
_: /etc/init.d/nginx start
```

como para pararlo:

```
_: /etc/init.d/nginx stop
```

E.3.1.4 Versiones instaladas

Software	Versión
nginx	0.7.65

Tabla E.3: Versión instalada de nginx.

E.3.2 Servidor web

Como servidor web usaremos WEBrick, que es el servidor web que viene por defecto con la instalación de Ruby. Instalando Ruby instalaremos a la vez el servidor web.

E.3.2.1 Instalación

Lo primero que hay que hacer es instalar Ruby y RubyGems. Para ello, consulta el anexo E.4 de instalación de Ruby.

Una vez instalado Ruby y RubyGems, instalaremos el paquete `ruby-dev`:

```
_: apt-get install ruby1.8-dev
```

Y ahora instalaremos el soporte necesario para interactuar con la base de datos:

```
_: apt-get install libmysqlclient-dev
_: gem install mysql
```

La aplicación web será desarrollada usando Sinatra. Antes de comenzar con la instalación, comprueba que tu versión de RubyGems es igual o superior a la 1.3.6. Esto puede hacerse fácilmente de la siguiente manera:

```
_: gem --version
1.3.6
```

Para instalar Sinatra, haremos lo siguiente:

```
_: gem install sinatra
Successfully installed rack-1.4.1
Successfully installed rack-protection-1.2.0
Successfully installed tilt-1.3.3
Successfully installed sinatra-1.3.2
4 gems installed
Installing ri documentation for rack-1.4.1...
...
Installing ri documentation for sinatra-1.3.2...
Installing RDoc documentation for rack-1.4.1...
...
Installing RDoc documentation for sinatra-1.3.2...
```

Nota: Puede llevar un tiempo hasta que el proceso de instalación muestre cosas por pantalla.

E.3.2.2 Ejecución

Una vez que la instalación ha finalizado, vamos a crear la primera aplicación web. Guarda el siguiente fichero como `web.rb`:

```
1 require 'rubygems'
2 require 'sinatra'
3
4 get '/' do
5   'Hello world!'
6 end
```

y lanza el servidor web:

```
_: ruby web.rb
```

Nota: Para salir Ctrl + C.

En tu navegador web preferido ve a la dirección `http://localhost:4567` y encontrarás la aplicación web que acabamos de crear.

E.3.2.3 Añadiendo soporte para la base de datos

Para interactuar con la base de datos usaremos ActiveRecord. Este componente es parte de Ruby On Rails, pero también existe como una *gem* independiente. Vamos a instalarla:

```
_: gem install activerecord
```

Ahora vamos a crear una segunda aplicación web. Guarda el siguiente fichero como `web2.rb`:

```
1 require 'rubygems'
2 require 'sinatra'
3 require 'active_record'
4
```

```
5  class Article < ActiveRecord::Base
6  end
7
8  get '/' do
9     'Hello there!'
10 end
```

y lanza el servidor web como antes:

```
_: ruby web2.rb
```

En tu navegador web ve a la dirección `http://localhost:4567` y encontrarás la aplicación web. Muestra lo mismo que la primera aplicación web, pero hemos incluido (aunque no usado) el soporte para la base de datos.

E.3.2.4 Versiones instaladas

Software	Versión
Ruby	1.8.7
RubyGems	1.8.21
Sinatra	1.3.2
ActiveRecord	3.2.3

Tabla E.4: Versiones instaladas de Ruby, RubyGems, Sinatra y ActiveRecord.

E.3.3 Base de datos

Como servidor de base de datos usaremos MySQL.

E.3.3.1 Instalación

Para instalar MySQL, haz lo siguiente:

```
_: apt-get install mysql-server
```

E.3.3.2 Configuración

Para configurar los ajustes básicos hay que editar el fichero `/etc/mysql/my.cnf`. Por ejemplo, si vamos a aceptar conexiones de otra máquina, hay que modificar el parámetro `bind_address`. En nuestro caso, lo modificaremos para aceptar conexiones del servidor web:

```
bind_address = 155.210.155.73
```

Nota: Si estás usando Ubuntu 10.04 debido a un *bug* es mejor que comentes toda la línea. Además nosotros usaremos dos servidores web, así que mejor la comentamos:

```
#bind_address = 155.210.155.73
```

Vamos a reiniciar el servidor para que los cambios surtan efecto:

```
_: /usr/bin/service mysql restart
```

Para comprobar que ha sido instalado correctamente, podemos hacer lo siguiente:

```
_: mysql -u root -p     # Introduce MySQL's root password
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 5.1.61-0ubuntu0.10.04.1 (Ubuntu)
...
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE mydb;
```

E.3.3.3 Ejecución

Para ejecutar el servidor de bases de datos usaremos el programa `service` localizado en `/usr/bin/service`. Sirve tanto para iniciarlo:

```
_: /usr/bin/service mysql start
```

como para pararlo:

```
_: /usr/bin/service mysql stop
```

E.3.3.4 Versiones instaladas

Software	Versión
mysql	5.1.61

Tabla E.5: Versión instalada de MySQL.

E.4 Ruby

Antes de poder programar en el lenguaje de programación Ruby deberemos instalar una serie de paquetes. Además de la instalación de los elementos necesarios para programar en Ruby también se instalarán una serie de herramientas de ayuda.

E.4.1 Instalación

Empezaremos instalando Ruby, IRB (Interactive Ruby Shell) y RDoc, la herramienta de generación de documentación de Ruby:

```
_: apt-get install ruby irb rdoc
```

A continuación instalaremos RubyGems. Para ello, descarga el paquete más actual de rubyforge. Durante el resto de la instalación usaremos como ejemplo el paquete 1.8.10, pero los pasos son análogos para cualquier otra versión.

Descomprimimos el paquete:

```
_: tar xvf rubygems-1.8.10.tgz
```

E instalamos RubyGems:

```
_: cd rubygems-1.8.10
_: ruby setup.rb
```

Una vez instalado, comprobamos la versión:

```
_: gem --version
1.8.10
```

Y la actualizamos a la última versión disponible. Es posible que antes de hacer este paso haya que actualizar el sistema operativo.

```
_: gem update --system
_: gem --version
1.8.24
```

En caso de que sea necesario actualizar el sistema operativo, se hace de esta manera:

```
_: apt-get upgrade
```

E.4.2 Problemas

Puede que durante la ejecución de programas Ruby, o la instalación de *gems* te encuentres con el siguiente error: `'require': no such file to load - mkmf (LoadError)`. La manera de solucionarlo es instalando el paquete `ruby1.8-dev`:

```
_: apt-get install ruby1.8-dev
```

También puede aparecer el error `no such file to load - net/https (LoadError)`. Para solucionarlo, instala `libopenssl-ruby`:

```
_: apt-get install libopenssl-ruby
```

E.4.3 Versiones instaladas

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
RubyGems	1.8.10 (o superior)

Tabla E.6: Versiones instaladas de Ruby y RubyGems.

E.5 Libvirt

E.5.1 Instalación

Instalaremos `libvirt-ruby` mediante apt-get:

```
_: apt-get install libvirt-ruby
```

E.5.2 Versiones instaladas

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
libvirt-ruby	0.0.7-1

Tabla E.7: Versión instalada de libvirt.

E.6 MCollective

MCollective es un sistema de despliegue de servidores y de sistemas de ejecución de trabajos en paralelo. Su función principal dentro del ámbito de la administración de sistemas es la ejecución de una acción sobre un conjunto de servidores. Para lograr esto, MCollective se apoya en un *middleware* que proporciona un servicio publicador-suscriptor. En este tipo de programas, las publicaciones y suscripciones suelen implementarse mediante el paso de mensajes asíncrono. Una de las posibilidades que se pueden usar con MCollective es RabbitMQ (Anexo E.7), que implementa el estándar de paso de mensajes AMQP (Advanced Message Queuing Protocol).

Dentro de la arquitectura MCollective, los componentes más importantes son:

- Cliente: Indica las órdenes a ejecutar.
- *Middleware*: Lleva las órdenes del cliente a los servidores.
- Servidor: Cumple las órdenes que le son enviadas.

El funcionamiento habitual de MCollective consiste en el lanzamiento de una orden desde el cliente que es transmitida a todos los servidores (*broadcast*) y éstos la llevan a cabo. En la figura E.1 se puede apreciar como la instrucción emitida por el cliente llega al corredor de mensajes (*broker*) y éste lo hace llegar a los servidores para que la cumplan.

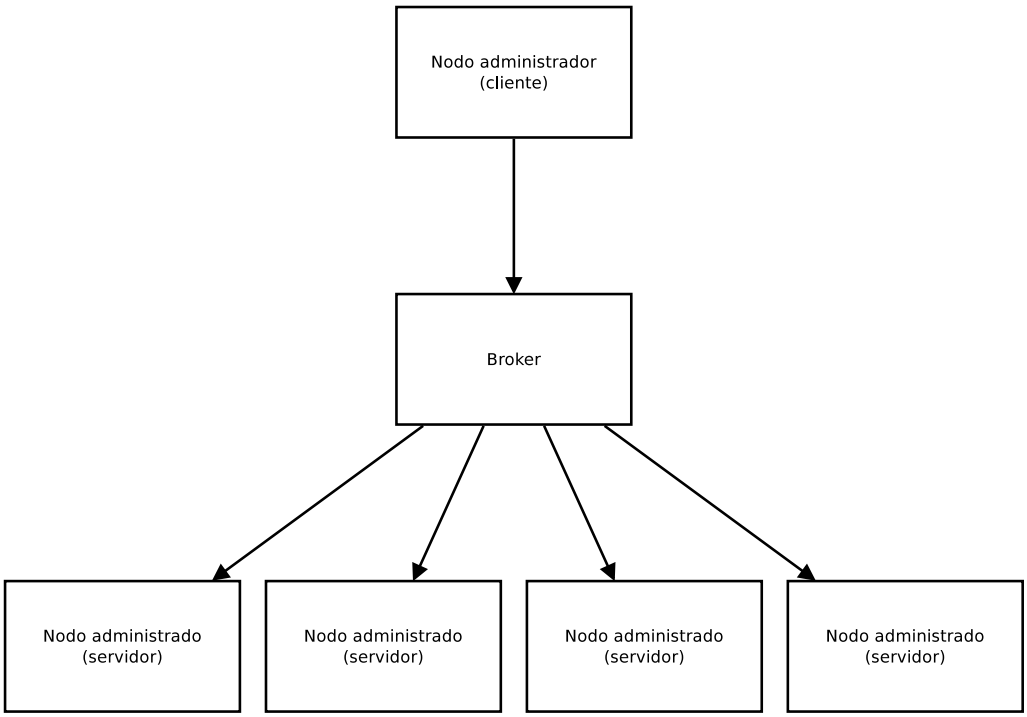

Figura E.1: Arquitectura de MCollective.

E.6.1 Instalación

El primer paso consiste en la instalación de RubyGems. Para ello, consulta el anexo E.4 de instalación de Ruby y RubyGems.

Descargamos los paquetes `mcollective`, `mcollective-client` y `mcollective-common` de `http://downloads.puppetlabs.com/mcollective/`:

```
_: wget http://downloads.puppetlabs.com/mcollective/\
mcollective_1.2.1-1_all.deb
_: wget http://downloads.puppetlabs.com/mcollective/\
mcollective-client_1.2.1-1_all.deb
_: wget http://downloads.puppetlabs.com/mcollective/\
mcollective-common_1.2.1-1_all.deb
```

Dependiendo de si el nodo va a ser administrador o administrado tendremos que instalar unos u otros paquetes:

Paquete	Nodo administrador	Nodo administrado
mcollective-common	X	X
mcollective-client	X	
mcollective		X

Podemos instalar los paquetes de la siguiente manera:

```
_: dpkg -i mcollective*.deb
```

Una vez instalado MCollective, instalamos la librería `libstomp-ruby`:

```
_: apt-get install libstomp-ruby
```

E.6.2 Configuración

A continuación hay que editar el fichero de configuración de MCollective. En los nodos administradores (clientes) se encuentra en `/etc/mcollective/client.cfg` mientras que en los nodos administrados (servidores) se encuentra en `/etc/mcollective/server.cfg`. Nos aseguramos de que contengan los siguientes valores:

```
plugin.stomp.host = 155.210.155.ABC # Direccion IP del MQ broker
plugin.stomp.port = 61613
plugin.stomp.user = mcollective
plugin.stomp.password = mcollective
```

E.6.3 Comprobación de la instalación

En el servidor stomp lanza el siguiente comando:

```
_: /usr/sbin/service rabbitmq-server start
```

En los nodos administrados (servidores) lanza éste:

```
_: /usr/sbin/service mcollective start
```

Y en uno de los nodos administradores (clientes) lanzaremos el comando `mc-ping`. La salida obtenida al ejecutar el comando debería ser similar a la siguiente:

```
_: mc-ping
155.210.155.177                          time=46.06 ms
---- ping statistics ----
1 replies max: 46.06 min: 46.06 avg: 46.06
```

E.6.4 Versiones instaladas

Software	Versión
Ubuntu	10.04
MCollective	1.2.1-1
libstomp-ruby	1.8 (1.0.4-1)

Tabla E.8: Versiones instaladas de MCollective y libstomp.

E.7 RabbitMQ

RabbitMQ es un *middleware* de mensajería que implementa el estándar AMQP (Advanced Message Queuing Protocol).

E.7.1 Instalación de RabbitMQ

Empezaremos añadiendo la línea

```
deb http://www.rabbitmq.com/debian/ testing main
```

a nuestro fichero `/etc/apt/sources.list` de la siguiente manera:

```
_: echo "deb http://www.rabbitmq.com/debian/ testing main" >> \
/etc/apt/sources.list
```

Para evitar avisos acerca de paquetes que no han sido firmados, podemos añadir la clave pública de RabbitMQ a nuestra lista de claves:

```
_: wget http://www.rabbitmq.com/rabbitmq-signing-key-public.asc
_: apt-key add rabbitmq-signing-key-public.asc
```

Actualizamos el sistema con los nuevos cambios:

```
_: apt-get update
```

E instalamos el paquete de la manera habitual:

```
_: apt-get install rabbitmq-server
```

La instalación incluirá automáticamente los paquetes de Erlang necesarios para ejecutar RabbitMQ.

E.7.2 Instalación de los *plugins* para el soporte de AMQP y STOMP

RabbitMQ no basta para proporcionar el servicio de paso de mensajes que MCollective requiere. Para que sea capaz de proporcionarlo necesitamos instalar unos *plugins*. Para ello, vamos al fichero `/usr/lib/rabbitmq/lib/rabbitmq_server-2.7.1/plugins` y comprobamos si existen `amqp_client-2.7.1.ez` y `rabbitmq_stomp-2.7.1.ez`. Si no existen, los instalamos:

```
_: wget -q http://www.rabbitmq.com/releases/plugins/v2.7.1/\
amqp_client-2.7.1.ez
_: wget -q http://www.rabbitmq.com/releases/plugins/v2.7.1/\
rabbitmq_stomp-2.7.1.ez
```

Nota: Los nombres de los *plugins* pueden variar dependiendo de la versión instalada.

Activamos los *plugins*:

```
_: rabbitmq-plugins list
_: rabbitmq-plugins enable amqp_client          # El nombre puede cambiar
_: rabbitmq-plugins enable rabbitmq_stomp       # El nombre puede cambiar
```

y reiniciamos el servidor:

```
_: /usr/sbin/service rabbitmq-server restart
```

E.7.3 Configuración de la cuenta de MCollective

Añadimos el usuario `mcollective` y la contraseña `mcollective`:

```
_: rabbitmqctl add_user mcollective mcollective
```

Establecemos los permisos necesarios:

```
_: rabbitmqctl set_permissions -p / mcollective "^amq.gen-.*" ".*" ".*"
```

Y por seguridad borramos la cuenta de invitado:

```
_: rabbitmqctl delete_user guest
```

E.7.4 Comprobación de la instalación

Para comprobar que hemos instalado correctamente RabbitMQ podemos hacer lo siguiente:

```
_: invoke-rc.d rabbitmq-server start
Starting rabbitmq-server: SUCCESS
rabbitmq-server.
```

E.7.5 Versiones instaladas

Software	Versión
Ubuntu	10.04
RabbitMQ	2.7.1
Erlang	R13B03 (erts-5.7.4)

Tabla E.9: Versiones instaladas de RabbitMQ y Erlang.

E.8 Puppet

E.8.1 Instalación de libopenssl-ruby

Si nuestro sistema no tiene instalado `libopenssl-ruby` tenemos que instalarlo:

```
_: apt-get install libopenssl-ruby
```

E.8.2 Creación del entorno de instalación de Puppet

Puppet necesita para su correcto funcionamiento una herramienta llamada Facter. Esta herramienta permite obtener datos como el sistema operativo, la distribución de Linux o la dirección MAC de un ordenador. Antes de instalar Facter y Puppet, hay que dar valores a ciertas variables:

