

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 Jhon 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 mi familia sin ella, no hubiera podido alcanzar esta meta.
- A todas aquellas personas que de una u otra forma colaboraron en la realización del presente trabajo.

---

## Índice general

---

<b>Agradecimientos</b>	<b>2</b>
<b>1. Introducción</b>	<b>6</b>
1.1. Contenido de la tesis . . . . .	7
<b>2. Estado del arte</b>	<b>8</b>
2.1. Técnicas de Virtualización . . . . .	9
2.2. Soluciones de aprovisionamiento . . . . .	12
<b>3. Arquitectura de VirtShell</b>	<b>18</b>
3.1. Características . . . . .	19
3.2. Módulos . . . . .	20
3.2.1. Security . . . . .	20
3.2.2. Managment . . . . .	22
3.2.3. Provisioning . . . . .	23
3.2.4. Agents . . . . .	25
<b>4. Seguridad</b>	<b>26</b>
4.1. Autenticación . . . . .	26
4.1.1. Authentication Header . . . . .	27

---

4.2. Autorización . . . . .	29
<b>5. Adminsitración</b>	<b>32</b>
5.1. Particiones, anfitriones y Ambientes en VirtShell . . . . .	32
5.1.1. Particiones . . . . .	32
5.1.2. Asociación de anfitriones a particiones . . . . .	34
5.1.3. División de particiones en ambientes . . . . .	36
5.2. Instancias . . . . .	37
5.2.1. Creación de instancias en un ambiente . . . . .	38
5.3. Tareas . . . . .	40
5.4. Propiedades . . . . .	42
<b>6. Aprovisionamiento</b>	<b>45</b>
<b>7. Agentes</b>	<b>46</b>
7.1. Agente de Aprovisionamiento . . . . .	46
7.2. Agente de Monitoreo . . . . .	46
7.3. Agente de Administración . . . . .	47
<b>8. API</b>	<b>48</b>
8.1. Definición de API? . . . . .	48
8.2. Tipos de APIs . . . . .	48
8.2.1. RPC . . . . .	49
8.2.2. REST . . . . .	50
8.2.3. VirtShell API REST . . . . .	51
8.3. Formato de entrada y salida . . . . .	52
8.4. Codigos de error . . . . .	52
8.5. API Resources . . . . .	53
8.5.1. Groups . . . . .	53
8.5.2. Users . . . . .	58
8.5.3. Partitions . . . . .	62
8.5.4. Hosts . . . . .	67
8.5.5. Hosts . . . . .	72
8.5.6. Instances . . . . .	78

---

8.5.7. Tasks . . . . .	84
8.5.8. Properties . . . . .	89
8.5.9. Provisioners . . . . .	93
8.5.10. Images . . . . .	99
8.5.11. Packages . . . . .	104
8.5.12. Files . . . . .	107
8.6. API Calls . . . . .	111
8.6.1. Start Instance . . . . .	111
8.6.2. Stop Instance . . . . .	112
8.6.3. Restart Instance . . . . .	113
8.6.4. Clone Instance . . . . .	113
8.6.5. Execute command . . . . .	114
8.6.6. Copy files . . . . .	115
<b>9. Experiencia y Evaluación</b>	<b>118</b>
<b>A. Disponible en GitHub</b>	<b>119</b>
<b>B. Roadmap</b>	<b>120</b>
<b>Bibliografía</b>	<b>122</b>

---

## Índice de figuras

---

2.1. Máquina virtual . . . . .	10
2.2. Contenedores . . . . .	11
3.1. Visión general del framework de VirtShell . . . . .	21
5.1. Ejemplo de ambientes en un partición . . . . .	37

---

## Índice de cuadros

---

4.1. Tipos de permisos . . . . .	30
4.2. Atributos básicos . . . . .	30
8.1. Métodos HTTP para groups . . . . .	53
8.2. Métodos HTTP para users . . . . .	58
8.3. Métodos HTTP para partitions . . . . .	62
8.4. Métodos HTTP para enviroments . . . . .	67
8.5. Métodos HTTP para hosts . . . . .	72
8.6. Métodos HTTP para instances . . . . .	78
8.7. Métodos HTTP para tasks . . . . .	84
8.8. Métodos HTTP para properties . . . . .	89
8.9. Métodos HTTP para provisioners . . . . .	93
8.10. Métodos HTTP para images . . . . .	99
8.11. Métodos HTTP para packages . . . . .	104
8.12. Métodos HTTP para files . . . . .	107

---

# CAPÍTULO 1

---

## Introducción

---

Ideas para la introduccion:

En los últimos años se han producido importantes avances tecnológicos que han dado lugar a la demanda de nuevas aplicaciones relacionadas con la automatización de .....

Por un lado ha aumentado el número de aplicaciones de ..... que requieren estrategias ..... capaces de ..... . Por otro lado, las aplicaciones de ultima generacion. En esta tesis se propone una estrategia distinta de aprovisionamiento y administracion, que cumplen los requisitos de velocidad y facilidad de uso demandados por los usuarios.

Se propone un sistema web rapido y eficaz, capas de ..... para aprovisionar .....

basada en el popular metodo de ....., capaz de .....

La estrategia propuesta mejora muy notablemente la .....



## **1.1. Contenido de la tesis**

Además del capítulo de introducción, la tesis consta de otros XXXX capítulos y de NNNNN apéndices. A continuación se describe brevemente el contenido de cada uno de ellos.

En el capítulo 2 se presenta el estado del arte correspondiente a los tipos de virtualización existentes y las diferentes herramientas que existen en el mercado para realizar aprovisionamiento virtual.

...

En el apendice A, ...

---

## CAPÍTULO 2

---

### 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 áreas 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.

## 2.1. 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 2.1). 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.

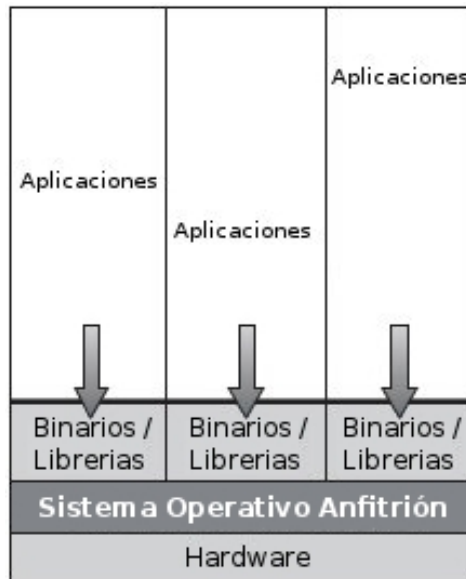
Figura 2.1: Máquina virtual



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 2.2). 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 <sup>1</sup>.

<sup>1</sup>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 2.2: Contenedores



De hecho, sin importar la técnica de virtualización que se use, la instalación de una máquina 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 más capacidad de cómputo, 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 más cercana de las ya existentes. En el momento de la creación óptima de la imagen un administrador puede tener dificultades y preguntas como: ¿cuál es la mejor configuración?, ¿cuáles paquetes y sus dependencias deberían ser instalados? 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 dependencia 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.

## 2.2. 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 <sup>2</sup> web para realizar

---

<sup>2</sup> 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.

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 <sup>3</sup> 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 específico, 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 características, 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 de-

---

<sup>3</sup>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

scribe los recursos del sistema y sus estados utilizando el lenguaje declarativo 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 específico 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, además 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 <sup>4</sup> es similar a las máquinas virtuales de otros sistemas. Básicamente están 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 válida 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 línea 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].

---

<sup>4</sup>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).

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

**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 <sup>5</sup> 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]

---

<sup>5</sup>es un programa intermediario que traduce los mensajes de un sistema desde un lenguaje a otro, a través de un medio de telecomunicaciones.

---

## CAPÍTULO 3

---

### Arquitectura de VirtShell

---

VirtShell Framework es una plataforma que proporciona herramientas para la automatización y gestión de infraestructura a través del protocolo HTTP <sup>1</sup>. En otros términos, VirtShell facilita la creación, despliegue, mantenimiento y monitoreo tanto de recursos virtuales como físicos por medio de una API REST [16]. Esto permite que cualquier desarrollo de software con acceso a internet (sitio web, aplicación móvil, etc.) pueda utilizar VirtShell e interactuar con la infraestructura tan solo haciendo llamadas a direcciones de internet.

VirtShell basa principalmente su funcionamiento en scripts de shell re-utilizables, que facilitan la instalación y configuración de cualquier tipo de servidor o grandes soluciones que involucren varios recursos virtuales sin importar su tamaño. Sin embargo el lenguaje de los scripts no necesariamente debe ser el shell también puede interactuar con cualquier lenguaje de programación que el usuario prefiera.

