Universidad del Valle

Escuela de Ingeniería de Sistemas y Computación

VirtShell - Framework para aprovisionamiento de soluciones virtuales

Tesis presentada por Carlos Alberto Llano Rodríguez
para obtener el grado de Maestría en Ingeniería con énfasis en Ingeniería de Sistemas

2016

Facultad de Ingeniería

Agradecimientos

- A Dios, por iluminarme en momentos dificiles.
- A mi familia sin ella, no hubiera podido alcanzar esta meta.
- A John Alexander Sanabria, profesor de la Facultad de Ingeniería de la Universidad del Valle, por su inmensa paciencia y apoyo a lo largo de todo el proyecto.
- A mis amigos colaboradores de la Universidad del Valle, por todo el apoyo y por creer siempre en este gran esfuerzo.
- A mis compañeros y amigos por sus palabras de aliento y constante apoyo.
- A todas aquellas personas que de una u otra forma colaboraron en la realización del presente trabajo.

Índice general

Agradecimientos **2**

1. Introducción **6**

2. Planteamiento del problema **9**

2.1. Problema 9

2.2. Objetivo General 10

2.3. Objetivos Específicos 10

3. Estado del arte **11**

3.1. Técnicas de Virtualización 12

3.2. Soluciones de aprovisionamiento 15

4. Arquitectura de VirtShell **21**

4.1. RPC 22

4.2. REST 23

4.3. Características 24

4.4. Módulos 26

4.4.1. Security 26

4.4.2. Managment 27

4.4.3. Provisioning 29
4.4.4. Agents 31

5. Seguridad **32**
5.1. Autenticación 32
5.1.1. Authentication Header 33
5.2. Autorización 35

6. Adminsitración **38**
6.1. Particiones, anfitriones y Ambientes en VirtShell 38
6.1.1. Anfitriones 38
6.2. Instancias 39
6.2.1. Particiones 40
6.2.2. Asociación de anfitriones a particiones 41
6.2.3. División de particiones en ambientes 43
6.2.4. Creación de instancias en un ambiente 44
6.3. Tareas 46
6.4. Propiedades 48

7. Aprovisionamiento **52**
7.1. Almacenamiento y envio de archivos 52
7.2. Imágenes 55
7.2.1. Imágenes tipo ISO 55
7.2.2. Imágenes tipo contenedor 57
7.3. Aprovisionadores 59
7.3.1. Scripts de aprovisionamiento 60
7.3.2. Dependencias 65
7.4. Instalación y Actualización de paquetes 65

8. Agentes **67**

9. API **68**
9.1. Definición de API 68
9.1.1. VirtShell API REST 68

9.2. Formato de entrada y salida . . . 69
9.3. Codigos de error . . . 70
9.4. API Resources . . . 71
9.4.1. Groups . . . 71
9.4.2. Users . . . 76
9.4.3. Partitions . . . 80
9.4.4. Hosts . . . 85
9.4.5. Hosts . . . 89
9.4.6. Instances . . . 96
9.4.7. Tasks . . . 101
9.4.8. Properties . . . 105
9.4.9. Provisioners . . . 110
9.4.10. Images . . . 116
9.4.11. Packages . . . 121
9.4.12. Files . . . 124
9.5. API Calls . . . 128
9.5.1. Start Instance . . . 128
9.5.2. Stop Instance . . . 129
9.5.3. Restart Instance . . . 130
9.5.4. Clone Instance . . . 130
9.5.5. Execute command . . . 131
9.5.6. Copy files . . . 132

10. Conclusiones **135**

A. Disponible en GitHub **137**

B. Roadmap **138**

C. Archivo de respuestas (pressed) **140**

Bibliografía **142**

Índice de figuras

3.1. Máquina virtual 13
3.2. Contenedores 14

4.1. Visión general del framework de VirtShell 27

6.1. Ejemplo de ambientes en un partición 44

Índice de cuadros

5.1. Tipos de permisos 36
5.2. Atributos básicos 36

9.1. Métodos HTTP para groups 71
9.2. Métodos HTTP para users 76
9.3. Métodos HTTP para partitions 80
9.4. Métodos HTTP para enviroments 85
9.5. Métodos HTTP para hosts 89
9.6. Métodos HTTP para instances 96
9.7. Métodos HTTP para tasks 101
9.8. Métodos HTTP para properties 105
9.9. Métodos HTTP para provisioners 110
9.10. Métodos HTTP para images 116
9.11. Métodos HTTP para packages 121
9.12. Métodos HTTP para files 125

CAPÍTULO 1

Introducción

La aparición de ambientes de computación centrados en la nube, los cuales se caracterizan por ofrecer servicios bajo demanda, ha favorecido el desarrollo de diversas herramientas que apoyan los procesos de aprovisionamiento en demanda de servicios y ambientes de computación orientados al procesamiento de tareas de larga duración y manejo de grandes volúmenes de datos. Estos ambientes dinámicos de computación son desarrollados mayormente a través de técnicas de programación ágil las cuales se caracterizan por ofrecer rápidos resultados e integración a gran escala de componentes de software. Es así como los equipos de DevOps [1] se convierten en un elemento fundamental ya que potencia la estabilidad y uniformidad de los distintos ambientes de prueba y producción de modo que los procesos de integración y despliegue se hagan de forma automatizada.

Las herramientas de aprovisionamiento automático de infraestructura son el eje central de estos equipos ya que es a través de ellas que el personal de desarrollo y operaciones son capaces de hablar un mismo lenguaje y establecer los requerimientos y

[1] DevOps consiste en traer las prácticas del desarrollo ágil a la administración de sistema y el trabajo en conjunto entre desarrolladores y administradores de sistemas. DevOps no es una descripción de cargo o el uso de herramientas, sino un método de trabajo enfocado a resultados.

necesidades a satisfacer. Sin embargo, las herramientas actuales de aprovisionamiento adolecen de servicios que faciliten la especificación de infraestructura a través de un API [2] estandarizado que posibilite la orquestación del despliegue de infraestructura a través de Internet.

En este documento se presenta una herramienta de aprovisionamiento con orientación a servicios que permite el despliegue y orquestación de plataformas y servicios a través de un API RESTful [3]. Además de lo mencionado anteriormente, la tesis consta de 8 capítulos más y de 3 apéndices.

El segundo capítulo presenta el planteamiento del problema, en donde se aclara el objeto y alcance del trabajo realizado. La definición del marco teórico que permite entender la importancia de la virtualización en la actualidad, las técnicas de virtualización usadas y la sinopsis de las soluciones de aprovisionamiento mas conocidas actualmente, son presentadas en el tercer capítulo.

En el cuarto capitulo se introduce y elabora la arquitectura planteada en VirtShell. Se describe los requisitos que se tuvieron en cuenta para elaborar la estructura del framework, las alternativas estudiadas y se reseña los módulos y sus características que conforman a VirtShell.

Los siguientes 4 capítulos se encargan de describir cada uno de los módulos diseñados, ilustrando sus funcionalidades y la forma en que interactúan de manera conjunta para administrar la infraestructura y realizar el aprovisionamiento de los recursos virtualizados. Adicionalmente se muestran ejemplos del uso del API.

En el ultimo capitulo se muestra de forma detallada la documentación del API de VirtShell. Se indican los recursos y los métodos HTTP con que cuenta cada módulo, de igual forma se enseñan mas ejemplos de como interactuar con el API.

[2] API: Application Programming Interface, conjunto de subrutinas, funciones y procedimientos que ofrece un software para ser utilizado por otro software como una capa de abstracción.

[3] RESTful hace referencia a un servicio web que implementa la arquitectura REST

Finalmente, los apéndices son utilizados para presentar información relacionada con la implementación del framework.

CAPÍTULO 2

Planteamiento del problema

2.1. Problema

En la actualidad, con la creciente adopción de modelos de computación como el *Cloud Computing* [1] y el *Grid Computing* [2], los ambientes computacionales se han tornado cada vez más sofisticados y complejos, requiriendo de soluciones que traten de manera integral el aprovisionamiento de diferentes servicios sobre ambientes virtuales capaces de atender la variable demanda computacional, a través del despliegue de infraestructuras elásticas de computación.

Hoy en día, se encuentran diversas soluciones que abordan el problema de aprovisionamiento usando diferentes enfoques para el despliegue y orquestación de plataformas y servicios. Sin embargo, los enfoques actuales carecen de mecanismos de comunicación que permitan interoperar entre diferentes aplicaciones; En general, las solu-

[1] conocida también como servicios en la nube, es un paradigma que permite ofrecer servicios de computación a través de una red, que usualmente es Internet.

[2] Un grid es un sistema de computación distribuido que permite coordinar computadoras de diferente hardware y software y cuyo fin es procesar una tarea que demanda una gran cantidad de recursos y poder de procesamiento.

ciones actuales presentan dificultades para ser accedidas en una red, como internet y ejecutadas de manera remota.

En consecuencia, para lograr la interoperabilidad con los actuales modelos de computación, este proyecto propone como objetivo principal, plantear el diseño de un framework orientado al web, cuyas partes soporten una arquitectura que permita el eficiente aprovisionamiento de software de manera automática para ambientes virtualizados. De igual modo, se propone realizar la ejemplificación del framework en un ambiente virtualizado.

Para lograrlo será necesario evaluar diferentes estilos arquitectónicos, realizar un modelo del framework, definir la plataforma en la que se realizará la ejemplificación y definir el mecanismo de aprovisionamiento que se utilizará para aprovisionar ambientes en máquinas virtuales.

2.2. Objetivo General

Diseñar un framework web, que permita el aprovisionamiento de software automático, para ambientes virtualizados.

2.3. Objetivos Específicos

- Evaluar diferentes herramientas de aprovisionamiento que se utilizan en la actualidad.
- Evaluar diferentes estilos arquitecturales que soporten el framewok de aprovisionamiento.
- Evaluar diferentes mecanismos de aprovisionamiento.
- Realizar una ejemplificación del framework.

CAPÍTULO 3

Estado del arte

La tecnología de virtualización y la computación en la nube son dos áreas de investigación muy activas. La investigación que produce estos campos es demasiada para enumerar. Cada una tiene múltiples conferencias de investigación. Por ejemplo, la ACM Symposium on Cloud Computing (SOCC) [1] es uno de los lugares más conocidos en investigación de la computación en la nube y tecnologías de virtualización.

La importancia de estas dos areas en la industria radica en que las nubes han transformado la forma de hacer computación. En general una empresa que use los servicios de una nube no necesita preocuparse tanto acerca de la administración de la infraestructura, las copias de seguridad, el mantenimiento, la depreciación, fiabilidad, rendimiento y tal vez de la seguridad. Estas tareas son ahora realizadas por los proveedores de la nube en otro lugar fuera de la empresa que los contrata.

En la actualidad existen una gran oferta de nubes. Algunas de estas son públicas y están disponibles para cualquiera que esté dispuesto a pagar por el uso de los recursos, las demás son privadas para una organización. Del mismo modo, diferentes nubes

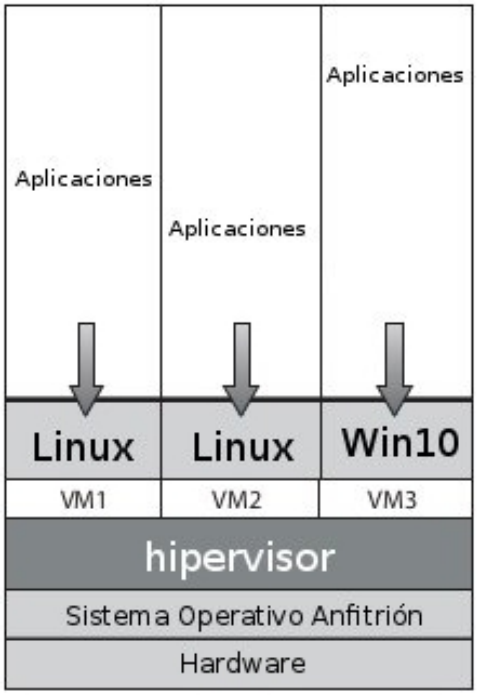
ofrecen cosas diferentes. Algunas dan a sus usuarios el acceso a hardware físico, pero la mayoría permiten virtualizar sus entornos. Algunas ofrecen las máquinas virtuales, desnudas o no, y nada más, pero otras ofrecen software que está listo para utilizar y se pueden combinar de manera interesante, o plataformas que hacen que sea fácil para sus usuarios desarrollar nuevos servicios [2].

En ese contexto, una de las principales características de la computación en la nube es la virtualización, la cual crea la ilusión de múltiples máquinas (virtuales), cada una potencialmente ejecuta un sistema operativo completamente, diferente usando el mismo hardware físico. La virtualización se puede aplicar a computadoras, sistemas operativos, dispositivos de almacenamiento de información, aplicaciones o redes. Esto permite que las empresas ejecuten mas de un sistema virtual, ademas de múltiples sistemas operativos y aplicaciones, en un único servidor, de esta manera se logra economía de escala y una mayor eficiencia.

3.1. Técnicas de Virtualización

Actualmente predominan dos técnicas de virtualización. La primera técnica se denomina virtualización de hardware y consiste en que el software subyacente que ejecuta las máquinas virtuales conocido como hipervisor, crea y corre maquinas virtuales proporcionando una interfaz que es idéntica a la del servidor físico (también conocido como maquina anfitriona). El hipervisor, interactúa directamente con la CPU en el servidor físico, ofreciendo a cada uno de los servidores virtuales una total autonomía e independencia (Figura 3.1). Incluso pueden coexistir en una misma maquina distintos servidores virtuales funcionando con distintos sistemas operativos. Esta técnica es la mas desarrollada y hay diferentes productos que cada fabricante ha ido desarrollando y adaptando, como por ejemplo Xen, KVM, VMWare y VirtualBox.

Figura 3.1: Máquina virtual



La segunda técnica es conocida como virtualización del sistema operativo. En esta técnica lo que se virtualiza es el sistema operativo completo el cual corre directamente sobre la maquina física. A este tipo de máquinas virtuales se les denomina contenedores, los cuales acceden por igual a todos los recursos del sistema (Figura 3.2). Esta técnica tiene una ventaja que es a su vez una desventaja: todas las máquinas virtuales usan el mismo Kernel que el sistema operativo, lo que reduce los errores y multiplica el rendimiento, pero a su vez solo puede haber un mismo tipo de sistema operativo en los contenedores, no se puede combinar Windows, Linux, Etc. Este sistema también es un acercamiento a lo que seria una virtualización nativa [1].

[1]Tipo de virtualización en que intervienen las características del hardware. Los fabricantes preparan, sobre todo, los procesadores para que maquinas virtuales puedan trabajar con ellos más directamente.

Figura 3.2: Contenedores

De hecho, sin importar la técnica de virtualización que se use, la instalación de una maquina virtual (o de un contenedor) requiere normalmente de la generación e instalación de una imagen y a su vez de la instalación y configuración de paquetes de software. Estas tareas generalmente son realizadas por los técnicos de los proveedores de la nube. Cuando un usuario de la nube solicita un nuevo servicio o mas capacidad de computo, el administrador selecciona la imagen apropiada para clonar e instalar en los nodos de la nube. Si no existe una imagen apropiada para los requerimientos del cliente, se crea y configura una nueva que cumpla con la solicitud. Esta creación de una nueva imagen puede ser realizada modificando la imagen mas cercana de las ya existentes. En el momento de la creación optima de la imagen un administrador puede tener dificultades y preguntas como: ¿cuál es la mejor configuración?, ¿cuáles paquetes y sus dependencias deberían ser instaladas? y ¿cómo encontrar una imagen que mejor llene las expectativas?. Esto hace que la automatización y simplificación de este proceso sea una prioridad para los proveedores, ya que la depedencia entre entre paquetes de software y la dificultad de mantenimiento agrega tiempo a la creación

de las máquinas virtuales. En otras palabras, uno de los principales objetivos de los proveedores de la nube es brindar mas flexibilidad y agilidad a la hora de satisfacer los requerimientos de los usuarios finales.

3.2. Soluciones de aprovisionamiento

En el mercado existen muchas soluciones que permiten la interacción con diferentes ambientes de virtualización. Estas soluciones usan diferentes enfoques para realizar despliegues de software en las máquinas virtuales de manera rápida, controlada y automática. Sin embargo la mayoría de las soluciones no tienen la capacidad de manejar de manera simultanea las dos técnicas de virtualización antes mencionadas, algunas se centran solo en manejar máquinas virtuales y otras pocas solo hacen aprovisionamiento sobre contenedores.

Así mismo, hay soluciones de aprovisionamiento que han incorporado su propio lenguaje buscando mayor flexibilidad y fácil configuración de las tareas. Sin embargo, esto implica incorporar una curva de aprendizaje bastante alta, lo que se traduce en un gran esfuerzo inicial para contar con toda la infraestructura automatizada. En ese mismo orden de ideas, existen a su vez, soluciones cuya curva de aprendizaje es mucho menor lo que las hace mas atractivas para muchos ingenieros de Tecnologías de la Información (TI).

Adicionalmente, así como se encuentran soluciones o herramientas de aprovisionamiento de desarrollo propietario que cobran por sus funcionalidades mas importantes o por el numero de máquinas que pueden aprovisionar, existen herramientas de código abierto o de uso libre que permiten trabajar con un número considerable de máquinas virtuales. Al revisar alrededor de 40 diferentes herramientas de aprovisionamiento se logro identificar dos características que no se encuentran en las soluciones actuales. La primera trata de la ausencia de una interfaz o API web para realizar aprovisionamiento remoto y la segunda se refiere a que las herramientas se limitan

a aprovisionar las máquinas virtuales sin ofrecer mecanismos de administración y monitoreo de la red aprovisionada o de los anfitriones que albergan los recursos virtuales.

A continuación se describirán algunas de las herramientas mas conocidas actualmente son descritas a continuación:

Fabric es una herramienta de automatización que usa SSH [2] para hacer despliegues de aplicaciones y administración de tareas. Fabric es una librería gratuita hecha en python y su forma de interactuar es por medio de linea de comandos. Por otra parte permite cargar y descargar archivos que pueden ser ejecutados por su conjunto de funciones [3].

Chef es una de las herramientas más conocidas de automatización de infraestructura de nube, esta escrita en Ruby y Erlang. Utiliza un lenguaje de dominio especifico, expresado también en Ruby para la escritura y configuracion de los recursos que deben ser creados, a esto se le denomina recetas". Estas recetas contienen los recursos que deben ser creados. Chef se puede integrar con plataformas basadas en la nube, como Rackspace, Internap, Amazon EC2, Cloud Platform Google, OpenStack, SoftLayer y Microsoft Azure. Adicionalmente puede aprovisionar sobre contenedores si se instala la librería indicada.

Chef contiene soluciones para sistemas de pequeña y gran escala [9].Es uno de los cuatro principales sistemas de gestión de configuración en Linux, junto con Cfengine, Bcfg2 y Puppet. La version gratuita puede ser utilizada, pero no cuenta con todo el conjunto de caracteristicas, administracion y soporte brindados por la herramienta. Estos se pueden obtener pagando una licencia de uso.

Puppet es una herramienta diseñada para administrar la configuración de sistemas similares a Unix y a Microsoft Windows de forma declarativa. El usuario describe los recursos del sistema y sus estados utilizando el lenguaje declarativo

[2]SSH (Secure SHell, en español: intérprete de órdenes seguro) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red

que proporciona Puppet. Esta información es almacenada en archivos denominados manifiestos Puppet. Puppet descubre la información del sistema a través de una utilidad llamada Facter, y compila los manifiestos en un catalogo especifico del sistema que contiene los recursos y las dependencias de dichos recursos, estos catálogos son ejecutados en los sistemas de destino [10]. Puppet es de uso gratuito para redes muy pequeñas de hasta solo 10 nodos.

