API Python de OpenStack Para Administración de Usuarios

Carlos Miguel Hernández Romero

Proyecto Integrado 2º ASIR **Junio 2013**

Índice de contenido

1. Introducción.

El proyecto tiene como objetivo aprender a usar la API Python de OpenStack como herramienta para realizar una aplicación en Python, en donde al administrador se le facilitara la tarea a la hora de tener que eliminar el contenido de un usuario (Paso de un usuario de primero a segundo) o eliminarlo completamente (Un alumno de segundo termina el ciclo).

Vamos a ver como funciona por dentro OpenStack ,de que partes esta compuesto y un ejemplo de las múltiples cosas que se podrían hacer con la API Python de OpenStack.

Todo el contenido de la aplicación se aloja en el Github:
https://github.com/miguelhr/AdminOpenStack

2. ¿Que es OpenStack?

OpenStack es un proyecto Open Source, que cuenta con un gran número de colaboradores trabajando en el mismo, no sólo desarrolladores sino expertos en otras ramas de la informática como redes y almacenamiento.

También mencionar que se trata de un proyecto cuyo origen es la NASA y al que, con el paso del tiempo, se le han unido empresas y organizaciones de fama mundial como HP, Dell, IBM, etc.

El objetivo de OpenStack es crear una plataforma en software libre para cloud computing que cumpla con las necesidades de los proveedores de nubes publicas y privadas, independientemente de su tamaño, que sea fácil de implementar y masivamente escalable

3. ¿Que es una API Web RESTful?

Las API web de tipo RESTful son las que se están imponiendo en internet a la hora de realizar comunicaciones de aplicaciones con servicios web disponibles. Poco a poco esta desbancando SOAP.

En un principio su finalidad es simple, poder permitir a los desarrolladores de software crear programas que interactúen con otras aplicaciones. Un buen ejemplo son todas las aplicaciones existentes que interactúan o utilizan de alguna manera a otras aplicaciones tales como Facebook, Twitter o incluso mapas como los de Google Maps.

Las APIS REST tienen un conjunto de características que la definen:

- Cada uno de ellos es de acceso único a través de una URI (Universal Resource Identifier – identificador de recursos universal).
- El formato utilizado mas extendidos para la transferencia son XML y JSON.
- Hay cuatro diferentes formas de comunicarnos con un servidor, mediante los métodos POST, GET, PUT, y DELETE (recientemente se agregaron algunos otros como PATCH y OPTIONS pero aun no son soportados por todas las librerías web)

4. APIs Python de OpenStack.

OpenStack esta formado por varios componente, todos ellos trabajando en conjunto. La aplicación no utiliza una sola API si no varias comunicándose entre si mediante APIs RESTful, las cuales se explican a continuación:

Nova: Es el componente que se encarga de la gestión de Instancias.
Cinder: Nos da la posibilidad de obtener almacenamiento de volúmenes.
Keystone: Proporciona servicios de autenticación y autorización a todos los servicios de OpenStack.
Glance: Nos permite gestionar el almacenamiento de las imágenes para nuestras instancias.
Neutron: Conocido en versiones anteriores como “Quantum” es el encargados de las redes virtuales.

También tenemos que mencionar que cuenta con una aplicación web llamada “Horizon” cuyo objetivo principal es proporcionar una interfaz de los servicios de OpenStack ,al administrador del cloud y a los usuario.

Horizon no proporciona toda la funcionalidad que podemos conseguir a través del intérprete de comandos, pero lo “poco” que hace lo hace correctamente.

5. Ficheros que Compone la Aplicación.

La aplicación esta compuesta por dos documentos “credentials.py y userdelStack.py”.
En el fichero credentials.py en donde se alojan las credenciales de los diferentes servicios, los cuales serán llamados cuando sean necesarios.
En userdelStack.py es donde se encuentra el grueso de la aplicación y la mayor parte del código.
Ahora vamos a proceder a explicar las diferentes partes en la que se compone la aplicación y la función de cada una de ellas.