VirtShell es un framework de código abierto y bajo la licencia BSD, que permite utilizarlo para proyectos de cualquier tipo, incluso comerciales.

---

<sup>1</sup>Hypertext Transfer Protocol o HTTP, es el protocolo de comunicación que permite las transferencias de información en la World Wide Web

## 3.1. Características

**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 de 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.

**Escalable** Al ser VirtShell de código abierto, el API puede modificarse, crecer fácilmente y versionarse de diferentes maneras.

**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** VirtShell fue diseñado con la idea de cargar código dinámicamente fácilmente, permitiendo extender el comportamiento del framework agregando plugins en tiempo de ejecución. Adicionalmente 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, 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 permiten declarar el listado de dependencias de recursos virtuales que tiene un script 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, 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.

## 3.2. Módulos

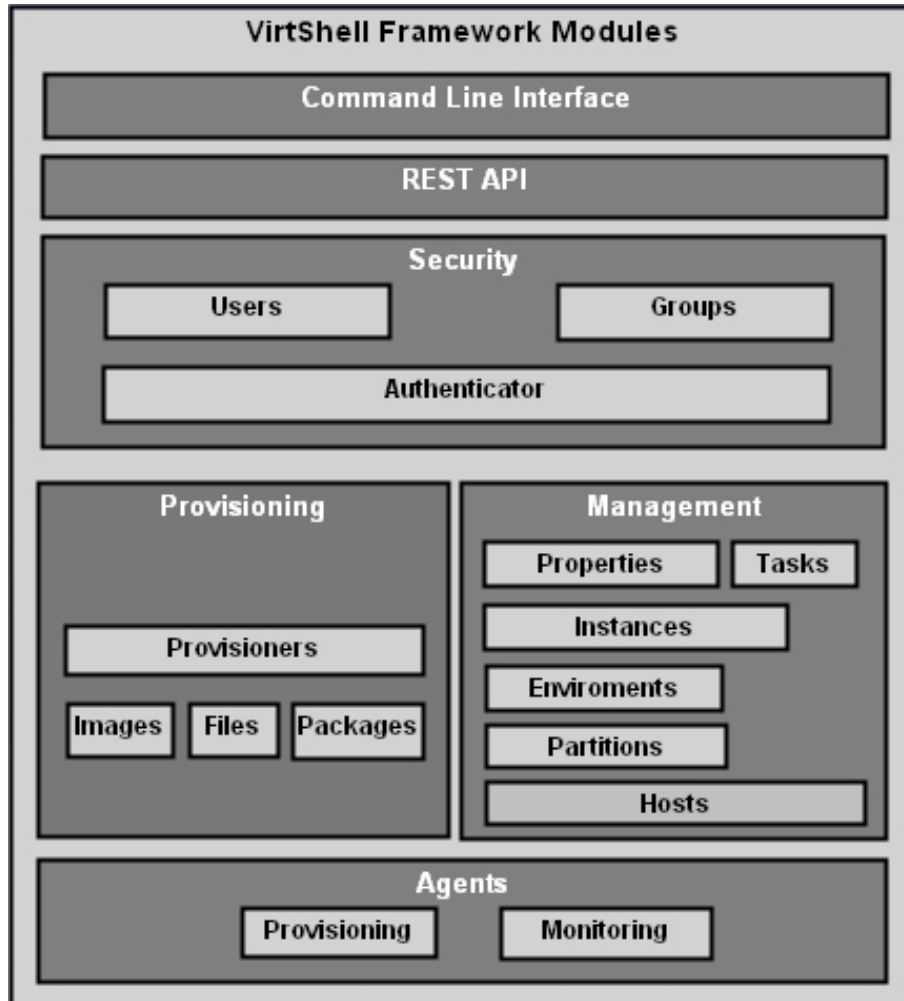
VirtShell Framework consiste de características organizadas en 13 módulos. Estos módulos son agrupados en Seguridad, Administración y Aprovisionamiento. Estos elementos se usan de manera separada pero trabajan juntos para proveer la información necesaria para que los agentes realicen su trabajo en los hosts que albergaran los recursos virtuales, como se muestra en la figura 3.1.

Las siguientes secciones detallan los módulos disponibles para cada característica.

### 3.2.1. Security

Seguridad consiste de los módulos de usuarios (users), grupos (groups) y el modulo de autenticación (authenticator). 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.

Figura 3.1: Visión general del framework de VirtShell



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 es, de hecho, quien decide si tiene permiso para ingresar al sistema y el nivel de acceso a un recurso dado. En el capítulo 4 se detalla el proceso de autenticación y autorización.

### 3.2.2. Managment

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

El módulo de **anfitriones** lleva registro de nodos físicos, servidores o máquinas virtuales, que se encuentren conectados a la red y que permitan albergar instancias virtuales. Los anfitriones 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 en 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 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 a 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 con los módulos de VirtShell es a través de un API REST, una petición de creación de un nuevo recurso virtual, puede tomar algo de tiempo, y puede ser un tiempo 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.

### 3.2.3. Provisioning

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

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 <sup>2</sup>, 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 liviano a VirtShell, los scripts de aprovisionamiento, deben estar registrados en un repositorio git publico. VirtShell se encarga de descargar el repositorio y de ejecutar los scripts de aprovisionamiento, de acuerdo a la forma configurada en que se deba ejecutar.

Adicionalmente, los aprovisionadores, cuentan con una manera de especificar las dependencias del nuevo recurso virtual. VirtShell se encarga de resolver las dependencias antes y de suministrar información de esas dependencias a los scripts de

---

<sup>2</sup>git es un software de control de versiones diseñado por Linus Torvalds

aprovisionamiento si estos lo requieren.

VirtShell utiliza como lenguaje de referencia, para la creación de scripts de aprovisionamiento, el lenguaje shell, el cual 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, facilitando la escritura de los mismos o permitiendo hacerlos independientes del sistema operativo en el cual van a ejecutarse.

El módulo de **imágenes** proporciona la información necesaria de las imágenes que encuentran registradas en el sistema. Cada vez que se crea un nuevo recurso virtual en un anfitrión, se especifica el nombre de la imagen 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 y 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 publico de los diferentes proveedores.

El módulo de imágenes cuenta también, con la característica de crear automáticamente nuevas imágenes de tipo iso a partir de los releases 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 enviar archivos a uno o mas recursos virtuales de manera simultanea en un directorio especificado. Adicionalmente, permite especificar los permisos que tendrán los archivos. Cabe anotar, que estos archivos deben estar previamente subidos al sistema.

El módulo de **paquetes** facilita realizar funciones tales como la instalación de nuevos paquetes de software, actualización de paquetes existentes en uno o mas recursos virtuales de manera simultanea.

### 3.2.4. Agents

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

VirtShell, cuenta con tres tipos de agentes: de aprovisionamiento, monitoreo y administración. El agente de aprovisionamiento es de tipo reactivo y se encarga de dar respuesta a todas las peticiones de aprovisionamiento sobre uno o mas recursos virtuales en el anfitrión.

El agente de monitoreo es completamente autónomo y su función es la de supervisar al agente de aprovisionamiento y reportar el estado de salud de los anfitriones y recursos virtuales que se encuentran creados en su interior.

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

Los agentes son instalados de manera automática en el anfitrión cuando este es agregado al sistema por medio del API REST, liberando de esta tarea al administrador del sistema.

---

## CAPÍTULO 4

---

### Seguridad

---

#### 4.1. 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 4.1: 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: OPN5J17HBGZHT7JJ3X82:
  frJIUN8DYpKDtOLCwoHGTY/45U
```

#### 4.1.1. 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 <sup>1</sup> [17], por lo que la parte de la firma de la cabecera de autorización variará de una

---

<sup>1</sup>En la criptografía, un código de autenticación de mensajes en clave-hash (HMAC) es una construcción específica para calcular un código de autenticación 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 4.2: cabecera de una solicitud de autorización

```
1  Authorization = VirtShellUserId + ":" + Signature;
2
3  Signature = Base64( HMAC-SHA1( UTF-8-Encoding-Of(
4      YourSecretAccessKeyID, StringToSign ) ) );
5
6  StringToSign = HTTP-Verb + "\n" +
7  Host + "\n" +
8  Content-MD5 + "\n" +
9  Content-Type + "\n" +
10 Date + "\n" +
11 CanonicalizedResource;
12
13 CanonicalizedResource = <HTTP-Request-URI, from the
    protocol name up
    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 [17]).

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*<sup>2</sup>. El parámetro de la petición de la firma se construye codificado en Base64.

**4.1.1.0.1. 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 denomina *canonización*.

**4.1.1.0.2. Tiempo de sello** Un sello de tiempo <sup>2</sup> 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.

## 4.2. 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

---

<sup>2</sup>fecha y hora obtenida del sistema en que se genera la petición HTTP

en la tabla 4.1.