Juju es una herramienta de configuración y administración de servicios en nubes publicas. Permite crear ambientes completos con unos pocos comandos, cuenta con cientos de servicios pre-configurados y disponibles en la tienda de juju. Se puede usar a través de una interfaz gráfica o de linea de comandos. Juju permite re-crear un ambiente de producción en portátiles usando contenedores enfocado a pruebas. El uso de juju es gratuito pero se debe pagar por el uso de la nube publica [4].

CFEngine es un sistema basado en el lenguaje escrito por Mark Burgess, diseñado específicamente para probar y configurar software. CFEngine es como un lenguaje de muy alto nivel. Su objetivo es crear un único archivo o conjunto de archivos que describen la configuración de cada máquina de la red. CFEngine se ejecuta en cada host, y analiza cada archivo (o archivos), que especifica una política para la configuración del sistema. La configuración de la máquina es verificada contra el modelo y, si es necesario, cualquier desviación de la configuración es corregida. [11]

CFEngine cuenta con una versión gratuita y la versión empresarial que cuenta con interfaz gráfica, soporte y reportes.

Ansible es una herramienta de código libre desarrollada en python y comercialmente ofrecida por AnsibleWorks los cuales la definen como un motor de orquestación muy simple que automatiza las tareas de despliegue. Ansible no usa agentes, solo necesita tener instalado Python en las máquinas hosts y las tareas las realiza por medio de ssh. Ansible puede trabajar mediante un solo archivo de configuración que contendría todo o por medio de varios archivos organizados en una estructura de directorios [7].

Bcfg2 esta escrito en Python y permite gestionar la configuración de un gran numero de ordenadores mediante un modelo de configuración central. Bcfg2 funciona con un modelo simple de configuración del sistema, modelando elementos intuitivos como paquetes, servicios y archivos de configuración (así como las dependencias entre ellos). Este modelo de configuración del sistema se utiliza para la verificación y validación de las máquinas, permitiendo una auditoria robusta de los sistemas desplegados. La especificación de la configuración de Bcfg2 está escrita utilizando un modelo XML declarativo. Toda la especificacion puede ser validada utilizando los validadores de esquema XML ampliamente disponibles. Bcfg2 no tiene soporte para contenedores. Es gratuito y cuenta con una lista limitada de plataformas en las cuales trabaja bien.[12]

Cobbler es una plataforma que busca el rápido despliegue de servidores y en general computadores en una infra-estructura de red por medio de linea de comandos, se basa en el modelo de scripts y cuenta con una completa base de simples comandos, que permite hacer despliegues de manera rápida y con poca intervención humana. Cobbler es capaz de instalar máquinas físicas y máquinas virtuales. Cobbler, es una pequeña y ligera aplicacion, que es extremadamente facil de usar para pequeños o muy grandes despliegues. Es de uso gratuito y no cuenta con soporte para contenedores. [13]

SmartFrog es un framework para servicios de configuración, descripción, despliegue y administración del ciclo de vida de máquinas virtuales. Consiste de un lenguaje declarativo, un motor que corre en los nodos remotos y ejecuta plantillas escritas en el lenguaje de SmartFrog y un modelo de componentes. El lenguaje soporta encapsulación (que es similar a las clases de python), herencia y composición que permite personalizar y combinar configuraciones. SmartFrog, permite enlaces estáticos y dinámicos entre componentes, que ayudan a soportar diferentes formas de conexión en tiempo de despliegue.

El modelo de componentes, administra el ciclo de vida a través de cinco estados: instalado, iniciado, terminado y fallido. Esto permite al motor del SmartFrog detectar fallas y reiniciar automáticamente re-despliegues de los componentes [6].

SmartFrog es desarrollado y mantenido por un equipo de investigación en los laboratorios de Hewlett-Packard en Bristol, Inglaterra, así como por el laboratorio Europeo de Hewlett-Packard y por contribuciones de otros usuarios de SmartFrog y desarrolladores externos a HP. Se utiliza en la investigación de HP, específicamente en la automatización de la infraestructura y automatización de servicios, ademas de ser solo utilizado en determinados productos de HP.

Amazon EC2 es un API propietario de Amazon que maneja un enfoque manual, el cual permite desplegar imágenes de máquinas virtuales conocidas como AMI (Amazon Machine Images) [8], que son las imágenes que se utilizan en Amazon para arrancar instancias virtuales. El concepto de las AMIs [3] es similar a las máquinas virtuales de otros sistemas. Básicamente estan compuestas de una serie de ficheros de datos que conforman la imagen y luego un xml que especifica ciertos valores necesarios para que sea una imagen valida para Amazon.

Docker composer permite describir un conjunto de contenedores que se relacionan entre ellos. Docker composer define una aplicación multicontenedor en un archivo con las mismas propiedades que se indicarían en un archivo individual de docker. Docker composer usa archivos en formato yaml para describir las características de los servicios en cada contenedor [5]. Es completamente gratuito.

Vagrant es una herramienta de linea de comando que permite la creación y configuración de entornos de desarrollo virtualizados. Originalmente se desarrolló para VirtualBox y sistemas de configuración tales como Chef, Salt y Puppet. Sin embargo desde la versión 1.1 Vagrant es capaz de trabajar con múltiples proveedores, como VMware, Amazon EC2, LXC y DigitalOcean [14].

Vagrant ofrece múltiples opciones para realizar aprovisionamientos, desde scritps en shell hasta complejos sistemas de configuración.

[3] La AMI de Amazon es una imagen mantenida y compatible que ofrece Amazon Web Services para su uso en Amazon Elastic Compute Cloud (Amazon EC2).

SaltStack es un sistema de manejo de configuración cuyo objetivo es garantizar que un servicio este corriendo o que una aplicación haya sido instalada o desplegada. Salt está construido en Python y al igual que Chef, CFEngine y Puppet utiliza un esquema de cliente (salt minions) - servidor (salt master), cuyo método de conexión con los minions se realiza a través de un broker messages [4] llamado ZeroMQ (0MQ), que no solo garantiza una conexión segura sino que la hace confiable y rápida. Salt tiene una versión gratuita llamada Salt Open sin embargo para obtener todos los beneficios se debe pagar la versión empresarial. Salt cuenta con soporte para contenedores. [15]

[4] es un programa intermediario que traduce los mensajes de un sistema desde un lenguaje a otro, a través de un medio de telecomunicaciones.

CAPÍTULO 4

Arquitectura de VirtShell

VirtShell Framework es concebido como una plataforma que proporciona herramientas para la automatización y gestión de infraestructura, facilitando tareas como la creación, despliegue, mantenimiento y monitoreo tanto de recursos virtuales como físicos vía web. Así mismo, es pensada para que cualquier desarrollo de software con acceso a Internet (sitio web, aplicación móvil, etc.) pueda interactuar con la infraestructura virtual tan solo consumiendo un API [1] de Internet.

Las motivaciones mencionadas conducen a los requisitos para una arquitectura destinada a separar claramente responsabilidades, usando protocolos abiertos, modificables, escalables, y al mismo tiempo adecuado para la creación rápida de nuevas acciones.

En la búsqueda del adecuado estilo arquitectural para el API de VirtShell, se evaluaron dos estilos de servicios web: el estilo *Remote Procedure Call* (RPC) y el estilo arquitectural REST (*Representational State Transfer*) [16]. La evaluación dio como resultado que el estilo arquitectural que mejor se acomodaba a los requisitos plantea-

[1] API (Application Program Interface) es un conjunto de rutinas, protocolos y herramientas para la construcción de aplicaciones de software

dos era el estilo arquitectural REST, debido a que es un estilo nativo del Web, lo que hace que la información disponible esta regida por las mismas normas que rigen los sitios web, ademas, el estilo ofrece mejor escalabilidad, acoplamiento y rendimiento. Un detallado comparativo entre los dos estilos se encuentra en [22]. A continuación se muestran a grandes rasgos las diferencias entre ambos estilos y razones adicionales para la preferir el estilo REST.

4.1. RPC

RPC se caracteriza generalmente como un único URI a través del protocolo HTTP en el que se pueden llamar muchas operaciones, por lo general solo se usan las operaciones GET y POST. Cuando se pasa una solicitud estructurada, esta incluye el nombre de la operación a invocar y los argumentos que desea pasar a la operación; la respuesta será devuelta también en un formato estructurado.

Una cosa a tener en cuenta es que por lo general RPC hace todo el informe de errores en el cuerpo de la respuesta; el código de estado HTTP no variará, lo que significa que hay que fijarse en el valor de retorno para determinar si se ha producido un error.

Muchas implementaciones de RPC también proporcionan documentación para sus usuarios finales a través del protocolo en sí. Por ejemplo para SOAP lo hace a través del WSDL. Esta característica de auto-documentado puede proporcionar información muy valiosa para el consumidor sobre cómo interactuar con el servicio.

En resumen, los puntos a tener en cuenta acerca de RPC son:

- Un extremo de servicio, muchas operaciones.
- Un extremo de servicio, un método HTTP (normalmente POST).
- Formato de solicitud predecible estructurada, formato de respuesta estructurada y predecible.
- Formato de informe de errores predecibles estructurada.

- Documentación estructurada de operaciones disponibles.

Dicho todo esto, RPC es a menudo un mala elección para hacer APIs web porque:

- No se puede determinar a través de la URI la disponibilidad de varios recursos.
- Falta de almacenamiento en caché de HTTP
- La imposibilidad de usar verbos HTTP nativos para operaciones comunes
- Falta de códigos de respuesta, se requiere la introspección de los resultados para determinar si se ha producido un error.
- Los clientes no podrán consumir formatos de serialización alternativos.
- Los formatos de mensajes a menudo imponen restricciones innecesarias sobre los tipos de datos que se pueden enviar o devolver.

En pocas palabras, RPC no utiliza las capacidades completas del protocolo HTTP.

4.2. REST

REST es un estilo de arquitectura de software que proporciona un enfoque practico y consistente para solicitar y modificar datos en torno a la especificación del protocolo HTTP.

El termino REST es la abreviatura para “Representational State Transfer.”, el cual aprovecha las fortalezas de los protocolos HTTP y HTTPS. Un buen API REST debe constar de:

- Usar URIs como identificadores únicos de los recursos.
- Aprovechar el espectro completo de verbos HTTP para las operaciones sobre los recursos.

- Permitir el manejo de varios formatos de representación de recursos.

- Proporcionar vinculación entre los recursos para indicar las relaciones. (Por ejemplo, enlaces hipermedia, como los encontrados en los antiguos documentos HTML plano)

Todo esta teoría dice cómo deben actuar los servicios REST, pero dice muy poco acerca de la forma de implementarlos. REST es más una consideración arquitectónica; sin embargo, esto significa que al momento de diseñar un API REST se debe considerar muchas opciones, algunas como que formato se va a exponer, como se reportaran los errores, como se comunicara los métodos HTTP disponibles, como se manejaran características de autenticación, como se suministraran las credenciales en cada petición, etc.

En pocas palabras, un API REST proporciona una increíble flexibilidad y potencia, pero requiere de tomar muchas decisiones con el fin de proporcionar una sólida experiencia y calidad para los consumidores.

4.3. Características

VirtShell es un framework de código abierto y bajo la licencia BSD, que permite utilizarlo para proyectos de cualquier tipo, incluso comerciales, sus características principales son:

Programable VirtShell esta orientado a realizar el aprovisionamiento de sus instancias principalmente por medio de scripts escritos en shell, permitiendo aprovechar todas las estructuras y utilidades del lenguaje de programación. Sin embargo, el lenguaje de shell no es de uso obligatorio, el metodo de aprovisionamiento puede ser el de la preferencia del usuario.

Repetible VirtShell ofrece herramientas para que los scritps de aprovisionamiento sean configurables y puedan ser ejecutados varias veces en diferentes ambientes de desarrollo o producción.

Modular VirtShell es un framework organizado de forma modular. Los módulos se encuentran agrupados en categorías que ofrecen las herramientas necesarias para la administración y aprovisionamiento de múltiples recursos virtuales. De igual manera y dada las características de REST, los módulos están diseñados para que se puedan dividir en diferentes servidores obteniendo ”micro APIs”, lo que permite dividir los procesos y atender diferentes tipos de operaciones del API.

Seguro VirtShell provee varias capacidades y servicios para aumentar la privacidad y el control de acceso a los diferentes recursos. Los servicios de seguridad permiten crear redes y controlar el acceso a las instancias creadas, así como definir y administrar políticas de acceso a usuarios y permisos sobre cualquier recurso del sistema como por ejemplo scripts de creación y aprovisionamiento.

Extensible Al ser VirtShell de código abierto, el API puede modificarse, crecer fácilmente y versionarse de diferentes maneras. Adicionalmente, VirtShell fue diseñado con la idea de cargar código dinámicamente, permitiendo extender el comportamiento del framework agregando plugins en tiempo de ejecución. Asi mismo, VirtShell permite extender el comportamiento del shell desplegando comandos propios que proporcionan ahorro en tiempo y en complejidad.

Inyección de dependencias virtuales VirtShell adopta la idea del patrón de Inyección de Dependencias [2] [17] para conseguir scripts de aprovisionamiento mas desacoplados. De esta manera facilita la configuración de las dependencias que tiene un recurso virtual de otras máquinas virtuales. Para ello, el framework permite declarar el listado de dependencias de recursos virtuales que tiene un script de aprovisionamiento encargándose del correcto acople entre los diferentes recursos virtuales.

Interoperable Al seguir el estilo arquitectónico REST y contar con la documentación detallada sobre cada uno de los recursos y urls que expone el API de VirtShell,

[2] es un patrón de diseño orientado a objetos, en el que se suministran objetos a una clase en lugar de ser la propia clase quien cree el objeto. El término fue acuñado por primera vez por Martin Fowler.

se logra una capacidad clave para la administración remota de la infraestructura virtual. Esta capacidad se refiere al hecho de poder desarrollar aplicaciones en cualquier plataforma y para cualquier dispositivo electrónico, lo que permite funcionar con otros productos o sistemas existentes o futuros.

4.4. Módulos

VirtShell Framework consiste de características organizadas en 13 módulos. Estos módulos son agrupados en Seguridad, Administración y Aprovisionamiento. Estos elementos se usan de manera separada pero trabajan juntos para proveer la información necesaria para que los agentes realicen su trabajo en los hosts que albergaran los recursos virtuales, como se muestra en la figura 4.1.

Las siguientes secciones detallan los módulos disponibles para cada característica.

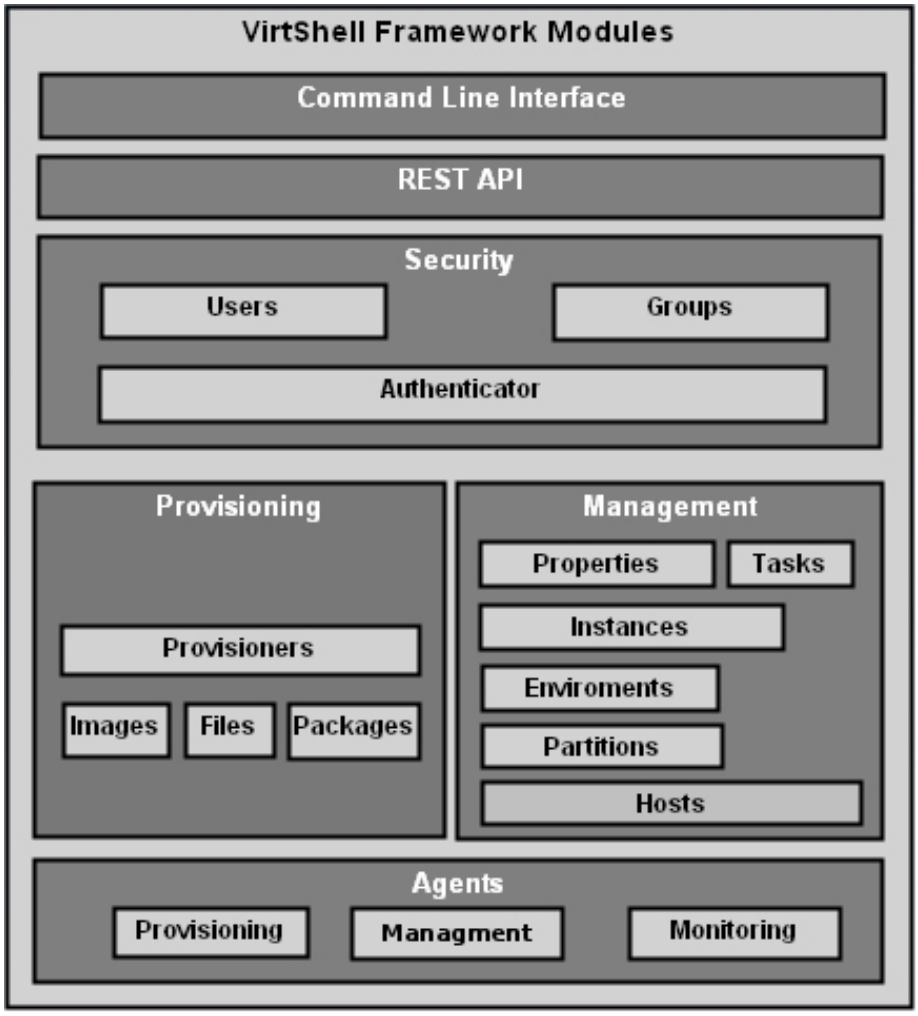
4.4.1. Security

Seguridad consiste de los módulos de *users* (usuarios), *groups* (grupos) y el modulo *authenticator* (de autenticación). El control de los usuarios y grupos son elementos clave en la administración del framework. Los **Usuarios** pueden ser personas reales, es decir, cuentas ligadas a un usuario físico en particular o cuentas que existen para ser usadas por aplicaciones específicas.

Los **Grupos** son expresiones lógicas de organización, reuniendo usuarios para un propósito común. Los usuarios dentro de un mismo grupo pueden leer, escribir o ejecutar los recursos que pertenecen a ese grupo.

El módulo de **autenticación** soporta el proceso por el cual cuando un usuario se presenta a la aplicación puede validar su identidad. Este módulo es quien decide0 si el usuario tiene permiso para ingresar al sistema y el nivel de acceso a un recurso dado. En el capitulo 4 se detalla el proceso de autenticación y autorización.

Figura 4.1: Visión general del framework de VirtShell



4.4.2. Managment

Administración consiste de los modulos *hosts* (anfitriónes), *partitions* (particiones), *enviroments* (ambientes), *instances* (instancias), *properties* (propiedades) y *tasks* (tareas).

El módulo de **anfitriónes** lleva registro de nodos físicos, servidores o máquinas virtuales, que se encuentren conectados a la red y que permitan albergar instancias virtuales. Los anfitriónes son clasificados de acuerdo a diferentes combinaciones de

CPU, memoria, almacenamiento y capacidad de trabajo en red, dando flexibilidad para elegir la opcion mas adecuada para las necesidades de las aplicaciones destino. En otras palabras el tipo de anfitrión determina el hardware del nodo físico que sera usado por los recursos virtuales.