```
_: FACTER_DIR="/root/facter-1.6.4"
_: PUPPET_DIR="/root/puppet-2.7.9"
_: PATH=$PATH:$FACTER_DIR/bin:$PUPPET_DIR/puppet/bin
_: RUBYLIB=$FACTER_DIR/lib:$PUPPET_DIR/lib
_: export PATH RUBYLIB
```

E.8.3 Instalación de Facter

Instalaremos Facter mediante una instalación por fuentes. Para ello primero tenemos que obtener las fuentes:

```
_: wget http://puppetlabs.com/downloads/facter/facter-1.6.4.tgz
```

Una vez conseguidas las fuentes lo descomprimimos:

```
_: gzip -d -c facter-1.6.4.tgz | tar xf -
```

Y posteriormente lo instalamos:

```
_: cd facter-1.6.4
_: ruby install.rb
```

E.8.4 Instalación de Puppet

Instalaremos Puppet también mediante fuentes. Obtenemos las fuentes:

```
_: wget http://puppetlabs.com/downloads/puppet/puppet-2.7.9.tgz
```

Lo descomprimimos:

```
_: gzip -d -c puppet-2.7.9.tgz | tar xf -
```

Y lo instalamos:

```
_: cd puppet-2.7.9
_: ruby install.rb
```

E.8.5 Configuración de Puppet

Nos aseguramos de que el fichero `/etc/puppet/puppet.conf` existe en nuestro sistema. Si no es así, podemos crearlo de la siguiente manera:

```
_: puppetmasterd --genconfig > /etc/puppet/puppet.conf
```

Tenemos que asegurarnos de que existen tanto un usuario como un grupo `puppet`. Si no existen, los creamos:

```
_: groupadd puppet
_: useradd -g puppet puppet
```

Por último tenemos que asegurarnos de que el directorio `/var/lib/puppet/rrd` pertenece al usuario y grupo `puppet`. Si no pertenece, cambiamos los permisos:

```
_: chown puppet /var/lib/puppet/rrd
_: chgrp puppet /var/lib/puppet/rrd
```

Nota: Si este directorio todavía no existe seguimos adelante con la instalación y ya revisitaremos este punto.

E.8.6 Comprobación de la instalación

Vamos a comprobar de manera rápida que la instalación de Facter y Puppet fue exitosa:

```
_: facter
[Muestra datos del sistema]
_: puppet describe -s user
[Muestra una descripcion del tipo usuario]
```

Ahora creamos un manifiesto simple en el fichero `manifest.pp`:

```
file {'testfile':
  path    => '/tmp/testfile',
  ensure  => present,
  mode    => 0640,
  content => "I'm a test file.",
}
```

Lo aplicamos:

```
_: puppet apply manifest.pp
```

Y comprobamos el contenido del fichero `/tmp/testfile` que Puppet debería haber creado:

```
_: cat /tmp/testfile
I'm a test file.
```

Ahora es momento de volver al directorio `/var/lib/puppet/rrd` y comprobar que tanto el usuario como el grupo de ese directorio tienen como valor `puppet`.

E.8.6.1 Comprobación del maestro de Puppet

Para comprobar que el maestro de puppet puede ejecutarse como un demonio tenemos que hacer:

```
_: puppet master
Could not prepare for execution: Got 1 failure(s) while initializing: \
change from directory to file failed:
Could not set 'file on ensure: Is a directory - /var/lib/puppet/facts
```

Si aparece este fallo tenemos que ir al fichero de configuración `/etc/puppet/puppet.conf` y comentar la línea con la propiedad `factdest`. Esta es la solución propuesta para el bug.

Si lo ejecutamos de nuevo no debería dar más problemas:

```
_: puppet master
```

E.8.6.2 Comprobación del agente de Puppet

Nota: Supondremos que el maestro de Puppet se está ejecutando en una máquina que tiene por nombre `puppet.example.com`. La máquina desde la que se ejecuta el agente tiene por nombre `node1.example.com`

Desde la máquina que tiene el agente, lanzamos la siguiente orden:

```
_: puppet agent --server=puppet.example.com \
--no-daemonize --verbose   # Usa nombres de dominio completos
warning: peer certificate won't be verified in this SSL session
info: Caching certificate for ca
warning: peer certificate won't be verified in this SSL session
warning: peer certificate won't be verified in this SSL session
info: Creating a new SSL certificate request for appscale-image1.cloud.net
info: Certificate Request fingerprint (md5): ...
warning: peer certificate won't be verified in this SSL session
[Se queda colgado]
[Ctrl + C]
```

Vamos a la máquina en la que se está ejecutando el maestro y hacemos lo siguiente:

```
_: puppet cert --sign node1.example.com
notice: Signed certificate request for node1.example.com
notice: Removing file Puppet::SSL::CertificateRequest node1.example.com at \
'/etc/puppet/ssl/ca/requests/node1.example.com.pem'
```

Y de vuelta a la máquina del agente:

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
warning: peer certificate won't be verified in this SSL session
info: Caching certificate for node1.example.com
notice: Starting Puppet client version 2.7.9
info: Caching certificate_revocation_list for ca
info: Caching catalog for node1.example.com
info: Applying configuration version '1331813472'
info: Creating state file /var/lib/puppet/state/state.yaml
notice: Finished catalog run in 0.03 seconds
[Se queda colgado]
[Ctrl + C]
notice: Caught INT; calling stop
```

Si no tienes una configuración preparada para tu nodo te aparecerá un error, pero no te preocupes: Puppet está funcionando correctamente. El error será similar a éste:

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
notice: Starting Puppet client version 2.7.9
info: Caching certificate_revocation_list for ca
err: Could not retrieve catalog from remote server: Error 400 on SERVER: \
Could not find default node or by name with 'node1.example.com, node1' \
on node node1.example.com
notice: Using cached catalog
err: Could not retrieve catalog; skipping run
[Se queda colgado]
[Ctrl + C]
notice: Caught INT; calling stop
```

E.8.6.3 Comprobación del sistema Puppet

Creamos el fichero `/etc/puppet/manifests/site.pp` en la máquina del maestro con el siguiente contenido:

```
# Create "/tmp/testfile" if it doesn't exist.
class test_class {
    file { "/tmp/testfile":
       ensure => present,
       mode   => 644,
       owner  => root,
       group  => root
    }
}

# tell puppet on which client to run the class
node 'node1' {            # Notice it is node1 and not node1.example.com
    include test_class
}

node 'puppet' {           # Notice it is puppet and not puppet.example.com
}
```

Y desde la máquina del agente tecleamos esto:

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
```

Comprobamos que en la máquina del agente se ha creado el fichero `/tmp/testfile`:

```
_: cat /tmp/testfile
I'm a test file.
```

E.8.7 Problemas

Durante la ejecución de Puppet pueden ocurrir varios errores. Como la información de Puppet puede ser en ocasiones algo críptica se muestra a continuación una colección de los errores más comunes y su solución:

```
_: puppet master
err: /File[/var/lib/puppet/facts]/ensure: change from directory to file \
failed: Could not set 'file on ensure: Is a directory - /var/lib/puppet/facts
```

Solución: Comenta la propiedad factdest en el fichero de configuración de Puppet en la máquina del maestro.

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
err: Could not retrieve catalog from remote server: Error 400 on SERVER: \
Could not find default node or by name with ...
```

Solución: Necesitas incluir el nombre del agente en el fichero `site.pp` de la máquina del maestro. O puedes incluir un nodo por defecto:

```
node 'default' {
}
```

```
_: puppet master
err: Could not prepare for execution: Retrieved certificate does not match \
private key; please remove certificate from server and regenerate it with \
the current key
```

Solución: Cambia el nombre del directorio `/etc/puppet/ssl` a `/etc/puppet/ssl.old` y prueba de nuevo.

```
_: puppet agent --server=puppet.example.com --no-daemonize --verbose
err: Could not send report: SSL_connect returned=1 errno=0 state=SSLv3 read \
server certificate B: certificate verify failed. This is often because the \
time is out of sync on the server or client
```

Solución: Compleja, mejor mira aquí.

```
_: puppet master
err: Could not run: Could not create PID file: /var/lib/puppet/run/master.pid
```

Solución: El maestro de Puppet ya se está ejecutando.

```
_: puppet apply manifest.pp
err: Could not evaluate: Puppet::Util::Log requires a message
```

Solución: Puppet ha salido de manera abrupta. Si estás ejecutando un proveedor asegúrate de que sales de él con `return` y no con `exit`.

E.8.8 Versiones instaladas

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
Facter	1.6.4
Puppet	2.7.9
libopenssl-ruby	4.2

Tabla E.10: Versiones instaladas de Puppet y Facter.

E.9 Neptune

Neptune es un lenguaje específico de dominio que permite al usuario ejecutar código en la plataforma AppScale.

E.9.1 Instalación

El primer paso consiste en la instalación de RubyGems. Para ello, consulta el anexo E.4 de instalación de Ruby y RubyGems. Cuando ese paso ya esté completado usaremos las RubyGems para instalar Neptune:

```
_: gem install neptune
```

Nota: Puede que en vez de `gem` tengas que usar `gem1.8` dependiendo de cómo estén configuradas las RubyGems.

E.9.2 Comprobación de la instalación

Puedes verificar que Neptune se ha instalado satisfactoriamente mirando si existe el ejecutable en `/usr/bin/neptune` y `/usr/lib/ruby/gems/1.8/gems/neptune-0.1.4/bin/neptune`.

E.9.3 Versiones instaladas

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
RubyGems	1.8.10 (o superior)
neptune	0.1.4

Tabla E.11: Versión instalada de neptune.

E.10 God

En ocasiones, debido a fallos de Puppet o porque es más sencillo para la monitorización de procesos simples, se ha usado la herramienta de monitorización God.

E.10.1 Instalación

El primer paso consiste en la instalación de RubyGems. Para ello, consulta el anexo E.4 de instalación de Ruby y RubyGems. Cuando ese paso ya esté completado instalaremos los paquetes `ruby1.8-dev` y `libopenssl-ruby`:

```
_: apt-get install ruby1.8-dev
_: apt-get install libopenssl-ruby
```

Cuando estos paquetes ya se hayan instalado, procedemos a instalar God:

```
_: gem install god
```

E.10.2 Comprobación de la instalación

Puedes verificar que God se ha instalado satisfactoriamente comprobando la versión instalada:

```
_: god --version
Version 0.12.1
```

E.10.3 Versiones instaladas

Software	Versión
Ubuntu	10.04
Ruby	1.8.7
RubyGems	1.8.10 (o superior)
god	0.12.1

Tabla E.12: Versión instalada de god.

Anexo F
Código fuente

F.1 generic-module

F.1.1 mcollective_leader.rb

```
require 'mcollective'
include MCollective::RPC

# MCollective leader client used to help with leader election algorithm
class MCollectiveLeaderClient < MCollectiveClient


   # Creates a new MCollective leader client.
   def initialize(client)
      super(client)
   end


   # Asks all nodes for their ID.
   def ask_id

      ids = []
      regex = /^.*ID: (\d+).*$/

      puts "Retrieving nodes' ids via MCollective Leader client"
      output = Helpers.rpcresults @mc.ask_id()

      puts "Complete output:"
      puts "----------------------------"
      puts "#{output}"
      puts "----------------------------"

      output.each_line do |line|
         m = line.match(regex)
         if m
            ids << m[1].to_i()      # Send them as integers
         end
      end

      return ids
```

```
    end


    # Sends all nodes the leader's ID.
    def new_leader(id)

        puts "Sending new leader information via MCollective Leader client
            "
        output = @mc.new_leader(:leader_id => id)
        return output

    end


end
```

F.1.2 genericp_helper.rb

```
LAST_MAC_FILE = "/tmp/cloud-last-mac"


# Defines and starts a virtual machine.
def start_vm(vm, ip_roles, img_roles, pm_up)

    # This function is cloud-type independent: define a new virtual
        machine and
    # start it

    # Get virtual machine's MAC address
    puts "Getting VM's MAC address"
    mac_address = get_vm_mac()

    # Get virtual machine's image disk
    puts "Getting VM's image disk"
    disk = get_vm_disk(vm, ip_roles, img_roles)

    # Define a new virtual machine
    id = rand(10000)      # Choose a number for domain name randomly
    vm_name = "myvm-#{id}"
    vm_uuid = `uuidgen`
    vm_mac  = mac_address
    vm_disk = disk
    vm_mem  = resource[:vm_mem]
    vm_ncpu = resource[:vm_ncpu]
    myvm = VM.new(vm_name, vm_uuid, vm_disk, vm_mac, vm_mem, vm_ncpu)

    # Write virtual machine's domain file
    domain_file_name = "cloud-%s-%s.xml" % [resource[:name], vm_name]
    domain_file_path = File.dirname(resource[:vm_domain]) +
                       "/" + "#{domain_file_name}"
    template_path = resource[:vm_domain]
    CloudInfrastructure.write_domain(myvm, domain_file_path,
        template_path)
    puts "Domain file written"
```

```
    # Choose a physical machine to host the virtual machine
    pm = pm_up[rand(pm_up.count)] # Choose randomly

    # Copy ssh key to physical machine
    puts "Copying ssh key to physical machine"
    pm_user = resource[:pm_user]
    pm_password = resource[:pm_password]
    out, success = CloudSSH.copy_ssh_key(pm_user, pm, pm_password)

    # Copy the domain definition file to the physical machine
    puts "Copying the domain definition file to the physical machine..."
    domain_file_src = domain_file_path
    domain_file_dst = "/tmp/" + domain_file_name

    out, success = CloudSSH.copy_remote(domain_file_src, pm,
        domain_file_dst, pm_user)
    if success
        puts "domain definition file copied"

        # Delete the local copy
        File.delete(domain_file_src)
    else
        err "#{vm_name} impossible to copy domain definition file"
    end

    # Define the domain in the physical machine
    puts "Defining the domain in the physical machine..."
    unless CloudInfrastructure.define_domain(pm_user, pm, vm_name,
        domain_file_dst)
        err "Impossible to define #{vm_name} domain"
    end

    # Start the domain
    puts "Starting the domain..."
    unless CloudInfrastructure.start_domain(pm_user, pm, vm_name)
        err "#{vm_name} impossible to start"
    end

    # Save the domain's name
    puts "Saving the domain's name..."
    unless CloudInfrastructure.save_domain_name(pm_user, pm, vm_name)
        err "#{vm_name} impossible to save in domains file"
    end

    # Save the new virtual machine's MAC address
    file = File.open(LAST_MAC_FILE, 'w')
    file.puts(mac_address)
    file.close

    # Send the new virtual machine's MAC address to all nodes
    mcc = MCollectiveFilesClient.new("files")
    mcc.create_file(LAST_MAC_FILE, mac_address)
    #mcc.disconnect

end
```

```
################################################################################

# Auxiliar functions
################################################################################



# Gets the virtual machine's mac address.
def get_vm_mac()

   if File.exists?(LAST_MAC_FILE)
      file = File.open(LAST_MAC_FILE, 'r')
      mac = MAC_Address.new(file.read().chomp())
      mac_address = mac.next_mac()
      file.close
   else
      mac = MAC_Address.new(resource[:starting_mac_address])
      mac_address = mac.mac
   end

   return mac_address

end


# Gets the virtual machine's disk image.
def get_vm_disk(vm, ip_roles, img_roles)

   # TODO What if a machine has different roles?
   role = :undefined
   index = 0
   ip_roles.each do |r, ips|
      index_aux = 0      # Reset the index for each role
      ips.each do |ip|
         if vm == ip
            puts "vm: #{vm} == ip: #{ip}"
            role = r
            index = index_aux
            puts "role: #{role}"
         else
            index_aux += 1
         end
      end
   end

   puts "Finished iterating. role: #{role}, index: #{index}"
   disk = img_roles[role][index]

   return disk

end
```

```
#################################################################################



# Checks if a machine is alive
def alive?(ip)

   ping = "ping -q -c 1 -w 4"
   result = `#{ping} #{ip}`
   return $?.exitstatus == 0

end

# Gets all the roles a node has.
def get_vm_roles(roles, vm)

   # The roles array is a map of roles - IP addresses. The 'IP addresses
      ' value
   # can be either a single value or an array of values.

   vm_roles = []
   roles.each do |role, ips|
      if ips == vm
         vm_roles << role
      elsif ips.is_a?(Array) && ips.include?(vm)
         vm_roles << role
      end
   end
   return vm_roles

end
```

F.1.3 cloudmonitor.rb

```
# Generic monitor functions for a distributed infrastructure
module CloudMonitor

   PING = "ping -q -c 1 -w 4"

