Universidad del Valle

Escuela de IngenierÃıa de Sistemas y ComputaciÃşn

# VirtShell - Framework para aprovisionamiento de soluciones virtuales

Tesis presentada por Carlos Alberto Llano RodrÃıguez
para obtener el grado de MaestrÃıa en IngenierÃıa con Ãľnfasis
en IngenierÃıa de Sistemas

2015

Facultad de IngenierÃıa

# 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 compaÃśeros 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

# Índice de figuras

# Índice de cuadros

# CAPÍTULO 1

# Introducción

# CAPÍTULO 2

## Aprovisionamiento de recursos virtuales

La computación en la nube ha sido un punto importante de investigación en la industria recientemente. Esta puede ser descrita como una nueva clase de computación en la cual dinámicos y escalables recursos pueden ser provistos sobre internet. Para los usuarios esto es transparente y ellos solo pagan lo que usan de acuerdo a niveles de servicio establecidos con los proveedores de nubes.

Una de las principales características de la computación en la nube es la virtualización, la cual consiste en crear una versión virtual de un recurso tecnologico en lugar de usar una versión física. La virtualización se puede aplicar a computadoras, sistemas operativos, dispositivos de almacenamiento de información, aplicaciones o redes permitiendo que las empresas ejecuten más de un sistema virtual, además de múltipls sistemas operativos y aplicaciones, en un único servidor, de esta manera se logra economía de escala y una mayor eficiencia.

En la actualidad predominan dos tecnicas de virtualizacion, la primera tecnica se denomina virtualización de hardware y consiste en crear un hardware sintético el cual usan las maquinas virtuales como propio, la idea es virtualizar el sistema opera-

tivo completo el cual se ejecuta sobre un software llamado el hipervisor, su función es interactuar directamente con la CPU en el servidor físico, ofreciendo a cada uno de los servidores virtuales una total autonomía e independencia. Incluso pueden coexistir en una misma máquina distintos servidores virtuales funcionando con distintos sistemas operativos. Esta tecnica es la mas desarrollada y hay diferentes clases que cada fabricante ha ido desarrollando y adaptando, como por ejemplo Xen, KVM, VMWare y VirtualBox.

La segunda tecnica es conocida como virtualización del sistema operativo. En esta técnica lo que se virtualiza es el sistema operativo completo el cual corre directamente virtual sobre la máquina física. En esta técnica las maquinas virtuales son llamadas contenedores, los cuales acceden por igual a todos los recursos del sistema. La ventaja es a su vez una desventaja: Todas las maquinas virtuales usan el mismo Kernel que el sistema operativo lo que reduce mucho 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 mezclar Windows-Linux-Etc. Este sistema, también se acerca mucho a lo que seria una virtualización nativa.

Sin importar la tecnica de virtualización que se use, la instalación de una maquina virtual (o de un contenedor) requiere normalmente de la generación e instalación de una imagen y la instalación y configuración de paquetes de software. Estas tareas generalmente son realizadas por técnicos de los proveedores de la nube. Cuando un usuario de la nube solicita un nuevo servicio o mas capacidad de computo, el administrador selecciona la apropiada imagen para clonar e instalar en los nodos de la nube. Si no hay una imagen apropiada para los requerimientos del cliente se crea y configura una nueva que cumpla con la solicitud. Esta creación de una nueva imagen puede ser realizada modificando la imagen mas cercada de las ya existentes. En el momento de la creación optima de la imagen un administrador puede tener dificultades y preguntás como, cual es la mejor configuración?, cuales paquetes y sus dependencias deberían ser instaladas? y como encontrar una imagen que mejor llene las expectativas?.

Es por esta razón que los proveedores de la nube desean cada vez mas automatizar y simplificar este proceso porque la dependencia entre paquetes de software y la dificultad de mantenimiento agrega tiempo a la creación de las maquinas virtuales. En otras palabras los proveedores de nube quieren dar mas flexibilidad y agilidad a la hora de satisfacer los requerimientos de los usuarios finales.

Existen varias soluciones que permiten la interacción con los diferentes ambientes de virtualización. Estas soluciones usan diferentes enfoques para realizar despliegues de software en las maquinas virtuales, que dan un rápido, controlado y automático despliegue de software, en todas las maquinas de una red físicas o virtualizadas, permitiendo mejorar los tiempos de instalación de nuevas funcionalidades de forma confiable y segura de la misma forma que ayudan a disminuir el tiempo y el costo de los despliegues de aplicaciones y servicios. Sin embargo no todas las soluciones son de codigo abierto, algunas son de desarrollo propietario, en donde solo ofrecen el API al público pero no el código de la solución como tal y manejan sus propias herramientas de virtualización.