El módulo de **particiones** permite dividir los anfitriones en secciones aisladas de disponibilidad. Cada partición puede ser usada para definir áreas geográficas separadas o simplemente para dividir los nodos físicos en subgrupos destinados a diferentes usos o equipos de trabajo o tipos de clientes.

El módulo de **ambientes** permite dividir lógicamente una partición en subredes de trabajo más pequeñas, con lo que se crean grupos más pequeños con diferentes fines. En los ambientes de trabajo se configuran los usuarios que tienen permiso para interactuar trabajar con el.

El módulo de **instancias** es un recurso virtual o máquina virtual o contenedor con parámetros y capacidades definidas en la cual se puede instalar el software deseado. Un usuario puede crear, aprovisionar, actualizar y finalizar instancias en VirtShell tanto como necesite dando la sensación de elasticidad [3] de la red.

El módulo de **propiedades** permite consultar información de sistema, de las instancias o de los anfitriones físicos. La información que puede ser consultada es toda aquella que el sistema tenga disponible o que se pueda consultar por medio de comandos de sistema o comandos propios de VirtShell. Ejemplos de información que se puede consultar por medio de las propiedades es porcentajes de memoria y cpu usada, numero de procesos en ejecución, etc. Las propiedades pueden ser consultadas en una sola maquina, o simultáneamente en varias maquinas, o a un conjunto de maquinas de acuerdo a prefijos en su nombre.

Finalmente, el módulo de **tareas** da la información y estado sobre las diferentes tareas o trabajos ejecutados en el sistema. Debido a que el medio para interactuar

[3] La elasticidad es una de las propiedades fundamentales de la nube. La elasticidad consiste en la potencia de escalar los recursos informáticos ampliándolos y reduciéndolos dinámicamente.

con los módulos de VirtShell es a través de un API REST, una petición de creación de un nuevo recurso virtual, puede ser largo si el aprovisionamiento involucra varias maquinas. Para evitar, tener una petición esperando respuesta, VirtShell crea una tarea que sera ejecutada de manera asincrónica, dando como respuesta, un identificador de la tarea, para que esta pueda ser consultada posteriormente y conocer el estado de la petición.

4.4.3. Provisioning

Aprovisionamiento consiste de los módulos *provisioners* (aprovisonadores), *images* (imágenes), *files* (archivos) y *packages* (paquetes).

El módulo de **aprovisionadores** define un marco de aprovisionamiento de un recurso virtual y contiene las configuraciones necesarias para apoyar ese marco. La configuración fundamental de un aprovisionador, es la ruta del repositorio git [4], donde se encuentran los scripts y archivos necesarios para realizar el aprovisionamiento. Del mismo modo la forma de ejecutar los scripts, hace parte de la configuración básica. Para ser mas ligero a VirtShell, los scripts de aprovisionamiento, deben estar registrados en un repositorio git público. VirtShell se encarga de descargar el repositorio y de ejecutar los scripts de aprovisionamiento, de acuerdo a la configuración especificada.

Adicionalmente, los aprovisonadores cuentan con una manera de especificar las dependencias del nuevo recurso virtual. VirtShell se encarga de resolver las dependencias antes de realizar el aprovisionamiento del nuevo recurso, a su vez, suministra información de ellas a los scripts de aprovisionamiento si estos lo requieren.

VirtShell utiliza como lenguaje de referencia, para la creación de scripts de aprovisionamiento, el lenguaje shell. Este cuenta con los recursos suficientes para interactuar con los diferentes sistemas operativos de las instancias. Sin embargo estos comandos se pueden extender usando el paquete de comandos propios de VirtShell, los cuales pueden abstraer muchas de las operaciones del shell. Esto facilita la es-

[4] git es un software de control de versiones diseñado por Linus Torvalds

critura de los mismos o permite hacerlos independientes del sistema operativo en el cual van a ejecutarse.

El módulo de **imágenes** proporciona la información necesaria de las imágenes que se encuentran registradas en el sistema. Cada vez que se crea un nuevo recurso virtual en un anfitrión, se especifica el nombre de la imagen almacenada en el sistema que sera usada, de una de las que se encuentra en el sistema.

Las imágenes que se manejan en VirtShell son de dos tipos: ISO [5] y para contenedores. Las de tipo ISO, se encuentran guardadas en el repositorio de VirtShell, y su uso se enfoca a maquinas virtuales que interactuan con hypervisors. Las imágenes de tipo contenedor, se emplean para tecnologías de visualización de sistema operativo como LXC y Docker. Estas son descargadas automáticamente de los repositorios de dominio público de los diferentes proveedores.

El módulo de imágenes cuenta también, con la característica de crear automáticamente nuevas imágenes de tipo ISO a partir de las *releases* (liberaciones) base de las diferentes distribuciones de sistemas operativos linux. Una vez creada la nueva ISO esta sera guardada en el repositorio interno para su posterior uso.

El módulo de **archivos** proporciona una manera de almacenar archivos en VirtShell, los cuales pueden ser usados para almacenar

El módulo de **archivos** proporciona una manera de almacenar archivos en VirtShell, los cuales pueden ser usados para almacenar información necesaria para crear imágenes o para enviarlos a uno o mas recursos virtuales de manera simultánea en un directorio especificado. Adicionalmente, permite especificar los permisos que tendrán los archivos.

El módulo de **paquetes** facilita realizar funciones tales como la instalación de nuevos paquetes de software y actualización de paquetes existentes en uno o mas recursos

[5] Una imagen ISO es un archivo informático donde se almacena una copia o imagen exacta de un sistema de archivos o ficheros de un disco óptico, normalmente un disco compacto (CD) o un disco versátil digital (DVD).

virtuales de manera simultanea.

4.4.4. Agents

Los Agentes son servicios que se ejecutan localmente en cada anfitrión, y que se encuentra bajo la gestión de VirtShell. Estos son instalados y configurados en cada anfitrión de manera automática. Los agentes actúan con un cierto grado de autonomía con el fin de completar tareas en nombre del servidor.

CAPÍTULO 5

Seguridad

5.1. Autenticación

La autenticación es el proceso de demostrar la identidad al sistema. La cual es un factor clave para las decisiones de control de acceso. Las solicitudes HTTP a VirtShell, se conceden o deniegan en parte sobre la base de la identidad del solicitante.

El VirtShell el API REST utiliza un esquema HTTP personalizado basado en una llave-HMAC (Hash Message Authentication Code) para la autenticación. Para autenticar una solicitud, primero se concatenan los elementos seleccionados de la solicitud para formar una cadena. A continuación se utiliza una clave secreta de acceso para calcular el HMAC de esa cadena. Informalmente a este proceso se le denomina ̈la firma de la solicitud; y al resultado del algoritmo HMCA se le refiere como la "firma", ya que simula las propiedades de seguridad de una firma real. Esta última se agrega como un parámetro de la petición utilizando la sintaxis descrita en esta sección.

Cuando el sistema recibe una solicitud fehaciente, se obtiene la clave secreta de

acceso y se utiliza de la misma manera que se calcula una ”firma”del mensaje que recibió. A continuación se compara la firma que se calcula con la firma presentada por el solicitante. Si estas coinciden, el sistema llega a la conclusión de que el solicitante tiene acceso a la clave secreta y por lo tanto esta autorizado para conectarse con el servidor de VirtShell. Si las dos firmas no coinciden, la solicitud se descarta y el sistema responde con un mensaje de error.

Ejemplo de una petición autenticada:

Listing 5.1: Petición HTTP con firma

```
1 GET /api/virtshell/provisioners/database_prov_pdn HTTP/
    1.1
2 Host: host1.edu.co
3 Date: Fri, 01 Jul 2011 19:37:58 +0000
4
5 Authorization: 0PN5J17HBGZHT7JJ3X82:
    frJIUN8DYpKDtOLCwoHGTY/45U
```

5.1.1. Authentication Header

En VirtShell el encabezado de autorización HTTP tiene la siguiente forma:

Authorization: UserId:Signature

Los usuarios tendrán un ID de clave de acceso (VirtShell Access Key ID) y una clave secreta de acceso (VirtShell Secret Access Key) cuando se registran. Para la petición de autenticación, el VirtShell Access Key Id identifica la clave secreta que se utilizó para calcular la firma, y el usuario que realiza la solicitud.

Para la firma de los elementos de la petición se usa el RFC 2104 HMAC-SHA1 [1] [18], por lo que la parte de la firma de la cabecera de autorización variará de una

[1] En la criptografía, un código de autentificación de mensajes en clave-hash (HMAC) es una construcción específica para calcular un código de autentificación de mensaje (MAC) que implica una función hash criptográfica en combinación con una llave criptográfica secreta.

petición a otra. Si la solicitud de la firma calculada por el sistema coincide con la firma incluida en la solicitud, el solicitante habrá demostrado la posesión de la clave secreta de acceso. La solicitud será procesada bajo la identidad, y con la autoridad, de la promotora que emitió la clave.

A continuación se muestra la pseudo-gramática que ilustra la construcción de la cabecera de la solicitud de autorización (\n significa el punto de código Unicode U +000 comúnmente llamado salto de línea).

Listing 5.2: cabecera de una solicitud de autorización

```
1  Authorization = VirtShellUserId + ":" + Signature;
2
3  Signature = Base64( HMAC-SHA1( UTF-8-Encoding-Of(
      YourSecretAccessKeyID, StringToSign ) ) );
4
5  StringToSign = HTTP-Verb + "\n" +
6  Host + "\n" +
7  Content-MD5 + "\n" +
8  Content-Type + "\n" +
9  Date + "\n" +
10 CanonicalizedResource;
11
12 CanonicalizedResource = <HTTP-Request-URI, from the
      protocol name up
13 to the query string (resource path)>
```

HMAC-SHA1 es un algoritmo definido por la RFC 2104 (ver la RFC 2104 con llave Hashing para la autenticación de mensajes [18]).

El algoritmo toma como entrada dos cadenas de bytes: una clave y un mensaje. Para la solicitud de autenticación, se utiliza la clave secreta (YourSecretAccessKeyID) como la clave, y la codificación UTF-8 del StringToSign como el mensaje. La salida de

HMAC-SHA1 es también una cadena de bytes, llamado .[el] resumen". El parámetro de la petición de la firma se construye codificado en Base64.

5.1.1.0.1. Solicitud canónica para firmar Cuando el sistema recibe una solicitud autenticada, compara la solicitud de firma calculada con la firma proporcionada en la solicitud de StringToSign. Por esta razón, se debe calcular la firma con el mismo método utilizado por VirtShell. A este proceso de poner una solicitud en una forma acordada para la firma se denomino çanonización".

5.1.1.0.2. Tiempo de sello Un sello de tiempo [2] válido (utilizando el HTTP header Date) es obligatorio para solicitudes autenticadas. Por otra parte, el tiempo del sello enviado por un usuario, que se encuentra incluido en una solicitud autenticada, debe estar dentro de los 15 minutos de la hora del sistema cuando se recibe la solicitud. En caso contrario, la solicitud fallará con el código de estado de error RequestTimeTooSkewed. La intención de estas restricciones es limitar la posibilidad de que solicitudes interceptadas pueden ser reproducidas por un adversario. Para una mayor protección contra las escuchas, se debe utilizar el transporte HTTPS para solicitudes autenticadas.

5.2. Autorización

VirtShell es un framework multi-usuario que ofrece protección a recursos basado en los conceptos de permisos de Unix. El mecanismo de protección determina que usuarios están autorizados para acceder a los recursos de: archivos, imágenes, instancias y aprovisionadores, presentes en el sistema.

Al igual que en Unix, la técnica usada para ofrecer protección a los recursos consiste en hacer que el acceso dependa de la identidad del usuario. El esquema usado es una lista de acceso condensada por cada recurso que se desea proteger. La lista de acceso se divide en tres grupos de caracteres, representando los permisos del usuario propietario, del grupo propietario, y de los otros, respectivamente, como se muestra

[2] fecha y hora obtenida del sistema en que se genera la petición HTTP

en la tabla 5.1.

Cuadro 5.1: Tipos de permisos

Permisos	**Pertenece**
rwx———	usuario
—r-x—	grupo
———r-x	otros

Por ejemplo, los caracteres -rw-r–r– indican que el usuario propietario del recurso tiene permisos de lectura y escritura, pero no de ejecución (rw-), mientras que los usuarios que pertenecen al grupo propietario y los demás usuarios solo tienen permiso de lectura (r– y r–). Mientras tanto, los caracteres rwxrwx— indican que el usuario propietario del recurso y todos los usuarios que pertenecen al grupo propietario tienen permisos de lectura, escritura y ejecución (rwx y rwx), mientras que los demás usuarios no pueden acceder (—).

La es siguiente una descripción de los tres atributos básicos que se manejan en VirtShell:

Cuadro 5.2: Atributos básicos

Atributo	**Descripción**
Lectura	Permite a un usuario ver el contenido de cualquier recurso.
Escritura	Permite a un usuario crear, modificar y eliminar un recurso.
Ejecución	Permite a un usuario ejecutar instancias virtuales. Por ejemplo: iniciar, detener, pausar, clonar o actualizar paquetes. (El usuario también debe tener permiso de lectura).

Un ejemplo que muestra la asignación de permisos para una instancia es el siguiente:

```
1 curl -sv -X POST \
2   -H 'accept: application/json' \
```

```
3   -H 'X-VirtShell-Authorization: UserId:Signature' \
4   -d '{ "name": "transactional_log",
5         "memory": 1024,
6         "cpus": 2,
7         "hdsize": "2GB",
8         "operating_system": "ubuntu_server_14.04.2_amd64"
            ,
9         "description": "Server transactional only for
            store logs",
10        "provisioner": "all_backend",
11        "host_type": "GeneralPurpose",
12        "driver": "lxc",
13        "permissions": "rwx------"
14      }' \
15    'http://localhost:8080/virtshell/api/v1/instances'
```

Como se observa, en la información enviada en la petición POST al servidor HTTP de VirtShell, el usuario esta solicitando crear una instancia en donde especifica que solo el propietario tiene permiso para interactuar con ella. Cabe aclarar que si no se especifican los permisos en la información enviada, VirtShell asigna todos los permisos al recurso creado, dejándolo publico para todos los usuarios del sistema.

CAPÍTULO 6

Adminsitración

La capa de Administración de VirtShell proporciona una infraestructura de servicios para la gestión de cualquier dispositivo registrado y creado a traves del sistema. Este capítulo busca darle explicación a las funcionalidades de administración.

6.1. Particiones, anfitriones y Ambientes en VirtShell

En VirtShell hay tres conceptos que son muy importantes y se extienden a través de todos los servicios: las particiones, los ambientes y las instancias. Las tres se asocian con la mayoría de los componentes en VirtShell, y el dominio de ellos es crucial para una buena administración de los dispositivos.

6.1.1. Anfitriones

Los anfitriones no son mas que nodos físicos, servidores o máquinas virtuales, que alojaran recursos virtuales. VirtShell ofrece la posibilidad de clasificarlos de acuerdo a combinaciones de capacidad de CPU, memoria, almacenamiento y red. El objetivo que busca la clasificación es proporcionar flexibilidad para elegir la combinación de recursos adecuada para las aplicaciones.

Los tipos de anfitriones se agrupan en familias basadas en perfiles de aplicación de destino. Estos grupos incluyen: de propósito general, con procesadores de alto desempeño, de memoria optimizada y de almacenamiento optimizado.

Propósito general Esta familia proporciona un equilibrio de recursos informáticos, de memoria y red, por lo que constituye una buena opción para muchas aplicaciones.

Procesadores de alto desempeño Esta familia ofrece procesadores que alcanzan alto desempeño en tareas complejas.

Memoria optimizada Esta familia esta optimizada para aplicaciones con un uso intenso de la memoria.

Almacenamiento optimizado Esta familia promete anfitriones con alta capacidad de almacenamiento, optimizado para un desempeño de E/S muy alto.

6.2. Instancias

A un servidor virtual en VirtShell, se le denomina instancia, las cuales pueden ser maquinas virtuales que corren sobre algún hipervisor o también pueden ser contenedores que se ejecutan directamente sobre el sistema operativo del anfitrión.

La elección de la tecnología de virtualización depende de las preferencias u objetivos que se busque con las aplicaciones de la instancia. En la actualidad VirtShell soporta Virtualbox como hipervisor para maquinas virtuales y para los contenedores soporta LXC [1] [19] y Docker [2] [21].

[1] LXC (Linux Containers) es una tecnología de virtualización a nivel de sistema operativo (SO) para Linux.

[2] Docker es un proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de Virtualización a nivel de sistema operativo en Linux.

6.2.1. Particiones

Las particiones consisten de uno o más anfitriones, los cuales pueden ser nodos físicos, servidores o incluso máquinas virtuales. El objetivo principal que busca una partición, es agrupar las máquinas que albergaran recursos virtuales en partes aisladas, de tal manera que una partición no tenga acceso a los recursos de otra. Estas partes pueden pueden estar ubicadas en un mismo sitio físico o por el contrario pueden estar distribuidas en diferentes zonas geográficas de todo el mundo.

Si solo se cuenta con un numero fijo de maquinas (o anfitriones) ubicadas en un mismo sitio físico como por ejemplo un datacenter [3], lo que se obtiene con las particiones es la posibilidad de dividir esas maquinas en subgrupos que puedan ser destinados para diferentes equipos o divisiones dentro de una organización.

Al contar con maquinas distribuidas en diferentes zonas geográficas la elección de una partición u otra se basa principalmente en la cercanía de los visitantes o clientes, ya que a menor distancia entre los servidores y ellos, menores son los tiempos de respuesta y mejor la experiencia de usuario.

Las particiones también favorecen la disponibilidad. Si se distribuyen sus instancias a través de múltiples particiones y una instancia falla, puede diseñar su aplicación para que una instancia en otra partición pueda atender las peticiones.

Cuando se crea una nueva partición, VirtShell la crea completamente vacia, sin anfitriones. Para asociar anfitriones a una partición se debe crear un anfitrión y vincularlo con la partición como se verá mas adelante en este mismo capítulo. Un ejemplo de como crear una partición usando el API se muestra en el siguiente código:

[3]Un data center también llamado centro de datos es un espacio acondicionado especialmente para contener los equipos y sistemas de TI

Listing 6.1: Petición HTTP para crear una partición

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{
       "name": "development_co",
       "description": "Collection of servers oriented to
          development team in colombia."
      }' \
   'http://localhost:8080/api/virtshell/v1/partitions'
```

En este ejemplo se muestra como crear una partición con nombre developtment_co, la cual se encuentra sin anfitriones asociados. En la siguiente sección se explicara la forma de asociar anfitriones a una partición.

6.2.2. Asociación de anfitriones a particiones

Cuando se crea un anfitrión en VirtShell este debe ser asociado a una sola partición, asignándole un tipo de los mencionados anteriormente, estableciendo la capacidad de disco y memoria RAM con las que cuenta, indicando también el sistema operativo y las IP con las que se conecta a la red. Una vez el anfitrión es creado en el sistema este queda asignado a la partición elegida. El siguiente ejemplo muestra como crear un anfitrión y asociarlo a una partición usando el API:

Listing 6.2: Petición HTTP para crear un anfitrión

```
curl -sv -X POST \
  -H 'accept: application/json' \
    -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"name": "host-01-pdn",
       "os": "Ubuntu_12.04_3.5.0-23.x86_64",
       "memory": "16GB",
       "capacity": "120GB",
```

```
        "enabled": "true",
        "type" : "GeneralPurpose",
        "local_ipv4": "15.54.88.19",
        "local_ipv6": "ff06:0:0:0:0:0:0:c3",
        "public_ipv4": "10.54.88.19",
        "public_ipv6": "yt06:0:0:0:0:0:0:c3",
        "partition": "development_co"}' \
    'http://localhost:8080/virtshell/api/v1/hosts'
```

En este ejemplo se muestra la creación de un anfitrión con nombre host-01-pdn que esta clasificado como de uso general y que queda asociado a la partición development_co creada en la sección anterior.

Al consultar la partición nuevamente por medio del API se puede observar como el anfitrión se encuentra asociado a la partición developtment_co. En la siguiente consulta al API se muestra el resultado de la asociación:

Listing 6.3: Petición HTTP para consultar una partición por su nombre

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://<host>:<port>/api/virtshell/v1/partitions/
        development_co'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "uuid": "efa1777c-cad7-11e5-9956-625662870761",
  "name": "development_co",
  "description": "Collection of servers oriented to
     development team in colombia.",
```

```
  "hosts": [ "host-01-pdn" ],
  "created": {"at":"1429207233",
              "by":"1a900cdc-cad8-11e5-9956-625662870761"
                 }
}
```

Adicionalmente, cuando un anfitrión es agregado a una partición, VirtShell instala automáticamente los agentes que realizaran tareas de aprovisionamiento y monitoreo. Los agentes se explicaran mas adelante.