   # Checks if the <vm> machine is alive.
   def self.ping(vm)

      result = `#{PING} #{vm}`
      if $?.exitstatus == 0
         puts "[CloudMonitor]: #{vm} is up"
         return true
      else
         puts "[CloudMonitor]: #{vm} is down"
         return false
      end

   end
```

```
    # Checks if MCollective is installed in <vm>.
    def self.mcollective_installed(user, vm)

       installed = true

       # Client configuration file
       client_file = "/etc/mcollective/client.cfg"
       command = "cat #{client_file} > /dev/null 2> /dev/null"
       out, success = CloudSSH.execute_remote(command, user, vm)
       unless success
          puts "[CloudMonitor]: #{client_file} does not exist on #{user}@
              #{vm}"
          installed = false
       end

       # Server configuration file
       server_file = "/etc/mcollective/server.cfg"
       command = "cat #{server_file} > /dev/null 2> /dev/null"
       out, success = CloudSSH.execute_remote(command, user, vm)
       unless success
          puts "[CloudMonitor]: #{server_file} does not exist on #{user}@
              #{vm}"
          installed = false
       end

       return installed

    end


    # Checks if MCollective is running in <vm>.
    def self.mcollective_running(user, vm)

       command = "ps aux | grep -v grep | grep mcollective"
       out, success = CloudSSH.execute_remote(command, user, vm)
       unless success
          puts "MCollective is not running on #{vm}"
          command = "/usr/bin/service mcollective start"
          out, success = CloudSSH.execute_remote(command, user, vm)
          unless success
             puts "[CloudMonitor]: Impossible to start mcollective on #{
                 user}@#{vm}"
             return false
          else
             puts "[CloudMonitor]: MCollective is running now on #{vm}"
             return true
          end
       else
          puts "[CloudMonitor]: MCollective is running on #{vm}"
          return true
       end

    end

end
```

F.1.4 cloudinfrastructure.rb

```
module CloudInfrastructure

   # Constants

   VIRSH_CONNECT = "virsh -c qemu:///system"
   DOMAINS_FILE = "/tmp/defined-domains"


   ##########################################################################

   # Domain functions
   ##########################################################################



   # Writes the virtual machine's domain file.
   def self.write_domain(virtual_machine, domain_file_path,
      template_path)

      require 'erb'
      template = File.open(template_path, 'r').read()
      erb = ERB.new(template)
      domain_file = File.open(domain_file_path, 'w')
      domain_file.write(erb.result(virtual_machine.get_binding))
      domain_file.close

   end


   # Defines a domain for a virtual machine on a physical machine.
   def self.define_domain(pm_user, pm, vm_name, domain_file_name)

      command = "#{VIRSH_CONNECT} define #{domain_file_name}"
      out, success = CloudSSH.execute_remote(command, pm_user, pm)
      return success

   end


   # Starts a domain on a physical machine.
   def self.start_domain(pm_user, pm, vm_name)

      command = "#{VIRSH_CONNECT} start #{vm_name}"
      out, success = CloudSSH.execute_remote(command, pm_user, pm)
      return success

   end


   # Saves the virtual machine's domain name in a file.
   def self.save_domain_name(pm_user, pm, vm_name)

      command = "echo #{vm_name} >> #{DOMAINS_FILE}"
```

```
      out, success = CloudSSH.execute_remote(command, pm_user, pm)
      return success

   end


   # Shuts down a domain.
   def self.shutdown_domain(domain, pm_user, pm)

      command = "#{VIRSH_CONNECT} shutdown #{domain}"
      out, success = CloudSSH.execute_remote(command, pm_user, pm)
      return success

   end


   # Undefines a domain.
   def self.undefine_domains(domain, pm_user, pm)

      command = "#{VIRSH_CONNECT} undefine #{domain}"
      out, success = CloudSSH.execute_remote(command, pm_user, pm)
      return success

   end

end
```

F.1.5 genericp_main.rb

```
################################################################################

# Start cloud functions
################################################################################


# Test function
def mifunciondetest(resource, error_function)

   puts "El nombre del recurso es %s" % [resource[:name]]
   error_function.call "Error en GENERIC MAIN"

end

# Starting function for leader node.
def leader_start(cloud_type, vm_ips, vm_ip_roles, vm_img_roles, pm_up,
                 monitor_function)

   # We are the leader
   puts "#{MY_IP} is the leader"

   # Check wether virtual machines are alive or not
   alive = {}
   vm_ips.each do |vm|
      alive[vm] = false
```

```
    end

    puts "Checking whether virtual machines are alive..."
    vm_ips.each do |vm|
       if alive?(vm)
          debug "[DBG] #{vm} is up"
          alive[vm] = true
       else
          debug "[DBG] #{vm} is down"
          puts "#{vm} is down"
       end
    end

    # Monitor the alive machines. Start and configure the dead ones.
    deads = false
    vm_ips.each do |vm|
       if alive[vm]
          # If they are alive, monitor them
          puts "Monitoring #{vm}..."
          monitor_vm(vm, vm_ip_roles, monitor_function)
          puts "...Monitored"
       else
          # If they are not alive, start and configure them
          puts "Starting #{vm}..."
          start_vm(vm, vm_ip_roles, vm_img_roles, pm_up)
          puts "...Started"
          deads = true
       end
    end

    # Wait for all machines to be started
    unless deads

       # If not already started, start the cloud
       unless File.exists?("/tmp/cloud-#{resource[:name]}")

          # Copy important files to all machines
          puts "Copying important files to all virtual machines"
          copy_cloud_files(vm_ips, cloud_type)      # TODO Move it to
             monitor and call it each time for one vm?

          # Start the cloud
          if start_cloud(vm_ips, vm_ip_roles)

             # Make cloud nodes manage themselves
             #auto_manage(cloud_type)     # Only if cloud was started
                properly FIXME Uncomment after tests

             # Create file
             cloud_file = File.open("/tmp/cloud-#{resource[:name]}", 'w')
             cloud_file.puts(resource[:name])
             cloud_file.close

             puts "===================="
             puts "== Cloud started =="
             puts "===================="
```

```
         else
            puts "Impossible to start cloud"
         end
      end      # unless File

   end      # unless deads

end


# Starting function for common (non-leader) nodes.
def common_start()

   # We are not the leader or we have not received our ID yet
   puts "#{MY_IP} is not the leader"

   cloud_leader = CloudLeader.new()

   if cloud_leader.id == -1

      # If we have not received our ID, let's assume we will be the
         leader
      cloud_leader.set_id(0)
      cloud_leader.set_leader(0)

      puts "#{MY_IP} will be the leader"

      # Create your ssh key
      CloudSSH.generate_ssh_key()

   else

      # If we have received our ID, try to become leader
      puts "Trying to become leader..."

      # Get all machines' IDs
      mcc = MCollectiveLeaderClient.new("leader")
      ids = mcc.ask_id()

      # See if some other machine is leader
      exists_leader = false
      ids.each do |id|
         if id < cloud_leader.id
            exists_leader = true
            break
         end
      end

      # If there is no leader, we will be the new leader
      if !exists_leader
         mcc.new_leader(cloud_leader.id.to_s())
         puts "...#{MY_IP} will be leader"

         # Create your ssh key
         CloudSSH.generate_ssh_key()
      else
```

```
         puts "...Some other machine is/should be leader"
      end
      mcc.disconnect

      return
   end

end


# Starting function for nodes which do not belong to the cloud.
def not_cloud_start(cloud_type, vm_ips, vm_ip_roles, vm_img_roles, pm_up
   )

   # Try to find one virtual machine that is already running
   alive = false
   vm_leader = ""
   vm_ips.each do |vm|
      if alive?(vm)
         puts "#{vm} is up"
         alive = true
         vm_leader = vm
         break
      end
   end

   if !alive
      puts "All virtual machines are stopped"
      puts "Starting one of them..."

      # Start one of the virtual machines
      vm = vm_ips[rand(vm_ips.count)]     # Choose one randomly
      puts "Starting #{vm} ..."

      start_vm(vm, vm_ip_roles, vm_img_roles, pm_up)

      # That virtual machine will be the "leader" (actually the chosen
         one)
      vm_leader = vm

      # Copy important files to it
      #copy_cloud_files(vm_leader, cloud_type)

      puts "#{vm_leader} is being started"
      puts "Once started, do 'puppet apply manifest.pp' on #{vm_leader}"
   else
      puts "#{vm_leader} is already running"
      puts "Do 'puppet apply manifest.pp' on #{vm_leader}"
   end

end


# Monitoring function for leader node.
def leader_monitoring(monitor_function)
```

```
    puts "#{MY_IP} is the leader"

    # Do monitoring
    deads = []
    vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data()
    vm_ips.each do |vm|
       puts "Monitoring #{vm}..."
       unless monitor_vm(vm, vm_ip_roles, monitor_function)
          deads << vm
       end
       puts "...Monitored"
    end

    # Check pool of physical machines
    pm_up, pm_down = check_pool()

    if deads.count == 0
       puts "=========================="
       puts "== Cloud up and running =="
       puts "=========================="
    else
       # Raise again the dead machines
       deads.each do |vm|
          start_vm(vm, vm_ip_roles, vm_img_roles, pm_up)
       end
    end

end


################################################################################

# Stop cloud functions
################################################################################


# Shuts down virtual machines.
def shutdown_vms()

    # Get pool of physical machines
    pms = resource[:pool]

    # Shut down virtual machines
    pms.each do |pm|

       pm_user = resource[:pm_user]
       pm_password = resource[:pm_password]

       # Copy ssh key to physical machine
       puts "Copying ssh key to physical machine"
       out, success = CloudSSH.copy_ssh_key(pm_user, pm, pm_password)

       # Bring the defined domains file from the physical machine to this
           one
       out, success = CloudSSH.get_remote(CloudInfrastructure::
          DOMAINS_FILE,
```

```
                                        pm_user, pm,
                                        CloudInfrastructure::
                                            DOMAINS_FILE)
      if success

         puts "#{CloudInfrastructure::DOMAINS_FILE} exists in #{pm}"

         # Open file
         defined_domains = File.open(CloudInfrastructure::DOMAINS_FILE,
             'r')

         # Stop nodes
         puts "Shutting down domains"
         defined_domains.each_line do |domain|
            domain.chomp!
            unless CloudInfrastructure.shutdown_domain(domain)
               err "#{domain} impossible to shut down"
            end
         end

         # Undefine local domains
         puts "Undefining domains"
         defined_domains.rewind
         defined_domains.each_line do |domain|
            domain.chomp!
            unless CloudInfrastructure.undefine_domain(domain)
               err "#{domain} impossible to undefine"
            end
         end

         # Delete the defined domains file on the physical machine
         puts "Deleting defined domains file"
         command = "rm -rf #{CloudInfrastructure::DOMAINS_FILE}"
         out, success = CloudSSH.execute_remote(command, pm_user, pm)

         # Delete all the domain files on the physical machine. Check
             how the
         # name is defined on 'start_vm' function.
         puts "Deleting domain files"
         domain_files = "cloud-%s-*.xml" % [resource[:name]]
         command = "rm /tmp/#{domain_files}"
         out, success = CloudSSH.execute_remote(command, pm_user, pm)

      else
         # Some physical machines might not have any virtual machine
             defined.
         # For instance, no virtual machine will be defined if they were
              already
         # defined and running when we started the cloud.
         puts "No #{CloudInfrastructure::DOMAINS_FILE} file found in #{
             pm}"
      end

   end   # pms.each

end
```

```
# Stops cron jobs on all machines.
def stop_cron_jobs(cloud_type)

   mcc = MCollectiveCronClient.new("cronos")
   string = "init-#{cloud_type}"
   mcc.delete_line(CRON_FILE, string)

end


# Deletes cloud files on all machines.
def delete_files()

   puts "Deleting cloud files on all machines..."

   # Create an MCollective client so that we avoid errors that appear
   # when you create more than one client in a short time
   mcc = MCollectiveFilesClient.new("files")

   # Delete leader, id, last_id and last_mac files on all machines (
      leader included)
   mcc.delete_file(CloudLeader::LEADER_FILE)             # Leader ID
   mcc.delete_file(CloudLeader::ID_FILE)                 # ID
   mcc.delete_file(CloudLeader::LAST_ID_FILE)            # Last ID
   mcc.delete_file(LAST_MAC_FILE)                        # Last MAC
      address
   mcc.disconnect       # Now it can be disconnected

   # Delete rest of regular files on leader machine
   files = [CloudInfrastructure::DOMAINS_FILE,           # Domains file
            "/tmp/cloud-#{resource[:name]}"]             # Cloud file
   files.each do |file|
      if File.exists?(file)
         File.delete(file)
      else
         puts "File #{file} does not exist"
      end
   end

end


##############################################################################

# Auxiliar functions
##############################################################################


# Checks the pool of physical machines are OK.
def check_pool()

   machines_up = []
   machines_down = []
   machines = resource[:pool]
```

```
    machines.each do |machine|
       if alive?(machine)
          debug "[DBG] #{machine} (PM) is up"
          machines_up << machine
       else
          debug "[DBG] #{machine} (PM) is down"
          machines_down << machine
       end
    end
    return machines_up, machines_down

end


# Obtains the virtual machine's data
#def obtain_vm_data(ip_function, img_function)
#
#   vm_ips = []
#   vm_ip_roles = []
#   vm_img_roles = []
#   puts "Obtaining virtual machines' data"
#   vm_ips, vm_ip_roles = ip_function.call(resource[:ip_file])
#   vm_img_roles = img_function.call(resource[:img_file])
#   return vm_ips, vm_ip_roles, vm_img_roles
#
#end


# Checks if this node is the leader
def leader?()

    cloud_leader = CloudLeader.new()

    return cloud_leader.leader?

end


# Monitors a virtual machine.
# Returns false if the machine is not alive.
def monitor_vm(vm, ip_roles, monitor_function)

    # Check if it is alive
    alive = CloudMonitor.ping(vm)
    unless alive
       err "#{vm} is not alive. Impossible to monitor"

       # If it is not alive there is no point in continuing
       return false
    end

    # Get user and password
    user = resource[:vm_user]
    password = resource[:root_password]

    # Send it your ssh key
```

```
    # Your key was created when you turned into leader
    puts "Sending ssh key to #{vm}"
    out, success = CloudSSH.copy_ssh_key(user, vm, password)
    if success
       puts "ssh key sent"
    else
       err "ssh key impossible to send"
    end

    # Check if MCollective is installed and configured
    mcollective_installed = CloudMonitor.mcollective_installed(user, vm)
    unless mcollective_installed
       err "MCollective is not installed on #{vm}"
    end

    # Make sure MCollective is running.
    # We need this to ensure the leader election, so ensuring MCollective
    # is running can not be left to Puppet in their local manifest. It
       must be
    # done explicitly here and now.
    mcollective_running = CloudMonitor.mcollective_running(user, vm)
    unless mcollective_running
       err "MCollective is not running on #{vm}"
    end

    # A node may have different roles
    vm_roles = get_vm_roles(ip_roles, vm)


    # Check if they have their ID
    # If they are running, but they do not have their ID:
    #   - Set their ID before they can become another leader.
    #   - Set also the leader's ID.
    success = CloudLeader.vm_check_id(user, vm)
    unless success

       cloud_leader = CloudLeader.new()

       # Set their ID (based on the last ID we defined)
       id = cloud_leader.last_id
       id += 1
       CloudLeader.vm_set_id(user, vm, id)
       cloud_leader.set_last_id(id)

       # Set the leader's ID
       leader = cloud_leader.leader
       CloudLeader.vm_set_leader(user, vm, leader)

       # Send the last ID to all nodes
       mcc = MCollectiveFilesClient.new("files")
       mcc.create_file(CloudLeader::LAST_ID_FILE, id.to_s)
       #mcc.disconnect
    end

    # TODO Copy cloud files each time a machine is monitored?
    # TODO Copy files no matter what or check first if they have them?
```

```
    # Use copy_cloud_files if we copy no matter what. Modify it if we
       check
    # We should copy no matter what in case they have changed

    # Depending on the type of cloud we will have to monitor different
       components.
    # Also, take into account that one node can perform more than one
       role.
    print "#{vm} will perform the roles: "
    p vm_roles
    vm_roles.each do |role|
       monitor_function.call(vm, role)
    end

    return true

end


# Copies important files to all machines inside <ips>.
def copy_cloud_files(ips, cloud_type)
# Use MCollective?
#  - Without MCollective we are able to send it to both one machine or
    multiple
#    machines without changing anything, so not really.

    ips.each do |vm|

       if vm != MY_IP

          # Cloud manifest
          file = "init-#{cloud_type}.pp"
          path = "/etc/puppet/modules/#{cloud_type}/manifests/#{file}"
          out, success = CloudSSH.copy_remote(path, vm, path)
          unless success
             err "Impossible to copy #{file} to #{vm}"
          end