Cuadro 4.1: 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 4.2: 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        "hdspace": "2GB",
8        "operating_system": "ubuntu_server_14.04.2_amd64"
9        ,
10       "description": "Server transactional only for
11       store logs",
12       "provisioner": "all_backend",
13       "host_type": "GeneralPurpose",
14       "driver": "lxc",
15       "permissions": "rwx-----"
16     }' \
17     '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 está 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 público para todos los usuarios del sistema.

---

## CAPÍTULO 5

---

### Administación

---

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

### 5.1. Particiones, anfitriones y Ambientes en VirtShell

En VirtShell hay tres conceptos que son muy importantes y se extiende a través de todos los servicios, y que simplemente no puede dejar de tener en cuenta: Las particiones, los ambientes y las instancias. Las tres se asocian con la mayoría de las cosas en VirtShell, y el dominio de ellos es crucial para una buena administración de los dispositivos.

#### 5.1.1. 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 organizar las máquinas que albergaran recursos virtuales en partes aisladas de las demás. Estas partes pueden estar ubicadas en un mismo sitio físico o por el contrario puede estar distribuidas en diferentes zonas geográficas de todo el mundo.

Si solo se cuenta con un número fijo de máquinas (o anfitriones) ubicadas en un mismo sitio físico como por ejemplo un datacenter <sup>1</sup>, lo que se obtiene con las particiones es la posibilidad de dividir esas máquinas en subgrupos que puedan ser destinados para diferentes equipos o divisiones dentro de una organización.

Al contar con máquinas 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 distribuye 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 vacía, sin anfitriones. Para asociar anfitriones a una partición se debe crear un anfitrión y vincularlo con la partición como se verá más adelante en este mismo capítulo. Un ejemplo de cómo crear una partición usando el API se muestra en el siguiente código:

Listing 5.1: Petición HTTP para crear una partición

```
1 curl -sv -X POST \  
2   -H 'accept: application/json' \  
3   -H 'X-VirtShell-Authorization: UserId:Signature' \  
4   -d '{  
5       "name": "development_co",  
6       "description": "Collection of servers oriented to
```

<sup>1</sup>Un data center también llamado centro de datos es un espacio acondicionado especialmente para contener a todos los equipos y sistemas de TI

```
7         development team in colombia."
8     }, '\n'
9     'http://localhost:8080/api/virtshell/v1/partitions'
```

### 5.1.2. Asociación de anfitriones a particiones

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, 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.

Cuando se crea un anfitrión en VirtShell este debe ser asociado a una sola partición, asignándole un tipo, con alguno de los mencionados anteriormente, estableciendo las 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 usando el API:

Listing 5.2: Petición HTTP para crear un host

```
1 curl -sv -X POST \  
2   -H 'accept: application/json' \  
3   -H 'X-VirtShell-Authorization: UserId:Signature' \  
4   -d '{ "name": "host-01-pdn",  
5       "os": "Ubuntu_12.04_3.5.0-23.x86_64",  
6       "memory": "16GB",  
7       "capacity": "120GB",  
8       "enabled": "true",  
9       "type" : "GeneralPurpose",  
10      "local_ipv4": "15.54.88.19",  
11      "local_ipv6": "ff06:0:0:0:0:0:0:c3",  
12      "public_ipv4": "10.54.88.19",  
13      "public_ipv6": "yt06:0:0:0:0:0:0:c3",  
14      "partition": "development_co"}', \  
15   '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 development\_co. En la siguiente consulta al API se muestra el resultado de la asociación:

Listing 5.3: Petición HTTP para consultar una partición por su nombre

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

Respuesta:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "uuid": "efa1777c-cad7-11e5-9956-625662870761",
5   "name": "development_co",
6   "description": "Collection of servers oriented to
7     development team in colombia.",
8   "hosts": [ "host-01-pdn" ],
9   "created": { "at": "1429207233",
10               "by": "1a900cdc-cad8-11e5-9956-625662870761"
11             }
12 }
```

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

### 5.1.3. División de particiones en ambientes

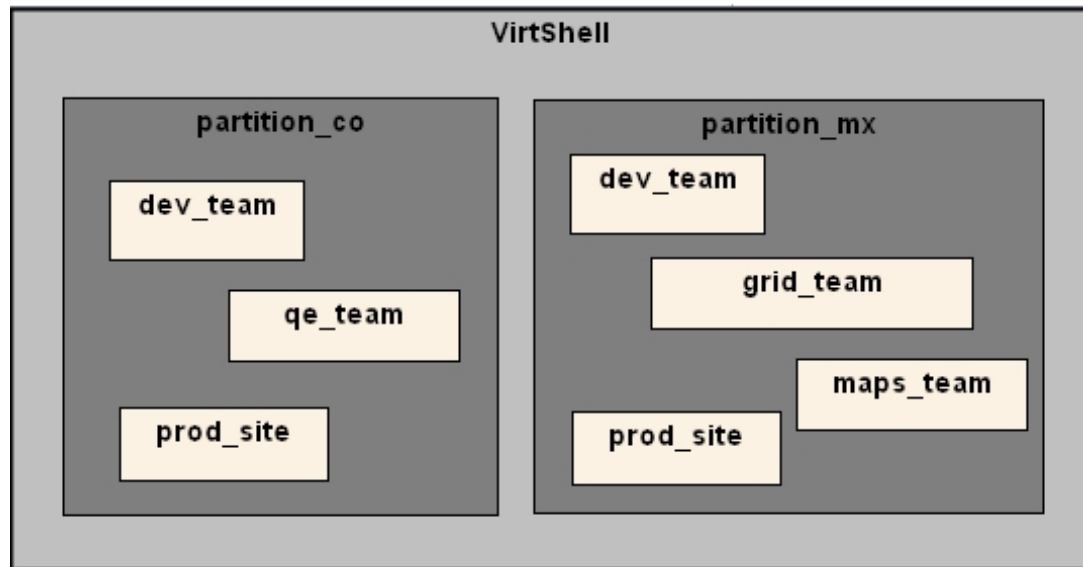
Las particiones se refieren a la forma de organizar lugares físicos. Los ambientes por el contrario, son lugares lógicos y aislados dentro de una partición. Cada partición puede estar compuesta por uno o varios ambientes, y un ambiente puede contener uno o más instancias. Cada ambiente pertenece a una sola partición. La figura 5.1 muestra un ejemplo de cómo dos particiones contienen varios ambientes destinados a equipos de trabajo diferentes.

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

Listing 5.4: Petición HTTP para crear un ambiente

```
1 curl -sv -X POST \
```

Figura 5.1: Ejemplo de ambientes en un partición



```
2 -H 'accept: application/json' \  
3 -H 'X-VirtShell-Authorization: UserId:Signature' \  
4 -d '{  
5     "name": "bigdata_laboratory",  
6     "description": "Collection of servers oriented to  
7         big data.",  
8     "partition": "bogota_partition_co"  
9 }' \  
'http://localhost:8080/api/virtshell/v1/enviroments'
```

## 5.2. 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 obje-

tivos 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 <sup>2</sup> [18] y Docker <sup>3</sup> [19].

Las instancias son designadas a solo ambiente de trabajo en el momento de ser creadas. En el instante de su creación, se debe especificar las características de CPU, memoria y capacidad de disco que se desea. De igual manera, se configura el sistema operativo, los permisos para los demás usuarios del sistema, el tipo de anfitrión que necesita y la tecnología que se desea usar para virtualizar.

### 5.2.1. Creación de instancias en un ambiente

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 y 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 sigu-

---

<sup>2</sup>LXC (Linux Containers) es una tecnología de virtualización en el nivel de sistema operativo (SO) para Linux.

<sup>3</sup>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.



iente código:

Listing 5.5: Petición HTTP para crear una instancia

```
1 curl -sv -X POST \  
2   -H 'accept: application/json' \  
3   -H 'X-VirtShell-Authorization: UserId:Signature' \  
4   -d '{ "name": "transactional_log",  
5       "memory": 8024,  
6       "cpus": 2,  
7       "hdsiz": "20GB",  
8       "enviroment": "development_co",  
9       "operating_system": "ubuntu_server_14.04.2_amd64"  
10      ,  
11      "description": "Server transactional only for  
12      store logs",  
13      "provisioner": "all_backend",  
14      "host_type": "GeneralPurpose",  
15      "driver": "lxc"  
16    }' \  
17   'http://localhost:8080/api/virtshell/v1/instances'
```

En esta solicitud HTTP se observa que la instancia a crear debe tener el sistema operativo es Linux Ubuntu, contar con 8 Gigabytes de memoria RAM, dos núcleos, 20 Gigabyte de disco duro, y que 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 5.6: 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 5.7: 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 que permite realizar son: iniciar, detener, reiniciar, cerrar y clonar. Así mismo, el API posibilita la fácil y rápida búsqueda de una o mas instancias.