6.2.3. División de particiones en ambientes

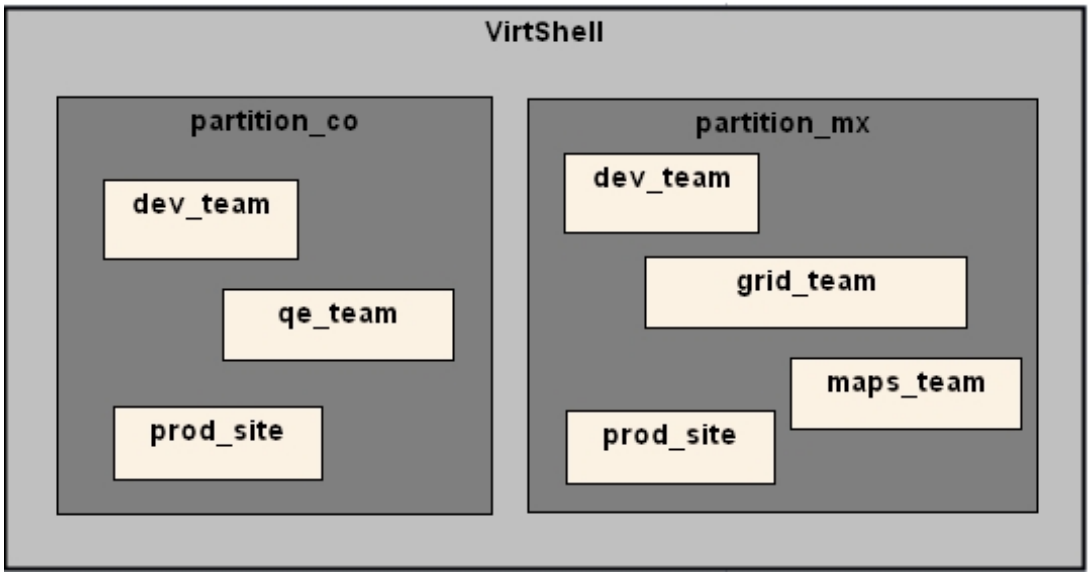
Las particiones se refieren a la forma de organizar lugares físicos. Los ambientes por el contrario, son lugares lógicos y aislados dentro de una partición. Cada partición puede estar compuesta por uno o varios ambientes, y un ambiente puede contener una o mas instancias. Cada ambiente pertenece a una sola partición. La figura 6.1 muestra un ejemplo de como dos particiones contiene varios ambientes destinados a equipos de trabajo diferentes.

Un ejemplo de como crear un ambiente asociado a una partición usando el API se muestra en el siguiente código:

Listing 6.4: Petición HTTP para crear un ambiente

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{
       "name": "bigdata_laboratory",
       "description": "Collection of servers oriented to
          big data.",
       "partition": "bogota_partition_co"
```

Figura 6.1: Ejemplo de ambientes en un partición




```
8       }' \
9     'http://localhost:8080/api/virtshell/v1/enviroments'
```

6.2.4. Creación de instancias en un ambiente

Las instancias son designadas a un solo ambiente de trabajo en el momento de ser creadas. En el instante de su creación, se debe especificar el aprovisionador encargado de instalar los paquetes y aplicaciones, los permisos para los demás usuarios del sistema, el tipo de anfitrión que necesita y la tecnología que se desea usar para virtualizar.

Cuando se recibe una solicitud de creación de una instancia, VirtShell selecciona un anfitrión, de los que se encuentran configurados en la partición a la cual pertenece el ambiente donde se quiere crear la instancia. En la solicitud de creación se debe especificar el tipo de anfitrión que se requiere para el correcto funcionamiento de las aplicaciones que se ejecutaran ahí. Si la partición no cuenta con el tipo de anfitrión solicitado, se rechazara la petición detallando el inconveniente. Si por el contrario la selección del anfitrión es exitosa, VirtShell procede a enviar la solicitud al agente de

aprovisionamiento del anfitrión escogido.

Debido a que una solicitud de aprovisionamiento puede involucrar mas de una instancia, el servidor de VirtShell responde la solicitud de aprovisionamiento con el estado de *in-progress* (en progreso) y a su vez con el identificador de una tarea. La tarea será ejecutada de manera asincrónica. El usuario por medio del API REST puede consultar el estado y avance de la misma (Las tareas se ampliaran en la siguiente sección).

Un ejemplo de la creación de una instancia usando el API se muestra en el siguiente código:

Listing 6.5: Petición HTTP para crear una instancia

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "name": "transactional_log",
        "enviroment": "development_co",
        "description": "Server transactional only for
           store logs",
        "provisioner": "all_backend",
        "host_type": "GeneralPurpose"
      }' \
   'http://localhost:8080/api/virtshell/v1/instances'
```

En esta solicitud HTTP se observa que la instancia a crear tiene asociado el aprovisionador de nombre all_backend, el cual como se vera en el capitulo de aprovisionamiento, es el que contiene la configuración de la imagen y características físicas de la maquina como memoria RAM, CPU y tamaño en disco. También se nota en la solicitud que la instancia debe ser creada en el ambiente de nombre development_co en una maquina física de uso general. Adicionalmente la instancia debe ser un contenedor que se ejecute sobre LXC.

Si la solicitud de las características es exitosa, el servidor responderá de la siguiente manera:

Listing 6.6: Ejemplo de respuesta HTTP a la solicitud de crear una instancia

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 { "create": "in progress", "task": "bbf6669b-1b83-4fe7-
    ae2a-2b31738c4165" }
```

Si por el contrario, la solicitud no es exitosa, el servidor responderá con el respectivo código de error HTTP y la razón por la cual no se pudo procesar exitosamente la petición. Un ejemplo de una petición no procesada es la siguiente:

Listing 6.7: Ejemplo de respuesta HTTP con error a la solicitud de crear una instancia

```
1 HTTP/1.1 500 OK
2 Content-Type: application/json
3 { "create": "error", "reason": "The enviroment doesn't
    have the host_type requested" }
```

Adicional a la creación de instancias, el API REST de VirtShell proporciona URIs para su gestión. Las operaciones sobre el ciclo de vida de una instancia son: iniciar, detener, reiniciar, cerrar y clonar. Así mismo, el API posibilita la fácil y rápida búsqueda de una o mas instancias.

6.3. Tareas

Una tarea es una actividad asincrónica, que se ejecuta en *background* (segundo plano), y se crea de manera automática cuando se solicita una acción al servidor. Una tarea puede tomar un largo periodo de tiempo para completar su trabajo.

Las tareas asíncronas resuelven los problemas relacionados con los tiempos de espera de las conexiones, los tiempos de espera de la solicitudes o los tiempos de vida

de las peticiones HTTP, antes de que estas sean cerradas por el servidor de aplicaciones web o por el navegador de manera automática, de acuerdo a sus parámetros de configuración.

Las operaciones de larga duración en VirtShell son las que involucran acciones en las instancias. Debido a que estas ultimas pueden estar físicamente lejanas del servidor, el tiempo de espera de una petición HTTP puede incrementarse lo suficiente para que esta sea finalizada por el servidor web. Asimismo, las operaciones de aprovisionamiento de las instancias, las cuales consisten principalmente en instalación y configuración de paquetes son tareas que involucran componentes realizados por terceros sobre los cuales no se tiene control sobre sus tiempos de respuesta.

Cuando VirtShell recibe una petición y esta es considerada de larga duración, en el cuerpo de la respuesta se envía la acción recibida con el estado de *in progress* acompañado del identificador único UUID [4] de la tarea. Por ejemplo, si se recibe una petición de instalación de un paquete en una o mas instancias, la respuesta HTTP seria de la siguiente forma:

Listing 6.8: Ejemplo de respuesta HTTP referenciando a una tarea

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 { "install-package": "in progress", "task": "908b4d78-
   8b7a-40d7-9ecf-5036eeb5526b" }
```

Para consultar por el estado de una tarea, se invoca el API de VirtShell, enviando en la url, el identificador UUID recibido de la primera petición. El servidor responderá con un documento en formato JSON detallando la tarea y su estado. El siguiente es un ejemplo de una consulta sobre una tarea y la respuesta del servidor:

Listing 6.9: Ejemplo de consulta de una tarea

```
1 curl -sv -H 'accept: application/json'
```

[4] UUID son las siglas en inglés del Identificador Universalmente Único.

```
    -H 'X-VirtShell-Authorization: UserId:Signature' \
    'http://<host>:<port>/api/virtshell/v1/tasks/
      908b4d78-8b7a-40d7-9ecf-5036eeb5526b'
```

Respuesta:

Listing 6.10: Ejemplo del detalle de una tarea

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```json
{
  "uuid": "908b4d78-8b7a-40d7-9ecf-5036eeb5526b",
  "description": "install-package in instance database_01
    ",
  "status" : "installing",
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
    -8994-feff819cdc9f"},
  "last_update": "1429207435",
  "log": "installing package couchdb"
}
```

6.4. Propiedades

Las propiedades sirven para obtener información de las instancias y anfitriones. Por medio de ellas se pueden conocer características de configuración de red, cantidad de memoria en uso, porcentaje de procesador usado en un determinado momento, numero de procesos en ejecución y una gran variedad de información útil del sistema.

Las propiedades pueden ser consultadas para una o mas instancias al mismo tiempo, definiendo el nombre de cada una de ellas. También pueden ser consultadas varias

características del sistema al mismo tiempo. El siguiente ejemplo muestra la forma de consultar varias propiedades de una instancia:

Listing 6.11: Ejemplo consultando varias propiedades a una instancia

```
curl -sv -X GET \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "properties": [{"name": "memory"}, {"name": "cpu"
    }],
        "hosts": [{"name": "WebServer001"}]}' \
   'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

Listing 6.12: Respuesta

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```json
{
  "name": "WebServer001",
  "memory": 8024,
  "cpu": 87.4
}
```

En el segundo ejemplo se muestra la forma de consultar varias propiedades a una o mas instancias:

Listing 6.13: Ejemplo consultando propiedades a varias instancias

```
curl -sv -X GET \
  -H 'accept: application/json' \
```

```
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "properties": [{"name": "memory"}, {"name": "cpu"
     }],
         {"name": "Database00", "range": "[1-3]"}]}' \
   'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

Listing 6.14: Respuesta

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```
```json
{
  properties: [
    {
     "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
     "name": "Database001",
     "memory": 4024,
     "cpu": 2
    },
    {
     "id": "591b3828-7aaf-4833-a94c-ad0df44d59a4",
     "name": "Database002",
     "memory": 4024,
     "cpu": 1
    },
    {
     "id": "f7c81039-5c88-423b-8b0d-c124483d586b",
     "name": "Database003",
     "memory": 4024,
     "cpu": 3
```
```

```
24          }
25      ]
26  }
```

CAPÍTULO 7

Aprovisionamiento

VirtShell cuenta con una colección de recursos que permite el aprovisionamiento de instancias,independientemente de la infraestructura de visualización que se use para estas. El conjunto de recursos de esta capa, esta diseñado para que sea una solución sencilla de usar, fiable y repetible, con una curva de aprendizaje muy baja para los administradores, desarrolladores y administradores de TI. Este capítulo busca describir todo lo que se requiere para llevar a cabo un correcto aprovisionamiento en las instancias creadas a través del sistema.

7.1. Almacenamiento y envio de archivos

El módulo de archivos proporciona una manera de almacenar archivos en VirtShell y de enviarlos a uno mas instancias. Para enviar archivos a las instancias, estos deben estar almacenados previamente en el sistema. En la petición HTTP de envió de archivos se debe especificar la ruta donde se encuentra el archivo, en la maquina en la que se esta invocando el API, asimismo, la carpeta donde desea guardarse y los permisos con que contara el archivo. Si la carpeta no existe, el sistema la creara automáticamente. A continuación se muestra un ejemplo de como enviar un archivo

a VirtShell.

Listing 7.1: Petición HTTP para subir un archivo al sistema

```
curl -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -H "Content-Type: multipart/form-data" \
  -F "file_data=@/path/to/file/seed_file.txt;filename=
     seed_file_ubuntu-14_04.txt" \
  -F "folder_name=ubuntu_seeds" \
  -F "permissions=rwxrwx---",
  'http://<host>:<port>/api/virtshell/v1/files'
```

La respuesta que retorna VirtShell, muestra la url en donde almaceno el archivo y el identificador UUID asignado en el sistema. Un ejemplo se muestra a continuación:

Listing 7.2: Respuesta HTTP del almacenamiento de un archivo.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "create": "success",
  "location": "http://<host>:<port>/api/virtshell/v1/
     files/ubuntu_seeds/seed_file_ubuntu-14_04.txt",
  "uuid": "51702a78-b23c-4625-bdda-3704cd0924b8"
}
```

Existen dos formas para el envió de archivos a las instancias, la primera consiste en enviar un archivo a un destino en una o mas instancias, como se muestra a continuación:

Listing 7.3: Ejemplo del envio de un archivo a una instancia.

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "uuid_file": "0d832c60-7066-4d37-bd72-
     ce6ac4f61bcc",
        "destination": "$MYSQL_HOME/my.cnf"
        "instances": [
            {"name": "database\_server\_01"}
        ] }' \
  'http://localhost:8080/virtshell/api/v1/instances/copy\
     _files/'
```

Y la segunda forma se trata de enviar el contenido una carpeta del sistema, en donde pueden existir uno o mas archivos a una carpeta destino, especificando las instancias a las cuales sera enviado. Un ejemplo se muestra a continuación:

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "folder_name": "mysql_configuration",
        "destination": "$MYSQL_HOME/my.cnf"
        "instances": [
            {"name": "database\_server\_01"},
            {"name": "database\_server\_02"},
            {"name": "database\_server\_03"}
        ] }' \
  'http://localhost:8080/virtshell/api/v1/instances/copy\
     _files/'
```

El envió de archivos a las instancias genera una tarea en el sistema. La respuesta se da la siguiente manera:

Listing 7.4: Ejemplo de respuesta HTTP para el envio de archivos

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "copy-files": "in progress", "task": "49d0cca2-e297-
   47c4-976f-845e1ce8f475" }
```

7.2. Imágenes

Como se describió en la arquitectura, las imágenes que se manejan en VirtShell son de dos tipos: ISO y *templates* para contenedores. A continuación se presentan las características de cada una de ellas.

7.2.1. Imágenes tipo ISO

Las imágenes ISO son usadas para la creación de maquinas virtuales que interactuan con un hipervisor. Estas son versiones modificadas que se encuentran en los repositorios oficiales de las distintas distribuciones de Linux o de cualquier otro sistema operativo. Para que una imagen pueda ser usada en VirtShell, esta debe cumplir con los siguientes requisitos:

- Que su proceso de instalación sea completamente desatendido (o automático), esto quiere decir que no requiera intervención humana.
- Debe tener instalado un servidor SSH con un usuario valido para VirtShell.

VirtShell cuenta con un servicio que permite crear versiones desatendidas de la distribución de Linux Ubuntu. Para crear una versión desatendida es necesario proveer un mecanismo para responder a las preguntas formuladas durante el proceso de instalación, sin tener que introducir manualmente las respuestas mientras la instalación está transcurriendo. Esto se hace posible creando un archivo de preconfiguración (*pressed file*) con las respuestas y solicitud de instalación de paquetes como el del servidor SSH. Este archivo debe ser enviado al servidor de VirtShell a través del API REST. Previamente en la sección de archivos en este mismo capítulo se mostró como almacenar un archivo en el sistema. Una vez el archivo de respuestas

se encuentra en VirtShell, se enviá la petición HTTP al API solicitando la creación de la nueva imagen desatendida como se muestra a continuación:

Listing 7.5: Petición HTTP para crear una imagen desatendida

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"name": "ubuntu_server_14.04.2_amd64",
       "type": "iso",
       "os": "ubuntu",
       "timezone": "America/Bogota",
       "permissions" : "rwxrwxr--",
       "preseed_url": "https://<host>:<port>/api/
          virtshell/v1/files/seeds/seed_ubuntu14-04.txt",
       "download_url": "http://releases.ubuntu.com/raring
          /ubuntu-14.04-server-amd64.iso"}' \
   'http://localhost:8080/api/virtshell/v1/image'
```

Como se observa en la anterior petición, para crear una imagen desatendida, ademas de especificar el archivo de respuestas, se debe definir la url (en el campo download_url) donde se encuentra la imagen ISO que el servicio de VirtShell usara para la creación. Un ejemplo de un archivo de respuestas se encuentra en el apéndice C.

La creación de una imagen desatendida es un proceso que puede tomar algunos minutos en completarse, como se mencionó en el capitulo de administración, VirtShell crea una tarea asíncrona retornando un identificador que permite consultar el estado de la creación de la imagen más adelante. La respuesta HTTP seria de la siguiente forma:

Listing 7.6: Ejemplo de respuesta HTTP para la creación de una imagen

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "create-image": "in progress", "task": "84de8acd-6095-
   40b1-a318-e7c6706213a3" }
```

7.2.2. Imágenes tipo contenedor

Este tipo de imágenes son usadas para la creación de contenedores que interactuan directamente con el sistema operativo. Se encuentran divididas en dos subtipos, las imágenes para contenedores creados con tecnología LXC, y las imágenes para contenedores creados con tecnología Docker.

La Creación de un contenedor LXC, en general, implica la creación de un sistema de ficheros raíz propio para el contenedor. LXC delega este trabajo a *templates* (plantillas), las cuales pueden ser encontradas en las distribuciones de Linux bajo el directorio /usr/share/lxc/templates, e incluyen *templates* para crear contenedores con los sistemas operativos Ubuntu, Debian, Fedora, Oracle, Centos y Gentoo entre otros [20]. La idea con estos *templates* es modificarlos de acuerdo a las necesidades del usuario y subirlos al sistema por medio del API REST, una vez el *template* se encuentre en el sistema, se debe crear como una imagen disponible para contenedores LXC como se muestra a continuación:

Listing 7.7: Petición HTTP para crear una imagen para contenedores LXC

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"name": "Oracle",
       "type": "lxc-container",
       "os": "oracle",
       "permissions" : "rwxrwxr--",
```

```
9         "download_url": "https://<host>:<port>/api/
              virtshell/v1/files/images/3514296#file-lxc-
              oracle}' \
10     'http://localhost:8080/api/virtshell/v1/image'
```

Como se observa en la petición HTTP, se debe especificar que la imagen es de tipo lxc-container y precisar la ubicación del template en el sistema. La creación de este tipo de imágenes no genera una tarea, dado que el template ya esta creado previamente por el usuario.

