Universitat Politècnica de València

Departamento de Sistemas Informáticos y Computación

Tesis Doctoral

Gestión de infraestructuras virtuales configuradas dinámicamente

---

Doctorando:
Miguel Caballer Fernández

Directores:
Dr. D. Ignacio Blanquer Espert
Dr. D. Germán Moltó Martínez

Abril 2014

Agradecimientos

Primero quiero agradecer a “Mariajo” los ánimos para acabar esta tesis y conseguir acabar el doctorado.

Agradezco a toda mi familia el amor recibido a lo largo de toda una vida, hasta llegar a este momento. En especial quiero acordarme de mi tío José Vicente, que nos dejó este verano y que me acuerdo mucho de él.

Quiero agradecer a mis directores de tesis, tanto Nacho como Germán por el apoyo en el trabajo de esta tesis así como todos los trabajos que hemos hecho juntos en el grupo.

También quiero destacar a Carlos de Alfonso y a Fernando Alvarruiz que también han apoyado en muchas de las ideas que iniciaron esta tesis.

A Vicente Hernández como director de GRyCAP por darme la oportunidad de entrar en el grupo y trabajar en él durante estos últimos 12 años.

A Vicente Gimenez por su ayuda en el diseño de la interfaz web del Infrastructure Manager.

A todos los compañeros del GRyCAP, tanto a los actuales como los que ya se han ido, por hacer de éste un gran grupo de trabajo y amigos con los que da gusto trabajar.

Gestión de infraestructuras virtuales configuradas dinámicamente

En los últimos años y con el auge las tecnologías de virtualización y de las infraestructuras Cloud, se abre un nuevo abanico de posibilidades para el acceso a recursos de cómputo para el ámbito científico. Las tecnologías Cloud permiten "*acceso ubicuo, adaptado y bajo demanda en red a un conjunto compartido de recursos de computación"*. Estas tecnologías permiten que el acceso a grandes cantidades de recursos virtualizados sea mucho más sencillo para el científico.

A pesar de que existen en la actualidad diferentes proveedores Cloud, diferente software para el despliegue de plataformas Cloud, diferentes gestores de máquinas virtuales, y otros componentes, resulta todavía complejo para los usuarios el acceso a los recursos. Además, esta variedad dificulta la interoperabilidad. Por tanto el objetivo principal de esta tesis es avanzar en el estado del arte en el acceso a infraestructuras de cómputo personalizadas y configuradas de forma dinámica, con una orientación principal hacia la comunidad científica. Este trabajo se concretará en una plataforma que permita un despliegue y gestión de infraestructuras Cloud sencillo, de forma que los investigadores solo tengan que centrarse en las tareas propias de sus aplicaciones.

Una plataforma Cloud para investigación debe contemplar todos los aspectos necesarios para la creación y gestión de las infraestructuras ad-hoc, partiendo de que el investigador debe poder expresar sus requerimientos, tanto del hardware virtual como del software, sobre los recursos que va a necesitar para la ejecución de su aplicación. En base a los requerimientos definidos por el usuario, el sistema debe crear la infraestructura del usuario, teniendo en cuenta aspectos como la selección de proveedores Cloud y de imágenes de máquinas virtuales, así como de los procesos de configuración. El sistema también debe permitir que el usuario defina reglas que permitan adaptar dinámicamente el número de recursos (o unidades) de cómputo (elasticidad horizontal) así como las características de los mismos (elasticidad vertical) a la carga efectiva del sistema. Por último la plataforma debe proporcionar interfaces tanto a nivel de usuario, mediante aplicaciones de línea de comandos o interfaces gráficas, como a nivel programático para que capas de mayor nivel puedan hacer uso de la funcionalidad mediante un API. La tesis pretende tanto avanzar en las especificaciones y arquitecturas software cómo desarrollar y testear un prototipo.

Gestió de infraestructures virtuals configurades dinàmicament

En els últims anys i amb l'auge les tecnologies de virtualització i de les infraestructures Cloud, s'obrin noves possibilitats per a l'accés de recursos de còmput per a l'àmbit científic. Les tecnologies Cloud permeten "*accés ubic, adaptat i baix demanda en xarxa a un conjunt compartit de recursos de computació*". Aquestes tecnologies permeten que l'accés a grans quantitats de recursos virtualitats siga molt més senzill per al científic.

A pesar de que en l'actualitat hi ha diferents proveïdors Cloud, diferent software per al desplegament de plataformes Cloud, diferents gestors de màquines virtuals, i altres components, encara es molt complex per als usuaris el accés al recursos. A mes esta varietat dificulta la interoperabilitat. Per tant l'objectiu principal de l'esta tesi és avançar en l'estat de l'art en l'accés a infraestructures de còmput personalitzades i configurades de forma dinàmica, amb una orientació principal cap a la comunitat científica. Este treball es concretarà en una plataforma que permeta un desplegament i gestió d'infraestructures Cloud senzill, de manera que els investigadors només hagen de centrar-se en les tasques pròpies de les seues aplicacions

Una plataforma Cloud per a investigació ha de contemplar tots els aspectes necessaris per a la creació i gestió de les infraestructures ad-hoc, començant per que l'investigador ha de poder expressar els seus requeriments, tant del maquinari virtual com del programari, sobre els recursos que necessitarà per a l'execució de la seua aplicació. Basant-se en els requeriments definits per l'usuari, el sistema ha de crear la infraestructura de l'usuari, tenint en compte aspectes com la selecció de proveïdors Cloud i d'imatges de màquines virtuals, així com dels processos de configuració. El sistema també ha de permetre que l'usuari definisca regles que permeten adaptar dinàmicament la quantitat de recursos (o unitats) de còmput (elasticitat horitzontal) així com les característiques dels mateixos (elasticitat vertical) a la càrrega efectiva del sistema. Finalment la plataforma ha de proporcionar interfícies tant a nivell d'usuari, per mitjà d'aplicacions de línia de comandos o interfícies gràfiques, com a nivell programàtic perquè capes de major nivell puguen fer ús de la funcionalitat per mitjà d'un API. La tesi pretén tant d'avançar en les especificacions i arquitectures programari com desenrotllar i testejar un prototip

Dynamic management of virtual infrastructures

In the last years with the advent of virtualization technologies and Cloud infrastructures, new possibilities are offered in the scientific area to access computational resources. This technologies offer: "*ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources*". These technologies enable the scientists to access a great number of virtual resources.

Despite that, currently there are many different Cloud providers, different Cloud software stacks to deploy Cloud platforms, different virtual machine managers, and other components, it is still complex for the users to access the resources. Moreover this diversity complicates the interoperability. So the main objective of this thesis is to move forward in the state of art in the dynamic management of computing infrastructures, with a main focus in the scientific community. This work will provide a platform to deploy and manage Cloud infrastructures easily, so the scientists can focus in their own application issues.

A Cloud platform for investigation must consider all the necessary aspects to the creation and management of the infrastructures. The first step is the specification of the user requirements, both hardware and software, about the resources needed to execute his application. Based on these requirements the system must create the user infrastructure, considering aspects as the Cloud deployment and virtual machine image selection, the contextualization process, etc. The system must enable the user to dynamically modify both the number of computational resources (or units) (horizontal elasticity) and their features (vertical elasticity). Finally the platform must provide interfaces both to user level with command line applications or graphical ones and at programmatic level, enabling upper level layers to access the functionality using an API. The thesis expects to move forward in the specification of software architectures and also deploying and testing a prototype.

Contenidos

1 Introducción 1
1.1 Motivación 2
2 Estado del arte 5
2.1 Cloud computing 5
2.2 Proveedores Cloud 7
2.3 Cloud Management Platforms (CMPs) 11
2.4 APIs de Conexión IaaS 13
2.4.1 Estándares 14
2.4.2 APIs de agregación 15
2.5 Lenguajes de especificación de requerimientos Cloud 17
2.6 Herramientas de contextualización 18
2.7 Catálogos de imágenes de máquinas virtuales 21
2.8 Trabajos Relacionados 23
3 Objetivos 31
4 Arquitectura 35
4.1 El Lenguaje de Descripción de Infraestructuras RADL 36
4.1.1 Forma general de un documento RADL 37
4.1.2 Características a considerar en una infraestructura de ejecución 39
4.1.3 Ejemplos 47
4.2 Infrastructure Manager 49
4.2.1 API 50
4.2.2 Cloud Selector 53
4.2.3 Cloud Connector 56
4.2.4 Configuration Manager 60
4.2.5 Gestión de la elasticidad 64
4.2.6 Recetas incluidas en el IM 67
4.3 VMRC 69

4.3.1 Capacidades y Requerimientos 70
4.3.2 Convención de URIs 71
4.4 Diagrama de interacción 71
4.5 Herramientas de Cliente 72
4.5.1 Cliente de línea de comandos 72
4.5.2 Interfaz web 74
5 Casos de Uso 79
5.1 Conjunto de imágenes base creadas/usadas 79
5.2 Proveedores Cloud utilizados 79
5.3 Lanzamiento de infraestructuras científicas 80
5.3.1 Clúster Hadoop 81
5.3.2 Clúster EMI 84
5.3.3 Análisis de mutaciones 88
5.3.4 XWCH en Windows 92
5.4 Servicios de alto nivel 96
5.4.1 CodeCloud 96
5.4.2 EC3: Elastic Cloud Computing Cluster 103
5.4.3 TRENCADIS 104
5.4.4 Plataforma docente ODISEA 106
6 Conclusiones y Trabajos Futuros 109
6.1 Resumen y Contribuciones Principales 109
6.2 Trabajos Futuros 111
7 Proyectos de investigación asociados 113
7.1 CodeCloud 113
7.1.1 Aportaciones 114
7.2 GE3CEAC 114
7.2.1 Aportaciones 115
8 Publicaciones 117
Anexo I 119
Anexo II 129

API XML-RPC 129
API REST 132
Referencias 135
Acrónimos 141

1 Introducción

En todos los ámbitos científicos es básico el acceso a recursos de cómputo para la resolución de problemas de cálculo intensivo. Dentro del cálculo intensivo se suelen diferenciar entre los problemas de computación de altas prestaciones (High Performance Computing – HPC) donde se requiere resolver problemas de gran dimensión, que requieren del uso de múltiples procesadores de forma colaborativa para resolverlo de forma más rápida, y los de computación masiva (High Troughput Computing – HTC) donde se suele requerir procesar gran cantidad de trabajos de dimensión "relativamente" pequeña y que son independientes entre sí.

Inicialmente se necesitaba el acceso a recursos de computación muy especializados usando grandes supercomputadores. Este tipo de infraestructuras están diseñadas para la ejecución de aplicaciones HPC y aunque permite la ejecución de problemas de tipo HTC, suele producirse un bajo aprovechamiento de los recursos del supercomputador (red de altas prestaciones, memoria, etc.). Más adelante se extendió el uso de clúster de PCs mucho más asequibles para los centros de investigación, permitiendo que los centros de investigación pudieran tener sus propios recursos locales. Este tipo de plataformas, más orientadas a los problemas de tipo HTC, permitían también la ejecución de aplicaciones HPC con buenas prestaciones. Gracias a los incrementos en los anchos de banda de las redes de comunicación, producidos en los últimos años, se impulsó la idea de unir diferentes recursos computacionales, de distintas organizaciones, para crear una infraestructura global de computación distribuida conocida como el Grid [1],[2]. Esto ha permitido que los científicos puedan acceder a una importante cantidad de recursos, pero dicho acceso no es sencillo. Aunque las tecnologías Grid se basan en el uso de estándares para proporcionar un acceso homogéneo y ubicuo a los recursos, no pueden esconder la propia heterogeneidad de los recursos, tanto a nivel de hardware como a nivel de software. Esto provoca que la migración de las aplicaciones científicas al Grid no sea sencilla para muchas de ellas, necesitando en la mayoría de los casos un gran apoyo por otros grupos del ámbito de la informática.

En los últimos años, con el auge de las tecnologías de virtualización y de las infraestructuras Cloud [3], se abre un nuevo abanico de posibilidades para el acceso a recursos de cómputo para el ámbito científico. Estas tecnologías permiten "*acceso ubicuo, adaptado y bajo demanda en red a un conjunto compartido de recursos de computación"*. Una de las grandes ventajas que proporcionan las tecnologías Cloud a los científicos, es que la infraestructura no les impone restricciones en cuanto a hardware ni a software, ya que usando técnicas de virtualización permite que, a diferencia de lo que ocurre en el Grid, sea la infraestructura que se adapte a la aplicación y no al contrario. Este cambio permite que la migración de las aplicaciones científicas a entornos de cálculo distribuido sea mucho más sencilla

para el científico en el caso de las tecnologías Cloud. Además otra de las características fundamentales de estas tecnologías es que permite la adaptación elástica y dinámica del tamaño de la infraestructura, permitiendo adaptarse a las necesidades de las aplicaciones en cada momento. La elasticidad evita la creación a priori de una gran infraestructura, evitando sobredimensionar las infraestructuras para procesar picos de carga temporales.

No obstante, en la actualidad existen diferentes proveedores Cloud, diferente software para el despliegue de plataformas Cloud que complican el acceso de forma sencilla y homogénea. En esta tesis se propone un nuevo modelo de especificación, despliegue y monitorización de infraestructuras que se demuestra sobre una serie de componentes que facilitan el acceso a los diferentes sistemas Cloud actuales. Estos componentes facilitan además la reconfiguración elástica de las infraestructuras virtuales para automatizar todos los procesos necesarios y que el científico solo tenga que preocuparse de ejecutar sus aplicaciones.

1.1 Motivación

Como se ha comentado en la introducción, el auge las tecnologías de virtualización y de las infraestructuras Cloud permite el acceso a grandes cantidades recursos de cómputo. Estas tecnologías ya están siendo utilizadas a nivel empresarial, pero aún resulta escasa su utilización en muchas áreas del ámbito científico, que podrían obtener grandes beneficios.

En la actualidad existen un gran número de proveedores Cloud así como distintas herramientas de gestión de plataformas Cloud (Cloud Management Platforms – CMPs). Esto es una ventaja a la hora de elegir diferentes proveedores y para acceder a gran cantidad de recursos disponibles, pero tiene un gran inconveniente provocado por el uso de diferentes sistemas de acceso no interoperables. Por tanto se deberán analizar todas estas plataformas y paquetes software para definir un conjunto de funcionalidades básicas necesarias y métodos homogéneos de acceder a dichas funcionalidades.

Se deben estudiar las necesidades de un investigador a la hora de crear una infraestructura virtual para poder ejecutar sus aplicaciones. Por tanto se deberán analizar todas las tecnologías existentes que puedan proporcionar las capacidades que las plataformas Cloud básicas no proporcionen y que permitan contemplar todas las funcionalidades que el investigador va a necesitar.

Una necesidad básica a la hora de poder acceder a los recursos en entornos Cloud es la de ser capaz de expresar los requerimientos sobre la infraestructura virtual que se va a necesitar para la ejecución de una aplicación científica. Y debe poder hacerlo usando un lenguaje sencillo que permita que usuarios no avanzados en informática puedan utilizarlo en el despliegue de sus infraestructuras, así como a los más avanzados expresar con detalle sus necesidades.

En el caso de las máquinas virtuales, al igual que en las máquinas físicas, es necesario tener instalado un S.O. y un conjunto de aplicaciones para poder su completo funcionamiento. Este proceso de instalación no es del todo sencillo y además requiere de bastante tiempo para su finalización. Otra de las ventajas de la virtualización es que permite la reutilización de imágenes de disco de otras máquinas virtuales de manera que no es necesario realizar el paso de instalación del sistema. Una imagen de máquina virtual es la encapsulación de un S.O., unas aplicaciones y una determinada configuración. Dichas imágenes pueden ser instanciadas sobre un hardware concreto, con la ayuda de un hipervisor, para ejecutar una máquina virtual. Por tanto es necesario disponer de algún sistema de catalogación que permita almacenar e indexar dichas imágenes permitiendo a otros usuarios la reutilización de las mismas. Dicho sistema debe almacenar toda la información relevante sobre las imágenes y su exposición usando algún tipo de API mediante protocolos estándar, de tal manera que permita su procesado de forma automatizada.

Aunque la catalogación de imágenes permite elegir y utilizar aquella más adecuada a las necesidades del usuario, no siempre se adapta perfectamente a las necesidades del usuario de manera que es necesario completarla con todos aquellos requisitos que haya expresado y que no sean cumplidos por la imagen base. A este proceso se le denomina contextualización, y que la plataforma también deberá tener en cuenta para poder completar la creación de la infraestructura solicitada por el usuario.

Por tanto en la sección del estado del arte se analizarán todas las tecnologías y aplicaciones existentes que contemplen todos estos aspectos, para más adelante definir unos objetivos claros en función de las características que aportan los sistemas actuales.

2 Estado del arte

2.1 Cloud computing

Una de las definiciones más extendidas sobre que es el Cloud es la especificada por el National Institute of Standards and Technology (NIST) en [3]:

> *"Cloud Computing es un modelo tecnológico que permite el acceso ubicuo, adaptado y bajo demanda en red a un conjunto compartido de recursos de computación configurables compartidos (por ejemplo: redes, servidores, equipos de almacenamiento, aplicaciones y servicios), que pueden ser rápidamente aprovisionados y liberados con un esfuerzo de gestión reducido o interacción mínima con el proveedor del servicio."*

El NIST también define las características básicas que debe proporcionar, así como los modelos de servicio y despliegue posibles.

Dentro de las características esenciales están:

- Auto-servicio bajo demanda: Un consumidor puede aprovisionar de manera unilateral capacidades de cómputo, almacenamiento en red, etc., en la medida en que las requiera y con la mínima interacción humana por parte del proveedor del servicio.
- Acceso desde la red: Las capacidades están disponibles en la red y son accesibles a través de mecanismos estándar que promueven el uso desde plataformas clientes heterogéneas, como un PC o un dispositivo móvil.
- Conjunto de recursos: Los recursos computacionales del proveedor se habilitan para servir a múltiples consumidores mediante un modelo "multi-tenant", con varios recursos tanto físicos como virtuales asignados y reasignados de acuerdo con los requerimientos de los consumidores. Existe un sentido de independencia de ubicación en cuanto a que el consumidor no posee control o conocimiento sobre la ubicación exacta de los recursos que se le están proveyendo aunque puede estar en capacidad de especificar ubicación a un nivel de abstracción alto; por ejemplo, país, estado o centro de datos.
- Elasticidad rápida: Las capacidades pueden ser rápidamente y elásticamente aprovisionadas, en algunos casos automáticamente, para escalar rápidamente añadiendo recursos y poder liberar los recursos también de manera veloz.
- Servicios de monitorización: Los sistemas Cloud controlan automáticamente y optimizan el uso de recursos mediante una capacidad de monitorizac-

ión a cierto nivel de abstracción adecuado al tipo de servicio; por ejemplo, almacenamiento, procesamiento, ancho de banda y cuentas de usuario activas. El uso de estos recursos puede ser monitorizado, controlado y reportado, proporcionando transparencia tanto para el proveedor como para el consumidor por el servicio utilizado.

Dentro de los modelos de servicio tenemos:

- SaaS – Software as a Service: Mediante este modelo el consumidor utiliza las aplicaciones del proveedor que están alojadas en una infraestructura Cloud. El consumidor no administra ni controla la infraestructura, virtual ni física, que soporta estos servicios, pero sí algunos parámetros de configuración.
- PaaS – Platform as a Service: En este modelo, el consumidor puede desplegar en la infraestructura del proveedor aplicaciones creadas por el primero, incluso adquiridas, usando lenguajes de programación y herramientas del proveedor. De nuevo, el consumidor no controla la infraestructura, virtual ni física, que soporta estos servicios, pero controla las aplicaciones o servicios desplegados y algunas variables del entorno de las aplicaciones.
- IaaS – Infrastructure as a Service: Este modelo permite al consumidor aprovisionar recursos computacionales como almacenamiento, procesamiento, redes y otros elementos fundamentales en donde el consumidor puede desplegar y correr software arbitrario, que puede incluir sistemas operativos y aplicaciones. El consumidor tiene control sobre la infraestructura virtual pero no (o es muy limitado) sobre la infraestructura física, que sigue siendo competencia del proveedor de servicio.

Por último, los modelos de despliegue son:

- Cloud privado (también conocido como *on-premise*): La infraestructura de este tipo de Cloud es operada únicamente para una organización. Puede ser administrada por la organización, por un tercero o una combinación de ambas.
- Cloud comunitario: La infraestructura de este tipo de Cloud es compartida por varias organizaciones dentro de una misma comunidad donde se comparten ciertos asuntos (seguridad, investigación, políticas, etc.). Puede ser administrada por una o varias de las organizaciones de la comunidad, por un tercero, o una combinación de ambas.
- Cloud público: La infraestructura de este tipo de Cloud está disponible para el público en general. Puede ser administrada por una organización empresarial, académica, o gubernamental o una combinación de varias.

- Cloud híbrido: Es la composición de dos o más nubes, por ejemplo privada y pública, que mantienen su entidad propia pero que coexisten por tener tecnología que permite compartir datos o aplicaciones entre las mismas.

En otros artículos se comentan otra serie de ventajas de las tecnologías Cloud [4]. Desde el punto de vista del consumidor de infraestructuras Cloud hay tres aspectos nuevos que proporciona el Cloud:

- La ilusión de un conjunto de recursos infinito disponibles bajo demanda,permitiendo eliminar la necesidad de crear planes para el crecimiento de los recursos en el futuro.
- La eliminación del gasto inicial,permitiendo a las empresas el acceso a los recursos con un gasto inicial bajo pudiendo crecer a medida que crecen sus necesidades.
- La posibilidad de usar un modelo de pago por uso, permitiendo solo pagar por aquellos componentes que realmente se están utilizando y liberar aquellos no que se usan evitando costes.

2.2 Proveedores Cloud

En la actualidad existen numerosos proveedores de servicios Cloud dentro de cada uno de los modelos descritos con anterioridad. Existen muchos estudios realizados donde se comentan las especificaciones de los proveedores actuales: [5], [6] y [7]. A continuación vamos a mostrar un resumen de los proveedores con mayor auge en la actualidad, centrándonos en las de tipo IaaS dado que en el ámbito de la computación científica son las que mayor potencial y flexibilidad tienen, comentando sus características básicas, mostrando un resumen de sus características principales en la Tabla 1.

Amazon Web Services (AWS) [8]: Fue el pionero de los proveedores de servicios Cloud en el año 2006 y es la infraestructura Cloud publica más grande en la actualidad[1]. Inicialmente proporcionaba solo funcionlidad basada en el modelo IaaS, mediante los servicios de cómputo *Elastic Compute Cloud* (EC2) y de almacenamiento *Simple Storage Service* (S3). Con la evolución de las tecnologías ha ido ampliando la cantidad de servicios de manera que en la actualidad permite un gran número de servicios adicionales, que ya no permite encasillarla como solo un proveedor de tipo IaaS. Entre los servicios disponibles cabe destacar los siguientes:

- Elastic Block Store (EBS): proporciona volúmenes de almacenamiento a nivel de bloque diseñados para utilizarlos con las instancias de Amazon EC2.

[1] http://readwrite.com/2013/08/21/gartner-aws-now-5-times-the-size-of-other-cloud-vendors-combined

- Relational Database Service (RDS) y DynamoDB: proporciona soporte para el almacenamiento de bases de datos tanto de tipo SQL como de tipo NoSQL
- Auto Scaling permite escalar automáticamente la capacidad de una infraestructura desplegada sobre Amazon EC2 para aumentarla o reducirla de acuerdo con las condiciones que defina.
- Elastic Load Balancing distribuye automáticamente el tráfico entrante de las aplicaciones entre varias instancias de Amazon EC2.
- Elastic Beanstalk permite gestionar de manera automática los detalles de despliegue del aprovisionamiento de capacidad, equilibrio de carga, auto-escalado y gestión del estado de una aplicación.
- Virtual Private Cloud (Amazon VPC) permite aprovisionar una sección de red aislada de forma lógica del resto de la red de AWS, donde puede lanzar recursos de AWS en una red virtual definida por el usuario.
- CloudFormation es un servicio que ofrece un método sencillo de crear una colección de recursos de AWS relacionados entre sí para ofrecerlos de una manera ordenada y predecible.
- OpsWorks es un servicio de gestión de aplicaciones que permite el uso de herramientas de contextualización para configurar las MVs una vez lanzadas.
- Simple Workflow Service: es un servicio de flujo de trabajo indicado para crear aplicaciones escalables y flexibles.

Google Cloud Platform [9]**:** a diferencia del caso de Amazon, Google se centró inicialmente en proporcionar una plataforma tipo PaaS con el servicio *Goggle App Engine*, lanzado en 2011, que permite ejecutar aplicaciones web en la infraestructura de Google.

Google App Engine admite aplicaciones escritas en varios lenguajes de programación. Proporciona un entorno de ejecución Java de App Engine, que permite crear aplicaciones a través de tecnologías Java estándar, que incluyen JVM, servlets Java y el lenguaje de programación Java, o cualquier otro lenguaje que utilice un intérprete o compilador basado en Java como, por ejemplo, JavaScript o Ruby. App Engine también ofrece un entorno de tiempo de ejecución Python dedicado.

Recientemente ha ampliado sus productos para ofrecer también servicios de tipo IaaS para computación y almacenamiento. Dichos servicios se acaban de poner "en producción" a finales de 2013. Entre los servicios disponibles cabe destacar los siguientes:

- Google Compute Engine: Permite el lanzamiento de MVs utilizando S.O. Debian o CentOS. De manera similar a AWS, también proporciona un servicio de balanceo de carga entre varias instancias.
- Load Balancing: De manera similar a AWS, también proporciona un servicio de balanceo de carga entre varias instancias.
- Google Cloud Storage: Servicio similar al S3 de AWS, proporcionando almacenamiento de objetos de gran tamaño.
- Google Cloud SQL/DataStore: Proporciona soporte para el almacenamiento de bases de datos tanto de tipo MySQL como de tipo NoSQL
- Google BigQuery: Este servicio proporciona funcionalidad para realizar consultas de tipo SQL a grande conjuntos de datos (*multi-terabyte datasets*)

Microsoft Windows Azure [10]: Al igual que el caso de Google, Microsoft entra en las tecnologías Cloud proporcionando servicios de tipo PaaS en 2010 que permite ejecutar aplicaciones web en la infraestructura Azure. Dichas aplicaciones deben funcionar sobre Windows Server 2008 R2. Pueden estar desarrolladas en .NET, PHP, C++, Ruby, Java. Además del servicio de ejecución, dispone de diferentes mecanismos de almacenamiento de datos: tablas NoSQL, blobs, blobs para streaming, colas de mensajes o discos NTFS para operaciones de lectura/escritura a disco.

De la misma forma que otros proveedores Cloud, han ido añadiendo nuevas funcionalidades para proporcionar también servicios de tipo IaaS, inicialmente con la introducción del nuevo tipo de rol denominado "VM Role" que permite el lanzamiento de MVs de tipo Windows. Desde Abril de 2013[2] tiene ya en producción los denominados *Servicios de Infraestructura*, proporcionando los siguientes servicios:

- Máquinas Virtuales: Permite el lanzamiento de cualquier tipo de MV (basada Windows o en Linux) tanto usando el conjunto de IMVs base proporcionado por Azure como imágenes subidas por usuario en el formato utilizado por el hipervisor de Microsoft (Hyper-V), realizando un proceso de preparado de imagen específico para la plataforma de Azure (similar al caso de EC2).
- Almacenamiento: Azure proporcionan tres tipos de almacenamiento bases de datos SQL basadas en las tecnologías de SQL Server, tablas que ofrecen

[2] http://blogs.msdn.com/b/windowsazure/archive/2013/04/16/the-power-of-and.aspx

funcionalidad NoSQL para las aplicaciones que requieren el almacenamiento de grandes de datos no estructurados y blobs para almacenar grandes cantidades de datos, que luego pueden ser montadas como particiones NTFS.

- Red Virtual: permite crear en una sección de red aislada lógicamente y conectarla de forma segura al centro de datos local o a un único equipo cliente mediante una conexión IPsec.

Con la expansión de las tecnologías Cloud en los últimos años, en la actualidad existen muchas empresas que proporcionan servicios Cloud en cualquiera de sus modelos, aunque en general el más abundante es el modelo IaaS. Varios ejemplos son Rackspace[3], Linode[4], GoGrid[5], RightScale[6], Salesforce.com[7], IBM[8], HP[9] etc.

Todas estas plataformas proporcionan el acceso a sus infraestructuras con un modelo de despliegue de Cloud público y cobrando por los servicios mediante "pago por uso". De tal manera, los usuarios solo pagan por aquellos recursos que utilizan.

Este tipo de plataformas proporcionan una gran cantidad de recursos y una funcionalidad base que ofrece muchas posibilidades tanto en el ámbito comercial como en el científico. En esta tesis vamos a tratar de proporcionar un conjunto de funcionalidades adicionales para aumentar esta funcionalidad base y hacerla más cercana y fácil al investigador.

[3] http://www.rackspace.com/

[4] http://www.linode.com/

[5] http://www.gogrid.com/

[6] http://www.rightscale.com/

[7] http://www.salesforce.com/

[8] http://www.ibm.com/es/cloud-computing/index.html

[9] https://www.hpcloud.com/

Tabla 1. Comparación proveedores Cloud

	Amazon AWS	Microsoft Azure	Google Cloud
S.O. Soportados	- Windows - Cualquier Linux	- Windows - Cualquier Linux	- Debian - CentOS
Almacenamiento	- Objetos - SQL - NoSQL - Volúmenes	- Objetos - SQL - NoSQL	- Objetos - SQL - NoSQL
Balanceo de Carga	SI	SI	SI
Red	Avanzada	Avanzada	Simple
Tipos de Instancias	23	7	5
Mayor instancia	- 32 VCPU (88 ECU) - 244 GB RAM - 4 x 840 GB HD	- 8 VCPU - 56 GB RAM - 605 GB HD	- 16 VCPU (22 GCEU) - 104 GB RAM - 10 TB HD[10]
Soporte para GPUs	SI	NO	NO
U. de rendimiento	ECU[11]	1.6GHz VCPU	GCEU[12]
Autoescalado	SI	NO	NO
Lanzamiento de MVs Coordinado	SI	NO	NO
Contextualización	SI	NO	NO

2.3 Cloud Management Platforms (CMPs)

Las CMPs proporcionan la funcionalidad para gestionar un conjunto de recursos físicos sobre los que desplegar un conjunto de máquinas virtuales. Como todo gestor de recursos, tienen un funcionamiento similar a los sistemas de gestión de recursos (Local Resource Management Systems – LRMS), como Torque, SGE, etc. Por tanto deben gestionar el uso los recursos, equilibrar la carga, monitorización de los recursos físicos, gestión del ciclo de vida de los trabajos (en este caso las MVs), autenticación, etc. Además se basan en el uso de software de virtualización (denominados hipervisores) como pueden ser KVM, XEN, VMWare, VirtualBox, Hyper-V, etc. encargados de la ejecución de las MVs sobre los recursos físicos.

[10] 10TB de tamaño máximo. Cobro del almacenamiento aparte del precio de la instancia

[11] http://aws.amazon.com/ec2/faqs/#What_is_an_EC2_Compute_Unit_and_why_did_you_introduce_it

[12] https://developers.google.com/compute/docs/instances

Hay varios proyectos de software libre que trabajan en el desarrollo de CMPs, de manera que posibilitan que tanto empresas como centros de investigación puedan desplegar dentro de sus instalaciones sus propias plataformas Cloud. En este artículo [11] se encuentra una buena revisión de los CMPs más populares en la actualidad, entre los que cabe destacar OpenNebula [12], OpenStack [13], Eucalyptus [14], o Nimbus [15]. Dada esta diversidad, en los últimos años han surgido diferentes iniciativas que tratan de unificar criterios para la definición de un API estándar único para el acceso a las plataformas Cloud IaaS como Open Cloud Computing Interface (OCCI) [16] o Cloud Infrastructure Management Interface (CIMI) [17].

A continuación se proporciona una breve descripción de cada uno de ellos, mostrando un resumen de sus características principales en la Tabla 2.

OpenNebula proporciona un entorno IaaS de software libre. Tiene un diseño modular que permite la integración con diferentes sistemas de almacenamiento, configuraciones de red, e hipervisores. Permite la gestión de los recursos, la migración de MVs en vivo, crear "snapshots" de las MVs en ejecución, la gestión de fallos de los recursos físicos, etc. Además permite la integración con otras plataformas Cloud externas (como la de Amazon EC2) para permitir una mayor escalabilidad. Esto permite, por ejemplo, el control de varios despliegues de OpenNebula desde un único punto central.

OpenNebula proporciona diferentes interfaces de acceso, entre ellos el estándar OCCI de tipo REST, el interfaz propio de Amazon EC2 y la propia API interna de OpenNebula mediante XML-RPC.

OpenNebula soporta tres tipos de autorización: la básica usando nombre de usuario/contraseña, usando certificados X.509 o Lightweight Directory Access Protocol (LDAP). Además utiliza un sistema de listas de control de acceso (ACLs de su acrónimo en inglés Access Control List) para permitir definir los permisos de los usuarios/grupos con mayor detalle.

En cuanto al sistema de almacenamiento permite usar tanto sistemas de almacenamiento compartido entre los nodos de la infraestructura (NFS, GlusterFS, Lustre), como sistemas no compartidos usando SSH para la transferencia de ficheros, así como sistemas de tipo Logical Volume Manager (LVM) con CoW (copy-on-write).

OpenStack es un conjunto de componentes de software libre para la creación de Clouds tanto públicos como privados. Los componentes principales que incluye OpenStack son: OpenStack compute (Nova), OpenStack Object Storage (Swift), OpenStack Image Service (Glance) y OpenStack Identity Service (Keystone).

El servicio Nova está diseñado para aprovisionar y gestionar un gran número de máquinas virtuales, para crear una plataforma Cloud escalable. Swift se utiliza para

la creación de un sistema de almacenamiento de objetos redundante y escalable usando un conjunto de servidores estándar para crear un sistema de almacenamiento de gran tamaño (similar al servicio S3 de Amazon). Glance proporciona un sistema para registro, descubrimiento y entrega de imágenes de máquinas virtuales. Por último Keystone es el servicio de gestión de identidades que permite la autenticación y autorización de usuarios. Soporta diferentes tipos de autenticación incluyendo la estándar usuario – contraseña como sistemas basados en el en uso de tokens y logins similares a los utilizados por Amazon WS.

Eucalyptus implementa un Cloud privado tipo IaaS que es accesible vía un API compatible con los proporcionados por Amazon EC2 y S3. Tiene cinco componentes de alto nivel: Cloud Controller que gestiona los recursos virtuales. Cluster Controller que controla la ejecución de máquinas virtuales. Walrus es el sistema de almacenamiento. Storage Controller proporciona un sistema de almacenamiento de bloques que incluye soporte para semánticas del tipo de Amazon EBS (volúmenes orientados a bloques). Finalmente Node Controller está instalado en cada nodo de cálculo para controlar las actividades de las MVs, incluyendo la ejecución, inspección, y terminación de las instancias de las MVs.

El proyecto **Nimbus** está trabajando en dos productos: Nimbus Platform y Nimbus Infrastructure.

Nimbus Infrastructure se define como "una plataforma IaaS de software libre compatible con Amazon EC2/S3". Para ello Nimbus proporciona su propia implementación de un sistema de almacenamiento compatible con S3 mejorado con sistema de gestión de cuotas. También proporciona servicios de cómputo compatibles con EC2, y un cliente Cloud que internamente utiliza WSRF (Web Services Resource Framework) [18].

Nimbus Platform proporciona un conjunto de herramientas adicionales para facilitar la gestión de infraestructuras y facilitar la integración con otros Clouds (OpenStack y AWS). Actualmente incluye cloudinit.d [19] para el lanzamiento, control y monitorización de aplicaciones Cloud y el Context Broker que coordina el lanzamiento de clústeres virtuales. Ambas herramientas serán explicadas con más detalle en la sección 2.5.

2.4 APIs de Conexión IaaS

Uno de los problemas que surgen de la existencia de diferentes proveedores de plataformas Cloud es la existencia de diferentes protocolos de conexión. Cabe destacar el caso de los protocolos usados por AWS, que al ser el pionero y el de mayor tamaño de los proveedores Cloud de tipo IaaS en la actualidad, está "imponiendo" sus APIs propietarias como estándar "de facto". Al ser un API propietaria tiene el riesgo de que el propietario pueda cambiar la especificación sin necesidad de consenso con los demás proveedores de la misma, o puede imponer

restricciones al uso de la misma. Por tanto se deben buscar alternativas estándar para el acceso interoperable a los proveedores Cloud. Sin embargo existe una gran dificultad para encontrar un API estándar único para el acceso a las plataformas Cloud IaaS.

Tabla 2. Comparación CMPs

	OpenStack	Eucalyptus	Nimbus	OpenNebula
Interfaces	- EC2/S3, - OCCI	- EC2/S3	- EC2/S3, - REST API	- XML-RPC - EC2/S3 - OCCI
Hipervisores	- KVM - XEN - VMware Vsphere - LXC - UML - MS HyperV	- KVM - XEN - VMware EE	- KVM - XEN	- KVM - XEN - VMware
Red	- Bridges creados aut. - MVs solo IP privada - IP forwarding	- Bridges creados aut. - MVs solo IP privada - IP forwarding	- IPs usando DHCP - No crea Bridges	- Ebtables, Open vSwitch y 802.1Q tagging - No crea Bridges - IPs configuradas en las MVs
Despliegue	- Componentes independientes pueden ser instalados en diferentes nodos - Nodos de cálculo necesitan software de OpenStack	- Componentes independientes pueden ser instalados en diferentes nodos - Nodos de cálculo necesitan software de Eucalyptus	- Software instalado en nodo fontend y en nodos de internos	- Software instalado en nodo fontend
Almacenamiento	- Swift (http/s) - Unix filesystem (ssh)	- Walrus (http/s)	- Cumulus (http/s)	- Sistema de ficheros Unix (ssh, sistema compartido o LVM con CoW)
Autenticación	- Credenciales X.509 - LDAP	- Credenciales X.509	- Credenciales X.509 - Grids	- Credenciales X.509 - LDAP - ssh rsa keypair - Usuario/contraseña - LDAP

2.4.1 Estándares

En los últimos años han surgido diferentes iniciativas que tratan de unificar criterios para la definición de un API estándar único para el acceso a las plataformas Cloud IaaS. A continuación se muestran las más destacadas:

Open Cloud Computing Interface (OCCI) [16]: es una "recomendación" del Open Grid Forum (OGF) de un protocolo y un API de tipo REST para el acceso remoto a plataformas Cloud de tipo IaaS. Depende de los proveedores Cloud el proporcionar acceso a sus plataformas usando dicho API. En la actualidad los paquetes de software Cloud OpenNebula y OpenStack disponen de dicho soporte.

