Universidad Zaragoza

Proyecto Fin de Carrera
Ingeniería en Informática

Diseño e implementación de un sistema de ejecución de trabajos distribuidos

David Ceresuela Palomera

Director: Javier Celaya

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

Curso 2011/2012
Junio 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

Capítulo 1
Introducción

[Revisar]

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 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 modificado añadiéndole la posibilidad de gestionar infraestructuras distribuidas. La modificación realizada se

ha validado usando tres ejemplos de infraestructuras distribuidas, que se explican a continuación.

La primera de ellas es AppScale [2], una implementación *open source* del App Engine de Google [7]. App Engine permite alojar aplicaciones web en la infraestructura que Google posee. Además del alojamiento de aplicaciones web, AppScale también ofrece las APIs [1] 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 [6]. La API de MapReduce permite escribir aplicaciones que hagan uso del *framework* MapReduce. La última API, 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 [2]. 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 infraestructura es una infraestructura de ejecución de trabajos. Este tipo de infraestructuras está especializada 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. Dentro de esta infraestructura distribuida los ejemplos más claros que podemos encontrar son Condor y Torque.

La tercera y última es la de servicios web en tres capas. Este tipo de infraestructura 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] API (del inglés *Application Programming Interface*, Interfaz de programación de aplicaciones) es el conjunto de funciones y procedimientos que ofrece una biblioteca para ser utilizado por otro *software* como una capa de abstracción.

[2] 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. Análisis de las herramientas de administración de virtualización *hardware*.
2. Estudio de algunas de las infraestructuras distribuidas existentes profundizando en la parte relativa a la ejecución de trabajos distribuidos.
3. Investigación de las herramientas de gestión de configuración existentes más relevantes y elección de aquella que mayor facilidad de integración y uso proporcione.
4. Extensión de la herramienta de gestión de configuración para que soporte la puesta en marcha y el mantenimiento de un sistema de ejecución de trabajos distribuidos.

1.3 Trabajos previos

Desde un primer momento se decidió trabajar con la herramienta de 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 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 se eligió desde el inicio, sobre Condor, porque ...

1.4 Tecnología utilizada

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 [10] para la redacción de esta memoria.
- Dia para la elaboración de los diagramas que aparecen en esta memoria.

1.5 Organización de la memoria

El resto de este documento queda organizado de la siguiente manera:

Capítulo 2 Análisis de las herramientas e infraestructuras utilizadas.

Capítulo 3 Extensión de Puppet para gestión de infraestructuras distribuidas.

Capítulo 4 Diseño de recursos distribuidos específicos.

Capítulo 5 Validación de la solución planteada.

Capítulo 6 Conclusiones.

1.6 Agradecimientos

Agradecimientos

Capítulo 2
Análisis de las herramientas e infraestructuras utilizadas

[Revisar]

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

2.1 Análisis de la herramienta de virtualización *hardware*

A la hora de hacer una virtualización *hardware* hay varias opciones entre las que elegir. Las más ampliamente usadas son Xen y KVM. La principal diferencia entre ambas es que Xen ofrece paravirtualización y KVM ofrece virtualización nativa.

La virtualización nativa permite hacer una virtualización *hardware* completa de manera eficiente. No requiere de ninguna modificación en el sistema operativo de la máquina virtual, pero necesita de un procesador con soporte para virtualización. KVM 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 opetativo Debian, se eligió KVM para dar soporte a las máquinas virtuales. Esto significa que podemos usar cualquier sistema operativo para las máquinas virtuales, sin necesidad de hacer ninguna modificación.

2.2 Análisis de las infraestructuras de ejecución de trabajos distribuidos

[Revisar] AppScale es una implementación *open source* 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.

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 de la API de Neptune, que no es tan sencilla. En general esto 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 especificarse, 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 especificar, 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.

[Revisar] La otra infraestructura de ejecución de trabajos distribuidos que se ha elegido ha sido Torque. Torque es una de las infraestructuras clásicas en lo que a ejecución de trabajos se refiere. Una infraestructura Torque está compuesta de un nodo maestro y tantos nodos de computación como se desee. Una vez puesta en marcha la infraestructura, los usuarios que tengan permiso pueden mandar sus trabajos al nodo maestro. El nodo maestro, valiéndose de un planificador, 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.