5.1 Credentials.py.

Tendremos que importar el modulo “os” el cual nos facilita el diccionario de “environ”. Con ello conseguiremos establecer las variables de entorno que usamos en nuestra maquina con el comando source y nuestra archivo de identificación de la API y podremos autenticarnos.

```
import os
def get_keystone_creds():
    d = {}
    d['username'] = os.environ['OS_USERNAME']
    d['password'] = os.environ['OS_PASSWORD']
    d['auth_url'] = os.environ['OS_AUTH_URL']
    d['tenant_name'] = os.environ['OS_TENANT_NAME']
    return d

def get_nova_creds():
    d = {}
    d['username'] = os.environ['OS_USERNAME']
    d['api_key'] = os.environ['OS_PASSWORD']
```

```
    d['auth_url'] = os.environ['OS_AUTH_URL']
    d['project_id'] = os.environ['OS_TENANT_NAME']
    return d

def get_credentials():
    d = {}
    d['username'] = os.environ['OS_USERNAME']
    d['password'] = os.environ['OS_PASSWORD']
    d['auth_url'] = os.environ['OS_AUTH_URL']
    d['tenant_name'] = os.environ['OS_TENANT_NAME']
    return d

def get_cinder_credentials():
    d = [os.environ['OS_USERNAME'],
    os.environ['OS_PASSWORD'],
    os.environ['OS_TENANT_NAME'],
    os.environ['OS_AUTH_URL']]
    return d
```

5.2 UserdelStack.py.

Importamos los diferente módulos que vamos a usar:

```
import novaclient.v1_1.client as nvclient
from keystoneclient.v2_0 import client
from neutronclient.v2_0 import client as neuclient
from cinderclient.v2 import client as cinderclient
import glanceclient.v2.client as glclient
import commands
from keystoneclient.apiclient import exceptions as api_exceptions
from credentials import get_keystone_creds
from credentials import get_nova_creds
from credentials import get_cinder_credentials
import time
import sys
import os
```

Esta parte intentamos que si usuario comete algún error a la hora de introducir los parámetros salte un error indicándoles los datos a introducir.

Si el numero de argumentos es mayor o inferior a 3 o el argumento segundo es distinto de -completo o -parcial nos saltara el error userdelStack.py <-completo o -parcial> <usuario>:

```
if len(sys.argv) > 3 or len(sys.argv) < 3 or sys.argv[1]!="-completo" and sys.argv[1]!="-parcial":
    print "Por favor, si quiere eliminar un usuario, la sitaxis es"
    print "userdelStack.py <-completo o -parcial> <usuario>"
```