Cloud Infrastructure Management Interface (CIMI) [17]: CIMI es el intento de estandarización del Distributed Management Task Force (DMTF) para simplificar la gestión de infraestructuras Cloud. Al igual que OCCI es un API de tipo REST que permite que los recursos se codifiquen usando JavaScript Object Notation (JSON) o eXtensible Markup Language (XML). Es un estándar de reciente creación (28 de Agosto de 2012) y aun no hay ningún software o proveedor que lo implemente.

TCloud API Specification [20]: TCloud es una propuesta de API Cloud impulsada por la empresa Telefónica que está basado en la especificación de API vCloud 0.8 publicado por VMWare. Además, TCloud extiende y mejora las capacidades avanzadas para la gestión de Cloud computing, que incluyen la provisión de elementos de red, almacenamiento compartido para datos de servicio, monitorización, gestión de copias de seguridad, etc. En la actualidad solo Claudia [21], desarrollado por la propia empresa, lo implementa.

De estos tres estándares OCCI es el que tiene mayor número de implementaciones. Tiene el inconveniente que diferentes CMPs implementan diferentes versiones (OpenNebula la versión 0.8 y OpenStack la 1.1) que no son totalmente compatibles haciendo difícil la interoperabilidad.

2.4.2 APIs de agregación

A pesar de la existencia de estándares, su adopción es limitada en la industria. Como respuesta a esta carencia, las denominadas APIs de agregación tratan de facilitar el acceso homogéneo a diferentes proveedores Cloud proporcionando una compatibilidad cruzada entre las APIs. A continuación se detallan los proyectos actuales de mayor trascendencia:

Apache Libcloud[13]: Apache Libcloud es una librería en Python que trata de proporcionar el acceso a servicios de cómputo, almacenamiento, balanceo de carga y DNS proporcionados por un gran número de proveedores Cloud de manera homogénea.

Apache jClouds[14]: Apache jClouds es una librería en Java para el acceso a diferentes proveedores Cloud. Permite al acceso a servicios de cómputo y de

[13] http://libcloud.apache.org/

[14] http://jclouds.apache.org

almacenamiento. Proporciona un API que permite tanto el uso de funciones abstractas interoperables o funciones específicas para cada plataforma Cloud.

Deltacloud[15]: DeltaCloud en un API desarrollado por RedHat para abstraer las diferencias entre los diferentes Clouds. DeltaCloud consiste en un servidor y un conjunto de drivers necesarios para conectar con los proveedores Cloud. Proporciona acceso a la funcionalidad mediante tres sistemas: un API nativa en Ruby, un API al servidor REST y acaban de desarrollar un api para C/C++.

Fog[16]: Es una librería en Ruby para permitir el acceso a los servicios básicos (almacenamiento, computación y DNS) de varios proveedores Cloud, en concreto Amazon Web Services, Google Compute y Rackspace.

Aunque en principio estas herramientas son muy útiles, dada la heterogeneidad de las APIs utilizadas por las diferentes plataformas Cloud es casi imposible usar un API genérica para crear MVs en diferentes plataformas Cloud. Esto obliga a las APIs de agregación a tener un conjunto reducido de funciones comunes a todos los proveedores Cloud, y gran conjunto de funciones que son específicas para cada uno de ellos. Ello provoca que para poder acceder a un conjunto de proveedores es necesario utilizar para cada uno de ellos sus funciones específicas, y por tanto generando casi tanto código como usando su API propia. Además en el caso concreto de LibCloud no soporta algunas de las funciones básicas (en concreto parar y relanzar MVs) en todos sus "drivers". Además el soporte de OpenNebula (uno de los principales CMPs usados en investigación) es muy limitado y no usa su API nativa. Por otra parte, no permite de forma general la gestión de información sobre las redes de las plataformas Cloud. En el caso de DeltaCloud, tiene problemas similares con el acceso a OpenNebula y en el caso del API tipo REST es necesario lanzar tantos servidores como proveedores Cloud se quiera acceder, lo que complica su despliegue. El caso de Fog, tiene un soporte de proveedores muy limitado (solo tres) además de ser una librería de Ruby, que limita mucho su acceso desde otros lenguajes de programación. Dada la dificultad de encontrar tanto un protocolo estándar como una librería que permite un acceso sencillo y homogéneo, en el ámbito de la tesis, hemos desarrollado una capa de abstracción para acceder de forma sencilla y eficaz a todos los proveedores Cloud posibles, incluyendo entre ellos el mayor número de estándares posibles.

[15] http://deltacloud.apache.org/

[16] http://fog.io/

2.5 Lenguajes de especificación de requerimientos Cloud

Una de las necesidades básicas a la hora del despliegue de infraestructuras Cloud es disponer de un lenguaje que permita al usuario expresar los requerimientos de sus aplicaciones a ejecutar en el Cloud.

En primer lugar, es necesario definir las características hardware de la máquina. En este contexto uno de los lenguajes estándar actuales es el OVF (Open Virtualization Format) [22]. Es un lenguaje independiente de la plataforma y del fabricante que permite la descripción de Virtual Appliances (VAs). Una VA es el conjunto de MVs con todo el software necesario configurado que proporciona un determinado servicio. OVF permite describir todos los datos necesarios para el despliegue de un conjunto de MVs en cualquier proveedor Cloud: características hardware (CPU, memoria, etc), imágenes de los discos, las redes con las que se interconectan, etc. Además tiene una serie de apartados adicionales donde se pueden añadir información sobre los paquetes software de las MVs.

OVF está basado en XML por lo que de forma nativa se permite su extensión, como por ejemplo la del lenguaje SDF (Service Description File) definido en el proyecto Claudia [21] que le añade capacidades de definición de reglas de elasticidad simples.

Este lenguaje tiene una serie de ventajas muy importantes, partiendo de que es un estándar definido por la DMTF (Distributed Management Task Force) y que es ampliamente aceptado por numerosos desarrolladores de software de virtualización como VMWare o VirtualBox y que es fácilmente extensible. Pero a la vez tiene una serie de inconvenientes: el primero es que no es un lenguaje sencillo de entender para un usuario no acostumbrado a los lenguajes de tipo XML, además al ser un lenguaje orientado al despliegue, y no a la expresión de requisitos sobre las MVs, no contempla la posibilidad de usar comparadores (<, >, >=, <=) en las asignaciones de las características de las MVs, ni el uso de operadores booleanos (and, or) a la hora de expresar los requerimientos de las infraestructuras.

Otros estándares Cloud existentes como OCCI no definen ningún formato para la descripción de las MVs, sino que simplemente definen un conjunto de atributos en formato de pares clave – valor usando cierta notación para el nombre del atributo. En el caso de CIMI sí que define un lenguaje basado en XML para la descripción de las MVs. En este caso el lenguaje es menos completo que el de OVF y no contempla ciertos aspectos a la hora de definir una VA como son la agrupación de MVs o los componentes de software. Además el lenguaje es aún más complejo que el definido en OVF.

Algunos de los paquetes software que proporcionan la funcionalidad para crear Clouds privados, como OpenNebula, proporcionan un lenguaje propio para la

definición de las MVs en lo que denominan templates[17]. Dicho lenguaje, basado en en el uso de pares clave – valor, permite la definición de las características necesarias para poder desplegar una MV en el despliegue de OpenNebula. Es un lenguaje bastante sencillo pero que solo permite definir las características físicas de una MV.

Otra posibilidad es DADL (Distributed Application Description Language) [23], una extensión del lenguaje SmartFrog [24] (un entorno para configurar, desplegar y gestionar sistemas de software distribuidos) diseñado para expresar la arquitectura, comportamiento y necesidades de una aplicación distribuida y características de los recursos Cloud disponibles. Este lenguaje, al igual que la mayoría de los anteriores, se basa en la especificación de las necesidades de las aplicaciones a nivel hardware, olvidándose de la parte software que es vital para tener configurado una infraestructura Cloud para poder hacer uso correcto de ella.

Todos estos lenguajes aportan una serie de características importantes, pero no proporcionan toda la capacidad expresiva necesaria ni la facilidad en el lenguaje usado, si la capacidad de expresar requerimientos de configuración de software. Por tanto en el desarrollo de la tesis definiremos un nuevo lenguaje que, tomando los mejores conceptos de los lenguajes analizados, tenga toda la funcionalidad necesaria para que el investigador pueda expresar los requerimientos de los recursos que necesita para ejecutar sus aplicaciones, tanto a nivel hardware como software y de configuración.

2.6 Herramientas de contextualización

Las herramientas de contextualización en entornos Cloud se utilizan para facilitar todas las tareas necesarias para tomar una MV base e instalar y configurar todo el software necesario para que tenga la funcionalidad necesaria para el usuario.

En los últimos tiempos este tipo de herramientas (denominadas *DevOps*) han cobrado gran difusión de manera que han aparecido gran multitud de ellas, entre ellas cabe destacar Ansible, Puppet, Capistrano, Chef y CFEngine. Este tipo de herramientas permiten automatizar la gestión de sistemas desde un único punto utilizando un lenguaje de alto nivel, que se denominan de forma general con "DSL" (de su acrónimo en inglés domain-specific language) basado en políticas de configuración con su propia sintaxis. La mayoría de estas herramientas se han usado tradicionalmente para el despliegue y configuración de aplicaciones en centros de datos y que recientemente se han adaptado a los entornos Cloud.

CFEngine [25] en la actualidad se distribuye tanto como un proyecto de software libre como un producto comercial con una serie de características adicionales.

[17] http://opennebula.org/documentation:archives:rel4.0:template

CFEngine es una de las primeras herramientas de configuración de software, iniciado en 1993.

CFEngine tiene una estructura completamente distribuida. Cada nodo puede funcionar de forma independiente del resto, manteniendo los ficheros necesarios para su configuración. Pero también puede funcionar usando un repositorio centralizado en un único servidor. CFEngine utiliza un conjunto de servicios ejecutándose en los nodos que se encargan de conectarse con el servidor central, de forma periódica, para asegurarse la correcta configuración. En este caso el servidor central solo actuaría como servidor de ficheros, de manera que una vez descargados es el propio cliente el que ejecuta todas las tareas de configuración en local. La comunicación con el servidor se hace usando un modelo "pull". Los clientes mantienen una cache local de los ficheros de configuración, y si detectan que hay una versión modificada se conectan con el servidor para descargársela.

CFEngine usa un lenguaje declarativo propio para expresar el estado previsto del sistema basado en el contexto. Usa una estructura modular para permitir reutilizar funciones en distintas tareas.

CFEngine pone a disposición de sus usuarios un repositorio de tipo git[18] con ejemplos, herramientas adiciones y lo que ellos denominan sketches, es decir, un conjunto de ficheros de configuración para la instalación/configuración de múltiples herramientas estándar, para facilitar la reutilización.

Puppet [26] apareció en 2003 de manos de reductive labs, actualmente denominado Puppet labs. Su objetivo es facilitar las tareas de los administradores de sistemas ocultando los detalles de implementación de la configuración usando un lenguaje declarativo orientado en realizar las tareas de forma independiente del sistema operativo. La filosofía básica de Puppet es la creación de una capa de abstracción para facilitar las tareas de automatización de la configuración de los sistemas sin tener que lidiar con los detalles de implementación en cada sistema operativo. Esta capa de abstracción facilita la reutilización de código y la modularidad usando herencia de clases. El DSL de Puppet está basado en Ruby, y también permite incluir directamente código Ruby para dar mayor extensibilidad al lenguaje de manera que el usuario pueda programar su propia funcionalidad.

Puppet está diseñado para funcionar en forma de cliente/servidor. Todos los ficheros de configuración con las "recetas" escritas en el lenguaje declarativo de Puppet son almacenadas en el nodo servidor denominado "Puppet Master". Este nodo central se encarga de muchas de las tareas (a diferencia del caso de CFEngine) como analizar las recetas y compilarlas para generar catálogos. Estos catálogos son conjuntos de ficheros XML que serán recogidos por los clientes o agentes Puppet.

[18] https://github.com/cfengine/design-center

Los clientes solo se encargan de implementar la funcionalidad requerida al comparar los catálogos almacenados en su cache local con los del servidor y finalmente reporta al servidor los cambios realizados. El Puppet Master también puede enviar notificaciones a los clientes en caso de que los ficheros de configuración hayan cambiado para hacer que el cliente obtenga los nuevos catálogos (estrategia de tipo "pull").

Al igual que en el caso de CFEngine, Puppet proporciona un repositorio de recetas, denominado Puppet Forge[19], para un gran conjunto de aplicaciones, pero a diferencia del caso anterior no solo proporciona un repositorio simple, sino que le añade un conjunto de metadatos como los requisitos, la valoración de los usuarios, etc. que permite un acceso más simple a las recetas. Además proporciona utilidades para la búsqueda, instalación y configuración de dichas recetas en la instalación propia del usuario, facilitando mucho la reutilización de las mismas.

Chef [27] aparece en 2009 de la mano de Opscode. Chef ha sido diseñado con la filosofía de "infraestructura como código" para proporcionar flexibilidad y sencillez a la hora de gestionar infraestructuras. De forma similar a los otros productos, proporciona un lenguaje específico para especificar las reglas de funcionamiento, que permite la inclusión de código Ruby de manera que de mayor flexibilidad para que el usuario pueda programar su propia funcionalidad.

La estructura de Chef es de tipo cliente/servidor. El servidor Chef es el encargado de almacenar todos los ficheros de configuración, transferirlos a los clientes cuando los soliciten y almacenar el estado de los clientes. Los clientes son lo que se encargan de la compilación y ejecución de las tareas de configuración.

Chef también proporciona una herramienta, denominada *cookbooks*[20], con una funcionalidad muy similar a la de Puppet forge, con un repositorio con un conjunto de metadatos y herramientas para la instalación automatizada de las "recetas".

Capistrano [28] es un entorno para la ejecución de comandos en paralelo en un conjunto de máquinas remotas usando SSH. Tiene un esquema más sencillo que todos los anteriores de tal manera que los nodos a gestionar no necesitan de ningún software especial para ser gestionados con Capistrano, solamente necesitan el intérprete de Ruby. Se le indican en la configuración un conjunto de nodos a gestionar, que pueden estar agrupados según sus roles dentro de la infraestructura. A diferencia de las herramientas anteriores usa una estrategia "push" para la configuración de los nodos, de manera que es el nodo configurador el que tiene el control de cuándo se realizan las configuraciones de los nodos. Capistrano se

[19] http://forge.puppetlabs.com/

[20] http://community.opscode.com/cookbooks

encarga de su configuración accediendo a ellos usando SSH estándar. Utiliza un lenguaje sencillo de tipo DSL que tiene su origen en Rake[21], que permite la definición de tareas que deben ser aplicadas en conjuntos de máquinas agrupados por roles.

Ansible [29] es una herramienta muy reciente (Abril de 2012) y ha sido creada por algunos de los desarrolladores del proyecto Puppet. Tiene un esquema similar a Capistrano, usando una estrategia de tipo "push" mediante SSH para acceder a los nodos. También tiene unos requerimientos de instalación muy simples. Tan solo es necesario el intérprete de Python (que viene por defecto en todas las distribuciones de Linux). Para la definición de las recetas se utiliza el lenguaje YAML (YAML Ain't Markup Language[22]). Al ser un proyecto muy reciente aún no tiene el soporte de los proyectos anteriores y aunque tiene repositorios de recetas (usando proyectos de GitHub[23]) el soporte no es tan completo como los casos de Chef o Puppet.

También cabe destacar el caso de la herramienta desarrollada dentro del ámbito del grupo de investigación GRyCAP [30]. Es una herramienta sencilla desarrollada en Python, muy sencilla y fácil de utilizar, aunque con una funcionalidad más limitada que los productos anteriores, pero sin la necesidad de instalar ningún componente adicional. De esta manera se hace muy sencillo su utilización en cualquier tipo de MV, con el simple requisito de tener instalado un intérprete de Python. Esta herramienta utiliza un conjunto de ficheros en XML donde se definen los denominados Application Deployment Description (ADD) donde se especifican todos los pasos necesarios para desplegar una aplicación (compilación, instalación, post procesado, etc.). El contextualizador puede funcionar tanto en modo aplicación simple como en modo cliente-servidor donde tanto los paquetes como los ficheros con las ADDs pueden ser descargador del nodo servidor.

2.7 Catálogos de imágenes de máquinas virtuales

El almacenamiento e indexación de imágenes de máquinas virtuales (IMV) se está convirtiendo en una tarea crucial para los investigadores para poder escoger de forma apropiada la IMV que más se ajusta a las necesidades de un usuario entre el gran número de MVs que se utilizan en la actualidad. Los catálogos permiten además compartir IMVs de tipo genérico que pueden ser reutilizadas para diferentes propósitos, aumentando la colaboración entre los científicos. Por ello en la actualidad existen diferentes repositorios de IMVs que se describen a continuación.

[21] http://rake.rubyforge.org/

[22] http://www.yaml.org/

[23] https://github.com/

VMware Marketplace [31], desarrollado por VMWare Inc. permite la indexación de IMVs para usar con su propio gestor de virtualización. Proporciona un interfaz web donde se introduce una categorización de las imágenes dependiendo del tipo de organizaciones a las que está orientada (educación, salud, servicios financieros, etc.), el tipo de tecnología que utiliza (desarrollo de aplicaciones, bases de datos, almacenamiento, etc.), la categoría (aplicaciones, Cloud, centro de datos, etc.). Además permite la búsqueda usando palabras clave para encontrar la IMVs para el usuario. El portal es de ámbito comercial y la descarga de las imágenes puede tener un coste económico dependiendo del tipo de imagen a descargar. Para subir imágenes al repositorio el usuario debe estar registrado en el portal y usar el software VMWare Studio que puede ser descargado de la web de VMWare sin coste.

Amazon EC2 proporciona un repositorio de las imágenes **Amazon Machine Images** (AMIs) [32] usadas en su propia infraestructura. Las imágenes están organizadas jerárquicamente usando diferentes características (proveedor, sistema operativo, región), y sólo proporciona herramientas de búsqueda usando palabras clave. Amazon proporciona funcionalidad para subir nuevas AMIs a su repositorio a través de su API. Hay que tener en cuenta que para poder subir estas imágenes, debe ser un usuario registrado de Amazon y que el almacenamiento de dichas imágenes supone un coste.

Cloud Market [33] es un repositorio que cataloga las imágenes de Amazon EC2. Dicho portal escanea Amazon EC2 y extrae la información propia de la IMV (plataforma, kernel, etc.). Los usuarios pueden reclamar dichas imágenes y añadir más información sobre las mismas. Proporciona un conjunto de herramientas simples para buscar las imágenes especificando parámetros como: plataforma, arquitectura, y etiquetas adicionales.

StratusLab Marketplace [34] es un repositorio desarrollado dentro del marco del proyecto StratusLab[24] del 7º programa marco de la unión europea. En este caso solo proporciona la indexación de IMVs almacenadas en URLs remotas. Los metadatos de las imágenes están en formato RDF-XML[25], pero solo permite almacenar información sobre el sistema operativo (nombre, versión, arquitectura). En cuanto al interfaz proporciona tanto un interfaz web como uno REST, lo que permite el acceso automatizado. Para las búsquedas avanzadas utiliza el lenguaje SPARQL[26]

[24] http://stratuslab.eu

[25] http://www.w3.org/TR/REC-rdf-syntax/

[26] http://www.w3.org/TR/rdf-sparql-query/

La mayoría de los repositorios analizados están diseñados para ser utilizados con un determinado hipervisor o están unidos a un sistema Cloud determinado. Además solo proporcionan unas herramientas muy básicas de búsqueda y en la mayor parte de los casos únicamente a través de un interfaz web, lo que no permite realizar búsquedas avanzadas de forma automatizada. El único que proporciona este tipo de funcionalidades es el StratusLab Marketplace pero tiene un conjunto muy reducido de metadatos, lo que dificulta una correcta búsqueda e indexación de imágenes.

2.8 Trabajos Relacionados

En las secciones anteriores se han mostrado diferentes componentes necesarios para alcanzar los objetivos marcados para la tesis (lenguaje de expresión de requerimientos sencillo, catálogo, soporte de múltiples proveedores, contextualización flexible, readaptación dinámica, soporte de estándares, etc.). Pero cada uno de ellos solo es capaz de proporcionar alguna de funcionalidad concreta de las requeridas y no permite abordar de forma global el lanzamiento y gestión de infraestructuras virtuales configuradas dinámicamente. A continuación se van a mostrar un conjunto de soluciones que tratan de proporcionar esa visión global. Son sistemas que funcionan como capas de software superiores a los proveedores de Cloud públicos o a los CMPs de manera que complementen dicha funcionalidad de cara a facilitar el despliegue de infraestructuras virtuales. A continuación vamos a mostrarlos analizando sus características, con respecto a los objetivos marcados en la tesis para revisar las principales necesidades a desarrollar.

Algunos proveedores Cloud como Amazon proveen cierta funcionalidad para el despliegue de infraestructuras. **CloudFormation** [35] ofrece a desarrolladores y administradores de sistemas un método sencillo de crear una colección de recursos de AWS relacionados entre sí para ofrecerlos de una manera ordenada y predecible. Proporciona una funcionalidad muy sencilla y básica para lanzar varias MVs de forma coordinada. Además recientemente Amazon ha puesto disponible un nuevo servicio denominado **OpsWorks** [36] (aún en versión beta) que permite integrar el uso de recetas Chef para la contextualización de las MVs.

Ventajas:

- Sencillez e integración con el resto de servicios de AWS.

Inconvenientes:

- Solo para Amazon EC2

El proyecto Nimbus de la Universidad de Chicago, además del conjunto de herramientas básicas proporcionadas por una plataforma IaaS, ha desarrollado el **Nimbus Context Broker** [37]. Este software permite la contextualización de MVs para la creación de lo que ellos denominan “One-Click Virtual Clusters”. Esta herramienta usa una serie de conceptos de “proveer” o “requerir” un rol de una MV,

para crear conjuntos de MVs y configurarlas de acuerdo a los roles que tengan dentro de un clúster.

Ventajas:

- Permite expresar de forma bastante sencilla las relaciones entre nodos con los conceptos de "proveer" y "requerir".

Inconvenientes:

- El lanzamiento de MVs se realiza con los comandos propios de Nimbus, por lo que solo es posible utilizar proveedores Cloud que usen este software. Aunque este inconveniente es ligeramente mitigado por un componente denominado *IaaS Gateway* que permite la conexión de un despliegue Cloud de Nimbus se conecte con Amazon EC2 o potencialmente con otros proveedores que usen el interfaz EC2.
- Uso de imágenes base estáticas: el usuario debe proporcionar la IMV que de utilizar y que debe conocer de antemano sus características para saber lo idónea que puede ser para sus necesidades en cada caso. Esto no permite la reutilización de otras imágenes existentes (creadas por él o por otros usuarios) que pueden ser mejor para su caso de uso.
- La contextualización tiene algunos problemas: Los scripts deben estar pre-alojados en las MV, por lo que las imágenes debe estar ya preparadas para cada aplicación que se vaya a lanzar en ella. Y no permite re-contextualización, es decir, en caso de añadir un nuevo nodo al sistema, no es capaz de re-configurar las máquinas ya existentes para aceptar al nuevo nodo.

Dentro del mismo grupo de investigación de Nimbus, también han desarrollado otro sistema [38] y [39] que permite la gestión de infraestructuras sobre el Cloud de forma elástica. Este sistema permite mediante el uso de sistema distribuido de sensores tomar las decisiones sobre la elasticidad de la infraestructura. En este caso para evitar los problemas de reconfiguración que tenían usando su Context Broker han cambiado el sistema de contextualización para usar una combinación de Chef para la instalación del software necesario en cada nodo y un nuevo componente, el "**Recontextualization Broker**", que han desarrollado para la configuración inicial y posterior reconfiguración (en caso de añadir o eliminar nodos) de los nodos del clúster. Con este nuevo desarrollo solucionan algunos de los problemas que tenían con su Context Broker pero tampoco sirve como una solución general. Al igual que en el caso anterior usan Nimbus como base, solo permitiendo acceso a Clouds Nimbus o EC2. Al igual que en el caso anterior tiene el inconveniente del uso de imágenes base estáticas.

Y siguiendo dentro del grupo de Nimbus han desarrollado otro sistema denominado **cloudinit.d** [19] para el lanzamiento, control y monitorización de aplicaciones Cloud. Está diseñado para automatizar la creación de las MVs, su contextualización y su coordinación en el lanzamiento.

Ventajas:

- Da soporte para múltiples Clouds.
- Permite sincronizar distintos "runlevels".
- Monitoriza el estado de los servicios de las MVs (usando scripts creados por el usuario).

Inconvenientes:

- El usuario debe crear scripts para instalar sus servicios en las MVs, esto es bueno porque permite generalizar un despliegue, pero implica que el usuario debe conocer esos detalles (debe ser usuario avanzado).
- Uso de imágenes base estáticas.

Finalmente comentar el servicio **Phantom** [40] proporcionado por la plataforma Cloud de Nimbus, que funciona de manera similar al CloudFormation y OpsWorks de AWS de manera que un usuario puede definir reglas de elasticidad y recetas de Chef para desplegar y gestionar de forma elástica su infraestructura. Este servicio, aunque es muy potente, no sirve como una solución general. Es un servicio propio de la plataforma de Nimbus proporcionando solo acceso a su propia plataforma o a Amazon EC2. Al igual que en el caso anterior tiene el inconveniente del uso de imágenes base estáticas.

Claudia [21] permite el despliegue de un conjunto de MVs en entornos IaaS, usando una extensión del lenguaje OVF que han denominado Service Description File (SDF). En el despliegue no permiten la instalación de nuevo software, solo se basan en los ya instalados en las imágenes a lanzar. En el lenguaje SDF han añadido un apartado de elasticidad en el que permiten definir reglas para controlar la evolución del tamaño del clúster. Para ello definen una serie de valores que denominan "Key Performance Indicators" (KPI) sobre los que se definirán las reglas de elasticidad. Está diseñado como un sistema de plugins para permitir añadir nuevos tipos de proveedores, aunque en la actualidad solo tienen soporta para OpenNebula.

Ventajas:

- Funcionalidad muy completa, incluyendo la gestión de la elasticidad.
- Sistema de plugins que permite la fácil extensión.

Inconvenientes:

- Su arquitectura es relativamente compleja y necesita de varios servicios funcionando, lo que hace complejo su despliegue.
- Tienen cosas pendientes de desarrollar: De momento solo existe soporte para OpenNebula. Aunque lo comentan en sus especificaciones no dan soporte multi-Cloud.
- El Lenguaje SDF, aunque permite definir con detalle las infraestructuras, al igual que el OVF, en el que está basado, es un lenguaje XML nada amigable para usuarios no avanzados, puesto que necesita crear grandes documentos con multitud de detalles de las MVs para poder definir una simple MV.
- Uso de imágenes base estáticas.

Apache **Whirr** [41] Permite el lanzamiento de clústeres en entornos Cloud (en concreto sobre EC2 y RackSpace). El usuario define el número de instancias de cada tipo que necesita, y le indica una serie de roles que debe tener. Está diseñado para el lanzamiento de clústeres con Hadoop [42], aunque permite ser extendido añadiendo nuevas clases Java que implementen la instalación de nuevos tipos de roles.

Ventajas:

- Muy sencillo de usar.
- Extensible.

Inconvenientes:

- Inicialmente diseñado para el lanzamiento de clústeres con Hadoop, aunque permite la extensión mediante un sistema de plugins, éste no es nada sencillo e implica programar una serie de clases en Java.
- Solo tiene soporte actualmente para EC2 y RackSpace y no es fácilmente extensible.
- Uso de imágenes base estáticas.

Wrangler [43] tiene una orientación mucho más específica para el lanzamiento de MVs de forma general. Permite que un usuario defina un documento XML con sus requerimientos. En dicho documento el usuario también debe indicar los scripts necesarios para configurar dicho nodo para adoptar el rol necesario dentro de la infraestructura creada. A diferencia de herramientas anteriores, estos scripts se almacenan en el nodo "coordinador", de manera que no hace falta prepararlos en la MV con anterioridad. La única preparación que necesitan las imágenes de las MVs es que deben tener instalado el agente "wrangler" configurado para conectarse al nodo coordinador.

Ventajas:

- Lenguaje de definición en XML.
- Scripts de configuración fuera de las MVs.

Inconvenientes:

- Uso de imágenes base estáticas.
- Las IMVs necesitan tener instalado el agente "wrangler".
- El usuario debe indicar detalles específicos del proveedor Cloud donde se lanzará la infraestructura, como puede ser el tipo de instancia a usar en EC2, etc. Aunque cada nodo puede estar en un proveedor distinto, es el usuario el que debe especificarlo.

Vagrant [44] fue inicialmente diseñado para el lanzamiento de MVs sobre la plataforma de virtualización VirtualBox, pero su diseño modular le permite añadir nuevos "proveedores" de forma sencilla. Actualmente da soporte a VirtualBox, VMWare Fusion y EC2. Permite el lanzamiento de conjuntos de MVs usando lo que denominan entornos "multi-maquina". Permite el soporte de herramientas como Chef, Puppet o Ansible o el uso del shell scripts para la instalación y configuración automática de software. Pero necesita que estas herramientas ya se encuentren instaladas y configuradas en las IMVs con anterioridad. Proporciona el concepto de "box" para encapsular las IMVs de cada proveedor distinto. En los casos de VirtualBox y VMWare este fichero contiene la propia imagen, en el caso de EC2 solo una referencia a la AMI a utilizar. Todos los ficheros "box" deben estar almacenados en el nodo gestor de Vagrant para poder ser utilizados en el sistema.

Ventajas:

- Lenguaje de definición con formato sencillo basado en Ruby de manera que con pares clave = valor se definen la mayoría de los atributos de las MVs.
- Permite el uso de diferentes herramientas de contextualización.
- Scripts de configuración fuera de las MVs.
- Extensibilidad sencilla.

Inconvenientes:

- Uso de imágenes base estáticas.
- Las IMVs necesitan tener instalado las herramientas de contextualización.
- Solo da soporte a EC2.

Tabla 3. Comparativa de sistemas de despliegue de infraestructuras virtuales

	Wrangler	Whirr	Claudia	Nimbus	AWS	Vagrant
Sistemas soportados	EC2, Eucalyptus, OpenNebula	EC2	OpenNebula	EC2, Nimbus	EC2	VirtualBox, VMWare, EC2
Contextualiza MVs	SI	SI	NO	SI	SI	SI
Lenguaje de Contextualización	NO	NO	NO	SI	SI	SI
MV preconfiguradas	SI	SI	SI	NO	NO[27]	SI
Lenguaje de despliegue sencillo	SI	SI	NO	SI	SI	SI
Sistema Extensible	SI	SI	SI	NO	NO	SI
Independiente del proveedor	NO	NO	NO	SI	NO	SI
Uso de catálogo de Imágenes de MVs.	NO	NO	NO	NO	NO	NO
Gestión de Elasticidad	SI	NO	SI	SI	SI[28]	SI

En la Tabla 3 se muestra a modo de resumen una comparativa de los sistemas de despliegue de infraestructuras virtuales mostrando sus principales características. Un problema generalizado en todos los proyecto analizados es la utilización de imágenes base estáticas. Esto provoca que el usuario deba conocer de antemano las características de las imágenes de MV existentes o crear una propia para cada caso, para saber lo idónea que puede ser para sus necesidades en cada caso. Esto no permite la reutilización de otras imágenes existentes (creadas por él o por otros usuarios) que pueden ser mejor para su caso de uso. Otro problema muy extendido, que también complica la reutilización de IMVs, es que las imágenes deben estar previamente preparadas con algún tipo de software especialmente configurado para poder ser utilizadas por la herramienta. Por ejemplo en el caso de EC2, este tipo de problemas hace que existan decenas de miles imágenes base. Además la mayoría de las herramientas analizadas no permiten el uso de herramientas de contextualización como Puppet, Chef o Ansible lo que hace más difícil las tares de instalación y configuración del software. Este problema es aún mayor en algunos sistemas que necesitan que dichos scripts estén pre-alojados en las MVs. Finalmente la

[27] Todas las AMI en EC2 debe estar preconfiguradas para su uso en EC2, pero el servicio CloudFormation no tiene ningún requisito extra.

[28] Lo proporciona usando otro servicio de Amazon denominado "Auto Scaling" http://aws.amazon.com/es/autoscaling/

extensibilidad del sistema de manera sencilla para permitir la adición de nuevos tipos de proveedores Cloud solo es tenida en cuenta de forma clara en el caso de Wrangler y Vagrant. Por tanto a continuación vamos a exponer los objetivos de esta tesis.

3 Objetivos

El objetivo general de esta tesis es crear una plataforma para facilitar a los investigadores el acceso a infraestructuras de cómputo personalizadas y configuradas de forma dinámica. Para ello se van diseñar un conjunto de componentes software para el despliegue y gestión elástica de infraestructuras sobre plataformas Cloud, para la ejecución de aplicaciones científicas.

Dicha plataforma debe contemplar todos los aspectos necesarios para la creación y gestión de las infraestructuras, por tanto debe contemplar los siguientes aspectos:

- **Expresión de requerimientos**: El investigador debe poder expresar sus requerimientos, hardware, software y de configuración, sobre los recursos que va a necesitar para la ejecución de su aplicación.
- **Creación de la infraestructura:** En base a los requerimientos definidos por el usuario el sistema debe crear la infraestructura del usuario. Para ello debe realizar todos los pasos necesarios que incluyen:
    - **Selección de Imágenes de máquina virtual**: Búsqueda y selección de la mejor imagen de máquina virtual disponible. Para ello será necesario tener un **sistema de catalogación** de dichas imágenes, junto con toda la información relevante de las mismas, incluyendo todo el software disponible.
    - **Selección de Clouds**: Elección del mejor proveedor Cloud, de todos los disponibles para el usuario. Hay que tener en cuenta que tanto la selección de la imagen como la del Cloud están relacionadas, por tanto se deberán elegir de forma combinada.
    - **Despliegue en el proveedor Cloud**: conexión con el proveedor Cloud elegido y lanzamiento efectivo. Esto se debe hacer de forma i**ndependiente del sistema** Cloud elegido creando un **sistema de modular** que permita la adición de nuevos sistemas Cloud.
    - **Contextualización:** Una vez seleccionada una imagen se deberá realizar un proceso denominado de contextualización para, en caso necesario, instalar software adicional requerido por el usuario, así como realizar todos los pasos necesarios para su configuración, tanto a nivel individual, como a nivel de clúster de máquinas virtuales.
- **Gestión de la elasticidad:** El sistema debe permitir que el usuario modifique el número de recursos de cómputo (**elasticidad horizontal**) así como las características de los mismos (**elasticidad vertical**). Tomando las

medidas necesarias para que el sistema se reconfigure correctamente para que cumpla con los requisitos del usuario.

- **Interfaces:** La plataforma debe proporcionar interfaces tanto a nivel de usuario, mediante aplicaciones de comandos o interfaces gráficas, como a nivel programático para que capas de mayor nivel puedan hacer uso de la funcionalidad mediante un API.

Todos estos objetivos deben cumplir con las siguientes características:

- Facilidad de uso: su uso debe ser sencillo para permitir que usuarios no avanzados en informática puedan utilizarlo para el despliegue de sus infraestructuras.
- Independencia de el/los proveedores Cloud disponibles, de manera que sea transparente al usuario la ubicación real de sus MVs.
- Capacidad de reutilización de imágenes de MV creadas con anterioridad: Esta característica tienes dos funciones: primero permitir que se puedan obtener imágenes de MVs con aplicaciones cuyo proceso de instalación es complejo o imposible de automatizar, aumentando el abanico de aplicaciones disponibles y, segundo, hacer más rápido el despliegue de las MVs para aquellas aplicaciones que, aunque puedan ser fáciles de instalar de forma automática, requieran mucho tiempo.
- Soporte de múltiples proveedores de IaaS Cloud: acceder al mayor número de plataformas Cloud de tipo público (EC2, Google, Azure) y a los principales CMPs (OpenNebula, OpenStack) y permitir de forma sencilla su extensibilidad en el futuro.
- Uso de estándares: para permitir que en el futuro, los proveedores que usen dichos estándares puedan ser utilizados.

Para conseguir todos estos objetivos se deberán realizar las siguientes tareas:

- Especificar un lenguaje de definición de requerimientos sobre los recursos virtuales. Debe ser un lenguaje sencillo de comprender para usuarios no avanzados y que solo necesite aquella información relevante para el investigador y su aplicación sin introducir elementos extra.
- Diseñar un servicio de catálogo de imágenes de máquinas virtuales que nos permita indexar y catalogar las imágenes junto con toda la información relacionada con dicha imagen.
- Diseñar e implementar un conjunto de servicios que proporcione, mediante algún tipo de API usando métodos estándar de conexión, (XML-RPC, WS, REST, etc.), un conjunto de funciones razonablemente sencillo que contemple toda la funcionalidad requerida. De esta forma también

permitiría a otras capas de mayor nivel de abstracción poder desarrollar componentes de mayor nivel para construir servicios más complejos.

- Implementar clientes, tanto de línea de comandos como con interfaz gráfica (por ejemplo de tipo aplicación web), para que el usuario final pueda acceder a dicha funcionalidad.

4 Arquitectura

La arquitectura de la plataforma ha sido diseñada teniendo en cuenta los objetivos marcados en la tesis. La Figura 1 muestra un diagrama con la arquitectura donde se muestran todos los componentes que forman la plataforma de gestión de infraestructuras virtuales. El componente central es el denominado *Infrastructure Manager* (IM), que se encarga de orquestar el lanzamiento y gestión de las infraestructuras virtuales. El IM expone su funcionalidad tanto a las herramientas cliente como a otras capas software de alto nivel a través de sus APIs. Tanto para definir los requisitos necesarios para desplegar una infraestructura, como para obtener la información de las mismas de forma sencilla, se ha definido un lenguaje llamado *Resource and Application Description Language* (RADL). Dicho lenguaje permite definir las características hardware (CPUs, memoria, etc.), software (aplicaciones instaladas) y de configuración. Para gestionar un catálogo de imágenes de máquinas virtuales se ha desarrollado el componente *Virtual Machine Image Repository & Catalog* (VMRC) que permite indexar y almacenar IMVs junto con un conjunto de metadatos que permitan describir sus características tanto hardware como software. Finalmente las herramientas cliente permiten el acceso a la funcionalidad del sistema a usuarios finales. A continuación vamos a describir con detalle cada uno de ellos.

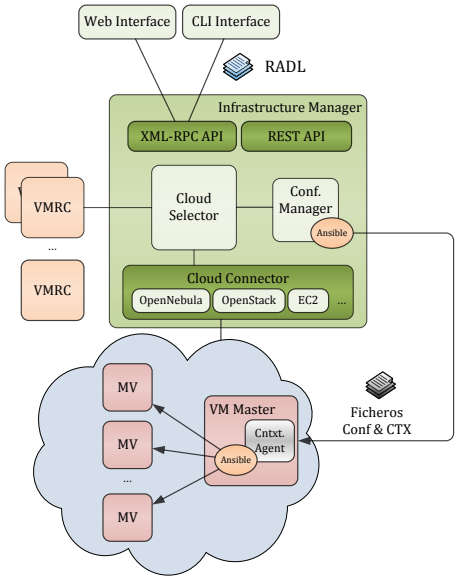

Figura 1. Arquitectura del gestor de la infraestructura.