La Creación de contenedores usando Docker, requiere de imágenes que se encuentran en repositorios públicos administrados por el equipo de Docker [1]. En un repositorio de imágenes Docker, las imágenes son guardadas por medio de la combinación del nombre del sistema operativo y un *tag* (etiqueta) suministrado por el creador de la imagen, lo que permite contar con mas de una imagen por sistema operativo. La especificación de una imagen Docker en VirtShell, se hace de la siguiente manera:

Listing 7.8: Petición HTTP para crear una imagen para contenedores Docker

```
1 curl -sv -X PUT \
2   -H 'accept: application/json' \
3   -H "Content-Type: text/plain" \
4   -H 'X-VirtShell-Authorization: UserId:Signature' \
5   -d '{"name": "centos:centos6",
6        "type": "docker-container",
7        "os": "centos",
8        "permissions" : "rwxrwxr--"}' \
9    'http://localhost:8080/api/virtshell/v1/image'
```

En la petición HTTP, con el fin de mantener el estándar con los repositorios Docker, el nombre de la imagen en VirtShell debe ser el mismo que se maneja en

[1] Repositorio oficial de imágenes Docker: https://github.com/docker-library/official-images

los repositorios públicos, esto es, la combinación del nombre del sistema operativo y el tab dado por el creador, separados por el carácter dos puntos (:). La creación de una imagen Docker en VirtShell genera una tarea, la cual consiste en verificar la existencia de la imagen en el repositorio público. Si la imagen es válida, su referencia será agregada al sistema para que el agente de aprovisionamiento sea el que la descargue del repositorio público, en caso contrario, se notificará la razón del error en la tarea. Un ejemplo del mensaje que retorna la creación de una imagen Docker se muestra a continuación:

Listing 7.9: Ejemplo de respuesta HTTP para la creación de una imagen

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 { "create-image": "checking", "task": "44ab19d5-7967-4bb9
    -b39e-f61f983af72b" }
```

7.3. Aprovisionadores

Este módulo permite definir la configuración física necesaria para crear una nueva instancia y como aprovisionarla. La definición de un aprovisionador de manera general, consiste en especificar las características de memoria RAM, el numero de CPUs y el tamaño en disco que se espera tenga el nuevo recurso virtual; seleccionar una imagen del repositorio; especificar el numero de instancias del mismo tipo que se van a crear; determinar la ubicación de los scripts encargados de instalar y configurar la instancia y finalmente, de especificar las dependencias de otras instancias. La estructura de un aprovisionador se muestra a continuación:

Listing 7.10: Estructura de un aprovisionador

```
1 {
2   "name": "backend-services",
3   "description": "Installs/Configures a backend server",
4   "launch", 1,
```

```
5   "memory": 16,
6   "cpus": 4,
7   "hdsize": 40,
8   "image": "ubuntu_server_14",
9   "builder": "https://github.com/janutechnology/
        backend_server.git",
10  "executor": "sh run.sh",
11  "tag": "backend",
12  "permissions": "xwrxwrxwr",
13  "depends": [
14        {"provisioner_name": "mysql_database"},
15        {"provisioner_name": "rabbitmq_server"},
16        {"provisioner_name": "analytic_server"}
17  ]
18 }
```

Para configurar exitosamente un aprovisionador, se debe tener en cuenta que la unidad de medida manejada para la memoria y el tamaño en disco es el giga-byte (GB). También hay que tener en cuenta, que el campo *launch* (lanzar), permite especificar el numero de instancias del mismo tipo que deben ser creadas y aprovisionadas, todas conservando las mismas características físicas y de aprovisionamiento. Si este campo no es definido por defecto se creara una sola instancia. El campo de *tag* permite clasificar los aprovisionadores para facilitar búsquedas por medio de agrupaciones. Los campos para definir los scripts de aprovisionamiento y las dependencias serán explicados en los siguientes párrafos.

7.3.1. Scripts de aprovisionamiento

El elemento fundamental para el aprovisionamiento es el *builder* (constructor), este referencia a un repositorio git que contiene los archivos, instruciones y scripts necesarios que serán aplicados en tiempo de construcción de la instancia. Si un aprovisionador no contiene un *builder*, VirtShell simplemente instalara en la instancia la imagen seleccionada, configurándola para que permita el acceso por medio del pro-

tocolo SSH.

El *builder* no tiene predeterminado una estructura de directorios ni define los archivos que debe contener, su organización y contenido queda al libre criterio del usuario, el cual debe garantizar que el *builder* cuente con todos los recursos para realizar un aprovisionamiento exitoso. El único requisito esencial, es la existencia de un script que actuá como punto de partida y el cual es definido en el campo executor del aprovisionador. En este campo se define el nombre del script principal y la forma en que debe ser invocado. Este script contiene el orden de las instrucciones de aprovisionamiento, puede ser un script con un solo conjunto de instrucciones que hace todo el trabajo o por el contrario puede ser un script tipo director que contiene el orden de llamado a otros scripts encargados de tareas especificas como instalación y configuración, ambos escenarios se muestran a continuación:

Listing 7.11: Escenarios de estructura de un constructor

```
+ basic_web_server     |   + rabbitmq_server
  + files              |     + etc
    - example.com      |       + rabbitmq
    - index.html       |         - rabbitmq.config
  - install.sh         |       + default
                       |         - rabbitmq-server
                       |    - install.sh
                       |    - configure.sh
                       |    - main.sh
```

En el ejemplo de la izquierda, el script en shell install.sh actuá como punto entrada del aprovisionamiento, y contiene todas las instrucciones de instalación y configuración de un sitio web sencillo. La carpeta de nombre files en el directorio contiene dos archivos creados previamente, que facilitan la configuración de un sitio web estático. El listado 7.12 muestra el contenido del script install.sh y la forma en que aprovisiona el sitio web.

Por otro lado, en el ejemplo de la derecha, se ha separado la instalación y la configuración en scripts diferentes, el script main.sh es el punto de partida, el cual se encarga de llamar los otros scripts. De igual manera existe una carpeta con el nombre etc, que contiene archivos para completar el aprovisionamiento.

Listing 7.12: Ejemplo de un script shell

```
1  #!/bin/bash
2
3  echo "Installing nginx..."
4  sudo apt-get install -y nginx
5
6  echo "Creating root directory..."
7  sudo mkdir -p /var/www/example.com/public_html
8
9  echo "Setting up permissions..."
10 sudo chown -R www-data:www-data /var/www/example.com/
     public_html
11 sudo chmod 755 /var/www
12
13 echo "Creating the page..."
14 sudo cp files/index.html /var/www/example.com/public_html
15
16 echo "Creating the new virtual host file..."
17 sudo cp files/example.com /etc/nginx/sites-available/
     example.com
18 sudo ln -s /etc/nginx/sites-available/example.com /etc/
     nginx/sites-enabled/example.com
19 sudo rm /etc/nginx/sites-enabled/default
20
21 echo "Restarting nginx..."
22 sudo service nginx restart
```

Los scripts de aprovisionamiento pueden estar escritos en cualquier lenguaje de programación. El uso de shell para escribir los scripts no es mandatorio. El usuario puede usar el lenguaje de su preferencia siempre y cuando la imagen base del sistema operativo cuente con las librerías y ejecutables necesarios para que VirtShell invoque el script inicial con la forma definida en el campo executor y este no presente errores en su ejecución.

Shell fue escogido como el lenguaje base para los scripts de aprovisionamiento, debido a que soporta todas las estructuras propias de los lenguajes modernos. Además permite la utilización de todas las primitivas del sistema operativo de control de procesos, interrupciones y utilidades para diseñar programas de comandos por el usuario. VirtShell cuenta con un conjunto propio de comandos que simplifican la escritura de los scripts de aprovisionamiento abstrayendo por ejemplo el sistema operativo en el cual se ejecutan o uniendo varios comandos de shell en uno solo. los comandos con los que cuenta VirtShell son:

vs_package Permite instalar o remover paquetes al sistema operativo.

vs_repository Permite agregar repositorios al sistema operativo.

vs_network Permite configurar dispositivos de red.

vs_directory Permite crear directorios en el sistema de archivos y configurarle sus permisos de usuario y grupo.

vs_copy_file Permite copiar archivos locales del repositorio o archivos remotos a la nueva instancia, al igual que vs_directory, también permite configurarle permisos de usuario y grupo.

vs_service Permite iniciar, detener o reiniciar servicios instalados en una instancia.

vs_property Permite consultar propiedades de instancias remotas, como por ejemplo: la ip, memoria, nombre de la maquina.

Los comandos de VirtShell favorecen la portabilidad de los scripts, permitiendo que estos puedan ser ejecutados en diferentes versiones de sistema operativo sin

modificar sus instrucciones. A continuación se muestra la portabilidad y también la forma en que simplifica el script de aprovisionamiento mostrado en el listado 7.12 usando los comandos de VirtShell:

Listing 7.13: Ejemplo de un script shell usando los comandos de VirtShell

```
1  #!/bin/bash
2
3  echo "Installing nginx..."
4  vs_package install nginx
5
6  echo "Creating root directory..."
7  vs_directory -d /var/www/example.com/public_html -g www-
       data:www-data -p 755
8
9  echo "Creating the page..."
10 vs_copy_file files/index.html /var/www/example.com/
       public_html
11
12 echo "Creating the new virtual host file..."
13 vs_copy_file files/example.com /etc/nginx/sites-available
       /example.com
14 sudo ln -s /etc/nginx/sites-available/example.com /etc/
       nginx/sites-enabled/example.com
15 sudo rm /etc/nginx/sites-enabled/default
16
17 echo "Restarting nginx..."
18 vs_service nginx restart
```

Los comandos pueden ser modificados o extendidos de acuerdo a las necesidades del usuario o de aprovisionamiento.

7.3.2. Dependencias

En el momento de crear un aprovisionador, las dependencias de este, son aquellas aplicaciones o servicios externos hospedados en otras instancias de las cuales se depende, para el correcto funcionamiento de los servicios o aplicaciones que se ejecutaran en la instancia que define el nuevo aprovisionador.

Las dependencias no son mas que un listado de aprovisionadores que deben existir antes de la ejecución del aprovisionador padre. Si al crear una instancia su aprovisionador tiene configurado un listado de dependencias ”hijas”, VirtShell verifica primero en el ambiente al cual pertenece, si estas dependencias ya se encuentran creadas con anterioridad, sino es asi, planea primero la ejecución de las dependencias hijas y cuando estas ya se encuentren listas, enviá el aprovisionador padre de la instancia deseada.

7.4. Instalación y Actualización de paquetes

Este módulo proporciona una manera de instalar o actualizar paquetes a una o mas instancias. En una petición HTTP se debe especificar el o los nombres de los paquetes asi como el o los nombres de las instancias en las cuales se va instalar o actualizar los paquetes definidos. Una peticion de este estilo genera una tarea en el sistema, retornando el identificador de la misma para ser consultado posteriormente. A continuación se muestra un ejemplo de como instalar paquetes en varias instancias.

Listing 7.14: Petición HTTP para instalar paquetes en las instancias

```
1 curl -sv -X POST \
2   -H 'accept: application/json' \
3   -H "Content-Type: text/plain" \
4   -H 'X-VirtShell-Authorization: UserId:Signature' \
5   -d '{ "packages": [{"name": "git"}, {"name": "nginx"}],
6         "instances": [{"name": "WebServer_", "range": "[1
            -9]"}]}' \
```

```
7    'http://localhost:8080/api/virtshell/v1/
       install_packages'
```

En esta solicitud se desea que a las instancias que contengan en su nombre el prefijo WebServer_, y cuyo sufijo se encuentre en el rango entre 1 y 9, se les instale los paquetes git y nginx. Esta solicitud generara una tarea para consultar su estado como se ha indicado anteriormente.

CAPÍTULO 8

Agentes

Los agentes son servicios que se ejecutan localmente en cada anfitrión. VirtShell instala y configura los agentes en cada uno de los anfitriones de manera automática, liberando de esta tarea al administrador del sistema. Existen tres tipos de agentes: de aprovisionamiento, monitoreo y administración.

Agente de Aprovisionamiento El agente de aprovisionamiento se encarga de instalar y configurar: paquetes, librerías y aplicaciones en las instancias.

Agente de Monitoreo El agente de monitoreo es completamente autónomo y sus funciones consisten en supervisar al agente de aprovisionamiento y reportar el estado de salud de los anfitriones e instancias que se encuentran creados en su interior.

Agente de Administración El agente de administración se encarga de gestionar las instancias, permitiendo detener, iniciar, clonar, eliminar y obtener información especifica de cada instancia que se ejecuta en el anfitrión.

CAPÍTULO 9

API

9.1. Definición de API

API significa “Application Programming Interface”, y como término, especifica cómo debe interactuar el software.

En términos generales, cuando nos referimos a las API de hoy, nos referimos más concretamente a las API web, que son manejadas a través del protocolo de transferencia de hipertexto (HTTP). Para este caso específico, entonces, una API especifica cómo un consumidor puede consumir el servicio que el API expone: cuales URI están disponibles, qué métodos HTTP puede utilizarse con cada URI, que parámetros de consulta se acepta, lo que los datos que pueden ser enviados en el cuerpo de la petición, y lo que el consumidor puede esperar como respuesta.

9.1.1. VirtShell API REST

En el VirtShell API REST un usuario enviá una solicitud al servidor para realizar una acción determinada (como la creación, recuperación, actualización o eliminación de un recurso virtual), y el servidor realiza la acción y enviá una respuesta, a menudo

en la forma de una representación del recurso especificado.

En el VirtShell API, el usuario especifica una acción con un verbo HTTP como POST, GET, PUT o DELETE. Especificando un recurso por un URI único global de la siguiente forma:

https://[host]:[port]/virtshell/api/v1/resourcePath?parameters

Debido a que todos los recursos del API tienen una única URI HTTP accesible, REST permite el almacenamiento en cache de datos y esta optimizado para trabajar con una infraestructura distribuida de la web.

En esta sección se detalla los recursos y operaciones que puede realizar un usuario del API para realizar aprovisionamientos automáticos desde cualquier plataforma de desarrollo. El VirtShell API provee acceso a los objetos en el VirtShell Server, esto incluye los hosts, imágenes, archivos, templates, aprovisionadores, instancias, grupos y usuarios. Por medio del API podrá crear ambientes, maquinas virtuales y contenedores personalizados, realizar configuraciones y administrar los recursos físicos y virtuales de manera programática.

9.2. Formato de entrada y salida

JSON (JavaScript Object Notation) es un formato de datos común, independiente del lenguaje que proporciona una representación de texto simple de estructuras de datos arbitrarias. Para obtener mas información, ver json.org.

El VirtShell API solo soporta el formato json para intercambio de información. Cualquier solicitud que no se encuentre en formato json resultara en un error con código 406 (Content Not Acceptable Error).

9.3. Codigos de error

Aqui se presenta una lista de codigos de error que pueden resultar de una petición al API en cualquier recurso.

- **400 Bad Request** La solicitud no pudo ser procesada con éxito porque el URI no era válido. El cuerpo de la respuesta contendrá una razón del fracaso de la petición. Esta respuesta indica error permanente.
- **403 Forbidden** La solicitud no pudo ser procesada con éxito porque la identidad del usuario no tiene acceso suficiente para procesar la solicitud. Esta respuesta indica error permanente.
- **406 Content Not Acceptable** Un recurso genera este error de acuerdo al tipo de cabeceras enviadas en la petición. Esta respuesta indica un error permanete e indica un formato de salida no soportado. La respuesta de este tipo de error no contiene un contenido debido a la inhabilidad del servidor para generar una respuesta en el formato solicitado.
- **404 Not Found** La solicitud no pudo ser procesada con éxito porque la solicitud no era válida. Lo más probable es que no se encontró la url. Esta respuesta indica error permanente.
- **500 Server Error** La solicitud no pudo ser procesada debido a que el servidor encontró una condición inesperada que le impidió cumplir con la petición.
- **501 Not Implemented** La solicitud no se pudo completar porque el servidor o bien no reconoce el método de petición o el recurso solicitado no existe.

Los errores que no sean de codigo 406 (Content Not Acceptable) contienen una respuesta en formato json, que contiene un breve mensaje explicado el error con más detalle. Por ejemplo, una consulta POST /virtshell/api/v1/hosts, con un cuerpo vacio, daría lugar a la siguiente respuesta:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{"error": "Missing input for create instance"}
```

9.4. API Resources

9.4.1. Groups

Representan los grupos registrados en VirtShell. Los metodos soportados son:

Cuadro 9.1: Métodos HTTP para groups

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/users/:name	Gets one group by ID.
list	GET	/hosts	Retrieves the list of groups.
create	POST	/users/	creates a new group.
delete	DELETE	/users/:name	Deletes an existing group.

