VirtShell - Framework para aprovisionamiento de soluciones virtuales

CARLOS ALBERTO LLANO RODRIGUEZ

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERIA
INGENIERIA DE SISTEMAS Y COMPUTACION
SANTIAGO DE CALI
2013

VirtShell - Framework para aprovisionamiento de soluciones virtuales

CARLOS ALBERTO LLANO RODRIGUEZ

Tesis de grado para optar el título de
Magíster en Ingeniería con Énfasis en Ingeniería de Sistemas y Computación.

Director
JHON ALEXANDER SANABRIA, Ph.D.

UNIVERSIDAD DEL VALLE
FACULTAD DE INGENIERIA
INGENIERIA DE SISTEMAS Y COMPUTACION
SANTIAGO DE CALI
2013

Santiago de Cali, Junio 22 de 2013

Doctor
****
Decano Académico de la Facultad de Ingeniería
Universidad del Valle
Ciudad

Certifico que el presente proyecto de grado, titulado "VirtShell - Framework para aprovisionamiento de soluciones virtuales" realizado por CARLOS ALBERTO LLANO RODRIGUEZ, estudiante de Maestría en Ingeniería de Sistemas y Computación, se encuentra terminado y puede ser presentado para sustentación.

Atentamente,

Ing. JHON ALEXANDER SANABRIA, Ph.D.
Director del Proyecto

Santiago de Cali, Junio 22 de 2013

Doctor
****
Decano Académico de la Facultad de Ingeniería
Universidad del Valle
Ciudad

Por medio de ésta, presentamos a usted el proyecto de grado titulado "VirtShell - Framework para aprovisionamiento de soluciones virtuales" para optar el título de Magíster en Ingeniería con Énfasis en Ingeniería de Sistemas y Computación..

Esperamos que este proyecto reúna todos los requisitos académicos y cumpla el propósito para el cual fue creado, y sirva de apoyo para futuros proyectos en la Universidad Javeriana relacionados con la materia.

Atentamente,

CARLOS ALBERTO LLANO RODRIGUEZ

ARTICULO 23 de la Resolución No 13 del 6 de Julio de 1946
del Reglamento de la Pontificia Universidad Javeriana.

"La Universidad no se hace responsable por los conceptos emitidos por sus alumnos en sus trabajos de Tesis. Sólo velará porque no se publique nada contrario al dogma y a la moral Católica y porque las Tesis no contengan ataques o polémicas puramente personales; antes bien, se vea en ellas el anhelo de buscar la Verdad y la Justicia"

Nota de Aceptación:

Aprobado por el comité de Trabajo de Grado en
cumplimiento de los requisitos exigidos por la
Universidad del valle para optar el
título de Magíster en Ingeniería con Énfasis en Ingeniería de Sistemas y Computación..

****

Decano Académico de la Facultad de Ingeniería

JESÚS ALEXANDER ARANDA BUENO, Ph.D.
Director de la Carrera de Ingeniería
de Sistemas y Computación

JHON ALEXANDER SANABRIA, Ph.D.
Director de Tesis

PERSONA1
Jurado

PERSONA2
Jurado

Carlos Alberto Llano Rodriguez

A mi Yayita y mi Papá que están en el cielo y se que me hicieron fuerza desde allá, A mi Mamá que la amo mucho y que todo el tiempo me preguntaba por la tesis, A mis hermanos Teresa y Julio, que los amo mucho, A mi novia Leidy obvio también la amo, A todos ellos agradezco su paciencia y comprensión, los quiero muchísimo a todos.

Agradecimientos

El autor expresa su agradecimiento:

- 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

Índice de cuadros

Índice de figuras

Índice de Anexos

Resumen

Introducción

1 Aprovisionamiento de máquinas virtuales

El trabajo realizado sobre aprovisionamiento de aplicaciones y software, en maquinas virtualizadas, ha evolucionado rápidamente. En el mercado se encuentran soluciones, que permiten la interacción con ambientes de virtualización como Xen, KVM, VMWare y VirtualBox.

Herramientas como SmartFrog [?] y Cobbler [?] , ofrecen alternativas basadas en scripts y modelos, que dan un rápido, controlado y automático despliegue de software, en todas las maquinas de una red físicas o virtualizada s, 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 [?].

A continuación se presenta los diferentes enfoques de aprovisionamiento que existen y posteriormente las herramientas que existen en el mercado que abordan el tema de aprovisionamiento sobre maquinas virtuales.