4.1 El Lenguaje de Descripción de Infraestructuras RADL

Uno de los objetivos de la tesis es permitir al usuario poder expresar sus requerimientos, hardware, software y de configuración, sobre los recursos que va a necesitar para la ejecución de su aplicación. Para ello se ha diseñado, basándose en los lenguajes analizados en el estado del arte, el Lenguaje de Descripción de Aplicaciones y Recursos para proporcionar las siguientes características:

- Debe permitir especificar los requerimientos de las aplicaciones sobre las MVs donde se ejecutarán.
- Debe abordar los requisitos hardware (número procesadores, RAM, procesador, etc.) así como requisitos software (paquetes, bases de datos, librerías, etc. y todos los aspectos de configuración necesarios de las MVs.
- El lenguaje y la terminología debe ser común al utilizado por el repositorio de MVs.
- Debe ser sencillo de entender por usuarios no avanzados.

El lenguaje tiene una doble función. Por un lado, sirve como lenguaje de especificación de requisitos de las aplicaciones sobre los recursos cloud necesarios para su correcta ejecución. En este caso la aplicación que interprete el RADL será el responsable de instanciarlo para generar una definición de una MV, bien en un formato tipo OVF o en un lenguaje propio de un CMP o un hipervisor concreto. Por otro lado, se utiliza también para mostrar la información de las MVs en funcionamiento. El uso de un lenguaje común tanto para proveedores como para consumidores facilita el proceso de búsqueda de aquellas MVs que satisfacen los requisitos de las aplicaciones.

Cabe mencionar que los requisitos de las aplicaciones comprenden una serie de características, que pueden ser divididas en dos categorías:

- Características asociadas a la imagen de una MV. Estas características serán utilizadas para interrogar al repositorio sobre las imágenes que cumplen ciertos requisitos. Por ejemplo, el sistema operativo o las aplicaciones
- Características que dependen del despliegue de las MVs. Por ejemplo, la cantidad de memoria, o el número de CPUs. Estas características serán utilizadas para desplegar de forma adecuada las MVs.

El RADL se ha diseñado partiendo de trabajos y estándares existentes, entre los que cabe destacar: OVF, OCCI, o el lenguaje *classads* de Condor [45]. En la medida de lo posible, se ha tratado de desarrollar un lenguaje versátil, fácilmente extensible y adaptable.

4.1.1 Forma general de un documento RADL

Un documento RADL consiste en la declaración de los diferentes elementos que compondrán la infraestructura, junto con las características que deben presentar. La Figura 2 muestra una visión general de un documento RADL.

```
# Definición de cero o más redes.
# Una instrucción "network" por cada red que a definir
network <id> [(<características>)]

# Definición de los sistemas o MVs.
# Una instrucción system por cada tipo de MV a definir.
system <id> [(<características>)]

# Definición de las recetas de configuración ansible
configure <id> (<recetas ansible>)

# Despliegue de un número de instancias de las MVs
deploy <id> <num_instancias> [<cloud_id>]

# Instrucciones de contextualización
contextualize [max_time] (
    system <system_id> configure <configure_id> [step <num>]
    …
)
```

Figura 2. Esquema documento RADL.

Una descripción de la infraestructura usando RADL incluirá cinco tipos de elementos:

- Características del entorno: dispositivos, servicios, etc. no proporcionadas por las MVs, y cuya existencia no depende del despliegue de las MVs. Las MVs deben poder interactuar con dicho entorno. Algunos ejemplos de características del entorno son redes de interconexión, servicios de dominio de Windows, Storage Area Networks, etc. Actualmente se considera únicamente la característica de red, que debe ser entendida como una red de área local a la que pueden conectarse las MVs. Una red se define mediante la palabra clave "network".
- Definición de uno o más tipos de nodos. Se usa la palabra clave "system" para definir un tipo de nodo.
- Apartado de configuración, usando la palabra clave "configure", donde se especificarán, usando el lenguaje de Ansible [29], las recetas necesarias para la configuración de las MVs. Más adelante se explicará cómo este apartado se integra con el IM para facilitar la configuración.
- Instrucciones de despliegue: Las instrucciones de despliegue indican cuantas máquinas de cada uno de los tipos definidos anteriormente, van a

ser desplegadas de forma efectiva. De forma opcional se puede indicar una etiqueta que identifique al proveedor Cloud que se quiere usar para lanzar el conjunto de MVs indicadas.

- Instrucciones de contextualización: Finalmente el lenguaje se completa con una serie de instrucciones de contextualización. Estas instrucciones son opcionales y permiten especificar, a usuarios avanzados, cómo se van a aplicar las recetas indicadas en los apartados de configuración en el conjunto de nodos desplegados.

Las características que debe cumplir un sistema virtual se declaran mediante una expresión de acuerdo con la gramática que aparece en la Figura 3, que usa notación de tipo BNF (Backus-Naur Form).

```
<expression> := <expression> <bool_connector> <expression>
      | <attribute> <operator> <value>
      | soft <punctuation> ( <expression> )
      | <attribute> <contains> ( <tuple> )
<tuple> := field <operator> <value> | <tuple> <bool_connector>
<tuple>
<punctuation> := integer
<attribute> := string
<bool_connector> := and | or
<operator> := = | > | < | >= | <=
<value> := ' string ' | number [ <qualifier> ]
<qualifier> := B | M | G | K
```

Figura 3. Gramatica BNF para definir las características del un sistema virtual.

Algunos de los atributos de una MV son multivaluados (por ejemplo los discos o las aplicaciones). En algunos casos el orden de los elementos de un atributo es relevante dado que afecta al comportamiento del sistema. Este ocurre por ejemplo con los discos de una MV. En tal caso RADL sigue una sintaxis LDAP para la numeración de los elementos de un atributo tipo array (por ejemplo, attribute.0.field, attribute.1.field, etc.). En aquellos casos en que los componentes de un atributo no siguen ningún orden en particular, se propone una sintaxis alternativa, introduciendo la palabra clave "contains" para reflejar el comportamiento tipo colección. Un ejemplo de un atributo no ordenable es el conjunto de aplicaciones que están instaladas en una MV.

La palabra clave "soft" sirve para indicar una característica que es interesante que se cumpla, pero no es esencial. La palabra "soft" va acompañada de una puntuación (número entero) que será evaluada para determinar la valoración final de una instancia de MV, expresada en una escala definida por el usuario.

Algunas propiedades sólo permiten el operador de igualdad como comparación. En particular, las propiedades cuyo valor es un *string*, excepto os.version y application.version.

4.1.2 Características a considerar en una infraestructura de ejecución

A continuación se enumeran las propiedades consideradas para definir la infraestructura de ejecución de una aplicación. Cabe distinguir entre las propiedades del entorno (instrucción "network") y las propiedades de las MVs (instrucción "system").

La definición de las redes de forma separada de las MVs es necesaria dado que en general habrá distintas MVs conectadas a una misma red. También se hace de esta manera en OVF, OCCI u OpenNebula, por ejemplo.

a. Características del entorno

Como se ha comentado previamente, en la versión actual del lenguaje se considera únicamente las características de las redes, entendidas como redes de área local a las que pueden conectarse las MVs. Los atributos de una red son:

- outbound (string): `yes` indica que la red definida tendrá una IP pública que podrá ser accedida desde el exterior, mientras que `no` indicará que se trata de una red privada que no garantiza la conectividad desde el exterior. En caso de no indicarse este parámetro, el valor por defecto será `no`.
- outports (string): Lista separada por comas de los pares "puerto externo" – "puerto interno" que deben ser accesibles desde el exterior (a través de una IP pública) de una red. En el caso de que solo se indique un puerto se supondrá el mismo para los dos casos. En caso de no indicarse este parámetro el valor por defecto dependerá de si se ha indicado el parámetro outbound. En caso de indicar el parámetro outbound a `yes` el valor por defecto de outports será permitir el acceso a todos los puertos. En caso de que el valor de outbound sea `no` el valor por defecto será no permitir el acceso a ningún puerto.

b. Características de una MV

Los distintos atributos de las MVs que pueden ser indicados en el RADL, basándose en distintas ObjectClasses de LDAP, son representados por las siguientes palabras clave.

- image_type (string). Tipo de fichero de la imagen de disco de la MV. Ej: `vmdk`, `qcow`, `qcow2`, `raw`…
- virtual_system_type (string). El valor del atributo final de esta clase es el hipervisor sobre el que desplegar la MV. Ej: (extraído del estándar de OVF) `vmx-4` para la cuarta generación de hardware virtual del VMWare, `xen-3` para la tercera generación de hardware virtual de Xen. Corresponde al elemento *vssd:VirtualSystemType* de la sección VirtualHardwareSection de OVF.

- price (valor flotante positivo): El valor del precio por hora de la instancia de la MV. Este valor solo se aplica en el caso de Clouds de tipo público.
- cpu: representa las características de las CPUs virtuales. Los atributos considerados son:
    - count (entero positivo). Número de CPUs/cores virtuales.
    - arch (string). Arquitectura. Puede ser `i686` o `x86_64` En OVF se especificaría mediante la propiedad *VirtualQuantity* y *AllocationUnits* de un recurso de tipo procesador.
    - performance (valor flotante positivo, seguido de un string que indica la unidad de medida). Este campo indica una medida de las prestaciones que pretendemos que tenga el procesador solicitado. En este caso la unidad de medida es difícil de estandarizar, aunque en OVF se utilizan la frecuencia de reloj del procesador mediante el campo *AllocationUnits* (en unidades de 10^9 Hercios). Realmente este valor no es siempre indicativo de la potencia del procesador y puede no ser válido en todos los entornos virtuales. Por tanto para dar mayor flexibilidad el campo unidad es libre y será responsabilidad del servicio que despliegue el RADL interpretarlo correctamente. Posibles valores serán: ECU, GCEU, etc.
- memory: se refiere a la memoria RAM de la MV. Los atributos considerados son:
    - size (entero positivo, seguido de un carácter que indica unidad: `B` (byte), `K` (kilobyte), `M` (megabyte), `G` (gigabyte)). Cantidad de memoria de la MV. En OVF se especificaría mediante las propiedades *VirtualQuantity* y *AllocationUnits* de un recurso de tipo memoria.
- disk: es un array de discos cuyos elementos están ordenados para reflejar el orden en la MV (disk.0, disk.1, etc). El disco 0 es especial, dado que es el disco de arranque del sistema. OVF considera la definición de discos por separado de las MVs que los utilizan, de manera que es posible que distintas MVs compartan un mismo disco. En general no es posible montar un mismo disco de forma simultánea en varias MVs, y además complica la definición de las MVs, por lo que se consideró mejor incluirlo dentro de la definición de la MV. Las atributos de un disco que se consideran son:
    - image.url (string): url del fichero con la imagen del disco. La URL deberá cumplir la convención de URIs descrito en la sección 4.3.2.
    - image.name (string): Nombre con el que la imagen del disco está registrada en el VMRC.

- type (string): tipo de disco. Los posibles valores son: `swap`, `iso`, `filesystem`.
- device (string): establece el nombre del dispositivo en caso de los discos vacíos.
- size (entero positivo, seguido de letra que indica unidad): tamaño del disco. Útil para especificar el tamaño de un disco vacío que se creará para la MV. Sólo permitido si el disco no es el cero y no se ha especificado la propiedad 'image'. En OVF, esta propiedad correspondería a los atributos *ovf:capacity* y *AllocationUnits* del elemento *Disk* de la sección *DiskSection*.
- free_size: espacio libre del disco. Sólo permitido para el disco cero. Aunque en OVF no existe una propiedad para expresar el espacio libre de un disco, sí que se puede expresar la capacidad total del disco (atributo *ovf:capacity*) y el tamaño realmente usado (*ovf:populatedSize*).
- os: sistema operativo del disco (ver punto "Sistema Operativo" más adelante).
- applications: aplicaciones instaladas en el disco (ver punto "applications" más adelante).

- net_interface: es un array de interfaces de red, por lo que se debe cualificar mediante un índice para cada posible interfaz (net_interface.0, net_interface.1, etc.). Los atributos considerados para una interfaz de red son:
  - connection (string). Nombre de una red lógica a la que debe conectarse la interfaz. Debe corresponder a una red definida mediante la sentencia "network". Corresponde al elemento *rasd:Connection* de OVF.
  - ip: dirección IP de la interfaz. Este valor puede tener dos funciones:
    - Poner una IP fija a la interfaz (dependerá del hipervisor el implementarla)
    - Devolver el valor de la IP de una MV ya creada.
  - dns_name: Nombre a asignar a la IP de dicho interfaz. En el caso de la interfaz 0 será el nombre asignado al host. Es muy importante tener en cuenta que esto solo será visible dentro del ámbito de la infraestructura creada. Este nombre dns debe ser utilizado con cuidado en el caso de que se vayan a hacer

despliegues múltiples de un mismo tipo de nodo, pues todos tendrán asignados el mismo nombre, con los problemas que eso ocasionará. Para evitar estos problemas se ha definido un valor de sustitución `#N#` que en caso de ser utilizado en el nombre dns del nodo será sustituido por el número de la instancia que se haya lanzado. De tal manera que en caso de usar un nombre dns como este `nodo-#N#` para lanzar 2 nodos, el sistema le asignará los nombres nodo-0 y nodo-1 respectivamente.

- os: se considera el sistema operativo como una propiedad de un disco. Por tanto, se usa la notación: disk.<num>.os.<atributo>, donde <num> será un número de disco, que deberá ser el cero. Los atributos de un sistema operativo son:
    - name (string). Nombre del Sistema Operativo (ej. "linux", "windows", "mac os x")
    - flavour (string). Ej: "ubuntu", "windows xp", "windows 7"
    - version (string). Debe ser una cadena formada por números enteros positivos separados por puntos. Ej: "10.04", "7.1.2".
    - credentials. Indica la forma de acceder a la máquina, que puede ser bien mediante usuario y contraseña, o bien mediante clave pública. A su vez, credentials está compuesto por los siguientes atributos:
        - type (string). El valor puede ser `user` (usuario y contraseña) o `public_key`.
        - username (string). Nombre del usuario.
        - password (string). La contraseña de acceso.
        - public_key (string). Clave pública para acceder a la máquina.
        - private_key (string). Clave privada para acceder a la máquina.
- applications: Aplicaciones instaladas en un disco. Se usa la notación: disk.<num>.applications contains (<expression>), donde <expression> es una condición expresada sobre los campos de una aplicación, que son:
    - name (string). Nombre de la aplicación, que debe ser identificativo (ej: "org.apache.tomcat"). Al igual que en OVF, se propone usar notación inversa de dominio para especificar aplicaciones, como "org.apache.tomcat" o "com.vmware.tools".

- version (string). Debe ser una cadena formada por números enteros positivos separados por puntos. Ej: "10.04", "7.1.2".
- preinstalled (string). `yes` indica que la aplicación se encuentra instalada en la MV guardada en el repositorio. "no" indica que la aplicación se podrá instala al desplegar la MV. El valor por defecto es `no`.
- path: siguiendo con el esquema definido en el VMRC también vamos a añadir aquí el path de la aplicación. Está pensado para modelar una MV ya existente y no tanto para el lenguaje de consultas.

En OVF cada aplicación instalada en una MV puede declararse mediante una sección *ProductSection*, con una propiedad *ovf:class* (corresponde a la propiedad "nombre" definida aquí) y un elemento *Version*.

c. Instrucción de despliegue

La instrucción de despliegue indica de forma efectiva el número de MVs que se van a lanzar. Esta instrucción no es obligatoria lo que permite crear documentos RADL puramente declarativos, donde no se lanza ninguna MV, o usar un documento RADL solo con instrucciones "deploy" haciendo referencia a tipos de nodos previamente declarados en un documento anterior. Esto dota de mayor flexibilidad al lenguaje facilitando las tareas de los servicios de despliegue de MVs.

d. Apartado de configuración

El apartado de configuración permite que el usuario pueda indicar tareas de configuración a su infraestructura de manera que pueda conseguir que una vez desplegada tenga todos los componentes, no solo instalados, si no también configurados y preparados para funcionar. Para ello se ha elegido el lenguaje YAML utilizado por la herramienta Ansible [29], que permite abstraer del sistema operativo que haya por debajo con una amplia funcionalidad[29]. Por ejemplo en la Figura 4 se muestra un ejemplo de cómo instalar y configurar el servidor web de apache para el conjunto de nodos del grupo "webservers". En concreto esta receta primero instala el paquete "httpd". Después se genera el fichero de configuración "httpd.conf" usando un template "httpd.j2", donde además se indica que en caso de que dicho fichero sea modificado deberá notificar al servicio apache para reiniciarse (definido en la sección handlers). Finalmente se asegura de que el servicio "httpd" está arrancado correctamente. Las tareas indicadas en la sección "handlers" solo se ejecutarán si una tarea anterior se lo notifica.

[29] http://www.ansibleworks.com/docs/playbooks.html

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: pkg=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running
    service: name=httpd state=started
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

Figura 4. Ejemplo de receta YAML de Ansible.

Una de las características interesantes de Ansible es que las tareas son idempotentes, es decir que solo se realizan los cambios que son necesarios. Por ejemplo en el caso de que se indique la instalación de una aplicación Ansible comprueba si ya está instalada para no volver a instalarla. Esto permite que las ejecuciones repetidas de una receta sean más seguras y fiables puesto que solo realizará los cambios que sean estrictamente necesarios.

Dentro del RADL puede aparecer tantas secciones "configure" como desee el usuario. De forma general solo se aplicarán directamente a la infraestructura, aquellos que tengan el mismo nombre que algún "system" definido. Los otros se podrán utilizar para definir partes comunes a varios tipos de nodos y luego utilizando la instrucción "include" del lenguaje de Ansible reutilizar dicho código (en el ejemplo se mostrará con más claridad). En el apartado de contextualización se puede definir con más detalle cómo "lanzar" los apartados de configuración.

El propio Ansible proporciona una serie de variables, denominadas "facts"[30] que proporcionan información sobre las maquinas configuradas en el sistema. No obstante, para dar mayor funcionalidad al lenguaje, el IM pone a disposición del usuario un conjunto de variables que serán accesibles desde el lenguaje YAML de las recetas Ansible, que pueden ser muy útiles a la hora de preparar recetas de configuración (como veremos en el apartado de casos de uso). Las variables son las siguientes:

[30] http://www.ansibleworks.com/docs/playbooks_variables.html#information-discovered-from-systems-facts

- IM_NODE_HOSTNAME / IM_ MASTER _HOSTNAME: Nombre del nodo actual / master.
- IM_NODE_FQDN / IM_MASTER_FQDN: Nombre completo del nodo actual / master.
- IM_NODE_DOMAIN / IM_MASTER_DOMAIN: Dominio del nodo actual / master.
- IM_NODE_NUM / IM_MASTER_NUM: Numero de la instancia del nodo actual / master.
- IM_NODE_VMID: Identificador de la instancia en el proveedor Cloud lanzado.
- IM_NODE_ANSIBLE_IP: IP del nodo actual. Es la IP que ha usado Ansible para configurar el nodo.

También se añaden una serie de variables que proporcionan la información diponible en el VMRC sobre las aplicaciones preinstaladas en las IMVs de la siguiente manera:

- IM_[nombre aplicación]_VERSION: Versión de la aplicación [nombre aplicación].
- IM_[nombre aplicación]_PATH: Directorio de instalación de la aplicación [nombre aplicación].

e. Apartado de contextualización

En este apartado, que es opcional, los usuarios con necesidades especiales de contextualización podrán indicar qué recetas indicadas en los apartados de configuración van a ser aplicadas, en qué orden, y en los distintos tipos de nodos a desplegar.

Cada línea de la forma `system <system_id> configure <configure_id> [step < num>]` especifica qué apartado "configure" debe ser aplicado en el conjunto de nodos del tipo "system_id". Además el campo opcional "step" permite especificar el conjunto de apartados configure que pueden ser ejecutados en paralelo. Todos aquellos que tengan el mismo valor de "step" serán ejecutados de forma paralela, de manera que se puedan solapar las tareas de contextualización. La aplicación de los apartados se realizará en el mismo orden que aparezca especificado, independientemente del valor de su "step". Adicionalmente se podrá indicar un número entero después de la palabra clave "contextualize" para indicar el tiempo máximo estimado (en segundos) para la fase de contextualización. Este valor servirá para evitar que el proceso de contextualización se quede atascado de forma indefinida, en caso de que alguno de los pasos produzca algún problema de bloqueo de procesos.

En la mayoría de los casos no será necesario especificar este apartado. En este caso el IM aplicará para cada MV aquel apartado de configuración que coincida con el nombre del "system", que serán todos ejecutados en paralelo para obtener las mejores prestaciones. Es decir sería equivalente a incluir un apartado contextualización como se muestra en la Figura 5.

```
system nodoA (
…
)

system nodoB (
…
)

configure nodoA (
…
)

configure nodoB (
…
)
…

contextualize (
    system nodoA configure nodoA step 1
    system nodoB configure nodoB step 1
)
```

Figura 5. Ejemplo de apartado de contextualización.

Pero el apartado "contextualize" puede ser muy útil en caso de recetas que deban ser ejecutadas en un orden concreto de manera que primero se apliquen un conjunto de recetas en todos los nodos del sistema y luego se apliquen las siguientes. Esto puede ocurrir en recetas que configuran determinados elementos en todos los nodos que proporcionan cierta información que se necesita para la correcta configuración de otra receta. También permite ejecutar en paralelo un conjunto de recetas para disminuir el tiempo de contextualización. Por ejemplo en el caso de configuración de una herramienta de tipo cliente-servidor, en una primera etapa, se podrían ejecutar las recetas de configuración tanto del servidor como del cliente (para reducir el tiempo de despliegue), que pueden tener una duración relativamente larga. Después en una segunda etapa se lanzaría el programa cliente, ya que éste necesita que el servidor esté arrancado para conectar.

El ejemplo de la Figura 6 muestra el código RADL para el caso comentado. Primero se ejecutaría la receta "conf_server" en los nodos de tipo "nodoA" y la receta "conf_client" en los de tipo "nodoB" en paralelo y una vez finalizadas de nuevo la "launch_client" en los de tipo "nodoB", donde ya se puede asegurar que el servidor ha sido totalmente configurado y estará esperando las peticiones de los

clientes. Además en este caso se ha indicado que el tiempo máximo estimado para la contextualización será de 20 minutos. Esto implica que si dicho proceso tarda más de los 20 minutos indicados se finalizará el proceso de contextualización y se considerará como fallido.

En el caso de que no se indique el valor de "step" en alguna de las líneas del apartado se supondrá que dicha tarea de configuración se ejecutará de forma exclusiva sin paralelizarse con ninguna. En los casos de uso se verán ejemplos de la utilidad de este apartado.

```
contextualize 1200 (
    system nodoA configure conf_server step 1
    system nodoB configure conf_client step 1
    system nodoB configure launch_client step 2
)
```

Figura 6. Ejemplo de apartado "contextualize".

4.1.3 Ejemplos

a. Ejemplos sencillos:

En la Figura 7 se muestra un ejemplo sencillo donde se demuestra que con muy pocas líneas es fácil obtener un conjunto de máquinas con las características que desee un usuario.

Se pide una MV que tenga una cantidad de memoria superior a 1024 MBytes de RAM. Con respecto al software, se requiere que un servidor Tomcat esté instalado (sin especificar ninguna versión concreta), indicando que se deben lanzar dos instancias de dicha MV. De esta manera un usuario solo indica lo básico para sus necesidades y deja que el sistema que interprete el RADL complete las características de la MV que finalmente desplegará.

```
system nodeS (
memory.size>=1024M and
disk.0.applications contains (name='org.apache.tomcat')
)
deploy nodeS 2
```

Figura 7. Ejemplo de RADL minimo.

La Figura 8 muestra un segundo ejemplo sencillo donde ya se especifican más detalles concretos de las MVs a desplegar.

Se pide una MV que tenga una CPU y una cantidad de memoria superior a 1024 MBytes de RAM. Con respecto al software, precisa una plataforma Linux Ubuntu. La aplicación requiere que Octave esté instalado (sin especificar ninguna versión

concreta). Además requiere que tenga una conexión de red con salida al exterior de la red local.

```
network red (outbound = 'yes')

system nodoA (
cpu.count=1 and cpu.arch='i686' and
memory.size>=1024M and
net_interface.0.connection='red' and

disk.0.os.name='linux' and
disk.0.os.flavour='ubuntu' and
disk.0.applications contains (name='org.gnu.octave')
)

deploy nodoA 1
```

Figura 8. Ejemplo RADL sencillo.

b. Ejemplo complejo:

En el caso del ejemplo de la Figura 9 se va a mostrar toda la potencia del lenguaje para expresar unos requerimientos más complejos que en el caso anterior.

Se piden 10 MVs a desplegar que tengan al menos dos CPUs y una cantidad de memoria superior a 1024 MBytes de RAM. Adicionalmente, si el recurso dispone de más de 2048 MBytes de RAM entonces hay que ponderarlo de forma positiva. La aplicación requiere un disco con más de 500 MBytes libres. Con respecto al software, precisa una plataforma Linux y si puede ser un Ubuntu Linux superior a la versión 9.10, entonces se puntúa favorablemente. La aplicación requiere que Octave esté instalado (sin especificar ninguna versión concreta) y también Java 1.6 y Tomcat 5.0. Se requiere también Ant versión superior a 1.5.0. Ant puede ser instalado en el momento del despliegue, aunque se puntúa favorablemente el que la máquina ya lo tenga instalado. Se requiere que los recursos computacionales estén conectados entre sí por red. Además el atributo "dns_name" indica el nombre a asignar a la IP de dicho interfaz. En el caso de la interfaz 0 será el nombre asignado al host. Es muy importante tener en cuenta que esto solo será visible dentro del ámbito de la infraestructura creada. En este caso se ha utilizado el valor de sustitución "#N#", de tal manera que en el caso de lanzar 2 nodos, el sistema le asignará los nombres nodo-0 y nodo-1 respectivamente.

Además en este ejemplo aparece el apartado "configure" que permite usar el lenguaje Ansible para definir las tareas de configuración. En caso del ejemplo se contemplan dos partes, la primera que añade un usuario al sistema (indicando el valor encriptado de la misma) y una segunda para instalar los paquetes cliente y servidor de LRMS Torque y configurarlo de manera que el servidor sea el nodo master.

```
network red (outbound = 'yes')

system nodoB (
cpu.count>=2 and cpu.arch='i686' and
memory.size>=1024M and
soft 10 (memory.size>=2048M) and
net_interface.0.connection='red' and
net_interface.0.dns_name='node-#N#' and

disk.0.free_size>500M and
disk.0.os.name='linux' and
disk.0.os.flavour='ubuntu' and
disk.0.os.version>='9.10' and
disk.0.applications contains (name='org.gnu.octave') and
disk.0.applications contains (name='org.openjdk.java' and version='1.6') and
disk.0.applications contains (name='org.apache.tomcat' and version='5.0') and
disk.0.applications contains (name='org.apache.ant' and version>'1.5.0' and
soft 50 (preinstalled="yes"))
)

configure common (
@begin
---
  - tasks:
    - user: name=user1 password=a7ae2ax1k0a
@end
)

configure nodeB (
@begin
---
  - tasks:
    - include: common.yml
    - yum: pkg=${item} state=installed
      with_items:
      - torque-client
      - torque-server
    - copy: content=$IM_MASTER_FQDN dest=/etc/torque/server_name
@end
)

deploy nodoB 10
```

Figura 9. Ejemplo RADL complejo.

En el apartado de Casos de Uso se podrán ver ejemplos más complejos de documentos RADL donde se utilizan todas las características del lenguaje.

4.2 Infrastructure Manager

El Infrastructure Manager (IM) es el componente central de la arquitectura y por lo tanto es el principal aporte de la tesis. El IM tiene como objetivo principal proporcionar una serie de funciones para el despliegue efectivo de toda la infraestructura inicial necesaria para el lanzamiento de una aplicación en un proveedor Cloud, entendiendo como infraestructura un conjunto de máquinas virtuales conectadas entre sí por medio de alguna red de interconexión, para la ejecución de algún tipo de aplicación. El IM también proporciona funciones para

más adelante ir modificándola bajo demanda y de forma elástica tras el despliegue de la misma. El usuario de la aplicación, o una capa superior de programación serán los que deberán elegir cuando y cómo realizar la modificación de la infraestructura usando esta API para obtener sus objetivos. El IM expone una API con un conjunto de funciones sencillo que permite la creación, consulta del estado, adición y/o eliminación de MVs en el despliegue. La creación se va a basar en el lenguaje RADL para la definición de los requisitos de todas y cada una de la MVs necesarias para crear el conjunto de recursos de cómputo. El IM ha sido desarrollado íntegramente en Python[31] dada la gran cantidad de librerías disponibles para dicho lenguaje que facilitan su integración con otros componentes.

En la Figura 1 se muestra el esquema del gestor de infraestructura. En la parte superior se muestran las APIs del gestor, que exponen la funcionalidad al exterior. En la parte inferior se encuentra la capa denominada "Cloud Connector" que se encargará de proporcionar un acceso homogéneo a los diferentes middleware Cloud. En el centro del IM se encontrarán el "Cloud Selector" y el "Configuration Manager". El primero se encargará de conectar, en caso necesario, con el/los servicios VMRC para obtener el listado de IMVs que mejor se adapten a las necesidades expresadas en el RADL por el usuario y combinarlo con el listado de proveedores Cloud disponibles para elegir la mejor opción. El segundo se encarga de la gestión de la fase de configuración permitiendo la ejecución de una serie de "recetas" que permiten configurar correctamente la infraestructura para permitir la ejecución de la aplicación deseada usando la herramienta Ansible. A continuación explicaremos con más detalle cada uno de los componentes.

4.2.1 API

Para exponer la funcionalidad del sistema hacia otros componentes de alto nivel, o a las propias herramientas de cliente, el IM proporciona dos APIs. La primera es en forma de servicio XML-RPC, que podemos denominar como API nativa. En segundo lugar proporciona un API tipo REST. Una de las ventajas de las APIs de tipo REST es que permiten la creación de clientes de forma muy sencilla e independiente del lenguaje de programación. En ambos casos hay versiones con encriptación para securizar mediante Secure Sockets Layer (SSL) las conexiones con el IM.

a. Funciones del API

A continuación se muestran las 10 funciones que proporciona el API del gestor de la infraestructura (se muestran todos los detalles de las mismas en el Anexo II):

[31] http://www.python.org/

- **GetInfrastructureList:** Recibe los datos de autenticación y consulta el listado de todas las infraestructuras que han sido creadas por el usuario indicado.
- **CreateInfrastructure**: Recibe un documento RADL para crear y configurar de forma adecuada todas las MVs especificadas.
- **GetInfrastructureInfo**: Proporciona una lista con todas las MVs que pertenecen a la infraestructura en el momento de la llamada.
- **GetVMInfo**: Proporciona la información sobre la MV indicada como parámetro en formato RADL.
- **AlterVM**: Cambia las propiedades de la MV, para proporcionar funcionalidades para la gestión de la elasticidad vertical.
- **DestroyInfrastructure**: Libera todos los recursos relacionados con la infraestructura especificada, eliminando todas las MVs que la componen.
- **AddResource**: Añade los recursos indicados en el documento RADL al despliegue actual y reconfigura el sistema completo. Permite la gestión de la elasticidad horizontal. El nuevo documento RADL especificado se "fusionará" con el originalmente enviado en el momento de la creación de la infraestructura. Por tanto el nuevo documento podrá contener solo instrucciones de tipo "deploy" haciendo referencia a algunos de los "system" ya definidos. En caso de que se definan nuevos "system" estos serán añadidos. En el caso de que se utilice algún "system" o "network" previamente definido, la nueva especificación será ignorada, conservando los objetos previos. En caso de querer modificar algún "system" se deberá utilizar la función AlterVM.
- **RemoveResource**: Elimina las MVs especificadas del despliegue actual y reconfigura el sistema completo. Permite la gestión de la elasticidad horizontal.
- **StopInfrastructure**: Para (pero no destruye) todas las MV de la infraestructura. De tal manera que se mantiene toda la información del disco de las MVs, pero no se producirá gasto (en términos de horas de CPU) por dichas MVs.
- **StartInfrastructure**: Arranca de nuevo todas las MVs de la infraestructura, previamente paradas con la función StoptInfrastructure. Estas dos funciones permiten que infraestructuras que durante su uso han almacenado información en disco, que puede ser necesaria en futuros usos de la misma, puedan ser paradas, y por tanto dejar de producir gasto por uso de CPU, durante un periodo de tiempo para más adelante volver a iniciarlas sin pérdida de dicha información.

- **Reconfigure**: Permite cambiar las definiciones de las configuraciones de las MVs en funcionamiento, para pasar una fase de reconfiguración para darles una nueva funcionalidad a las MVs en ejecución. Esto permite tener una infraestructura en ejecución y una vez acabada la tarea para la que estaban configuradas, reconfigurarlas para ejecutar una tarea diferente, sin tener la necesidad de destruir la infraestructura y volverla a lanzar. Esto solo se deberá hacer en los casos en los que las los requisitos sobre la infraestructura (arquitectura de la CPU, Sistema Operativo, etc.) de las configuraciones nueva y antigua sean "compatibles". También se puede llamar sin cambiar los apartados de configuración, para forzar de nuevo el proceso de reconfiguración. Esto puede servir para los casos en que el proceso de contextualización dependa de algún fichero que haya sido modificado y dicha modificación necesite de una reconfiguración del sistema para propagar dicho cambio sobre todas las MVs de la infraestructura.

b. Datos de Autorización

Hay un parámetro que debe ser incluido en todas las llamadas del API: los datos de autorización. Este parámetro es indicado de forma específica como el último parámetro en todas las funciones del API nativa y en el caso del API REST dentro de la cabecera "AUTHORIZATION" de las peticiones HTTP. El parámetro está compuesto por un conjunto de pares clave – valor donde se indican para cada componente de la arquitectura del sistema los valores necesarios para la autenticación. La Figura 10 muestra varios ejemplos de dichos pares.

Los valores que se deben indicar para cada componente son:

- type: El tipo del componente. Bien puede ser uno de los componentes del sistema como el Infrastructure Manager o VMRC o bien el tipo del proveedor Cloud, de los implementados hasta el momento: OpenNebula, EC2, OpenStack, OCCI, LibCloud o LibVirt.
- username: El nombre del usuario para la autenticación. En el caso de EC2 u OpenStack se refiere al Access Key ID.
- password: La contraseña para la autenticación. En el caso de EC2 u OpenStack se refiere al Secret Access Key.
- host: La dirección del servidor para indicar a qué proveedor en concreto corresponde dichos datos de autenticación. Tanto en el caso de EC2 como en del IM este dato no es utilizado.
- Id: Un identificador o etiqueta, único para cada elemento, que se pueda utilizar como etiqueta en el apartado "deploy" del RADL para elegir dicho proveedor para lanzar un conjunto de MVs.

```
id = one1; type = OpenNebula; host = svr:2633; username = usr; password = pass
id = one2; type = OpenNebula; host = srv2:2633; username = usr; password = pass
id = libvirt; type = LibVirt; host = server; username = user; password = pass
type = InfrastructureManager; username = user; password = pass
type = VMRC; host = http://server:8080/vmrc; username = user; password = pass
id = ec2; type = EC2; user = id; password = key
id = ost; type = OpenStack; host = server:8773; username = id; password = key
id = occi; type = OCCI; host = server:4567; username = user; password = pass
```

Figura 10. Ejemplos de datos de autenticación.

Estos datos de autorización los utiliza el IM para obtener los datos de los diferentes proveedores Cloud. De esta manera no es necesario mantener en el IM un listado de diferentes proveedores sino que es el propio usuario el que define dicho listado de forma dinámica en cada llamada al IM. Esto se debe tener en cuenta al realizar sucesivas llamadas al IM, de manera que los datos de autorización deben ser correctos para realizar las modificaciones de las infraestructuras creadas con anterioridad.

De la misma manera el IM utiliza dichos datos para la gestión del acceso a los datos de las infraestructuras. El IM no tiene un conjunto de usuarios predefinidos, sino que cuando un usuario accede al IM se asocia su par usuario-contraseña a las infraestructuras que ha creado de tal manera que sólo con ese mismo par podrá acceder a los datos de las mismas, manteniendo así la privacidad y seguridad de acceso a sus infraestructuras, evitando que se modifiquen infraestructuras de otros usuarios. Esto evita la necesidad de tener un administrador que gestione todos los usuarios que pueden acceder a la plataforma, si no que da libertad de acceso al servicio a cualquier usuario.

4.2.2 Cloud Selector

El Cloud Selector (CS) (ver Figura 1) es el componente encargado de seleccionar la mejor combinación posible entre las imágenes de MVs disponibles y proveedores Cloud para ubicar las diferentes MVs especificadas en el documento RADL. Para ello deberá contactar con el/los servidores de catálogo (VMRC) para la selección de las mejores imágenes de MVs disponibles, de acuerdo los requerimientos especificados por el usuario. En caso de que el usuario haya especificado una imagen específica el IM no necesitará contactar con el VMRC y se utilizará dicha imagen. De las credenciales del usuario extraerá el conjunto de proveedores Cloud a los que tiene acceso y deberá combinar dicha información para elegir el mejor par "Cloud, IMV" disponible. El CS deberá seleccionar los proveedores Cloud que sean compatibles con las imágenes seleccionadas por el catalogo. De igual manera deberá escoger las imágenes del tipo adecuado para ser lanzadas en el proveedor Cloud seleccionado.

Una funcionalidad interesante para el CS sería proporcionar funciones de conversión y migración de imágenes de unos proveedores Cloud a otros. Pero dada la falta de soporte para dicha funcionalidad por los proveedores Cloud, esta funcionalidad no ha sido implementada. Las tareas de conversión de imágenes, en muchos casos no es problemática puesto que existen herramientas como *qemu-img* del emulador QEMU[32] que permiten la conversión entre varios formatos. En algunos casos dependiendo del tipo plataforma de virtualización y del tipo de discos usados (SCSI, Volúmenes de Datos, etc.) surgen problemas al arrancar los S.O. Además en el caso de imágenes de gran tamaño este proceso puede ser bastante costoso. Pero el problema principal que hace que la migración sea imposible en mucho de los casos es que aunque en los proveedores Cloud públicos, en general, es posible descargar las IMV disponibles, no ocurre igual con los CMPs, que solo proporcionan funciones para añadir imágenes al repositorio de imágenes del proveedor. Además en la mayoría de las CMPs suele ser necesario tener permisos especiales para poder añadir imágenes al repositorio. Por tanto se ha determinado que la ubicación física de las imágenes marcará el proveedor Cloud donde se podrán lanzar las MVs.