### 5.3. 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, y esta pueda 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 <sup>4</sup> 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 5.8: 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 5.9: Ejemplo de consulta de una tarea

```
1 curl -sv -H 'accept: application/json'
2 -H 'X-VirtShell-Authorization: UserId:Signature' \
3 'http://<host>:<port>/api/virtshell/v1/tasks/908b4d78-8b7a-40d7-9ecf-5036eeb5526b'
```

Respuesta:

---

<sup>4</sup>UUID son las siglas en inglés del Identificador Universalmente Único.

Listing 5.10: Ejemplo del detalle de una tarea

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 '''
4 '''json
5 {
6   "uuid": "908b4d78-8b7a-40d7-9ecf-5036eeb5526b",
7   "description": "install-package in instance database_01",
8   "status" : "installing",
9   "created": {"at": "1429207233", "by": "92d30f0c-8c9c-11e5-8994-feff819cdc9f"},
10  "last_update": "1429207435",
11  "log": "installing package couchdb"
12 }
```

## 5.4. 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 5.11: Ejemplo consultando varias propiedades a una instancia

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

```
3 -H "Content-Type: text/plain" \  
4 -H 'X-VirtShell-Authorization: UserId:Signature' \  
5 -d '{ "properties": [{ "name": "memory"}, { "name": "cpu"  
    }],  
6     "hosts": [{ "name": "WebServer001"}]}' \  
7 'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

Listing 5.12: Respuesta

```
1 HTTP/1.1 200 OK  
2 Content-Type: application/json  
3 ''  
4 '' json  
5 {  
6   "name": "WebServer001",  
7   "memory": 8024,  
8   "cpu": 87.4  
9 }
```

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

Listing 5.13: Ejemplo consultando propiedades a varias instancias

```
1 curl -sv -X GET \  
2 -H 'accept: application/json' \  
3 -H "Content-Type: text/plain" \  
4 -H 'X-VirtShell-Authorization: UserId:Signature' \  
5 -d '{ "properties": [{ "name": "memory"}, { "name": "cpu"  
    }],  
6     [{ "name": "Database00", "range": "[1-3]"}]}' \  
7 'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

Listing 5.14: Respuesta

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 ''
4 '''json
5 {
6   properties: [
7     {
8       "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
9       "name": "Database001",
10      "memory": 4024,
11      "cpu": 2
12    },
13    {
14      "id": "591b3828-7aaf-4833-a94c-ad0df44d59a4",
15      "name": "Database002",
16      "memory": 4024,
17      "cpu": 1
18    },
19    {
20      "id": "f7c81039-5c88-423b-8b0d-c124483d586b",
21      "name": "Database003",
22      "memory": 4024,
23      "cpu": 3
24    }
25  ]
26 }
```

---

## CAPÍTULO 6

---

### Aprovisionamiento

---

---

## CAPÍTULO 7

---

### 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. Existen tres tipos de agentes: de aprovisionamiento, monitoreo y administración.

#### 7.1. Agente de Aprovisionamiento

El agente de aprovisionamiento se encarga de instalar y configurar paquetes, librerías y aplicaciones en las instancias. De igual manera se encarga de resolver las dependencias de otras instancias que tenga la instancia que se este procesando. Es decir, .....

#### 7.2. Agente de Monitoreo

El agente de monitoreo es completamente autónomo y su función es la de supervisar al agente de aprovisionamiento y reportar el estado de salud de los anfitriones y recursos virtuales que se encuentran creados en su interior.



### **7.3. Agente de Administración**

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

---

## CAPÍTULO 8

---

### API

---

#### 8.1. 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: cuáles 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.

#### 8.2. Tipos de APIs

Las APIs web pueden ser divididas en dos categorías generales:

- Remote Procedure Call (RPC)

- REpresentational State Transfer (REST)

### 8.2.1. 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 autodocumentado 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 mal elección para las API web:

- 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.

### 8.2.2. REST

REST es un estilo de arquitectura de software que proporciona un enfoque práctico 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 metodos HTTP disponibles, como se manejaran características de autenticación, como se suministrarán las credenciales en cada petición, etc.

En pocas palabras, la mayoría 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.

### 8.2.3. VirtShell API REST

En el VirtShell API REST un usuario envía 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 envía 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 está 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.

### 8.3. 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](http://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).

### 8.4. 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 permanente 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 código 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 vacío, daría lugar a la siguiente respuesta:

```
1 HTTP/1.1 400 Bad Request
2 Content-Type: application/json
3
4 {"error": "Missing input for create instance"}
```

## 8.5. API Resources

### 8.5.1. Groups

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

Cuadro 8.1: Métodos HTTP para groups

Acción	Método 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:

```
1 {
2   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3   "name": "web_development_team",
4   "users": [ ... list of members of the group ... ],
5   "created": [ {"at": "timestamp"}, {"by": user_id} ]
6 }
```

Ejemplo:

```
1 {
2   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3   "name": "web_development_team",
4   "users": [
5     {"username": "user1", "id": "a146cae4-8c90-11e5-
6       8994-feff819cdc9f"},
7     {"username": "user2", "id": "a146d00c-8c90-11e5-
8       8994-feff819cdc9f"}
9   ]
10  "created": [{"at": "1447696674"}, {"by": "a379e8e6-8c8b-
11    11e5-8994-feff819cdc9f"}]
12 }
```

### 8.5.1.1. Ejemplos de peticiones HTTP

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

```
1 curl -X POST \
2   -H 'accept: application/json' \
3   -H 'X-VirtShell-Authorization: UserId:Signature' \
4   -H "Content-Type: multipart/form-data" \
```



```
5  -d '{
6      "name": "database_team"
7  }' \
8  'http://<host>:<port>/api/virtshell/v1/groups'
```

Respuesta:

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

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

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

Respuesta:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4     "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
5     "name": "web_development_team",
6     "users": [
7         { "username": "user1", "id": "a146cae4-8c90-11e5-
8           8994-feff819cdc9f" },
```

```
8      {"username": "user2", "id": "a146d00c-8c90-11e5-  
      8994-feff819cdc9f"}  
9    ]  
10   "created": [{"at": "1447696674"}, {"by": "a379e8e6-8c8b-  
      11e5-8994-feff819cdc9f"}]  
11 }
```

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

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

Respuesta:

```
1 HTTP/1.1 200 OK  
2 Content-Type: application/json  
3 {  
4   "groups": [  
5     {  
6       "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",  
7       "name": "web_development_team",  
8       "users": [  
9         {"username": "user1", "id": "a146cae4-8c90-11e5-  
          8994-feff819cdc9f"},  
10        {"username": "user2", "id": "a146d00c-8c90-11e5-  
          8994-feff819cdc9f"}  
11      ],  
12      "created": [{"at": "1447696833"}, {"by": "d2372efa-  
          8c8b-11e5-8994-feff819cdc9f"}]
```

```
13     },
14     {
15         "uuid": "a379f19c-8c8b-11e5-8994-feff819cdc9f",
16         "name": "math_team",
17         "users": [
18             { "username": "user3", "id": "a146cae4-8c90-11e5-8994-feff819cdc9f" }
19         ],
20         "created": [{ "at": "1421431233", "by": "18489280-8c91-11e5-8994-feff819cdc9f" }]
21     },
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 }
```

#### 8.5.1.1.4. 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  ' ' '
4  ' ' 'json
5  { "delete": "success" }

```

### 8.5.2. Users

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

Cuadro 8.2: 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:

```

1  {
2  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",

```

```
3  "username": "virtshell",
4  "type": "system/regular",
5  "login": "user@mail.com",
6  "groups": [ ... list of users ...],
7  "created": {"at": timestamp, "by": user_uuid},
8  "modified": {"at": timestamp, "by": user_uuid}
9 }
```

Ejemplo:

```
1  {
2    "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3    "username": "virtshell",
4    "type": "system/regular",
5    "login": "user@mail.com",
6    "groups": [ {"name": "web_development_team"},
7                {"name": "production"}
8    ],
9    "created": {"at": "1429207233", "by": "92d30f0c-8c9c-11e5-
10               -8994-feff819cdc9f"},
11    "modified": {"at": "1529207233", "by": "92d31132-8c9c-
12               11e5-8994-feff819cdc9f"}
13 }
```

### 8.5.2.1. Ejemplos de peticiones HTTP

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

```
1 curl -X POST \
2     -H 'accept: application/json' \
3     -H 'X-VirtShell-Authorization: UserId:Signature' \
4     -H "Content-Type: multipart/form-data" \
5     -d {
```

```
6     "username": "virtshell",
7     "type": "system/regular",
8     "login": "user@mail.com",
9     "groups": [ {"name": "web_development_team"},
10                 {"name": "production"} ]
11 } \
12 'http://<host>:<port>/api/virtshell/v1/users '
```