          #TODO Use a user-provided function to copy important files to
             all nodes?

       end
    end

end


# Makes the cloud auto-manageable through crontab files.
def auto_manage(cloud_type)

    cron_file = "crontab-#{cloud_type}"
    path = "/etc/puppet/modules/#{cloud_type}/files/cron/#{cron_file}"

    if File.exists?(path)
       file = File.open(path, 'r')
       line = file.read().chomp()
```

```
      file.close

      # Add the 'puppet apply manifest.pp' line to the crontab file
      mcc = MCollectiveCronClient.new("cronos")
      mcc.add_line(CRON_FILE, line)
      mcc.disconnect
   else
      err "Impossible to find cron file at #{path}"
   end

end
```

F.1.6 vm.rb

```
# Virtual machine class
class VM

   attr_accessor :vm


   # Creates a description of a virtual machine.
   def initialize(name, uuid, disk, mac, mem, ncpu)
      @vm = {
         :name => "#{name}",
         :uuid => "#{uuid}",
         :disk => "#{disk}",
         :mac  => "#{mac}",
         :mem  => "#{mem}",
         :ncpu => "#{ncpu}"}
   end


   # Provides binding for ERB templates.
   def get_binding
      binding()
   end

end
```

F.1.7 mcollective_client.rb

```
require 'mcollective'
include MCollective::RPC

# MCollective base client
class MCollectiveClient


   # Creates a new MCollective base client.
   def initialize(client)
      @client = client
```

```
      @mc = rpcclient(@client)
   end


   # Disconnects an MCollective client.
   def disconnect
      puts "Disconnecting MCollective client..."
      @mc.disconnect
   end


end
```

F.1.8 gedit.sh

```
gedit genericp_*.rb &
```

F.1.9 mcollective_files.rb

```
require 'mcollective'
include MCollective::RPC

# MCollective files client used to manage files
class MCollectiveFilesClient < MCollectiveClient


   # Creates a new MCollective files client.
   def initialize(client)
      super(client)
   end


   # Cretes a new file in <path> containing <content>.
   def create_file(path, content)

      puts "Sending path and content via MCollective Files client"
      @mc.create(:path => path, :content => content)
      printrpcstats

   end


   # Appends <content> to the file located at <path>.
   def append_content(path, content)

      puts "Sending path and content via MCollective Files client"
      @mc.append(:path => path, :content => content)
      printrpcstats

   end
```

```
   # Deletes a file located at <path>.
   def delete_file(path)

      puts "Sending path via MCollective Files client"
      @mc.delete(:path => path)
      printrpcstats

   end


end
```

F.1.10 mcollective_cron.rb

```
require 'mcollective'
include MCollective::RPC

# MCollective cron client used to manage crontab files
class MCollectiveCronClient < MCollectiveClient


   # Creates a new MCollective cron client.
   def initialize(client)
      super(client)
   end


   # Adds the line <line> to the crontab file located at <path>.
   def add_line(path, line)

      puts "Sending path and line via MCollective Cron client"
      @mc.add_line(:path => path, :line => line)
      printrpcstats

   end


   # Deletes lines that contain the <string> string in the crontab
      located at <path>.
   def delete_line(path, string)

      puts "Sending path and string via MCollective Cron client"
      @mc.delete_line(:path => path, :string => string)
      printrpcstats

   end


end
```

F.1.11 cloudconstants.rb

```
# Some constants

MY_IP = Facter.value(:ipaddress)
CRON_FILE = "/var/spool/cron/crontabs/root"
```

F.1.12 mac.rb

```
# MAC address
class MAC_Address

   attr_reader :mac

   # Creates a new MAC_Address object.
   def initialize(value=nil)
      @mac = value ? value: "52:54:00:00:00:00"
   end


   # Obtains the next MAC address.
   def next_mac

      mac_string = @mac.delete(":")
      mac_int = mac_string.to_i(16)
      mac_int += 1
      mac_hex = mac_int.to_s(16)
      result = mac_hex[0..1] + ":" + mac_hex[2..3] + ":" + mac_hex[4..5]
          + ":" +
               mac_hex[6..7] + ":" + mac_hex[8..9] + ":" + mac_hex
                  [10..11]
      return result

   end


   # Generates an array of <many> MAC addresses starting from this one.
   def generate_array(many)

      result = []
      result << @mac
      for i in 1..many
         mac = MAC_Address.new(result[i - 1])
         result << mac.next_mac
      end
      return result

   end


end
```

F.1.13 cloudleader.rb

```
# Generic leader election methods for a distributed infrastructure
class CloudLeader

   ID_FILE      = "/tmp/cloud-id"
   LEADER_FILE  = "/tmp/cloud-leader"
   LAST_ID_FILE = "/tmp/cloud-last-id"

   attr_reader :id, :leader, :last_id

   def initialize(id_file = ID_FILE, leader_file = LEADER_FILE,
                  last_id_file = LAST_ID_FILE)

      @id_file      = id_file
      @leader_file  = leader_file
      @last_id_file = last_id_file

      @id      = init_id()
      @leader  = init_leader()
      @last_id = init_last_id()

   end


   ################################################################################

   private

   # Gets the node's ID by reading the node's id_file.
   def init_id()

      if File.exists?(@id_file)
         id_file = File.open(@id_file, 'r')
         id = id_file.read().chomp().to_i()
         id_file.close
      else
         id = -1
      end
      return id

   end


   # Gets the leader's ID by reading the node's leader_file.
   def init_leader()

      if File.exists?(@leader_file)
         leader_file = File.open(@leader_file, 'r')
         leader = leader_file.read().chomp().to_i()
         leader_file.close
      else
         leader = -1
      end
      return leader
```

```
    end


    # Gets the last defined ID in the ID file.
    def init_last_id()

        if File.exists?(@last_id_file)
            file = File.open(@last_id_file, 'r')
            id = file.read().chomp().to_i
            file.close
        else
            id = @id
        end
        return id

    end


    ################################################################################

    public

    # Sets the node's ID.
    def set_id(id)

        file = File.open(@id_file, 'w')
        file.puts(id)
        file.close
        @id = id

    end


    # Sets the leader's ID in the node.
    def set_leader(leader)

        file = File.open(@leader_file, 'w')
        file.puts(leader)
        file.close
        @leader = leader

    end


    # Sets last defined ID in the ID file.
    def set_last_id(id)

        file = File.open(@last_id_file, 'w')
        file.puts(id)
        file.close
        @last_id = id

    end
```

```
    # Checks if this node is leader.
    def leader?

        return @id == @leader && @id != -1

    end


    ##############################################################################

    # Remote ID functions
    ##############################################################################


    # Checks if the remote node has their ID file.
    def self.vm_check_id(user, vm, id_file = ID_FILE)

        command = "cat #{id_file} > /dev/null 2> /dev/null"
        out, success = CloudSSH.execute_remote(command, user, vm)
        return success

    end


    # Sets the node's ID on a remote node.
    def self.vm_set_id(user, vm, id, id_file = ID_FILE)

        command = "echo #{id} > #{id_file}"
        out, success = CloudSSH.execute_remote(command, user, vm)
        return success

    end


    # Sets the leader's ID on a remote node.
    def self.vm_set_leader(user, vm, leader, leader_file = LEADER_FILE)

        command = "echo #{leader} > #{leader_file}"
        out, success = CloudSSH.execute_remote(command, user, vm)
        return success

    end

end
```

F.1.14 cloudssh.rb

```
# Generic ssh functions for a distributed infrastructure
module CloudSSH

    SSH_PATH = "/root/cloud/ssh"
    SSH_KEY = "id_rsa"

    # Generates a new ssh key to be used in all machines.
```

```
    def self.generate_ssh_key(path = SSH_PATH, file = SSH_KEY)

       puts "Creating #{path} directory..."
       result = `mkdir -p #{path}`
       unless $?.exitstatus == 0
          puts "Could not create #{path} directory"
       end

       puts "Deleting previous keys..."
       result = `rm -rf #{path}/*`
       unless $?.exitstatus == 0
          puts "Could not create #{path}/#{file} key"
       end

       puts "Generating key..."
       result = `ssh-keygen -t rsa -N '' -f #{path}/#{file}`
       unless $?.exitstatus == 0
          puts "Could not create #{path}/#{file} key"
       end

       puts "Evaluating agent and adding identity..."

       # Must be done in one command
       result = `eval \`ssh-agent\` ; ssh-add #{path}/id_rsa`
       unless $?.exitstatus == 0
          puts "Could not add #{path}/#{file} key"
       end

    end


    # Copies an ssh key to a machine.
    def self.copy_ssh_key(user, ip, password, path = SSH_PATH, file =
       SSH_KEY)

       command_path = "/etc/puppet/modules/generic-module/provider/"
       identity_file = "#{path}/#{file}.pub"     # Be careful, copy
          PUBLIC KEY
       if password != ""
          puts "password is not empty, using ssh_copy_id.sh shell script"
          result = `#{command_path}/ssh_copy_id.sh #{user}@#{ip} #{
             identity_file} #{password}`
          success = $?.exitstatus == 0
       else
          puts "password is empty, using ssh-copy-id command"
          result = `ssh-copy-id -i #{identity_file} #{user}@#{ip}`
          success = $?.exitstatus == 0
       end
       return result, success
    end


    # Executes a command on a remote machine.
    def self.execute_remote(command, user, ip, path = SSH_PATH, file =
       SSH_KEY)
```

```
      result = `ssh #{user}@#{ip} -i #{path}/#{file} '#{command}'`
      exit_code = $?.exitstatus
      success = (exit_code == 0)
      return result, success, exit_code
   end


   # Copies a file to a remote machine.
   def self.copy_remote(src_file, dst_ip, dst_file, dst_user = "root",
                        path = SSH_PATH, file = SSH_KEY)

      result = `scp -i #{path}/#{file} #{src_file} #{dst_user}@#{dst_ip
         }:#{dst_file}`
      success = $?.exitstatus == 0
      return result, success
   end


   # Gets a file from a remote machine.
   def self.get_remote(src_file, src_user, src_ip, dst_file,
                       path = SSH_PATH, file = SSH_KEY)

      result = `scp -i #{path}/#{file} #{src_user}@#{src_ip}:#{src_file}
          #{dst_file}`
      success = $?.exitstatus == 0
      return result, success

   end
end
```

F.1.15 ssh_copy_id.sh

```
#!/usr/bin/env expect

##############################
#
# Author : Kowshik Prakasam
#
# An expect script to automatically login to each host as a particular
   user and
# install the user's public key using ssh-copy-id
#
# If successful, exits with zero (0) status
# If unsuccessful, exits with non-zero status
#
# Usage : sshcopy.exp <user@host> <key_file> <password>
# Example : sshcopy.exp root@128.111.55.234 /home/appscale/.appscale/
   appscale joe
#
##############################

# Procedure to interact with ssh-copy-id command
# Parameter : password
proc sshcopyid { password } {
```

```
  expect {
    # Send password at 'Password' prompt and tell expect to continue(i.e
       . exp_continue)
    -re "\[P|p]assword:" { exp_send "$password\r"
                          exp_continue }

    #Returning 1 as ssh-copy-id has failed
    -re "^\[P|p]ermission denied*" { return 1}

    #Answering yes to ssh host
    -nocase  "are you sure you want to continue connecting (yes/no)?" {
       exp_send "yes\r"
                                                                    exp_co

                                                                       }



    # Tell expect stay in this 'expect' block and for each character
       that ssh-copy-id prints while doing the copy
    # Reset the timeout counter back to 0
    -re .                { exp_continue  }
    timeout              { return 1      }

    #Returning 0 as ssh-copy-id was successful
    eof                  { return 0      }
  }
}

#Parsing command-line arguments
set host [lrange $argv 0 0]
set key_file [lrange $argv 1 1]
set password [lrange $argv 2 2]

#Setting timeout to an arbitrary value of 3 that works well for ssh-copy
   -id
set timeout 3

# Execute ssh-copy-id command
eval spawn ssh-copy-id -i $key_file $host

#Get the result of ssh-copy-id
set sshcopyid_result [sshcopyid $password]

# If ssh-copy-id was successful
if { $sshcopyid_result == 0 } {
  #Exit with zero status
  exit 0
}

# Error attempting ssh-copy-id, so exit with non-zero status
exit 1
```

F.2 appscale

F.2.1 appscale.rb

```
Puppet::Type.newtype(:appscale) do
   @doc = "Manages AppScale clouds formed by KVM virtual machines."


   ensurable do

      desc "The cloud's ensure field can assume one of the following
         values:
   `running`: The cloud is running.
   `stopped`: The cloud is stopped.\n"

      newvalue(:stopped) do
         provider.stop
      end

      newvalue(:running) do
         provider.start
      end

   end


   # General parameters

   newparam(:name) do
      desc "The cloud name"
      isnamevar
   end

   newparam(:ip_file) do
      desc "The file with the cloud description in YAML format"
   end

   newparam(:img_file) do
      desc "The file containing the qemu image(s). You must either
         provide " +
           "one image from which all copies shall be made or provide " +
           "an image for every instance"
   end

   newparam(:vm_domain) do
      desc "The XML file with the virtual machine domain definition. " +
           "Libvirt XML format must be used"
   end

   newproperty(:pool, :array_matching => :all) do
      desc "The pool of physical machines"
   end
```

```
    # Virtual machine parameters
    newparam(:vm_mem) do
       desc "The virtual machine's maximum amopunt of memory. " +
            "In KiB: 2**10 (or blocks of 1024 bytes)."
       defaultto "1048576"
    end

    newparam(:vm_ncpu) do
       desc "The virtual machine's number of CPUs"
       defaultto "1"
    end


    # Infrastructure parameters

    newparam(:pm_user) do
       desc "The physical machines' user. It must have proper permissions
          "
       defaultto "dceresuela"
    end

    newparam(:pm_password) do
       desc "The physical machines' password"
       defaultto ""
    end

    newparam(:starting_mac_address) do
       desc "Starting MAC address for new virtual machines"
       defaultto "52:54:00:01:00:00"
    end

    newparam(:vm_user) do
       desc "Virtual machines' user"
       defaultto "root"
    end

    newparam(:root_password) do
       desc "Virtual machines' root password"
       defaultto "root"
    end


    # AppScale parameters

    newproperty(:controller, :array_matching => :all) do
       desc "The controller node"
    end

    newproperty(:servers, :array_matching => :all) do
       desc "The server nodes"
    end


    newproperty(:master, :array_matching => :all) do
       desc "The master node"
    end
```

```
    newproperty(:appengine, :array_matching => :all) do
       desc "The appengine nodes"
    end

    newproperty(:database, :array_matching => :all) do
       desc "The database nodes"
    end

    newproperty(:login, :array_matching => :all) do
       desc "The login node"
    end

    newproperty(:open, :array_matching => :all) do
       desc "The open nodes"
    end

    newproperty(:zookeeper, :array_matching => :all) do
       desc "The zookeeper nodes"
    end

    newproperty(:memcache, :array_matching => :all) do
       desc "The memcache nodes"
    end


    newparam(:app_email) do
       desc "AppScale administrator e-mail"
       defaultto "david@gmail.com"
    end

    newparam(:app_password) do
       desc "AppScale administrator password"
       defaultto "appscale"
    end

end
```

F.2.2 appscalep.rb

```
Puppet::Type.type(:appscale).provide(:appscalep) do
    desc "Manages AppScale clouds formed by KVM virtual machines"

    # Require appscale auxiliar files
    require File.dirname(__FILE__) + '/appscale/appscale_yaml.rb'
    require File.dirname(__FILE__) + '/appscale/appscale_functions.rb'

    # Require generic files
    require '/etc/puppet/modules/generic-module/provider/
       mcollective_client.rb'
    Dir["/etc/puppet/modules/generic-module/provider/*.rb"].each { |file|
        require file }

    # Commands needed to make the provider suitable
```

```
    commands :ping => "/bin/ping"
    commands :grep => "/bin/grep"
    commands :ps   => "/bin/ps"

    # Operating system restrictions
    confine :osfamily => "Debian"

    # Some constants
    #   They are in the generic cloud files

    # Makes sure the cloud is running.
    def start

       puts "Starting cloud %s" % [resource[:name]]

       # Zona de test
       mifunciondetest(resource, method(:err))

       # Check existence
       if !exists?
          # Cloud does not exist => Startup operations

          # Check pool of physical machines
          puts "Checking pool of physical machines..."
          pm_all_up, pm_up, pm_down = check_pool()
          if !pm_all_up
             puts "Some physical machines are down"
             pm_down.each do |pm|
                debug "[DBG] - #{pm}"
             end
          end

          # Obtain the virtual machines' IPs
          puts "Obtaining the virtual machines' IPs..."
          #vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data(method(:
             appscale_yaml_ips),
          #                                                  method(:
             appscale_yaml_images))
          vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data()

          # Check whether you are one of the virtual machines
          puts "Checking whether this machine is part of the cloud..."
          part_of_cloud = vm_ips.include?(MY_IP)
          if part_of_cloud
             puts "#{MY_IP} is part of the cloud"

             # Check if you are the leader
             if leader?()
                leader_start("appscale", vm_ips, vm_ip_roles,
                   vm_img_roles, pm_up,
                            method(:appscale_monitor))
             else
                common_start()
             end
          else
             puts "#{MY_IP} is not part of the cloud"
```

```
            not_cloud_start("appscale", vm_ips, vm_ip_roles,
               vm_img_roles, pm_up)
         end

      else

         # Cloud exists => Monitoring operations
         puts "Cloud already started"

         # Check if you are the leader
         if leader?()
            leader_monitoring(method(:appscale_monitor))
         else
            puts "#{MY_IP} is not the leader"      # Nothing to do
         end
      end

   end


   # Makes sure the cloud is not running.
   def stop

      puts "Stopping cloud %s" % [resource[:name]]

      if !exists?
         err "Cloud does not exist"
         return
      end
      if status != :running
         err "Cloud is not running"
         return
      end
      if exists? && status == :running

         puts "It is an appscale cloud"

         # Stop cloud infrastructure
         appscale_cloud_stop(MY_IP)    # TODO What if we run stop on a
            different machine than start?

