UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

DSIC

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Cloud computing

Laboratorio 2

Contenido

1. Introducción

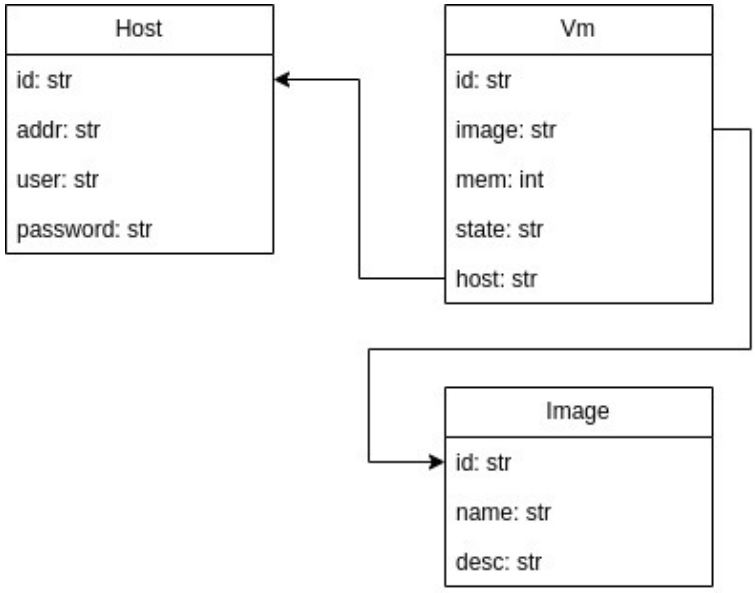
En este laboratorio continuamos mejorando el IaaS que se comenzó en el laboratorio 1. En concreto, se plantea incorporar las siguientes funcionalidades:

- Gestión de imágenes
- Control de usuarios
- Control de accesos
- Compartición de recursos
- Gestión de cuotas de usuario
- Publicación de funcionalidades por medio de un servicio RESTful
- CLI

En las siguientes secciones elaboramos estas funcionalidades.

2. Gestión de imágenes

En el laboratorio anterior se asumió la existencia de imágenes en el repositorio del IaaS. Llegó el momento de mejorar este aspecto. Para ello, se propone ampliar el modelo de datos del IaaS tal y como se describe en el siguiente diagrama.



Como se puede observar, la base de datos recogerá toda la información sobre las imágenes soportadas en el sistema. Será por tanto necesario ampliar la estructura de la base de datos, añadiendo la creación de una nueva tabla a la función *init()* definida en el laboratorio anterior.

Además, será necesario añadir nuevas operaciones de gestión de imágenes al módulo core.py. Se proponen las siguientes:

Operación	**Descripción**
addImage(url:str, img:dict)	Añade una nueva imagen al IaaS. El parámetro *url* especifica su ubicación origen. El parámetro *img* es un diccionario que contiene al menos los campos *name* y *desc*.
saveVmAsImage(vmId:str, img:dict)	Crea una nueva imagen a partir de una vm existente. El parámetro *vmId* especifica la vm. El parámetro *img* es un diccionario que contiene al menos los campos *name* y *desc*.

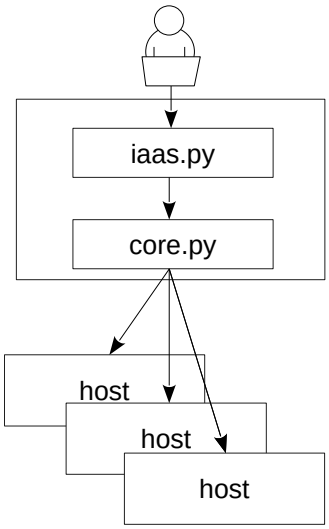
	Si la vm está en ejecución, se recomienda pararla previamente, para evitar pérdidas de datos en la imagen.
removeImage(imgId)	Elimina la imagen especificada.
listImages(query:str="")	Lista las imágenes que verifican el filtro especificado en *query*.

Es necesario recordar que las imágenes se almacenan en el repositorio */export/iaas/images* exportado en el IaaS. Se propone guardar las imágenes como ficheros *<imgId>.qcow2*.