2.3 Análisis de la herramienta de gestión de configuración

Puppet es una herramienta de gestión de configuración basada en un lenguaje declarativo. A través de este lenguaje se modelan 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).

La agrupación de uno o más recursos en un fichero de texto da lugar a un manifiesto. En general, un manifiesto contiene la información necesaria para realizar la configuración de un nodo. Cuando a Puppet se le da la orden de aplicar un manifiesto los pasos que hace son (Figura 2.1):

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

Normalment 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 especificado en el manifiesto.

Puppet, al igual que otras herramientas de configuración, trata de que los nodos converjan hacia un estado concreto, pero no garantiza que esto ocurra en una única ejecución. Es posible que sean necesarias varias ejecuciones de Puppet, aun cuando todo va bien, para alcanzar el

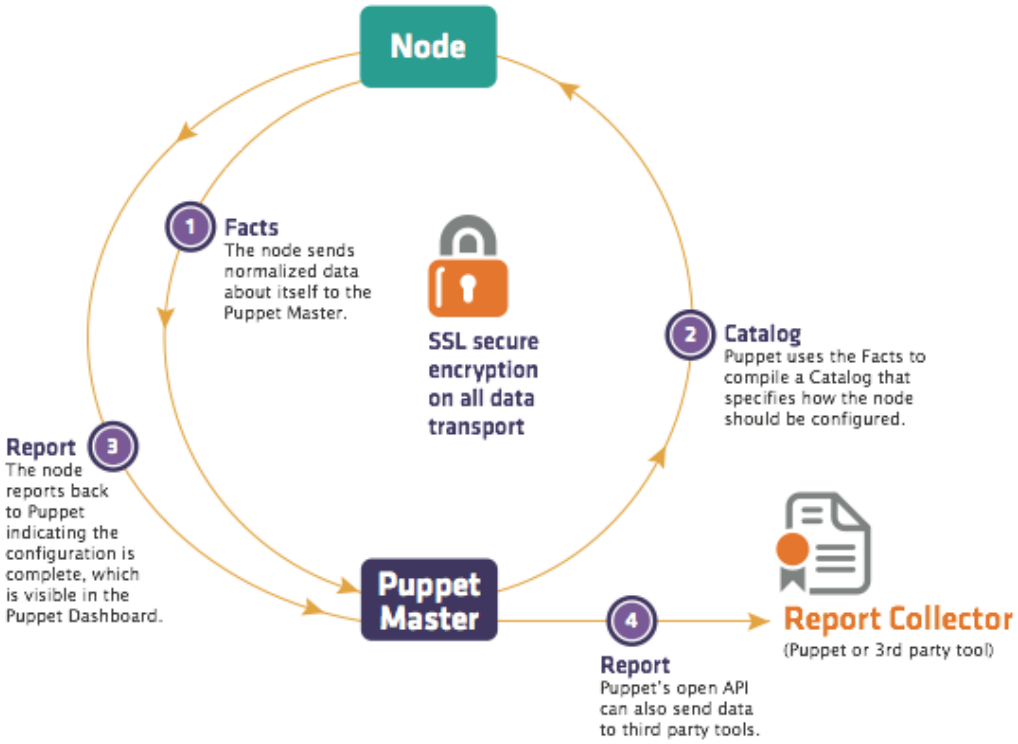

Figura 2.1: Flujo de datos en Puppet.

estado deseado. Aunque esto pueda contrastar con la ejecución habitual de los programas, no es tan excepcional: 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 Puppet no valdría para poner ambos servicios en marcha, sino que harían falta dos iteraciones como mínimo.

Capítulo 3
Modelado de configuración automática de infraestructuras distribuidas

[Revisar]

La extensión a la herramienta de configuración Puppet se ha hecho mediante el uso de tipos y proveedores personalizados. Mediante la definición de un nuevo tipo estamos añadiendo un nuevo recurso que Puppet puede administrar; pero para que Puppet sepa cómo administrar ese nuevo recurso debemos proporcionar un proveedor en el que se le indique lo que tiene que hacer.