         # Shutdown and undefine all virtual machines explicitly created
             for this cloud
         shutdown_vms()

         # Stop cron jobs on all machiness
         stop_cron_jobs("appscale")    # TODO Be careful of the order: 1
             stop cron and 2 stop appscale or the other way?

         # Delete files
         delete_files()

         # Note: As all the files deleted so far are located in the /tmp
             directory
         # only the machines that are still alive need to delete these
            files.
```

```
        # If the machine was shut down, these files will not be there
            the next
        # time it is started, so there is no need to delete them.

        puts "==================="
        puts "== Cloud stopped =="
        puts "==================="

    end

end


def status
    if File.exists?("/tmp/cloud-#{resource[:name]}")
        return :running
    else
        return :stopped
    end
end


# Ensure methods
def create
    return true
end


def destroy
    return true
end


def exists?
    return File.exists?("/tmp/cloud-#{resource[:name]}")
end


#############################################################################

# Properties need methods
#############################################################################

def pool
end

def pm_user
end

def starting_mac_address
end

def root_password
end

def app_email
```

```
    end

    def app_password
    end


end
```

F.2.3 appscalep_helper.rb

```
##############################################################################

# Auxiliar functions for appscale provider
##############################################################################


# The functions in this file are defined the same in all providers, but
    each
# one implements them in their own way. Thus, the headers cannot be
    modified.

# Starts an AppScale cloud formed by <vm_ips> performing <vm_ip_roles>
def start_cloud(vm_ips, vm_ip_roles)

    puts "Starting the cloud"
    if (resource[:app_email] == nil) || (resource[:app_password] == nil)
       err "Need an AppScale user and password"
       exit
    else
       puts "app_email = #{resource[:app_email]}"
       puts "app_password = #{resource[:app_password]}"
    end
    puts  "Starting an appscale cloud"

    # Start appscale cloud
    return appscale_cloud_start(vm_ips, vm_ip_roles,
                                resource[:app_email], resource[:
                                   app_password],
                                resource[:root_password])

end


# Obtains vm data from manifest parameters.
def obtain_vm_data()

#    if resource[:controller] != nil && resource[:servers] != nil
#
#       return obtain_appscale_data_default(resource[:controller],
#                                           resource[:servers])
#
#    elsif resource[:master] != nil && resource[:appengine] != nil &&
#          resource[:database] != nil && resource[:login] != nil &&
#          resource[:open] != nil
```

```
#
#      return obtain_appscale_data_custom(resource[:master],
#                                         resource[:appengine],
#                                         resource[:database],
#                                         resource[:login],
#                                         resource[:open],
#                                         resource[:zookeeper],
#                                         resource[:memcache])
#
#   end

   ips, ip_roles = appscale_yaml_ips(resource[:ip_file])
   img_roles     = appscale_yaml_ips(resource[:img_file])
   return ips, ip_roles, img_roles

end
```

F.2.4 appscale-run-instances.tcl

```
#!/usr/bin/env expect

# Description:
#   Interacts with the appscale-run-instances tool
#
# Synopsis:
#   appscale-run-instances.tcl <file> <e-mail> <password>
#
# Arguments:
#   - File: AppScale YAML configuration file.
#   - e-mail: AppScale administration e-mail.
#   - Password: AppScale administration password.
#
# Examples:
#   _$: appscale-run-instances.tcl ips.yaml user@mail.com appscale
#
#
# Author:
#   David Ceresuela


# Procedure to interact with appscale-run-instances command
# Parameter : user
# Parameter : password
proc runinstances { user password } {
  expect {
    # Send e-mail address
    -re "e-mail address:" { exp_send "$user\r"
                            exp_continue }

    # Send password
    -re "new password:" { exp_send "$password\r"
                          exp_continue }

    # Send password again to verify
```

```
    -re "again to verify:" { exp_send "$password\r"
                             exp_continue }


    # Tell expect stay in this 'expect' block and for each character
       that
    # appscale-run-instances prints while doing the copy
    # Reset the timeout counter back to 0
    -re .                  { exp_continue  }
    timeout                { return 1      }

    # Returning 0 as appscale-run-instances was successful
    eof                    { return 0      }
  }
}

#Parsing command-line arguments
set yaml     [lrange $argv 0 0]
set user     [lrange $argv 1 1]
set password [lrange $argv 2 2]

#Setting timeout to an arbitrary value of 120 that works well for
   appscale-run-instances
set timeout 120

# Execute appscale-run-instances command
eval spawn /usr/local/appscale-tools/bin/appscale-run-instances --ips
   $yaml

#Get the result of appscale-run-instances
set runinstances_result [runinstances $user $password]

# If appscale-run-instances was successful
if { $runinstances_result == 0 } {
  #Exit with zero status
  exit 0
}

# Error attempting appscale-run-instances, so exit with non-zero status
exit 1
```

F.2.5 appscale_yaml.rb

```
require 'yaml'

##
# Obtains the IP addresses from the ip_file file. It does NOT check
   whether
# the file has the proper format.
# Different roles obtained from AppScale wiki:
#    http://code.google.com/p/appscale/wiki/Placement_Support

def appscale_yaml_ips(path)
```

```
    ips = []
    ip_roles = {}

    file = File.open(path)
    tree = YAML::parse(file)

    if tree != nil

       tree = tree.transform

       # Default deployment (from appscale-tools/lib/node_layout.rb)
       controller = tree[:controller]
       servers =    tree[:servers]

       ip_roles[:controller] = get_elements(controller)
       ip_roles[:servers]    = get_elements(servers)

       ips = ips + ip_roles[:controller]
       ips = ips + ip_roles[:servers]

       # Custom deployment (from appscale-tools/lib/node_layout.rb)
       master =    tree[:master]
       appengine = tree[:appengine]
       database =  tree[:database]
       login =     tree[:login]
       open =      tree[:open]
       zookeeper = tree[:zookeeper]
       memcache =  tree[:memcache]

       ip_roles[:master]    = get_elements(master)
       ip_roles[:appengine] = get_elements(appengine)
       ip_roles[:database]  = get_elements(database)
       ip_roles[:login]     = get_elements(login)
       ip_roles[:open]      = get_elements(open)
       ip_roles[:zookeeper] = get_elements(zookeeper)
       ip_roles[:memcache]  = get_elements(memcache)

       ips = ips + ip_roles[:master]
       ips = ips + ip_roles[:appengine]
       ips = ips + ip_roles[:database]
       ips = ips + ip_roles[:login]
       ips = ips + ip_roles[:open]
       ips = ips + ip_roles[:zookeeper]
       ips = ips + ip_roles[:memcache]

       ips = ips.uniq

       # Delete all the roles that have no IP address associated
       ip_roles.delete_if{ |role, ips| ips == [] }

       file.close

       return ips, ip_roles
    end

end
```

```
# Obtains the disk images from the img_file file.
def appscale_yaml_images(path)

   img_roles = {}

   file = File.open(path)
   tree = YAML::parse(file)

   if tree != nil

      tree = tree.transform

      # Default deployment (from appscale-tools/lib/node_layout.rb)
      controller = tree[:controller]
      servers    = tree[:servers]

      # Custom deployment (from appscale-tools/lib/node_layout.rb)
      master    = tree[:master]
      appengine = tree[:appengine]
      database  = tree[:database]
      login     = tree[:login]
      open      = tree[:open]
      zookeeper = tree[:zookeeper]
      memcache  = tree[:memcache]

      # Maybe we have been given only an image for all virtual machines
      all = tree[:all]

      if all == nil

         # Default deployment
         img_roles[:controller] = get_elements(controller)
         img_roles[:servers]    = get_elements(servers)

         # Custom deployment
         img_roles[:master]    = get_elements(master)
         img_roles[:appengine] = get_elements(appengine)
         img_roles[:database]  = get_elements(database)
         img_roles[:login]     = get_elements(login)
         img_roles[:open]      = get_elements(open)
         img_roles[:zookeeper] = get_elements(zookeeper)
         img_roles[:memcache]  = get_elements(memcache)

         # Delete all the roles that have no disk image associated
         img_roles.delete_if{ |role, img| img == [] }

      else
         img_roles[:all] = get_elements(all)
      end

      file.close

      return img_roles
   end
```

```
end


# Writes a hash in a file using the YAML format.
# The hash should resemble something like
# {:controller => ["192.168.1.1"], :servers => ["192.168.1.2",
   "192.168.1.3"]}
# if you are writing the default deployment IP addresses.
# The custom deployment is done in a similar fashion.
# Either case, the key-value pairs must follow the {symbol => array}
   pattern.
def appscale_write_yaml_file(hash, path)

   # Get and clean the hash
   hash_yaml = hash.to_yaml(:Indent => 0).to_s
   hash_yaml = hash_yaml.gsub("!ruby/symbol ", ":")
   hash_yaml = hash_yaml.gsub("!ruby/sym ", ":")

   # Write to file
   file = File.open(path, 'w')
   file.write(hash_yaml)
   file.close

end


################################################################################

# Auxiliar functions
################################################################################


# Transforms the given elements to an array.
def get_elements(array)

   elements = []
   if array != nil
      elements = array.to_a
   end
   return elements

end
```

F.2.6 appscale_functions.rb

```
# Starts an AppScale cloud.
def appscale_cloud_start(app_ips, app_roles,
                         app_email=nil, app_password=nil, root_password=
                            nil)

   require 'expect'

   # Check arguments
```

```
    puts "appscale_cloud_start called with:"
    puts "  - app_email == #{app_email}"
    puts "  - app_password == #{app_password}"
    puts "  - root_password == #{root_password}"

    script_keys = "appscale-add-keypair.tcl"
    script_run  = "appscale-run-instances.tcl"
    #ips_yaml = resource[:ip_file]
    script_path = "/etc/puppet/modules/appscale/lib/puppet/provider/
       appscale/appscale"

    # Write ips.yaml file
    puts "Writing AppScale ips_yaml file"
    puts "Hash received: "
    p app_roles
    ips_yaml = "/etc/puppet/modules/appscale/files/auto-ips.yaml"
    appscale_write_yaml_file(app_roles, ips_yaml)
    puts "AppScale ips_yaml file written"

    # Add key pairs
    puts "About to add key pairs"
    debug "[DBG] ips.yaml file: #{ips_yaml}"
    result = `#{script_path}/#{script_keys} #{ips_yaml} #{root_password}`
    if $?.exitstatus == 0
       puts "Key pairs added"
       puts "result = |||#{result}|||"
    else
       err "Impossible to add key pairs"
       return false
    end

    # Run instances
    puts "About to run instances"
    puts "This may take a while (~ 5 min), so please be patient"
    result = `#{script_path}/#{script_run} #{ips_yaml} #{app_email} #{
       app_password}`
    if $?.exitstatus == 0
       puts "Instances running"
       puts "result = |||#{result}|||"
    else
       err "Impossible to run appscale instances"
       return false
    end

    # Start monitoring
    puts "Start monitoring"
    app_ips.each do |vm|

       # Find the role
       # TODO What if a machine has different roles?
       # role = :undefined
       # app_roles.each do |r, ips|
       #    ips.each do |ip|
       #       if vm == ip then role = r end
       #    end
       # end
```

```
      # Get all vm's roles
      roles = app_roles.select { |r, ips| ips.include?(vm) }      # Be
         careful,
      # Ruby 1.8.7 returns an array instead of a hash, so we get
         something like
      # [[:appengine, [2, 3, 4]], [:database, [3, 4]]] which is an array
          of
      # arrays with the role in the first place of the innermost arrays.

      # Monitor every one of them
      roles.each do |role_array|
         role = role_array[0]
         puts "Calling appscale_monitor on #{vm} as #{role}"
         appscale_monitor(vm, role)
      end
   end

   return true

end


# Stops an AppScale cloud.
def appscale_cloud_stop(vm)

   user = resource[:vm_user]

   # Terminate instances
   command = "/root/appscale-tools/bin/appscale-terminate-instances"
   CloudSSH.execute_remote(command, user, vm)

end


# Monitors a virtual machine belonging to an AppScale cloud.
def appscale_monitor(vm, role)

   command = "ps aux"
   user = resource[:vm_user]
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "[AppScale monitor] Impossible to execute #{command} in #{vm}"
      return
   end

   # AppMonitoring calls god, so look for god processes who look like
      this:
   # /usr/bin/ruby1.8 /usr/bin/god -c /root/appscale/AppMonitoring/
      config/global.god
   if out.include? "/usr/bin/god -c /root/appscale/AppMonitoring"

      # AppMonitoring is running
      puts "[AppScale monitor] AppMonitoring is running"

      # Check the appscale controller is running
```

```
      if out.include? "AppController/djinnServer.rb"

         # Everything looks right so let AppScale handle it
         puts "[AppScale monitor] AppController is running"

      else

         # If AppMonitoring is running but the AppController is not,
         # AppMonitoring will take care
         puts "[AppScale monitor] AppMonitoring will take care of
             AppController"

      end

   else

      # AppMonitoring is not running
      puts "[AppScale monitor] AppMonitoring is not running"

      if out.include? "AppController/djinnServer.rb"

         # AppController is running
         puts "[AppScale monitor] AppController is running"

      end

      # Try to start the AppMonitoring
      puts "[AppScale monitor] Starting AppMonitoring on #{vm}"
      command = "/etc/init.d/appscale-monitoring start"
      out, success = CloudSSH.execute_remote(command, user, vm)
      if success
         puts "[AppScale monitor] Successfully started AppMonitoring"
      else
         err "[AppScale monitor] Impossible to start AppMonitoring in #{
             vm}"
         return
      end

      # Since the AppMonitoring will take care of the AppController, we
          do
      # not have to start the AppController


   end

   # If we have made it so far, let's try to apply the manifest

   # Copy the manifest
   puts "Copying manifest"
   path = "/etc/puppet/modules/appscale/files/appscale-manifests/basic.
      pp"
   out, success = CloudSSH.copy_remote(path, vm, "/tmp")
   unless success
      err "[AppScale monitor] Impossible to copy basic manifest to #{vm
          }"
      return
```

```
    end

    # Apply the manifest
    puts "Applying manifest"
    command = "puppet apply /tmp/basic.pp"
    out, success = CloudSSH.execute_remote(command, user, vm)
    unless success
       err "[AppScale monitor] Impossible to run puppet in #{vm}"
       return
    end

    # Analyze the output
    if out.include? "should be directory (noop)"
       err "[AppScale monitor] Missing directory in #{vm}"
       puts out
       return
    end

end
```

F.2.7 appscale_parsing.rb

```
# Appscale parsing functions to obtain virtual machine's data from
    manifest's
# arguments.
# All different roles have been obtained from the node_layout file
    located at
# appscale-tools/lib/node_layout.rb

################################################################################

# Default deployment
################################################################################


# Obtains the IP addresses and disk images from the resource[:controller
    ],
# and resource[:servers] arguments.
# appscale { 'myappscale'
#   ...
# }
def obtain_appscale_data_default(controller, servers)

#                          master, appengine, database, login, open,
#                          zookeeper, memcache)

    ips, ip_roles = appscale_parse_ips_default(controller, servers)
    img_roles     = appscale_parse_images_default(controller, servers)

    return ips, ip_roles, img_roles

end
```

```
# Obtains the IP addresses from the resource[:controller], resource[:
    server] and
# resource[:database] arguments.
# appscale { 'myappscale'
#   ...
# }
def appscale_parse_ips_default(controller, servers)

   ips_index = 0     # Because the IPs are in the first component of the
                     # controller and server arrays, and that is array
                        [0]
   ips = []
   ip_roles = {}