La selección del "mejor" proveedor no es una tarea sencilla. El primer paso es definir los criterios por los que se van a ordenar los proveedores para elegir el mejor. Existen diferentes opciones: precio, prestaciones, precio/prestaciones, localidad geográfica, etc. Además se ha de tener en cuenta si se puede obtener la información necesaria sobre los proveedores Cloud para dichos criterios. El problema con la selección en el caso de tipos de proveedores diferentes, (Clouds on-premise, Clouds públicos, plataformas de virtualización) es que cada uno de ellos puede proporcionar diferente tipo de información. Por ejemplo en el caso de los Clouds públicos no es posible conocer el número de recursos totales ni utilizados, aunque sí te aseguran una calidad de servicio que permite saber las prestaciones de las MVs lanzadas (por ejemplo las denominadas ECUs en el caso de Amazon EC2). En el caso de los de tipo on-premise, suele pasar lo contrario, sí que es posible saber la cantidad de recursos disponibles, pero no siempre pueden garantizar las prestaciones de las MVs (en términos de potencia de CPU), pues dependen de las características de los nodos físicos donde estén alojadas.

En la selección de proveedores Cloud de tipo público, existen varios trabajos interesantes: STRATOS [46] facilita el despliegue de aplicaciones Cloud sobre múltiples proveedores Cloud usando el coste como el objetivo principal, usando algoritmos de optimización multi criterio. CompatibleOne [47] considera, no solo las restricciones impuestas por el usuario, sino tambien un amplio rango de objetivos: económicos, energéticos, geográficos, etc. para seleccionar el mejor

[32] http://wiki.qemu.org/Main_Page

proveedor Cloud. Otros trabajos como [6], [48] y [49] proponen diferentes soluciones para la selección del mejor proveedor Cloud, como el uso de Service Level Agreements (SLAs) o el algoritmo de los k-vecinos. Otra solución muy extendida es la selección del proveedor más barato (siempre que cumpla ciertos requisitos de calidad de servicio) como en [50] y [51]. No obstante todas estas soluciones tampoco son sencillas de automatizar puesto que la mayoría de los proveedores no proporcionan la información sobre los precios de sus instancias a través de su API de forma que pueda ser procesada de forma automática.

En el caso de combinar proveedores de tipo privado y público una técnica muy extendida es la denominada Cloud bursting, en la cual se eligen los Clouds on-premise locales en primer lugar, y cuando la demanda sobrepasa las capacidades locales se eligen Clouds de tipo público para satisfacer dicha sobredemanda.

El IM trata de hacer sencillas las tareas para los usuarios no avanzados y de dar flexibilidad a los usuarios expertos. En este caso el IM permite elegir de forma manual en el RADL el proveedor Cloud para las MVs a lanzar. Pero en el caso de usuarios no avanzados, el CS elegirá el proveedor Cloud de forma automática. Para ello el CS primero seleccionará la "mejor" IMV de acuerdo a los requisitos del usuario. Para la ponderación de las imágenes, se valora la idoneidad de la imagen con respecto a los requisitos impuestos por el usuario. Para ello se usa la ponderación de aplicaciones instaladas en la MV, de acuerdo a los valores indicados por el usuario en el RADL. En el caso de que no se indique un "peso" se supondrá un valor de 1. En caso de igualdad entre varias imágenes se selecciona aquella que tenga el menor número de aplicaciones instaladas. Esto permite evitar sobrecargas innecesarias en la MV, como por ejemplo tener instalado un servidor web o un contenedor Tomcat, con el consumo de recursos que eso implica (memoria, disco, CPU), en el caso que dichas aplicaciones no sean necesarias para el usuario.

Una vez seleccionada la imagen, se elegirá el proveedor Cloud en el que se encuentra alojada dicha imagen. En caso de que varias imágenes alojadas en diferentes proveedores Cloud tengan la misma valoración, se elegirá el proveedor Cloud que aparezca antes en los datos de autenticación. De manera que el orden en el que el usuario indica los datos de autenticación determina su preferencia por los mismos.

Pero también hay que tener en cuenta ciertas dependencias entre las MVs que hacen que un grupo de ellas (o todas) tengan que lanzarse sobre el mismo proveedor Cloud. Una de las más comunes es que un grupo de MVs estén unidas mediante una red privada. En este caso el IM lo resuelve de manera que fuerza que todas las MVs que estén unidas por una red privada se lancen dentro del mismo proveedor Cloud, asegurando la conectividad de red local. Para ello se obtiene, para el conjunto de MVs que deben ser lanzadas en el mismo proveedor, la suma de las ponderaciones de las "mejores" imágenes disponibles en dicho proveedor para cada una de ellas.

En el caso de que en el proceso de lanzamiento de una MV, se produzca algún error, el CS volverá intentar lanzar de nuevo la MV en el mismo proveedor. Una vez se sobrepasan un numero de errores consecutivos definidos como parámetro de configuración en el servicio de IM (por defecto el valor es de tres), el CS elegirá el siguiente par "IMV – Cloud" con mejor valoración y volverá a realizar el intento. En el caso de que haya restricciones de red que hagan que tenga que lanzar todas las MVs en el mismo Cloud, el IM deberá cancelar todas las MVs del grupo y repetir el proceso de selección de proveedor para todo el grupo.

4.2.3 Cloud Connector

Como se ha mostrado en el apartado de estado del arte, en la actualidad existen diferentes CMPs, con diferentes interfaces de conexión. Aunque en los últimos años han surgido diferentes iniciativas que tratan de unificar criterios para la definición de un API estándar único para el acceso a las plataformas Cloud de tipo IaaS, en la actualidad no hay ninguno extendido de forma "real". Por tanto y dado que las denominadas APIs de agregación para Cloud también tienen ciertos inconvenientes, descritos en la sección 2.4.2, se ha desarrollado una capa, denominada Cloud Connector (CC), que permita interactuar con los diferentes proveedores Cloud de forma homogénea y sencilla. Se han implementado un conjunto de operaciones reducido que permiten obtener toda la funcionalidad necesaria para el IM. De tal manera que permita de forma sencilla añadir nuevos tipos de proveedores.

El ciclo de vida de las MVs se puede alterar mediante 6 funciones:

- Lanzar MV: Crear y lanzar una MV en base a su RADL.
- Destruir MV: Destruir una MV.
- Obtener Información de la MV: Obtener la información sobre una MV en formato RADL.
- Parar MV: Para (pero no destruye) una MV.
- Iniciar MV: Inicia una MV (previamente parada con la función anterior).
- Modificar MV: Modifica las características de una MV, para proporcionar la elasticidad vertical, aunque esta funcionalidad no suelen proporcionarla los proveedores Cloud.

a. Creación de redes

La creación de las redes virtuales para la interconexión de las MVs es una tarea compleja de gestionar en los diferentes entornos. La mayoría de los proveedores Cloud públicos proporcionan funcionalidad para la creación de redes de

interconexión (como el servicio Virtual Private Cloud[33] de Amazon) que permiten crear una configuración de red totalmente personalizada. En cambio en los proveedores Cloud de tipo privado la gestión de las redes suele estar en manos del administrador de la plataforma y no se permite el acceso a los usuarios. De tal manera que de forma general se suele proporcionar una red de tipo público y una o más redes de tipo privado. En el caso del lanzamiento de las infraestructuras, el IM solo podrá usar las redes de acuerdo a las definidas por el proveedor Cloud.

b. OpenNebula

En el caso de OpenNebula usamos el API XML-RPC[34] que proporciona para tener un acceso nativo a la funcionalidad del mismo. A la hora de definir una MV se utiliza un lenguaje para definir la plantilla de la misma. En ella se puede especificar todas las características, tipo de y número de CPUs, cantidad de memoria, etc. lo que permite crear una instancia con las mismas características definidas por el usuario en el RADL.

En cuanto a las interfaces de red, aunque las plantillas de OpenNebula permiten la definición de cuantos interfaces quiera el usuario, como se ha comentado anteriormente dependen del conjunto de redes "virtuales" que haya definido el administrador de la plataforma. Por tanto el conector revisará las redes creadas para crear las interfaces que haya definido el usuario. En caso de que no se pueda realizar, por ejemplo, en caso de que se pida una interfaz con IP pública y el proveedor de OpenNebula no la pueda proporcionar el sistema mostrará el error.

En cuanto al acceso a las MVs por SSH se usan principalmente el nombre de usuario y contraseña de la IMV (proporcionado por el VMRC or por el usuario en el RADL). OpenNebula también permite la especificación de la clave pública SSH para el acceso mediante la clave privada, aunque en este caso la imagen de la MV debe estar previamente preparada con la instalación de los paquetes de contextualización específicos de OpenNebula. Dado que uno de los objetivos es evitar la necesidad de usar imágenes pre-configuradas se decidió evitar esta opción.

c. Amazon EC2, OpenStack

En el caso de estos proveedores se ha utilizado el API proporcionado por EC2 dado que OpenStack proporciona uno compatible. De tal manera que con algunas pequeñas modificaciones se pueden acceder a ambos proveedores. Hay otros CMPs como Eucalyptus que también usan dicho interfaz EC2 y que por tanto deberían poder accederse directamente o con ligeras modificaciones, pero que no han sido probados en el desarrollo de la tesis.

[33] http://aws.amazon.com/vpc/

[34] http://opennebula.org/documentation:archives:rel4.0:api

En este caso, a diferencia del caso anterior, utilizan un conjunto de tipos de instancias prefijados de tal manera que el IM no podrá crear una MV idéntica a la pedida, si no que instanciará una MV con el tipo de instancia que cumpla todos los requisitos definidos pero con el menor tamaño (precio) posible. En el caso de EC2 el plugin tiene un listado estático de los tipos de instancias disponibles (de manera similar a la solución utilizada por LibCloud), así como su precio. Dicho valor se utiliza como referencia para seleccionar las instancias más baratas. Aunque este precio varía con el tiempo, el orden de las instancias según su precio se mantiene.

Otro detalle a tener en cuenta son los interfaces de red. En principio todas las instancias de EC2 tienen un solo interfaz de red con una IP privada y una redirección de puertos desde una IP externa a dicho interfaz. Además se le pueden añadir todas las IPs externas necesarias usando las denominadas IPs "elásticas". De esta manera puede proporcionar una IP privada y externa. Para configuraciones más avanzadas se debe usar el servicio VPC donde sí que se pueden crear todo tipo de configuraciones de red. De momento el soporte para VPC no ha sido implementado en el conector para EC2. En el caso de OpenStack el funcionamiento es parecido pero, el redireccionamiento de la IP externa no se crea por defecto, de tal manera que en este caso será necesario pedir una IP "flotante", si el usuario pide una red accesible desde el exterior.

En cuanto al acceso por SSH a las MVs hay varias opciones. En este caso no se suele utilizar un esquema de nombre de usuario – contraseña, sino que se utilizan pares clave pública – privada. Dependiendo de la información que proporcione el usuario en el RADL (el campo disk.0.os.credentials) existen varias posibilidades:

- El usuario ya tiene un par clave pública – privada
    - El usuario indica la clave pública y privada:
        - En este caso se sube la clave pública a EC2 creando el "keypair" correspondiente, usando la privada para el acceso.
    - El usuario indica la clave privada y un nombre de clave pública:
    - El sistema usará dicho nombre como nombre del "keypair" para EC2 y usará la clave privada para el acceso. Dicho "keypair" debe existir previamente.
- El usuario no tiene par de claves:
    - El usuario no proporcionará ninguna información de credenciales de acceso y el sistema creará un "keypair" y proporcionará a través del sistema de información del IM (el atributo disk.0.os.credentials.private_key) la clave privada para que el usuario pueda acceder a la MV.

d. OCCI

El plugin de OCCI se ha desarrollado dado que es el estándar más extendido, aunque en el momento del desarrollo de este plugin solo existía la implementación de OpenNebula. El API de OCCI es de tipo REST usando XML como lenguaje para los contenidos de las definiciones de los componentes.

Al igual que en el caso anterior como hay tipos de instancias prefijados el IM no puede dar una MV idéntica a la pedida, si no que instanciará una MV con el tipo de instancia que cumpla todos los requisitos definidos pero con el menor tamaño posible (en base al número de cores y tamaño de memoria).

e. LibVirt

Este conector permite dar al IM un acceso a infraestructuras virtualizadas, es decir un simple servidor, sin necesidad de tener que instalar un CMP. Y que permite más adelante dar el paso al Cloud usando los mismos RADLs para describir las infraestructuras virtuales.

Para ello se ha utilizado LibVirt [52], dado que la librería más extendida para el acceso de plataformas virtualizadas.

En este caso al igual en el de OpenNebula se pueden crear las MVs con los requisitos exactos especificados por el usuario. El acceso por SSH a las MVs también será utilizando el esquema nombre de usuario – contraseña que deberá obtenerse del VMRC or deberá ser especificada por el usuario en el RADL.

f. LibCloud

Aunque en el análisis del estado del arte se ha comentado los problemas del uso de las denominadas APIs de agregación, como una solución general para el acceso a las plataformas Cloud, se ha implementado un plugin que utiliza LibCloud. Este plugin es más una prueba de concepto que un plugin de utilidad real. De esta manera, aunque la funcionalidad es limitada, permite lanzar y eliminar MVs en ciertos entornos no soportados directamente por el resto de plugins.

```
type = LibCloud; driver = RACKSPACE; user = user; key = pass
type = LibCloud; driver = EC2; access_id = user; secret = pass
```

Figura 11. Ejemplos de datos de autenticación adicionales para LibCloud.

En este caso el usuario deberá proporcionar todos los datos necesarios para establecer la conexión con el proveedor Cloud elegido. En primer lugar el nombre del driver a utilizar así como todos los parámetros necesarios para la conexión. Para indicar todos estos parámetros se utilizarán los datos de autenticación indicados anteriormente, pero en este caso usando valores diferentes a los estándar para el resto de plugins del sistema. Se muestran unos ejemplos en la Figura 11.

El problema de este tipo de librerías es que dada la heterogeneidad de las APIs utilizadas por las diferentes plataformas Cloud hay funciones concretas para casi todos los entornos que hace que el código necesario sea casi tan específico como usar el API propio de la plataforma. Por ejemplo se han realizado pruebas con el plugin de LibCloud para el acceso a EC2. En este caso para poder acceder a las MVs es imprescindible la gestión de los "keypairs", por tanto hay que hacer llamadas a las funciones específicas de EC2 dentro de LibCloud "ex_create_keypair" o "ex_import_keypair" para poder crearlos, de manera que el código debe ser demasiado concreto para cada plugin soportado por LibCloud.

4.2.4 Configuration Manager

La gestión de la contextualización de las infraestructuras es controlada por el componente denominado "Configuration Manager" (CM). Este componente se encarga de la orquestación de todos los pasos necesarios para configurar las infraestructuras, usando las herramientas de contextualización necesarias.

El primer paso es definir el funcionamiento de Configuration Manager a la hora de la uso de la herramienta de contextualización. Al igual que todo software para su correcto funcionamiento debe ser instalado, con lo que surge un problema ¿Quién contextualiza al contextualizador? Esta es una pregunta muy importante pues esta herramienta es la base de la configuración de la infraestructura Cloud. Para esta pregunta hay dos opciones a elegir:

La primera es "obligar" a que las IMVs vengan ya con el software preinstalado y configurado. Esta opción tiene la ventaja que descarga mucho de trabajo al Configuration Manager al eliminar dicha tarea (que en algunos casos puede ser compleja). Pero esta opción tiene el inconveniente de que complica la reutilización de IMVs puesto que tienen que estar específicamente preparadas para un entorno en concreto. La segunda opción es usar una serie de scripts básicos que instalen y configuren el contextualizador, de forma específica para cada infraestructura. Además al tratarse de herramientas de tipo cliente-servidor, es necesario configurar los clientes con una dirección fija para el servidor. Este tipo de soluciones son las adoptadas por plataformas como EC2[35] y otras compatibles con ellas como OpenStack[36].

Por cuestiones de flexibilidad se ha optado por la segunda opción de tal manera que uno de los pasos de la contextualización será la instalación y configuración, en todas las MVs de las herramientas de contextualización. Esto permitirá que se puedan reutilizar todo tipo de IMVs existentes, sin necesidad de ningún paso

[35] http://docs.amazonwebservices.com/AWSEC2/latest/UserGuide/AESDG-chapter-instancedata.html

[36] http://docs.openstack.org/trunk/openstack-compute/admin/content/metadata-service.html

previo. Los únicos requisitos que va a imponer el IM para poder utilizar una IMV serán que permita el acceso por SSH (usando las credenciales registradas en el VMRC o proporcionadas por el usuario en el RADL) y que el usuario tenga permisos de administración (para instalar y configurar el software) o bien pueda obtenerlos a partir del comando "sudo".

El siguiente paso es la selección de la herramienta de contextualización a utilizar, teniendo en cuenta que va a tener que ser instalada en la infraestructura a configurar. Por tanto la sencillez y rapidez de instalación será un punto importante a la hora de su elección. Para la realización del primer prototipo las capacidades de contextualización eran muy sencillas y se decidió utilizar el contextualizador desarrollado dentro del grupo investigación [30]. Esta sencilla herramienta permitía la instalación y un conjunto reducido de tareas de configuración, con la gran ventaja de no tener ningún tipo de requerimiento y ser fácil de utilizar.

Más adelante con la evolución del sistema se hizo mayor hincapié en la parte de contextualización con lo que se analizaron con detalle las herramientas actuales. Existen varios trabajos anteriores ([53], [54]) en los que se analizan diferentes características de los diferentes componentes software para la contextualización de MVs. Inicialmente se eligió Puppet dado que era el más utilizado y con un mayor soporte de la comunidad, desarrollando una segunda versión del sistema de contextualización. Tras diversas pruebas se comprobaron dos problemas: Uno inicial de escalabilidad que imponía unos requisitos hardware muy grandes (en cuanto memoria y CPU) al nodo "Puppet Master" cuando el número de nodos a configurar aumentaba. El segundo es debido a la filosofía "pull" (la más usada por las herramientas analizadas), es decir que son los nodos los que "piden" ser configurados de forma independiente y asíncrona, de manera que no se puede controlar desde un solo host la configuración de todos los nodos, para asegurar su correcta instalación y configuración en un momento fijado. De manera que para confirmar que un nodo se configura correctamente en un momento determinado hay que conectarse a dicho nodo para "forzar" la actualización, lo que implica conectarse por SSH a dicho nodo duplicando muchas tareas.

Posteriormente con la aparición de un nuevo software, denominado Ansible, se cambió el IM para usar dicho contextualizador. Ansible sigue la misma filosofía de herramienta sencilla con pocos requerimientos de nuestro contextualizador inicial, tiene un lenguaje con gran funcionalidad y extensibilidad, y ha demostrado una gran escalabilidad en sistemas de gran tamaño. Además implementa una estrategia de tipo "push" de manera que es el nodo configurador el que tiene la iniciativa a la hora de indicar cuándo se realizan las configuraciones, que es más adecuada en el entorno del IM para asegurar la contextualización de las infraestructuras en el momento que el IM indique. Además Ansible proporciona un API en Python para acceder a toda su funcionalidad lo cual también es un valor añadido para su uso desde la plataforma (desarrollada en Python).

Ansible utiliza SSH como método de acceso a las MVs. SSH es el método estándar de acceso a las máquinas Linux y por tanto viene instalado por defecto en casi todas las configuraciones estándar de las distribuciones. Aunque en principio Ansible está desarrollado para sistemas Linux hemos comprobado que también puede ser utilizado en máquinas con S.O. Windows. En este caso sí que necesitan una preparación previa para poder ser utilizadas: instalación y configuración del servicio OpenSSH y el intérprete de Python usando la herramienta Cygwin[37].

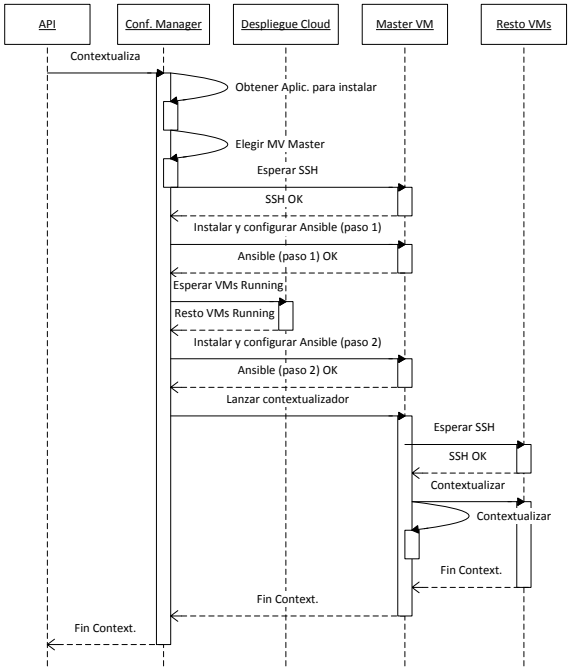

Figura 12. Diagrama de interacción del Configuration Manager.

Por lo tanto todas las MVs de la infraestructura deben tener el servicio SSH activado y accesible. En muchos proveedores Cloud la obtención de IPs públicas puede tener ciertas restricciones. Por tanto, para evitar la necesidad de IPs públicas en todas la MVs, el Configuration Manager se encarga de preparar la configuración de Ansible en una de las máquinas y hace que está se encargue de configurar al resto de las MVs. Por ello, para que este componente pueda configurar una infraestructura, al menos una de las MVs de la misma deberá tener una IP accesible desde la máquina en la que se haya lanzado el servicio del IM.

[37] http://www.cygwin.com/

El Configuration Manager es lanzado por el IM una vez que la infraestructura ha sido creada en el proveedor Cloud elegido. A partir de este momento los pasos que realiza para completar las tareas de contextualización son los siguientes, mostrando las interacciones en el diagrama de secuencia de la Figura 12:

1. Revisar la información obtenida del VMRC para cada una de las MVs comparando con los requisitos expresados por el usuario en el RADL, para conseguir la lista de aplicaciones a instalar.
2. Elegir una MV como "master". Esta MV se encargará de configurar al resto de las MVs mediante Ansible. Por tanto debe tener Linux como S.O. (para poder instalar Ansible), tener una IP pública (accesible desde la máquina que aloja el servicio IM) y tener conexión con todas las MVs lanzadas en la infraestructura (para poder configurarlas). En caso de que existan varias opciones el CM elegirá la que primero haya sido declarada en el documento RADL.
3. Esperar que la MV master se encuentre en ejecución y con el SSH activo. Usará las credenciales obtenidas del VMRC para poder acceder a ella, o las especificadas por el usuario en el RADL.
4. Configurar Ansible en la MV "master". Esto implica instalar el software y sus dependencias. Para realizar este paso se utiliza, de nuevo, la herramienta Ansible para instalar y configurar todo lo necesario. En este caso Ansible debe estar instalado en la máquina que aloja el servicio del IM, que será la encargada de configurar a la MV "master" accediendo mediante SSH. En este paso también se copian todos los ficheros de recetas que sean necesarios. Esto incluye un conjunto de recetas "base" que tiene preparadas el IM para facilitar ciertas tareas de configuración (dicha recetas son descritas en el apartado 4.2.6), así como las recetas indicadas por el usuario en el RADL.
5. Esperar que todas las MVs se encuentren en estado de ejecución. Este paso es necesario para asegurarse que en pasos posteriores se dispone de las direcciones IPs de todas las MVs, ya que en algunos proveedores Cloud las IPs de una MV no se asignan hasta que están en ejecución.
6. Finalizar la configuración de Ansible, con aquellos datos dependientes de las IPs de las MV lanzadas. Básicamente la creación del fichero de hosts de la maquina master y el fichero de "inventario" de Ansible[38] con todas las IPs de las MVs a configurar.

[38] http://docs.ansible.com/intro_inventory.html

7. Lanzar el contextualizador Ansible. Primero esperará que todas las MVs tengan el SSH activo y accesible desde la máquina master. Después se encargará de, usando las recetas de contextualización, configurar toda la infraestructura usando el acceso por SSH.

El Configuration Manager también es lanzado cada vez que se produce algún cambio en la infraestructura, ya sea añadiendo o quitando nodos, para reconfigurar toda la infraestructura y para adaptarse a los cambios. El proceso que sigue es el mismo que el descrito anteriormente, a partir del paso 5. La adición de un nuevo nodo, implica el despliegue y configuración de dicho nodo y la reconfiguración del resto, para incluir los cambios producidos por la aparición del nuevo nodo. Por tanto es, en general, una operación más costosa que la de eliminación ya que en el caso de la eliminación, simplemente hay que reconfigurar el resto de nodos para eliminar la información sobre dicho nodo.

a. Gestión de repositorios de recetas

A diferencia de otros sistemas de contextualización el proyecto Ansible no dispone de un sistema predefinido para la compartición de recetas o módulos, así que dan la recomendación de hacerlo mediante proyectos de GitHub. Por ello para facilitar el uso de recetas propias de los usuarios permitiendo la reutilización de las mismas y disminuir la cantidad de código YAML dentro de los RADL, el Configuration Manager tiene integrado un sistema para descargar recetas desde los diferentes repositorios de GitHub existentes en la actualidad. Para ello se ha creado un tipo especial de aplicación que son aquellas que empiezan por "ansible.modules" y que tendrán este formato:

```
ansible.modules.<usuario_github>/<nombre_repositorio>
```

Este requerimiento de aplicación deberá indicarse en el RADL de la siguiente manera:

```
disk.0.applications contains (name='ansible.modules.micafer/ansible-
playbook-hadoop')
```

Esto provocará que el Configuration Manager se descargue la receta en el directorio adecuado de configuración para poder hacer un "include" desde los apartados de configuración del RADL. En el apartado 4.2.6 se mostrarán algunos ejemplos.

4.2.5 Gestión de la elasticidad

La gestión de la elasticidad, como una de las características clave de las infraestructuras Cloud, es una tarea muy importante y repercute en la gestión de la infraestructura, que deberá permitir a las infraestructuras virtuales adaptarse a los requerimientos dinámicos de las aplicaciones.

Dos modelos de elasticidad han sido contemplados en el IM (Figura 13). Por una parte la elasticidad horizontal, que permite aumentar o disminuir el tamaño de un

conjunto de MVs, añadiendo o quitando MVs. Por otra parte, la elasticidad vertical permite que una MV individual pueda adaptarse de forma dinámica a los requerimientos de las aplicaciones (principalmente en términos de CPU y memoria).

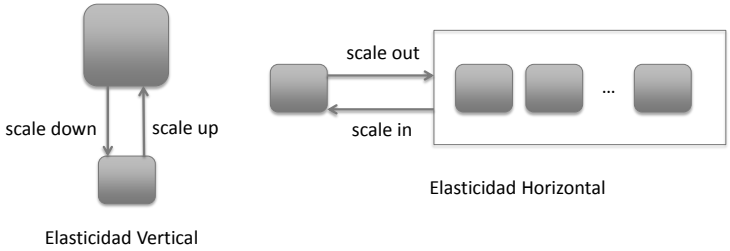

Figura 13. Modelos de elasticidad.

Por ejemplo un caso típico donde se pueden aprovechar las ventajas de la elasticidad horizontal es el caso de las aplicaciones High Throughput Computing (HTC) de tipo de barrido de parámetros, que de forma dinámica generan conjuntos de trabajos para ser ejecutados. Este tipo de trabajos inicialmente pueden requerir el despliegue de un conjunto de MVs para su ejecución. Pero en el caso de que la tasa de creación pudiera crecer y necesitar mayor número de recursos, la plataforma podría adaptarse añadiendo un conjunto de MVs nuevas. En el caso contrario, al disminuir la tasa de trabajos, se podría reducir el número de MVs, evitando tener recursos sin utilizar que, en el caso de un proveedor de Cloud público, lleva un sobrecoste asociado.

En el caso de la elasticidad vertical, si una aplicación científica se está ejecutando en una MV y necesita una cantidad de memoria superior a la inicialmente prevista, la MV puede aumentar sus prestaciones en cuanto a memoria para adaptarse a las nuevas necesidades, evitando bajada de prestaciones en la ejecución.

En la actualidad la gestión de la elasticidad horizontal es más sencilla pues es soportada de forma intrínseca por todos los CMPs. En el caso de la vertical su gestión es mucho más compleja ya que necesita que todos los elementos que participan de la virtualización lo soporten: el CMP, el gestor de virtualización y el sistema operativo de la MV. La mayoría de los sistemas operativos actuales (Windows, Linux, etc.) dan soporte para este tipo de funcionalidades con las técnicas denominadas “memory ballooning” [55]. También la mayoría de los hipervisores proporcionan soporte en mayor o menor medida (KVM, XEN, Hyper-V, VMWare, etc.). Pero en la actualidad ningún CMP tiene soporte nativo para este tipo de funcionalidades. Por eso dentro del GRyCAP se ha trabajado en la ampliación del API de OpenNebula para dar soporte para la funcionalidad de cambio de memoria de MVs en funcionamiento sobre plataformas KVM [56].

También hay que tener en cuenta que la elasticidad vertical puede suponer la migración de la MV a otro nodo físico de mayor potencia, en el caso de la adición de recursos. Esto implicaría que el CMP debería gestionar la denominada migración en vivo (*live migration*) para evitar interrupciones en el funcionamiento de la MV. La migración en vivo depende de la configuración concreta de cada despliegue Cloud y puede no ser soportada en muchos casos.

El IM proporciona funciones para dar soporte para las dos técnicas de elasticidad: La funciones AddResource y RemoveResource del API permite la gestión de la elasticidad horizontal y la función AlterVM de la vertical. Esto permite que con la combinación con otro software, que permita monitorización, de forma similar a la realizada en [56] o en [57], se pueda realizar la gestión elástica de las infraestructuras.

a. Sistema de monitorización

Una de las tareas fundamentales a la hora de la gestión de la elasticidad es la de la obtención de información de monitorización de la infraestructura para poder tomar las decisiones adecuadas. Para facilitar estas tareas, aunque el IM no proporciona directamente funciones para la monitorización, se ha integrado con Ganglia [58] dado que es un software muy extendido para las tareas de monitorización de infraestructuras.

El sistema permite que en caso de que la infraestructura tenga instalado Ganglia, la información de monitorización se pueda mostrar integrada con el resto de información de la infraestructura en formato RADL. Para que el IM pueda obtener la información de Ganglia este debe estar configurado de forma que una de las MVs (accesible desde el IM) tenga el denominado Ganglia Meta Daemon (gmetad), encargado de centralizar la información de toda la infraestructura, y el resto de nodos debe disponer de Ganglia Monitoring Daemon (gmond) configurado para enviar la información al nodo "gmetad".

Para dar mayor facilidad, para usuarios no avanzados, el propio IM tiene configuradas una serie de recetas que permiten instalar Ganglia de forma sencilla. De tal manera que, añadiendo un par de requisitos en el RADL (como veremos en el siguiente apartado), se pueda obtener la información de monitorización.

Con la integración con Ganglia a la información obtenida por el IM en el formato RADL con la información "básica" se añadirán una serie de nuevos parámetros:

- cpu.usage: Porcentaje de uso de la CPU.
- disk.free: Cantidad de disco libre.
- memory.free: Cantidad de memoria libre.
- swap: Cantidad de memoria swap total.

- swap.free: Cantidad de memoria swap libre.

4.2.6 Recetas incluidas en el IM

El IM proporciona una serie de recetas para facilitar ciertas de tareas en la configuración de infraestructuras, especialmente en las de tipo clúster. Dichas recetas se han almacenado en un conjunto de repositorios de GitHub: *https://github.com/micafer/ansible-playbooks*, de tal manera que el usuario pueda instalarlos indicando el nombre del usuario y el nombre del repositorio. A continuación mostraremos cómo puede el usuario acceder a ellas para disponer de dicha funcionalidad.

Las podemos agrupar en tres bloques: En el primero se muestran un conjunto recetas para facilitar la instalación de aplicaciones comunes en el ámbito científico:

Ganglia: Como se ha comentado en el apartado anterior, para facilitar la instalación de Ganglia y permitir la monitorización de las infraestructuras el IM proporciona una serie de recetas que permiten instalar Ganglia de forma sencilla. De tal manera que añadiendo un par de requisitos en el RADL se pueda tener Ganglia totalmente operativo en todos los nodos, usando el master como nodo central. Simplemente hay que añadir el siguiente requisito en el nodo "master"

```
disk.0.applications contains (name='gmetad')
```

y este en el resto de nodos:

```
disk.0.applications contains (name='ganglia')
```

En este caso y de forma transparente al usuario se descargará el repositorio de GitHub *https://github.com/micafer/ansible-playbook-ganglia* y configurará ambos tipos de nodos.

También se puede descargar dicho repositorio para incluir la llamada a la configuración de Ganglia dentro de una receta. Primero deberá indicar la siguiente aplicación el RADL para que el IM lo instale en nodo Ansible para su utilización:

```
disk.0.applications contains (name='ansible.modules.micafer/ansible-playbook-ganglia
')
```

Para luego incluir el siguiente código YAML en la receta del nodo "monitorizado":

```
- include: ganglia/tasks/ganglia.yml is_client=yes ganglia_gmetad= $IM_MASTER_FQDN
```

Y el siguiente en el nodo "master"

```
- include: ganglia/tasks/ganglia.yml is_client=no ganglia_gmetad=
<MASTER_NODE_NAME_OR_IP>
```

Hadoop [42]: Dada la popularidad de este tipo de clústeres, el IM proporciona recetas para configurar este tipo de clústeres, tanto montando el sistema de ficheros HDFS como el sistema de ejecución MapReduce. De esta manera se configuran los nodos de almacenamiento y cálculo:

```
- include: hadoop/tasks/hadoop-wn.yml hadoop_master=$IM_MASTER_FQDN
```

Y de esta manera el nodo central de control:

```
- include: hadoop/tasks/hadoop-master.yml hadoop_master= $IM_MASTER_FQDN
```

En este caso el usuario sí que deberá indicar de forma activa el repositorio de GitHub *https://github.com/micafer/ansible-playbook-hadoop*, de la siguiente manera:

```
disk.0.applications contains (name='ansible.modules.micafer/ansible-playbook-hadoop')
```

NTP: permite configurar el servicio NTP usado para la sincronización del reloj de los nodos de un clúster, tarea imprescindible en cualquier nodo que participe en una infraestructura de tipo Grid.

```
- vars:
   ntp_server: ntp.upv.es
  tasks:
  - include: ntp/tasks/ntp.yml
```

En este caso, al igual que en el caso de Hadoop, el usuario deberá indicar de forma activa el repositorio de GitHub *https://github.com/micafer/ansible-playbook-ntp*, de la siguiente manera:

```
disk.0.applications contains (name='ansible.modules.micafer/ansible-playbook-ntp')
```

También se han incluido un conjunto de plantillas para la configuración de ficheros básicos:

hosts.conf: Esta plantilla permite la creación de un fichero con las IPs de todos los nodos que forman el clúster. Esto permite que Ansible la gestione de forma dinámica a medida que se añaden o eliminan nodos de una infraestructura. Se utiliza de la siguiente manera:

```
- template: src=utils/templates/hosts.conf dest=/tmp/wn.list
```

ssh_known_hosts.conf: Posibilita la generación del fichero /etc/ssh/ssh_known_hosts para permitir el acceso ssh entre los nodos de la infraestructura.

```
- template: src=utils/templates/ssh_known_hosts.conf dest=/etc/ssh/ssh_known_hosts
```

Por ultimo tambien se han incluido variables del sistema para facilitar la detección del S.O.: Variables que tendrán valor cierto o falso para indicar el S.O. de la MV a

configurar. De tal manera que se pueden usar con la opción condicional "when" de Ansible para ejecutar determinada funcionalidad en dependencia del S.O. utilizado.

- fedora_os, debian_os, redhat_os
- linux, windows

4.3 VMRC

Como se ha comentado anteriormente, en la actualidad con el amplio uso de la virtualización y dado que las aplicaciones imponen requerimientos tanto software como hardware, se ha producido una creación masiva de IMVs. Por mostrar un dato, en Amazon EC2 hay en la actualidad un total de 21.718 imágenes públicas registradas. Además en el caso de EC2 las capacidades de búsqueda son muy básicas ya que sólo proporciona herramientas de búsqueda usando el nombre de la AMI o por palabras clave. Para poder utilizarlas, compartirlas y reutilizarlas evitando trabajo extra para los investigadores, todas estas IMVs deben ser correctamente indexadas.

El sistema Virtual Machine image Repository & Catalog (VMRC)[39] ha sido diseñado para permitir a los usuarios (y/o administradores Cloud) indexar y almacenar IMVs junto con un conjunto de metadatos que permitan describir sus características tanto hardware como software. El sistema ofrece funciones de búsqueda avanzada que permite al usuario especificar sus requerimientos y que el catálogo le devuelva un listado con las IMVs que satisfacen dichos requerimientos.

El sistema VMRC puede ser desplegado en el ámbito de un Cloud privado para almacenar e indexar las diferentes IMVs usadas en la plataforma Cloud. Pero también permite indexar IMVs almacenadas en otros proveedores Cloud. Además se puede utilizar para federar diferentes Cloud privados de manera que un solo despliegue de VMRC puede proporcionar imágenes para diferentes sistemas Cloud. El catálogo permite obtener una descripción en OVF de las imágenes, de forma independiente de los hipervisores

El sistema VMRC es un servicio web que permite la indexación de IMVs almacenadas tanto en su proprio repositorio como en repositorios externos (como Amazon EC2). El catálogo permite funcionalidades de búsqueda avanzada de manera que los usuarios puedan encontrar las IMVs que satisfagan un conjunto de requerimientos, tanto software como hardware. El creador de la IMV especifica las características de la IMV (hipervisor, arquitectura, tamaño del disco, etc.) y la configuración software (aplicaciones y librerías instaladas) y la registra en el catálogo. Los ficheros de la IMV pueden ser subidos después al repositorio del VMRC o simplemente especificar la URL de la misma.

[39] http://www.grycap.upv.es/vmrc

4.3.1 Capacidades y Requerimientos

El creador de la IMV expresa las capacidades de la IMV en forma de un lenguaje sencillo de pares clave-valor. Por ejemplo en la porción de código que aparece en la Figura 14 se muestra la descripción de una IMV creada para el hipervisor KVM, con una arquitectura de 32 bits, 10000 MBytes de disco duro y con Ubuntu 11.10 instalado. Para conectar con ella los usuarios podrán usar el nombre de usuario "user" y la contraseña "password". Además se indica que tiene la aplicación Octave 3.8 y Java JDK 1.6 instalados.

```
system.name = MyImage8
system.hypervisor = KVM
cpu.arch = i686
disk.size = 10000
disk.os.name = Linux
disk.os.flavour = Ubuntu
disk.os.version = 11.10
disk.os.credentials.user = user
disk.os.credentials.password = password
disk.applications contains (name = org.gnu.octave, version = 3.8 )
disk.applications contains (name = com.java-jdk, version = 1.6, path
= /usr/local/bin/java )
```

Figura 14. Ejemplo 1 consulta VMRC.