1.1. Enfoques de aprovisionamiento

Hoy en día las herramientas de despliegue de aplicaciones, ofrecen diferentes niveles de automatización, los cuales se clasifican en manual, basados en scripts, basados en lenguajes y basados en modelos. La tabla 1, detalla las tareas que requiere cada enfoque.

Tabla 1: Tareas

El enfoque manual es apropiado para despliegues de escala simple y pequeña, en donde no se necesitan hacer numerosos cambios e instalaciones, los cuales pueden ser realizados con intervención humana. Para grandes escalas, el enfoque basado en scripts presenta ventajas sobre el enfoque manual, apesar de que requiere aprender el lenguaje script que se decida usar [?]. Para escalas de grandes y complejos despliegues, un enfoque basando en lenguaje, como SmartFrog [?], podría ser una mejor alternativa. Este enfoque requiere invertir en el entendimiento de la complejidad del framework y desarrollar las plantillas y configuraciones de las dependencias. Finalmente, los ambientes que requieren de cambios en tiempos de ejecución son mejor manejados por el enfoque basado en modelos. La tabla 2, hace una comparación cualitativa, de los diferentes enfoques de despliegue.

Tabla 2: Cuadro Comparativo

Este cuadro comparativo no esta indicando que exista un óptimo enfoque, la elección de un enfoque depende de la escala y tamaño del sistema, las configuraciones que tengan que realizarse, el numero del instalaciones, entre otros. Como puede notarse los cuatro enfoques son sinérgicos y ellos incrementan el nivel de abstracción de manual a modelo.

1.2. Herramientas de despliegue actuales

En esta sección, se describirán herramientas de código abierto diseñadas con enfoques de despliegue basados en scripts, lenguajes y modelos. Existen muchas otras herramientas para realizar despliegue las cuales están descritas en [?], la gran mayoría de estas herramientas son comerciales y se limitan a configuración de servidores remotos en redes privadas y no abordan el tema de virtualización.

Nixes, SmartFrog, Radia, Cobbler, son ejemplos de los enfoques explicados anteriormente de código abierto que manejan virtualización con Xen, KVM y VMWARE. Amazon EC2 y HP Utility Data Center, por el contrario tienen un 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.

1.2.1. Nixes

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

1.2.2. 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 SF y un modelo de componentes [?]. El lenguaje SF soporta encapsularon (que es similar a las clases de python), herencia y composición que permite personalizar y combinar configuraciones. SF, 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 SF detectar fallas y reiniciar automáticamente re-despliegues de los componentes [?].

1.2.3. Radia

Herramienta de administración de cambios que utiliza un enfoque basado en modelos [?]. 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.

1.2.4. 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. [?]

1.2.5. 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.

1.2.6. 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. [?]

1.2.7. 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. [?]

1.2.8. Shef

1.2.9. vagrant

2 VirtShell: Framework para aprovisionamiento de soluciones virtuales

2.1. Diseño Inicial de VirtShell

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. Con el API de aprovisionamiento se podrá crear ambientes y máquinas virtuales personalizadas, realizar configuraciones y administrar los recursos físicos y virtuales de manera programática.

3.1. Provisioning API conceptos

VirtShell esta construido sobre seis conceptos:

Nodes Un nodo puede ser cualquier máquina física o virtual, los nodos están destinados a ser administrados por el software de aprovisionamiento, manteniendo comunicación con el servidor central.

Appliances Un appliance es una unidad lógica de software, puede ser una máquina virtual o la instalación de un software dentro de una máquina virtual.

Enviroments Un enviroment representa un ambiente virtual computacional, compuesto de dos o más appliances.

LogManager El logmanager es una unidad lógica que comprende el manejo de logs enviados desde los appliances.

Archetypes Los arquetipos comprenden las características, comportamientos y políticas de los appliances.

Users Los usuarios definen los permisos sobre los nodos, los appliances y el sistema en general.