3.1 Modelado de recursos y configuración automática en Puppet

3.2 Modelado de recursos distribuidos: el recurso *cloud*

El modelado de un recurso distribuido plantea ciertos desafíos al modelo anterior: se puede pensar que para modelar un recurso distribuido basta con que Puppet envíe a cada nodo la configuración necesaria para garantizar el comportamiento deseado, pero, ¿qué pasa cuando ese nodo falla? Si no hacemos nada, el recurso dejará de mantenerse en el estado deseado. Por lo tanto, a la hora de administrar un recurso distribuido hay que asegurarse de que los nodos están operativos y cumpliendo con su función. Asimismo, un recurso distribuido puede presentar elementos comunes con otros recursos distribuidos, tales como una monitorización básica. Entre los recursos de Puppet, por ejemplo un usuario y un paquete, no hay tantos elementos comunes.

Tenemos, pues, dos tipos de recursos claramente diferenciados: los recursos clásicos de Puppet (a los que llamaremos recursos locales) y los recursos distribuidos. Para entender mejor el modelado de estos recursos se puede hacer una analogía con la programación orientada a objetos (Figura 3.1). En esta analogía el recurso distribuido sería una instancia de una metaclase que proporcionaría los atributos y métodos comunes a todo recurso distribuido. La instanciación de ese recurso en un recurso distribuido de tipo AppScale, Torque, web u otros sería similar a la instanciación de una clase a partir de la metaclase. De igual manera, el recurso distribuido instanciado podría añadir nuevos atributos y métodos y modificar el comportamiento de los métodos de la metaclase, ya que no tienen por qué iniciarse de igual manera el recurso AppScale y el recurso Torque. Los recursos locales de Puppet quedarían como clases que se instanciarían a partir de una metaclase de recursos locales.

Para modelar un recurso *cloud*, se han considerado como fundamentales los siguientes parámetros:

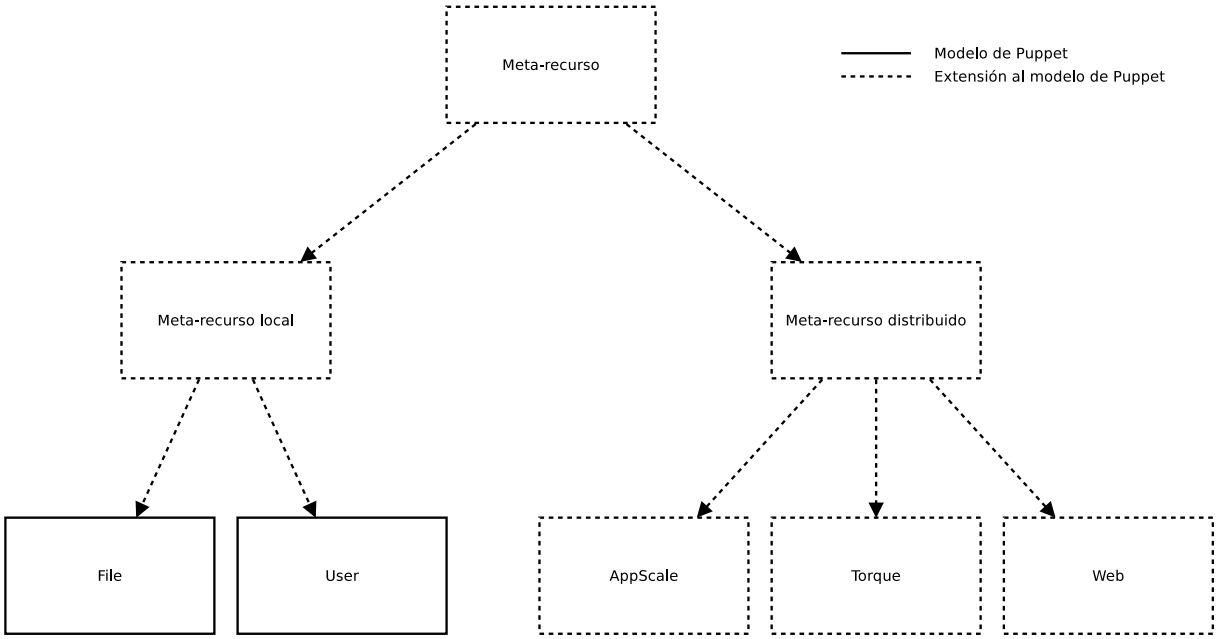

Figura 3.1: Modelado teórico de recursos en Puppet.

- Nombre: Para identificar al *cloud* de manera única.
- Fichero de direcciones IP: Para describir qué dirección IP está asociada a cada nodo del *cloud*. Como la asociación viene dada por un par <rol, dirección IP>también se le llama a este parámetro fichero de roles.
- Fichero de imágenes de disco: Para asignar a cada nodo la correspondiente imagen de disco duro.
- Fichero de dominio: Para definir una máquina virtual especificando sus características *hardware*.
- Conjunto de máquinas físicas: Para indicar qué máquinas físicas pueden ejecutar las máquinas virtuales definidas.

Para modelar un recurso *cloud* de tipo Torque podríamos usar algo similar a:

```
torque {'torque-cloud':
   ip_file  => "/etc/puppet/modules/torque/files/jobs-ip.yaml",
   img_file => "/etc/puppet/modules/torque/files/jobs-img.yaml",
   domain   => "/etc/puppet/modules/torque/files/mycloud-template.xml",
   pool     => ["155.210.155.70"],
   ensure   => running,
}
```

3.3 Diseño del proveedor de recursos distribuidos

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".

En un recurso cloud la definición en el fichero tipo contendría los parámetros propios de ese tipo de cloud. En el ejemplo anterior éstos serían: `ip_file`, `img_file`, `domain` y `pool` (el parámetro `ensure` es común a todo recurso puppet y se define automáticamente). Una vez modelado el recurso *cloud*, queda como tarea proporcionar un proveedor que se encargue de llevar el *cloud* al estado que se le indique desde el manifiesto Puppet. Un *cloud* podrá estar en dos estados: funcionando o parado. Si tiene que estar funcionando, el proveedor se encargará de todos los pasos necesarios para llevar al *cloud* a ese estado, que a grandes rasgos son:

- Comprobación de la existencia del *cloud*: si existe se realizarán tareas de mantenimiento, si no existe se creará.
- Comprobación del estado del conjunto de máquinas físicas.
- Obtención de las direcciones IP de los nodos y los roles que les han sido asignados.
- 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.
- Cuando todas las máquinas virtuales estén funcionando se procede a inicializar el *cloud*.
- Operaciones de puesta en marcha particulares a cada tipo de *cloud*.

De la misma manera, si decidimos que el *cloud* tiene que estar parado, el proveedor se encargará de realizar los pasos necesarios para ello. Estos pasos son:

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

Una parte muy importante del proveedor es el parámetro `ip_file`. En este fichero se define una asociación entre la dirección IP del nodo y el rol que cumplirá dentro del *cloud*. Siguiendo con el ejemplo del recurso *cloud* de tipo Torque, el fichero de roles en el que se especifica quién es el nodo maestro y la lista de nodos de computación sería similar a este:

```
---
:head: 155.210.155.73
:compute:
- 155.210.155.177
```

3.4 Implementación del proveedor [Temporal]

La primera opción que se barajó fue la de usar *Faces* de Puppet. *Faces* es una API para crear subcomandos y acciones dentro de Puppet. Analizada a fondo, esta API no proporcionaba una ventaja muy superior a la ejecución de comandos desde la consola del sistema operativo, y no interesaba crear una abstracción que facilitara el trabajo para posteriormente estar usando continuamente la línea de comandos.