En caso de que todo sea correcto, entonces empezara a crear todas las funciones. Cada una de ella tiene indicado su función con un comentario con ”#”:

```
else:
#Eliminar todas las instancias asociadas a un proyecto.
   def instancias():
      totalinstancias=nova.servers.list(search_opts={'all_tenants': True})
      contador=0
      for instancias in totalinstancias:
         if proyecto in instancias.tenant_id:
            contador=contador+1
      if contador>0:
         print "El proyecto %s tiene la/las instancia/s:" % proyecto
         for instancias in totalinstancias:
            if proyecto in instancias.tenant_id:
               print instancias.name
               totalgruposeguridad = neutron.list_security_groups()
               for gruposeguridad in totalgruposeguridad:
                  for objeto in totalgruposeguridad[gruposeguridad]:
                     if proyecto in objeto['tenant_id'] and
objeto['name']==instancias.security_groups[0]['name']:
                        nova.servers.remove_security_group(instancias.id,objeto['id'])
               nova.servers.delete(instancias.id)

#Eliminar todos los grupos de seguridad asociadas a un proyecto.
   def gruposeguridad():
      totalgruposeguridad = neutron.list_security_groups()
      contador=0
      for gruposeguridad in totalgruposeguridad:
         for objeto in totalgruposeguridad[gruposeguridad]:
            if proyecto in objeto['tenant_id']:
               contador=contador+1
      if contador>1:
         print "El proyecto %s tiene el/los grupo/s de seguridad:" % proyecto
         totalgruposeguridad = neutron.list_security_groups()
         for gruposeguridad in totalgruposeguridad:
            for objeto in totalgruposeguridad[gruposeguridad]:
               if proyecto in objeto['tenant_id']:
                  print objeto['name']
                  neutron.delete_security_group(objeto['id'])

#Eliminar todos los snapshots de volumenes asociadas a un proyecto.
   def snapshoptvolumenes():
      totalsnap_volu=cinder.volume_snapshots.list(search_opts={'all_tenants': True})
      contador=0
      for snap_volu in totalsnap_volu:
         resultado = commands.getoutput("cinder  snapshot-show %s" % snap_volu.id)
         if proyecto in resultado.split('|')[23].strip():
            contador=contador+1
```

```
        if contador>0:
            print "El proyecto %s tiene el/los snapshot/s de volumene/s:" % proyecto
            totalsnap_volu=cinder.volume_snapshots.list(search_opts={'all_tenants': True})
            for snap_volu in totalsnap_volu:
                resultado = commands.getoutput("cinder  snapshot-show %s" % snap_volu.id)
                if proyecto in resultado.split('|')[23].strip():
                    print snap_volu.name
                    cinder.volume_snapshots.delete(snap_volu.id)
            time.sleep(4)

#Eliminar todos los volumenes asociadas a un proyecto.
    def volumenes():
        totalvolu=cinder.volumes.list(search_opts={'all_tenants': True})
        contador=0
        for volu in totalvolu:
            if proyecto in volu._info['os-vol-tenant-attr:tenant_id']:
                contador=contador+1
        if contador>0:
            print "El proyecto %s tiene el/los volumen/es:" % proyecto
            for volu in totalvolu:
                if proyecto in volu._info['os-vol-tenant-attr:tenant_id']:
                    print volu.name
                    cinder.volumes.delete(volume=volu.id)

#Eliminar todos las IP flotantes de un proyecto.
    def ipflotante():
        totalipflota = neutron.list_floatingips()
        contador=0
        for ipflota in totalipflota:
            for objeto in totalipflota[ipflota]:
                if proyecto in objeto['tenant_id']:
                    contador=contador+1
        if contador>0:
            print "El proyecto %s tiene la/las IP flotante/s:" % proyecto
            for ipflota in totalipflota:
                for objeto in totalipflota[ipflota]:
                    if proyecto in objeto['tenant_id']:
                        print objeto['floating_ip_address']
                        neutron.delete_floatingip(objeto['id'])

#Eliminar todos las imagenes de un proyecto.
    def imagenes():
        listaimage=glance.images.list()
        contador=0
        for image in listaimage:
            resultado = commands.getoutput("glance show %s" % image.id)
            if proyecto in resultado.split(' ')[20].split('\n')[0]:
                contador=contador+1
```

```
        if contador>0:
            print "El proyecto %s tiene la/las imagen/es:" % proyecto
            listaimage=glance.images.list()
            for image in listaimage:
                resultado = commands.getoutput("glance show %s" % image.id)
                if proyecto in resultado.split(' ')[20].split('\n')[0]:
                    print image.name
                    glance.images.delete(image.id)

#Eliminar todos los routers de un proyecto.
    def routers():
        routers=neutron.list_routers(tenant_id=proyecto)
        if len(routers['routers'])>0:
            print "El proyecto %s tiene el/los router/s:" % proyecto
            for router in neutron.list_routers(tenant_id=proyecto)["routers"]:
                neutron.remove_gateway_router(router["id"])
                for port in neutron.list_ports(tenant_id=proyecto)["ports"]:
                    if port["device_id"] == router["id"]:
                        neutron.remove_interface_router(router["id"],{'port_id':port["id"]})
                        print router["name"]
                        neutron.delete_router(router["id"])

#Eliminar todos las subredes de un proyecto.
    def subredes():
        subredes=neutron.list_subnets(tenant_id=proyecto)
        if len(subredes['subnets'])>0:
            print "El proyecto %s tiene la/las subred/es:" % proyecto
            for objeto in subredes["subnets"]:
                print objeto['name']
                neutron.delete_subnet(objeto['id'])

#Eliminar todos las redes de un proyecto.
    def redes():
        redes=neutron.list_networks(tenant_id=proyecto)
        if len(redes['networks'])>0:
            print "El proyecto %s tiene la/las red/es:" % proyecto
            for objeto in redes["networks"]:
                print objeto['name']
                neutron.delete_network(objeto['id'])

#Eliminar usuario.
    def usuario():
        print "Se va a eliminar el usuario %s" % sys.argv[2]
        keystone.users.delete(keystone.users.find(name=sys.argv[2]).id)
```

```
#Eliminar proyecto.
   def proyectos():
      print "Se va a eliminar el proyecto con id %s" % proyecto
      keystone.tenants.delete(proyecto)
```