Como se puede ver las características de las IMVs usan un subconjunto de todas las características definidas en el lenguaje RADL. Con este mismo tipo de lenguaje de tipo clave-valor el usuario puede hacer las consultas al VMRC. Al igual que se definía en el RADL se pueden expresar requerimientos de tipo "hard" y de tipo "soft" con su valor correspondiente.

En el ejemplo 2 (Figura 15) se muestra una consulta para obtener las IMVs que cumplan los siguientes requerimientos: Debe tener arquitectura i686 y haber sido creada para el hipervisor KVM. El sistema operativo debe ser Ubuntu 9.10 o superior. Y debe tener instalado Tomcat versión 7 o superior, Java JDK o superior. Finalmente en caso de tener Octave versión 3.8 o superior será mejor valorado.

```
system.hypervisor = KVM
system.arch = i686
disk.os.name = Linux
disk.os.flavour = Ubuntu
disk.os.version >= 9.10
disk.applications contains (name = org.apache.tomcat, version >= 7.0)
disk.applications contains (name = com.java-jdk, version >= 1.5)
soft 50 disk.applications contains (name = org.gnu.octave, version >=
3.8)
```

Figura 15. Ejemplo 2 consulta VMRC.

4.3.2 Convención de URIs

Se ha definido una convención de nombres para las URIs a la hora del registro de IMVs en el VMRC, cuando las IMVs están almacenadas en los propios proveedores Cloud. El uso de esta convención permite al IM conocer su ubicación y cómo acceder a ellas. El campo de protocolo de la URI se utiliza para especificar el tipo de proveedor Cloud: *one* (OpenNebula), *ec2* (Amazon EC2) y *ost* (OpenStack). En el caso de OpenNebula y OpenStack, los campos Dirección y Puerto tienen su función por estándar y la ruta se utiliza para especificar el ID de las imágenes, un número entero para OpenNebula y una cadena de texto en OpenStack. En el caso de EC2, el campo de dirección se utiliza para especificar la región en la que se almacena la imagen y la ruta de acceso para almacenar el nombre de la AMI.

- one://server:port/image-id
- ost://server:port/ami-id
- ec2://region/ami-id

4.4 Diagrama de interacción

Para mostrar cómo interaccionan los diferentes componentes de la plataforma, la Figura 16 muestra el diagrama para el caso de la función de creación de infraestructuras, dado que es la de mayor complejidad y donde se utilizan todos los componentes. En el resto de funciones las interacciones son similares a este (como en el caso de añadir o borrar recursos a una infraestructura) o son más sencillos y apenas tienen interacciones con distintos componentes (como listar infraestructuras, obtener información de las MVs, etc.).

La creación de una infraestructura empieza con la llamada por parte del usuario al API del sistema indicando el documento RADL con la descripción de la infraestructura así como los datos de autenticación para acceder a los proveedores Cloud. El primer paso que realiza el IM es contactar con el servicio VMRC para obtener la lista de IMVs que más se adecuan a los requisitos indicados por el usuario en el RADL. En el caso de que el usuario indique de forma específica la imagen a utilizar (indicando el valor `disk.0.image.url` en el RADL usando la convención del URIs descrita en la sección 4.3.2) no será necesaria la conexión con el VMRC. Usando dicho listado y el de proveedores Cloud disponibles el Cloud Selector elegirá los pares IMV – Cloud más adecuados para cada MV a lanzar. En ese momento, mediante el Cloud Connector conectará con los proveedores Cloud elegidos para realizar el lanzamiento efectivo de las MVs. El CC devolverá los identificadores de las MVs en formato normalizado de tal manera que identifiquen de forma univoca a una MV dentro del entorno del IM y permitan el acceso a la información de la misma. Por ejemplo en el caso de una MV en EC2 incluirá tanto el identificado de la MV como la región en la que ha sido lanzada. En ese momento la operación retornará al usuario el identificador que se le haya asignado a dicha

infraestructura. A partir de ese momento y de forma asíncrona, el Configuration Manager empezará las tareas de contextualización que aparecen en la Figura 12.

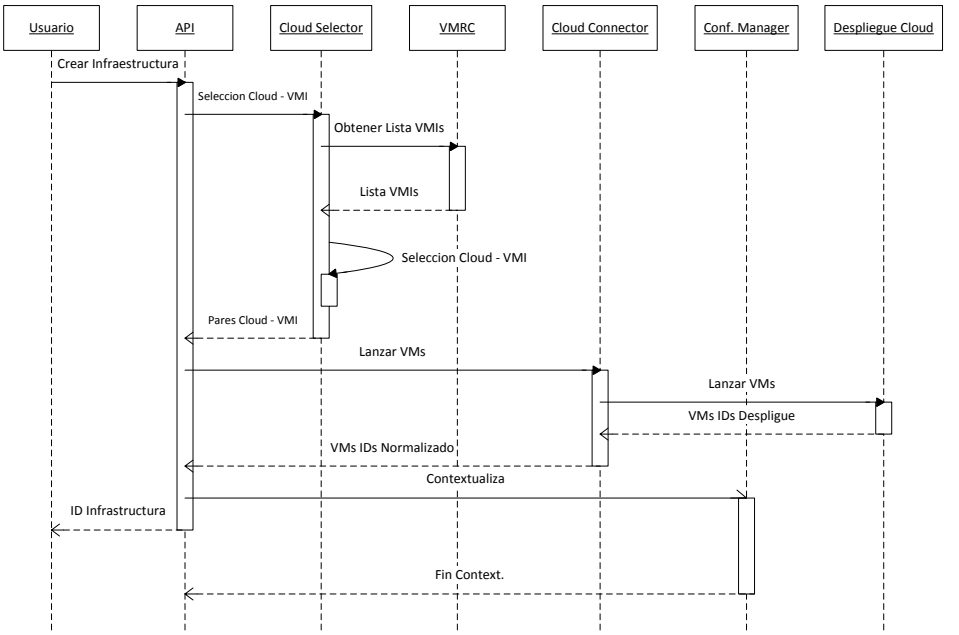

Figura 16. Diagrama de interacción de la función Crear Infraestructura.

4.5 Herramientas de Cliente

Para acceder a la funcionalidad del servicio del Infrastructure Manager, no desde otra capa software de alto nivel, sino por parte de los usuarios finales, se han desarrollado dos herramientas: una de tipo línea de comandos y una aplicación web.

4.5.1 Cliente de línea de comandos

La herramienta de cliente de línea de comando ha sido desarrollada en Python (como el resto de componentes del IM) de tal manera que, aunque está diseñada para usarse en entornos Linux, puede utilizarse en cualquier plataforma que disponga del intérprete de Python (Windows, OS X, etc.).

A continuación detallamos el uso y los parámetros de la herramienta que aparecen en la Figura 17:

- Parámetros generales:
    - -u|--xmlrpc-url <url>: Indica la URL del servicio XML-RPC del Infrastructure Manager al que conectarse. Si no se indica, por defecto es http://localhost:8888

- -a|--auth_file <filename>: Indica el fichero con los datos de autenticación. Este parámetro es obligatorio.

```
usage: client.py [-u|--xmlrpc-url <url>] [-a|--auth_file <filename>] operation
op_parameters
  options:
    --version           show program's version number and exit
    -h, --help          show this help message and exit
    -a AUTH_FILE, --auth_file=AUTH_FILE
                        File with authentication data
    -u XML-RPC, --xmlrpc-url=XML-RPC
                        URL of the InfrastructureManager service.
Operations:
  list
  create          <radl_file>
  destroy         <inf_id>
  getinfo         <inf_id> [status|radl|cloud]
  getcontmsg      <inf_id>
  getvminfo       <inf_id> <vm_id>
  addresource     <inf_id> <radl_file>
  removeresource  <inf_id> <vm_id>
  alter           <inf_id> <vm_id> <radl_file>
  start           <inf_id>
  stop            <inf_id>
  reconfigure     <inf_id> <radl_file>
```

Figura 17. Parametros del cliente de línea de comandos.

- Operaciones:
  - list: Muestra los identificadores de las infraestructuras creadas por el usuario.
  - create: Crea una infraestructura usando el fichero RADL indicado como parámetro.
  - destroy: Elimina la infraestructura con identificador indicado como parámetro.
  - getinfo: Muestra la información de la infraestructura con identificador indicado como parámetro.
  - getcontmsg: Muestra la información de contextualización de la infraestructura con identificador indicado como parámetro.
  - getvminfo: Muestra la información de la MV con identificador indicado como segundo parámetro de la infraestructura con identificador indicado como primer parámetro.
  - addresource: Añade los recursos especificados en el RADL del fichero indicado como segundo parámetro a la infraestructura con identificador indicado como primer parámetro.

- removeresource: Elimina la MV con identificador indicado como segundo parámetro de la infraestructura con identificador indicado como primer parámetro.
- start: Arranca la infraestructura, previamente para con la operación stop, indicada como parámetro.
- stop: Para (pero sin eliminar) la infraestructura indicada como parámetro.
- alter: Modifica la MV con identificador indicado como segundo parámetro de la infraestructura con identificador indicado como primer parámetro usando el RADL del fichero indicado como tercer parámetro.
- reconfigure: Modifica los datos de configuración de la infraestructura y vuelve a lanzar el proceso de contextualización.

4.5.2 Interfaz web

Para dar un acceso más sencillo e intuitivo para usuarios menos familiarizados con aplicaciones de línea de comandos, también se ha desarrollado una interfaz web en PHP.

Para el acceso a la interfaz web el usuario debe autenticarse por medio de un usuario y contraseña de que debe dar de alta el usuario administrador de la herramienta web. En la parte izquierda se muestra el menú donde aparecen los apartados que pueden ser gestionados por la aplicación: Infraestructuras, Credenciales, RADLs, y la parte de administración (Usuarios y Grupos) en caso de tener los privilegios adecuados.

Cuando se accede al apartado de Infraestructuras la aplicación muestra el listado de las infraestructuras que se encuentran lanzadas por el usuario. A través de este listado el usuario puede gestionar sus infraestructuras existentes (Figura 18). En ese listado se muestran los identificadores de las MV que forman parte de la infraestructura. Desde aquí el usuario podrá eliminar la infraestructura o añadir nuevos recursos a la misma. Al pinchar en los identificadores de las MVs se muestra la información de la MV en concreto (Figura 19): En la parte superior se muestra el estado de la misma. Después la información del proveedor Cloud donde ha sido lanzada y las IPs de conexión. A continuación aparece la información sobre las características de la MV mostrando el nombre y el valor de los atributos definidos en el RADL. Desde esta misma página, el usuario puede terminar la MV y eliminarla de la infraestructura.

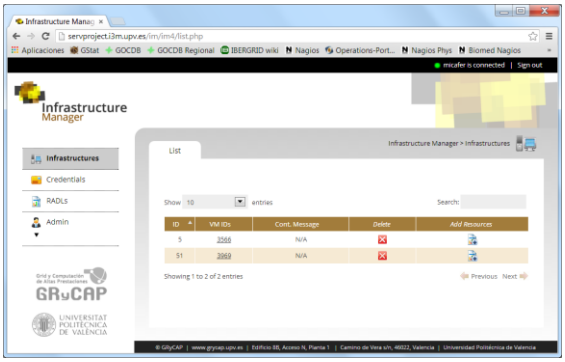

Figura 18. Listado de Infraestructuras.

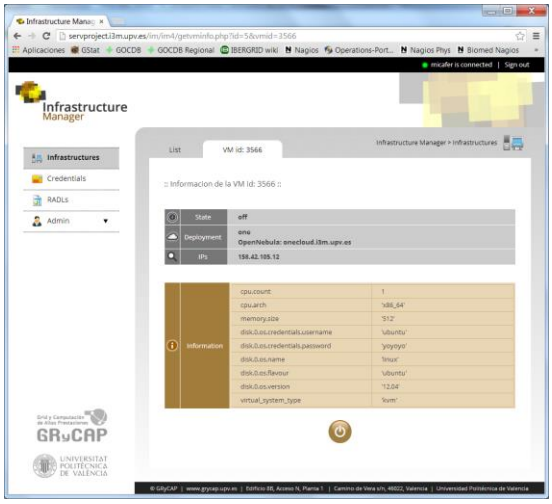

Figura 19. Información de la MV.

En el apartado de Credenciales el usuario podrá gestionar el listado de las credenciales de acceso a cada uno de los componentes de la arquitectura del sistema para formar los datos de autorización necesarios para conectar con el API (ver sección 4.2.1b). En la Figura 20 se muestra el listado desde donde el usuario podrá editar, eliminar o deshabilitar temporalmente las credenciales. Esta opción permite tener almacenadas todas las credenciales de los despliegues Cloud a los que el usuario tenga acceso y habilitar sólo aquellas que se quieran usar al desplegar una infraestructura concreta

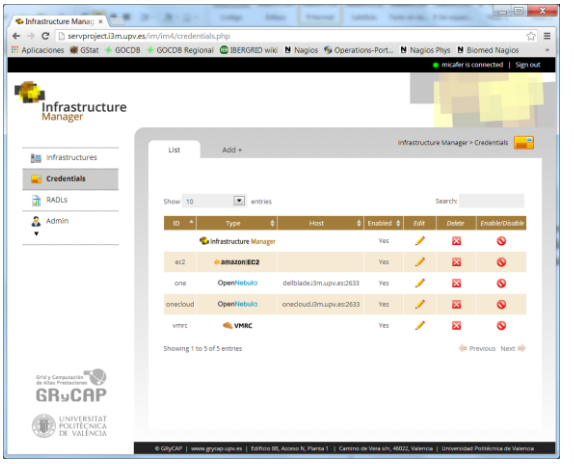

Figura 20. Listado de credenciales de acceso.

En cuanto a la sección RADL, permite gestionar un repositorio de documentos RADL para crear sus infraestructuras (Figura 21). Esto permite tener un conjunto de definiciones de infraestructuras preparadas para poderlas modificar o lanzar simplemente con un click del ratón (botón),

En el formulario de edición del RADL (Figura 22), además de poder editarlo, e incluirle un nombre y una descripción, el usuario puede gestionar qué usuarios de la aplicación pueden acceder a dicho RADL, de manera que se permita la compartición de los diferentes RADLs entre diferentes grupos de usuarios, facilitando así su reutilización. El sistema de permisos de acceso a los RADLs es similar al utilizado en los sistemas de Linux. De esta manera el usuario puede dar permisos de lectura "r", que permite que los usuarios puedan ver el RADL, de escritura "w" para modificarlo, y de ejecución "x" que permite crear infraestructuras usando dicho RADL. Del mismo modo puede aplicar el mismo conjunto de permisos para los "otros" usuarios pudiendo permitir el acceso público a un RADL para todos los usuarios.

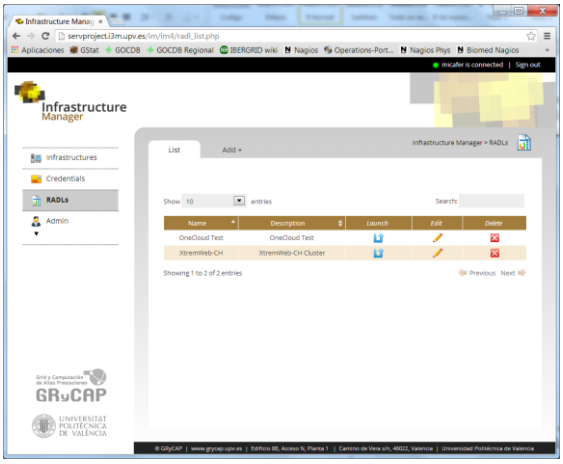

Figura 21. Listado de RADLs.

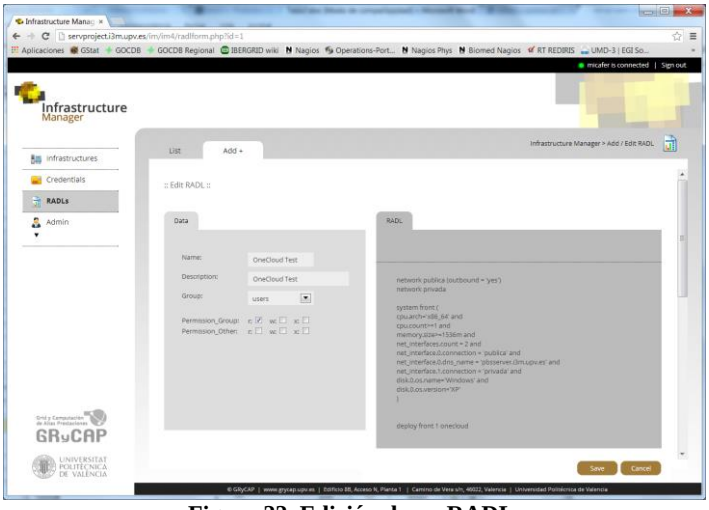

Figura 22. Edición de un RADL.

Finalmente el apartado de "Admin" solo aparece en caso de que el usuario que haya accedido a la aplicación web tenga permisos de administración. En este caso el administrador del sistema puede crear el conjunto de usuarios que podrán acceder a la interfaz web del IM (Figura 23). Indicando su nombre de usuario, contraseña, y el conjunto de grupos a los que pertenece dicho usuario. Esto permite crear grupos de usuarios agrupados por diferentes áreas y favorecer posteriormente la compartición de los documentos RADL entre ellos.

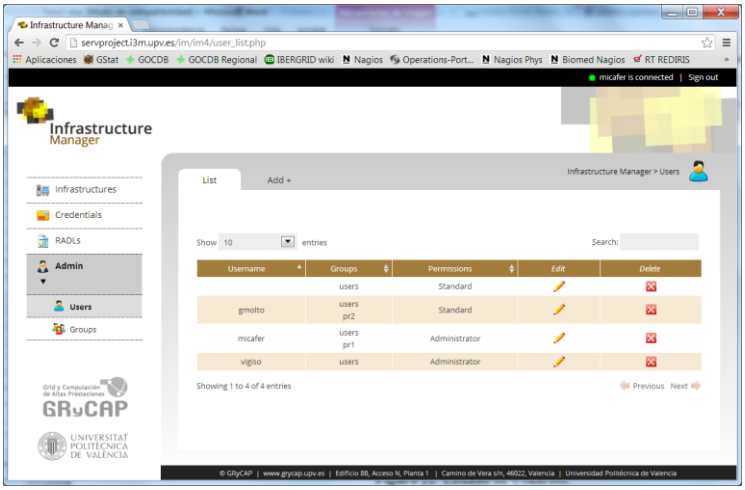

Figura 23. Listado de Usuarios.

5 Casos de Uso

A continuación se van a mostrar diferentes casos de usos del sistema para demostrar la funcionalidad del mismo así como para mostrar los tiempos necesarios para el despliegue de las infraestructuras usando diferentes tipos y tamaños para mostrar la versatilidad y escalabilidad del sistema propuesto. Inicialmente se abordarán casos de uso de la plataforma, desde el punto de vista de un usuario que accede directamente a la funcionalidad del servicio para lanzar sus infraestructuras virtuales, para el posterior lanzamiento de sus aplicaciones. Después se mostrarán otros servicios que, usando el API acceden a la funcionalidad del IM para crear un sistema de mayor complejidad con funcionalidad agregada.

5.1 Conjunto de imágenes base creadas/usadas

Para la realización de los siguientes casos de uso se han creado un conjunto sencillo de IMVs base con varios sistemas operativos y solo se han instalado aquellas aplicaciones que por su complejidad o duración de la instalación, recomendaban su instalación en la imagen base.

Como imágenes base de S.O. se han utilizado las más demandadas por las aplicaciones mostradas en el caso de uso. Además se ha tratado de tener tanto distribuciones de Linux basadas en la rama "Debian" (Debian, Ubuntu, Mint, etc.) como es Ubuntu como distribuciones de la rama "RedHat" (RedHat Entreprise, Fedora, openSUSE, etc.) como son Scientific Linux (SL) o Amazon Linux. En cuanto a versiones, se ha elegido la versión 12.04 (LTS) de Ubuntu dado que es la que tiene soporte de larga duración y es una de las más extendidas en la actualidad. En cuanto a SL se han elegido dos releases distintas la 5.8 y 6.4 para realizar pruebas con diferentes versiones. También se ha probado Amazon Linux (versión 2012.09) dado que es una de las imágenes base proporcionadas por Amazon para desplegar MVs en EC2. Además en el caso de la imagen de SL 5.8 se ha creado una con la instalación de Globus Toolkit 4 [59] para las pruebas en las que se despliegan servicios Grid con dicho middleware.

Por último se ha completado el listado de imágenes base con dos imágenes de Windows (versiones XP y Server 2008), para las pruebas con dicho S.O. Como se ha comentado anteriormente dicha imagen tiene preinstalado OpenSSH y Python usando Cygwin.

5.2 Proveedores Cloud utilizados

Las pruebas se han realizado usando dos proveedores Cloud diferentes. El primero de ellos (denominado ONE) es un Cloud de tipo privado dentro de nuestras propias instalaciones (*on-premise*) usando OpenNebula 4.2 compuesto por un conjunto de doce blades Dell (M600, M610 y M910) con dos procesadores QuadCore y 16 GB

de RAM, en el caso de los dos primeros y cuatro procesadores QuadCore y 64 GB de RAM, en el caso de los M910, con un total de 120 cores. El sistema utiliza como almacenamiento una partición con OCFS2 compartida para todos los nodos a través de un sistema de almacenamiento por iSCSI. El esquema es el mostrado en la Figura 24.

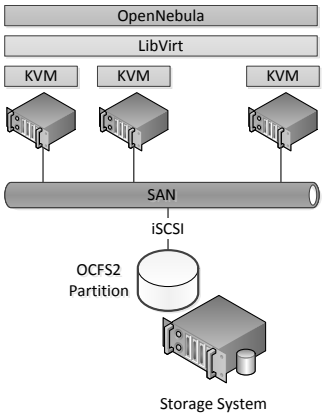

Figura 24. Plataforma Cloud privada con OpenNebula.

Como segundo proveedor, en este caso de tipo público se ha utilizado el servicio Amazon EC2, dado que AWS es el mayor proveedor Cloud de la actualidad.

5.3 Lanzamiento de infraestructuras científicas

En este apartado vamos a mostrar ejemplos de cómo se pueden lanzar infraestructuras virtuales para la ejecución de diferentes aplicaciones científicas. En las medidas de tiempo mostradas en los siguientes casos de uso, el tiempo usado para desplegar las infraestructuras se ha descompuesto en los siguientes pasos (básicamente los descritos en la sección 4.2.4):

- Master arrancada: Tiempo transcurrido desde que el usuario indica la creación de la infraestructura al IM hasta que la máquina designada como master se encuentra en estado "running". Este tiempo depende de dos aspectos, el tipo y tamaño de imagen que se utilice y lo ocupado que se encuentre el proveedor Cloud donde se lance. En el caso de grandes proveedores Cloud (como Amazon EC2) la carga del sistema tiene una influencia mucho menor que en proveedores Cloud locales de tamaño mediano – pequeño. En general las imágenes base utilizadas tienen un tamaño pequeño, de apenas 1GB (salvo en el caso de la instalación de Windows XP que sobrepasa los 5GB) con lo que el tiempo suele ser pequeño (del orden de 1-2 minutos). En el caso de EC2 las AMIs utilizadas son de tipo EBS (EBS-backed) cuyo tiempo de despliegue es muy corto.

- Master accesible: Tiempo necesario para tener el puerto SSH abierto en el nodo master. Este tiempo depende del tiempo usado por el S.O. de la MV para arrancar y lanzar el servicio SSH.
- Ansible configurado: Tiempo necesario para configurar Ansible en la máquina master, lo que implica instalar el software y sus dependencias. El proceso de instalación de Ansible es relativamente simple dado que tiene un conjunto de dependencias pequeño.
- Sistema configurado: Este proceso implica primero esperar que todas las MVs tengan el servicio SSH activo y accesible desde la máquina master para luego realizar la instalación y configuración indicada en el RADL. Este proceso se hace en paralelo en todos los nodos de manera que puede producir ciertos cuellos de botella en el acceso a la red o al disco, que dependerá de la configuración propia de la plataforma Cloud a la hora del almacenamiento de las IMVs así como del host físico donde se despliegue finalmente el conjunto de MVs.

En los procesos de adición y eliminación de un nodo y posterior reconfiguración del sistema, los pasos son los siguientes:

- MV arrancada/eliminada: Este tiempo incluye tanto el tiempo de arrancar la MV en el proveedor Cloud (o su eliminación) como la preparación de los archivos necesarios, tanto para la configuración de la nueva MV como para la reconfiguración del resto de nodos por parte de Ansible. Ambos procesos se realizan en paralelo de tal forma en los casos de eliminación (donde es un proceso casi inmediato) y en los de adición en los que la MV tiene un proceso de arranque rápido (en la mayoría de las pruebas es así) el tiempo lo marca la preparación de los ficheros de Ansible para la reconfiguración.
- Sistema reconfigurado: Este tiempo incluye el tiempo necesario hasta tener acceso por SSH al nuevo nodo, así como la configuración completa del nuevo nodo y la reconfiguración de todos los demás para añadir al nuevo nodo.

5.3.1 Clúster Hadoop

En la actualidad Hadoop con su implementación MapReduce [60] es ampliamente utilizado en multitud de campos de la ciencia para el procesado de grandes volúmenes de datos. Por tanto es muy útil tener una manera sencilla de lanzar este tipo de clústeres sobre plataformas Cloud.

En concreto para la aplicación BLAST (Basic Local Alignment Search Tool) [61], que es una de las herramientas más utilizadas en el campo de la bioinformática,

existe un software desarrollado para su ejecución en clústeres de tipo Hadoop denominado Hadoop BLAST[40]

```
network privada
network publica (outbound = 'yes')

system front (
cpu.arch='x86_64' and cpu.count>=2 and
memory.size>=1536m and
net_interface.0.connection = 'publica' and
net_interface.1.connection = 'privada' and
disk.0.os.name='linux' and
disk.0.os.flavour='ubuntu' and
disk.0.os.version='12.04' and
disk.0.applications contains (name='ansible.modules.micafer/ansible-playbook-
hadoop')
)

system wn (
cpu.arch='x86_64' and
memory.size>=1536m and
net_interface.0.connection='privada' and
disk.0.os.name='linux' and
disk.0.os.flavour='ubuntu' and
disk.0.os.version='12.04'
)

configure front (
@begin
---
  - tasks:
    - include: hadoop/tasks/hadoop-master.yml hadoop_master=$IM_MASTER_FQDN

@end
)

configure wn (
@begin
---
  - tasks:
    - include: hadoop/tasks/hadoop-wn.yml hadoop_master=$IM_MASTER_FQDN

@end
)

deploy front 1
deploy wn 5
```

Figura 25. RADL caso clúster Hadoop.

[40] http://salsahpc.indiana.edu/tutorial/hadoopblast.html

Dado que este tipo de clústeres es muy demandado en la actualidad el IM incorpora una receta para facilitar la tarea de instalación de Hadoop para usuarios no avanzados. De manera que con el RADL que aparece en la Figura 25 se puede lanzar un clúster Hadoop totalmente configurado (con un conjunto de valores por defecto) del tamaño que desee el usuario.

En este caso se utilizan las recetas proporcionadas por el IM (como se indica en la sección 4.2.6), de tal manera que los pasos a realizar no aparecen en el propio RADL. Básicamente consisten en la instalación de la máquina virtual de Java, descargar el paquete de Hadoop, para descomprimirlo y modificar los ficheros de configuración para crear el clúster con las MVs lanzadas. Finalmente se lanzan los diferentes servicios en cada uno de los nodos para tener la infraestructura Hadoop en funcionamiento.

Tabla 4. Tiempos de creación de las infraestructuras virtuales del caso Hadoop

	6 nodos ONE	**11 nodos ONE**	**11 nodos EC2**	**51 nodos EC2**
Master arrancada	3:30	9:22	0:32	0:44
Master accesible	1:06	1:32	0:27	0:11
Ansible configurado	1:55	1:35	2:52	4:13
Sistema configurado	5:34	9:44	5:38	9:18
TOTAL Creación	12:05	22:13	9:29	14:26
Nueva MV arrancada	0:55	1:06	2:05	2:59
Reconfiguración	4:18	4:27	3:41	8:24
TOTAL Adición	5:13	5:33	5:46	11:23
MV eliminada	0:20	0:25	1:58	2:56
Reconfiguración	0:58	1:07	1:19	3:40
TOTAL Eliminación	1:18	1:32	3:17	6:36

La Tabla 4 muestra los tiempos necesarios para lanzar un clúster Hadoop tanto en la plataforma OpenNebula local como en Amazon EC2, variando el número de nodos de cálculo entre 5 y 50. En el caso de EC2 las instancias seleccionadas por el IM para los WN han sido las “m1.small”, dado que solo indica que requiere un core y 1.5GB de memoria. En cambio para el front-end el IM elige una instancia de tipo “c1.medium”, ya que es la más barata con 1.5GB de memoria y dos cores de proceso.

En cuanto a los tiempos de despliegue, en el caso de la infraestructura de OpenNebula se puede apreciar como cuando el número de nodos aumenta, los tiempos de todas las tareas aumentan, dado que el despliegue sufre una sobrecarga, en la red al acceder todas las MVs simultáneamente a la partición OCFS2 compartida por red en todos los nodos. En el caso de EC2 al ser una plataforma de gran tamaño la diferencia es mucho menor.

Se puede ver que las tareas de instalar Ansible, arrancar y eliminar MV, tardan más tiempo en el caso de EC2. Esto ocurre dado que las tareas de preparar y configurar (o reconfigurar) la MV master para la configuración del resto de nodos es realizada por el Ansible desde la maquina donde está alojado el IM, en el caso de los nodos en el Cloud on-premise de OpenNebula dichas tareas son en red local, y en el caso de EC2 necesitan acceso a la red externa de la UPV y por tanto son más lentas. En caso necesario se podría lanzar el servicio IM en alguna instancia EC2 mejorando las prestaciones de estas fases. También cabe destacar que la fase de configuración del sistema es sensiblemente más rápida en el caso de EC2, dado que los diferentes cuellos de botella de acceso a la red y a disco que se producen en el Cloud on-premise. Este tipo de patrones veremos que se repite en todos los casos de estudio.

5.3.2 Clúster EMI

EGI-InSPIRE[41] (Integrated Sustainable Pan-European Infrastructure for Researchers in Europe) es un proyecto co-financiado por la Comisión Europea, como resultado de la colaboración de más de 100 instituciones y 40 países. Entre estas instituciones se encuentran 37 Iniciativas Grid Nacionales (NGIs), dos Organizaciones de Investigación Intergubernamentales Europeas (EIROs) y ocho miembros de la región de Asia-Pacífico.

El proyecto EGI-InSPIRE tiene como misión establecer una e-Infraestructura Europea de Grid (EGI, European Grid Infraestructure) sostenible y confiable que pueda dar soporte a las necesidades de análisis de datos a gran escala que tienen hoy en día los científicos europeos y sus socios internacionales. Para ello cuenta con los proveedores de infraestructuras Grid europeos y del resto del mundo y con las nuevas tecnologías de Infraestructuras de Computación Distribuida (DCIs) que se están desarrollando.

Este caso es similar al anterior, en el que lanzamos una infraestructura de tipo clúster para el cálculo científico. Pero en este caso se realiza el despliegue de un clúster destinado a formar parte de la infraestructura del proyecto EGI. Para ello hay que instalar el software desarrollado por otro proyecto europeo denominado EMI (European Middleware Initiative)[42]. En concreto en este caso se ha instalado la

[41] http://www.egi.eu/about/egi-inspire/

[42] http://www.eu-emi.eu/

versión 2 del software de EMI, instalando un clúster con Torque como el LRMS. Este caso es muy interesante para demostrar tanto la funcionalidad del despliegue como para la posterior gestión de la elasticidad para "sites" dentro del proyecto EGI. La infraestructura será lanzada expresando todos los requerimientos en el documento RADL que se muestra en la Figura 26.

```
network publica (outbound = 'yes')
network privada

system front (
cpu.arch='x86_64' and cpu.count>=2 and
memory.size>=1024m and
net_interface.0.connection = 'publica' and
net_interface.0.dns_name = 'pbsserver.i3m.upv.es' and
net_interface.1.connection = 'privada' and
net_interface.1.dns_name = 'front' and
disk.0.os.name='linux' and
disk.0.os.flavour='scientific' and
disk.0.os.version>='6' and
disk.0.applications contains (name='ansible.modules.micafer/ansible-playbook-ntp')
)

system wn (
cpu.arch='x86_64' and
memory.size>=1024m and
net_interface.0.connection = 'privada' and
net_interface.0.dns_name = 'wn-#N#' and
disk.0.os.name='linux' and
disk.0.os.flavour='scientific' and
disk.0.os.version>='6'
)

configure common (
@begin
- action: template src=utils/templates/hosts.conf dest=/root/wn-list.conf
- get_url: url=http://…/EGI-trustanchors.repo dest=/etc/yum.repos.d/EGI-
trustanchors.repo
- yum: name=http://…/epel-release-6-8.noarch.rpm state=present
- yum: name=http://emisoft.web.cern.ch/emisoft/dist/EMI/2/sl6/x86_64/base/emi-
release-2.0.0-1.sl6.noarch.rpm state=present
- yum: pkg=${item} state=installed
  with_items:
    - ca-policy-egi-core
    - emi-torque-client
- get_url: url=$RSCF/$item dest=/root/$item
  with_items:
    - site-info.def
    - groups.conf
    - users.conf
@end
)
configure front (
@begin
---
  - vars:
```

```
     ntp_server: ntp.upv.es
     RSCF: http://webgrycap.i3m.upv.es/RSCF/UMD
    tasks:
    - include: common.yml
    - include: ntp/tasks/ntp.yml
    - yum: pkg=emi-torque-server state=installed
    - command: /usr/sbin/create-munge-key creates=/etc/munge/munge.key
    - file: path=/etc/munge/munge.key owner=munge group=munge mode=0400
    - service: name=munge state=started
    - command: /opt/glite/yaim/bin/yaim -c -s site-info.def -n TORQUE_client -n
TORQUE_server chdir=/root
    - command: /sbin/iptables -I INPUT 2 -p tcp --dport 15001:15004 -j ACCEPT
    - command: /sbin/iptables -I INPUT 2 -p udp --dport 15001:15004 -j ACCEPT
@end
)
configure wn (
@begin
---
  - vars:
     ntp_server: ntp.upv.es
     RSCF: http://webgrycap.i3m.upv.es/RSCF/UMD
    tasks:
    - include: common.yml
    - yum: pkg=emi-wn state=installed
    - include: ntp/tasks/ntp.yml
    - copy: src=/etc/munge/munge.key dest=/etc/munge/munge.key owner=munge
group=munge mode=0400
    - service: name=munge state=started
    - command: /opt/glite/yaim/bin/yaim -c -s site-info.def -n TORQUE_client -n WN
chdir=/root
    - lineinfile: dest=/var/lib/torque/mom_priv/config regexp=pbsserver
line="\$pbsserver front"
    - service: name=pbs_mom state=restarted
    - service: name=iptables state=stopped
@end
)

deploy front 1
deploy wn 5
```

Figura 26. RADL caso clúster EMI.

Los pasos descritos en las secciones "configure" del RADL son los estándar definidos para la instalación y configuración del software EMI[43]. El usuario debe crear todos los ficheros necesarios para la configuración: site-info.def, groups.conf, users.conf y ponerlos accesibles desde una URL o puede incluirlos en la propia receta.

Una de las ventajas del uso del IM para este tipo de despliegues es que permite de forma sencilla la adición o eliminación de nodos con una simple llamada al IM, y

[43] https://twiki.cern.ch/twiki/bin/view/EMI/GenericInstallationConfigurationEMI2

todo el sistema será reconfigurado para seguir con su funcionalidad. Lo que facilita la gestión de clústeres que pueden adaptarse de forma elástica a la demanda.

Se han realizado tres pruebas usando los dos proveedores Cloud descritos anteriormente. Se ha lanzado una prueba inicial con 6 nodos en OpenNebula y EC2 y una prueba de mayor tamaño con 32 nodos en EC2. En el caso de EC2 las instancias seleccionadas por el IM han sido las mismas que en el caso anterior: para los WN han sido las "m1.small", y para el front-end la "c1.medium".

Tabla 5. Tiempos de creación de las infraestructuras virtuales del caso EMI

	6 nodos ONE	**6 nodos EC2**	**31 nodos EC2**
Master arrancada	0:55	0:41	0:45
Master accesible	1:06	0:36	0:36
Ansible configurado	2:58	2:58	3:40
Sistema configurado	25:50	14:06	18:17
TOTAL Creación	30:49	18:21	23:18
Nueva MV arrancada	0:47	2:14	2:45
Reconfiguración	26:28	13:31	15:20
TOTAL Adición	27:15	15:45	18:05
MV eliminada	0:47	2:10	2:40
Reconfiguración	1:41	2:35	5:35
TOTAL Eliminación	2:28	4:45	10:38

La Tabla 5 muestra los tiempos para cada uno de los pasos. Como se pueden ver en los resultados el tiempo necesario para tener desplegado un clúster EMI-2 de tamaño pequeño o mediano totalmente funcional es de unos 20 o 30 minutos, donde la fase de mayor tiempo es la de configuración del sistema. Esto se produce ya que en esta fase se realiza la instalación de todos los paquetes necesarios del software EMI en todos los nodos y el proceso de configuración completo. Como la imagen básica en bastante sencilla el número de paquetes es bastante grande (254 paquetes que requieren un total de 112 MB). El tiempo para añadir un nodo es muy similar (normalmente inferior, dado que solo una MV se configura por completo) ya que el tiempo de configurar el nuevo nodo es el mismo en todos los casos, y hay pequeñas variaciones que dependen de la carga del sistema Cloud en un momento determinado. Finalmente la tarea de eliminación es la más rápida dado que eliminar

una MV es una tarea rápida y solo implica la fase de reconfiguración de los nodos del sistema que, como se muestra en la tabla, tarda unos pocos minutos.

5.3.3 Análisis de mutaciones

El proceso de análisis de mutaciones consiste en la identificación de variantes significativas en el genoma de un individuo con respecto al genoma de referencia y las bases de datos de mutaciones conocidas. Con la generalización de las técnicas de secuenciación masivas (comúnmente conocidas como NGS, Next Generation Sequencing) este proceso se ha incrementado en complejidad y en necesidad de recursos. En este ámbito existen una serie de herramientas ampliamente utilizadas por los investigadores que consisten en el preprocesado, comparación con la referencia, identificación de variantes y búsqueda de las variantes en bases de datos. Las herramientas utilizan un genoma consenso de referencia contra el que se compara los distintos fragmentos del genoma de un individuo.

Un flujo de trabajo simplificado y representativo en este tipo de estudios consiste en la comparación (mapeo) y el análisis de la variabilidad (variant calling). Para realizar este proceso, además de disponer de las herramientas adecuadas es necesario disponer de la referencia del genoma debidamente indexada y la capacidad de interrogar a las bases de datos de variantes. La receta descargará el genoma de referencia y realizará el preprocesado necesario para su utilización.