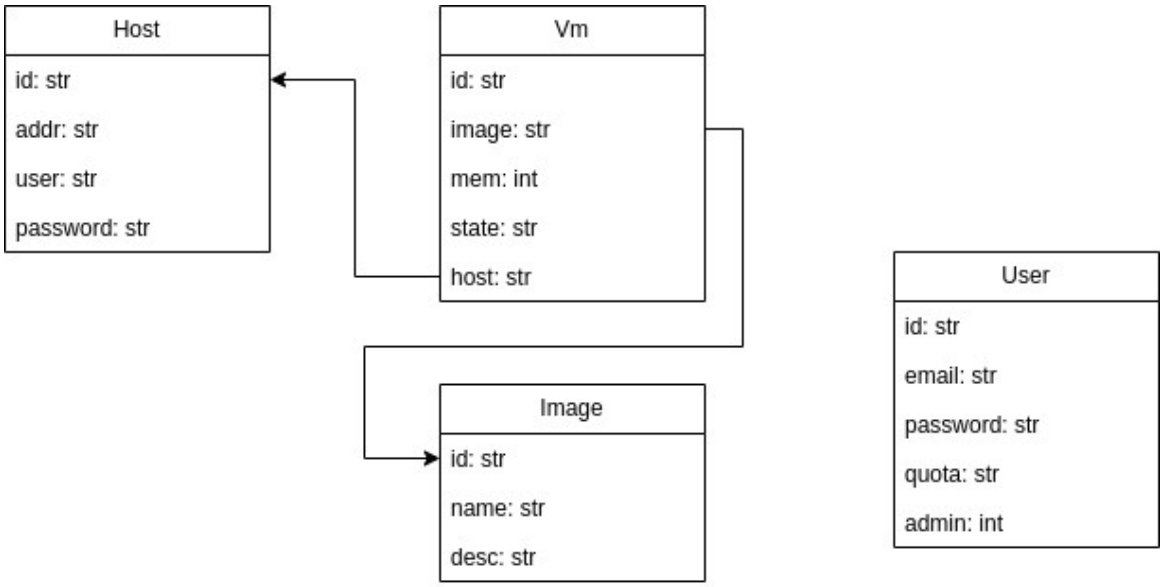
Para obtener una imagen a partir de una url, se recomienda utilizar la función *urllib.request.urlretrieve()*.

3. Gestión de usuarios

Hasta este punto, se dispone de un IaaS que permite gestionar recursos (hosts, vms e imágenes) a un nivel muy básico, sin ningún tipo de gestión de usuarios, ni control de accesos. Se propone recubrirlo con un nuevo módulo, que denominaremos *iaas.py*, que utilizará las funcionalidades del módulo *core.py*, y que añadirá funcionalidades avanzadas como gestión de usuarios, control de accesos, compartición de recursos, etc. La arquitectura del IaaS resultante sería la siguiente:



Para implementar la gestión de usuarios, será necesario ampliar el modelo de datos que maneja el IaaS, tal y como se refleja en el siguiente diagrama.



Como se puede observar, un usuario puede ser administrador. Un usuario administrador podrá acceder a todas las funcionalidades y a todos los recursos del IaaS. En concreto, sólo un administrador podrá:

- Añadir y eliminar usuarios.
- Añadir y eliminar hosts.

Se propone implementar las siguientes operaciones en el nuevo módulo *iaas.py*.

Operación	**Descripción**
init()	Inicializa el core del Iaas, invocando la función *core.init()* y crea las tablas necesarias para añadir funcionalidades avanzadas. Entre otras, la tabla *users*. También se asegurará que inicialmente existe al menos un usuario administrador (e.g. root/root)
login(email:str, password:str)	Autentica a un usuario. Si la operación tiene éxito, devuelve un token cifrado, que contiene al menos el id del usuario autenticado y la fecha de autenticación. El resto de operaciones de *iaas.py* debe aceptar como primer parámetro este token.
validateToken(token:str)	Valida el token especificado. Si la operación tiene éxito devuelve la información del usuario autenticado. Es una operación de utilidad, que será usada internamente por el resto de operaciones.
addUser(token:str, user:dict)	Crea un nuevo usuario. El parámetro *user* es un diccionario que al menos contiene *email* y *password*. Esta operación sólo puede ser invocada por un administrador. Si la operación tiene éxito, devuelve el usuario creado.
removeUser(token:str, userId:str)	Elimina el usuario especificado. Sólo puede ser invocada por un administrador. Podría conllevar la destrucción de todos los recursos asignados al usuario.
updateUser(token:str, userId:str, data:dict)	Actualiza el usuario especificado. Sólo puede ser invocada por un administrador o el propio usuario.
listUsers(token:str, query:str="")	Lista los usuarios especificados en el filtro *query*.