## 2.1. Alternativas de despliegue actuales

En esta sección, se describirán las herramientas mas significativas que existen indicando las ventajas y desventajas de cada una.

### 2.1.1. Nixes

Nixes es una herramienta usada para instalar, mantener, controlar y monitorear aplicaciones en PlanetLab [1]. Nixes consiste de un conjunto de scripts bash, un archivo de configuracion, y un respositorio web, y puede automaticamente instalar, actualizar y resolver dependencias solo de paquetes RPM.

Para sistemas de pequeña escala, Nixes es fácil de usar: los usuarios simplement e crean el archivo de configuración para cada aplicación y modifican los scripts a desplegar en los nodos. Pero para grandes y complejos sistemas, Nixes no es efectivo, porque el no provee un mecanismo automático de flujo de trabajo.

### 2.1.2. SmartFrog

SmartFrog (SF) es un framework para servicios de configuración, descripción, despliegue y administración del ciclo de vida. 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 SF, administra el ciclo de vida atraves de cinco estados: instalado, iniciado, terminado y fallido. Esto permite al motor del SmartFrog detectar fallas y reiniciar automáticamente re-despliegues de los componentes [2].
SmartFrog es desarrollado y mantenido por un equipo de investigación en los laboratorios de Hewlett-Packard en Bristol, Inglaterra, asi como por el laboratorio Europeo de Hewlett-Packard y adicional con contribuciones de otros usuarios de SmartFrog y desarrolladores externos a HP. Se utiliza en la investigación de HP especificamente en la automatización de la infraestructura y automatización de servicios, además de ser utilizado en determinados productos de HP.

### 2.1.3. Radia

Herramienta de administración de cambios que utiliza un enfoque basado en modelos [3]. Para cada dispositivo administrado, el administrador define un estado deseado, el cual es mantenido como un modelo en un repositorio central. Nixes, usa seis maquinascalculosmodelos: paquete (configuración, instalación, entradas de registro, binarios, entre otras); mejores prácticas; dependencias de software (relaciones con otros componentes de software, sistemas operativos y hardware); infraestructura (servidores, almacenamiento y elementos de red); inventarios de software (software instalado actualmente) e interoperabilidad entre modelos de servicios administrados.

### 2.1.4. Cobbler

Cobbler es una plataforma que busca el rápido despliegue de servidores y en general computadores en una infra-estructura de red, 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 al igual que SmartFrog es capaz de instalar máquinas físicas y máquinas virtuales. Cobbler, es una pequeña y ligera aplicación, que es extremadamente fácil de usar para pequeños o muy grandes despliegues. [**?** ]

### 2.1.5. Amazon EC2

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

### 2.1.6. HP Utility Data Center

HP Utility Data Cente (UDC) es un producto comercial, que se centra en la administración automatizada de servidores de red, usando el concepto de ”infraestructura programable ”. Los elementos de hardware, como nodos de servidores, switches, firewalls y elementos de almacenamiento, son cableados en una infraestructura de configuración. El software de administración UDC permite configurar combinaciones de estos componentes en servidores virtuales usando cableados virtuales. [**?** ]

### 2.1.7. Oracle VM Templates

Oracle VM Templates, es un producto comercial de la empresa Oracle, cuyo objetivo es realizar despliegues rapidos de aplicaciones Oracle y no-Oracle, con base

en imagenes de software pre-configuradas manualmente. Cuenta con una interfaz grafica que permite crear y administrar servidores virtuales con facilidad. [**?** ]

### 2.1.8. Chef

Chef es una herramienta de gestión de la configuración escrito en Ruby y Erlang. Utiliza un lenguaje de dominio especifico escrito tamibién en Ruby para la escritura y configuración de 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. Chef contiene soluciones para sistemas de peque na y gran escala. [4]

Es uno de los cuatro principales sistemas de gestión de la configuración en Linux, junto con Cfengine, Bcfg2 y Puppet.