Seguidamente recuperamos los clientes de cada uno de los servicios importados mas arriba. Los doble “**” indican la versión especifica que viene cuando importamos cada modulo en este ejemplo la **v1_1** (import novaclient.**v1_1**.client as nvclient):

```
   credsnova = get_nova_creds()
   credskeystone = get_keystone_creds()
   credscinder = get_cinder_credentials()

   nova = nvclient.Client(**credsnova)
   keystone = client.Client(**credskeystone)
   neutron = neuclient.Client(**credskeystone)
   cinder = cinderclient.Client(*credscinder)
   glance_endpoint =
keystone.service_catalog.url_for(service_type='image',endpoint_type='publicURL')
   glance = glclient.Client(glance_endpoint, token=keystone.auth_token)
```

El tercer argumento que introducimos el la linea de comando es el usuario que queremos eliminar, es este paso es donde comprobamos si el usuario existe, en caso contrario nos mostrara un error donde dirá que el usuario no existe:

```
#Obtener informacion de usuario.
   try:
      infousuario = keystone.users.find(name=sys.argv[2])
      listatenant=[];

   except api_exceptions.NotFound:
      print "no existe el usuario %s." % sys.argv[2]
      sys.exit(0)
```

Si el usuario existe pasaremos a ver que proyectos tiene el usuario, dando como resultado el nombre del proyecto y su id:

```
#Obtener proyectos del usuario elegido.
   print "El usuario %s tiene el/los proyecto/s:" %  infousuario.name
   for tenant in keystone.tenants.list():
      for tenant_user in tenant.list_users():
         if infousuario.id in tenant_user.id:
            listatenant.append(tenant.id)
            print tenant.name, tenant.id
```

Si el proyecto pertenece a varios usuario no queremos que el contenido de estos proyectos sea eliminado, por ese motivo con este apartado esos proyectos sera obviados y no los eliminara. Finalmente mostrara en pantalla a que usuario pertenece ese proyecto:

```
#Que proyectos pertenecen a mas de un usuario.
   for proyecto in listatenant:
      if len(keystone.tenants.list_users(tenant=proyecto)) > 1:
         print "No se puede eliminar el proyecto con id %s porque pertenece a mas de un usuario." %
proyecto
         print "Los usuarios son:"
         usutenant=keystone.tenants.list_users(tenant=proyecto)
         for tenant in usutenant:
            print tenant.name
```

En caso de que el proyecto solo pertenezca al usuario que queremos eliminar mirara el segundo argumento de nuestro comando y distinguirá si queremos eliminar solo el contenido del proyecto, o por el contrario de forma completa, incluido usuario y proyecto. De esta manera empezara a llamar a todas las funciones:

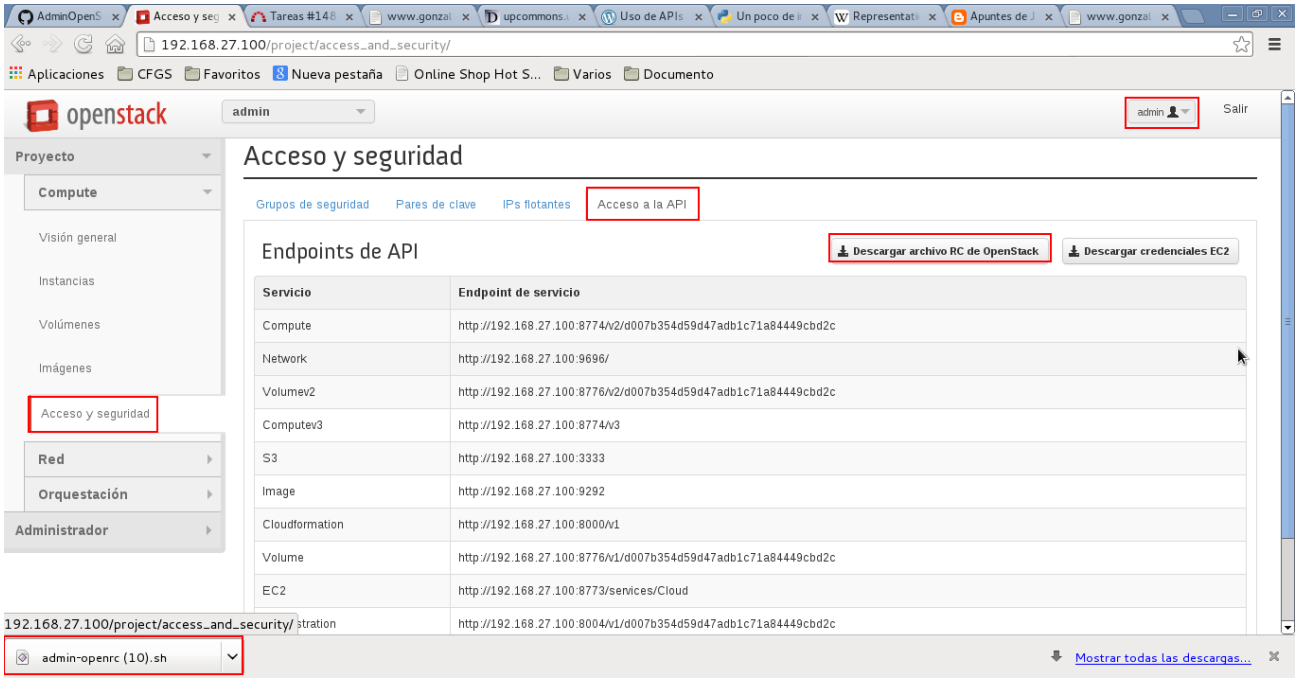
```
      else:
         if sys.argv[1]=="-parcial":
            print "se va a eliminar el contenido del proyecto con id %s" % proyecto
            instancias()
            snapshoptvolumenes()
            gruposeguridad()
            volumenes()
            ipflotante()
            imagenes()
            routers()
            subredes()
            redes()
         if sys.argv[1]=="-completo":
            print "se va a eliminar el proyecto con id %s y todo su contenido" % proyecto
            instancias()
            snapshoptvolumenes()
            gruposeguridad()
            volumenes()
            ipflotante()
            imagenes()
            routers()
            subredes()
            redes()
            usuario()
            proyectos()
```

6. Modo de Uso.

En esta sección explicaremos el modo de uso de la aplicación y las diferentes opciones disponibles.

Para poder usar la aplicación es imprescindible que usemos un usuario con el rol admin, ya que sino no tendremos los privilegios necesarios.