La segunda opción que se barajó fue la de la creación de un tipo y un proveedor para ese tipo. Esta opción sí que presenta una ventaja considerable: podemos usar el tipo para modelar la infraestructura distribuida y podemos usar el proveedor para indicar cómo iniciar y mantener esa infraestructura. Esta aproximación se acerca más al modelo que usa Puppet, ya que definimos la infraestructura como si fuera un recurso más de los que posee Puppet. Así pues, esta es la aproximación que se tomó para realizar la extensión.

Capítulo 4
Diseño de recursos distribuidos específicos

[Revisar]

4.1 Diseño y uso 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.

4.1.1 Manifiesto de recurso distribuido

La sintaxis del manifiesto distribuido no se verá afectada por los dos tipos de despliegue posibles, pero sí que tendrá que reflejar los parámetros necesarios para realizar las tareas de administración de la infraestructura. Éste podría ser un ejemplo de un manifiesto para una infraestructura de tipo AppScale:

```
appscale {'mycloud':
   ip_file      => "/etc/puppet/modules/appscale/files/appscale-ip.yaml
      ",
   img_file     => "/etc/puppet/modules/appscale/files/appscale-img.yaml
      ",
   domain       => "/etc/puppet/modules/appscale/files/mycloud-template.
      xml",
   pool         => ["155.210.155.70"],
   app_email    => "user@mail.com"
   app_password => "password"
   ensure       => running,
```

```
}
```

4.1.2 Fichero de roles

El fichero de roles sí que debe reflejar los dos posibles tipos de despliegue. 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.

Un fichero de roles para este despliegue sería de esta forma:

```
---
:controller: 155.210.155.73
:servers:
- 155.210.155.177
- 155.210.155.178
```

Por otra parte, 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.

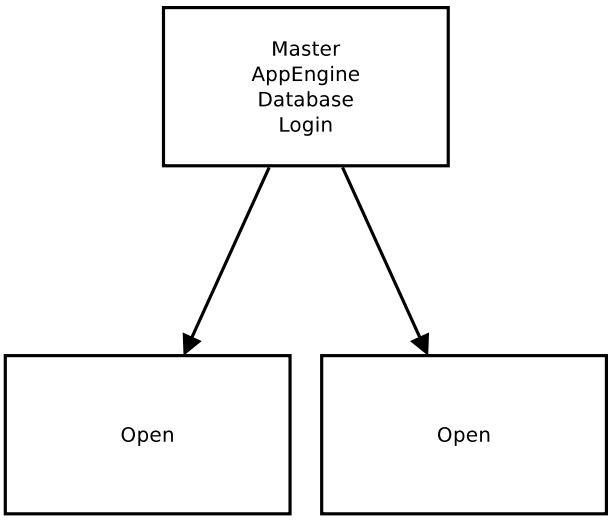

Figura 4.1: Infraestructura AppScale en despliegue personalizado.

Hay multitud de despliegues posibles combinando estos roles, pero será de especial interés para nosotros el que permite ejecutar trabajos de computación en AppScale (Figura 4.1). Un despliegue de este tipo podría conseguirse con un fichero similar a éste:

```
---
:master:    155.210.155.73
:appengine: 155.210.155.73
:database:  155.210.155.73
:login:     155.210.155.73
:open:
- 155.210.155.177
- 155.210.155.178
```

4.2 Diseño y uso 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 4.2). 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.