### 2.1.9. Puppet

Puppet es una herramienta dise nada para administrar la configuración de sistemas similares a Unix y a Microsoft Windows de forma declarativa. El usuario describe los recursos del sistema y sus estados utilizando el lenguaje declarativo 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 catálogo específico del sistema que contiene los recursos y la dependencia de dichos recursos, estos catálogos son ejecutados en los sistemas de destino. [5]

### 2.1.10. Cfengine

Cfengine es un sistema basado en el lenguaje escrito por Mark Burgess, dise nado específicamente para probar y configurar software. Cfengine es como un lenguaje de muy alto nivel. La idea de Cfengine es crear un único archivo o conjunto de archivos de configuración que describen la configuración de cada host 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 del host es verificada contra el modelo y, si es necesario, cualquier desviación de la configuración es corregida. [6]

### 2.1.11. Bcfg2

Bcfg2 está escrito en Python y permite gestionar la configuración de un gran número 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, permitiendo una auditoría robusta de los sistemas desplegados. La especificación de la configuración de Bcfg2 estÃą escrita utilizando un modelo XML declarativo. Toda la especificación puede ser validada utilizando los validadores de esquema XML ampliamente disponibles. [7]

# CAPÍTULO 3

# API REST

Este capitulo esta destinado a los desarrolladores que deseen interactuar con el VirtShell API, para realizar aprovisionamientos automáticos desde cualquier plataforma de desarrollo. El VirtShell API es un API REST que provee accesso a los objetos en el VirtShell Server, esto incluye los hosts, imagenes, archivos, templates, aprovisionadores y usuarios. Por medio del API podrá crear ambientes, máquinas virtuales y contenedores personalizados, realizar configuraciones y administrar los recursos físicos y virtuales de manera programática.

## 3.1. Hosts

Representan las máquinas físicas; un host es un anfitrion de máquinas virtuales o contenedores. Los metodos soportados son:

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

Representación del recurso de un host:

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

Ejemplo:

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "host-01-pdn",
  "os": "Ubuntu_12.04_3.5.0-23.x86_64",
  "memory": "16GB",
  "capacity": "120GB",
```

```
  "enabled": "true|false",
  "type":"StorageOptimized|GeneralPurpose|HighPerformance
    ",
  "local_ipv4": "15.54.88.19",
  "local_ipv6": "ff06:0:0:0:0:0:0:c3",
  "public_ipv4": "10.54.88.19",
  "public_ipv6": "yt06:0:0:0:0:0:0:c3",
  "instances": [
    ... instances resource is here
  ],
  "created":["at":"timestamp", "by":1234]
}
```

### 3.1.1. Ejemplos de peticiones HTTP

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

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

Response:

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

#### 3.1.1.2. Obtener un host- GET /virtshell/api/v1/hosts/:id

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

Response:

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

```
      "name": "name2",
      "id": "17173587-C4E9-4369-9C43-FCBF5E075973"
    }
  ],
  "created":["at":"20130625105211", "by":10]
}
```

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

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

Response:

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

```
            "id": "72C05559-0590-4DA6-BE56-28AB36CB669C"
          },
          {
            "name": "name2",
            "id": "17173587-C4E9-4369-9C43-FCBF5E075973"
          }
        ],
        "created":["at":"20130625105211", "by":10]
      },
      {
        "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
        "name": "host-01-pdn",
        "os": "Ubuntu_12.04_3.5.0-23.x86_64",
        "memory": "16GB",
        "capacity": "120GB",
        "enabled": "true",
        "type" : "GeneralPurpose",
        "local_ipv4": "15.54.88.19",
        "local_ipv6": "ff06:0:0:0:0:0:0:c3",
        "public_ipv4": "10.54.88.19",
        "public_ipv6": "yt06:0:0:0:0:0:0:c3",
        "instances": [
          {
            "name": "name3",
            "id": "DE11CC9A-482F-4033-A7F8-503EE449DD0A"
          },
          {
            "name": "name4",
            "id": "17173587-C4E9-4369-9C43-FCBF5E075973"
          },
        ],
        "created":["at":"20130625105211", "by":10]
```

```
52      }
53    ]
54  }
```

#### 3.1.1.4. Actualizar un host - PUT /virtshell/api/v1/hosts/:id

```
curl -sv -X PUT \
  -H 'accept: application/json' \
    -H 'X-VirtShell-Authorization: UserId:Signature' \
  -d '{"memory": "24GB",
     "capacity": "750GB"}' \
   'http://localhost:8080/api/virtshell/v1/hosts?id=
      ab8076c0-db91-11e2-82ce-0002a5d5c51b'
```

Response:

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

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

#### 3.1.1.5. Eliminar un host - DELETE /virtshell/api/v1/hosts/:id

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

Response:

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

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

# CAPÍTULO 4

## Recomendaciones

- Implementar una interfaz web que permita administrar los ambientes y maquinas virtuales.
- Implementar los agentes de monitoreo de recursos.
- 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.

# Bibliografía

# Bibliografía


[1] EECS Department, Northwestern University, http://www.aqualab.cs.northwestern.edu/projects/149-nixes-tool-set, 2013.

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

[3] SmartFrog, https://radia.accelerite.com/solutions, 2015.

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

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

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

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