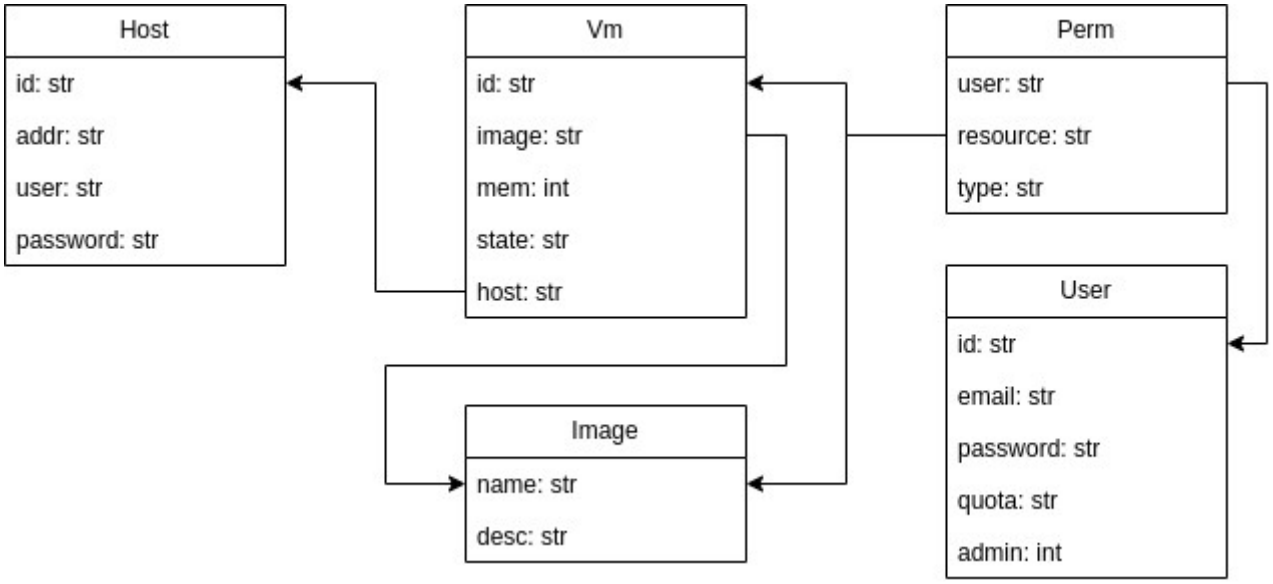
Para cifrar/descifrar el token se aconseja utilizar el paquete cryptocode, debido a su sencillez.

```
> pip install cryptocode

import cryptocode
myEncryptedMessage = cryptocode.encrypt("I like trains", "password123")
originalMessage = cryptocode.decrypt(myDecryptedMessage, "password123")
```

4. Control de accesos

En un IaaS serio los recursos están compartimentados en cuentas, y cada cuenta únicamente puede acceder a sus recursos. Vamos a implementar esta funcionalidad implantando un sistema de permisos en nuestro modelo de datos.



Como se puede observar, la entidad *Perm* representa un permiso que posee un usuario sobre un recurso. En principio, los recursos que maneja nuestro IaaS son imágenes y vms. Los hosts sólo pueden ser gestionados por un administrador. Con este mecanismo ya resulta sencillo conocer quién tiene acceso a qué. Siempre que un usuario no administrador acceda a imágenes o vms, primero será necesario consultar sus permisos. Un administrador siempre puede acceder a cualquier recurso.

Será necesario implementar las siguientes operaciones en el módulo *iaas.py*.