   # Get the IPs that are under the "controller" and "servers"
      attributes
   ip_roles[:controller] = []
   ip_roles[:controller] << controller[ips_index].chomp

   path = servers[ips_index]
   ip_roles[:servers] = get_from_file(path)

   # Add the IPs to the array
   ips = ips + ip_roles[:controller]
   ips = ips + ip_roles[:servers]

   ips = ips.uniq

   return ips, ip_roles

end


# Obtains the disk images from the resource[:controller], and resource[:
    servers]
# arguments.
# appscale { 'myappscale'
#   ...
# }
def appscale_parse_images_default(controller, servers)

   img_index = 1     # Because the images are in the second component of
       the
                     # controller and server arrays, and that is array
                        [1]
   img_roles = {}


   # Get the disk images that are under the "controller" and "servers"
   # attributes
   img_roles[:controller] = []
   img_roles[:controller] << controller[img_index].chomp

   path = servers[img_index]
   img_roles[:servers] = get_from_file(path)
```

```
   return img_roles

end


##############################################################################

# Custom deployment
##############################################################################


def obtain_appscale_data_custom(master, appengine, database, login, open
   ,
                                zookeeper, memcache)

   ips, ip_roles = appscale_parse_ips_custom(master, appengine, database
      , login,
                      open, zookeeper, memcache)
   img_roles     = appscale_parse_images_custom(master, appengine,
      database,
                      login, open, zookeeper, memcache)

   return ips, ip_roles, img_roles

end


# Obtains the IP addresses from the resource[:controller], resource[:
   server] and
# resource[:database] arguments.
# appscale { 'myappscale'
#   ...
# }
def appscale_parse_ips_custom(master, appengine, database, login, open,
                              zookeeper, memcache)

   ips_index = 0     # Because the IPs are in the first component of the
                     # controller and server arrays, and that is array
                        [0]
   ips = []
   ip_roles = {}


   # Get the IPs that are under the "master", "appengine", "database", "
      login",
   # "open", "zookeeper" and "memcache" attributes
   ip_roles[:master] = []
   ip_roles[:master] << master[ips_index].chomp

   path = appengine[ips_index]
   ip_roles[:appengine] = get_from_file(path)

   path = database[ips_index]
   ip_roles[:database] = get_from_file(path)
```

```
    ip_roles[:login] = []
    ip_roles[:login] << login[ips_index].chomp

    path = open[ips_index]
    ip_roles[:open] = get_from_file(path)

    path = zookeeper[ips_index]
    ip_roles[:zookeeper] = get_from_file(path)

    path = memcache[ips_index]
    ip_roles[:memcache] = get_from_file(path)

    # Add the IPs to the array
    ips = ips + ip_roles[:master]
    ips = ips + ip_roles[:appengine]
    ips = ips + ip_roles[:database]
    ips = ips + ip_roles[:login]
    ips = ips + ip_roles[:open]
    ips = ips + ip_roles[:zookeeper]
    ips = ips + ip_roles[:memcache]

    ips = ips.uniq

    return ips, ip_roles

end


# Obtains the disk images from the resource[:controller], and resource[:
    servers]
# arguments.
# appscale { 'myappscale'
#   ...
# }
def appscale_parse_images_custom(master, appengine, database, login,
    open,
                                     zookeeper, memcache)

    img_index = 1     # Because the images are in the second component of
        the
                      # controller and server arrays, and that is array
                         [1]
    img_roles = {}


    # Get the disk images that are under the "master", "appengine", "
       database",
    # "login", "open", "zookeeper" and "memcache" attributes
    img_roles[:master] = []
    img_roles[:master] << master[img_index].chomp

    path = appengine[img_index]
    img_roles[:appengine] = get_from_file(path)

    path = database[img_index]
    img_roles[:database] = get_from_file(path)
```

```
    img_roles[:login] = []
    img_roles[:login] << login[img_index].chomp

    path = open[img_index]
    img_roles[:open] = get_from_file(path)

    path = zookeeper[img_index]
    img_roles[:zookeeper] = get_from_file(path)

    path = memcache[img_index]
    img_roles[:memcache] = get_from_file(path)

    return img_roles

end


################################################################################

# Auxiliar functions
################################################################################


# Gets all the file lines in an array.
def get_from_file(path)

    array = []
    file = File.open(path)
    if file != nil
       array = file.readlines.map(&:chomp)    # Discard the final '\n'
       file.close
    end

    return array

end
```

F.2.8 appscale-add-keypair.tcl

```
#!/usr/bin/env expect

# Description:
#   Interacts with the appscale-add-keypair tool
#
# Synopsis:
#   appscale-add-keypair.tcl <password>
#
# Arguments:
#   - File: AppScale YAML configuration file.
#   - Password: Root password for all machines.
#
# Examples:
#   _$: appscale-add-keypair.tcl ips.yaml my_password_is_abcd
```

```
#
#
# Author:
#   David Ceresuela


# Procedure to interact with appscale-add-keypair command
# Parameter : password
proc addkeypair { password } {
  expect {
    # Send password
    -re "SSH password of root:" { exp_send "$password\r"
                                  exp_continue }

    # Tell expect stay in this 'expect' block and for each character
       that
    # appscale-add-keypair prints while doing the copy
    # Reset the timeout counter back to 0
    -re .                { exp_continue  }
    timeout              { return 1      }

    # Returning 0 as appscale-add-keypair was successful
    eof                  { return 0      }
  }
}

#Parsing command-line arguments
set yaml     [lrange $argv 0 0]
set password [lrange $argv 1 1]

#Setting timeout to an arbitrary value of 120 that works well for
   appscale-add-keypair
set timeout 120

# Execute appscale-add-keypair command
eval spawn /usr/local/appscale-tools/bin/appscale-add-keypair --ips
   $yaml --auto

#Get the result of appscale-add-keypair
set addkeypair_result [addkeypair $password]

# If appscale-add-keypair was successful
if { $addkeypair_result == 0 } {
  #Exit with zero status
  exit 0
}

# Error attempting appscale-add-keypair, so exit with non-zero status
exit 1
```

F.3 torque

F.3.1 torque.rb

```
Puppet::Type.newtype(:torque) do
   @doc = "Manages Torque clouds formed by KVM virtual machines."


   ensurable do

      desc "The cloud's ensure field can assume one of the following
         values:
   `running`: The cloud is running.
   `stopped`: The cloud is stopped.\n"

      newvalue(:stopped) do
         provider.stop
      end

      newvalue(:running) do
         provider.start
      end

   end


   # General parameters

   newparam(:name) do
      desc "The cloud name"
      isnamevar
   end

#    newparam(:ip_file) do
#       desc "The file with the cloud description in YAML format"
#    end
#
#    newparam(:img_file) do
#       desc "The file containing the qemu image(s). You must either
   provide " +
#            "one image from which all copies shall be made or provide "
   +
#            "an image for every instance"
#    end

   newparam(:vm_domain) do
      desc "The XML file with the virtual machine domain definition. " +
           "Libvirt XML format must be used"
   end

   newproperty(:pool, :array_matching => :all) do
      desc "The pool of physical machines"
   end


   # Virtual machine parameters
   newparam(:vm_mem) do
      desc "The virtual machine's maximum amopunt of memory. " +
           "In KiB: 2**10 (or blocks of 1024 bytes)."
```

```
      defaultto "1048576"
   end

   newparam(:vm_ncpu) do
      desc "The virtual machine's number of CPUs"
      defaultto "1"
   end


   # Infrastructure parameters

   newparam(:pm_user) do
      desc "The physical machines' user. It must have proper permissions
         "
      defaultto "dceresuela"
   end

   newparam(:pm_password) do
      desc "The physical machines' password"
      defaultto ""
   end

   newparam(:starting_mac_address) do
      desc "Starting MAC address for new virtual machines"
      defaultto "52:54:00:01:00:00"
   end

   newparam(:vm_user) do
      desc "Virtual machines' user"
      defaultto "root"
   end

   newparam(:root_password) do
      desc "Virtual machines' root password"
      defaultto "root"
   end


   # Torque parameters
   newproperty(:head, :array_matching => :all) do
      desc "The head node's information"
   end

   newproperty(:compute, :array_matching => :all) do
      desc "The compute nodes' information"
   end

end
```

F.3.2 torquep_helper.rb

```
################################################################################

# Auxiliar functions for torque provider
```

```
#################################################################################


# The functions in this file are defined the same in all providers, but
    each
# one implements them in their own way. Thus, the headers cannot be
    modified.

# Starts a torque cloud formed by <vm_ips> performing <vm_ip_roles>.
def start_cloud(vm_ips, vm_ip_roles)

   puts "Starting the cloud"
   return torque_cloud_start(vm_ip_roles)

end


# Obtains vm data from manifest parameters.
def obtain_vm_data()

   puts "Obtaining virtual machines' data"
   return obtain_torque_data(resource[:head], resource[:compute])

end
```

F.3.3 torquep.rb

```
Puppet::Type.type(:torque).provide(:torquep) do
   desc "Manages Torque clouds formed by KVM virtual machines"

   # Require torque auxiliar files
   require File.dirname(__FILE__) + '/torque/torque_yaml.rb'
   require File.dirname(__FILE__) + '/torque/torque_functions.rb'
   require File.dirname(__FILE__) + '/torque/torque_parsing.rb'

   # Require generic files
   require '/etc/puppet/modules/generic-module/provider/
      mcollective_client.rb'
   Dir["/etc/puppet/modules/generic-module/provider/*.rb"].each { |file|
       require file }

   # Commands needed to make the provider suitable
   commands :ping => "/bin/ping"
   commands :grep => "/bin/grep"
   commands :ps   => "/bin/ps"

   # Operating system restrictions
   confine :osfamily => "Debian"

   # Some constants
   #   They are in the generic cloud files

   # Makes sure the cloud is running.
   def start
```

```
puts "Starting cloud %s" % [resource[:name]]

# Check existence
if !exists?
   # Cloud does not exist => Startup operations

   # Check pool of physical machines
   puts "Checking pool of physical machines..."
   pm_up, pm_down = check_pool()
   unless pm_down.empty?
      puts "Some physical machines are down"
      pm_down.each do |pm|
         puts " - #{pm}"
      end
   end

   # Obtain the virtual machines' IPs
   puts "Obtaining the virtual machines' IPs..."
   #vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data(method(:
      torque_yaml_ips),
   #                                                  method(:
      torque_yaml_images))
   vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data()


   # Check whether you are one of the virtual machines
   puts "Checking whether this machine is part of the cloud..."
   part_of_cloud = vm_ips.include?(MY_IP)
   if part_of_cloud
      puts "#{MY_IP} is part of the cloud"

      # Check if you are the leader
      if leader?()
         leader_start("torque", vm_ips, vm_ip_roles, vm_img_roles,
             pm_up,
                      method(:torque_monitor))
      else
         common_start()
      end
   else
      puts "#{MY_IP} is not part of the cloud"
      not_cloud_start("torque", vm_ips, vm_ip_roles, vm_img_roles,
          pm_up)
   end

else

   # Cloud exists => Monitoring operations
   puts "Cloud already started"

   # Check if you are the leader
   if leader?()
      leader_monitoring(method(:torque_monitor))
   else
      puts "#{MY_IP} is not the leader"      # Nothing to do
```

```
        end
    end

end


# Makes sure the cloud is not running.
def stop

    puts "Stopping cloud %s" % [resource[:name]]

    if !exists?
        err "Cloud does not exist"
        return
    end
    if status != :running
        err "Cloud is not running"
        return
    end
    if exists? && status == :running

        puts "It is a torque cloud"

        # Stop cloud infrastructure
        #vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data(method(:
            torque_yaml_ips),
        #                                                   method(:
            torque_yaml_images))
        vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data()
        torque_cloud_stop(vm_ip_roles)

        # Shutdown and undefine all virtual machines explicitly created
            for this cloud
        shutdown_vms()

        # Stop cron jobs on all machines
        stop_cron_jobs("torque")      # TODO Check order

        # Delete files
        delete_files()

        # Note: As all the files deleted so far are located in the /tmp
            directory
        # only the machines that are still alive need to delete these
            files.
        # If the machine was shut down, these files will not be there
            the next
        # time it is started, so there is no need to delete them.

        puts "===================="
        puts "== Cloud stopped =="
        puts "===================="

    end

end
```

```
    def status
       if File.exists?("/tmp/cloud-#{resource[:name]}")
          return :running
       else
          return :stopped
       end
    end


    # Ensure methods
    #def create
    #   return true
    #end


    #def destroy
    #   return true
    #end


    def exists?
       return File.exists?("/tmp/cloud-#{resource[:name]}")
    end


    ###############################################################################

    # Properties need methods
    ###############################################################################

    def pool
    end

    def pm_user
    end

    def starting_mac_address
    end

    def root_password
    end

    def head
    end

    def compute
    end

end
```

F.3.4 torque_parsing.rb

```
# Torque parsing functions to obtain virual machine's data from manifest
   's
# arguments.

# Obtains the IP addresses and disk images from the resource[:head] and
# resource[:compute] arguments.
# torque { 'mytorque'
#   ...
#   head => ["155.210.155.73", "/var/tmp/master.img"],
#   compute => ["/files/compute-ip.txt", "/files/compute-img.txt"],
#   ...
# }
def obtain_torque_data(head, compute)

   ips, ip_roles = torque_parse_ips(head, compute)
   img_roles     = torque_parse_images(head, compute)
   return ips, ip_roles, img_roles

end


# Obtains the IP addresses from the resource[:head] and resource[:
   compute]
# arguments.
# torque { 'mytorque'
#   ...
#   head => ["155.210.155.73", "/var/tmp/master.img"],
#   compute => ["/files/compute-ip.txt", "/files/compute-img.txt"],
#   ...
# }
def torque_parse_ips(head, compute)

   ips_index = 0     # Because the IPs are in the first component of the
       head
                     # and compute arrays, and that is array[0]
   ips = []
   ip_roles = {}

   # Get the IPs that are under the "head" and "compute" attributes
   ip_roles[:head] = []
   ip_roles[:head] << head[ips_index].chomp

   path = compute[ips_index]
   ip_roles[:compute] = get_from_file(path)

   # Add the IPs to the array
   ips = ips + ip_roles[:head]
   ips = ips + ip_roles[:compute]

   ips = ips.uniq

   return ips, ip_roles

end
```

```
# Obtains the disk images from the resource[:head] and resource[:compute
   ]
# arguments.
# torque { 'mytorque'
#   ...
#   head => ["155.210.155.73", "/var/tmp/master.img"],
#   compute => ["/files/compute-ip.txt", "/files/compute-img.txt"],
#   ...
# }
def torque_parse_images(head, compute)

   img_index = 1     # Because the images are in the first component of
      the head
                     # and compute arrays, and that is array[1]
   img_roles = {}


   # Get the disk images that are under the "head" and "compute"
      attributes
   img_roles[:head] = []
   img_roles[:head] << head[img_index].chomp

   path = compute[img_index]
   img_roles[:compute] = get_from_file(path)

   return img_roles

end


################################################################################

# Auxiliar functions
################################################################################


# Gets all the file lines in an array.
def get_from_file(path)

   array = []
   file = File.open(path)
   if file != nil
      array = file.readlines.map(&:chomp)    # Discard the final '\n'
      file.close
   end

   return array

end
```

F.3.5 torque_functions.rb

```
# Starts a torque cloud.
def torque_cloud_start(torque_roles)

   head    = torque_roles[:head]
   compute = torque_roles[:compute]

   # Start services

   # Start head node
   start_head(head)

   # Start compute nodes
   compute.each do |vm|
      start_compute(vm, head)
   end


   # Start monitoring
   torque_monitor(head, :head)
   compute.each do |vm|
      torque_monitor(vm, :compute)
   end

   return true

end


################################################################################

# Start node functions
################################################################################


# Starts a head node.
def start_head(head)

   user = resource[:vm_user]
   puts "Starting trqauthd on head node"
   check_command = "ps aux | grep -v grep | grep trqauthd"
   out, success = CloudSSH.execute_remote(check_command, user, head)
   unless success
      command = "/etc/init.d/trqauthd start > /dev/null 2> /dev/null"
      out, success = CloudSSH.execute_remote(command, user, head)
      unless success
         err "Impossible to start trqauthd in #{head}"
         return false
      end
   end

   puts "Starting pbs_server on head node"
   check_command = "ps aux | grep -v grep | grep pbs_server"
   out, success = CloudSSH.execute_remote(check_command, user, head)
   unless success
      command = "/bin/bash /root/cloud/torque/start-pbs-server"
      out, success = CloudSSH.execute_remote(command, user, head)
```

