VirtShell
Framework para aprovisionamiento de soluciones virtuales

Carlos Alberto Llano R.
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle
Cali, Valle del Cauca
Email: carlos_llano@hotmail.com

John Alexander Sanabria
Escuela de Ingeniería de Sistemas y Computación
Universidad del Valle
Cali, Valle del Cauca
Email: john.sanabria@correounivalle.edu.co

***Resumen*—The abstract goes here.**

I. Introducción

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

Las herramientas de aprovisionamiento automático de infraestructura son el eje central de estos equipos ya que es a través de ellas que el personal de desarrollo y operaciones son capaces de hablar un mismo lenguaje y establecer los requerimientos y necesidades a satisfacer. Sin embargo, las herramientas actuales de aprovisionamiento adolecen de servicios que faciliten la especificación de infraestructura a través de un API [2] estandarizado que posibilite la orquestación del despliegue de infraestructura a través de Internet.

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

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

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

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

En la ultima sección se muestra la documentación del API de VirtShell. Se indican los recursos y los métodos HTTP con que cuenta cada módulo que permiten interactuar con el API.

II. Arquitectura

VirtShell Framework es concebido como una plataforma que proporciona herramientas para la automatización y gestión de infraestructura, facilitando tareas como la creación, despliegue, mantenimiento y monitoreo tanto de recursos virtuales como físicos vía web. Así mismo, es pensada para

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

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

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

que cualquier desarrollo de software con acceso a Internet (sitio web, aplicación móvil, etc.) pueda interactuar con la infraestructura virtual tan solo consumiendo un API de Internet.

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

En la búsqueda del adecuado estilo arquitectural para el API de VirtShell, se evaluaron dos estilos de servicios web: el estilo *Remote Procedure Call* (RPC) y el estilo arquitectural REST (*Representational State Transfer*) [17]. La evaluación dio como resultado que el estilo arquitectural que mejor se acomodaba a los requisitos planteados era el estilo arquitectural REST, debido a que es un estilo nativo del Web, lo que hace que la información disponible esta regida por las mismas normas que rigen los sitios web, ademas, el estilo ofrece mejor escalabilidad, acoplamiento y rendimiento. Un detallado comparativo entre los dos estilos se encuentra en [23].

II-A. Características

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

- *Programable* VirtShell esta orientado a realizar el aprovisionamiento de sus instancias principalmente por medio de scripts escritos en shell, permitiendo aprovechar todas las estructuras y utilidades del lenguaje de programación. Sin embargo, el lenguaje de shell no es de uso obligatorio, el metodo de aprovisionamiento puede ser el de la preferencia del usuario.
- *Repetible* VirtShell ofrece herramientas para que los scritps de aprovisionamiento sean configurables y puedan ser ejecutados varias veces en diferentes ambientes de desarrollo o producción.
- *Modular* VirtShell es un framework organizado de forma modular. Los módulos se encuentran agrupados en categorías que ofrecen las herramientas necesarias para la administración y aprovisionamiento de múltiples recursos virtuales. De igual manera y dada las características de REST, los módulos están diseñados para que se puedan dividir en diferentes servidores obteniendo "micro APIs", lo que permite dividir los procesos y atender diferentes tipos de operaciones del API.
- *Seguro* VirtShell provee varias capacidades y servicios para aumentar la privacidad y el control de acceso a los diferentes recursos. Los servicios de seguridad permiten crear redes y controlar el acceso a las instancias creadas, así como definir y administrar políticas de acceso a usuarios y permisos sobre cualquier recurso del sistema como por ejemplo scripts de creación y aprovisionamiento.
- *Extensible* Al ser VirtShell de código abierto, el API puede modificarse, crecer fácilmente y versionarse de diferentes maneras. Adicionalmente, VirtShell fue diseñado con la idea de cargar código dinámicamente, permitiendo extender el comportamiento del framework agregando plugins en tiempo de ejecución. Asi mismo, VirtShell permite extender el comportamiento del shell desplegando comandos propios que proporcionan ahorro en tiempo y en complejidad.
- *Inyección de dependencias virtuales* VirtShell adopta la idea del patrón de Inyección de Dependencias [4] [18] para conseguir scripts de aprovisionamiento mas desacoplados. De esta manera facilita la configuración de las dependencias que tiene un recurso virtual de otras máquinas virtuales. Para ello, el framework permite declarar el listado de dependencias de recursos virtuales que tiene un script de aprovisionamiento encargándose del correcto acople entre los diferentes recursos virtuales.
- *Interoperable* Al seguir el estilo arquitectónico REST y contar con la documentación detallada sobre cada uno de los recursos y urls que expone el API de VirtShell, 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.