Representación del recurso de un grupo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "web_development_team",
  "users": [ ... list of members of the group ...],
  "created":[ {"at":"timestamp"}, {"by":user_id}]
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "web_development_team",
  "users": [
      {"username": "user1", "id": "a146cae4-8c90-11e5-
         8994-feff819cdc9f"},
      {"username": "user2", "id": "a146d00c-8c90-11e5-
         8994-feff819cdc9f"}
  ]
  "created":[{"at":"1447696674"}, {"by":"a379e8e6-8c8b-
     11e5-8994-feff819cdc9f"}]
}
```

9.4.1.1. Ejemplos de peticiones HTTP

9.4.1.1.1. Crear un nuevo grupo - POST /api/virtshell/v1/grupos

```
curl -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -H "Content-Type: multipart/form-data" \
  -d '{
        "name": "database_team"
      }' \
  'http://<host>:<port>/api/virtshell/v1/groups'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{ "create": "success" }
```

9.4.1.1.2. Obtener un grupo - GET /api/virtshell/v1/groups/:name

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://<host>:<port>/api/virtshell/v1/groups/
        web_development_team'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "web_development_team",
  "users": [
      {"username": "user1", "id": "a146cae4-8c90-11e5-
         8994-feff819cdc9f"},
      {"username": "user2", "id": "a146d00c-8c90-11e5-
         8994-feff819cdc9f"}
  ]
  "created":[{"at":"1447696674"}, {"by":"a379e8e6-8c8b-
     11e5-8994-feff819cdc9f"}]
}
```

9.4.1.1.3. Obtener todos los grupos - GET /api/virtshell/v1/groups

```
curl -sv -H 'accept: application/json'
```

```
    -H 'X-VirtShell-Authorization: UserId:Signature' \
    'http://localhost:8080/api/virtshell/v1/groups'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "groups": [
    {
      "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
      "name": "web_development_team",
      "users": [
          {"username": "user1", "id": "a146cae4-8c90-11e5
             -8994-feff819cdc9f"},
          {"username": "user2", "id": "a146d00c-8c90-11e5
             -8994-feff819cdc9f"}
      ],
      "created":[{"at":"1447696833"}, {"by":"d2372efa-
         8c8b-11e5-8994-feff819cdc9f"}]
    },
    {
      "uuid": "a379f19c-8c8b-11e5-8994-feff819cdc9f",
      "name": "math_team",
      "users": [
          {"username": "user3", "id": "a146cae4-8c90-11e5
             -8994-feff819cdc9f"}
      ],
      "created":[{"at":"1421431233"}, {"by":"18489280-
         8c91-11e5-8994-feff819cdc9f"}]
    },
```

```
22      {
23        "uuid": "a379f3d6-8c8b-11e5-8994-feff819cdc9f",
24        "name": "chemical_team",
25        "users": [
26            {"username": "user4", "id": "F8489280-8c91-11e5
                -8994-feff819cdc9f"},
27            {"username": "user5", "id": "18489780-8c91-11e5
                -8994-feff819cdc9f"}
28        ],
29        "created":[{"at":"1424109633"}, {"by":"d2373576-
            8c8b-11e5-8994-feff819cdc9f"}]
30      },
31  }
```

9.4.1.1.4. Eliminar un grupo - DELETE /api/virtshell/v1/groups/:name

Para eliminar un grupo se debe tener en cuenta que no debe tener usuarios asociados a el.

```
1  curl -sv -X DELETE \
2     -H 'accept: application/json' \
3     -H 'X-VirtShell-Authorization: UserId:Signature' \
4     'http://localhost:8080/api/virtshell/v1/groups/
        web_development_team'
```

Respuesta:

```
1  HTTP/1.1 200 OK
2  Content-Type: application/json
3  ```
```

```
```json
{ "delete": "success" }
```

9.4.2. Users

Representan los usuarios registrados en VirtShell. Los metodos soportados son:

Cuadro 9.2: Métodos HTTP para users

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/users/:name	Gets one user by ID.
create	POST	/users/	creates a new user.
list	GET	/users	Retrieves the list of users.
delete	DELETE	/users/:name	Deletes an existing user.
update	PUT	/users/:name	Updates an existing user.

Representación del recurso de un usuario:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "username": "virtshell",
  "type": "system/regular",
  "login": "user@mail.com",
  "groups": [ ... list of users ...],
  "created": {"at": timestamp, "by": user_uuid},
  "modified": {"at": timestamp, "by": user_uuid}
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
```
```

```
  "username": "virtshell",
  "type": "system/regular",
  "login": "user@mail.com",
  "groups": [ {"name": "web_development_team"},
              {"name": "production"}
  ],
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
    -8994-feff819cdc9f"},
  "modified": {"at":"1529207233", "by":"92d31132-8c9c-
    11e5-8994-feff819cdc9f"}
}
```

9.4.2.1. Ejemplos de peticiones HTTP

9.4.2.1.1. Crear un nuevo usuario - POST /api/virtshell/v1/users

```
curl -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -H "Content-Type: multipart/form-data" \
  -d {
       "username": "virtshell",
       "type": "system/regular",
       "login": "user@mail.com",
       "groups": [ {"name": "web_development_team"},
                   {"name": "production"} ]
     } \
  'http://<host>:<port>/api/virtshell/v1/users'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "create": "success" }
```

9.4.2.1.2. Obtener un usuario - GET /api/virtshell/v1/users/:name

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://<host>:<port>/api/virtshell/v1/users/
        virtshell'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "username": "virtshell",
  "type": "system/regular",
  "login": "user@mail.com",
  "groups": [ {"name": "web_development_team"},
              {"name": "production"}
  ],
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
     -8994-feff819cdc9f"},
  "modified": {"at":"1529207233", "by":"92d31132-8c9c-
     11e5-8994-feff819cdc9f"}
}
```

9.4.2.1.3. Actualizar un usuario - PUT /api/virtshell/v1/users/:name

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -H "Content-Type: multipart/form-data" \
  -d '{"type": "system",
       "groups": [{"uuid": "a146cae4-8c90-11e5-8994-
          feff819cdc9f"},
                  {"uuid": "a146d00c-8c90-11e5-8994-
                     feff819cdc9f"}]}' \
   'http://localhost:8080/api/virtshell/v1/file/virtshell
      '
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "update": "success" }
```

9.4.2.1.4. Eliminar un usuario - DELETE /api/virtshell/v1/users/:name

```
curl -sv -X DELETE \
   -H 'accept: application/json' \
   -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/api/virtshell/v1/fles/virtshell
      '
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```json
{ "delete": "success" }
```

9.4.3. Partitions

Las particones permiten organizar las máquinas que albergaran recursos virtuales en partes aisladas de las demás.Los métodos soportados son:

Cuadro 9.3: Métodos HTTP para partitions

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/partitions/:name	Gets one partition by name.
list	GET	/partitions	Retrieves the list of partitions
create	POST	/partitions/	Inserts a new partition config
delete	DELETE	/partitions/:name	Deletes an existing partition.
update	PUT	/partitions/:name/host/:hostname	Add a host to partition.

Representación del recurso de una partición:

```
{
  "uuid": string,
  "name": string,
  "description": string,
  "hosts": [ ... list of hosts associated with the
     Partitions ...],
  "created": {"at": number, "by": string},
  "modified": {"at": number, "by": string}
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "development_co",
  "description": "Collection of servers oriented to
     development team in Colombia.",
  "hosts": [ ... list of hosts associated with the
     partition ...],
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
     -8994-feff819cdc9f"},
  "modified": {"at":"1529207233", "by":"92d31132-8c9c-
     11e5-8994-feff819cdc9f"}
}
```

9.4.3.1. Ejemplos de peticiones HTTP

9.4.3.1.1. Crear una nueva partición - POST /api/virtshell/v1/partitions

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{
       "name": "development_co",
       "description": "Collection of servers oriented to
          development team in colombia."
      }' \
   'http://localhost:8080/api/virtshell/v1/partitions'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
{ "create": "success" }
```

9.4.3.1.2. Obtener una partición- GET /api/virtshell/v1/partitions/:name

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://<host>:<port>/api/virtshell/v1/partitions/
        development_co'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "uuid": "efa1777c-cad7-11e5-9956-625662870761",
  "name": "backend_development_04",
  "description": "Servers for backend of the company",
  "hosts": [ ... list of hosts associated with the
     section ...],
  "created": {"at":"1429207233", "by":"1a900cdc-cad8-11e5
     -9956-625662870761"},
  "modified": {"at":"1529207233", "by":"2163b554-cad8-
     11e5-9956-625662870761"}
}
```

9.4.3.1.3. Obtener todas las particiones - GET /api/virtshell/v1/partitions

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://localhost:8080/api/virtshell/v1/partitions'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "partitions": [
    {
      "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
      "name": "development_co",
      "description": "Collection of servers oriented to
        development team in colombia.",
      "hosts": [ ... list of hosts associated with the
        section ...],
      "created": {"at":"1429207233", "by":"92d30f0c-8c9c-
        11e5-8994-feff819cdc9f"},
      "modified": {"at":"1529207233", "by":"92d31132-8c9c
        -11e5-8994-feff819cdc9f"}
    },
    {
      "uuid": "efa1777c-cad7-11e5-9956-625662870761",
      "name": "production_us_miami",
      "description": "Collection of servers oriented to
        production in us.",
      "hosts": [ ... list of hosts associated with the
        section ...],
      "created": {"at":"1429207233", "by":"1a900cdc-cad8-
        11e5-9956-625662870761"},
      "modified": {"at":"1529207233", "by":"2163b554-cad8
        -11e5-9956-625662870761"}
    }
  ]
}
```

9.4.3.1.4. Eliminar una partición - DELETE /api/virtshell/v1/partitions/:name

```
curl -sv -X DELETE \
   -H 'accept: application/json' \
   -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://<host>:<port>/api/virtshell/v1/partitions/
      backend_development_04'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```json
{ "delete": "success" }
```

9.4.3.1.5. Agregar un host a una partición - PUT /api/virtshell/v1/partitions/:name/

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  'http://localhost:8080/virtshell/api/v1/partitions/:
     name/host/:hostname'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "add_host": "success" }
```

9.4.4. Hosts

Representan subredes de trabajo más pequeñas asociadas a una partición. Los métodos soportados son:

Cuadro 9.4: Métodos HTTP para enviroments

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/enviroments/:name	Gets one enviroment by name.
list	GET	/enviroments	Retrieves the list of enviroments.
create	POST	/enviroments/	Inserts a new enviroment configuration.
delete	DELETE	/enviroments/:name	Deletes an existing enviroment.

Representación del recurso de un ambiente:

```
{
  "uuid": string,
  "name": string,
  "description": string,
  "users": [ user_resource],
  "partition": string,
  "created": {"at": number, "by": string},
  "modified": {"at": number, "by": string}
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "bigdata_test_01",
  "description": "Collection of servers oriented to big
    data.",
  "users": [ ... list of users allowed to use the
    enviroment ...],
```

```
  "partition": "partition associated with the enviroment",
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5-8994-feff819cdc9f"},
  "modified": {"at":"1529207233", "by":"92d31132-8c9c-11e5-8994-feff819cdc9f"}
}
```

9.4.4.1. Ejemplos de peticiones HTTP

9.4.4.1.1. Crear un nuevo ambiente - POST /api/virtshell/v1/enviroments

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{
        "name": "bigdata_test_01",
        "description": "Collection of servers oriented to big data.",
        "users": [ ... list of users allowed to use the enviroment ...],
        "partition": "partition associated with the enviroment"
      }' \
   'http://localhost:8080/api/virtshell/v1/enviroments'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "create": "success" }
```

9.4.4.1.2. Obtener un ambiente- GET /api/virtshell/v1/enviroments/:name

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://<host>:<port>/api/virtshell/v1/enviroments/
        backend_development'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "uuid": "efa1777c-cad7-11e5-9956-625662870761",
  "name": "backend_development",
  "description": "All backend of the company",
  "users": [ ... list of users allowed to use the
     enviroment ...],
  "partition": "partition associated with the enviroment"
     ,
  "created": {"at":"1429207233", "by":"1a900cdc-cad8-11e5
     -9956-625662870761"},
  "modified": {"at":"1529207233", "by":"2163b554-cad8-
     11e5-9956-625662870761"}
}
```

9.4.4.1.3. Obtener todos los ambientes - GET /api/virtshell/v1/enviroments

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://localhost:8080/api/virtshell/v1/enviroments'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "enviroments": [
    {
      "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
      "name": "bigdata_test_01",
      "description": "Collection of servers oriented to
        big data.",
      "users": [ ... list of users allowed to use the
        enviroment ...],
      "partition": "partition associated with the
        enviroment",
      "created": {"at":"1429207233", "by":"92d30f0c-8c9c-
        11e5-8994-feff819cdc9f"},
      "modified": {"at":"1529207233", "by":"92d31132-8c9c
        -11e5-8994-feff819cdc9f"}
    },
    {
      "uuid": "efa1777c-cad7-11e5-9956-625662870761",
      "name": "backend_development",
      "description": "All backend of the company",
      "users": [ ... list of users allowed to use the
        enviroment ...],
      "partition": "partition associated with the
        enviroment",
      "created": {"at":"1429207233", "by":"1a900cdc-cad8-
        11e5-9956-625662870761"},
      "modified": {"at":"1529207233", "by":"2163b554-cad8
        -11e5-9956-625662870761"}
    }
```

```
23   ]
24 }
```

9.4.4.1.4. Eliminar un ambiente - DELETE /api/virtshell/v1/enviroments/:name

```
curl -sv -X DELETE \
   -H 'accept: application/json' \
   -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://<host>:<port>/api/virtshell/v1/enviroments/
      backend_development'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```
```json
{ "delete": "success" }
```

9.4.5. Hosts

Representan las maquinas físicas; un host es un anfitrión de maquinas virtuales o contenedores. Los métodos soportados son:

Cuadro 9.5: Métodos HTTP para hosts

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/hosts/:name	Gets one host by name.
list	GET	/hosts	Retrieves the list of hosts.
create	POST	/hosts/	Inserts a new host configuration.
delete	DELETE	/hosts/:name	Deletes an existing host.
update	PUT	/hosts/:name	Updates an existing host.
```

Representación del recurso de un host:

```
{
  "uuid": string,
  "name": string,
  "os": string,
  "memory": string,
  "capacity": string,
  "enabled": string,
  "type":string,
  "local_ipv4": string,
  "local_ipv6": string,
  "public_ipv4": string,
  "public_ipv6": string,
  "instances": [ instance_resource ],
  "partition": string,
  "created":["at": number, "by": number]
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "host-01-pdn",
  "os": "Ubuntu_12.04_3.5.0-23.x86_64",
  "memory": "16GB",
  "capacity": "120GB",
  "enabled": "true|false",
  "type":"StorageOptimized|GeneralPurpose|HighPerformance
    ",
  "local_ipv4": "15.54.88.19",
  "local_ipv6": "ff06:0:0:0:0:0:0:c3",
  "public_ipv4": "10.54.88.19",
  "public_ipv6": "yt06:0:0:0:0:0:0:c3",
```

```
  "instances": [
    ... instances resource is here
  ],
  "partition": "development_co",
  "created":["at":"timestamp", "by":1234]
}
```

9.4.5.1. Ejemplos de peticiones HTTP

9.4.5.1.1. Crear un nuevo host - POST /api/virtshell/v1/hosts

```
curl -sv -X POST \
  -H 'accept: application/json' \
    -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"name": "host-01-pdn",
       "os": "Ubuntu_12.04_3.5.0-23.x86_64",
       "memory": "16GB",
       "capacity": "120GB",
       "enabled": "true",
       "type" : "GeneralPurpose",
       "local_ipv4": "15.54.88.19",
       "local_ipv6": "ff06:0:0:0:0:0:0:c3",
       "public_ipv4": "10.54.88.19",
       "public_ipv6": "yt06:0:0:0:0:0:0:c3",
       "partition": "development_co"}' \
   'http://localhost:8080/virtshell/api/v1/hosts'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "create": "success" }
```

9.4.5.1.2. Obtener un host- GET /api/virtshell/v1/hosts/:name

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://localhost:8080/api/virtshell/v1/hosts/host-
       01-pdn'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "host-01-pdn",
  "os": "Ubuntu_12.04_3.5.0-23.x86_64",
  "memory": "16GB",
  "capacity": "120GB",
  "enabled": "true",
  "type" : "StorageOptimized",
  "local_ipv4": "15.54.88.19",
  "local_ipv6": "ff06:0:0:0:0:0:0:c3",
  "public_ipv4": "10.54.88.19",
  "public_ipv6": "yt06:0:0:0:0:0:0:c3",
  "instances": [
    {
      "name": "name1",
      "id": "72C05559-0590-4DA6-BE56-28AB36CB669C"
    },
    {
      "name": "name2",
      "id": "17173587-C4E9-4369-9C43-FCBF5E075973"
    }
  ],
  "partition": "development_co",
```

```
  "created":["at":"20130625105211", "by":10]
}
```

9.4.5.1.3. Obtener todos los host - GET /api/virtshell/v1/hosts

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://localhost:8080/api/virtshell/v1/hosts'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "hosts": [
    {
      "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
      "name": "host-01-pdn",
      "os": "Ubuntu_12.04_3.5.0-23.x86_64",
      "memory": "16GB",
      "capacity": "120GB",
      "enabled": "true",
      "type" : "StorageOptimized",
      "local_ipv4": "15.54.88.19",
      "local_ipv6": "ff06:0:0:0:0:0:0:c3",
      "public_ipv4": "10.54.88.19",
      "public_ipv6": "yt06:0:0:0:0:0:0:c3",
      "instances": [
        {
          "name": "name1",
          "id": "72C05559-0590-4DA6-BE56-28AB36CB669C"
        },
        {
```

```
23            "name": "name2",
24            "id": "17173587-C4E9-4369-9C43-FCBF5E075973"
25          }
26        ],
27        "partition": "development_co",
28        "created":["at":"20130625105211", "by":10]
29      },
30      {
31        "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
32        "name": "host-01-pdn",
33        "os": "Ubuntu_12.04_3.5.0-23.x86_64",
34        "memory": "16GB",
35        "capacity": "120GB",
36        "enabled": "true",
37        "type" : "GeneralPurpose",
38        "local_ipv4": "15.54.88.19",
39        "local_ipv6": "ff06:0:0:0:0:0:0:c3",
40        "public_ipv4": "10.54.88.19",
41        "public_ipv6": "yt06:0:0:0:0:0:0:c3",
42        "instances": [
43          {
44            "name": "name3",
45            "id": "DE11CC9A-482F-4033-A7F8-503EE449DD0A"
46          },
47          {
48            "name": "name4",
49            "id": "17173587-C4E9-4369-9C43-FCBF5E075973"
50          },
51        ],
52        "partition": "development_mx",
53        "created":["at":"20130625105211", "by":10]
54      }
```

```
55   ]
56 }
```

9.4.5.1.4. Actualizar un host - PUT /api/virtshell/v1/hosts/:name

```
curl -sv -X PUT \
  -H 'accept: application/json' \
    -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"memory": "24GB",
     "capacity": "750GB"}' \
   'http://localhost:8080/api/virtshell/v1/hosts/host-01-
      pdn'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "update": "success" }
```

9.4.5.1.5. Eliminar un host - DELETE /api/virtshell/v1/hosts/:name

```
curl -sv -X DELETE \
   -H 'accept: application/json' \
   -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/api/virtshell/v1/hosts/host-01-
      pdn'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
‘‘‘
```

```
```json
{ "delete": "success" }
```

9.4.6. Instances

Representan las instancias de las máquinas virtuales o los contenedores. Los metodos soportados son:

Cuadro 9.6: Métodos HTTP para instances

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/provisioners/:name	Gets one provisioner by ID.
list	GET	/provisioners	Retrieves the list of provisioners.
create	POST	/provisioners/	Creates a new provisioner.
delete	DELETE	/provisioners/:name	Deletes an existing host.

Representación del recurso de un provisioner:

```
{
  "uuid": string,
  "name": string,
  "description": string,
  "enviroment": string,
  "provisioner": string,
  "host_type": string,
  "ipv4": string,
  "ipv6": string,
  "driver": string,
  "permissions": string,
  "created": {"at": timestamp, "by": string},
  "modified": {"at": timestamp, "by": string}
}
```
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "transactional_log",
  "description": "Server transactional only for store
    logs",
  "enviroment": "Enviroment name to which it belongs",
  "provisioner": "all_backend",
  "host_type": "GeneralPurpose | ComputeOptimized |
    MemoryOptimized | StorageOptimized",
  "ipv4": "172.16.56.104",
  "ipv6": "FE80:0000:0000:0000:0202:B3FF:FE1E:8329",
  "driver": "lxc | virtualbox | vmware | ec2 | kvm |
    docker",
  "permissions": "xwrxwrxwr",
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
    -8994-feff819cdc9f"},
  "modified": {"at":"1529207233", "by":"92d31132-8c9c-
    11e5-8994-feff819cdc9f"}
}
```

9.4.6.1. Ejemplos de peticiones HTTP

9.4.6.1.1. Crear una nueva instance - POST /api/virtshell/v1/instances

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "name": "transactional_log",
        "enviroment": "development_co",
```

```
6         "description": "Server transactional only for
              store logs",
7         "provisioner": "all_backend",
8         "host_type": "GeneralPurpose",
9         "driver": "lxc"
10      }' \
11    'http://localhost:8080/virtshell/api/v1/instances'
```

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 { "create": "in progress" }
```

9.4.6.1.2. Obtener un instance- GET /api/virtshell/v1/instances/:name

```
1 curl -sv -H 'accept: application/json'
2      -H 'X-VirtShell-Authorization: UserId:Signature' \
3      'http://<host>:<port>/api/virtshell/v1/instances/
          orders_colombia'
```