3.2. Provisioning API data model

Un recurso es un entidad de datos individual con un único identificador. Janu Provisioning API opera con seis tipos de recursos:

Node resource Representa un nodo; un nodo es un contenedor de appliances.

```
{
  "uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "name": "host-01-pdn",
  "os": "Ubuntu 12.04 3.5.0-23.x86_64",
  "memory": "16GB",
  "capacity": "5.120GB",
  "enabled": "true|false",
  "local_ipv4": ["15.54.88.19"],
  "local_ipv6": ["gf23::470:89gg:rh1h:17kj","ff06:0:0:0:0:0:0:c3"
     ],
  "public_ipv4": ["10.54.88.19"],
  "public_ipv6": ["yt06:0:0:0:0:0:0:c3"],
  "appliances": [
    appliance Resource
  ]
}
```

Log resource Representa una entrada de un log enviado por un appliance.

```
{
  "date": "25-03-2013",
  "hour": "17:13:01.540",
  "host_uuid": "ab8076c0-db91-11e2-82ce-0002a5d5c51b",
  "host_name": "host-01-pdn",
  "host_local_ipv4", ["15.54.88.19"],
  "appliance_uuid": "kj5436c0-dc94-13tg-82ce-9992b5d5c51b",
  "appliance_name": "webPython-08-dev",
  "appliance_local_ipv4": "192.168.10.29",
  "level": "debug|error|warning|info"
  "type": "io|memory|network|service|os|application|process",
  "msg": "Message....",
  "details": "...."
}
```

Appliance resource Representa un appliance; un appliance puede ser hijo de un ambiente virtual (enviroment), o puede estar contenido en otro appliance..

```
{
  "uuid": "9bf3ced0-ddd7-11e2-a28f-0800200c9a66",
  "name": "webPhp-02-pdn",
  "enabled": "true|false",
  "node_uuid":"1697ffe0-dddc-11e2-a28f-0800200c9a66",  "created"
     :["at":"20130625105211","by":10],
  "user":2,
  "type":"machine|application|package",
  "description","web site for university",
  "metadata": {
  "local_ipv4", ["15.54.88.19","172.16.37.81"],
  "local_ipv6", ["gf23::470:89gg:rh1h:17kj"],
  "public_ipv4", ["10.54.88.19","192.16.37.81"],
  "public_ipv6", ["ui23::470:89gg:rh1h:17kj"]
  },
  "arquetype": [
    arquetype Resource
  ],
  "appliance_ref":"rn8076c0-ws91-11e2-82ce-8962ad5c51b",
  "enviroment_ref":"84ef7950-ddd7-11e2-a28f-0800200c9a66",
  "modified":["at":"20130627105211","by":2]
}
```

Enviroment resource Representa un ambiente virtual.

```
{
  "uuid": "zx8076c0-sl91-11e2-82ce-0002a5d5c51b",
  "name": "EnvWeb-UVIng-pdn",
  "enabled": "true|false",
  "created":["at":"20130625105211","by":10],
  "description","web cluster for valley of university",
  "appliances":[{"quantity":2,"appliance_uuid":"a8abba70-ddd7-11
     e2-a28f-0800200c9a66"},
                {"quantity":1,"appliance_uuid":"db0126c4-ez76-11
                   e2-45cd-1236t7d9c61c"},
                ...
               ],
  ],
  "modified":["at":"20130627105211","by":2]
}
```

Archetype resource Representa un arquetipo; un arquetipo define el comportamiento de los ambientes virtuales y los appliances.

```
{
  "uuid": "zx8076c0-sl91-11e2-82ce-0002a5d5c51b",
```

```
  "name": "ArchWeb-qa",
  "enabled": "true|false",
  "created":["at":"20130625105211","by":10],
  "description","Archetype to describe a web site for valley of
     university",
  "machine_configuration": [
    "os": "Ubuntu 12.04 3.5.0-23.x86_64",
    "memory": "4096MB",
    "vcpu":1,

    "size":
    "time_zone": "America/Bogota",
    "host_uuid":"ab8076c0-db91-11e2-82ce-0002a5d5c51b",
    "lang":"en_US",
    "keyboard":"us",
    "rootpw": "$1$rB3lJzDT$PmMFJJiLgVPbnvJF6UZ1x",
    "network": {"method":"dhcp|bootp|static",
          "device":"eth0",
    "configuration": {
        "boot":"static",
         "ip":"10.0.2.15",
         "netmask":"255.255.255.0",
         "gateway":"10.0.2.254",
         "nameserver":"192.168.2.1,192.168.3.1"}
    },
    "partitioning": ["/":250, "swap":50, "/usr":500, "/tmp",100],
    "packages": ["@base","@core","httpd","http-devel","php",...],
    "post_scripts": "...",
  ],
  "application_configuration": "...",
  "package_configuration": "...",
  "modified":["at":"20130627105211","by":2]
}
```