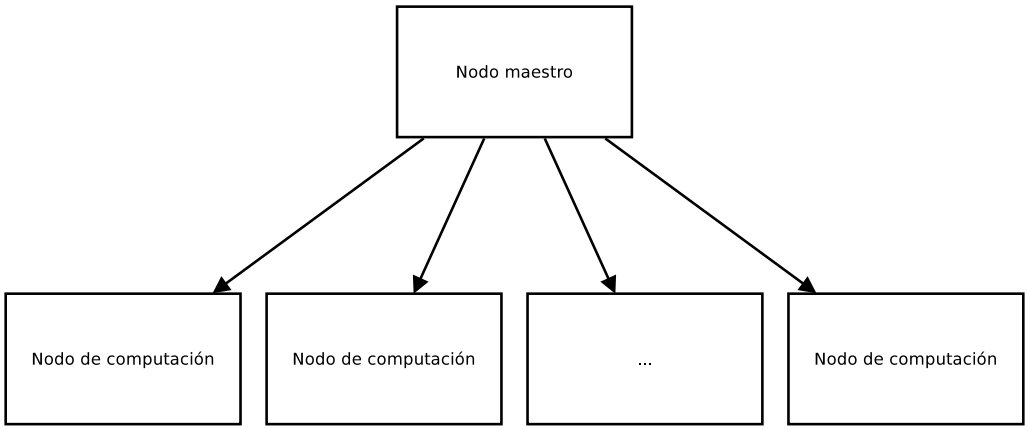

Figura 4.2: Infraestructura Torque.

4.2.1 Manifiesto de recurso distribuido

La sintaxis del manifiesto distribuido es similar a la usada en el ejemplo de AppScale, sólo que aquí no aparecen los parámetros de administración que aparecían en aquél, ya que Torque no requiere su uso:

```
torque {'mycloud':
   ip_file  => "/etc/puppet/modules/torque/files/jobs-ip.yaml",
   img_file => "/etc/puppet/modules/torque/files/jobs-img.yaml",
   domain   => "/etc/puppet/modules/torque/files/mycloud-template.xml",
   pool     => ["155.210.155.70"],
   ensure   => running,
}
```

4.2.2 Fichero de roles

El contenido del fichero de roles sí que será más sencillo que en el caso de AppScale, ya que en Torque tenemos únicamente los roles de nodo maestro y nodo de computación. La especificación completa de la sintaxis es la siguiente:

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

`compute` : La lista de máquinas que desempeñarán el rol de nodos de computación.

Un fichero de especificación de roles para una infraestructura Torque tendría un contenido similar a éste:

```
---
:head: 155.210.155.73
:compute:
- 155.210.155.177
- 155.210.155.178
```

4.3 Diseño y uso de un recurso distribuido para una infraestructura web de tres niveles

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.

Para demostrar la validez del modelo desarrollado se verá como, además de sobre las infraestructuras AppScale y Torque, también se puede aplicar dicho modelo sobre una infraestructura que no tiene nada que ver con la ejecución de trabajos: una infraestructura de servicios web. En el ejemplo se ha validado una infraestructura que consta de un balanceador de carga, dos servidores web y un servidor de bases de datos (Figura 4.3).

4.3.1 Manifiesto de recurso distribuido

La sintaxis en el manifiesto es fundamentalmente similar a la utilizada en el ejemplo de Torque, ya que tampoco en este caso tenemos parámetros de administración de la infraestructura:

```
web {'mycloud':
    ip_file  => "/etc/puppet/modules/web/files/web-ip.yaml",
    img_file => "/etc/puppet/modules/web/files/web-img.yaml",
    domain   => "/etc/puppet/modules/web/files/mycloud-template.xml",
    pool     => ["155.210.155.70"],
    ensure   => running,
}
```