El conjunto de herramientas instalado será el siguiente:

- FastQC[44]: Análisis estadístico de la calidad de las secuencias de entrada.
- Bowtie2[45]: Alineador que permite la comparación de un conjunto de secuencias con respecto a una referencia, utilizando la transformada de Burrows-Wheeler para el semilleo y el método de Smith-Waterson para la búsqueda.
- Samtools[46]: Conjunto de utilidades para la conversión de los resultados producidos por el Bowtie2, para facilitar su análisis.
- GATK[47]: Herramienta de Variant Calling que realiza un análisis estadístico y probabilístico de las diferencias entre las secuencias de entrada y la referencia, de forma que se pueda concluir la susceptibilidad de que una diferencia sea realmente significativa.

[44] http://www.bioinformatics.babraham.ac.uk/projects/fastqc/

[45] http://bowtie-bio.sourceforge.net/bowtie2/

[46] http://samtools.sourceforge.net

[47] http://www.broadinstitute.org/gatk/

- Galaxy[48]: Galaxy es una plataforma web que proporciona diferentes herramientas de para el procesado de datos en bioinformática.

```
network publica (outbound = 'yes')

system ngs (
cpu.arch='x86_64' and cpu.count>=2 and
memory.size>=1024M and
net_interface.0.connection = 'publica' and
disk.0.os.flavour='ubuntu' and
disk.0.os.version='12.04'
)

configure bowtie_install (
@begin
---
  - vars:
      install_dir: /opt/ngs
      bowtie2_url: http://sourceforge.net/projects/bowtie-
bio/files/bowtie2/2.1.0/bowtie2-2.1.0-linux-
x86_64.zip/download?use_mirror=garr&use_mirror=garr
      bowtie2_file: bowtie2-2.1.0-linux-x86_64.zip
      user: investigador
    tasks:
    - apt: pkg=unzip state=installed
    - file: path=${install_dir} state=directory
    - get_url: url=${bowtie2_url} dest=${install_dir}/${bowtie2_file}
    - command: unzip ${install_dir}/${bowtie2_file} chdir=${install_dir}
creates=${install_dir}/bowtie2-2.1.0
    - file: path=${install_dir}/bowtie2-2.1.0/bowtie2 mode=0755
    - user: name=${user} password=O3Je2Qxgx0w generate_ssh_key=yes shell=/bin/bash
@end
)

configure bowtie_launch (
@begin
---
  - vars:
      install_dir: /opt/ngs
      fasta_file: chromFa.tar.gz
      fasta_server: http://hgdownload.cse.ucsc.edu/goldenPath/galGal3/bigZips
      input_data: chr7.fa,chr8.fa,chr9.fa

    tasks:
    - file: path=${install_dir}/fasta state=directory
    - get_url: url=${fasta_server}/${fasta_file}
dest=${install_dir}/fasta/${fasta_file}
    - command: tar -xzf ${install_dir}/fasta/${fasta_file}
chdir=${install_dir}/fasta creates=${install_dir}/fasta/chrM.fa
    - command: ${install_dir}/bowtie2-2.1.0/bowtie2-build ${input_data} out_index
chdir=${install_dir}/fasta
      async: 3600
    - file: path=${install_dir}/fasta state=directory owner=${user} recurse=yes
@end
)

configure tools (
@begin
---
```

[48] http://galaxyproject.org

```
  - vars:
      install_dir: /opt/ngs
      fastqc_file: fastqc_v0.10.1.zip
      fastqc_server: http://www.bioinformatics.babraham.ac.uk/projects/fastqc/
      samtools_url: http://downloads.sourceforge.net/project/samtools/...
      samtools_file: samtools-0.1.19.tar.bz2
      gatk_server: http://webgrycap.i3m.upv.es/RSCF
      gatk_file: GenomeAnalysisTK-2.6-5.tar.bz2
      galaxy_file: tip.tar.gz
      galaxy_server: https://bitbucket.org/galaxy/galaxy-dist/get
    tasks:
    - apt: pkg=bzip2,openjdk-7-jre-headless,make,gcc,zlib1g-dev,libncurses5-dev
state=installed
    - file: path=${install_dir} state=directory
    - get_url: url=${fastqc_server}/${fastqc_file}
dest=${install_dir}/${fastqc_file}
    - command: unzip ${install_dir}/${fastqc_file} chdir=${install_dir}
creates=${install_dir}/FastQC
    - file: path=${install_dir}/FastQC/fastqc mode=0755

    - get_url: url=${samtools_url} dest=${install_dir}/${samtools_file}
    - command: tar -xjf ${install_dir}/${samtools_file} chdir=${install_dir}
creates=${install_dir}/samtools-0.1.19
    - command: make chdir=${install_dir}/samtools-0.1.19
creates=${install_dir}/samtools-0.1.19/samtools
      async: 600

    - get_url: url=${gatk_server}/${gatk_file} dest=${install_dir}/${gatk_file}
    - command: tar -xjf ${install_dir}/${gatk_file} chdir=${install_dir}
creates=${install_dir}/GenomeAnalysisTK-2.6-5-gba531bd

    - get_url: url=${galaxy_server}/${galaxy_file}
dest=${install_dir}/${galaxy_file}
    - command: tar -xzf ${install_dir}/${galaxy_file} chdir=${install_dir}
creates=${install_dir}/galaxy-galaxy-dist-9d4cbf2a1c13
    - shell: mv ${install_dir}/galaxy-galaxy-dist-* ${install_dir}/galaxy
creates=${install_dir}/galaxy
    - copy: dest=${install_dir}/galaxy/universe_wsgi.ini
src=${install_dir}/galaxy/universe_wsgi.ini.sample
    - ini_file: dest=${install_dir}/galaxy/universe_wsgi.ini section='server:main'
option=host value=0.0.0.0
    - ini_file: dest=${install_dir}/galaxy/universe_wsgi.ini section='app:main'
option=admin_users value=user1@bx.psu.edu
    - lineinfile: dest=${install_dir}/galaxy/start.sh regexp=nohup line="nohup sh
${install_dir}/galaxy/run.sh --reload > ${install_dir}/galaxy/server.log 2>
${install_dir}/galaxy/server.err < /dev/null &" create=yes
    - lineinfile: dest=${install_dir}/galaxy/start.sh regexp=exit line=exit
    - shell: /usr/bin/bash ${install_dir}/galaxy/start.sh
creates=${install_dir}/galaxy/server.log
@end
)

deploy ngs 1

contextualize (
    system ngs configure bowtie_install step 1
    system ngs configure bowtie_launch step 2
    system ngs configure tools step 2
)
```

Figura 27. RADL caso análisis de mutaciones.

Para este caso el RADL, mostrado en la Figura 27, no contempla el lanzamiento de un conjunto de MVs, sino de una MV individual para que posteriormente el investigador pueda acceder a la misma para realizar el procesado de sus secuencias de forma manual. Este caso de uso muestra una utilidad para proporcionar a un investigador con una MVs donde trabajar de forma sencilla. En este tipo de casos la capacidad de repetibilidad de la recetas es muy útil, ya que permite que el investigador pueda acceder a una MV siempre en las mismas condiciones de uso, independientemente del proveedor Cloud en el que lo lance y sin necesidad de mantener la MV en ejecución todo el tiempo. Por ejemplo puede lanzar la MV para realizar sus trabajos durante la semana laboral, pero si durante el fin de semana no va a utilizarla podría o bien eliminar la MV, en el caso de que no haya datos que mantener en la MV, o parar dicha MV, para volverla a lanzar más adelante ahorrando gastos en el Cloud. Además permite que inicie sus trabajos en MVs del Cloud privado de su institución, y en caso de necesitar una instancia de gran tamaño (no soportada por su institución) simplemente tendría que lanzar el RADL en un Cloud público que si le permita instancias del tamaño deseado.

Tabla 6. Tiempos de creación de la infraestructura virtual del caso NGS

	1 nodo ONE	**1 nodo EC2**
Master arrancada	1:30	0:28
Master accesible	1:06	0:27
Ansible configurado	1:35	2:33
Sistema configurado	15:44	5:52
TOTAL Creación	19:55	9:20

En este caso se ha utilizado las facilidades del apartado “contextualize” para paralelizar ciertas fases de la contextualización de larga duración. En concreto se han realizado tres fases: En la primera se instala el programa bowtie2. En la segunda se lanza en paralelo la instalación del resto de utilidades junto con la descarga e indexado de los datos genómicos deseados (los procesos más largos). De esta manera se reduce el tiempo de contextualización total realizando en paralelo tareas que son independientes.

En cuanto a los tiempos mostrados en la Tabla 6. El tiempo total ha sido de unos 20 minutos, teniendo en cuenta que se ha utilizado el genoma del pollo proporcionado por la UCSC[49] de un total de 319M de datos y que se ha realizado el indexado de

[49] http://hgdownload.soe.ucsc.edu/goldenPath/galGal4/bigZips/

los cromosomas 7, 8 y 9. En este caso los procesos de adición y eliminación de MVs no tienen sentido.

5.3.4 XWCH en Windows

En este caso de uso vamos a lanzar la infraestructura necesaria para lanzar una aplicación de Filogenetica denominada MetaPiga2 [62]. Este caso de uso, aunque también se orienta hacia la bioinformática, tiene una implementación totalmente diferente. Un estudio filogenético compara genéticamente varios individuos de una población. El objetivo es encontrar la relación evolutiva entre ellos, para determinar la distancia genómica entre especies o entre individuos potencialmente emparentados. Los estudios filogenéticos requieren realizar alineamientos múltiples, en los que no hay una referencia a comparar, sino que se comparan todos con todos, y realizar complejos análisis estadísticos.

Sin embargo, el interés principal de este caso de uso se centra en que la aplicación MetaPiga2 utiliza un sistema distribuido de cómputo basado en el paradigma de computación oportunística (XWCH [63]). Dicha aplicación permite ejecutarse en máquinas con S.O. Windows, lo que nos permite demostrar la funcionalidad del IM también con MV con dicho S.O.

```
network publica (outbound = 'yes')
network privada

system front (
cpu.arch='x86_64' and cpu.count>=2 and
memory.size>=1536m and
net_interface.0.connection = 'publica' and
net_interface.0.dns_name = 'server.i3m.upv.es' and
net_interface.1.connection = 'privada' and
disk.0.os.name='linux' and
disk.0.os.flavour='ubuntu' and
disk.0.os.version='12.04'
)

system wn (
cpu.arch='x86_64' and cpu.count>=2 and
memory.size>=1536m and
net_interface.0.connection = 'privada' and
net_interface.0.dns_name = 'wn-#N#' and
disk.0.os.name='Windows' and
disk.0.os.flavour='XP' and
disk.0.applications contains (name='openssh' and preinstalled='yes') and
disk.0.applications contains (name='python' and preinstalled='yes') and
disk.0.applications contains (name='cygwin' and preinstalled='yes')
)

configure wns1 (
@begin
---
  - vars:
      java_file: jre-7u25-windows-i586.exe
      src_url: http://webgrycap.i3m.upv.es/RSCF/XWCH
    tasks:
    - get_url: url=${src_url}/${java_file} dest=/tmp/${java_file} mode=0755
    - command: /tmp/${java_file} /s creates="/cygdrive/c/Archivos de
programa/Java/jre7/bin/java.exe"
```

```
    - get_url: url=${src_url}/XWCH-Installer.jar dest=/tmp/XWCH-Installer.jar

    - lineinfile: dest="/tmp/options.dat" regexp=^INSTALL_PATH=
line="INSTALL_PATH=C:\\\\Archivos de programa\\\\XtremWeb-CH_Worker" create=yes
    - lineinfile: dest="/tmp/options.dat" regexp=^serveraddress=
line=serveraddress=http://${IM_MASTER_FQDN}:8080/WorkerServiceService/WorkerService
    - lineinfile: dest="/tmp/options.dat" regexp=^port= line=port=2222
    - lineinfile: dest="/tmp/options.dat" regexp=^space= line=space=
    - lineinfile: dest="/tmp/options.dat" regexp=^task= line=task=
    - lineinfile: dest="/tmp/options.dat" regexp=^user= line=user=miguel

    - shell: java -jar XWCH-Installer.jar -options options.dat > /tmp/install.log 2>
/tmp/install.log chdir=/tmp creates="/cygdrive/c/Archivos de programa/XtremWeb-
CH_Worker/config.properties"
      ignore_errors: yes

    - lineinfile: dest=/tmp/worker.sh regexp=nohup line="nohup /cygdrive/c/Archivos\
de\ programa/XtremWeb-CH_Worker/startxw.bat > /tmp/worker.log 2> /tmp/worker.log <
/dev/null &" create=yes
    - lineinfile: dest=/tmp/worker.sh regexp=exit line=exit
@end
)

configure wns2 (
@begin
---
  - tasks:
    - shell: /usr/bin/bash /tmp/worker.sh creates=/tmp/worker.log
@end
)

configure front (
@begin
---
  - vars:
      install_file: xwchcoord-complete-package-linux-v3
      src_url: http://webgrycap.i3m.upv.es/RSCF/XWCH
    tasks:
    - apt: pkg=${item} state=installed
      with_items:
      - unzip
      - mysql-server
      - openjdk-6-jre-headless
    - service: name=mysql state=started
    - command: mysqladmin -u root password mysqlpass
      ignore_errors: yes
    - get_url: url=${src_url}/${install_file}.zip dest=/tmp/${install_file}.zip
    - command: unzip /tmp/${install_file}.zip chdir=/tmp
creates=/tmp/${install_file}/install-xwchcoord-v1.sh

    - lineinfile: dest=/tmp/${install_file}/install-xwchcoord-v1.sh
regexp=^use_mysql= line=use_mysql=1
    - lineinfile: dest=/tmp/${install_file}/install-xwchcoord-v1.sh
regexp=^ear_file= line=ear_file=xwch.2013.01.09.ear
    - lineinfile: dest=/tmp/${install_file}/install-xwchcoord-v1.sh
regexp=^mysql_root_password= line=mysql_root_password=mysqlpass
    - lineinfile: dest=/tmp/${install_file}/install-xwchcoord-v1.sh
regexp=^xwch_database_name= line=xwch_database_name=xwchdb
    - lineinfile: dest=/tmp/${install_file}/install-xwchcoord-v1.sh
regexp=^xwch_database_username= line=xwch_database_username=xwcuser
    - lineinfile: dest=/tmp/${install_file}/install-xwchcoord-v1.sh
regexp=^xwch_database_password= line=xwch_database_password=xwcpasswd
```

```
    - shell: /bin/bash install-xwchcoord-v1.sh > /tmp/install.log 2>
/tmp/install.log chdir=/tmp/xwchcoord-complete-package-linux-v3
creates=/root/glassfish-3.1.1/glassfish3/glassfish/domains/domain1/logs/server.log
    - shell: echo "use xwchdb;insert into SETTINGS values(2,'/tmp',3000,'1.8')" |
mysql -u root -pmysqlpass
      ignore_errors: yes
    - shell: echo "use xwchdb;insert into CLIENT values(1,1,'0cc0d453-8208-46c0-
8756-ba8329be10b4',0,'micafer1@upv.es',1,1,100,'Caballer','UPV','grycap','Miguel')"
| mysql -u root -pmysqlpass
      ignore_errors: yes

    - lineinfile: dest=/tmp/container.sh regexp=nohup line="nohup /root/glassfish-
3.1.1/glassfish3/glassfish/bin/asadmin start-domain domain1 > /tmp/containerst.log
2> /tmp/containerst.log < /dev/null &" create=yes
    - lineinfile: dest=/tmp/container.sh regexp=exit line=exit
    - shell: /bin/bash /tmp/container.sh creates=/tmp/container.log

    - get_url: url=${src_url}/warehouse.zip dest=/tmp/warehouse.zip
    - command: unzip /tmp/warehouse.zip chdir=/tmp creates=/tmp/javanode.jar
    - lineinfile: dest=/tmp/config.properties regexp=^xwserver=
line=xwserver=http://${IM_NODE_FQDN}:8080/WorkerServiceService/WorkerService
create=yes
    - lineinfile: dest=/tmp/config.properties regexp=^nodeport= line=nodeport=8105
    - lineinfile: dest=/tmp/config.properties regexp=^nodefolder=
line=nodefolder=/root/.xw_repository

    - lineinfile: dest=/tmp/javanode.sh regexp=nohup line="nohup java -Xrs -
classpath . -jar /tmp/javanode.jar /tmp/config.properties > /tmp/javanode.log 2>
/tmp/javanode.log < /dev/null &" create=yes
    - lineinfile: dest=/tmp/javanode.sh regexp=exit line=exit
    - shell: /bin/bash /tmp/javanode.sh creates=/tmp/javanode.log
    - shell: /bin/bash /tmp/container.sh creates=/tmp/container.log
    - wait_for: port=8080 delay=10
@end
)

deploy front 1
deploy wn 10

contextualize (
    system front configure front step 1
    system wn configure wns1 step 1
    system wn configure wns2 step 2
)
```

Figura 28. RADL caso XWCH.

En el RADL, que se muestra en la Figura 28, se describen dos tipos de MVs una para la maquina servidor del sistema desktop Grid y otra para el conjunto de nodos de cálculo. En caso de que ya se dispusiera de un nodo servidor y solo se deseará lanzar un conjunto de nodos de cálculo la receta sería idéntica pero eliminado el tipo de nodo "front" y poniendo la referencia correcta a dicho nodo en vez de "IM_MASTER_FQDN".

En cuanto a la contextualización, en la parte del servidor es necesaria la instalación y configuración del sistema de gestión de base de datos MySQL[50]. Después se

[50] http://www.mysql.com/

descarga, configura y lanza el servicio "Coordinador". Para luego descargar, configurar y lanzar el servicio de almacenamiento del sistema. Finalmente espera a que el puerto de servidor (8080) se encuentre activo para asegurar la finalización.

En el nodo de cálculo debe descargar e instalar la máquina virtual de Java para luego descargar el archivo "JAR" con el instalador del cliente, con el cual se instala y configura el software, para finalmente lanzar el servicio "worker" configurado para conectar con nuestro "Coordinador".

Como se puede ver la parte de configuración del "worker" se ha partido en dos apartados "configure" denominados "wns1" y "wns2" para separar la configuración de dichos nodos en dos fases.

En este caso se demuestra la utilidad de la sección "contextualize", ya que se han definido dos fases de configuración, una primera en la que se configuran tanto el nodo "Coordinador" como los "workers" y una segunda en la que se lanzan todos los nodos "workers" una vez que el Coordinador se encuentra en funcionamiento.

La Tabla 7 muestra los tiempos necesarios para el despliegue de la infraestructura mostrada con diferente número de nodos de cálculo en los dos proveedores Cloud utilizados. En este caso se ha añadido un nuevo tiempo a la tabla: MVs Arrancadas. Este es el tiempo necesario esperar a que todas las MVs se encuentren en estado de ejecución.

En este caso al desplegar un conjunto de MVs de mayor tamaño la copia de las imágenes de las MVs a los nodos de ejecución es un proceso largo, dado que se produce un cuello de botella, bien en la red, o en el acceso a disco. En el resto de casos este tiempo se solapa con todas las tareas anteriores, de tal manera que dicho tiempo no aparece.

En los dos primeros casos, como en las pruebas anteriores, la fase de mayor tiempo es la de configuración del sistema. Pero en la prueba de 11 nodos la fase de espera de arranque de las MVs supera este tiempo. Esto se produce porque la imagen de la MV de Windows es de mayor tamaño que el resto de IMVs utilizadas en pruebas anteriores (5 GB). Esto provoca un cuello de botella a la hora del acceso a la red y disco en el proveedor Cloud de OpenNebula en la copia de los 50 GB totales. Esta sobrecarga del sistema también afecta en la primera fase puesto que el arranque de la MV "master", porque aunque la imagen de la MV master sea de menor tamaño también se ve afectada por este cuello de botella. Mostrándose con más claridad en el caso de 11 MVs lanzadas. En el caso de EC2 no se produce este problema dado que es una gran infraestructura y además las imágenes utilizadas son de tipo EBS.

Tabla 7. Tiempos de creación de las infraestructuras virtuales del caso XWCH

	1 nodo ONE	6 nodos ONE	11 nodos ONE	1 nodo EC2	11 nodos EC2
Master arrancada	1:39	2:29	8:45	0:42	0:47
Master accesible	1:28	1:47	3:42	0:29	0:27
Ansible configurado	1:37	2:12	4:11	3:38	3:12
MVs Arrancadas	0:00	5:46	16:33	0:00	0:00
Sistema configurado	5:43	6:52	8:58	5:43	6:51
TOTAL Creación	10:27	13:08	42:09	10:32	11:17
Nueva MV arrancada	2:08	3:29	4:31	2:08	2:40
Reconfiguración	2:05	2:27	3:13	2:45	3:31
TOTAL Adición	4:13	5:56	7:44	4:53	6:11
MV eliminada	0:23	0:20	0:30	1:43	1:41
Reconfiguración	1:21	1:41	1:38	1:57	1:33
TOTAL Eliminación	1:44	2:01	2:08	3:40	3:14

5.4 Servicios de alto nivel

A parte de utilizar el servicio del IM de forma independiente para la creación y gestión de infraestructuras de forma directa por parte de los usuarios. El IM puede ser utilizado por otro software para proporcionales dicha funcionalidad, a través de su API, para acceder a su funcionalidad.

5.4.1 CodeCloud

El objetivo principal del proyecto CodeCloud [57], [64] es el de facilitar el desarrollo y puesta en funcionamiento de aplicaciones científicas en el Cloud, para que el esfuerzo necesario para ponerlas en funcionamiento se reduzca drásticamente.

Para ello propone la creación de servicios y componentes de alto nivel que abstraigan a los usuarios de los aspectos específicos del Cloud, proporcionando mecanismos que permitan desplegar infraestructuras científicas en el Cloud, de forma sencilla.

Se identifican los modelos de programación más habituales en la computación científica para gestionar el ciclo de vida de aplicaciones científicas, automatizando

los aspectos necesarios que faciliten la utilización de los datos de los usuarios en el Cloud.

En la Figura 29 se muestra el esquema funcional de CodeCloud. Básicamente el usuario define un trabajo con un lenguaje en XML denominado Cloud Job Description Language (CJDL). El usuario definirá todos los aspectos necesarios para lanzar su aplicación: la infraestructura necesaria (en RADL), los datos de entrada, de salida, ejecutables, etc. El IM será el encargado de lanzar toda la infraestructura, empezando por la MV de contenedor, y todas las MVs definidas por el usuario.

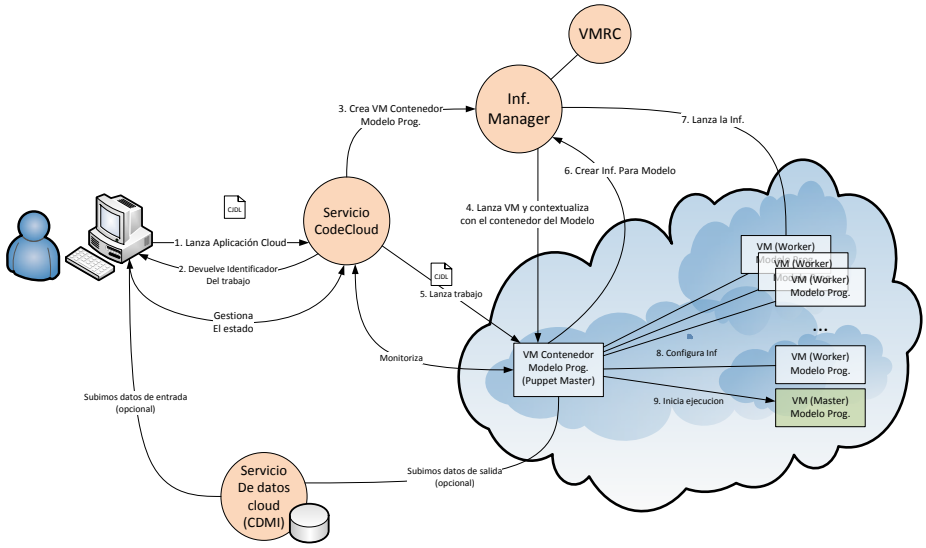

Figura 29. Esquema funcional de CodeCloud.

En este caso, a diferencia del IM "general", la parte de contextualización se encuentra separada del IM, y se ha integrado dentro del contenedor de los modelos de programación. De tal manera que el IM, solo lanza la infraestructura y es el contenedor el encargado de contextualizarla.

En el caso de CodeCloud se utilizan las funciones de adición y eliminación de recursos a infraestructuras para gestionar la elasticidad de forma automática. En la Figura 30 se muestra un esquema del funcionamiento del gestor de elasticidad

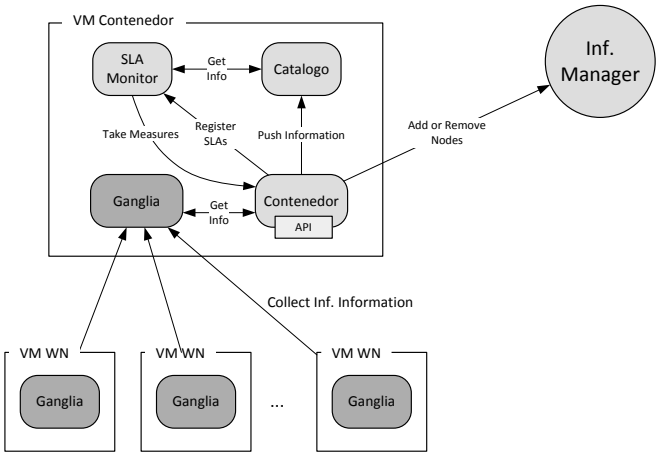

Figura 30. Esquema funcional del gestor de elasticidad

Para la monitorización de los nodos de la infraestructura se utiliza Ganglia (como hemos visto en el apartado 4.2.5a). Por tanto al contextualizar se instala en todos los nodos y se configuran para que el contenedor sea el nodo "master" donde se almacena la información combinada. Toda esa información es expuesta por el contenedor en forma de pares clave, valor. A los valores obtenidos con Ganglia (de momento el uso de CPU, uso de memoria y uso de disco) le añadirá una serie de valores propios del contenedor: % de ejecución, tiempo de ejecución, etc. Esta información será expuesta a través del API del contenedor de manera que pueda ser consultada por las propias aplicaciones lanzadas (deberían estar correctamente instrumentadas, es decir modificadas para poder acceder al API del contenedor). Además ese API también permite que las aplicaciones publiquen información propia (p.e. el valor de una variable) de manera que dicho valor pueda ser utilizado en las reglas de elasticidad.

Además de forma periódica toda esta información será enviada al Catálogo usando su interfaz REST. El catalogo irá almacenando de forma histórica esta información. Al inicio del contenedor, éste deberá registrar las reglas de elasticidad en el monitor. El monitor periódicamente evaluará estas reglas consultando la información histórica almacenada en el contenedor tomando las medidas necesarias, añadir/eliminar nodos (scale in/out) o aumentar/disminuir tamaño de las MVs (scaleup/down). Para ello deberá conectar con el contenedor para que aplique esas medidas, conectando con el Inf. Manager, para replegar/desplegar nodos y reconfigurando el resto de nodos de acuerdo al cambio de la infraestructura.

Para gestionar la elasticidad, el usuario deberá establecer una serie de criterios de elasticidad para que el sistema sea capaz de aumentar o decrementar el número de MVs en la infraestructura o de aumentar las capacidades de las MVs. Los criterios pueden ser de varios tipos:

- Elasticidad a nivel "Hardware". Métricas que dependen de los valores obtenidos en las MVs:
    - % de uso de CPU: *cpu.usage*
    - Memoria RAM libre: *memory.free*
    - Espacio en disco libre: *disk.free*
- Elasticidad dependiente del contenedor del modelo:
    - Valores generales de la ejecución:
        - Tiempo de ejecución: *execution.time*
        - Porcentaje de finalización: *execution.pct_finish*
    - Master/Slave: criterios propios de este modelo
        - Número de tareas:
            - en cola: *tasks.queued*
            - en ejecución: *tasks.running*
            - finalizadas: *tasks.finished*
            - totales: *tasks.total*
    - Workflow: criterios propios de este modelo. En este caso para cada una de las tareas del workflow podríamos obtener:
        - Estado (esperando, cerrada, en ejecución, etc.): task_name.state
        - Número de ejecuciones:
            - sin lanzar: *task_name.executions.waiting*
            - ejecutando: *task_name.executions.running*
            - finalizadas: *task_name.executions.finished*
            - total: *task_name.executions.total*
- Elasticidad dependiente de la propia aplicación del usuario:
    - En este caso se proporcionará un sistema para que el propio desarrollador de las aplicaciones pueda instrumentarlas y proporcionar sus propias métricas sobre las que luego definir las reglas de elasticidad.

A continuación se define un lenguaje para poder especificar los criterios sobre los que se gestionará la elasticidad en base a las métricas anteriores. El lenguaje va a tener la siguiente forma:

```
if <regla> then <acción> [for <filtro>]
```

El lenguaje define las reglas de elasticidad en base a tres elementos:

- **Regla**: condición que debe cumplirse para que se desencadene la ejecución de la regla. Las condiciones se componen de un conjunto de expresiones de evaluación unidas por un conector booleano (and u or). Las expresiones utilizan los valores proporcionados por las distintas métricas del sistema. Por "problemas" con la sintaxis del XML que provoca que no se puedan usar los símbolos < y >, para los comparadores vamos a usar las cadenas de texto: eq (=), ne(!=), gt(>), lt(<), ge(>=), le(<=).

  Dentro de las métricas es posible diferenciar dos grupos: los valores generales y los valores propios de las MVs de la infraestructura. Además en las métricas se ha de tener en cuenta que el sistema almacena valores históricos. Por tanto vamos a tener los valores generales que serán de tipo vector de tamaño n (siendo n el número de valores históricos almacenados) y los valores relacionados con las MVs que serán de tipo matriz de tamaño n*v (siendo v el número de MVs de la infraestructura). Solo hay un caso especial en la variable "execution" que no pertenece a ninguna MV sino que es un valor general de la ejecución, por tanto su tamaño será de n.

  Por tanto hay que usar funciones para reducir esos conjuntos de valores a valores únicos para realizar las comparaciones. Para ello se han definido tres funciones: min, max, avg, un operador: {<num>} y un operador [filter] para el filtrado por tipo. Con las tres operaciones se hace una reducción de los valores en una dimensión, por tanto para el caso de los valores de las MVs se deberá usar dos funciones para obtener un valor simple. En cuanto al operador {<num>}, permitirá seleccionar los "num" últimos elementos históricos del valor que precede al operador. Por último el filtrado por tipo permite seleccionar de un conjunto de valores de las MVs, solo aquellos de las MVs de un determinado tipo. Para ello se indica poniendo entre paréntesis el tipo de la MV después de la métrica a filtrar.

  Por ultimo indicar una "facilidad" del lenguaje: en caso en que el valor final no se haya reducido a un valor simple y no se haya usado ya el operador {}, se dará por supuesto que se utiliza dicho operador con el valor {1}, es decir que se utilizará para comparar el ultimo valor almacenado.

- **Acción:** Acción correctiva que debe ejecutar el sistema cuando la regla de elasticidad se cumpla. Las acciones a realizar son la 4 básicas dentro de los parámetros de elasticidad horizontal y vertical:

- Horizontal: Aumentar o disminuir el número de recursos a la infraestructura virtual.
  - ScaleOut: Añade recursos.
    - Parámetros:
      - Numero de recursos a añadir.
      - Tipo de recursos a añadir.
      - Número máximo de veces que se ejecuta dicha operación (Opcional). En caso de no indicarse no hay dicho máximo.
  - ScaleInOut
    - Parámetros:
      - Tipo de recursos a eliminar.
      - Número de recursos a eliminar.
      - Número máximo de veces que se ejecuta dicha operación (Opcional). En caso de no indicarse no hay dicho máximo.
- Vertical: Aumentar o disminuir las capacidades de un conjunto de MVs de la infraestructura virtual.
  - ScaleUp: Aumentar las capacidades.
    - Parámetros:
      - Capacidad de la MV: Memoria o CPU.
      - Cantidad a añadir.
  - ScaleDown
    - Parámetros:
      - Capacidad de la MV: Memoria o CPU.
      - Cantidad a eliminar.

- **Filtro**: Este elemento opcional permite que se elijan el conjunto de nodos sobre los que se van a tomar las acciones definidas en el apartado anterior (en el caso que sea aplicable). En caso de no indicarse, si la regla implica selección de nodos, por defecto se seleccionarán todos los nodos de la infraestructura. Para el filtrado de nodo se usará una sintaxis idéntica a la de la regla de elasticidad (con el mismo tipo de operadores) pero que en

este caso serán aplicados a cada una de las MVs de la infraestructura para comprobar si cumplen o no dicha condición. La única diferencia es que se puede añadir el prefijo "vm." delante de cualquier métrica de las MVs para indicar que nos referimos a la de la MV actual en la comparación. Por tanto al usar dicha métrica reducimos la dimensión de este tipo de valores para convertirla en un vector temporal.

Ejemplos:

- Si el valor medio de la media del uso de CPU de las máquinas de tipo "vm_type1", durante los últimos 5 periodos de tiempo es mayor que el 90 % despliega 2 máquinas de tipo "vm_type1"

```
if avg(avg(cpu.usage[vm_type1]{5})) gt 90% then ScaleIn(2, vm_type1)
```

- Si el valor medio del mínimo espacio de memoria libre de todas las máquinas durante los últimos 2 periodos de tiempo es menor de 50MB despliega 2 máquinas de tipo "vm_type1"

```
if avg(min(memory.free{2}) lt 50MB then ScaleIn(2, vm_type1)
```

- Si la aplicación lleva el 50% y han pasado más de 10 minutos de tiempo de ejecución despliega una máquina de tipo "vm_type1" y otra de tipo "vm_type2"

```
if execution.pct_finsh lt 50% and execution.time 10:00 then ScaleIn([1,2], [vm_type1, vm_type2])
```

- Si la aplicación lleva el 80% y han pasado menos de 5 minutos de tiempo de ejecución elimina dos máquinas de tipo "vm_type1".

```
if execution.pct_finsh > 80% and execution.time < 5:00 then ScaleOut(2, vm_type1)
```

- Si el mínimo de la media de memoria libre en los 5 últimos periodos es menos de 50 MB, a aquellas que tenga de media en los 5 últimos periodos, menos de 50 MB libres añádele 512 MB de memoria más.

```
if min(avg(memory.free{5})) lt 50MB then ScaleUp(512M, memory) for avg(vm.memory.free{5}) lt 50MB
```

- Si el máximo de memoria libre de los 5 últimos periodos es mayor de 512MB, a aquellas que tengan, de media en los 5 últimos periodos, más de 512 MB de memoria libre quítale 256 MB de memoria.

```
if max(avg(memory.free{5})) gt 512MB then ScaleDown(256MB, memory)
for avg(vm.memory.free{5}) gt 512MB
```

5.4.2 EC3: Elastic Cloud Computing Cluster

En el caso de EC3 [65] se utiliza el IM junto con un software de gestión de energía para clústeres de altas prestaciones (HPC) denominado CLUES [66] para el lanzamiento y gestión de forma elástica de clústeres de PCs sobre plataformas Cloud.

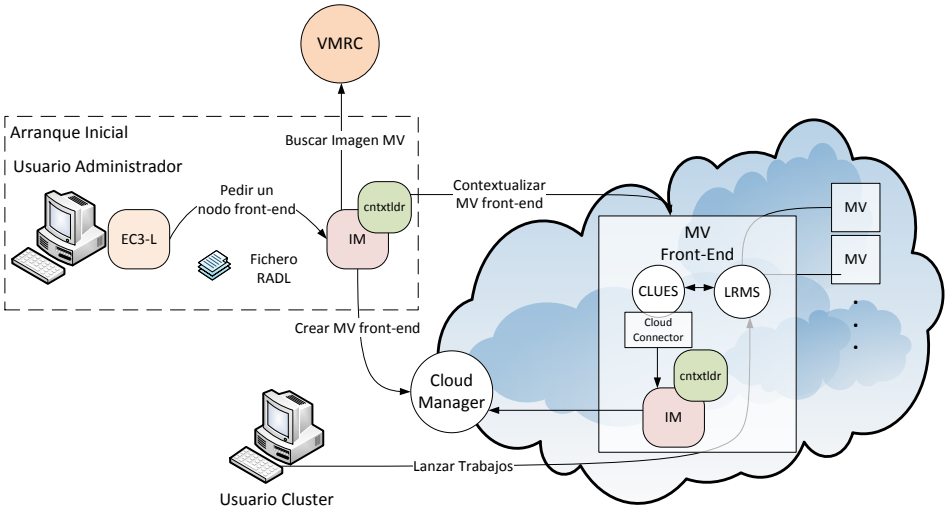

Figura 31. Arquitectura de EC3

En este caso se utiliza la potencia de IM para el lanzamiento de infraestructuras en el Cloud y la facilidad de gestión del tamaño de los clústeres, bajo demanda y de forma automática que proporciona CLUES. En la Figura 31 se muestra la arquitectura de EC3 y cómo interactúan los componentes de CLUES y del IM.

CLUES y el IM interactúan a dos niveles, primero en el lanzamiento inicial de la MV front-end del servidor elástico. En este paso el IM se encarga de lanzar la MV al proveedor Cloud deseado y de contextualizarla con el software del sistema LRMS elegido por el usuario y con CLUES totalmente configurado para dicho LRMS y para la interacción con el Cloud. El segundo nivel de interacción es a través de un conector Cloud desarrollado para que CLUES en vez de apagar y

encender nodos físicos, lance o termine MVs sobre plataformas Cloud usando el IM.

5.4.3 TRENCADIS

TRENCADIS [67] es un middleware Grid que proporciona acceso a repositorios federados de imágenes médicas DICOM (Digital Imaging and Communication in Medicine) e informes DICOM-SR, con el objetivo de facilitar la creación de conocimiento médico. Es uno de los pocos proyectos que combinan la utilización de tecnologías Grid y DICOM-SR para compartir anotaciones sobre imágenes médicas en entornos interinstitucionales.

El middleware TRENCADIS demostró efectividad a la hora de crear una base de conocimiento distribuido, pero también expuso la necesidad de contar con un soporte más eficiente, que requiera menos intervención de los administradores locales, facilitando las tareas de despliegue. Otra característica de TRENCADIS que debe ser mejorada, es la tolerancia a fallos. Para ello, es necesario mejorar la disponibilidad de los servicios y su recuperación cuando se produce algún fallo, lo cual es crítico en aplicaciones para salud.

La tecnología Cloud permite desplegar recursos bajo demanda, de una forma poco intrusiva. Esta característica es utilizada para mejorar el despliegue de los servicios de TRENCADIS y su disponibilidad, mediante la inclusión de nuevos componentes en su arquitectura para adecuar los servicios TRENCADIS a las tecnologías Cloud.