User resource Representa un usuario; un usuario es usado para identificar quien tiene autorización y permiso para acceder a los nodos, sus appliances y el sistema.

```
{
  "id": 10,
  "name": "username",
  "login": "login",
  "password": "bb6ea22989b3494e5ca80c2a159486b2",
  "enabled": "true|false",
  "created":["at":"20130625105211","by":10],
  "role", "root|user",
  "modified":["at":"20130627105211","by":2],
  "metadata": [
    "address": "...",
    "celular": "...",
    "email": "user@server.domain"
  ]
```

```
}
```

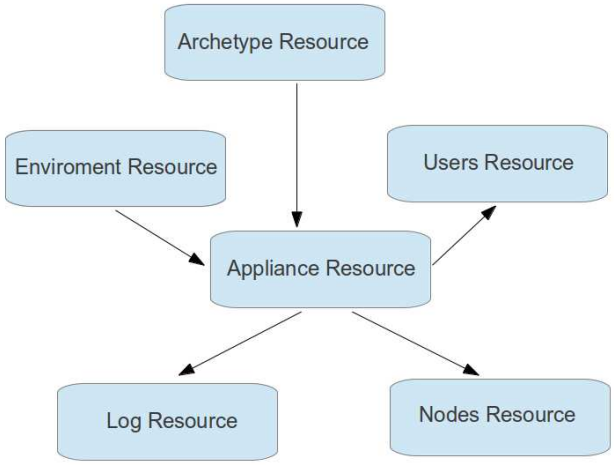

Figura 3.1: Visión de las relaciones entre recursos

El modelo de datos del Janu API esta basado en grupos de recursos, llamados colecciones:

Nodes collection Un colección de nodos consiste de todos los nodos que maneja el sistema. Es posible listar los nodos por país o por ID.

Log collection Una colección de logs consiste en todos los logs recibidos de los appliance. Es posible listar los logs por appliance, fecha o por tipo de log.

Appliance collection Una colección de appliances consiste de todos los appliance registrados en el sistema. Es posible listarlos también por nodo o por ID.

Enviroment collection Una colección de enviroments consiste de todas los enviroments creados en el sistema. Es posible listarlos por un usuario especifico.

Archetype collection Una colección de arquetipos consiste en todos los arquetipos creados por los usuarios. Es posible listar los arquetipos por usuario, tipo o por ID.

User collection Una colección de usuarios consiste en todos los usuarios con registrados en el sistema. Es posible listarlos por ID o por su propio identificador.

3.3. Provisioning API reference

El API esta organizado por tipo de recurso. Cada tipo de recurso tiene uno o mas representaciones de datos y uno o mas métodos.

Muchos recursos de este API retornan <code>application/json</code>y esperan el http header <code>Accept: application/json</code>. Si se omite o falla en pasar este header en la petición http, la petición fallara.

3.3.1. Nodes

Method	HTTP request	Description
list	GET /api/janu/nodes	Recibe una lista de recursos de nodos contenidos dentro del sistema.
get	GET /api/janu/nodes/{nodeId}	Retorna el recurso de un nodo especifico.
insert	POST /api/janu/nodes/{nodeId}	Crea un nuevo nodo en el sistema.
update	PUT /api/janu/nodes/{nodeId}	Actualiza un nodo en el sistema.
delete	DELETE /api/janu/nodes/{nodeId}	Elimina un nodo del sistema.

Tabla 3.1: Nodes API

3.3.2. Log

Method	HTTP request	Description
list	GET /api/janu/log/node/{nodeId}	Recibe una lista de recursos de log de un nodo especifico.
list	GET /api/janu/log/appliance/{applianceId}	Recibe una lista de recursos de log de un appliance especifico.

Tabla 3.2: Log API

3.3.3. Appliance

Method	HTTP request	Description
get	GET /api/janu/appliance/{applianceId}	Recibe un recurso de un appliance especifico.
insert	POST /api/janu/appliance/{applianceId}	Crea un nuevo appliance en el sistema.
update	PUT /api/janu/appliance/{applianceId}	Actualiza un appliance en el sistema.
delete	DELETE /api/janu/appliance/{applianceId}	Elimina un appliance del sistema.
stop	POST /api/janu/appliance/{applianceId}/stop	Detiene un appliance del sistema.
start	POST /api/janu/appliance/{applianceId}/start	Inicia un appliance del sistema.
clone	POST /api/janu/appliance/{applianceId}/clone	Clona un appliance del sistema.
move	POST /api/janu/appliance/{applianceId}/move/node/ {nodeId}	Mueve un appliance a un nodo especifico.
restart	POST /api/janu/appliance/{applianceId}/restart	Reinicia un appliance.