Es necesario definir varias variables de entorno, la forma mas sencilla es obtener el fichero openrc.sh de Horizon:



Ejecutamos en la consola:

```
root@precise64:/home/vagrant# source admin.sh
Please enter your OpenStack Password:
```

De esta forma estamos definiendo es esa sesión las variable de entorno siguientes:

```
OS_AUTH_URL
OS_TENANT_ID
OS_TENANT_NAME
OS_USERNAME
OS_PASSWORD
```

Una vez tengamos una sesión es hora de explicar la forma de uso de la aplicación.
En un principio tenemos dos opciones disponibles que son:

- -parcial: Podremos eliminar todo el contenido de un proyecto, pero sin eliminar el usuario y el propio proyecto.
- -completo: Se eliminara todo el contenido del proyecto inclusive el propio proyecto y el usuario

Una sabemos las opciones que tenemos disponibles, es el momento de explicar como es la sintaxis a introducir en la linea de comando. Un ejemplo de ello es:

```
python prueba2.py -parcial migue

<lenguaje de programación> <aplicación> <opción> <usuario>
```

Si cometemos algún fallo en la sintaxis no saltara un error dándonos los parámetros correctos a introducir:

```
root@precise64:/home/vagrant# python prueba2.py -parcial migue pepe
Por favor, si quiere eliminar un usuario, la sintaxis es
python userdelStack.py <-completo o -parcial> <usuario>
```

Si por el contrario introducimos un usuario que no existe:

```
root@precise64:/home/vagrant# python prueba2.py -parcial miguko
no existe el usuario miguko.
```

Ahora realizaremos una prueba real, donde habrá 2 usuario,el usuario migue pertenece al proyecto migue y al proyecto jose, y el usuario jose solamente pertenece al proyecto jose.
Dentro de ambos proyecto se encuentra varios objetos como instancias, volúmenes, etc.

En este caso, vamos a eliminar por completo el usuario migue,el proyecto jose no debería de borrarse por que pertenece a dos usuario (migue y jose), en cambio el proyecto migue y todo su contenido si.

Al introducir en la terminal el comando, primeramente no detectara que proyectos tiene el usuario.
Nos detecta que uno de los proyectos pertenece a mas de un usuario indicando cuales.
Seguidamente sigue con el siguiente proyecto que al pertenecer a un usuario si lo puede eliminar.
Va mostrando todo el contenido del proyecto indicando que va borrando.
Finalmente eliminar el usuario y el proyecto y finaliza correctamente:

```
root@precise64:/home/vagrant# python prueba2.py -completo migue
El usuario migue tiene el/los proyecto/s:
proyecto_jose 57b7800c80ab4695a7973e4d708fdf31
proyecto_migue cb6e34e08679456ea46d045ac5630802
No se puede eliminar el proyecto con id 57b7800c80ab4695a7973e4d708fdf31 porque pertenece a mas de un usuario.
Los usuarios son:
migue
jose
se va a eliminar el proyecto con id cb6e34e08679456ea46d045ac5630802 y todo su contenido
El proyecto cb6e34e08679456ea46d045ac5630802 tiene la/las instancia/s:
instantanea_migue
El proyecto cb6e34e08679456ea46d045ac5630802 tiene el/los snapshot/s de volumene/s:
volumeninstantanea_migue
El proyecto cb6e34e08679456ea46d045ac5630802 tiene el/los grupo/s de seguridad:
```

```
gruposeguridad_migue
default
El proyecto cb6e34e08679456ea46d045ac5630802 tiene el/los volumen/es:
volumen_migue
El proyecto cb6e34e08679456ea46d045ac5630802 tiene la/las IP flotante/s:
192.168.27.130
192.168.27.142
El proyecto cb6e34e08679456ea46d045ac5630802 tiene la/las imagen/es:
imagen_migue
El proyecto cb6e34e08679456ea46d045ac5630802 tiene el/los router/s:
router_migue
El proyecto cb6e34e08679456ea46d045ac5630802 tiene la/las subred/es:
subred_migue
El proyecto cb6e34e08679456ea46d045ac5630802 tiene la/las red/es:
red_migue
Se va a eliminar el usuario migue
Se va a eliminar el proyecto con id cb6e34e08679456ea46d045ac5630802
```