Respuesta:

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

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

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

Respuesta:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4     "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
5     "username": "virtshell",
```

```
6  "type": "system/regular",
7  "login": "user@mail.com",
8  "groups": [ {"name": "web_development_team"},
9              {"name": "production"}
10 ],
11 "created": {"at": "1429207233", "by": "92d30f0c-8c9c-11e5-
12            -8994-feff819cdc9f"},
13 "modified": {"at": "1529207233", "by": "92d31132-8c9c-
14            11e5-8994-feff819cdc9f"}
15 }
```

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

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

Respuesta:

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

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

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

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

Respuesta:

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

### 8.5.3. Partitions

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

Cuadro 8.3: 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:

```

1 {
2   "uuid": string,
3   "name": string,
4   "description": string,
5   "hosts": [ ... list of hosts associated with the
               Partitions ... ],
6   "created": {"at": number, "by": string},
7   "modified": {"at": number, "by": string}
8 }
```

Ejemplo:

```

1 {
2   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3   "name": "development_co",
4   "description": "Collection of servers oriented to
                   development team in Colombia.",
5   "hosts": [ ... list of hosts associated with the
               partition ... ],
6   "created": {"at": "1429207233", "by": "92d30f0c-8c9c-11e5-
              -8994-feff819cdc9f"},
7   "modified": {"at": "1529207233", "by": "92d31132-8c9c-
              11e5-8994-feff819cdc9f"}
8 }
```

### 8.5.3.1. Ejemplos de peticiones HTTP

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

```
1 curl -sv -X POST \  
2   -H 'accept: application/json' \  
3   -H 'X-VirtShell-Authorization: UserId:Signature' \  
4   -d '{  
5       "name": "development_co",  
6       "description": "Collection of servers oriented to  
7           development team in colombia."  
8   }' \  
9   'http://localhost:8080/api/virtshell/v1/partitions'
```

Response:

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

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

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

Response:

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

```
4  "uuid": "efa1777c-cad7-11e5-9956-625662870761",
5  "name": "backend_development_04",
6  "description": "Servers for backend of the company",
7  "hosts": [ ... list of hosts associated with the
               section ...],
8  "created": {"at": "1429207233", "by": "1a900cdc-cad8-11e5-
              -9956-625662870761"},
9  "modified": {"at": "1529207233", "by": "2163b554-cad8-
               11e5-9956-625662870761"}
10 }
```

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

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

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "partitions": [
5     {
6       "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
7       "name": "development_co",
8       "description": "Collection of servers oriented to
                       development team in colombia.",
9       "hosts": [ ... list of hosts associated with the
                     section ...],
10      "created": {"at": "1429207233", "by": "92d30f0c-8c9c-
                  11e5-8994-feff819cdc9f"},
```

```

11     "modified": { "at": "1529207233", "by": "92d31132-8c9c
12         -11e5-8994-feff819cdc9f" }
13 },
14 {
15     "uuid": "efa1777c-cad7-11e5-9956-625662870761",
16     "name": "production_us_miami",
17     "description": "Collection of servers oriented to
18         production in us.",
19     "hosts": [ ... list of hosts associated with the
20         section ... ],
21     "created": { "at": "1429207233", "by": "1a900cdc-cad8-
22         11e5-9956-625662870761" },
23     "modified": { "at": "1529207233", "by": "2163b554-cad8
24         -11e5-9956-625662870761" }
25 }
26 ]
27 }

```

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

```

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

```

Response:

```

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

```

```
5 { "delete": "success" }
```

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

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

Response:

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

### 8.5.4. Hosts

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

Cuadro 8.4: 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:

```
1 {
2   "uuid": string,
3   "name": string,
4   "description": string,
5   "users": [ user_resource ],
6   "partition": string,
7   "created": { "at": number, "by": string },
8   "modified": { "at": number, "by": string }
9 }
```

Ejemplo:

```
1 {
2   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3   "name": "bigdata_test_01",
4   "description": "Collection of servers oriented to big
5                   data.",
6   "users": [ ... list of users allowed to use the
7               enviroment ... ],
8   "partition": "partition associated with the enviroment",
9   "created": { "at": "1429207233", "by": "92d30f0c-8c9c-11e5-
10               -8994-feff819cdc9f" },
11   "modified": { "at": "1529207233", "by": "92d31132-8c9c-
12               11e5-8994-feff819cdc9f" }
13 }
```

#### 8.5.4.1. Ejemplos de peticiones HTTP

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

```
1 curl -sv -X POST \
```

```

2  -H 'accept: application/json' \
3  -H 'X-VirtShell-Authorization: UserId:Signature' \
4  -d '{
5      "name": "bigdata_test_01",
6      "description": "Collection of servers oriented to
7                      big data.",
8      "users": [ ... list of users allowed to use the
9                      enviroment ... ],
10     "partition": "partition associated with the
11                     enviroment "
12     }' \
13     'http://localhost:8080/api/virtshell/v1/enviroments'

```

Response:

```

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

```

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

```

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

```

Response:

```

1  HTTP/1.1 200 OK
2  Content-Type: application/json
3  {
4      "uuid": "efa1777c-cad7-11e5-9956-625662870761",
5      "name": "backend_development",

```

```
6  "description": "All backend of the company",
7  "users": [ ... list of users allowed to use the
           enviroment ...],
8  "partition": "partition associated with the enviroment"
9  ,
10 "created": {"at": "1429207233", "by": "1a900cdc-cad8-11e5-
           -9956-625662870761"},
11 "modified": {"at": "1529207233", "by": "2163b554-cad8-
           11e5-9956-625662870761"}
12 }
```

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

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

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "enviroments": [
5     {
6       "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
7       "name": "bigdata_test_01",
8       "description": "Collection of servers oriented to
           big data.",
9       "users": [ ... list of users allowed to use the
           enviroment ...],
10      "partition": "partition associated with the
           enviroment",
```



```

11     "created": { "at": "1429207233", "by": "92d30f0c-8c9c-
12         11e5-8994-feff819cdc9f" },
13     "modified": { "at": "1529207233", "by": "92d31132-8c9c-
14         -11e5-8994-feff819cdc9f" }
15 },
16 {
17     "uuid": "efa1777c-cad7-11e5-9956-625662870761",
18     "name": "backend_development",
19     "description": "All backend of the company",
20     "users": [ ... list of users allowed to use the
21         enviroment ... ],
22     "partition": "partition associated with the
23         enviroment",
24     "created": { "at": "1429207233", "by": "1a900cdc-cad8-
25         11e5-9956-625662870761" },
26     "modified": { "at": "1529207233", "by": "2163b554-cad8
27         -11e5-9956-625662870761" }
28 }
29 ]
30 }

```

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

```

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

```

Response:

```

1 HTTP/1.1 200 OK

```

```
2 Content-Type: application/json
3 '''
4 '''json
5 { "delete": "success" }
```

### 8.5.5. Hosts

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

Cuadro 8.5: 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:

```
1 {
2   "uuid": string,
3   "name": string,
4   "os": string,
5   "memory": string,
6   "capacity": string,
7   "enabled": string,
8   "type": string,
9   "local_ipv4": string,
10  "local_ipv6": string,
11  "public_ipv4": string,
12  "public_ipv6": string,
```

```
13  "instances": [ instance_resource ],
14  "partition": string,
15  "created":["at": number, "by": number]
16 }
```

Ejemplo:

```
1  {
2    "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3    "name": "host-01-pdn",
4    "os": "Ubuntu_12.04_3.5.0-23.x86_64",
5    "memory": "16GB",
6    "capacity": "120GB",
7    "enabled": "true|false",
8    "type": "StorageOptimized|GeneralPurpose|HighPerformance",
9    "local_ipv4": "15.54.88.19",
10   "local_ipv6": "ff06:0:0:0:0:0:0:c3",
11   "public_ipv4": "10.54.88.19",
12   "public_ipv6": "yt06:0:0:0:0:0:0:c3",
13   "instances": [
14     ... instances resource is here
15   ],
16   "partition": "development_co",
17   "created":["at":"timestamp", "by":1234]
18 }
```

### 8.5.5.1. Ejemplos de peticiones HTTP

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

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

```
3   -H 'X-VirtShell-Authorization: UserId:Signature' \  
4   -d '{"name": "host-01-pdn",  
5       "os": "Ubuntu_12.04_3.5.0-23.x86_64",  
6       "memory": "16GB",  
7       "capacity": "120GB",  
8       "enabled": "true",  
9       "type": "GeneralPurpose",  
10      "local_ipv4": "15.54.88.19",  
11      "local_ipv6": "ff06:0:0:0:0:0:0:c3",  
12      "public_ipv4": "10.54.88.19",  
13      "public_ipv6": "yt06:0:0:0:0:0:0:c3",  
14      "partition": "development_co"}' \  
15   'http://localhost:8080/virtshell/api/v1/hosts'
```

Response:

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

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

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

Response:

```
1 HTTP/1.1 200 OK  
2 Content-Type: application/json  
3 {  
4   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",  
5   "name": "host-01-pdn",
```

```
6  "os": "Ubuntu_12.04_3.5.0-23.x86_64",
7  "memory": "16GB",
8  "capacity": "120GB",
9  "enabled": "true",
10 "type" : "StorageOptimized",
11 "local_ipv4": "15.54.88.19",
12 "local_ipv6": "ff06:0:0:0:0:0:0:c3",
13 "public_ipv4": "10.54.88.19",
14 "public_ipv6": "yt06:0:0:0:0:0:0:c3",
15 "instances": [
16     {
17         "name": "name1",
18         "id": "72C05559-0590-4DA6-BE56-28AB36CB669C"
19     },
20     {
21         "name": "name2",
22         "id": "17173587-C4E9-4369-9C43-FCBF5E075973"
23     }
24 ],
25 "partition": "development_co",
26 "created": ["at": "20130625105211", "by": 10]
27 }
```

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

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

Response:

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

```
3 {
4   "hosts": [
5     {
6       "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
7       "name": "host-01-pdn",
8       "os": "Ubuntu_12.04_3.5.0-23.x86_64",
9       "memory": "16GB",
10      "capacity": "120GB",
11      "enabled": "true",
12      "type": "StorageOptimized",
13      "local_ipv4": "15.54.88.19",
14      "local_ipv6": "ff06:0:0:0:0:0:0:c3",
15      "public_ipv4": "10.54.88.19",
16      "public_ipv6": "yt06:0:0:0:0:0:0:c3",
17      "instances": [
18        {
19          "name": "name1",
20          "id": "72C05559-0590-4DA6-BE56-28AB36CB669C"
21        },
22        {
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 }
```

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

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

Response:

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

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

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

Response:

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

### 8.5.6. Instances

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

Cuadro 8.6: 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:

```
1 {
2   "uuid": string,
3   "name": string,
4   "memory": numeric,
5   "cpus": numeric,
6   "hdsize": string,
7   "description": string,
8   "enviroment": string,
9   "operating_system": string,
10  "provisioner": string,
11  "host_type": string,
12  "ipv4": string,
13  "ipv6": string,
14  "driver": string,
15  "permissions": string,
16  "created": {"at": timestamp, "by": string},
17  "modified": {"at": timestamp, "by": string}
18 }
```

Ejemplo:

```
1 {
2   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3   "name": "transactional_log",
4   "memory": 1024,
```

```
5  "cpus": 2,  
6  "hdsiz": "2GB",  
7  "description": "Server transactional only for store  
    logs",  
8  "enviroment": "Enviroment name to which it belongs",  
9  "operating_system": "ubuntu_server_14.04.2_amd64",  
10 "provisioner": "all_backend",  
11 "host_type": "GeneralPurpose | ComputeOptimized |  
    MemoryOptimized | StorageOptimized",  
12 "ipv4": "172.16.56.104",  
13 "ipv6": "FE80:0000:0000:0000:0202:B3FF:FE1E:8329",  
14 "driver": "lxc | virtualbox | vmware | ec2 | kvm |  
    docker",  
15 "permissions": "xwxwxwx",  
16 "created": {"at": "1429207233", "by": "92d30f0c-8c9c-11e5  
    -8994-feff819cdc9f"},  
17 "modified": {"at": "1529207233", "by": "92d31132-8c9c-  
    11e5-8994-feff819cdc9f"}  
18 }
```

### 8.5.6.1. Ejemplos de peticiones HTTP

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

```
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       "hdsiz": "2GB",
```

```
8     "enviroment": "development_co",
9     "operating_system": "ubuntu_server_14.04.2_amd64"
10    ,
11    "description": "Server transactional only for
12    store logs",
13    "provisioner": "all_backend",
14    "host_type": "GeneralPurpose",
15    "driver": "lxc"
16  }' \
17  'http://localhost:8080/virtshell/api/v1/instances'
```

Response:

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

#### 8.5.6.1.2. 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/
4     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   "memory": 1024,
7   "cpus": 2,
```

```
8  "hdsiz": "2GB",
9  "enviroment": "development_co",
10 "operating_system": "ubuntu_server_14.04.2_amd64",
11 "description": "Server transactional only for store
    logs",
12 "provisioner": "all_backend",
13 "host_type": "GeneralPurpose",
14 "drive": "lxc",
15 "created": {"at": "1429207233", "by": "92d30f0c-8c9c-11e5
    -8994-feff819cdc9f"},
16 "modified": {"at": "1529207233", "by": "cf744732-8f12-
    11e5-8994-feff819cdc9f"}
17 }
```

#### 8.5.6.1.3. Obtener todos las instances - GET /api/virtshell/v1/instances

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

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "instances": [
5     {
6       "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
7       "name": "transactional_log",
8       "memory": 1024,
9       "cpus": 2,
10      "hdsiz": "2GB",
```

```
11     "enviroment": "development_co",
12     "operating_system": "ubuntu_server_14.04.2_amd64",
13     "description": "Server transactional only for store
14         logs",
15     "provisioner": "all_backend",
16     "host_type": "GeneralPurpose",
17     "drive": "lxc",
18     "permissions": "xwxwxwxr",
19     "created": {"at": "1429207233", "by": "92d30f0c-8c9c-
20         11e5-8994-feff819cdc9f"},
21     "modified": {"at": "1529207233", "by": "cf744732-8f12
22         -11e5-8994-feff819cdc9f"}
23 },
24 {
25     "uuid": "cf744476-8f12-11e5-8994-feff819cdc9f",
26     "name": "orders_colombia",
27     "memory": 2024,
28     "cpus": 2,
29     "hds size": "4GB",
30     "operating_system": "ubuntu_server_14.04.2_amd64",
31     "description": "Server transactional dedicated to
32         receive orders",
33     "enviroment": "development_mx",
34     "provisioner": "all_backend",
35     "host_type": "StorageOptimized",
36     "drive": "docker",
37     "permissions": "xwxwxwxr",
38     "created": {"at": "1429207233", "by": "92d30f0c-8c9c-
39         11e5-8994-feff819cdc9f"},
40     "modified": {"at": "1529207233", "by": "92d31132-8c9c-
41         -11e5-8994-feff819cdc9f"}
42 }
```

```

37 ]
38 }

```

#### 8.5.6.1.4. 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
2 Content-Type: application/json
3 ''
4 ''json
5 { "delete": "in progress" }

```

### 8.5.7. Tasks

Representan una tarea en VirtShell. Los métodos soportados son:

Cuadro 8.7: 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:

```
1 {
2   "uuid": string,
3   "description": string,
4   "status" : string,
5   "type": string,
6   "object_uuid": string,
7   "created":["at":"timestamp", "by":string],
8   "last_update": "timestamp",
9   "log": string
10 }
```

Ejemplo:

```
1 {
2   "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 }
```

### 8.5.7.1. Ejemplos de peticiones HTTP

#### 8.5.7.1.1. 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" }
```

#### 8.5.7.1.2. 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/
4     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
7                 -8994-feff819cdc9f" },
8     "last_update": "1429207435",
9     "log": "summary of the task"
10 }
```

#### 8.5.7.1.3. 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",
9       "created": {"at":"1454433171", "by":"cc7f8e2c-ccf4-
10                  11e5-9956-625662870761"},
11       "last_update": "1454436771",
12       "log": "summary of the task"
13     }
14   ]
15 }
```

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

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

Response:

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

```
4  "tasks": [  
5    {  
6      "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",  
7      "description": "clone virtual machine database_01",  
8      "status" : "in progress",  
9      "created": {"at": "1429207233", "by": "92d30f0c-8c9c-  
10         11e5-8994-feff819cdc9f"},  
11      "last_update": "1429207435",  
12      "log": "summary of the task"  
13    },  
14    {  
15      "uuid": "a62ad146-ccf4-11e5-9956-625662870761",  
16      "description": "create container webserver_09",  
17      "status" : "sucess",  
18      "created": {"at": "1454433171", "by": "cc7f8e2c-ccf4-  
19         11e5-9956-625662870761"},  
20      "last_update": "1454436771",  
21      "log": "summary of the task"  
22    }  
  ]  
}
```

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

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

Response:

```

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

### 8.5.8. Properties

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

Cuadro 8.8: Métodos HTTP para properties

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

Representación del recurso de un paquete:

```

1 {
2   "properties": [
3     {"name": "propertie_name1"},
4     {"name": "propertie_name2"}
5   ],
6   "hosts": [
7     {"name": "Host_", "range": "[1-3]"},
8     {"name": "database_001"}
9   ],
10  "tags": [
11    {"name": "db"},
12    {"name": "web"}
13  ]
}
```

```
14 }
```

Ejemplo:

```
1 {
2   "properties": [
3     { "name": "memory" },
4     { "name": "cpu" }
5   ],
6   "hosts": [
7     { "name": "Host_", "range": "[1-3]" }
8   ]
9 }
```

### 8.5.8.1. Ejemplos de peticiones HTTP

#### 8.5.8.1.1. Obtener una o mas propiedades de una unica instancia - 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' \
5   -d '{ "properties": [{ "name": "memory" }, { "name": "cpu"
6     } ] },
7     "hosts": [{ "name": "WebServer" } ] }' \
   'http://localhost:8080/api/virtshell/v1/properties'
```

Respuesta:

```
1 HTTP/1.1 202 OK
```

```
2 Content-Type: application/json
3 {
4   "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
5   "name": "Database001",
6   "memory": 1024
7 }
```

**8.5.8.1.2. Obtener una o mas propiedades de una o mas instancias por tag - 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' \
5   -d '{ "properties": [{"name": "memory"}, {"name": "cpu"
6     }],
7     "tag": [{"name": "web"}]}' \
   '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 }
```

**8.5.8.1.3. 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' \
5     -d '{ "properties": [{"name": "memory"}, {"name": "cpu"
6         {"name": "Database00", "range": "[1-3]"}]}', \
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": "Database001",
8     "memory": 4024,
9     "cpu": 2
10  },
11  {
12     "id": "591b3828-7aaf-4833-a94c-ad0df44d59a4",
13     "name": "Database002",
14     "memory": 4024,
15     "cpu": 1
16  },
17  {
18     "id": "f7c81039-5c88-423b-8b0d-c124483d586b",
19     "name": "Database003",
20     "memory": 4024,
21     "cpu": 3
22  }
23 ]
24 }

```

### 8.5.9. Provisioners

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

Cuadro 8.9: 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:

```
1 {
2   "uuid": string,
3   "name": string,
4   "description": string,
5   "version": string,
6   "repository": string,
7   "executor": string,
8   "tag": string,
9   "permissions": string,
10  "depends": [ ... list of dependencies necessary for the
        builder ... ],
11  "created": {"at":timestamp, "by":string},
12  "modified": {"at":timestamp, "by":string}
13 }
```

Ejemplo:

```
1 {
2   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
3   "name": "backend-services-provisioner",
4   "description": "Installs/Configures a backend server",
5   "version": "1.5.8",
6   "repository": "https://github.com/janutechnology/
        VirtShell_Provisioners_Examples.git",
7   "executor": "run1.sh",
8   "tag": "backend",
9   "permissions": "xwxrwxrwx",
10  "depends": [ ... list of dependencies necessary for the
        builder ... ],
11  "created": {"at":"1429207233", "by":"92d30f0c-8c9c-11e5
        -8994-feff819cdc9f"},

```



```
12   "modified": { "at": "1529207233", "by": "92d31132-8c9c-  
13     11e5-8994-feff819cdc9f" }  
13 }
```

### 8.5.9.1. Ejemplos de peticiones HTTP

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

```
1 curl -sv -X POST \  
2   -H 'accept: application/json' \  
3   -H 'X-VirtShell-Authorization: UserId:Signature' \  
4   -d '{ "name": "backend-services-provisioner",  
5       "repository": "https://github.com/janutechnology/  
6         VirtShell_Provisioners_Examples.git",  
7       "executor": "run1.sh",  
8       "tag": "backend",  
9       "depends": [  
10         { "provisioner_name": "db-users", "version": "  
11           2.0.0" },  
12         { "provisioner_name": "db-transactional" }  
13       ]  
14     }', \  
15   'http://localhost:8080/virtshell/api/v1/provisioners '
```

Response:

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

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

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

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4     "name": "backend-services-provisioner",
5     "repository": "https://github.com/janutechnology/
    VirtShell_Provisioners_Examples.git",
6     "executor": "run1.sh",
7     "tag": "backend",
8     "depends": [
9         {"provisioner_name": "db-users", "version": "
    2.0.0"},
10        {"provisioner_name": "db-transactional"}
11    ],
12    "created": {"at": "1429207233", "by": "420aa2c4-8d96-
    11e5-8994-feff819cdc9f"},
13    "modified": {"at": "1529207233", "by": "92d31132-8c9c-
    11e5-8994-feff819cdc9f"}
14 }
```

#### 8.5.9.1.3. Obtener todos los provisioners - GET /api/virtshell/v1/provisioners

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

```
3      'http://localhost:8080/api/virtshell/v1/provisioners',
```

Response:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "provisioners": [
5     {
6       "name": "backend-services-provisioner",
7       "repository": "https://github.com/janutechnology/
8         VirtShell_Provisioners_Examples.git",
9       "executor": "run1.sh",
10      "tag": "backend",
11      "permissions": "xwrwxrwxr",
12      "depends": [
13        {"provisioner_name": "db-users", "version": "
14          2.0.0"},
15        {"provisioner_name": "db-transactional"}
16      ]
17    },
18    {
19      "name": "db-transactional",
20      "repository": "https://github.com/janutechnology/
21        VirtShell_Provisioners_Examples.git",
22      "executor": "run_db.sh",
23      "tag": "db",
24      "permissions": "xwrwxrwxr"
25    }
26  ]
27 }
```

**8.5.9.1.4. 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" }
```

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

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

Response:

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

### 8.5.10. Images

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

Cuadro 8.10: 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:

```
1 {  
2   "id": string,  
3   "name": string,  
4   "type": string,  
5   "os": string,  
6   "release": string,  
7   "version": string,  
8   "variant": string,  
9   "arch": string,  
10  "timezone": "America/Bogota",  
11  "key": string,  
12  "preseed_url": url,  
13  "download_url": url,  
14  "permissions" : string,  
15  "created":["at": timestamp,"by": string],  
16  "details": string  
17 }
```

Ejemplo:

```
1 {
2   "id": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
3   "name": "ubuntu_server_14.04.2_amd64",
4   "type": "iso",
5   "os": "ubuntu",
6   "release": "trusty",
7   "version": "14.04.2",
8   "variant": "server",
9   "arch": "amd64",
10  "timezone": "America/Bogota",
11  "preseed_url": "https://<host>:<port>/api/virtshell/v1/
    files/seeds/seed_ubuntu14-04.txt",
12  "permissions" : "rwxrw----",
13  "created":["at":"20150625105211","by":10]
14 }
```

### 8.5.10.1. Ejemplos de peticiones HTTP

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

```
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": "ubuntu_server_14.04.2_amd64",
6       "type": "iso",
7       "os": "ubuntu",
8       "release": "trusty",
9       "version": "14.04.2",
10      "variant": "server",
11      "arch": "amd64",
```

```
12     "timezone": "America/Bogota",
13     "key": "/home/callanor/.ssh/id_rsa.pub",
14     "permissions" : "rwxrwxr--",
15     "preseed_url": "https://<host>:<port>/api/virtshell/
      v1/files/seeds/seed_ubuntu14-04.txt"}', \
16 'http://localhost:8080/api/virtshell/v1/image'
```

Respuesta:

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

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

```
1 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  "release": "trusty",
9  "version": "14.04.2",
10 "variant": "server",
11 "arch": "amd64",
12 "timezone": "America/Bogota",
13 "preseed_url": "https://<host>:<port>/api/virtshell/v1/
    files/seeds/seed_ubuntu_14_04.txt",
14 "permissions" : "rwxrwxrwx",
15 "created":["at":"20130625105211","by":10]
16 }
```

#### 8.5.10.1.3. 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:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4   "images": [
5     {
6       "id": "b180ef2c-e798-4a8f-b23f-aaac2fb8f7e8",
7       "name": "ubuntu_server_14.04.2_amd64",
8       "type": "iso",
```



```
9      "os": "ubuntu",
10      "release": "trusty",
11      "version": "14.04.2",
12      "variant": "server",
13      "arch": "amd64",
14      "timezone": "America/Bogota",
15      "preseed_file": "https://<host>:<port>/api/
      virtshell/v1/files/seeds/seed_file.txt",
16      "permissions" : "rwxrw----",
17      "created": ["at": "20130625105211", "by": 10]
18  },
19  {
20      "id": "ca326181-bc84-4edb-bfc5-843037e7195e",
21      "name": "centos_server",
22      "type": "container",
23      "os": "centos",
24      "version": "7",
25      "arch": "x86_64",
26      "download_url": "https://<host>:<port>/api/
      virtshell/v1/files/images/3514296#file-lxc-centos
      ",
27      "permissions" : "rwxrwxr--",
28      "created": ["at": "20140625105211", "by": 12]
29  }
30 ]
31 }
```

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

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