Por todo esto, la nueva arquitectura desarrollada en [68] esta ideada con el objeto de desplegar la infraestructura TRENCADIS en entornos Clouds públicos, privados o híbridos, posibilitando la virtualización de los servicios y aprovisionando estos de forma elástica y dinámica cuando se requiera. Los componentes utilizados se describen a continuación:

- **VMRC:** Se utiliza como la capa de menor nivel, sobre la que se basa la plataforma. Se encarga de catalogar las IMVs que tienen instalados S.O y componentes software básicos de los servicios TRENCADIS
- **Interfaz Web del IM:** Se utiliza como repositorio de un conjunto de documentos RADL que describen la composición y configuración de los servicios TRENCADIS. Estos documentos especifican la MV requerida para un servicio, o un conjunto de servicios como los que engloba el TRENCADIS Center.
- **Infrastructure Manager:** El IM se utiliza como componente para el despliegue efectivo de todas las MVs necesarias para la plataforma TRENCADIS descritas en los documentos RADL almacenados en el interfaz web.

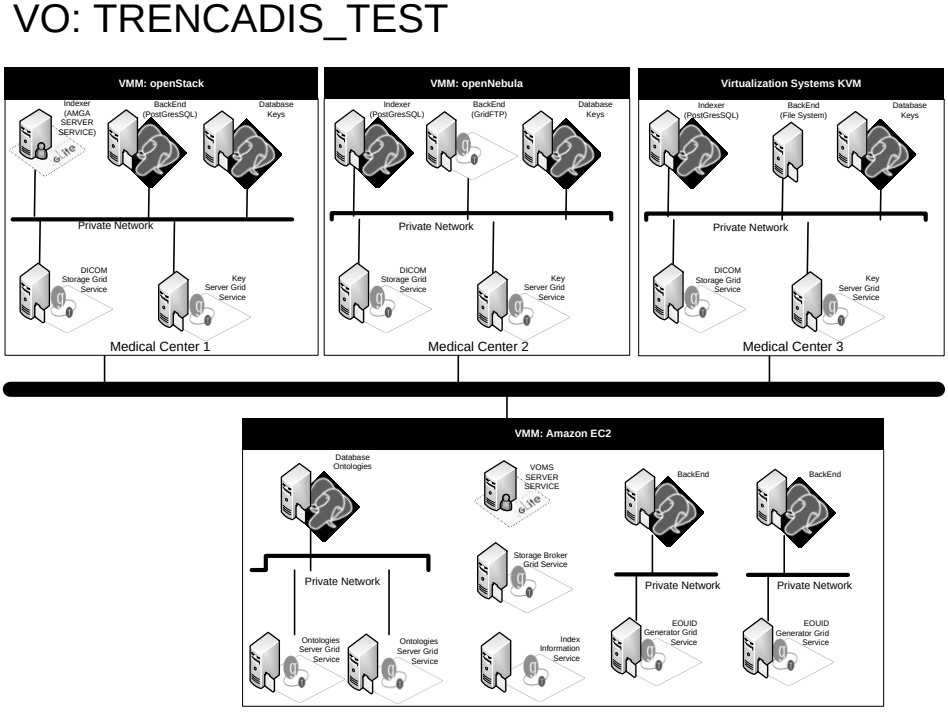

Figura 32. Despliegue infraestructura TRENCADIS.

En la Figura 32 se muestran los servicios que deben ser desplegados para conformar una infraestructura TRENCADIS completa para la VO: TENCADIS_TEST. Está formada por los servicios CORE y los SERVER. Los servicios CORE lo conforman dos tipos de servicios, que son los DICOM Storage Services y Storage Key Services. Estos servicios tienen que ser desplegados, al menos uno de cada tipo, en cada centros médicos involucrado en un despliegue TRENCADIS (Hospitales, Centros Especialidades etc.). Los servicios SERVER lo conforman un conjunto de cinco servicios, de los que solo se necesita desplegar una sola instancia. Los servicios SERVER no necesitan de su instalación en los centros médicos, únicamente en un centro (TRENCADIS Center) con lo que las restricciones de red suelen ser menores, de tal manera que puedan ser desplegados tanto en un Cloud privado como en uno público.

Como caso de uso se ha utilizado el lanzamiento de toda la infraestructura necesaria en el TRENCADIS Center mediante el IM (RADL completo incluido en el Anexo I). Dado que el TRENCADIS Center no tiene restricciones a la hora de su ubicación, se ha probado tanto en el Cloud privado de OpenNebula como en el público de EC2. La Tabla 8 muestra los tiempos necesarios para la creación de la infraestructura. Además el middleware TRENCADIS utiliza las facilidades

proporcionadas por el IM para la gestión elástica de la infraestructura, mostrando la información de monitorización de la infraestructura (como se indica en el apartado 4.2.5a) para comprobar el estado y lanzar o eliminar instancias de servicios dependiendo de la carga del sistema. En concreto se han monitorizado las instancias del servicio EOUID Generator, para que en caso de aumento de la carga (si el uso de CPU es superior al 80% durante 5 min) se lanzará una nueva instancia. En caso de una disminución de la carga (uso de CPU menor del 20% durante 10 minutos) se eliminará una instancia del servicio, siempre dejando al menos una instancia. La Tabla 8 muestra también los tiempos necesarios para el despliegue y eliminación de instancias del servicio EOUID Generator.

Tabla 8. Tiempos de creación del TRENCADIS Center.

	ONE	**EC2**
Master accesible	6:43	1:55
Ansible configurado	1:28	4:00
MVs Arrancadas	0:00	0:00
Sistema configurado	7:14	7:07
TOTAL Creación	15:25	13:02
Nueva MV arrancada	1:49	1:55
Reconfiguración	5:39	5:38
TOTAL Adición	7:28	7:33
MV eliminada	0:32	0:18
Reconfiguración	2:28	2:35
TOTAL Eliminación	3:00	2:43

5.4.4 Plataforma docente ODISEA

En caso de la plataforma ODISEA (On-demand Deployment of virtual Infrastructures to Support Educational Activities) utiliza el IM para crear infraestructuras virtuales avanzadas para la docencia universitaria, cursos, seminarios de postgrado, etc. tanto en Clouds públicos como privados.

En este caso se han distinguido dos tipos de infraestructuras: Las primeras la hemos denominado “Base Educational Computing Infrastructures” (BECI). Este tipo corresponde a despliegues que tienen una duración “larga”, es decir toda la duración de un curso. Las segundas las hemos denominado “Ad-Hoc Educational Computing Infrastructures” (AHECI) y hacen referencia a infraestructuras de menor duración y

que son desplegadas y modificadas de forma muy dinámica, como puede ser para hacer una práctica concreta de una asignatura.

La Figura 33 muestra de forma esquemática los pasos necesarios en la interacción con la plataforma ODISEA. Inicialmente el profesor debe analizar la infraestructura necesaria, tanto en términos de hardware como software, para impartir sus clases. El profesor pasará dichos requerimientos al administrador del sistema (*sysdamin*), que puede ser el mismo profesor en caso de tener los conocimientos técnicos necesarios. El administrador deberá comprobar si existe alguna IMV en el repositorio que se adapte a las necesidades indicadas, y en caso contrario crear una nueva y registrarla de forma adecuada en el VMRC. El administrador deberá identificar que software debe ser pre-instalado en la imagen, dada su complejidad o larga duración de la instalación, y cuál puede ser instalado en la fase de contextualización de la MV. El administrador deberá crear el documento RADL con la descripción de la infraestructura y todos los detalles de contextualización.

Las descripciones de las Infraestructuras se ponen a disposición del profesor a través de la interfaz web de IM. A través de la interfaz web el profesor podrá desplegar la infraestructura con un simple click del ratón.

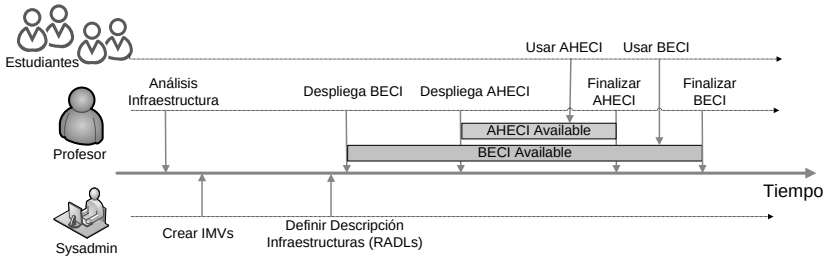

Figura 33. Línea temporal de uso de la plataforma ODISEA

La Figura 34 muestra la arquitectura de la plataforma ODISEA. El componente central es el IM encargado del lanzamiento y gestión de las infraestructuras virtuales. El IM usará el conjunto de documentos RADL almacenados en el denominado repositorio de descripciones infraestructuras virtuales (RIDs), seleccionará la mejor IMV del repositorio IMVs (RVMIs). Una vez lanzadas las MVs, las contextualizará tomando tanto los paquetes software como otro datos necesarios para la configuración de los repositorio de software (SR) y de datos educacionales (EDR). Finalmente el componente denominado Meta-infrastructure Manager (MM) se encargará de organizar y programar el lanzado y eliminado de las infraestructuras según la programación indicada por los usuarios.

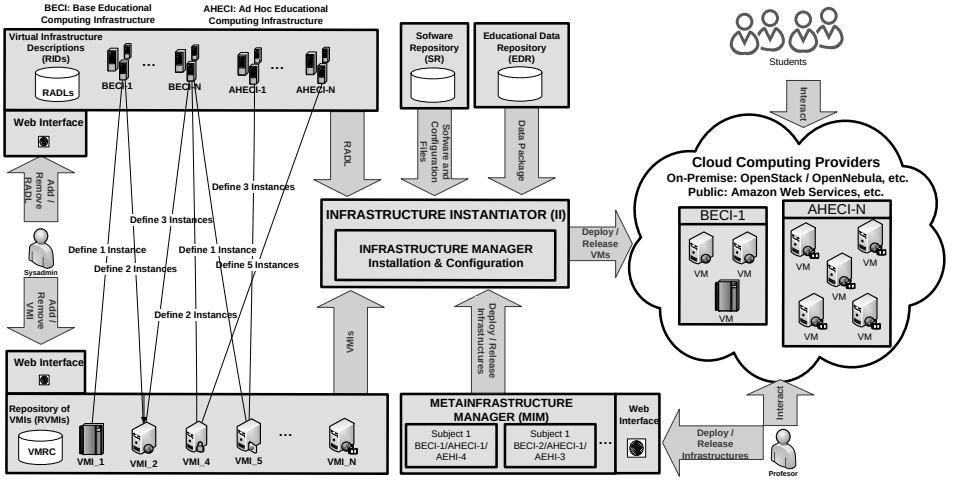

Figura 34. Arquitectura de la plataforma ODISEA

6 Conclusiones y Trabajos Futuros

6.1 Resumen y Contribuciones Principales

El objetivo de esta tesis ha sido avanzar en el estado del arte de la gestión elástica de infraestructuras Cloud ad-hoc. La tesis ha dado como resultado una plataforma software para el despliegue y gestión elástica de infraestructuras sobre proveedores Cloud. Esta plataforma ofrece un interfaz suficientemente sencillo para su uso por usuarios no avanzados en conocimientos informáticos, pero ofrece la suficiente flexibilidad y potencia para que pueda ser explotada por usuarios más expertos.

Para ello se desarrolló inicialmente el lenguaje RADL, tanto para expresar requerimientos sobre infraestructuras virtuales, como para mostrar información sobre las mismas. Dicho lenguaje permite no solo expresar requerimientos a nivel de hardware sino también a nivel de software y de configuración. Dicho lenguaje permite definir las infraestructuras de manera independiente de la plataforma de virtualización subyacente permitiendo usar el mismo RADL en cualquier proveedor Cloud. Esta característica supone un avance con respecto al estado del arte.

El RADL permite especificar la información relevante para el usuario, de manera que el sistema que lo interpreta complete el fichero con los elementos necesarios para tener la definición de una MV completa. Pero también permite, en el caso de usuarios más avanzados, expresividad suficiente para poder expresar todos los requisitos para personalizar completamente la infraestructura, lista para utilizar con sus aplicaciones.

En concreto en el RADL se ha incluido un apartado específico para la configuración de las infraestructuras, una vez lanzadas. En dicho apartado se ha utilizado el lenguaje YAML definido por la herramienta de contextualización Ansible donde se puede definir de forma independiente del sistema operativo todas las tareas de configuración necesarias para preparar la infraestructura para el uso por parte de la aplicación científica. De nuevo esta característica supone un avance con respecto al estado del arte.

Para la catalogación y almacenamiento de imágenes de máquinas virtuales se ha desarrollado el componente VMRC. Dicho sistema permite la búsqueda y selección de la mejor imagen de máquina virtual disponible, según las necesidades del usuario. También permite catalogar toda la información de las IMVs (formato de la imagen, S.O., aplicaciones instaladas, etc.) que pueda ser necesaria para el usuario.El servicio VMRC permite la compartición de IMVs de tal manera que se favorece la reutilización de las mismas, evitando que los usuarios tengan que crear nuevas imágenes para cada aplicación, favoreciendo la reutilización. Si bien existen otros sistemas de catálogo de IMVs, VMRC proporciona mucha mayor

expresividad y flexibilidad. El VMRC se ha puesto a disposición de la comunidad bajo licencia "Apache License, Version 2.0" a través de la web: http://www.grycap.upv.es/vmrc.

Finalmente el componente principal es el IM que permitir el despliegue efectivo de las infraestructuras sobre el mayor número posible de plataformas Cloud. El IM se ha diseñado de forma que sea sencilla su extensión en el futuro para añadir nuevas plataformas. El IM se encarga de interpretar el RADL y coordinar el lanzamiento y posterior contextualización de la infraestructura. El IM también se ha puesto a disposición de la comunidad bajo licencia "GPL 3.0" a través de la web: http://www.grycap.upv.es/im.

El IM proporciona capacidades de contextualización de las infraestructuras en tiempo de ejecución para instalar y configurar el software necesario para las aplicaciones científicas usando la herramienta Ansible. Dicha contextualización no necesita de ningún tipo de software pre-cargado en las IMVs, facilitando la reutilización de las mismas. Además para facilitar ciertas tareas de configuración desde Ansible a la hora de preparar las "recetas" de contextualización el IM proporciona una serie de variables con información sobres las MVs.

Dado que la reutilización es una de las características primordiales de la plataforma, para facilitar la reutilización de las recetas de Ansible, e integrarlas dentro de los apartados de contextualización de los RADLs, el IM permite la descarga de recetas de Ansible almacenadas en GitHub de forma automatizada. El usuario solo indica el repositorio donde se encuentra y la plataforma se encarga de la descarga en el lugar adecuado para poder ser utilizado desde el RADL.

La plataforma proporciona una característica muy importante como es la "repetibilidad". El lenguaje RADL permite definir las infraestructuras de manera independiente de las capas de virtualización. El IM permite interpretar un documento RADL de tal manera que el conjunto del MVs producido sea el mismo, independientemente de la plataforma Cloud subyacente, e independientemente del momento que se realice. Esto permite que el investigador pueda acceder al conjunto de MVs siempre en las mismas condiciones de uso sin necesidad de mantener las MVs en ejecución todo el tiempo.

Para facilitar la gestión de la elasticidad a capas de software superiores. El IM permite una integración con Ganglia para proporcionar información de monitorización sobre las infraestructuras virtuales. Proporcionando dicha información en el mismo formato RADL que el resto de información de las MVs. Esto permite que las capas superiores deleguen esta tarea en el IM pudiéndose centrar en la gestión propia de la elasticidad como ocurre en el caso de uso de CodeCloud.

Para demostrar la versatilidad de la plataforma así como la diversidad de casos en los que puede ser utilizada se han mostrado un amplio abanico de casos de uso. En ellos se demuestra su utilidad tanto en el caso del uso directo por parte de los usuarios, con las herramientas de cliente, como el caso de uso de las APIs para el acceso de la funcionalidad por otras capas software. En este último caso se ha mostrado su utilidad en plataformas muy diversas como la de creación de clústeres virtuales elásticos, en facilitar el desarrollo y puesta en funcionamiento de aplicaciones científicas en el Cloud o el uso en plataformas docentes.

En el caso de despliegue de infraestructuras se han probado diferentes casos de uso con diferente complejidad a la hora de configurar las infraestructuras, así como diferente número de recursos a lanzar. Se ha mostrado la escalabilidad de la plataforma en dichas pruebas permitiendo el lanzamiento de un número importante de MVs simultaneas (50) obteniendo razonables tiempos de respuesta.

6.2 Trabajos Futuros

Una de las tareas lógicas a la hora de continuar con el trabajo del IM es añadir nuevos tipos de proveedores para permitir abarcar el mayor número de infraestructuras posibles. Dentro de esta línea Eucalyptus sería uno de los objetivos iniciales ya que al proporcionar un API compatible con la de Amazon EC2, su incorporación sería casi inmediata. Actualmente se encuentra muy avanzado el plugin para el acceso a Azure y se planea continuar con la plataforma Google Cloud para completar el "trio" de grandes empresas del sector. Además se pueden probar otras infraestructuras como: Linode, Rackspace, etc.

Otra de las líneas de mejora del IM es relativa al soporte a la creación de redes de interconexión complejas. Dependiendo del soporte que proporcione cada infraestructura Cloud se podrá, o no, proporcionar unas características más avanzadas. En el caso de infraestructuras de tipo público como EC2 (con Virtual Private Cloud) o Azure (con Virtual Network) proporcionan soporte para la creación de todo tipo de redes complejas.

En la descripción del componente Cloud Selector se indican diferentes alternativas existentes a la hora de la selección del mejor proveedor Cloud a la hora de lanzar una infraestructura virtual. En la versión actual se ha elegido una opción simple para soluciones de tipo Cloud bursting. Se pueden analizar diferentes tipos de selección de Clouds, buscando el mejor ratio prestaciones/precio, permitiendo al usuario definir mínimos y máximos, tanto en prestaciones como en precio, etc. Para ello se deben estudiar sistemas automatizados para la obtención de precios de las diferentes plataformas Cloud (Cloudymetrics [69], CloudHarmony[51], etc.), así como

[51] http://cloudharmony.com

el estudio de las denominadas "spot instances"[52] de EC2 y otro tipo de instancias de precio variable.

Una funcionalidad interesante a añadir al IM es la de permitir registrar, de forma automática, las imágenes de las MVs lanzadas y contextualizadas (tanto en el despliegue Cloud como en el VMRC) para permitir que en posteriores lanzamientos de la misma MV, tenerla ya totalmente configurada y permitir un lanzamiento más rápido, al evitar (o al menos reducir al minimo) la contextualización. Esta funcionalidad podría ser a demanda del usuario, o bien por elección del IM al detectar que el proceso de contextualización es "largo" y podría compensar este registro.

Otra área a trabajar es la creación de infraestructuras distribuidas en diferentes tipos de Clouds, en los casos en los que haya redes privadas que unan las diferentes MVs. En la actualidad el IM obliga a que todas las MVs unidas por una misma red privada deban estar en el mismo despliegue Cloud. Se pueden usar alternativas como el uso de Virtual Private Networks (VPNs) para permitir en estos casos que los nodos estén distribuidos entre diferentes plataformas Cloud.

En cuanto a las herramientas de contextualización, en los últimos meses han aparecido nuevas herramientas como SaltStack[53] o Rexify[54]. Se pueden analizar estas nuevas herramientas comprobando si mejoran los resultados producidos por la herramienta usada en la actualidad: Ansible.

En cuanto a la parte de interfaz de usuario, se está trabajando tanto en la mejora del aspecto gráfico como en la mejora de la funcionalidad. En cuanto a la parte gráfica se está mejorando el aspecto general para hacerlo más amigable con el usuario y en la definición de forma gráfica de partes del RADL (número de CPUs, tipo de CPU, memoria, interfaces de red, aplicaciones, etc.). En cuanto a la parte funcional se está trabajando para permitir la planificación automática de lanzamiento/destrucción de infraestructuras. Esto facilitaría la tarea en el caso de infraestructuras cuyo tiempo de vida esté claramente acotado en el tiempo, como por ejemplo las MVs necesarias para una práctica de un máster.

[52] http://aws.amazon.com/ec2/spot-instances/

[53] http://www.saltstack.com

[54] www.rexify.org/

7 Proyectos de investigación asociados

Esta tesis se ha desarrollado dentro del marco del proyecto denominado: "Servicios avanzados para el despliegue y contextualización de aplicaciones virtualizadas para dar soporte a modelos de programación en entornos Cloud" con acrónimo CodeCloud y referencia TIN2010-17804 del inicialmente denominado Ministerio de Ciencia e Innovación y después Ministerio de Economía y Competitividad.

También ha participado en el ámbito del proyecto "Gestión Elástica y Eficiente de Entornos Cloud para la Ejecución de Aplicaciones Científicas" con acrónimo GE3CEAC y referencia GV/2012/076 de la Generalitat Valenciana.

En los siguientes apartados de este capítulo se describe con más detalle cada uno de los proyectos, además de resaltar los resultados que están más directamente relacionados con los desarrollos de la tesis.

7.1 CodeCloud

El objetivo general de este proyecto es facilitar el despliegue de aplicaciones en entornos Cloud, de manera que el esfuerzo requerido para adaptar aplicaciones a estos entornos pueda reducirse drásticamente. Esto permitiría ampliar el número de aplicaciones científicas que se beneficiarían de las infraestructuras Cloud, y reducir el tiempo del ciclo de producción. En particular, el proyecto se centra en dos objetivos:

- Proporcionar servicios de alto nivel para aprovechar las características de las plataformas Cloud desde el punto de vista del desarrollador de aplicaciones.
- Proporcionar mecanismos automáticos para construir AVs, considerando el proceso completo, desde la identificación de MVs adecuadas hasta su contextualización para la aplicación de usuario.

Los componentes que permitirán alcanzar los objetivos son los siguientes:

- Un repositorio de MVs que asociará metadatos a las imágenes, permitiendo la selección automática de recursos de acuerdo a reglas predefinidas.
- Un Lenguaje de Descripción de Aplicaciones y Recursos, que permitirá especificar los requerimientos de las aplicaciones sobre las MVs donde se ejecutarán.
- Un contextualizador, que permitirá adaptar las imágenes de las MVs a los requerimientos específicos de las aplicaciones, desplegando los componentes necesarios para su uso.

- Un gestor de ejecución en el Cloud, que permitirá orquestar los recursos existentes para poder ejecutar diferentes tipos de aplicaciones mediante modelos bien definidos y en coordinación con el repositorio de MVs.

Un conjunto de herramientas y componentes para aplicaciones basadas en "workflows", maestro/esclavo y MPI, adaptado para su uso en el Cloud.

7.1.1 Aportaciones

Los resultados obtenidos relacionados con los trabajos de ésta tesis son los siguientes:

- El lenguaje RADL definido permite especificar los requerimientos de las aplicaciones sobre las MVs donde se ejecutarán.
- Usando dicho lenguaje permite el lanzamiento efectivo de las MVs de forma agnóstica en diferentes proveedores Cloud.
- El Configuration Manager empleando la utilidad Ansible permite adaptar las imágenes de las MVs a los requerimientos específicos de las aplicaciones, instalando y configurando todo el software necesario.

7.2 GE3CEAC

Este proyecto plantea la gestión de infraestructuras Cloud para la ejecución de aplicaciones científicas de forma elástica con el objetivo de realizar un mejor aprovechamiento de los recursos (computacionales y energéticos) en los diferentes niveles involucrados (desde la propia aplicación hasta la infraestructura física). La gestión de la elasticidad se realizará de forma transversal a todos estos niveles de manera que la infraestructura subyacente se ajuste a los requisitos de ejecución dinámicos que demande la aplicación.

La Figura 35 resume la arquitectura principal del sistema de gestión elástica propuesto. El usuario dispone de su aplicación junto con sus requisitos computacionales (librerías, Sistema Operativo, calidad de servicio (QoS)). La ejecución de la aplicación se delega en un componente denominado Cloud Enactor que se encarga de orquestar todos los servicios subyacentes para conseguir la ejecución de la aplicación sobre la plataforma Cloud. Esto requiere acceder al catálogo de imágenes de máquinas virtuales para buscar la imagen más apropiada en base a los requisitos de la aplicación y obtenerla. Si es necesario, se procede a la fase de contextualización para desplegar la aplicación junto a sus requisitos en la(s) maquina(s) virtual(es). La ejecución de la aplicación puede requerir los servicios de gestión elástica ofrecidos por el Cloud Enactor para realizar un escalado horizontal de la aplicación (aumentando el número de máquinas virtuales) o un escalado vertical de la misma (aumentando las capacidades de las máquinas virtuales). Esto implica el desarrollo de interfaces y mecanismos que utilicen y coordinen las tecnologías habilitantes de elasticidad desde el punto de vista de las aplicaciones

para facilitar su uso. En el nivel de infraestructura, resulta necesario un gestor energético (Green-Aware Manager) que monitorice el estado de la infraestructura física para detectar cuando hay que apagar y encender los nodos de cómputo para mantener una adecuada calidad de servicio para el Gestor de Máquinas Virtuales (en la figura se muestra OpenNebula) y, por ende, a la aplicación.

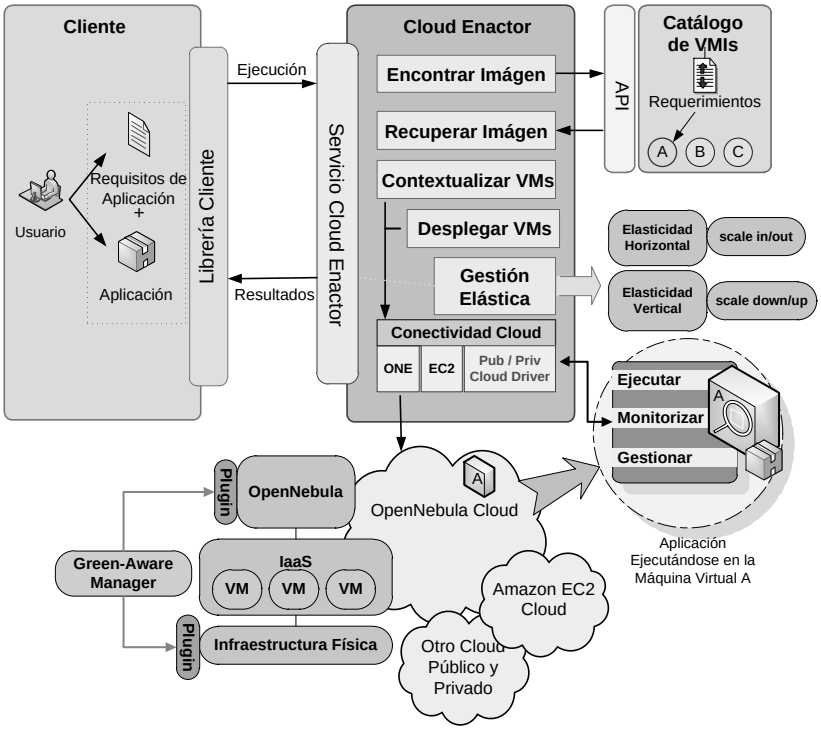

Figura 35. Arquitectura principal del sistema de gestión elástica

7.2.1 Aportaciones

Los resultados obtenidos en este proyecto que han sido fruto de ésta tesis son los siguientes:

- El IM desarrollado se corresponde con la funcionalidad asignada al componente denominado Cloud Enactor ya que se encarga de orquestar todos los servicios subyacentes para conseguir la ejecución de la aplicación sobre la plataforma Cloud. Incluyendo toda la funcionalidad indicada:
    - Acceder al catálogo de imágenes de máquinas virtuales para buscar la imagen más apropiada

- En caso necesario realizar la fase de contextualización para desplegar la aplicación junto a sus requisitos en la(s) maquina(s) virtual(es).
- Proporcionar servicios de gestión elástica para realizar un escalado horizontal o vertical de la infraestructura.

8 Publicaciones

Las publicaciones derivadas de los trabajos de la tesis en la han sido las siguientes:

- Caballer, M.; Blanquer, I.; Moltó, G.; de Alfonso, C.; "Dynamic management of virtual infrastructures". *Journal of Grid Computing*, 2014, ISSN 1570-7873, doi. 10.1007/s10723-014-9296-5.

  (Revista indexada en JCR con factor de impacto 2012: 1,603 – Q1)

- Caballer, M.; de Alfonso, C.; Alvarruiz, F.; Moltó, G.; "EC3: Elastic Cloud Computing Cluster". *Journal of Computer and System Sciences*, Volume 78, Issue 8, December 2013, pp. 1341 – 1351, ISSN 0022-0000, doi. 10.1016/j.jcss.2013.06.005.

  (Revista indexada en JCR con factor de impacto 2012: 1,000 – Q2)

- Caballer, M.; de Alfonso, C.; Moltó, G.; Romero, E.; García, A.; "CodeCloud: A Platform to Enable Execution of Programming Models on the Clouds". *Journal of Systems and Software*, 2014, ISSN 0164-1212, doi 10.1016/j.jss.2014.02.005.

  (Revista indexada en JCR con factor de impacto 2012: 1,135 – Q2).

- Segrelles, J.D.; Caballer, M.; Torres, E.; Moltó, G.; Blanquer, I.; Martí, L.; "Platform to Ease the Deployment and Improve the Availability of TRENCADIS Infrastructure". *Computing and Informatics*. En proceso de revisión.

  (Revista indexada en JCR con factor de impacto 2012: 0,254 – Q4).

- Moltó, G.; Caballer, M.; Romero, E.; de Alfonso, C.; "Elastic Memory Management of Virtualized Infrastructures for Applications with Dynamic Memory Requirements". *International Conference on Computational Science*, ICCS 2013. Publicado en Procedia Computer Science, Volume 18, 2013, pp. 159 – 168, ISSN 1877-0509, doi. 10.1016/j.procs.2013.05.179.

  (Congreso CORE2013 Tipo A)

- Moltó, G.; Caballer, M.; "Scalable Software Practice Environments Featuring Automatic Provision and Configuration in the Cloud". *Proceedings of 19th International Conference on Parallel and Distributed Processing Techniques and Applications* (PDPTA), pp. 658 – 664, 2013.

  (Congreso CORE2013 Tipo B)

- de Alfonso, C.; Caballer, M.; Alvarruiz, F.; Moltó, G.; and Hernández, V.; "Infrastructure Deployment over the Cloud". *Proceedings of 3rd IEEE*

International Conference on Cloud Computing Technology and Science (CloudCom), pp. 517 – 521, 2011.

(Congreso CORE2013 Tipo C)

- Carrión, J.V.; Moltó, G.; de Alfonso, C.; Caballer M.; and V. Hernández; "A Generic Catalog and Repository Service for Virtual Machine Images", *Proceedings of 2nd International ICST Conference on Cloud Computing* (CloudComp), pp. 1 – 15, 2010.
- Caballer, M.; García, A.; Moltó G.; de Alfonso, C.; "Towards SLA-driven Management of Cloud Infrastructures to Elastically Execute Scientific Applications". *Proceedings of 6th IBERIAN GRID INFRASTRUCTURE CONFERENCE* (Ibergrid), 207 – 218, 2012.
- Caballer, M.; de la Fuente, P.; Lozano, P.; Alonso, J.M.; de Alfonso, C.; Hernández, V.; "Easing the Structural Analysis Migration to the Cloud". *Proceedings of 7th IBERIAN GRID INFRASTRUCTURE CONFERENCE* (Ibergrid), 159 – 171, 2013.
- Segrelles, D.; Caballer, M.; Torres, E.; Moltó, G.; Blanquer, I.; "Platform to Ease the Deployment and Improve the Availability of TRENCADIS Infrastructure". *Proceedings of 7th IBERIAN GRID INFRASTRUCTURE CONFERENCE* (Ibergrid), 133 – 145, 2013.