Tabla 3.3: Appliance API

3.3.4. Enviroment

Method	HTTP request	Description
get	GET /api/janu/enviroment/{enviromentId}	Recibe un recurso de un enviroment especifico.
insert	POST /api/janu/enviroment/{enviromentId}	Crea un nuevo enviroment en el sistema.
update	PUT /api/janu/enviroment/{enviromentId}	Actualiza un enviroment en el sistema.
delete	DELETE /api/janu/enviroment/{enviromentId}	Elimina un enviroment del sistema.

Tabla 3.4: Enviroment API

3.3.5. Archetype

Method	HTTP request	Description
get	GET /api/janu/archetype/{archetypeId}	Recibe un recurso de un archetype especifico.
insert	POST /api/janu/archetype/{archetypeId}	Crea un nuevo archetype en el sistema.
update	PUT /api/janu/archetype/{archetypeId}	Actualiza un archetype en el sistema.
delete	DELETE /api/janu/archetype/{archetypeId}	Elimina un archetype del sistema.

Tabla 3.5: Archetype API

3.3.6. User

Method	HTTP request	Description
list	GET /api/janu/user/	Recibe una lista de recursos de usuarios del sistema.
get	GET /api/janu/user/{userId}	Recibe un recurso de un usuario especifico.
insert	POST /api/janu/user/{userId}	Crea un nuevo usuario en el sistema.
update	PUT /api/janu/user/{userId}	Actualiza un usuario en el sistema.
delete	DELETE /api/janu/user/{userId}	Elimina un usuario del sistema.

Tabla 3.6: User API

3.4. HTTP status codes

La API de aprovisionamiento intenta devolver códigos de estado HTTP apropiados para cada solicitud. La siguiente tabla describe los código HTTP significativos en el contexto del API de aprovisionamiento.

Code	Explanation
200 Ok	No error.
201 Created	Creation of a resource was successful.
304 Not modified	The resource hasn't changed since the time specified in the request's If-Modified-Since header.
400 Bad request	Invalid request URI or header, or unsupported nonstandard parameter.
401 Unauthorized	Authorization required.
403 Forbidden	Unsupported standard parameter, or authentication or authorization failed.
404 Not found	Resource (such as a feed or entry) not found.
405 Method Not Allowed	The specified method is not allowed against this resource.
409 Conflict	Specified version number doesn't match resource's latest version number.
500 Internal server error	Internal error. This is the default code that is used for all unrecognized server errors.
501 Not Implemented	A header you provided implies functionality that is not implemented.

Tabla 3.7: HTTP status codes

3.5. El estilo REST

REST es un estilo de arquitectura de software que proporciona un enfoque práctico y consistente para solicitar y modificar datos.

El termino REST es la abreviatura para R̈epresentational State Transfer.Ën el contexto del Janu API, se refiere al uso de verbos HTTP para recibir y modificar representaciones de datos guardadas por el sistema de aprovisionamiento.

En un sistema RESTful, los recursos se almacenan en un almacén de datos, un usuario envía una solicitud al servidor para realizar una acción determinada (como la creación, recuperación, actualización o eliminación de un recurso), y el servidor realiza la acción y envía una respuesta, a menudo en la forma de una representación del recurso especificado.

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

https://www.domain.edu.co/api/janu/resourcePath?parameters

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

3.6. Formato de datos - JSON

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 más información, ver json.org.

3.7. Ejemplos

3.8. Autenticación

La autenticación es el proceso de demostrar la identidad al sistema. La identidad es un factor importante en las decisiones de control de acceso. Las solicitudes se conceden o deniegan en parte sobre la base de la identidad del solicitante.

El Janu 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, utiliza una clave secreta de acceso para calcular el HMAC de esa cadena. Informalmente, llamamos a este proceso "la firma de la solicitud"; y llamamos a la salida del algoritmo HMAC la "firma"; ya que simula las propiedades de seguridad de una firma real. Por último, se agrega esta firma como un parámetro de la petición, con la sintaxis descrita en esta sección.

Cuando el sistema recibe una solicitud fehaciente, se obtiene la clave secreta de acceso que dicen tener, y lo utiliza de la misma manera que se calcula una "firma"del mensaje que recibió. A continuación, compara la firma que se calcula con la firma presentada por el solicitante. Si las dos firmas coinciden, el sistema llega a la conclusión de que el solicitante debe tener acceso a la clave secreta de acceso, y por lo tanto actúa con la autoridad del principal al que se emitió la clave. Si las dos firmas no coinciden, la

solicitud se descarta y el sistema responde con un mensaje de error.