```

3  -H 'X-VirtShell-Authorization: UserId:Signature' \
4  'http://<host>:<port>/api/virtshell/v1/images/
    ubuntu_server_14.04.2_amd64'

```

Respuesta:

```

1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 ' ' '
4 ' ' 'json
5 { "delete": "success" }

```

### 8.5.11. Packages

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

Cuadro 8.11: 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:

```

1 {
2   "packages": [

```

```
3     {"name": "package_name1"},
4     {"name": "package_name2"}
5 ],
6 "hosts": [
7     {"name": "Host_", "range": "[1-3]"},
8     {"name": "database_001"}
9 ],
10 "tags": [
11     {"name": "db"},
12     {"name": "web"}
13 ]
14 }
```

Ejemplo:

```
1 {
2   "packages": [
3     {"name": "git"},
4     {"name": "nginx"}
5   ],
6   "hosts": [
7     {"name": "Host_", "range": "[1-3]"}
8   ]
9 }
```

#### 8.5.11.1. Ejemplos de peticiones HTTP

##### 8.5.11.1.1. Instalar uno o mas paquetes - POST /api/virtshell/v1/install\_packages

```
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 '{ "packages": [{"name": "git"}, {"name": "nginx"}],  
6     "hosts": [{"name": "WebServer_", "range": "[1-3]"  
7         }]}', \  
    'http://localhost:8080/api/virtshell/v1/  
    install_packages '
```

Respuesta:

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

#### 8.5.11.1.2. Actualizar uno o mas paquetes - POST /api/virtshell/v1/upgrade\_packages

```
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 '{ "packages": [{"name": "git"}, {"name": "nginx"},  
6     {"name": "mc"}],  
7     "hosts": [{"name": "WebServer_", "range": "[1-3]"  
        }]}', \  
    'http://localhost:8080/api/virtshell/v1/  
    upgrade_packages '
```

Respuesta:

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

#### 8.5.11.1.3. Remover uno o mas paquetes - POST /api/virtshell/v1/remove\_packages

```
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 '{ "packages": [{"name": "apache2"}],
6         "hosts": [{"name": "WebServer_", "range": "[1-3]"
7         }]}', \
8   'http://localhost:8080/api/virtshell/v1/
9   remove_packages '
```

Respuesta:

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

### 8.5.12. Files

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

Cuadro 8.12: 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:

```
1 {  
2   "uuid": string,  
3   "name": string,  
4   "folder_name" : string,  
5   "download_url": url,  
6   "permissions": string,  
7   "created":["at":"timestamp", "by": string]  
8 }
```

Ejemplo:

```
1 {  
2   "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",  
3   "name": "ubuntu_seed_14-04.tex",  
4   "folder_name" : "ubuntu_seeds",  
5   "download_url": "https://<host>:<port>/api/virtshell/v1  
6     /files/ubuntu_seeds/ubuntu_seed_14-04.tex",  
7   "permissions": "rwxrwxrwx",  
8   "created": ["at":"20130625105211", "by":10]  
9 }
```

### 8.5.12.1. Ejemplos de peticiones HTTP

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

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

Respuesta:

```
1 HTTP/1.1 200 OK  
2 Content-Type: application/json  
3 {  
4   "create": "success",  
5   "location": "http://<host>:<port>/api/virtshell/v1/  
6       files/ubuntu_seeds/seed_file_ubuntu-14_04.txt"  
7 }
```

#### 8.5.12.1.2. 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.

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

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

Respuesta:

```
1 HTTP/1.1 200 OK
2 Content-Type: application/json
3 {
4     "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
5     "name": "file_name.extension",
6     "folder_name" : "folder_name",
7     "download_url": "http://<host>:<port>/api/virtshell/v1/
      files/ubuntu_seeds/seed_file_ubuntu-14_04.txt",
8     "permissions": "rwxrwxr-x",
9     "created":["at":"timestamp", "by":user_id]
10 }
```

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

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



Respuesta:

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

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

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

Respuesta:

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

## 8.6. API Calls

### 8.6.1. Start Instance

Permite iniciar una instancia.

#### 8.6.1.0.5. Iniciar una instance -

**POST** /virtshell/api/v1/instances/start\_instance/:id

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

Response:

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

#### 8.6.2. Stop Instance

Permite detener una instancia.

#### 8.6.2.0.6. Detener una instancia -

**POST** /virtshell/api/v1/instances/stop\_instance/:id

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

Response:

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

### 8.6.3. Restart Instance

Permite reiniciar una instancia.

#### 8.6.3.0.7. Reiniciar una instancia -

**POST** /virtshell/api/v1/instances/restart\_instance/:id

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

Response:

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

### 8.6.4. Clone Instance

Permite clonar una instancia.

#### 8.6.4.0.8. Clonar una instancia -

**POST** /virtshell/api/v1/instances/clone\_instance/:id

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

Response:

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

### 8.6.5. Execute command

Permite ejecutar un comando en una o mas instancias.

Representación del recurso para ejecutar un comando:

```
1 {
2   "instances": [ ... list of instances names, patterns(*|
3     [numeric:numeric]) or tags ...],
4   "command": string,
5   "created": {"at": timestamp, "by": string}
6 }
```

Ejemplo:

```
1 {
2   "instances": [
3     {"name": "database_server_01"},
4     {"name": "transactional_server_co"},
5     {"pattern": "web_server*"},
6     {"pattern": "grid_[1:5]"},
7     {"tag": "web"}
8   ],
9   "command": "apt-get upgrade",
10  "created": {"at": timestamp, "by": string}
11 }
```

**8.6.5.0.9. Ejecutar un comando en una o mas instancias -**  
**POST /virtshell/api/v1/instances/execute\_command/**

```
1 curl -sv -X POST \  
2   -H 'accept: application/json' \  
3   -H 'X-VirtShell-Authorization: UserId:Signature' \  
4   -d '{ "instances": [  
5       {"name": "database\_server\_01"},  
6       {"name": "transactional\_server\_co"},  
7       {"pattern": "web\_server*"},  
8       {"pattern": "grid\_server\_ [1:5]"},  
9       {"tag": "web"}  
10      ],  
11      "command": "apt-get upgrade" }' \  
12 'http://localhost:8080/virtshell/api/v1/instances/  
   execute\_command/'
```

Response:

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

### 8.6.6. Copy files

Permite ejecutar copiar uno archivo en una o mas instancias.

Representación del recurso para ejecutar un comando:

```
1 {  
2   "path": string,  
3   "destination": string,  
4   "instances": [ ... list of instances names, patterns(*|  
5       [numeric:numeric]) or tags ... ],  
6   "created": { "at": timestamp, "by": string }  
}
```

Ejemplo:

```
1 {
2   "path": "https://<host>:<port>/api/virtshell/v1/files/
      database_servers/msql/my.cnf",
3   "destination": "$MYSQL_HOME/my.cnf"
4   "instances": [
5     {"name": "database\_server\_01"},
6     {"name": "web\_server*"},
7     {"name": "grid\_ [1:5] "},
8     {"name": "transactional\_server\_co"},
9     {"tag": "web"}
10  ]
11 }
```

#### 8.6.6.0.10. Copiar un archivo en una o mas instancias - POST /virtshell/api/v1/instances/copy\_files/

```
1 curl -sv -X POST \
2   -H 'accept: application/json' \
3   -H 'X-VirtShell-Authorization: UserId:Signature' \
4   -d '{ "path": "https://<host>:<port>/api/virtshell/v1/
      files/database_servers/msql/my.cnf",
5       "destination": "$MYSQL_HOME/my.cnf"
6       "instances": [
7         {"name": "database\_server\_01"},
8         {"name": "web\_server*"},
9         {"name": "grid\_ [1:5] "},
10        {"name": "transactional\_server\_co"},
11        {"tag": "web"}
12      ] }' \
13   'http://localhost:8080/virtshell/api/v1/instances/copy\_
      _files/'
```

Response:

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

---

## CAPÍTULO 9

---

### Experiencia y Evaluación

---

...



---

## 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 eligió 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 automáticamente las instancias en función de políticas 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 específico.

---

---

## Bibliografía

---

---

---

## Bibliografía

---

- [1] ACM Symposium on Cloud Computing 2015, <http://acmsocc.github.io/2015/>, 2015.
- [2] Andrew S. Tanenbaum, 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 Compose, <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] HMAC: Keyed-Hashing for Message Authentication, <https://www.ietf.org/rfc/rfc2104.txt>, 1997
  - [18] Linux Containers, <https://linuxcontainers.org>, 2016
  - [19] Docker, <https://docker.com>, 2016