Anexo I

Documento RADL completo para el despliegue de todos los servicios necesarios en el TRENCADIS Center.

```
network publica (outbound = 'yes')
network privada

system iis (
cpu.arch='x86_64' and
cpu.count>=1 and
memory.size>=1024m and
net_interface.0.connection = 'publica' and
net_interface.0.dns_name = 'iis#N#.trencadistest.trencadis.es' and
disk.0.os.name='linux' and
disk.0.os.flavour='CentOS' and
disk.0.os.version>='5' and
disk.0.applications contains (name='globus' and version > '4' and
preinstalled='yes')
)

system eouid (
cpu.arch='x86_64' and
memory.size>=1024m and
net_interface.0.connection = 'publica' and
net_interface.0.dns_name = 'eouidgeneratorservice#N#.trencadistest.trencadis.es' and
net_interface.1.connection = 'privada' and
net_interface.1.dns_name = 'eouid#N#.localdomain' and
disk.0.os.name='linux' and
disk.0.os.flavour='CentOS' and
disk.0.os.version>='5' and
disk.0.applications contains (name='globus' and version > '4' and
preinstalled='yes')
disk.0.applications contains (name='postgresql-server')
)

system voms (
cpu.arch='x86_64' and
memory.size>=1024m and
net_interface.0.connection = 'publica' and
net_interface.0.dns_name = 'vomsserver.trencadistest.trencadis.es' and
disk.0.os.name='linux' and
disk.0.os.flavour='CentOS' and
disk.0.os.version>='5'
)

system dbontology (
cpu.arch='x86_64' and
memory.size>=1024m and
net_interface.0.connection = 'privada' and
net_interface.0.dns_name = 'backend_db_ontologies#N#.localdomain' and
disk.0.os.name='linux' and
disk.0.os.flavour='CentOS' and
disk.0.os.version>='5' and
disk.0.applications contains (name='postgresql-server')
)

system ontology (
cpu.arch='x86_64' and
memory.size>=1024m and
net_interface.0.connection = 'publica' and
```

```
net_interface.0.dns_name = 'ontologiesserverservice#N#.trencadistest.trencadis.es'
and
net_interface.1.connection = 'privada' and
net_interface.1.dns_name = 'ontologies#N#.localdomain' and
disk.0.os.name='linux' and
disk.0.os.flavour='CentOS' and
disk.0.os.version>='5' and
disk.0.application contains (name='globus' and version > '4' and preinstalled='yes')
disk.0.application contains (name='postgresql-server')
)

system storage (
cpu.arch='x86_64' and
memory.size>=1024m and
net_interface.0.connection = 'publica' and
net_interface.0.dns_name = 'storageservice#N#.trencadistest.trencadis.es' and
net_interface.1.connection = 'privada' and
net_interface.1.dns_name = 'storage#N#.localdomain' and
disk.0.os.name='linux' and
disk.0.os.flavour='CentOS' and
disk.0.os.version>='5' and
disk.0.applications contains (name='globus' and version > '4' and
preinstalled='yes')
disk.0.applications contains (name='postgresql-server')
)

configure backend (
@begin
- template: src=utils/templates/epel-es.repo dest=/etc/yum.repos.d/epel.repo

- yum: pkg=python-psycopg2 state=installed

- user: name=postgres password=poEBLWFQo0XHA

- service: name=postgresql state=started enabled=yes

- shell: sudo -u postgres initdb data creates=/var/lib/pgsql/data

- get_url: url=${RSCF}/COMMON/POSTGRESQL/CONFIGURATION/postgresql.conf
dest=/var/lib/pgsql/data/postgresql.conf force=yes
- local_action: get_url url=${RSCF}/COMMON/POSTGRESQL/CONFIGURATION/pg_hba.j2
dest=/tmp/pg_hba.j2
- template: src=/tmp/pg_hba.j2 dest=/var/lib/pgsql/data/pg_hba.conf

- shell: sudo -u postgres psql -c "alter user postgres with password
'postgresadmintrencadis';"

- service: name=postgresql state=restarted

- postgresql_db: db=DB_GateKeeper
  register: create_db_result
  ignore_errors: yes

- local_action: get_url
url=${RSCF}/COMMON/GATEKEEPER/CONFIGURATION/DB_GateKeeper_Schema.j2
dest=/tmp/DB_GateKeeper_Schema.j2
- template: src=/tmp/DB_GateKeeper_Schema.j2
dest=/var/lib/pgsql/data/DB_GateKeeper_Schema.SQL

- shell: su postgres -c "psql DB_GateKeeper <
/var/lib/pgsql/data/DB_GateKeeper_Schema.SQL"
  when: create_db_result is defined and create_db_result['changed']

- service: name=postgresql state=restarted
```

```
@end
)

configure grid_service (
@begin
- include: backend.yml

- file: path=/etc/grid-security/certificates state=directory

- get_url: url=${RSCF}/${VO}/CAs/${item} dest=/etc/grid-
security/certificates/${item}
  with_items:
    - ${CA_HASH}.0
    - ${CA_HASH}.r0
    - ${CA_HASH}.signing_policy

- get_url:
url=${RSCF}/${VO}/${SERVICE_TYPE}Service/CERTIFICATES/${IM_NODE_FQDN}/hostkey.pem
dest=/etc/grid-security/hostkey.pem mode=0400 owner=trencadis group=trencadis
- get_url:
url=${RSCF}/${VO}/${SERVICE_TYPE}Service/CERTIFICATES/${IM_NODE_FQDN}/hostcert.pem
dest=/etc/grid-security/hostcert.pem mode=0600 owner=trencadis group=trencadis

- file: src=/etc/grid-security/hostkey.pem dest=/etc/grid-security/containerkey.pem
state=link
- file: src=/etc/grid-security/hostcert.pem dest=/etc/grid-
security/containercert.pem state=link

- file: path=/home/trencadis/${SERVICE_TYPE}Service/tmp state=directory
owner=trencadis group=trencadis

- get_url: url=${RSCF}/COMMON/${SERVICE_TYPE}Service/CONFIGURATION/Registration.xml
dest=/home/trencadis/${SERVICE_TYPE}Service/Registration.xml owner=trencadis
group=trencadis

- local_action: get_url
url=${RSCF}/COMMON/${SERVICE_TYPE}Service/CONFIGURATION/${SERVICE_TYPE}Config.j2
dest=/tmp/${SERVICE_TYPE}Config.j2 owner=trencadis group=trencadis
- template: src=/tmp/${SERVICE_TYPE}Config.j2
dest=/home/trencadis/${SERVICE_TYPE}Service/${SERVICE_TYPE}Config.xml

- get_url:
url=${RSCF}/COMMON/${SERVICE_TYPE}Service/trencadis_infrastructure_services_${SERVIC
E_TYPE}.gar
dest=/home/trencadis/trencadis_infrastructure_services_${SERVICE_TYPE}.gar
owner=trencadis group=trencadis

- command: $GLOBUS_LOCATION/bin/globus-deploy-gar
/home/trencadis/trencadis_infrastructure_services_${SERVICE_TYPE}.gar
creates=$GLOBUS_LOCATION/lib/trencadis_infrastructure_services_${SERVICE_TYPE}.jar
  environment:
    GLOBUS_LOCATION: $GLOBUS_LOCATION
    ANT_HOME: $ANT_HOME
    JAVA_HOME: $JAVA_HOME

- get_url: url=${RSCF}/COMMON/TRENCADIS_COMMON_TOOLS/TRENCADIS_CommonTools.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_CommonTools.jar owner=trencadis
group=trencadis
- get_url: url=${RSCF}/COMMON/TRENCADIS_COMMON_TOOLS/LIB_DEPENDENCES/axis.jar
dest=${GLOBUS_LOCATION}/lib/axis.jar owner=trencadis group=trencadis

- get_url: url=${RSCF}/COMMON/GATEKEEPER/GateKeeper.jar
dest=${GLOBUS_LOCATION}/lib/GateKeeper.jar owner=trencadis group=trencadis
```

```
- get_url: url=${RSCF}/COMMON/GATEKEEPER/LIB_DEPENDENCES/${item}
dest=${GLOBUS_LOCATION}/lib/${item} owner=trencadis group=trencadis
  with_items:
  - cog-jglobus-1.2-060802.jar
  - commons-logging-1.1.jar
  - pg73jdbc2.jar

- get_url:
url=${RSCF}/COMMON/SQL_DATABASE_CONNECTIONPOOL/SQLDatabaseConnectionPool.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_CommonTools.jar owner=trencadis
group=trencadis

- get_url:
url=${RSCF}/COMMON/TRENCADIS_JAVA_API_VOMS_CLIENT/TRENCADIS_JAVA_API_VOMS_Client.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_JAVA_API_VOMS_Client.jar owner=trencadis
group=trencadis
- get_url: url=${RSCF}/COMMON/TRENCADIS_JAVA_API_VOMS_CLIENT/LIB_DEPENDENCES/${item}
dest=${GLOBUS_LOCATION}/lib/${item} owner=trencadis group=trencadis
  with_items:
  - bcprov-jdk14-145.jar
  - commons-cli-2.0.jar
  - commons-lang-2.1.jar
  - log4j-1.2.15.jar

- get_url: url=${RSCF}/COMMON/TRENCADIS_JAVA_API_IIS/TRENCADIS_JAVA_API_IIS.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_JAVA_API_IIS.jar owner=trencadis
group=trencadis

- local_action: get_url
url=${RSCF}/COMMON/${SERVICE_TYPE}Service/CONFIGURATION/hierarchy.j2
dest=/tmp/hierarchy.j2 owner=trencadis group=trencadis
- template: src=/tmp/hierarchy.j2
dest=${GLOBUS_LOCATION}/etc/globus_wsrf_mds_index/hierarchy.xml

- service: name=container state=started enabled=yes

@end
)

configure eouid (
@begin
---
  - vars:
    - CA_HASH: fea48927
    - VO: trencadistest
    - RSCF: http://webgrycap.i3m.upv.es/RSCF
    - UNIQUE_GateKeeper_BackEnd: 0
    - INSTANCIAS_IIS: 1
    - GLOBUS_LOCATION: $IM_APP_GLOBUS_PATH
    - ANT_HOME: $IM_APP_ANT_PATH
    - JAVA_HOME: $IM_APP_JAVA_PATH
    - UNIQUE_BackEnd: 0
    - CONSUMER_SERVICE: eouid
    - SERVICE_TYPE: EOUIDGenerator

    tasks:
    - include: grid_service.yml
@end
)

configure voms (
@begin
---
  - vars:
```

```
    - CA_HASH: fea48927
    - VO: trencadistest
    - RSCF: http://webgrycap.i3m.upv.es/RSCF

    tasks:
    - yum: pkg=mysql-server state=installed

    - yum: pkg=MySQL-python state=installed

    - service: name=mysqld state=started enabled=yes

    - command: /usr/bin/mysqladmin -u root password  mysqladmintrencadis
      ignore_errors: yes

    - mysql_db: db=db_${VO} state=present login_password=mysqladmintrencadis
login_user=root

    - mysql_user: name=trencadis password=trencadis_$VO priv=*.*:ALL state=present
login_password=mysqladmintrencadis login_user=root

    - file: path=${item} state=directory
      with_items:
        - /etc/grid-security
        - /etc/grid-security/certificates
        - /root/VOMSSERVER
        - /root/VOMSSERVER/services

    - get_url: url=${RSCF}/${VO}/CAs/${item} dest=/etc/grid-
security/certificates/${item}
      with_items:
        - ${CA_HASH}.0
        - ${CA_HASH}.r0
        - ${CA_HASH}.signing_policy

    - get_url: url=${RSCF}/${VO}/IIS/CERTIFICATES/hostkey.pem dest=/etc/grid-
security/hostkey.pem mode=0400

    - get_url: url=${RSCF}/${VO}/IIS/CERTIFICATES/hostcert.pem dest=/etc/grid-
security/hostcert.pem mode=0600

    - local_action: get_url url=${RSCF}/COMMON/VOMSSERVER/CONFIGURATION/site-info.j2
dest=/tmp/voms-site-info.j2

    - local_action: get_url
url=${RSCF}/COMMON/VOMSSERVER/CONFIGURATION/services/glite-voms.j2 dest=/tmp/glite-
voms.j2

    - template: src=/tmp/voms-site-info.j2 dest=/root/VOMSSERVER/site-info.def

    - template: src=/tmp/glite-voms.j2 dest=/root/VOMSSERVER/services/glite-voms

    - template: src=utils/templates/epel.repo dest=/etc/yum.repos.d/epel.repo
    - get_url: url=http://emisoft.web.cern.ch/emisoft/dist/EMI/1/sl5/repos/${item}
dest=/etc/yum.repos.d/${item}
      with_items:
        - emi1-base.repo
        - emi1-third-party.repo
        - emi1-updates.repo

    - yum: pkg=emi-voms-mysql state=installed disable_gpg_check=yes

    - command: /opt/glite/yaim/bin/yaim -c -s site-info.def -n VOMS
chdir=/root/VOMSSERVER
      environment:
```

```
        PATH: "{{ lookup('env','PATH') }}:/sbin"

@end
)

configure dbontology (
@begin
---
  - vars:
    - VO: trencadistest
    - RSCF: http://webgrycap.i3m.upv.es/RSCF
    - UNIQUE_BackEnd: 0
    - CONSUMER_SERVICE: ontology

    tasks:
    - user: name=postgres password=poEBLWFQo0XHA

    - service: name=postgresql state=started enabled=yes
      ignore_errors: yes

    - get_url: url=${RSCF}/COMMON/POSTGRESQL/CONFIGURATION/postgresql.conf
dest=/var/lib/pgsql/data/postgresql.conf force=yes

    - service: name=postgresql state=started enabled=yes

    - local_action: get_url url=${RSCF}/COMMON/POSTGRESQL/CONFIGURATION/pg_hba.j2
dest=/tmp/pg_hba.j2
    - template: src=/tmp/pg_hba.j2 dest=/var/lib/pgsql/data/pg_hba.conf

    - shell: sudo -u postgres psql -c "alter user postgres with password
'postgresadmintrencadis';"

    - service: name=postgresql state=restarted

    - shell: sudo -u postgres createdb DB_Ontologies
      ignore_errors: yes
      register: create_db_result

    - get_url:
url=${RSCF}/COMMON/OntologiesServerService/CONFIGURATION/DB_Ontologies_Schema.SQL
dest=/var/lib/pgsql/data/DB_Ontologies_Schema.SQL

    - shell: su postgres -c "psql DB_Ontologies <
/var/lib/pgsql/data/DB_Ontologies_Schema.SQL"
      when: create_db_result.rc == 0

    - service: name=postgresql state=restarted

@end
)

configure ontology (
@begin
---
  - vars:
    - CA_HASH: fea48927
    - VO: trencadistest
    - RSCF: http://webgrycap.i3m.upv.es/RSCF
    - UNIQUE_GateKeeper_BackEnd: 0
    - Unique_DBOntologies_BackEnd: 0
    - INSTANCIAS_IIS: 1
    - GLOBUS_LOCATION: $IM_APP_GLOBUS_PATH
    - ANT_HOME: $IM_APP_ANT_PATH
    - JAVA_HOME: $IM_APP_JAVA_PATH
```

```
    - UNIQUE_BackEnd: 0
    - CONSUMER_SERVICE: ontology

    tasks:
    - include: backend.yml

    - file: path=/etc/grid-security/certificates state=directory

    - get_url: url=${RSCF}/${VO}/CAs/${item} dest=/etc/grid-
security/certificates/${item}
      with_items:
        - ${CA_HASH}.0
        - ${CA_HASH}.r0
        - ${CA_HASH}.signing_policy

    - get_url:
url=${RSCF}/${VO}/OntologiesServerService/CERTIFICATES/${IM_NODE_FQDN}/hostkey.pem
dest=/etc/grid-security/hostkey.pem mode=0400 owner=trencadis group=trencadis

    - get_url:
url=${RSCF}/${VO}/OntologiesServerService/CERTIFICATES/${IM_NODE_FQDN}/hostcert.pem
dest=/etc/grid-security/hostcert.pem mode=0600 owner=trencadis group=trencadis

    - file: src=/etc/grid-security/hostkey.pem dest=/etc/grid-
security/containerkey.pem state=link

    - file: src=/etc/grid-security/hostcert.pem dest=/etc/grid-
security/containercert.pem state=link

    - file: path=/home/trencadis/OntologiesServerService/tmp state=directory
owner=trencadis group=trencadis

    - local_action: get_url
url=${RSCF}/COMMON/OntologiesServerService/CONFIGURATION/OntologiesServerServiceConf
ig.j2 dest=/tmp/OntologiesServerServiceConfig.j2 owner=trencadis group=trencadis
    - template: src=/tmp/OntologiesServerServiceConfig.j2
dest=/home/trencadis/OntologiesServerService/OntologiesServerServiceConfig.xml

    - get_url:
url=${RSCF}/COMMON/OntologiesServerService/trencadis_infrastructure_services_Ontolog
iesServer.gar
dest=/home/trencadis/trencadis_infrastructure_services_OntologiesServer.gar
owner=trencadis group=trencadis

    - command: $GLOBUS_LOCATION/bin/globus-deploy-gar
/home/trencadis/trencadis_infrastructure_services_OntologiesServer.gar
creates=$GLOBUS_LOCATION/lib/trencadis_infrastructure_services_OntologiesServer.jar
      environment:
        GLOBUS_LOCATION: $GLOBUS_LOCATION
        ANT_HOME: $ANT_HOME
        JAVA_HOME: $JAVA_HOME

    - get_url: url=${RSCF}/COMMON/TRENCADIS_COMMON_TOOLS/TRENCADIS_CommonTools.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_CommonTools.jar owner=trencadis
group=trencadis
    - get_url: url=${RSCF}/COMMON/TRENCADIS_COMMON_TOOLS/LIB_DEPENDENCES/axis.jar
dest=${GLOBUS_LOCATION}/lib/axis.jar owner=trencadis group=trencadis

    - get_url: url=${RSCF}/COMMON/GATEKEEPER/GateKeeper.jar
dest=${GLOBUS_LOCATION}/lib/GateKeeper.jar owner=trencadis group=trencadis
    - get_url: url=${RSCF}/COMMON/GATEKEEPER/LIB_DEPENDENCES/${item}
dest=${GLOBUS_LOCATION}/lib/${item} owner=trencadis group=trencadis
      with_items:
      - cog-jglobus-1.2-060802.jar
```

```
      - commons-logging-1.1.jar
      - pg73jdbc2.jar

    - get_url:
url=${RSCF}/COMMON/SQL_DATABASE_CONNECTIONPOOL/SQLDatabaseConnectionPool.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_CommonTools.jar owner=trencadis
group=trencadis

    - get_url:
url=${RSCF}/COMMON/TRENCADIS_JAVA_API_VOMS_CLIENT/TRENCADIS_JAVA_API_VOMS_Client.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_JAVA_API_VOMS_Client.jar owner=trencadis
group=trencadis
    - get_url:
url=${RSCF}/COMMON/TRENCADIS_JAVA_API_VOMS_CLIENT/LIB_DEPENDENCES/${item}
dest=${GLOBUS_LOCATION}/lib/${item} owner=trencadis group=trencadis
      with_items:
      - bcprov-jdk14-145.jar
      - commons-cli-2.0.jar
      - commons-lang-2.1.jar
      - log4j-1.2.15.jar

    - get_url: url=${RSCF}/COMMON/TRENCADIS_JAVA_API_IIS/TRENCADIS_JAVA_API_IIS.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_JAVA_API_IIS.jar owner=trencadis
group=trencadis

    - get_url:
url=${RSCF}/COMMON/TRENCADIS_JAVA_API_SQL_ONTOLOGIES_DATABASE/TRENCADIS_JAVA_API_SQL
_Ontologies_Database.jar
dest=${GLOBUS_LOCATION}/lib/TRENCADIS_JAVA_API_SQL_Ontologies_Database.jar
owner=trencadis group=trencadis

    - local_action: get_url
url=${RSCF}/COMMON/OntologiesServerService/CONFIGURATION/hierarchy.j2
dest=/tmp/hierarchy.j2 owner=trencadis group=trencadis
    - template: src=/tmp/hierarchy.j2
dest=${GLOBUS_LOCATION}/etc/globus_wsrf_mds_index/hierarchy.xml

    - service: name=container state=started enabled=yes

@end
)

configure iis (
@begin
---
  - vars:
    - CA_HASH: fea48927
    - VO: trencadistest
    - RSCF: http://webgrycap.i3m.upv.es/RSCF
    - GLOBUS_LOCATION: $IM_APP_GLOBUS_PATH

    tasks:
    - file: path=${item} state=directory
      with_items:
        - /etc/grid-security
        - /etc/grid-security/certificates

    - get_url: url=${RSCF}/${VO}/CAs/${item} dest=/etc/grid-
security/certificates/${item}
      with_items:
        - ${CA_HASH}.0
        - ${CA_HASH}.r0
        - ${CA_HASH}.signing_policy
```

```
    - get_url: url=${RSCF}/${VO}/IIS/CERTIFICATES/${IM_NODE_FQDN}/hostkey.pem
dest=/etc/grid-security/hostkey.pem mode=0400 owner=trencadis group=trencadis
    - get_url: url=${RSCF}/${VO}/IIS/CERTIFICATES/${IM_NODE_FQDN}/hostcert.pem
dest=/etc/grid-security/hostcert.pem mode=0600 owner=trencadis group=trencadis

    - file: src=/etc/grid-security/hostkey.pem dest=/etc/grid-
security/containerkey.pem state=link
    - file: src=/etc/grid-security/hostcert.pem dest=/etc/grid-
security/containercert.pem state=link

    - yum: pkg=tomcat5 state=installed

    - shell: $GLOBUS_LOCATION/lib/webmds/bin/webmds-create-context-file
/usr/share/tomcat5/conf/Catalina/localhost
creates=/usr/share/tomcat5/conf/Catalina/localhost/webmds.xml
      environment:
        GLOBUS_LOCATION: $GLOBUS_LOCATION

    - file: src=$GLOBUS_LOCATION/endorsed/xalan-2.6.jar
dest=/var/lib/tomcat5/webapps/xalan-2.6.jar state=link

    - lineinfile: dest=/usr/share/tomcat5/conf/server.xml state=present
regexp='port="8888"' line='<Connector port="8888" maxHttpHeaderSize="8192"'
insertafter='port="8080"'

    - lineinfile: dest=/usr/share/tomcat5/conf/server.xml state=absent
regexp='port="8080"'

    - service: name=container state=restarted enabled=yes

    - service: name=tomcat5 state=restarted enabled=yes

@end
)

configure storage (
@begin
---
  - vars:
    - CA_HASH: fea48927
    - VO: trencadistest
    - RSCF: http://webgrycap.i3m.upv.es/RSCF
    - UNIQUE_GateKeeper_BackEnd: 0
    - INSTANCIAS_IIS: 1
    - GLOBUS_LOCATION: $IM_APP_GLOBUS_PATH
    - ANT_HOME: $IM_APP_ANT_PATH
    - JAVA_HOME: $IM_APP_JAVA_PATH
    - UNIQUE_BackEnd: 0
    - CONSUMER_SERVICE: storage
    - SERVICE_TYPE: Storage

    tasks:
    - include: grid_service.yml
@end
)

deploy eouid 1
deploy voms 1
deploy ontology 1
deploy dbontology 1
deploy iis 1
deploy storage 1
```

Anexo II

A continuación se muestra con detalle la funcionalidad y parámetros de las funciones del API del IM.

API XML-RPC

A continuación se muestran las interfaces de las 10 funciones que proporciona el API nativa del gestor de la infraestructura:

GetInfrastructureList: Toma los datos de autenticación y consulta el listado de todas las infraestructuras que hayan sido creadas por el usuario indicado.

- Parámetros de salida:
    - Una lista de los identificadores de las infraestructuras creadas por el usuario.

CreateInfrastructure: Usa la información del documento RADL para crear y configurar de forma adecuada todas las MVs especificadas.

- Parámetros de entrada:
    - Un documento RADL con la definición de la infraestructura.
- Parámetros de salida:
    - Un identificador de la infraestructura creada, que más adelante podremos usar para la gestión de la misma.

GetInfrastructureInfo: Proporciona la información sobre todas las MVs que pertenezcan a la infraestructura en el momento de la llamada.

- Parámetros de entrada:
    - El identificador de la infraestructura.
- Parámetros de salida:
    - Mensaje de texto de salida del proceso de contextualización, o cadena vacía en caso de que no se haya producido todavía.
    - Listado de todos las IDs de las MVs creadas. Un identificador para cada una de ellas con el que podremos en el futuro consultar el estado y/o destruirla.

GetVMInfo: Proporciona la información sobre la MV indicada como parámetro.

- Parámetros de entrada:
    - El identificador de la infraestructura.

- El identificador de la MV obtenido en la función anterior.

- Parámetros de salida:
  - Todos los datos de la MV creada en formato RADL. Entre ellos todos los necesarios para acceder a cada una de ellas: IP de la máquina, protocolo para acceder, usuario y contraseña o certificado, etc. También indicará un valor tabulado del estado de la máquina. La Figura 36 muestra un diagrama con el ciclo de estados de las MVs en el IM.
    - pending: La MV ha sido lanzada pero aún se encuentra en estado de inicialización.
    - running: La MV ha sido creada con éxito y se encuentra en estado running. En este estado el S.O, "toma el control" de la MV.
    - configured: La MV se encuentra en estado running y además el proceso de contextualización ha finalizado con éxito. En el caso de que no se haya indicado ningún proceso de contextualización este estado será equivalente a "running"
    - stopped: La MV se encuentra parada o suspendida.
    - off: La MV se encuentra apagada o no existe.
    - failed: Se ha producido un error en la MV, bien en el lanzamiento o en la contextualización.
    - unknown: El IM no es capaz de determinar el estado de la MV.

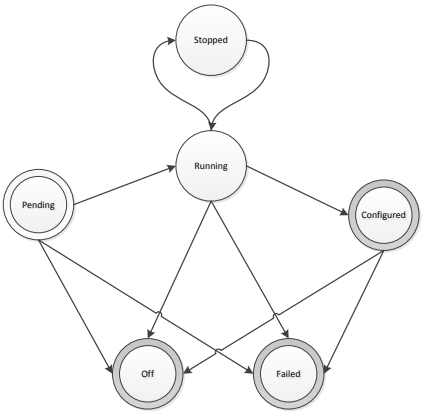

Figura 36. Diagrama de estados de las MVs.

AlterVM: Cambia las propiedades de la MV, para proporcionar funcionalidades para la gestión de la elasticidad vertical.

- Parámetros de entrada:
    - El identificador de la Infraestructura.
    - El identificador de la MV.
    - Un documento RADL con los parámetros a modificar de la MV.
- Parámetros de salida:
    - Todos los datos de la MV ya modificada, con el mismo formato de la función GetVMInfo.

DestroyInfrastructure: Libera todos los recursos relacionados con la infraestructura especificada, eliminando todas las MVs que la componen.

- Parámetros de entrada:
    - El identificador de la infraestructura.

AddResource: Añade los recursos indicados en el documento RADL al despliegue actual y reconfigurará el sistema completo. Permite la gestión de la elasticidad horizontal. El nuevo documento RADL especificado se "fusionará" con el originalmente enviado en el momento de la creación de la infraestructura. Por tanto el nuevo documento podrá contener solo instrucciones de tipo "deploy" haciendo referencia a algunos de los "system" ya definidos. En caso de que se definan nuevos "system" estos serán añadidos. En el caso de que se utilice algún "system" o "network" previamente definido, la nueva especificación será ignorada, conservando los objetos previos. En caso de querer modificar algún "system" se deberá utilizar la función AlterVM.

- Parámetros de entrada:
    - El identificador de la infraestructura.
    - Documento RADL con los nuevos recursos a añadir al despliegue.

RemoveResource: Eliminará las MV especificadas del despliegue actual y reconfigurará el sistema completo. Permite la gestión de la elasticidad horizontal.

- Parámetros de entrada:
    - El identificador de la infraestructura.
    - Identificadores de las maquinas a eliminar de la infraestructura separados por comas.

StopInfrastructure: Parará (pero no destruirá) todas las MV de la infraestructura. De tal manera que se mantendrá toda la información del disco de las MVs, pero no se producirá gasto (en términos de horas de CPU) por dichas MVs.

- Parámetros de entrada:
    - El identificador de la Infraestructura.

StartInfrastructure: Arrancará de nuevo todas las MV de la infraestructura, previamente paradas con la función StoptInfrastructure. Estas dos funciones permiten que infraestructuras que durante su uso han almacenado información en disco, que puede ser necesaria en futuros usos de la misma, puedan ser paradas, y por tanto dejar de producir gasto por uso de CPU, durante un periodo de tiempo para más adelante volver arrancar sin pérdida de dicha información.

- Parámetros de entrada:
    - El identificador de la infraestructura.

Reconfigure: Permite cambiar las definiciones de las configuraciones de las MVs en funcionamiento, para pasar una fase de reconfiguración para darles una nueva funcionalidad a las MVs en ejecución. Esto permite tener una infraestructura en ejecución y una vez acabada la tarea para la que estaban configuradas, reconfigurarlas para ejecutar una tarea diferente, sin tener la necesidad de destruir la infraestructura y volverla a lanzar. Esto solo se deberá hacer en los casos en los que las los requisitos sobre la infraestructura (arquitectura de la CPU, Sistema Operativo, etc.) de las configuraciones nueva y antigua sean "compatibles". También se puede llamar sin cambiar los apartados de configuración, para forzar de nuevo el proceso de reconfiguración. Esto puede servir para los casos en que el proceso de contextualización dependa de algún fichero que haya sido modificado y dicha modificación necesite de una reconfiguración del sistema para propagar dicho cambio sobre todas las MVs de la infraestructura.

- Parámetros de entrada:
    - El identificador de la infraestructura.
    - RADL, con los nuevos apartados de configuración y contextualización.

API REST

En cuanto al API REST la funcionalidad expuesta es la misma que la anterior y usando los mismos parámetros, pero mediante el clásico esquema de URLs y de "verbos" HTTP para representar las operaciones a realizar:

- GET /infraestructure: equivalente a GetInfrastructureList
- PUT /infraestructure: equivalente a CreateInfrastructure.

  - Recibe a través de los parámetros del post un documento RADL con de la infraestructura a crear.

- GET /inf/<id>: equivalente a GetInfrastructureInfo
  - El parámetro id de la URL es el id de la infraestructura a mostrar
- PUT /inf/<id>: equivalente a AddResource
  - Recibe a través de los parámetros del post un documento RADL con de los recursos a añadir.
- POST /inf/<id>: equivalente a StartInfrastructure, StopInfrastructure y Reconfigure.
  - Recibe a través de los parámetros del post un parámetro de texto denominado "op" donde se indica el valor "start" para la primera función, "stop" para la segunda y "reconfigure" para la tercera. En este último caso recibirá un parámetro extra "radl" con los nuevos apartados de configuración y contextualización.
- DELETE /inf/<id>: equivalente a DestroyInfrastructure
  - El parámetro id de la URL es el identificador de la infraestructura a eliminar.
- GET /vms/<infid>/<vmid>: equivalente a GetVMInfo
  - Los parámetros infid y vmid de la URL son los de la infraestructura y de la máquina virtual a mostrar.
- POST /vms/<infid>/<vmid>: equivalente a AlterVM
  - Los parámetros infid y vmid de la URL son los de la infraestructura y de la máquina virtual a modificar.
  - Recibe a través de los parámetros del post un documento RADL con los parámetros a modificar de la MV.
- DELETE /vms/<infid>/<vmid>: equivalente a RemoveResource
  - Los parámetros infid y vmid de la URL son los de la infraestructura y de la máquina virtual a eliminar.

Referencias

[1] I. Foster and C. Kesselman, Eds., *The Grid: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

[2] I. Foster and C. Kesselman, *The grid 2: Blueprint for a new computing infrastructure*. Morgan Kaufmann Publishers Inc., 2004.

[3] P. Mell and T. Grance, "The NIST Definition of Cloud Computing," *Natl. Inst. Stand. Technol.*, vol. 53, p. 50, 2009.

[4] M. Armbrust, A. Fox, R. Griffith, A. Joseph, and RH, "Above the clouds: A Berkeley view of cloud computing," *Univ. California, Berkeley, Tech. Rep. UCB* , pp. 07–013, 2009.

[5] N. Khan, A. Noraziah, E. I. Ismail, M. M. Deris, and T. Herawan, "Cloud Computing: Analysis of Various Platforms," *Int. J. E-entrepreneursh. Innov.*, vol. 3, no. 2, pp. 51–59, Jan. 2012.

[6] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Futur. Gener. Comput. Syst.*, vol. 25, no. 6, pp. 599–616, 2009.

[7] M. Armbrust, I. Stoica, M. Zaharia, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, and A. Rabkin, "A view of cloud computing," *Communications of the ACM*, vol. 53. p. 50, 2010.

[8] F. P. Miller, A. F. Vandome, and J. McBrewster, "Amazon Web Services," 2010.

[9] Google, "Google Cloud Platform." [Online]. Available: https://cloud.google.com/.

[10] Microsoft, "Windows Azure." [Online]. Available: http://www.microsoft.com/windowsazure/.

[11] G. von Laszewski, J. Diaz, F. Wang, and G. C. Fox, "Comparison of Multiple Cloud Frameworks," in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 734–741.

[12] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual Infrastructure Management in Private and Hybrid Clouds," *IEEE Internet Comput.*, vol. 13, 2009.

[13] OpenStack, “OpenStack Open Source Cloud Computing Software.” [Online]. Available: http://www.openstack.org.

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov, “The Eucalyptus Open-Source Cloud-Computing System,” *2009 9th IEEE/ACM Int. Symp. Clust. Comput. Grid*, 2009.

[15] University of Chicago, “Nimbus Project.” [Online]. Available: http://www.nimbusproject.org/.

[16] OCCI working group within the Open Grid Forum, “Open Cloud Computing Interface – Infrastructure.” 2011.

[17] Distributed Management Task Force, Inc, “Cloud Infrastructure Management Interface (CIMI) Model and REST Interface over HTTP Specification.” 2012.

[18] K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, and W. Vambenepe, “The WS-Resource Framework,” *Framework*. pp. 1–18, 2004.

[19] J. Bresnahan, T. Freeman, D. LaBissoniere, and K. Keahey, “Managing appliance launches in infrastructure clouds,” in *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*, 2011, pp. 12:1–12:7.

[20] Telefónica Investigación y Desarrollo S.A. Unipersonal., “Telefónica’s TCloud API Specification.” 2010.

[21] Morfeo Project, “Claudia,” 2013. [Online]. Available: http://claudia.morfeo-project.org/wiki.

[22] Dmtf, “Open Virtualization Format Specification,” *DMTF Virtualization Manag. VMAN Initiat.*, pp. 1–42, 2010.

[23] J. Mirkovic, T. Faber, P. Hsieh, G. Malaiyandisamy, and R. Malaviya, “DADL: Distributed Application Description Language,” *USC/ISI Tech. Report# ISI-TR-664*, 2010.

[24] P. Goldsack, J. Guijarro, S. Loughran, A. Coles, A. Farrell, A. Lain, P. Murray, and P. Toft, “The SmartFrog configuration management framework,” *ACM SIGOPS Oper. Syst. Rev.*, vol. 43, no. 1, p. 16, Jan. 2009.

[25] M. Burgess, “Cfengine: A site configuration engine,” *USENIX Assoc. Comput. Syst.*, vol. 8, no. 3, pp. 309–337, 1995.

[26] Puppet, “Puppet Labs: IT Automation Software for System Administrators.” [Online]. Available: http://www.puppetlabs.com.

[27] Opscode, “Chef.” [Online]. Available: http://www.opscode.com/chef/.

[28] Capistrano, “Capistrano: A remote server automation and deployment tool written in Ruby.” [Online]. Available: http://www.capistranorb.com.

[29] M. DeHaan, “Ansible.” [Online]. Available: http://ansible.cc/.

[30] G. Moltó and V. Hernández, “Management and Contextualization of Scientific Virtual Appliances,” in *Cloud Futures 2010: Advancing Research with Cloud Computing*, 2010.

[31] VMWare, “Virtual Appliance Marketplace.” [Online]. Available: https://solutionexchange.vmware.com/store.

[32] Amazon Web Services, “Amazon Machine Image (AMI).” [Online]. Available: https://aws.amazon.com/amis.

[33] CloudMarket, “The Cloud Market.” [Online]. Available: http://thecloudmarket.com/.

[34] StratusLab, “StratusLab Marketplace.” [Online]. Available: http://marketplace.stratuslab.eu/metadata.

[35] Amazon Web Services, “AWS CloudFormation.” [Online]. Available: http://aws.amazon.com/es/cloudformation/.

[36] Amazon Web Services, “AWS OpsWorks.” [Online]. Available: http://aws.amazon.com/opsworks/.

[37] K. Keahey and T. Freeman, “Contextualization: Providing One-Click Virtual Clusters,” *2008 IEEE Fourth Int. Conf. eScience*, 2008.

[38] P. Marshall, H. M. Tufo, K. Keahey, D. La Bissoniere, and M. Woitaszek, “Architecting a Large-scale Elastic Environment - Recontextualization and Adaptive Cloud Services for Scientific Computing.,” in *ICSOFT*, 2012, pp. 409–418.

[39] P. Marshall, K. Keahey, and T. Freeman, “Elastic Site: Using Clouds to Elastically Extend Site Resources,” *Clust. Cloud Grid Comput. (CCGrid), 2010 10th IEEE/ACM Int. Conf.*, 2010.

[40] University of Chicago, “Nimbus Phantom.” [Online]. Available: http://www.nimbusproject.org/phantom.

[41] Apache Software Fundation, “Apache Whirr.” [Online]. Available: http://whirr.apache.org/.

[42] T. White, *Hadoop : The Definitive Guide*, vol. 54. 2010, p. 258.

[43] G. Juve and E. Deelman, “Automating Application Deployment in Infrastructure Clouds,” in *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, 2011, pp. 658–665.

[44] HashiCorp, “Vagrant.” [Online]. Available: http://www.vagrantup.com/.

[45] R. Raman, M. Livny, and M. Solomon, “Matchmaking: distributed resource management for high throughput computing,” in *Proceedings. The Seventh International Symposium on High Performance Distributed Computing (Cat. No.98TB100244)*, 1998, pp. 140–146.

[46] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, “Introducing STRATOS: A Cloud Broker Service,” in *2012 IEEE Fifth International Conference on Cloud Computing*, 2012, pp. 891–898.

[47] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata, “CompatibleOne: The Open Source Cloud Broker,” *J. Grid Comput.*, Nov. 2013.

[48] C. Redl, I. Breskovic, I. Brandic, and S. Dustdar, “Automatic SLA Matching and Provider Selection in Grid and Cloud Computing Markets,” in *2012 ACM/IEEE 13th International Conference on Grid Computing*, 2012, pp. 85–94.

[49] S. Sundareswaran, A. Squicciarini, and D. Lin, “A Brokerage-Based Approach for Cloud Service Selection,” in *Proceedings of the 2012 IEEE 5th International Conference on Cloud Computing*, 2012, pp. 558–565.

[50] J. Tordsson, R. S. Montero, R. Moreno-Vozmediano, and I. M. Llorente, “Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers,” *Future Generation Computer Systems*, vol. 28. pp. 358–367, 2012.

[51] M. Zhang, R. Ranjan, M. Menzel, and A. Haller, “A Declarative Recommender System for Cloud Infrastructure Services Selection 2 A System for Cloud Service Selection,” in *9th International Conference on Economics of Grids, Clouds, Systems, and Services, GECON 2012*, 2012, pp. 102–113.

[52] M. Bolte, M. Sievers, G. Birkenheuer, O. Niehorster, and A. Brinkmann, “Non-intrusive virtualization management using libvirt,” *Des. Autom. Test Eur. Conf. Exhib. (DATE), 2010*, 2010.

[53] F. Önnberg, “Software Configuration Management: A comparison of Chef, CFEngine and Puppet,” University of Skövde., 2012.

[54] S. Pandey, “Investigating Community, Reliability and Usability of CFEngine, Chef and Puppet.” Norwegian Open Research Archives (NORA), 2012.

[55] C. A. Waldspurger, “Memory resource management in VMware ESX server,” *ACM SIGOPS Operating Systems Review*, vol. 36. p. 181, 2002.

[56] G. Moltó, M. Caballer, E. Romero, and C. de Alfonso, “Elastic Memory Management of Virtualized Infrastructures for Applications with Dynamic

Memory Requirements," in *Proceedings of the International Conference on Computational Science (ICCS 2013)*, 2013, pp. 159–168.

[57] M. Caballer, A. García, G. Moltó, and C. de Alfonso, "Towards SLA-driven Management of Cloud Infrastructures to Elastically Execute Scientific Applications," in *6th Iberian Grid Infrastructure Conference (IberGrid)*, 2012, pp. 207–218.

[58] M. L. Massie, B. N. Chun, and D. E. Culler, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30. pp. 817–840, 2004.

[59] I. Foster, "Globus Toolkit Version 4: Software for Service-Oriented Systems," *Journal of Computer Science and Technology*, vol. 21. pp. 513–520, 2006.

[60] J. Dean and S. Ghemawat, "MapReduce : Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 1–13, 2008.

[61] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool.," *J. Mol. Biol.*, vol. 215, no. 3, pp. 403–410, 1990.

[62] R. Helaers and M. C. Milinkovitch, "MetaPIGA v2.0: maximum likelihood large phylogeny estimation using the metapopulation genetic algorithm and other stochastic heuristics.," *BMC Bioinformatics*, vol. 11, p. 379, 2010.

[63] N. Abdennadher and R. Boesch, "Towards a peer-to-peer platform for high performance computing," in *Eighth International Conference on High-Performance Computing in Asia-Pacific Region (HPCASIA'05)*, 2005, p. 8 pp.–361.

[64] M. Caballer, C. De Alfonso, G. Moltó, E. Romero, I. Blanquer, and A. García, "CodeCloud: A Platform to Enable Execution of Programming Models on the Clouds," *J. Syst. Softw.*, 2014.

[65] M. Caballer, C. De Alfonso, F. Alvarruiz, and G. Moltó, "EC3: Elastic Cloud Computing Cluster," *J. Comput. Syst. Sci.*, 2013.

[66] C. de Alfonso, M. Caballer, F. Alvarruiz, and V. Hernández, "An energy management system for cluster infrastructures," *Comput. Electr. Eng.*, 2013.

[67] I. Blanquer Espert, V. Hernández García, F. J. Meseguer Anastásio, and J. D. Segrelles Quilis, "Content-based organisation of virtual repositories of DICOM objects," *Future Generation Computer Systems*, vol. 25. pp. 627–637, 2009.

[68] J. D. Segrelles, M. Caballer, E. Torres, G. Moltó, and I. Blanquer, "Platform to Ease the Deployment and Improve the Availability of TRENCADIS Infrastructure," in *7th Iberian Grid Infrastructure Conference (IberGrid)*, 2013.

[69] M. Smit, P. Pawluk, B. Simmons, and M. Litoiu, “A Web Service for Cloud Metadata,” in *2012 IEEE Eighth World Congress on Services*, 2012, pp. 361–368.

Acrónimos

A

ACL	Access Control List
ADD	Application Deployment Description
AHECI	Ad-Hoc Educational Computing Infrastructures
AMI	Amazon Machine Images
API	Application Programming Interface
AWS	Amazon Web Services

B

BECI	Base Educational Computing Infrastructures
BLAST	Basic Local Alignment Search Tool
BNF	Backus-Naur Form

C

CIMI	Cloud Infrastructure Management Interface
CJDL	Cloud Job Description Language
CM	Configuration Manager
CMP	Cloud Management Platform
CS	Cloud Selector

D

DADL	Distributed Application Description Language
DHCP	Dynamic Host Configuration Protocol
DICOM	Digital Imaging and Communication in Medicine

DMTF	Distributed Management Task Force
DNS	Domain Name System
DSL	Domain-Specific Language

E

EBS	Elastic Block Storage
EC2	Elastic Compute Cloud
EC3	Elastic Cloud Computing Cluster
ECU	EC2 Computing Unit
EGI	European Grid Initiative
EMI	European Middleware Initiative

G

GCEU	Google Compute Engine Unit

H

HTC	High-Throughput Computing
HTTP	Hypertext Transfer Protocol

I

IaaS	Infrastructure as a Service
IM	Infrastructure Manager
IMV	Imagen de Maquina Virtual

J

JSON	JavaScript Object Notation

K

KPI	Key Performance Indicators
KVM	Kernel-based Virtual Machine

L

LDAP	Lightweight Directory Access Protocol
LRMS	Local Resource Management Systems
LVM	Logical Volume Manager

M

MV	Máquina Virtual

N

NFS	Network File System
NGS	Next Generation Sequencing
NIST	National Institute of Standards and Technology

O

OCCI	Open Cloud Computing Interface
OCFS2	Oracle Cluster File System (version 2)
ODISEA	On-demand Deployment of virtual Infrastructures to Support Educational Activities
OGF	Open Grid Forum
ONE	OpenNebula
OVF	Open Virtualization Format

P

PaaS	Platform as a Service
PC	Personal Computer

R

RADL	Resource and Application Description Language
RDF	Resource Description Framework

REST	Representational State Transfer
RPC	Remote Procedure Call

S

S3	Simple Storage Service
SaaS	Software as a Service
SCSI	Small Computers System Interface
SDF	Service Description File
SGE	Sun Grid Engine
SL	Scientific Linux
SLA	Service Level Agreement
SO	Sistema Operativo
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SSH	Secure SHell
SSL	Secure Sockets Layer

U

UCSC	University of California Santa Cruz
URL	Uniform Resource Locator

V

VA	Virtual Appliance
VMRC	Virtual Machine Image Repository & Catalog
VO	Virtual Organization
VPC	Virtual Private Cloud

VPN	Virtual Private Network

W

WS	Web Services
WSRF	Web Services Resource Framework

X

XML	eXentsible Markup Language

Y

YAML	YAML Ain't Markup Language