III. 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 1.

Las siguientes secciones detallan los módulos disponibles para cada característica.

III-A. Security

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

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

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

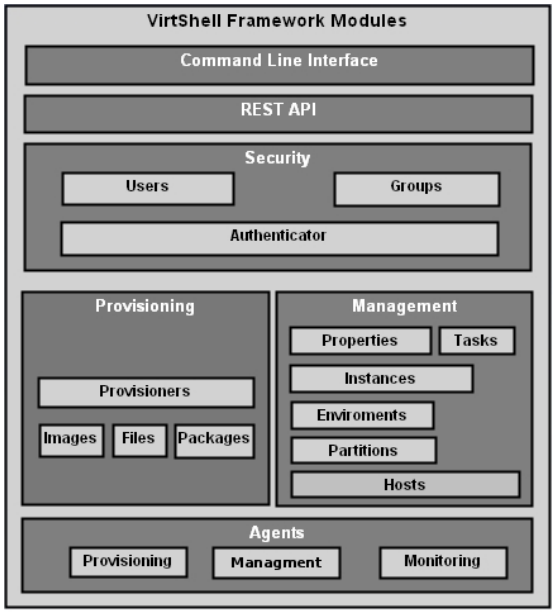

Figura 1: Visión general del framework de VirtShell

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

III-B. Managment

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

El módulo de **anfitriónes** lleva registro de nodos físicos, servidores o máquinas virtuales, que se encuentren conectados a la red y que permitan albergar instancias virtuales. Los anfitriónes son clasificados de acuerdo a diferentes combinaciones de CPU, memoria, almacenamiento y capacidad de trabajo en red, dando flexibilidad para elegir la opcion mas adecuada para las necesidades de las aplicaciones destino. En otras palabras el tipo de anfitrión determina el hardware del nodo físico que sera usado por los recursos virtuales.

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

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

El módulo de **instancias** es un recurso virtual o máquina virtual o contenedor con parámetros y capacidades definidas en la cual se puede instalar el software deseado. Un usuario puede crear, aprovisionar, actualizar y finalizar instancias en VirtShell tanto como necesite dando la sensación de elasticidad [5] de la red.

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

Finalmente, el módulo de **tareas** da la información y estado sobre las diferentes tareas o trabajos ejecutados en el sistema. Debido a que el medio para interactuar con los módulos de VirtShell es a través de un API REST, una petición de creación de un nuevo recurso virtual, puede ser largo si el aprovisionamiento involucra varias maquinas. Para evitar, tener una petición esperando respuesta, VirtShell crea una tarea que sera ejecutada de manera asincrónica, dando como respuesta, un identificador de la tarea, para que esta pueda ser consultada posteriormente y conocer el estado de la petición.

III-C. Provisioning

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

El módulo de **aprovisionadores** define un marco de aprovisionamiento de un recurso virtual y contiene las configuraciones necesarias para apoyar ese marco. La configuración fundamental de un aprovisionador, es la ruta del repositorio git [6], donde se encuentran los scripts y archivos necesarios para realizar el aprovisionamiento. Del mismo modo la forma de ejecutar los scripts, hace parte de la configuración básica. Para ser mas ligero a VirtShell, los scripts de aprovisionamiento, deben estar registrados en un repositorio git público. VirtShell se encarga de descargar el repositorio y de ejecutar los scripts de aprovisionamiento, de acuerdo a la configuración especificada.