Operación	**Descripción**
addHost(token:str, host:dict)	Un administrador añade un host. Si es administrador, se invocará a *core.addHost()*.
listHosts(token:str, query:str="")	Cualquier usuario puede listar los hosts. Se invocará *core.listHosts().*
updateHost(token:str, hostId:str, data:dict)	Un administrador actualiza un host. Si es administrador, se invocará *core.updateHost().*
removeHost(token:str, hostId:str)	Un administrador elimina un host. Si es administrador, se invocará a *core.removeHost()*.
addVm(token:str, vm:dict)	Un usuario añade una vm a partir de una imagen. Si el usuario no es administrador debe tener permisos sobre la imagen origen. Si es el caso, entonces se invoca *core.addVm()*. Al finalizar será necesario añadir un nuevo permiso del usuario creador sobre la vm creada.
listVms(token:str, query:str="")	Un usuario lista sus vms. Si el usuario no es administrador, será necesario obtener sus permisos sobre vms. Después invocará *core.listVms()* buscando dichas vms.
startVm(token:str, vmId:str)	Un usuario arranca su vm. Si el usuario no es administrador, será necesario comprobar su permiso sobre la vm. Después invocará *core.startVm()*.

stopVm(token:str, vmId:str)	Un usuario para su vm. Si el usuario no es administrador, será necesario comprobar su permiso sobre la vm. Después invocará *core.stopVm()*.
removeVm(token:str, vmId:str)	Un usuario elimina su vm. Si el usuario no es administrador, será necesario comprobar su permiso sobre la vm. Después invocará *core.removeVm()*. Deberán eliminarse todos los permisos existentes sobre la vm.
saveVmAsImage(token:str, vmId:str, img:dict)	Un usuario guarda su vm como nueva imagen. Si el usuario no es administrador, será necesario comprobar su permiso sobre la vm. Después invocará *core.saveVmAsImage()*. Deberá añadirse un nuevo permiso del usuario sobre la imagen creada.
addImage(token:str, url:str, img:dict)	Un usuario añade una imagen. Para ello, invoca *core.addImage()*. Al finalizar será necesario añadir un nuevo permiso del usuario creador sobre la imagen creada.
listImages(token:str, query:str="")	Un usuario lista sus imágenes. Si el usuario no es administrador, será necesario obtener sus permisos sobre imágenes. Después invocará *core.listImages()* buscando dichas imágenes.
removeImage(token:str, imgId:str)	Un usuario elimina su imagen. Si el usuario no es administrador, será necesario comprobar su permiso sobre la imagen. Después invocará *core.removeImage()*. Deberán eliminarse todos los permisos existentes sobre la imagen.

5. Compartición de recursos

Una vez implantado el sistema de permisos descrito en el apartado anterior, resulta sumamente sencillo compartir recursos entre distintos usuarios. En este escenario, compartir un recurso únicamente implicará crear un permiso adicional. De este modo, se propone implementar las siguientes operaciones en el módulo *iaas.py*:

Operación	**Descripción**
shareVm(token:str, vmId:str, userId:str)	Un usuario comparte su vm. Si el usuario no es administrador, será necesario comprobar su permiso sobre la vm. Después creará el permiso especificado.
unshareVm(token:str, vmId:str, userId:str)	Un usuario deja de compartir su vm. Si el usuario no es administrador, será necesario comprobar su permiso sobre la vm. Después eliminará el permiso especificado.
listVmShares(token:str, vmId:str)	Lista los permisos sobre una vm. Si el usuario no es administrador, será necesario comprobar su permiso sobre la vm.
shareImage(token:str, imgId:str, userId:str)	Un usuario comparte su imagen. Si el usuario no es administrador, será necesario comprobar su permiso sobre la imagen. Después creará el permiso especificado.
unshareImage(token:str, imgId:str, userId:str)	Un usuario deja de compartir su imagen. Si el usuario no es administrador, será necesario comprobar su permiso sobre la imagen. Después eliminará el permiso especificado.
listImageShares(token:str, imgId:str)	Lista los permisos sobre una imagen. Si el usuario no es administrador, será necesario comprobar su permiso sobre la imagen.

6. Gestión de cuotas de usuarios

En un IaaS serio cada cuenta posee unos limites de recursos que no puede superar. En esta sección se propone diseñar un sistema de cuotas que permita especificar estos límites, sobre cada usuario, y que el IaaS compruebe que los límites no se exceden. En caso de ser excedidos, el IaaS rechazará las operaciones que involucren reserva de nuevos recursos.