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
5   "name": "transactional_log",
6   "enviroment": "development_co",
7   "description": "Server transactional only for store
        logs",
8   "provisioner": "all_backend",
9   "host_type": "GeneralPurpose",
```

```
  "drive": "lxc",
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
     -8994-feff819cdc9f"},
  "modified": {"at":"1529207233", "by":"cf744732-8f12-
     11e5-8994-feff819cdc9f"}
  }
```

9.4.6.1.3. Obtener todos las instances - GET /api/virtshell/v1/instances

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://localhost:8080/api/virtshell/v1/instances'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "instances": [
    {
      "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
      "name": "transactional_log",
      "enviroment": "development_co",
      "description": "Server transactional only for store
         logs",
      "provisioner": "all_backend",
      "host_type": "GeneralPurpose",
      "drive": "lxc",
      "permissions": "xwrxwrxwr",
      "created": {"at":"1429207233", "by":"92d30f0c-8c9c-
        11e5-8994-feff819cdc9f"},
```

```
15        "modified": {"at":"1529207233", "by":"cf744732-8f12
            -11e5-8994-feff819cdc9f"}
16      },
17      {
18        "uuid": "cf744476-8f12-11e5-8994-feff819cdc9f",
19        "name": "orders_colombia",
20        "description": "Server transactional dedicated to
            receive orders",
21        "enviroment": "development_mx",
22        "provisioner": "all_backend",
23        "host_type": "StorageOptimized",
24        "drive": "docker",
25        "permissions": "xwrxwrxwr",
26        "created": {"at":"1429207233", "by":"92d30f0c-8c9c-
            11e5-8994-feff819cdc9f"},
27        "modified": {"at":"1529207233", "by":"92d31132-8c9c
            -11e5-8994-feff819cdc9f"}
28      }
29    ]
30  }
```

9.4.6.1.4. Eliminar una instance - DELETE /api/virtshell/v1/instances/:nae

```
1  curl -sv -X DELETE \
2     -H 'accept: application/json' \
3     -H 'X-VirtShell-Authorization: UserId:Signature' \
4     'http://<host>:<port>/api/virtshell/v1/instances/
        orders_colombia'
```

Response:

```
1  HTTP/1.1 200 OK
```

```
Content-Type: application/json
```
```
```json
{ "delete": "in progress" }
```

9.4.7. Tasks

Representan una tarea en VirtShell. Los métodos soportados son:

Cuadro 9.7: Métodos HTTP para tasks

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/tasks/:id	Gets one task by ID.
list	GET	/tasks	Retrieves the list of tasks.
get	GET	/tasks/status	Gets all task by status name.
create	POST	/tasks/	Creates a new task
update	PUT	/tasks/:id	Updates an existing task.

Representación del recurso de un task:

```
{
  "uuid": string,
  "description": string,
  "status" : string,
  "type": string,
  "object_uuid": string,
  "created":["at":"timestamp", "by":string],
  "last_update": "timestamp",
  "log": string
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
```
```

```
3   "description": "clone virtual machine database_01",
4   "status" : "pending|in progress|sucess|failed",
5   "type": "create_instance|delete_instance|
      restart_instance|...",
6   "object_uuid": "uuid of the object (instance, host,
      property, ...)",
7   "created":["at":"timestamp", "by":user_id],
8   "last_update": "timestamp",
9   "log": "summary of the task"
10 }
```

9.4.7.1. Ejemplos de peticiones HTTP

9.4.7.1.1. Crear una nueva tarea - POST /api/virtshell/v1/tasks

```
1 curl -sv -X POST \
2   -H 'accept: application/json' \
3     -H 'X-VirtShell-Authorization: UserId:Signature' \
4   -d '{ "description": "clone virtual machine database_01
      ",
5         "status" : "in progress"}' \
6    'http://localhost:8080/api/virtshell/v1/tasks'
```

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 { "create": "success" }
```

9.4.7.1.2. Obtener una tarea- GET /api/virtshell/v1/tasks/:id

```
1 curl -sv -H 'accept: application/json'
2      -H 'X-VirtShell-Authorization: UserId:Signature' \
```

```
3       'http://<host>:<port>/api/virtshell/v1/tasks/
          ab8076c0-db91-11e2-82ce-0002a5d5c51b'
```

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "description": "clone virtual machine database_01",
5   "status" : "in progress",
6   "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
      -8994-feff819cdc9f"},
7   "last_update": "1429207435",
8   "log": "summary of the task"
9 }
```

9.4.7.1.3. Obtener una tarea de acuerdo a su status- GET /api/virtshell/v1/tasks/:status

```
1 curl -sv -H 'accept: application/json'
2      -H 'X-VirtShell-Authorization: UserId:Signature' \
3      'http://<host>:<port>/api/virtshell/v1/tasks/sucess'
```

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "tasks": [
5     {
6       "uuid": "a62ad146-ccf4-11e5-9956-625662870761",
7       "description": "create container webserver_09",
8       "status" : "sucess",
```

```
      "created": {"at":"1454433171", "by":"cc7f8e2c-ccf4-
         11e5-9956-625662870761"},
      "last_update": "1454436771",
      "log": "summary of the task"
    }
  ]
}
```

9.4.7.1.4. Obtener todas las tareas - GET /api/virtshell/v1/tasks

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://<host>:<port>/api/virtshell/v1/tasks/'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "tasks": [
    {
      "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
      "description": "clone virtual machine database_01",
      "status" : "in progress",
      "created": {"at":"1429207233", "by":"92d30f0c-8c9c-
         11e5-8994-feff819cdc9f"},
      "last_update": "1429207435",
      "log": "summary of the task"
    },
    {
      "uuid": "a62ad146-ccf4-11e5-9956-625662870761",
      "description": "create container webserver_09",
      "status" : "sucess",
```

```
17        "created": {"at":"1454433171", "by":"cc7f8e2c-ccf4-
              11e5-9956-625662870761"},
18        "last_update": "1454436771",
19        "log": "summary of the task"
20      }
21    ]
22  }
```

9.4.7.1.5. Actualizar una tarea - PUT /api/virtshell/v1/tasks/:id

```
curl -sv -X PUT \
  -H 'accept: application/json' \
    -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"status": "sucess",
     "log": "....."}' \
   'http://localhost:8080/api/virtshell/v1/hosts/a62ad146
      -ccf4-11e5-9956-625662870761'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "update": "success" }
```

9.4.8. Properties

Representan propiedades de configuración de las máquinas virtuales o contenedores. Los metodos soportados son:

Cuadro 9.8: Métodos HTTP para properties

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/properties/	Install one or more packages.

Representación del recurso de un paquete:

```
{
  "properties": [
      {"name": "propertie_name1"},
      {"name": "propertie_name2"}
  ],
  "hosts": [
      {"name": "Host_", "range": "[1-3]"},
      {"name": "database_001"}
  ],
  "tags": [
    {"name": "db"},
    {"name": "web"}
  ]
}
```

Ejemplo:

```
{
  "properties": [
      {"name": "memory"},
      {"name": "cpu"}
  ],
  "hosts": [
      {"name": "Host_", "range": "[1-3]"}
  ]
}
```

9.4.8.1. Ejemplos de peticiones HTTP

9.4.8.1.1. Obtener una o mas propiedades de una unica instancia - POST /api/virtshell/v1/properties

```
curl -sv -X GET \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "properties": [{"name": "memory"}, {"name": "cpu"
     }],
         "hosts": [{"name": "WebServer"}]}' \
   'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

```
HTTP/1.1 202 OK
Content-Type: application/json
{
  "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
  "name": "Database001",
  "memory": 1024
}
```

9.4.8.1.2. Obtener una o mas propiedades de una o mas instancias por tag - POST /api/virtshell/v1/properties

```
curl -sv -X GET \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "properties": [{"name": "memory"}, {"name": "cpu"
     }],
         "tag": [{"name": "web"}]}' \
```

```
7     'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

```
1  HTTP/1.1 202 OK
2  Content-Type: application/json
3  {
4    properties: [
5      {
6       "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
7       "name": "WebServerPhp001",
8       "memory": 1024,
9       "cpu": 2
10     },
11     {
12      "id": "591b3828-7aaf-4833-a94c-ad0df44d59a4",
13      "name": "WebServerPhp002",
14      "memory": 1024,
15      "cpu": 1
16     }
17   ]
18 }
```

9.4.8.1.3. Obtener una o mas propiedades de una o mas instancias usando como prefijo un rango - POST /api/virtshell/v1/properties

```
1  curl -sv -X GET \
2    -H 'accept: application/json' \
3    -H "Content-Type: text/plain" \
4    -H 'X-VirtShell-Authorization: UserId:Signature' \
```

```
  -d '{ "properties": [{"name": "memory"}, {"name": "cpu"
     }],
         {"name": "Database00", "range": "[1-3]"}]}' \
   'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

```
HTTP/1.1 202 OK
Content-Type: application/json
{
  properties: [
    {
     "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
     "name": "Database001",
     "memory": 4024,
     "cpu": 2
    },
    {
     "id": "591b3828-7aaf-4833-a94c-ad0df44d59a4",
     "name": "Database002",
     "memory": 4024,
     "cpu": 1
    },
    {
     "id": "f7c81039-5c88-423b-8b0d-c124483d586b",
     "name": "Database003",
     "memory": 4024,
     "cpu": 3
    }
  ]
}
```

9.4.9. Provisioners

Representan los scripts que aprovisionan las máquinas virtuales o los contenedores. Los métodos soportados son:

Cuadro 9.9: Métodos HTTP para provisioners

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/provisioners/:name	Gets one provisioner by ID.
list	GET	/provisioners	Retrieves the list of provisioners.
create	POST	/provisioners/	Creates a new provisioner.
delete	DELETE	/provisioners/:name	Deletes an existing host.
update	PUT	/provisioners/:name	Updates an existing provisioner.

Representación del recurso de un provisioner:

```
{
  "uuid": string,
  "name": string,
  "description": string,
  "launch": number,
  "memory": number,
  "cpus": number,
  "hdsize": number,
  "image": string,
  "builder": string,
  "executor": string,
  "tag": string,
  "permissions": string,
  "depends": [ ... list of dependencies necessary for the
      builder ... ],
```

```
  "created": {"at":timestamp, "by":string},
  "modified": {"at":timestamp, "by":string}
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "backend-services-provisioner",
  "description": "Installs/Configures a backend server",
  "launch": 1,
  "memory": 4,
  "cpus": 2,
  "hdsize": 20,
  "image": "ubuntu_server_14.04.2_amd64",
  "builder": "https://github.com/janutechnology/
    VirtShell_Provisioners_Examples.git",
  "executor": "sh run1.sh",
  "tag": "backend",
  "permissions": "xwrxwrxwr",
  "depends": [ ... list of dependencies necessary for the
     builder ... ],
  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
    -8994-feff819cdc9f"},
  "modified": {"at":"1529207233", "by":"92d31132-8c9c-
    11e5-8994-feff819cdc9f"}
}
```

9.4.9.1. Ejemplos de peticiones HTTP

9.4.9.1.1. Crear un nuevo provisioner - POST /api/virtshell/v1/provisioners

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"name": "backend-services-provisioner",
       "launch": 1,
       "memory": 4,
       "cpus": 2,
       "hdsize": 20,
       "image": "ubuntu_server_14.04.2_amd64",
       "driver": "docker",
       "builder": "https://github.com/janutechnology/
          VirtShell_Provisioners_Examples.git",
       "executor": "sh run1.sh",
       "tag": "backend",
       "permissions": "xwrxwrxwr",
       "depends": [
            {"provisioner_name": "db-users", "version": "
               2.0.0"},
            {"provisioner_name": "db-transactional"}
        ]
     }' \
   'http://localhost:8080/virtshell/api/v1/provisioners'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "create": "success" }
```

9.4.9.1.2. Obtener un provisioner- GET /api/virtshell/v1/provisioners/:name

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://localhost:8080/api/virtshell/v1/provisioners
        /backend-services-provisioner'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
  {
    "name": "backend-services-provisioner",
    "launch": 1,
    "memory": 4,
    "cpus": 2,
    "hdsize": 20,
    "image": "ubuntu_server_14.04.2_amd64",
    "driver": "docker",
    "permissions": "xwrxwrxwr",
    "builder": "https://github.com/janutechnology/
       VirtShell_Provisioners_Examples.git",
    "executor": "sh run1.sh",
    "tag": "backend",
    "depends": [
        {"provisioner_name": "db-users", "version": "
           2.0.0"},
        {"provisioner_name": "db-transactional"}
    ],
    "created": {"at":"1429207233", "by":"420aa2c4-8d96-
       11e5-8994-feff819cdc9f"},
    "modified": {"at":"1529207233", "by":"92d31132-8c9c-
       11e5-8994-feff819cdc9f"}
  }
```

9.4.9.1.3. Obtener todos los provisioners - GET /api/virtshell/v1/provisioners

```
curl -sv -H 'accept: application/json'
     -H 'X-VirtShell-Authorization: UserId:Signature' \
     'http://localhost:8080/api/virtshell/v1/provisioners'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "provisioners": [
    {
      "name": "backend-services-provisioner",
      "launch": 1,
      "memory": 4,
      "cpus": 2,
      "hdsize": 20,
      "image": "ubuntu_server_14.04.2_amd64",
      "driver": "docker",
      "builder": "https://github.com/janutechnology/
        VirtShell_Provisioners_Examples.git",
      "executor": "sh run1.sh",
      "tag": "backend",
      "permissions": "xwrxwrxwr",
      "depends": [
          {"provisioner_name": "db-users", "version": "
            2.0.0"},
          {"provisioner_name": "db-transactional"}
      ]
    },
    {
```

```
23        "name": "db-transactional",
24        "launch": 2,
25        "memory": 8,
26        "cpus": 2,
27        "hdsize": 40,
28        "image": "ubuntu_server_14.04.2_amd64",
29        "driver": "docker",
30        "builder": "https://github.com/janutechnology/
            VirtShell_Provisioners_Examples.git",
31        "executor": "sh run_db.sh",
32        "tag": "db",
33        "permissions": "xwrxwrxwr"
34      }
35    ]
36  }
```

9.4.9.1.4. Actualizar un provisioner - PUT /api/virtshell/v1/provisioners/:name

```
1  curl -sv -X PUT \
2    -H 'accept: application/json' \
3    -H 'X-VirtShell-Authorization: UserId:Signature' \
4    -d '{ "executor": "run_backend.sh" }' \
5     'http://localhost:8080/api/virtshell/v1/provisioners/
        backend-services-provisioner
```

Response:

```
1  HTTP/1.1 200 OK
2  Content-Type: application/json
3
4  { "update": "success" }
```

9.4.9.1.5. Eliminar un provisioner - DELETE /api/virtshell/v1/provisioners/:name

```
curl -sv -X DELETE \
   -H 'accept: application/json' \
   -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/api/virtshell/v1/provisioners/
      backend-services-provisioner'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```json
{ "delete": "success" }
```

9.4.10. Images

Representan imagenes de máquinas virtuales o contenedores. Los metodos soportados son:

Cuadro 9.10: Métodos HTTP para images

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/images/:name	Gets one image by name.
list	GET	/images	Retrieves the list of images.
create	POST	/images/	Inserts a new image.
delete	DELETE	/images/:name	Deletes an existing image.

Representación del recurso de una imagen:

```
{
  "id": string,
  "name": string,
  "type": string,
  "os": string,
  "timezone": "America/Bogota",
  "key": string,
  "preseed_url": url,
  "download_url": url,
  "permissions" : string,
  "created":["at": timestamp,"by": string],
  "details": string
}
```

Ejemplo:

```
{
  "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
  "name": "ubuntu_server_14.04.2_amd64",
  "type": "iso",
  "os": "ubuntu",
  "timezone": "America/Bogota",
  "preseed_url": "https://<host>:<port>/api/virtshell/v1/
    files/seeds/seed_ubuntu14-04.txt",
  "download_url": "http://releases.ubuntu.com/raring/
    ubuntu-14.04-server-amd64.iso",
  "permissions" : "rwxrw----",
  "details": "ubuntu-trusty, version: 14.04.2, amd64-
    server"
  "created":["at":"20150625105211","by":10]
}
```

9.4.10.1. Ejemplos de peticiones HTTP

9.4.10.1.1. Crear una nueva imagen - POST /virtshell/api/v1/images

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"name": "ubuntu_server_14.04.2_amd64",
     "type": "iso",
     "os": "ubuntu",
     "timezone": "America/Bogota",
     "key": "/home/callanor/.ssh/id_rsa.pub",
     "permissions" : "rwxrwxr--",
     "preseed_url": "https://<host>:<port>/api/virtshell/
       v1/files/seeds/seed_ubuntu14-04.txt",
     "download_url": "http://releases.ubuntu.com/raring/
       ubuntu-14.04-server-amd64.iso"}' \
   'http://localhost:8080/api/virtshell/v1/image'
```

Respuesta:

```
HTTP/1.1 201 OK
Content-Type: application/json
{ "create": "success" }
```

9.4.10.1.2. Obtener una imagen - GET /virtshell/api/v1/images/:name

```
curl -sv -H 'accept: application/json'
```

```
2       -H 'X-VirtShell-Authorization: UserId:Signature' \
3       'http://localhost:8080/api/virtshell/v1/images/
          ubuntu_server_14.04.2_amd64'
```

Respuesta:

```
1  HTTP/1.1 200 OK
2  Content-Type: application/json
3  {
4    "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
5    "name": "ubuntu_server_14.04.2_amd64",
6    "type": "iso",
7    "os": "ubuntu",
8    "timezone": "America/Bogota",
9    "preseed_url": "https://<host>:<port>/api/virtshell/v1/
       files/seeds/seed_ubuntu_14_04.txt",
10   "download_url": "http://releases.ubuntu.com/raring/
       ubuntu-14.04-server-amd64.iso",
11   "permissions" : "rwxrwxrwx",
12   "created":["at":"20130625105211","by":10]
13 }
```

9.4.10.1.3. Obtener todas las imagenes - GET /virtshell/api/v1/images

```
1  curl -sv -H 'accept: application/json'
2       -H 'X-VirtShell-Authorization: UserId:Signature' \
3       'http://localhost:8080/api/virtshell/v1/images'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "images": [
    {
      "id": "b180ef2c-e798-4a8f-b23f-aaac2fb8f7e8",
      "name": "ubuntu_server_14.04.2_amd64",
      "type": "iso",
      "os": "ubuntu",
      "timezone": "America/Bogota",
      "preseed_file": "https://<host>:<port>/api/
        virtshell/v1/files/seeds/seed_file.txt",
      "download_url": "http://releases.ubuntu.com/raring/
        ubuntu-14.04-server-amd64.iso",
      "permissions" : "rwxrw----",
      "created":["at":"20130625105211","by":10]
    },
    {
      "id": "ca326181-bc84-4edb-bfc5-843037e7195e",
      "name": "centos:centos6",
      "type": "docker-container",
      "os": "centos",
      "permissions" : "rwxrwxr--",
      "created":["at":"20140625105211","by":12]
    }
  ]
}
```

9.4.10.1.4. Eliminar una imagen - DELETE /virtshell/api/v1/images/:name

```
curl -sv -X DELETE \
   -H 'accept: application/json' \
   -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://<host>:<port>/api/virtshell/v1/images/
      ubuntu_server_14.04.2_amd64'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```json
{ "delete": "success" }
```

9.4.11. Packages

Representan paquetes de software que se ejecutan en las máquinas virtuales o contenedores. Los metodos soportados son:

Cuadro 9.11: Métodos HTTP para packages

Acción	Metodo HTTP	Solicitud HTTP	Descripción
install	POST	/install_packages/	Install one or more packages.
upgrade	POST	/upgrade_packages/	Upgrade one or more packages.
remove	POST	/remove_packages/	Remove one or more packages.