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

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

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

VirtShell utiliza como lenguaje de referencia, para la creación de scripts de aprovisionamiento, el lenguaje shell. Este cuenta con los recursos suficientes para interactuar con los diferentes sistemas operativos de las instancias. Sin embargo estos comandos se pueden extender usando el paquete de comandos propios de VirtShell, los cuales pueden abstraer muchas de las operaciones del shell. Esto facilita la escritura de los mismos o permite hacerlos independientes del sistema operativo en el cual van a ejecutarse.

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

Las imágenes que se manejan en VirtShell son de dos tipos: ISO [7] y para contenedores. Las de tipo ISO, se encuentran guardadas en el repositorio de VirtShell, y su uso se enfoca a maquinas virtuales que interactuan con hypervisors. Las imágenes de tipo contenedor, se emplean para tecnologías de visualización de sistema operativo como LXC [8] [20] y Docker [9] [22]. Estas son descargadas automáticamente de los repositorios de dominio público de los diferentes proveedores.

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

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

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

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

III-D. Agents

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

IV. API

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

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

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

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

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

IV-A. 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.

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

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

[9]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.

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).

IV-B. Groups

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

Cuadro I: Métodos HTTP para groups

Método	Solicitud	Descripción
GET	/groups/:name	Gets one group by ID.
GET	/groups	Gets the list of groups.
POST	/groups/	creates a new group.
DELETE	/groups/:name	Deletes an existing group.

IV-C. Users

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

Cuadro II: Métodos HTTP para users

Método	Solicitud	Descripción
GET	/users/:name	Gets one user by ID.
POST	/users/	creates a new user.
GET	/users	Gets the list of users.
DELETE	/users/:name	Deletes an existing user.
PUT	/users/:name	Updates an existing user.

IV-D. Images

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

Cuadro III: Métodos HTTP para images

Método	Solicitud	Descripción
GET	/images/:name	Gets one image by name.
GET	/images	Retrieves the list of images.
POST	/images/	Inserts a new image.
DELETE	/images/:name	Deletes an existing image.

IV-E. Tasks

Representan una tarea en VirtShell. Los métodos soportados son:

Cuadro IV: Métodos HTTP para tasks

Método	Solicitud	Descripción
GET	/tasks/:id	Gets one task by ID.
GET	/tasks	Gets the list of tasks.
GET	/tasks/status	Gets all task by status.
POST	/tasks/	Creates a new task
PUT	/tasks/:id	Updates an existing task.

IV-F. Partitions

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

Cuadro V: Métodos HTTP para partitions

Método	Solicitud	Descripción
GET	/partitions/:name	Gets one partition.
GET	/partitions	Gets the list of partitions.
POST	/partitions/	Inserts a new partition.
DELETE	/partitions/:name	Deletes apartition.
PUT	/partitions/:name/ host/:hostname	Add a host to partition.

IV-G. Hosts

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

Cuadro VI: Métodos HTTP para hosts

Método	Solicitud	Descripción
GET	/hosts/:name	Gets one host by name.
GET	/hosts	Gets the list of hosts.
POST	/hosts/	Inserts a new host.
DELETE	/hosts/:name	Deletes an existing host.
PUT	/hosts/:name	Updates an existing host.

IV-H. Instances

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

Cuadro VII: Métodos HTTP para instances

Método	Solicitud	Descripción
GET	/provisioners/:name	Gets one provisioner by ID.
GET	/provisioners	Gets the list of provisioners.
POST	/provisioners/	Creates a new provisioner.
DELETE	/provisioners/:name	Deletes an existing host.

V. Conclusion

The conclusion goes here.

Acknowledgment

The authors would like to thank...

Referencias

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] LXC Templates Documentation, https://help.ubuntu.com/lts/serverguide/lxc.html, 2016.
[22] Docker, https://docker.com, 2016.
[23] REST - An Alternative to RPC for Web Services Architecture, http://web.uvic.ca/ērikj/seng422/resources/rest_paper.pdf, 2009.