Ejemplo de una petición autenticada:

```
GET /api/janu/appliance/{applianceId} HTTP/1.1
Host: host1.edu.co
Date: Fri, 01 Jul 2011 19:37:58 +0000

Authorization: 0PN5J17HBGZHT7JJ3X82:frJIUN8DYpKDtOLCwo//yllqDzg=
```

3.8.1. Authentication Header

El API REST utiliza el encabezado de autorización HTTP estándar para pasar información de autenticación. (El nombre de la cabecera estándar es insuficiente, ya que solo lleva la información de autenticación y no la de autorización). Bajo el esquema de autenticación de Janu, el encabezado de autorización tiene la siguiente forma.

```
Authorization: JanuUserId:Signature
```

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

Para la firma de los elementos de la petición se usa el RFC 2104HMAC-SHA1, por lo que la parte de la firma de la cabecera autorización variará de una 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 se emitió la clave.

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

```
Authorization = JanuUserId + ":" + Signature;

Signature = Base64( HMAC-SHA1( UTF-8-Encoding-Of(
   YourSecretAccessKeyID, StringToSign ) ) );
```

```
StringToSign = HTTP-Verb + "\n" +
Host + "\n" +
Content-MD5 + "\n" +
Content-Type + "\n" +
Date + "\n" +
CanonicalizedResource;

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). El algoritmo toma como entrada dos cadenas de bytes: una clave y un mensaje. Para la solicitud de autenticación, se utiliza la clave secreta (YourSecretAccessKeyID) como la clave, y la codificación UTF-8 del StringToSign como el mensaje. La salida de HMAC-SHA1 es también una cadena de bytes, llamado el resumen. El parámetro de la petición de la Firma se construye codificada en Base64.

3.8.2. 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 Janu. A este proceso de poner una solicitud en una forma acordada para la firma se denomino çanonización".

3.8.3. Tiempo de sello

Un sello de tiempo 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 reproducidos por un adversario.Para una mayor protección contra las escuchas, se debe utilizar el transporte HTTPS para solicitudes autenticadas.

3.8.4. Ejemplos de autenticación

Parametro	Valor
JanuUserId	13010f3e-3f46-4889-b989-592ce8fb30c6
JanuSecretAccessKey	c991f519-bed0-4dab-9165-6d9f722dc845 **Base64:** Yzk5MWY1MTktYmVkMC00ZGFiLTkxNjUtNmQ5ZjcyMmRjODQ1

Ejemplo de un objeto con GET

Este es un ejemplo que consulta por un enviroment dado su identificador.

Request	StringToSign
GET /api/janu/enviroment/45 HTTP/1.1 Host: host1.edu.co Date: Tue, 27 Mar 2007 19:36:42 +0000 Authorization: 13010f3e-3f46-4889-b989-592ce8fb30c6: Yzk5MWY1MmVkMC00ZGFiLTtNmQ5ZjcyMmRjODQ1	GET\n host1.edu.co\n \n \n Tue, 27 Mar 2007 19:36:42 +0000\n /api/janu/enviroment/45

Ejemplo de un objeto con DELETE

Este ejemplo remueve un usuario.

Request	StringToSign
DELETE /api/janu/user/9876 HTTP/1.1 Host: host1.edu.co Date: Tue, 27 Mar 2007 21:20:27 +0000 Authorization: 13010f3e-3f46-4889-b989-592ce8fb30c6: Yzk5MWY1MmVkMC00ZGFiLTtNmQ5ZjcyMmRjODQ1	DELETE\n host1.edu.co\n \n \n Tue, 27 Mar 2007 21:20:27 +0000\n /api/janu/user/9876

4 Implementación del framework

5 Pruebas y resultados

6 Conclusiones

7 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

[Mil80] Robin Milner. A Calculus of Communicating Systems. Lecture Notes in Computer Science, LNCS 92, 1980.

[Mil92] Robin Milner. Functions as Processes. *Journal of Mathematical Structures in Computer Science*, 2(2):119–141, 1992.

[PT94] Benjamin C. Pierce and David Turner. Concurrent Objects in a Process Calculus. In *Theory and Practice of Parallel Programming (TPPP), Sendai, Japan*, November 1994.

[Rue] Camilo Rueda. A Visual Programming Enviroment for Constraint-Based Musical Composition.

[Sar93] Vijay A. Saraswat. *Concurrent Constraint Programming.* The MIT Press, Cambridge, MA, 1993.