Representación del recurso de un paquete:

```
{
  "packages": [
      {"name": "package_name1"},
      {"name": "package_name2"}
  ],
  "instances": [
      {"name": "Host_", "range": "[1-3]"},
      {"name": "database_001"}
  ],
  "tags": [
    {"name": "db"},
    {"name": "web"}
  ]
}
```

Ejemplo:

```
{
  "packages": [
      {"name": "git"},
      {"name": "nginx"}
  ],
  "instances": [
      {"name": "Host_", "range": "[1-3]"}
  ]
}
```

9.4.11.1. Ejemplos de peticiones HTTP

9.4.11.1.1. Instalar uno o mas paquetes - POST /api/virtshell/v1/install_packages

```
curl -sv -X PUT \
```

```
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "packages": [{"name": "git"}, {"name": "nginx"}],
        "instances": [{"name": "WebServer_", "range": "[1
          -3]"}]}' \
   'http://localhost:8080/api/virtshell/v1/
     install_packages'
```

Respuesta:

```
HTTP/1.1 202 Accepted
Content-Type: application/json
{ "install_package": "accepted" }
```

9.4.11.1.2. Actualizar uno o mas paquetes - POST /api/virtshell/v1/upgrade_package

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "packages": [{"name": "git"}, {"name": "nginx"},
     {"name": "mc"}],
        "instances": [{"name": "WebServer_", "range": "[1
          -3]"}]}' \
   'http://localhost:8080/api/virtshell/v1/
     upgrade_packages'
```

Respuesta:

```
HTTP/1.1 202 Accepted
Content-Type: application/json
{ "install_package": "accepted" }
```

9.4.11.1.3. Remover uno o mas paquetes - POST /api/virtshell/v1/remove_packages

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H "Content-Type: text/plain" \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "packages": [{"name": "apache2"}],
        "instances": [{"name": "WebServer_", "range": "[1
          -3]"}]}' \
   'http://localhost:8080/api/virtshell/v1/
     remove_packages'
```

Respuesta:

```
HTTP/1.1 202 Accepted
Content-Type: application/json
{ "install_package": "accepted" }
```

9.4.12. Files

Representan toda clase de archivos que se requieran para crear o aprovisionar maquinas virtuales o contenedores. Los metodos soportados son:

Cuadro 9.12: Métodos HTTP para files

Acción	Metodo HTTP	Solicitud HTTP	Descripción
get	GET	/files/:id	Gets one file by ID.
create	POST	/files/	upload a new file.
delete	DELETE	/files/:id	Deletes an existing file.
update	PUT	/files/:id	Updates an existing file.

Representación del recurso de un archivo:

```
{
  "uuid": string,
  "name": string,
  "folder_name" : string,
  "download_url": url,
  "permissions": string,
  "created":["at":"timestamp", "by": string]
}
```

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "ubuntu_seed_14-04.tex",
  "folder_name" : "ubuntu_seeds",
  "download_url": "https://<host>:<port>/api/virtshell/v1
    /files/ubuntu_seeds/ubuntu_seed_14-04.tex",
  "permissions": "rwxrwxrwx",
  "created": ["at":"20130625105211", "by":10]
}
```

9.4.12.1. Ejemplos de peticiones HTTP

9.4.12.1.1. Subir un nuevo archivo - POST /api/virtshell/v1/images

```
curl -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -H "Content-Type: multipart/form-data" \
  -F "file_data=@/path/to/file/seed_file.txt;filename=
     seed_file_ubuntu-14_04.txt" \
  -F "folder_name=ubuntu_seeds" \
  -F "permissions=rwxrwx---",
  'http://<host>:<port>/api/virtshell/v1/files'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "create": "success",
  "location": "http://<host>:<port>/api/virtshell/v1/
     files/ubuntu_seeds/seed_file_ubuntu-14_04.txt",
  "uuid": "51702a78-b23c-4625-bdda-3704cd0924b8"
}
```

9.4.12.1.2. Obtener un archivo - GET /api/virtshell/v1/files/:id

Para descargar un archivo, primero recibirá la url apropiada que viene en la metadata provista por la url. Luego podrá descargarlo usando la url.

```
curl -sv -H 'accept: application/json'
```

```
    -H 'X-VirtShell-Authorization: UserId:Signature' \
    'http://<host>:<port>/api/virtshell/v1/files/?id=
      ab8076c0-db91-11e2-82ce-0002a5d5c51b'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "file_name.extension",
  "folder_name" : "folder_name",
  "download_url": "http://<host>:<port>/api/virtshell/v1/
    files/ubuntu_seeds/seed_file_ubuntu-14_04.txt",
  "permissions": "rwxrwxr-x",
  "created":["at":"timestamp", "by":user_id]
}
```

9.4.12.1.3. Actualizar un archivo - PUT /api/virtshell/v1/files/:id

```
curl -sv -X PUT \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -H "Content-Type: multipart/form-data" \
  -F "file_data=@/path/to/file/seed_file.txt;filename=
    seed_file_ubuntu-14_04_v2.txt" \
  -F "permissions=rwxrwxrwx" \
   'http://localhost:8080/api/virtshell/v1/file/8de7b824-
     d7d1-4265-a3a6-5b46cc9b8ed5'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json

{ "update": "success" }
```

9.4.12.1.4. Eliminar un archivo - DELETE /api/virtshell/v1/files/:id

```
curl -sv -X DELETE \
   -H 'accept: application/json' \
   -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/api/virtshell/v1/fles/ab8076c0-
      db91-11e2-82ce-0002a5d5c51b'
```

Respuesta:

```
HTTP/1.1 200 OK
Content-Type: application/json
```
```
```json
{ "delete": "success" }
```

9.5. API Calls

9.5.1. Start Instance

Permite iniciar una instancia.
```

9.5.1.0.5. Iniciar una instance - POST /virtshell/api/v1/instances/start_instance/:id

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/virtshell/api/v1/instances/
      start\_instance/420aa3f0-8d96-11e5-8994-feff819cdc9f
      '
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "start": "success" }
```

9.5.2. Stop Instance

Permite detener una instancia.

9.5.2.0.6. Detener una instancia - POST /virtshell/api/v1/instances/stop_instance/:id

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/virtshell/api/v1/instances/stop
      \_instance/420aa3f0-8d96-11e5-8994-feff819cdc9f'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "stop": "success" }
```

9.5.3. Restart Instance

Permite reiniciar una instancia.

9.5.3.0.7. Reiniciar una instancia - POST /virtshell/api/v1/instances/restart_instance/:id

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/virtshell/api/v1/instances/
      restart\_instance/420aa3f0-8d96-11e5-8994-
      feff819cdc9f'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "restart": "success" }
```

9.5.4. Clone Instance

Permite clonar una instancia.

9.5.4.0.8. Clonar una instancia - POST /virtshell/api/v1/instances/clone_instance/:id

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
   'http://localhost:8080/virtshell/api/v1/instances/
      clone\_instance/420aa3f0-8d96-11e5-8994-feff819cdc9f
      '
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "clone": "success" }
```

9.5.5. Execute command

Permite ejecutar un comando en una o mas instancias.

Representación del recurso para ejecutar un comando:

```
{
  "instances": [ ... list of instances names, patterns(*|
     [numeric:numeric]) or tags ...],
  "command": string,
  "created": {"at": timestamp, "by": string}
}
```

Ejemplo:

```
{
  "instances": [
      {"name": "database\_server\_01"},
      {"name": "transactional\_server\_co"},
      {"pattern": "web\_server*"},
      {"pattern": "grid\_[1:5]"},
      {"tag": "web"}
  ],
  "command": "apt-get upgrade",
  "created": {"at": timestamp, "by": string}
}
```

9.5.5.0.9. Ejecutar un comando en una o mas instancias - POST /virtshell/api/v1/instances/execute_command/

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "instances": [
          {"name": "database\_server\_01"},
          {"name": "transactional\_server\_co"},
          {"pattern": "web\_server*"},
          {"pattern": "grid\_server\_[1:5]"},
          {"tag": "web"}
        ],
        "command": "apt-get upgrade" }' \
  'http://localhost:8080/virtshell/api/v1/instances/
     execute\_command/'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "execute_command": "success" }
```

9.5.6. Copy files

Permite ejecutar copiar uno archivo en una o mas instancias.

Representación del recurso para ejecutar un comando:

```
{
  "path": string,
  "destination": string,
  "instances": [ ... list of instances names, patterns(*|
     [numeric:numeric]) or tags ...],
  "created": {"at": timestamp, "by": string}
}
```

Ejemplo:

```
{
  "uuid_file": "0d832c60-7066-4d37-bd72-ce6ac4f61bcc",
  "destination": "$MYSQL_HOME/my.cnf"
  "instances": [
      {"name": "database\_server\_01"},
      {"name": "web\_server*"},
      {"name": "grid\_[1:5]"},
      {"name": "transactional\_server\_co"},
      {"tag": "web"}
  ]
}
```

9.5.6.0.10. Copiar un archivo en una o mas instancias - POST /virtshell/api/v1/instances/copy_files/

```
curl -sv -X POST \
  -H 'accept: application/json' \
  -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{ "uuid_file": "0d832c60-7066-4d37-bd72-
    ce6ac4f61bcc",
      "destination": "$MYSQL_HOME/my.cnf"
      "instances": [
          {"name": "database\_server\_01"},
          {"name": "web\_server*"},
          {"name": "grid\_[1:5]"},
          {"name": "transactional\_server\_co"},
          {"tag": "web"}
      ] }' \
  'http://localhost:8080/virtshell/api/v1/instances/copy\
    _files/'
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
{ "copy_files": "success" }
```

CAPÍTULO 10

Conclusiones

Como resultado del presente trabajo, se concluye que el diseño de VirtShell framework permite administrar y aprovisionar servicios de infraestructura de TI, debido a que se apoya en las actuales tecnologías de virtualización, permitiendo utilizar diferentes sistemas operativos a través de su interfaz web, personalizarlos, gestionar sus permisos y crear tantos como se requiera.

Por otro lado al comparar VirtShell contra las soluciones de virtualización actuales, se observa que el API REST proporciona todas las funcionalidades requeridas para controlar completamente los recursos físicos y virtuales reduciendo el tiempo necesario para crear e iniciar instancias. Así mismo, al adoptar el estilo arquitectural REST, se heredan nada menos que las propiedades del World Wide Web, las cuales ofrecen mayores ventajas sobre las demás soluciones, estas se explican a continuación:

Rendimiento Se puede soportar usando caches que permitan mantener la información cerca del procesamiento. Esto puede ayudar aún más, reduciendo la sobrecarga asociada con la creación de solicitudes complejas.

Escalibilidad Si el aumento de la demanda exige aumentar el número de servidores

de VirtShell, esto puede hacerse sin preocuparse por la sincronización entre los mismos, debido a que los datos de la sesión están contenidos en cada petición.

Simplicidad REST es un estilo de arquitectura orientado a la simpleza. El argumento central de esta propiedad es el HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar el dominio de VirtShell.

Visibilidad Rest está diseñado para ser visible y simple, lo que significa que cada aspecto del servicio debe ser auto-descriptivo siguiendo las normas HTTP.

Portabilidad Un Software es portable si el puede correr en diferentes ambientes, el estilo arquitectural REST permite separar las preocupaciones de interfaz de usuario de las preocupaciones de almacenamiento de datos, mejorando la portabilidad. Adicionalmente el World Wide Web es altamente portable debido a su nivel de estandarización.

Por lo anterior se concluye que la principal fortaleza de VirtShell y su diferencia frente a las demás soluciones de virtualización consiste en exponer sus funcionalidades por medio de un API REST, lo que le concede ademas capacidades de integración con diferentes plataformas de desarrollo.

APÉNDICE A

Disponible en GitHub

El código fuente de VirtShell se encuentra a disposición de cualquiera que quiera bajarlo, extenderlo y usarlo para cualquier fin incluso comercial.

Se elgió GitHub como sistema de control del versiones dado su gran popularidad en la comunidad de desarrollo y por su conjunto de características que ofrece hoy en día y que lo hacen muy útil para el trabajo en equipo.

La Url en github es: https://github.com/janutechnology/VirtShell

APÉNDICE B

Roadmap

Un RoadMap (que podría traducirse como hoja de ruta) es una planificación del desarrollo de un software con los objetivos a corto y largo plazo. A continuación se da una visión general de hacia adónde apunta VirtShell en el futuro:

- Implementar una interfaz web que permita administrar los ambientes y maquinas virtuales.
- Implementar algun mecanismo de seguridad que permita revisar las tramas que llegan y salen de las maquinas virtuales y los hosts.
- Realizar un plan de pruebas funcionales para los ambientes que se aprovisionan.
- Validar los datos de entrada de los json, tipos y campos mandatorios.
- Cambiar la forma de seleccionar un host para que tenga en cuenta las métricas e información del sistema de los anfitriones candidatos.
- Implementar scripts que permitan el despliegue del servidor de VirtShell en uno o mas servidores con balanceadores de carga.
- Integrar VirtShell con diferentes nubes privadas como Amazon.

- Mejorar la separación de la base de datos en varios servidores, implementando una capa de abstracción que permita el ruteo dinámico de los datos.
- Crear el servicio que proporcione monitorización para las instancias.
- Crear el servicio de auto scaling que permita escalar automaticamente las instancias en función de politicas definidas.
- Agregar capacidad al módulo de tareas para que diferentes acciones del framework puedan ser programadas como trabajos que se deban ejecutar por la ocurrencia de un evento, o en un tiempo especifico.
- Extender el servicio de creación de imágenes desatendidas para soportar diferentes distribuciones de sistemas operativos.

APÉNDICE C

Archivo de respuestas (pressed)

```
# regional setting
d-i debian-installer/language                           string      en_US:en
d-i debian-installer/country                            string      US
d-i debian-installer/locale                             string      en_US
d-i debian-installer/splash                             boolean     false
d-i localechooser/supported-locales                     multiselect en_US.UTF-8
d-i pkgsel/install-language-support                     boolean     true

# keyboard selection
d-i console-setup/ask_detect                            boolean     false
d-i keyboard-configuration/modelcode                    string      pc105
d-i keyboard-configuration/layoutcode                   string      us
d-i keyboard-configuration/variantcode                  string      intl
d-i keyboard-configuration/xkb-keymap                   select      us(intl)
d-i debconf/language                                    string      en_US:en

# network settings
d-i netcfg/choose_interface                             select      auto
d-i netcfg/dhcp_timeout                                 string      5
d-i netcfg/get_hostname                                 string      {{hostname}}
d-i netcfg/get_domain                                   string      {{hostname}}

# mirror settings
d-i mirror/country                                      string      manual
d-i mirror/http/hostname                                string      archive.ubuntu.com
d-i mirror/http/directory                               string      /ubuntu
d-i mirror/http/proxy                                   string

# clock and timezone settings
d-i time/zone                                           string      {{timezone}}
d-i clock-setup/utc                                     boolean     false
d-i clock-setup/ntp                                     boolean     true

# user account setup
d-i passwd/root-login                                   boolean     false
d-i passwd/make-user                                    boolean     true
```

```
d-i passwd/user-fullname                                string      {{username}}
d-i passwd/username                                     string      {{username}}
d-i passwd/user-password-crypted                        password    {{pwhash}}
d-i passwd/user-uid                                     string
d-i user-setup/allow-password-weak                      boolean     false
d-i passwd/user-default-groups                          string      adm cdrom dialout lpadmin plugdev
     sambashare
d-i user-setup/encrypt-home                             boolean     false

# configure apt
d-i apt-setup/restricted                                boolean     true
d-i apt-setup/universe                                  boolean     true
d-i apt-setup/backports                                 boolean     true
d-i apt-setup/services-select                           multiselect security
d-i apt-setup/security_host                             string      security.ubuntu.com
d-i apt-setup/security_path                             string      /ubuntu
tasksel tasksel/first                                   multiselect Basic Ubuntu server
d-i pkgsel/include                                                                      string
     openssh-server
d-i pkgsel/upgrade                                      select      safe-upgrade
d-i pkgsel/update-policy                                select      none
d-i pkgsel/updatedb                                     boolean     true

# disk partitioning
d-i partman/confirm_write_new_label                     boolean     true
d-i partman/choose_partition                            select      finish
d-i partman/confirm_nooverwrite                         boolean     true
d-i partman/confirm                                     boolean     true
d-i partman-auto/purge_lvm_from_device                  boolean     true
d-i partman-lvm/device_remove_lvm                       boolean     true
d-i partman-lvm/confirm                                 boolean     true
d-i partman-lvm/confirm_nooverwrite                     boolean     true
d-i partman-auto-lvm/no_boot                            boolean     true
d-i partman-md/device_remove_md                         boolean     true
d-i partman-md/confirm                                  boolean     true
d-i partman-md/confirm_nooverwrite                      boolean     true
d-i partman-auto/method                                 string      lvm
d-i partman-auto-lvm/guided_size                        string      max
d-i partman-partitioning/confirm_write_new_label        boolean     true

# grub boot loader
d-i grub-installer/only_debian                          boolean     true
d-i grub-installer/with_other_os                        boolean     true

# finish installation
d-i finish-install/reboot_in_progress                   note
d-i finish-install/keep-consoles                        boolean     false
d-i cdrom-detect/eject                                  boolean     true
d-i debian-installer/exit/halt                          boolean     false
d-i debian-installer/exit/poweroff                      boolean     false
```

Bibliografía

Bibliografía

[1] ACM Symposium on Cloud Computing 2015, http://acmsocc.github.io/2015/, 2015.

[2] Andrew S. Tanembaum, Modern Operating Systems 4th Edition, 2014.

[3] Fabric, http://www.fabfile.org/, 2016.

[4] Ubuntu Juju, http://www.ubuntu.com/cloud/juju, 2016.

[5] Docker Composer, https://docs.docker.com/compose/, 2016.

[6] SmartFrog, http://smartfrog.org/display/sf/SmartFrog+Home, 2009.

[7] Ansible, https://www.ansible.com/, 2016.

[8] Amazon EC2, https://aws.amazon.com/ec2/, 2016.

[9] Chef, https://docs.chef.io/, 2015.

[10] Puppet, http://projects.puppetlabs.com/projects/puppet, 2015.

[11] Cfengine, https://www.gnu.org/software/cfengine/, 2001.

[12] Bdfg2, http://bcfg2.org/, 2015.

[13] Cobbler, https://cobbler.github.io/, 2016.

[14] Vagrant, https://www.vagrantup.com/, 2016.

[15] SaltStack, http://saltstack.com/community/, 2016.

[16] Architectural Styles and the Design of Network-based Software Architectures, https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm, 2000.

[17] Inversion of Control Containers and the Dependency Injection pattern, http://martinfowler.com/articles/injection.html, 2004.

[18] HMAC: Keyed-Hashing for Message Authentication, https://www.ietf.org/rfc/rfc2104.txt, 1997.

[19] Linux Containers, https://linuxcontainers.org, 2016..

[20] LXC Templates Documentation, https://help.ubuntu.com/lts/serverguide/lxc.html, 2016.

[21] Docker, https://docker.com, 2016.

[22] REST - An Alternative to RPC for Web Services Architecture, http://web.uvic.ca/~erikj/seng422/resources/rest_paper.pdf, 2009.