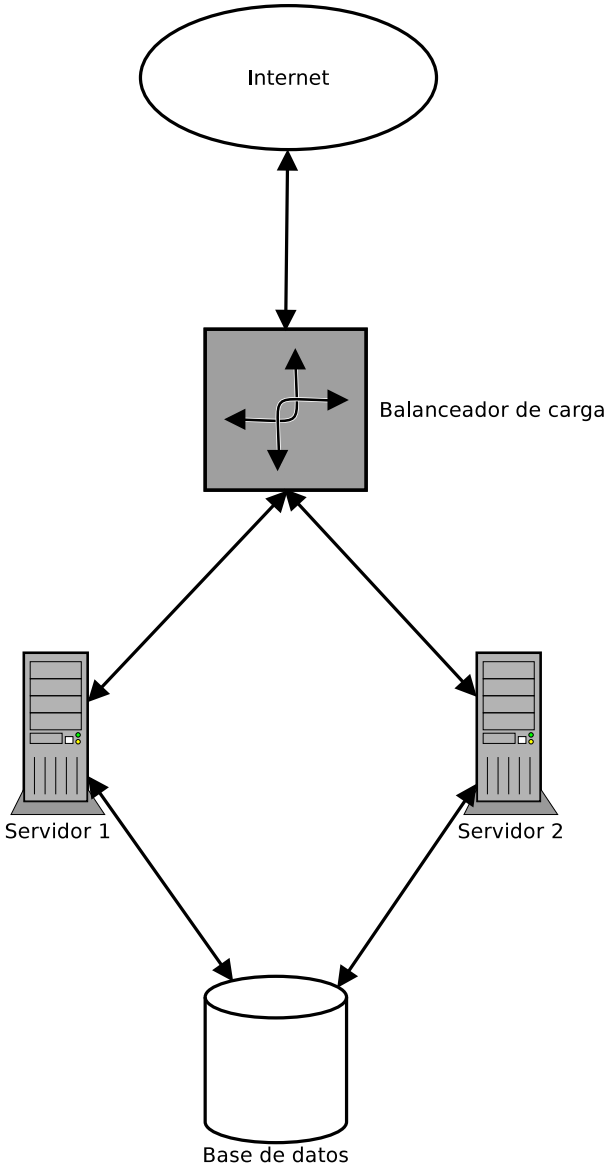

Figura 4.3: Infraestructura web de tres niveles.

4.3.2 Fichero de roles

El contenido del fichero de especificación de roles sí que poseerá valores distintos a los que tenía cualquiera de los dos ejemplos anteriores ya que estamos describiendo una infraestructura distinta. Los roles que pueden desempeñar los nodos dentro de una infraestructura web son:

`balancer` : La máquina que desempeñará el rol de balanceador de carga.

`server` : La lista de máquinas que desempeñarán el rol de servidor web.

`database` : La máquina que desempeñará el rol de servidor de base de datos.

Un ejemplo completo del fichero de especificación de roles tendría un contenido similar a éste:

```
---
:balancer: 155.210.155.175
:server:
- 155.210.155.73
- 155.210.155.178
:database: 155.210.155.177
```

Capítulo 5
Validación de la solución planteada

x Máquinas físicas
y Máquinas virtuales
AppScale: 1 maestro, dos esclavos

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 (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.

5.1 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.
- Apagado de una máquina virtual que no había empezado encendida y no era líder.
- Apagado de una máquina virtual que había empezado encendida y era líder.
- Puesta en marcha de la infraestructura desde una máquina que no pertenece al *cloud*.
- Puesta en marcha de la infraestructura desde una máquina que pertenece al *cloud*.

5.2 Prueba de infraestructura AppScale

5.3 Prueba de infraestructura Torque

Para probar la infraestructura Torque se han usado cuatro (de momento) máquinas virtuales alojadas en X máquinas físicas. Una de las máquinas virtuales actúa como nodo maestro y las

otras tres (de momento) 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.
- Parada del proceso servidor (pbs_server) en el nodo maestro.
- Parada del proceso planificador (pbs_sched) en el nodo maestro.
- Parada del proceso de ejecución de trabajos (pbs_mom) en un nodo de computación.
- Parada del proceso que monitoriza al proceso de autenticación en el nodo maestro.
- Parada del proceso que monitoriza al proceso servidor en el nodo maestro.
- Parada del proceso que monitoriza al proceso planificador en el nodo maestro.
- Parada del proceso que monitoriza al proceso de ejecución de trabajos en un nodo de computación.

5.4 Prueba de infraestructura web de tres niveles

Para probar la infraestructura web se han usado cuatro máquinas virtuales repartidas entre X 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.
- Parada del proceso servidor web.
- Parada del proceso base de datos.
- Parada del proceso que monitoriza al proceso balanceador de carga.
- Parada del proceso que monitoriza al proceso servidor web.
- Parada del proceso que monitoriza al base de datos.

Capítulo 6
Conclusiones

Conclusiones.

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: Elastic Compute Cloud. `http://aws.amazon.com/ec2/`, 2012.

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

[8] 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.

[9] 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.

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

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

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

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