```
      unless success
         err "Impossible to start pbs_server in #{head}"
         return false
      end
   end

   puts "Starting pbs_sched on head node"
   check_command = "ps aux | grep -v grep | grep pbs_sched"
   out, success = CloudSSH.execute_remote(check_command, user, head)
   unless success
      command = "/bin/bash /root/cloud/torque/start-pbs-sched"
      out, success = CloudSSH.execute_remote(command, user, head)
      unless success
         err "Impossible to start pbs_sched in #{head}"
         return false
      end
   end

end


# Starts a compute node.
def start_compute(compute, head)

   user = resource[:vm_user]
   puts "Starting pbs_mom on compute nodes"
   check_command = "ps aux | grep -v grep | grep pbs_mom"
   command = "/bin/bash /root/cloud/torque/start-pbs-mom"
   out, success = CloudSSH.execute_remote(check_command, user, compute)
   unless success
      out, success = CloudSSH.execute_remote(command, user, compute)
      unless success
         err "Impossible to start pbs_mom in #{compute}"
         return false
      end

      # Add the node to the compute node list on head
      add_compute_node(compute, head)
   end

end


################################################################################

# Monitor node functions
################################################################################


# Monitors a virtual machine belonging to a torque cloud.
def torque_monitor(vm, role)

   if role == :head
      puts "[Torque monitor] Monitoring head"
      monitor_head(vm)
      puts "[Torque monitor] Monitored head"
```

```
    elsif role == :compute
        puts "[Torque monitor] Monitoring compute"
        monitor_compute(vm)
        puts "[Torque monitor] Monitored compute"

    else
        puts "[Torque monitor] Unknown role: #{role}"
    end

end


# Monitors a head node.
def monitor_head(vm)

    user = resource[:vm_user]

    check_command1 = "ps aux | grep -v grep | grep trqauthd"
    check_command2 = "ps aux | grep -v grep | grep god | grep pbs-server.
        god"
    check_command3 = "ps aux | grep -v grep | grep god | grep pbs-sched.
        god"

    out1, success1 = CloudSSH.execute_remote(check_command1, user, vm)
    out2, success2 = CloudSSH.execute_remote(check_command2, user, vm)
    out3, success3 = CloudSSH.execute_remote(check_command3, user, vm)
    unless success1 && success2 && success3
        puts "[Torque monitor] God or trqauthd are not running in #{vm}"

        # Try to start monitoring again
        puts "[Torque monitor] Starting monitoring head on #{vm}"
        if start_monitor_head(vm)
            puts "[Torque monitor] Successfully started to monitor head on
                #{vm}"
        else
            err "[Torque monitor] Impossible to monitor head on #{vm}"
        end
    end

end


# Monitors a compute node.
def monitor_compute(vm)

    user = resource[:vm_user]

    # Obtain head node's IP
    #vm_ips, vm_ip_roles = torque_yaml_ips(resource[:ip_file])
    vm_ips, vm_ip_roles = torque_parse_ips(resource[:head], resource[:
        compute])
    head = vm_ip_roles[:head]

    # Check if the node is in the list of compute nodes
```

```
    command = "qmgr -c \"list node @localhost\""      # Get a list of all
        nodes
    out, success = CloudSSH.execute_remote(command, user, head)
    unless success
       err "[Torque monitor] Impossible to obtain node list from #{head}"
    end

    command = "hostname"
    out2, success = CloudSSH.execute_remote(command, user, vm)
    unless success
       err "[Torque monitor] Impossible to obtain hostname for #{vm}"
    end

    # Add the node to the list of compute nodes
    hostname = out2.chomp()
    if out.include? "Node #{hostname}"
       puts "[Torque monitor] #{hostname} (#{vm}) already in head's list
          node"
    else
       puts "[Torque monitor] Adding #{hostname} to the list of compute
          nodes"
       add_compute_node(vm, head)
    end

    # Start monitoring
    check_command = "ps aux | grep -v grep | grep god | grep pbs-mom.god"
    out, success = CloudSSH.execute_remote(check_command, user, vm)
    unless success
       puts "[Torque monitor] God is not running in #{vm}"

       # Try to start monitoring again
       puts "[Torque monitor] Starting monitoring compute on #{vm}"
       if start_monitor_compute(vm)
          puts "[Torque monitor] Successfully started to monitor compute
             on #{vm}"
       else
          err "[Torque monitor] Impossible to monitor compute on #{vm}"
       end
    end

end


################################################################################

# Stop functions
################################################################################


# Stops a torque cloud.
def torque_cloud_stop(torque_roles)

    head    = torque_roles[:head]
    compute = torque_roles[:compute]

    # Stop compute nodes
```

```
    compute.each do |vm|
       stop_compute(vm, head)
    end

    # Stop head node
    stop_head(head)

end


# Stops a head node.
def stop_head(head)

    user = resource[:vm_user]

    puts "Stopping pbs_sched on head node"
    command = 'pkill -f pbs-sched\(.\)god'    # We are looking for pbs-
       sched.god
    out, success = CloudSSH.execute_remote(command, user, head)
    if success
       command = "pkill pbs_sched"
       out, success = CloudSSH.execute_remote(command, user, head)
       unless success
          err "Impossible to stop pbs_sched in #{head}"
          return false
       end
    else
       err "Impossible to stop pbs_sched monitoring in #{head}"
    end

    puts "Stopping pbs_server on head node"
    command = 'pkill -f pbs-server\(.\)god'   # We are looking for pbs-
       server.god
    out, success = CloudSSH.execute_remote(command, user, head)
    if success
       command = "pkill pbs_server"
       out, success = CloudSSH.execute_remote(command, user, head)
       unless success
          err "Impossible to stop pbs_server in #{head}"
          return false
       end
    else
       err "Impossible to stop pbs_server monitoring in #{head}"
    end

    puts "Stopping trqauthd on head node"
    command = "/etc/init.d/trqauthd stop"     # Do not kill it, stop it
    out, success = CloudSSH.execute_remote(command, user, head)
    unless success
       err "Impossible to stop trqauthd in #{head}"
       return false
    end

end
```

```
# Stops a compute node.
def stop_compute(compute, head)

   user = resource[:vm_user]

   puts "Stopping pbs_mom on compute nodes"
   command = 'pkill -f pbs-mom\(.\)god'
   out, success = CloudSSH.execute_remote(command, user, compute)
   if success
      command = "pkill pbs_mom"
      out, success = CloudSSH.execute_remote(command, user, compute)
      unless success
         err "Impossible to stop pbs_mom in #{compute}"
         return false
      end

      # Add the node to the compute node list on head
      del_compute_node(compute, head)
   end

end


################################################################################

# Auxiliar start and stop node functions
################################################################################


# Adds a compute node to the list in the head node.
def add_compute_node(vm, head)

   user = resource[:vm_user]

   command = "hostname"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "Impossible to obtain hostname for #{vm}"
      return false
   end
   hostname = out.chomp()
   command = "qmgr -c \"create node #{hostname}\""
   out, success = CloudSSH.execute_remote(command, user, head)
   unless success
      err "Impossible to add #{hostname} as a compute node in #{head}"
      return false
   end
   command = "pbsnodes -c #{hostname}"
   out, success = CloudSSH.execute_remote(command, user, head)
   unless success
      err "Impossible to clear offline from #{hostname} in #{head}"
      return false
   end

end
```

```
# Deletes a compute node from the list in the head node.
def del_compute_node(vm, head)

   user = resource[:vm_user]

   command = "hostname"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "Impossible to obtain hostname for #{vm}"
      return false
   end
   hostname = out
   command = "qmgr -c \"delete node #{hostname}\""
   out, success = CloudSSH.execute_remote(command, user, head)
   unless success
      err "Impossible to delete #{hostname} as a compute node in #{head
         }"
      return false
   end

end


################################################################################

# Auxiliar monitor node functions
################################################################################


# Starts monitoring on head node.
def start_monitor_head(vm)

   user = resource[:vm_user]
   god_port = 17165

   # The trqauthd script is intelligent enough to be initiated as many
      times
   # as you want without problem: if it is already started it will not
      be
   # started again
   command = "/etc/init.d/trqauthd start > /dev/null 2> /dev/null"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "[Torque monitor] Impossible to run /etc/init.d/trqauthd start
          at #{vm}"
      return false
   end

   # Monitor head node pbs_server and pbs_sched processes with god

   # pbs_server is up and running
   path = "/etc/puppet/modules/torque/files/torque-god/pbs-server.god"
   command = "mkdir -p /etc/god"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
```

```
      err "[Torque monitor] Impossible to create /etc/god at #{vm}"
      return false
   end
   out, success = CloudSSH.copy_remote(path, vm, "/etc/god")
   unless success
      err "[Torque monitor] Impossible to copy #{path} to #{vm}"
      return false
   end
   port = god_port
   command = "god -c /etc/god/pbs-server.god -p #{port}"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "[Torque monitor] Impossible to run god in #{vm}"
      return false
   end

   # pbs_sched is up and running
   path = "/etc/puppet/modules/torque/files/torque-god/pbs-sched.god"
   command = "mkdir -p /etc/god"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "[Torque monitor] Impossible to create /etc/god at #{vm}"
      return false
   end
   out, success = CloudSSH.copy_remote(path, vm, "/etc/god")
   unless success
      err "[Torque monitor] Impossible to copy #{path} to #{vm}"
      return false
   end
   port += 1
   command = "god -c /etc/god/pbs-sched.god -p #{port}"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "[Torque monitor] Impossible to run god in #{vm}"
      return false
   end

   return true

end


# Starts monitoring on compute node.
def start_monitor_compute(vm)

   user = resource[:vm_user]
   god_port = 17165

   # Monitor compute node with god: pbs_mom is up and running
   path = "/etc/puppet/modules/torque/files/torque-god/pbs-mom.god"
   command = "mkdir -p /etc/god"
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "[Torque monitor] Impossible to create /etc/god at #{vm}"
      return false
   end
```

```
    out, success = CloudSSH.copy_remote(path, vm, "/etc/god")
    unless success
       err "[Torque monitor] Impossible to copy #{path} to #{vm}"
       return false
    end
    command = "god -c /etc/god/pbs-mom.god -p #{god_port}"
    out, success = CloudSSH.execute_remote(command, user, vm)
    unless success
       err "[Torque monitor] Impossible to run god in #{vm}"
       return false
    end

    return true

end
```

F.3.6 torque_yaml.rb

```
require 'yaml'


# Obtains the IP addresses from the ip_file file. It does NOT check
    whether
# the file has the proper format.
def torque_yaml_ips(path)

    ips = []
    ip_roles = {}

    file = File.open(path)
    tree = YAML::parse(file)

    if tree != nil

       tree = tree.transform

       # Deployment: head node + compute nodes
       head      = tree[:head]
       compute   = tree[:compute]

       # Get the IPs that are under the "head" and "compute" labels
       ip_roles[:head]      = get_elements(head)
       ip_roles[:compute]   = get_elements(compute)

       # Add the IPs to the array
       ips = ips + ip_roles[:head]
       ips = ips + ip_roles[:compute]

       ips = ips.uniq

       file.close

       return ips, ip_roles
    end
```

```
end


# Obtains the disk images from the img_file file.
def torque_yaml_images(path)

   img_roles = {}

   file = File.open(path)
   tree = YAML::parse(file)

   if tree != nil

      tree = tree.transform

      # Deployment: head node + compute nodes
      head      = tree[:head]
      compute   = tree[:compute]

      # Maybe we have been given only an image for all virtual machines
      all       = tree[:all]

      if all == nil
         img_roles[:head]      = get_elements(head)
         img_roles[:compute]   = get_elements(compute)
      else
         img_roles[:all]       = get_elements(all)
      end

      file.close

      return img_roles
   end

end


# Transforms the given elements to an array.
def get_elements(array)

   elements = []
   if array != nil
      elements = array.to_a
   end
   return elements

end
```

F.4 web

F.4.1 web.rb

```
Puppet::Type.newtype(:web) do
   @doc = "Manages web clouds formed by KVM virtual machines."


   ensurable do

      desc "The cloud's ensure field can assume one of the following
         values:
   `running`: The cloud is running.
   `stopped`: The cloud is stopped.\n"

      newvalue(:stopped) do
         provider.stop
      end

      newvalue(:running) do
         provider.start
      end

   end


   # General parameters

   newparam(:name) do
      desc "The cloud name"
      isnamevar
   end

#    newparam(:ip_file) do
#       desc "The file with the cloud description in YAML format"
#    end
#
#    newparam(:img_file) do
#       desc "The file containing the qemu image(s). You must either
   provide " +
#            "one image from which all copies shall be made or provide "
   +
#            "an image for every instance"
#    end

   newparam(:vm_domain) do
      desc "The XML file with the virtual machine domain definition. " +
           "Libvirt XML format must be used"
   end

   newproperty(:pool, :array_matching => :all) do
      desc "The pool of physical machines"
   end


   # Virtual machine parameters
   newparam(:vm_mem) do
      desc "The virtual machine's maximum amopunt of memory. " +
           "In KiB: 2**10 (or blocks of 1024 bytes)."
```

```
      defaultto "1048576"
   end

   newparam(:vm_ncpu) do
      desc "The virtual machine's number of CPUs"
      defaultto "1"
   end


   # Infrastructure parameters

   newparam(:pm_user) do
      desc "The physical machines' user. It must have proper permissions
         "
      defaultto "dceresuela"
   end

   newparam(:pm_password) do
      desc "The physical machines' password"
      defaultto ""
   end

   newparam(:starting_mac_address) do
      desc "Starting MAC address for new virtual machines"
      defaultto "52:54:00:01:00:00"
   end

   newparam(:vm_user) do
      desc "Virtual machines' user"
      defaultto "root"
   end

   newparam(:root_password) do
      desc "Virtual machines' root password"
      defaultto "root"
   end


   # Web parameters

   newproperty(:balancer, :array_matching => :all) do
      desc "The balancer node's information"
   end

   newproperty(:server, :array_matching => :all) do
      desc "The server nodes' information"
   end

   newproperty(:database, :array_matching => :all) do
      desc "The database node's information"
   end

end
```

F.4.2 webp_helper.rb

```
##############################################################################

# Auxiliar functions for web provider
##############################################################################


# The functions in this file are defined the same in all providers, but
    each
# one implements them in their own way. Thus, the headers cannot be
    modified.

# Starts a torque cloud formed by <vm_ips> performing <vm_ip_roles>.
def start_cloud(vm_ips, vm_ip_roles)

   puts "Starting the cloud"
   puts  "Starting a web cloud"

   # SSH keys have already been distributed when machines were
      monitorized,
   # so we do not have to distribute them again

   # Start web cloud
   return web_cloud_start(vm_ip_roles)

end


# Obtains vm data from manifest parameters.
def obtain_vm_data()

   puts "Obtaining virtual machines' data"
   return obtain_web_data(resource[:balancer], resource[:server],
                          resource[:database])

end
```

F.4.3 webp.rb

```
Puppet::Type.type(:web).provide(:webp) do
   desc "Manages web clouds formed by KVM virtual machines"

   # Require web auxiliar files
   require File.dirname(__FILE__) + '/web/web_yaml.rb'
   require File.dirname(__FILE__) + '/web/web_functions.rb'
   require File.dirname(__FILE__) + '/web/web_parsing.rb'

   # Require generic files
   require '/etc/puppet/modules/generic-module/provider/
      mcollective_client.rb'
   Dir["/etc/puppet/modules/generic-module/provider/*.rb"].each { |file|
       require file }
```

```
# Commands needed to make the provider suitable
commands :ping => "/bin/ping"
commands :grep => "/bin/grep"
commands :ps   => "/bin/ps"

# Operating system restrictions
confine :osfamily => "Debian"

# Some constants
#   They are in the generic cloud files

# Makes sure the cloud is running.
def start

   puts "Starting cloud %s" % [resource[:name]]

   # Check existence
   if !exists?
      # Cloud does not exist => Startup operations

      # Check pool of physical machines
      puts "Checking pool of physical machines..."
      pm_up, pm_down = check_pool()
      unless pm_down.empty?
         puts "Some physical machines are down"
         pm_down.each do |pm|
            puts " - #{pm}"
         end
      end

      # Obtain the virtual machines' IPs
      puts "Obtaining the virtual machines' IPs..."
      #vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data(method(:
         web_yaml_ips),
      #                                                  method(:
         web_yaml_images))
      vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data()

      # Check whether you are one of the virtual machines
      puts "Checking whether this machine is part of the cloud..."
      part_of_cloud = vm_ips.include?(MY_IP)
      if part_of_cloud
         puts "#{MY_IP} is part of the cloud"

         # Check if you are the leader
         if leader?()
            leader_start("web", vm_ips, vm_ip_roles, vm_img_roles,
               pm_up,
                        method(:web_monitor))
         else
            common_start()
         end
      else
         puts "#{MY_IP} is not part of the cloud"
```

```
            not_cloud_start("web", vm_ips, vm_ip_roles, vm_img_roles,
                pm_up)
        end

    else

        # Cloud exists => Monitoring operations
        puts "Cloud already started"

        # Check if you are the leader
        if leader?()
            leader_monitoring(method(:web_monitor))
        else
            puts "#{MY_IP} is not the leader"      # Nothing to do
        end
    end

end


# Makes sure the cloud is not running.
def stop

    puts "Stopping cloud %s" % [resource[:name]]

    if !exists?
        err "Cloud does not exist"
        return
    end
    if status != :running
        err "Cloud is not running"
        return
    end
    if exists? && status == :running

        puts "It is a web cloud"

        # Stop cloud infrastructure
        #vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data(method(:
            web_yaml_ips),
        #                                                   method(:
            web_yaml_images))
        vm_ips, vm_ip_roles, vm_img_roles = obtain_vm_data()
        web_cloud_stop(vm_ip_roles)

        # Shutdown and undefine all virtual machines explicitly created
            for this cloud
        shutdown_vms()

        # Stop cron jobs on all machines
        stop_cron_jobs("web")      # TODO Check order

        # Delete files
        delete_files()
```

```
        # Note: As all the files deleted so far are located in the /tmp
            directory
        # only the machines that are still alive need to delete these
            files.
        # If the machine was shut down, these files will not be there
            the next
        # time it is started, so there is no need to delete them.

        puts "===================="
        puts "== Cloud stopped =="
        puts "===================="

    end

end


def status
    if File.exists?("/tmp/cloud-#{resource[:name]}")
        return :running
    else
        return :stopped
    end
end


# Ensure methods
def create
    return true
end


def destroy
    return true
end


def exists?
    return File.exists?("/tmp/cloud-#{resource[:name]}")
end


##############################################################################

# Properties need methods
##############################################################################

def pool
end

def pm_user
end

def root_password
end
```

```
    def starting_mac_address
    end

    def balancer
    end

    def server
    end

    def database
    end

end
```

F.4.4 web_parsing.rb

```
# Web parsing functions to obtain virual machine's data from manifest's
# arguments.