Si en caso contrario quisiéramos eliminar solo el contenido del proyecto seria igual pero introduciendo “-parcial” en vez de “-completo”:

```
root@precise64:/home/vagrant# python prueba2.py -parcial migue
```

7. Problemas Encontrados.

Es importante comentar los problemas que me he encontrado a la hora de realizar la aplicación.
A la hora de querer listar o eliminar los pares de claves de un usuario o proyecto en concreto existe un bug de tipo "whishlist" en OpenStack , es decir, la solicitud de una nueva funcionalidad que no está soportada en el cliente nova. El problema no está en el cliente nova sino que la API de nova no permite obtener un par de claves de un usuario diferente del autenticado.

Por tanto, es una funcionalidad que no esta implementada ni en la API ni en el cliente y no sera posible utilizarla en la aplicación.

Al borrar el usuario y no borrar sus pares de claves simplemente se quedará un registro "basura" en la tabla de pares de claves, pero es la única opción que veo posible.

Otro problema que me he encontrado es a la hora de obtener el listado de los snapshots de volúmenes y de las imágenes/snapshots de instancias de un proyecto en concreto.

Para poder solucionarlo he tenido que hacer uso de la linea de comando, importando la librería “import commands” con la que de este modo si he podido obtener el resultado correcto.

Una curiosidad que me llevo un tiempo en dar con la solución es que una instantánea de volumen que estaba asociada a un volumen, al eliminar la instantánea de volumen y seguidamente el volumen, me indicaba que el volumen seguía asociado a una instantánea de volumen. Tras dar varias vueltas di con el problema.

Al parecer al eliminar la instantánea de volumen tarda varios segundo en hacerse efectivo. Como solución, importe la librería time “import time” y al eliminar las instantáneas de volúmenes ahí 4 segundos de espera, los cuales le dan el tiempo necesario para que se eliminen. Acto seguido sigue el proceso eliminando los volúmenes sin problema

Para encontrar una solución a algunos de estos problema, pedí ayuda a la lista de correo de OpenStack. Existen lista de multitud de idiomas, pero la inglesa en la que mas numero de post diarios ahí.

Esta en la dirección de la lista de correo de OpenStack de España:
http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-es

8. Conclusión.

Las conclusiones a las que se puede llegar de este proyecto son varias, tanto a nivel técnico como personal.

A nivel técnico, tras varias horas programando la aplicación y haciendo uso de los diferentes APIs de las que dispone OpenStack, llego a la conclusión de que es un gran acierto que OpenStack nos de la posibilidad de poder programar y de este modo adaptar a nuestras necesidades, automatizando trabajos que antes requerían cierto tiempo.

No solo estoy a favor de su uso en OpenStack si no a las miles de APIs que existen de diferentes servicios y que cada día se va generalizando como algo habitual a la hora de ofrecer un nuevo servicio.

A nivel personal son varias las conclusiones que se sacan con este proyecto. Para empezar y, quizá la mas importante, he cumplido mi objetivo de adquirir los conocimientos necesarios para poder manejar las diferentes APIs de OpenStack.

Además la realización de este proyecto me ha dado la oportunidad de saber mucho mas sobre OpenStack, algo que esta en estos momento esta en auge.

8. Bibliografía.

http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-es

http://www.ibm.com/developerworks/cloud/library/cl-openstack-pythonapis/index.html?ca=drs-

https://github.com/rajdeepd/openstack-samples/tree/master/lab1

https://github.com/stackforge/ospurge

https://github.com/iesgn/iesgncloud