# Obtains the IP addresses and disk images from the resource[:balancer]
# resource[:server] and resource[:database] arguments.
# web { 'myweb'
#   ...
#   balancer => ["155.210.155.175", "/var/tmp/dceresuela/lucid-lb.img"],
#   server   => ["/etc/puppet/modules/web/files/server-ips.txt",
#                "/etc/puppet/modules/web/files/server-imgs.txt"],
#   database => ["155.210.155.177", "/var/tmp/dceresuela/lucid-db.img"],
#   ...
# }
def obtain_web_data(balancer, server, database)

    ips, ip_roles = web_parse_ips(balancer, server, database)
    img_roles     = web_parse_images(balancer, server, database)

    return ips, ip_roles, img_roles

end


# Obtains the IP addresses from the resource[:balancer], resource[:
    server] and
# resource[:database] arguments.
# web { 'myweb'
#   ...
#   balancer => ["155.210.155.175", "/var/tmp/dceresuela/lucid-lb.img"],
#   server   => ["/etc/puppet/modules/web/files/server-ips.txt",
#                "/etc/puppet/modules/web/files/server-imgs.txt"],
#   database => ["155.210.155.177", "/var/tmp/dceresuela/lucid-db.img"],
#   ...
# }
def web_parse_ips(balancer, server, database)

    ips_index = 0     # Because the IPs are in the first component of the
                      # balancer, server and database arrays, and that is
```

```
                    # array[0]
    ips = []
    ip_roles = {}

    # Get the IPs that are under the "balancer", "server" and "database"
    # attributes
    ip_roles[:balancer] = []
    ip_roles[:balancer] << balancer[ips_index].chomp

    path = server[ips_index]
    ip_roles[:server] = get_from_file(path)

    ip_roles[:database] = []
    ip_roles[:database] << database[ips_index].chomp

    # Add the IPs to the array
    ips = ips + ip_roles[:balancer]
    ips = ips + ip_roles[:server]
    ips = ips + ip_roles[:database]

    ips = ips.uniq

    return ips, ip_roles

end


# Obtains the disk images from the resource[:balancer], resource[:server
    ] and
# resource[:database] arguments.
# web { 'myweb'
#   ...
#   balancer => ["155.210.155.175", "/var/tmp/dceresuela/lucid-lb.img"],
#   server   => ["/etc/puppet/modules/web/files/server-ips.txt",
#                "/etc/puppet/modules/web/files/server-imgs.txt"],
#   database => ["155.210.155.177", "/var/tmp/dceresuela/lucid-db.img"],
#   ...
# }
def web_parse_images(balancer, server, database)

    img_index = 1     # Because the images are in the second component of
        the
                      # balancer, server and database arrays, and that is
                      # array[1]
    img_roles = {}


    # Get the disk images that are under the "balancer", "server" and "
       database"
    # attributes
    img_roles[:balancer] = []
    img_roles[:balancer] << balancer[img_index].chomp

    path = server[img_index]
    img_roles[:server] = get_from_file(path)
```

```
    img_roles[:database] = []
    img_roles[:database] << database[img_index].chomp

    return img_roles

end


################################################################################

# Auxiliar functions
################################################################################


# Gets all the file lines in an array.
def get_from_file(path)

    array = []
    file = File.open(path)
    if file != nil
        array = file.readlines.map(&:chomp)    # Discard the final '\n'
        file.close
    end

    return array

end
```

F.4.5 web_functions.rb

```
# Starts a web cloud.
def web_cloud_start(web_roles)

    balancers = web_roles[:balancer]
    servers   = web_roles[:server]
    databases = web_roles[:database]

    # Start services

    # Start load balancers => Start nginx
    puts "Starting nginx on load balancers"
    balancers.each do |vm|
        start_balancer(vm)
    end

    # Start web servers => Start sinatra application
    puts "Starting ruby web3 on web servers"
    servers.each do |vm|
        start_server(vm)
    end

    # Database servers start at boot time, but check whether they have
        started
    # and if they have not, start them
```

```
    puts "Starting mysql on database servers"
    databases.each do |vm|
       start_database(vm)
    end


    # Start monitoring

    # Load balancers
    balancers.each do |vm|
       start_monitor_balancer(vm)
    end

    # Web servers
    servers.each do |vm|
       start_monitor_server(vm)
    end

    # Database servers
    databases.each do |vm|
       start_monitor_database(vm)
    end

    return true

end


################################################################################

# Start node functions
################################################################################


# Starts a load balancer.
def start_balancer(vm)

    command = "/etc/init.d/nginx start > /dev/null 2> /dev/null"
    user = resource[:vm_user]
    if vm == MY_IP
       result = `#{command}`
       unless $?.exitstatus == 0
          err "Impossible to start balancer in #{vm}"
          return false
       end
    else
       out, success = CloudSSH.execute_remote(command, user, vm)
       unless success
          err "Impossible to start balancer in #{vm}"
          return false
       end
    end

end
```

```
# Starts a web server.
def start_server(vm)

   # The ruby-web3 file should have been already copied at this point.
      It should
   # have been copied when installing the web server.
   command = "/etc/init.d/ruby-web3 start"
   user = resource[:vm_user]
   if vm == MY_IP
      result = `#{command}`
      unless $?.exitstatus == 0
         err "Impossible to start server in #{vm}"
         return false
      end
   else
      out, success = CloudSSH.execute_remote(command, user, vm)
      unless success
         err "Impossible to start server in #{vm}"
         return false
      end
   end

end


# Starts a database server.
def start_database(vm)

   check_command = "ps aux | grep -v grep | grep mysql"
   command = "/usr/bin/service mysql start"
   user = resource[:vm_user]
   if vm == MY_IP
      result = `#{check_command}`
      unless $?.exitstatus == 0
         result = `#{command}`
         unless $?.exitstatus == 0
            err "Impossible to start database in #{vm}"
            return false
         end
      end
   else
      out, success = CloudSSH.execute_remote(check_command, user, vm)
      unless success
         out, success = CloudSSH.execute_remote(command, user, vm)
         unless success
            err "Impossible to start database in #{vm}"
            return false
         end
      end
   end

end


################################################################################
```

```
# Monitor node functions
################################################################################


# Monitors a virtual machine belonging to a web cloud.
def web_monitor(vm, role)

   if role == :balancer
      puts "[Web monitor] Monitoring load balancer"

      # Run puppet
      unless start_monitor_balancer(vm)
         puts "[Web monitor] Impossible to monitor load balancer on #{vm
             }"
      end
      puts "[Web monitor] Monitored load balancer"

   elsif role == :server
      puts "[Web monitor] Monitoring web server"

      # Run puppet
      unless start_monitor_server(vm)
         puts "[Web monitor] Impossible to monitor web server on #{vm}"
      end

      puts "[Web monitor] Monitored web server"

   elsif role == :database
      puts "[Web monitor] Monitoring database"

      # Check god is running
      check_command = "ps aux | grep -v grep | grep god | grep database.
         god"
      user = resource[:vm_user]
      out, success = CloudSSH.execute_remote(check_command, user, vm)
      unless success
         puts "[Web monitor] God is not running in #{vm}"

         # Try to start monitoring again
         puts "[Web monitor] Starting monitoring database on #{vm}"
         if start_monitor_database(vm)
            puts "[Web monitor] Successfully started to monitor database
                on #{vm}"
         else
            err "[Web monitor] Impossible to monitor database on #{vm}"
         end
      end
      puts "[Web monitor] Monitored database"

   else
      puts "[Web monitor] Unknown role: #{role}"
   end

end
```

```
# Starts monitoring on load balancer.
def start_monitor_balancer(vm)

   # Copy the puppet manifest
   path = "/etc/puppet/modules/web/files/web-manifests/balancer.pp"
   out, success = CloudSSH.copy_remote(path, vm, "/tmp")
   unless success
      err "[Web monitor] Impossible to copy balancer manifest to #{vm}"
      return false
   end

   # Monitor load balancer with puppet
   # While god monitoring will be done in a loop this will only be done
      if
   # explicitly invoked, so we must call 'puppet apply' every time.
   command = "puppet apply /tmp/balancer.pp"
   user = resource[:vm_user]
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "[Web monitor] Impossible to run puppet in #{vm}"
      return false
   end

   return true

end


# Starts monitoring on web server.
def start_monitor_server(vm)

   # Copy the puppet manifests: general, start and stop
   path = "/etc/puppet/modules/web/files/web-manifests/server.pp"
   out, success = CloudSSH.copy_remote(path, vm, "/tmp")
   unless success
      err "[Web monitor] Impossible to copy server manifest to #{vm}"
      return false
   end

   path = "/etc/puppet/modules/web/files/web-manifests/server-start.pp"
   out, success = CloudSSH.copy_remote(path, vm, "/tmp")
   unless success
      err "[Web monitor] Impossible to copy server start manifest to #{
         vm}"
      return false
   end

   path = "/etc/puppet/modules/web/files/web-manifests/server-stop.pp"
   out, success = CloudSSH.copy_remote(path, vm, "/tmp")
   unless success
      err "[Web monitor] Impossible to copy server stop manifest to #{vm
         }"
      return false
   end
```

```
    # Monitor web servers with puppet: installation files and required
       gems
    command = "puppet apply /tmp/server.pp"
    user = resource[:vm_user]
    out, success = CloudSSH.execute_remote(command, user, vm)
    unless success
       err "[Web monitor] Impossible to run puppet (server.pp) in #{vm}"
       return false
    end

    # Monitor web servers with puppet: web server is up and running
    command = "puppet apply /tmp/server-start.pp"
    user = resource[:vm_user]
    out, success = CloudSSH.execute_remote(command, user, vm)
    unless success
       err "[Web monitor] Impossible to run puppet (server-start.pp) in
          #{vm}"
       return false
    end

    return true

end


# Starts monitoring on database.
def start_monitor_database(vm)

    user = resource[:vm_user]

    # Monitor database with god due to puppet vs ubuntu mysql bug
    # http://projects.puppetlabs.com/issues/12773
    # Therefore there is no puppet monitoring, only god
    path = "/etc/puppet/modules/web/files/web-god/database.god"
    command = "mkdir -p /etc/god"
    out, success = CloudSSH.execute_remote(command, user, vm)
    unless success
       err "[Web monitor] Impossible to create /etc/god at #{vm}"
       return false
    end
    out, success = CloudSSH.copy_remote(path, vm, "/etc/god")
    unless success
       err "[Web monitor] Impossible to copy #{path} to #{vm}"
       return false
    end
    command = "god -c /etc/god/database.god"
    out, success = CloudSSH.execute_remote(command, user, vm)
    unless success
       err "[Web monitor] Impossible to run god in #{vm}"
       return false
    end

    return true

end
```

```
#############################################################################

# Stop functions
#############################################################################


# Stops a web cloud.
def web_cloud_stop(web_roles)

   balancers = web_roles[:balancer]
   servers   = web_roles[:server]
   databases = web_roles[:database]

   # Stop services

   # Stop load balancers => Stop nginx
   puts "Stopping nginx on load balancers"
   balancers.each do |vm|
      stop_balancer(vm)
   end

   # Stop web servers => Stop sinatra application
   puts "Stopping ruby web3 on web servers"
   servers.each do |vm|
      stop_server(vm)
   end

   # Stop database servers => Stop mysql
   puts "Stopping mysql on database servers"
   databases.each do |vm|
      stop_database(vm)
   end

end


# Stops a load balancer.
def stop_balancer(vm)

   # It is being monitored explicitly with puppet, so we do not have to
      stop
   # monitoring explicitly, we just have to stop it

   command = "/etc/init.d/nginx stop > /dev/null 2> /dev/null"
   user = resource[:vm_user]
   out, success = CloudSSH.execute_remote(command, user, vm)
   unless success
      err "Impossible to stop balancer in #{vm}"
      return false
   end

end


# Stops a web server.
```

```
def stop_server(vm)

   # It is being monitored explicitly with puppet, so we do not have to
      stop
   # monitoring explicitly, we just have to stop it

   command = "/etc/init.d/ruby-web3 stop > /dev/null 2> /dev/null"
   user = resource[:vm_user]
   out, success = CloudSSH.execute_remote(command, user,vm)
   unless success
      err "Impossible to stop web server in #{vm}"
      return false
   end

end


# Stops a database server.
def stop_database(vm)

   user = resource[:vm_user]

   command = 'pkill -f database\(.\)god'    # We are looking for
      database.god
   out, success = CloudSSH.execute_remote(command, user, vm)
   if success
      command = "/usr/bin/service mysql stop"      # Do not kill it,
         stop it
      out, success = CloudSSH.execute_remote(command, user, vm)
      unless success
         err "Impossible to stop database in #{vm}"
         return false
      end
   else
      err "Impossible to stop database monitoring in #{vm}"
   end

end
```

F.4.6 web_yaml.rb

```
require 'yaml'


# Obtains the IP addresses from the ip_file file. It does NOT check
   whether
# the file has the proper format.
def web_yaml_ips(path)

   ips = []
   ip_roles = {}

   file = File.open(path)
   tree = YAML::parse(file)
```

```
    if tree != nil

        tree = tree.transform

        # Classic deployment: load balancer + web servers + database
        balancer = tree[:balancer]
        server   = tree[:server]
        database = tree[:database]

        # Get the IPs that are under the "balancer", "server" and "
            database" labels
        ip_roles[:balancer] = get_elements(balancer)
        ip_roles[:server]   = get_elements(server)
        ip_roles[:database] = get_elements(database)

        # Add the IPs to the array
        ips = ips + ip_roles[:balancer]
        ips = ips + ip_roles[:server]
        ips = ips + ip_roles[:database]

        ips = ips.uniq

        file.close

        return ips, ip_roles
    end

end


# Obtains the disk images from the img_file file.
def web_yaml_images(path)

    img_roles = {}

    file = File.open(path)
    tree = YAML::parse(file)

    if tree != nil

        tree = tree.transform

        # Classic deployment: load balancer + web servers + database
        balancer = tree[:balancer]
        server   = tree[:server]
        database = tree[:database]

        # Maybe we have been given only an image for all virtual machines
        all      = tree[:all]

        if all == nil
            img_roles[:balancer] = get_elements(balancer)
            img_roles[:server]   = get_elements(server)
            img_roles[:database] = get_elements(database)
        else
```

```
         img_roles[:all]      = get_elements(all)
      end

      file.close

      return img_roles
   end

end


# Transforms the given elements to an array.
def get_elements(array)

   elements = []
   if array != nil
      elements = array.to_a
   end
   return elements

end
```