A continuación se listan algunas métricas que se podrían tener en cuenta en la cuotas:

- Número de imágenes creadas.
- Número de vms creadas.
- Tiempo de CPU consumido.
- Memoria consumida.
- Bytes leídos, bytes escritos en disco.
- Bytes leídos, bytes escritos en red.
- etc.

Es sencillo recoger muchas de estas métricas desde Libvirt, utilizando para ello un código parecido al siguiente, que ha sido extraído de la siguiente URL:

https://libvirt-python.readthedocs.io/monitoring-performance/

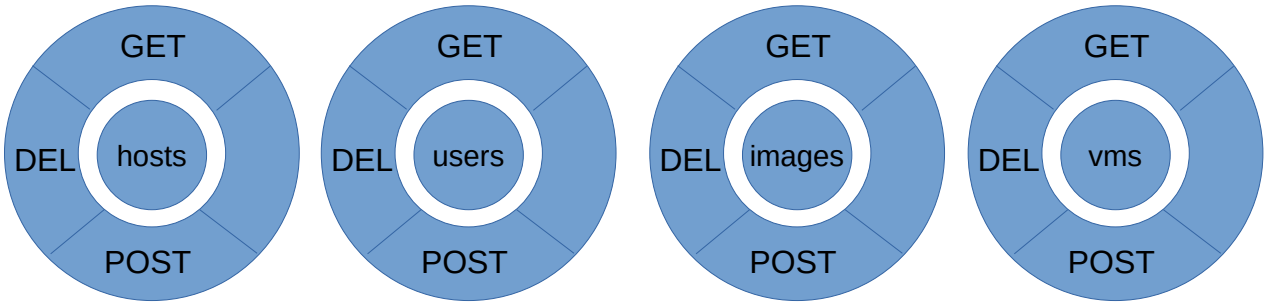
```
import libvirt

conn = libvirt.open("qemu:///system")
dom = conn.lookupByID(1)
cpu_stats = dom.getCPUStats(False)
for (i, cpu) in enumerate(cpu_stats):
   print("CPU {} Time: {}".format(i, cpu["cpu_time"] / 1000000000.))
stats = dom.getCPUStats(True)
print("cpu_time:    {cpu_time}\n"
      "system_time: {system_time}\n"
      "user_time:   {user_time}".format(**stats[0]))
stats = dom.memoryStats()
print("memory used:")
for name in stats:
    print("{} ({})".format(stats[name], name))

tree = ElementTree.fromstring(dom.XMLDesc())
iface = tree.find("devices/interface/target").get("dev")
stats = dom.interfaceStats(iface)
print("read bytes:    {}\n"
      "read packets:  {}\n"
      "read errors:   {}\n"
      "read drops:    {}\n"
      "write bytes:   {}\n"
      "write packets: {}\n"
      "write errors:  {}\n"
      "write drops:   {}\n".format(*stats))
```

7. Servicio RESTful

En esta sección se propone publicar toda la funcionalidad del IaaS por medio de un servicio RESTful. En concreto, el servicio proporcionará acceso a 4 colecciones de recursos principales: hosts, usuarios, imágenes y vms.

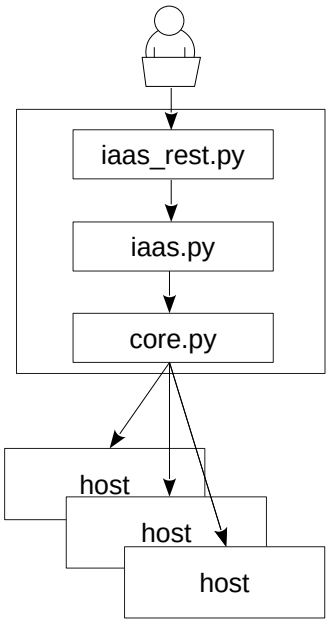


A continuación se resume la API del servicio RESTful a implementar:

Método	URL	Comentarios
Recurso sessions		
`POST`	`/iaas/sessions`	Crea una sesión. Contenido (JSON): {email,password} Resultado (text): token
Recurso hosts		
`GET`	`/iaas/hosts`	Lista los hosts. Cabeceras: Authorization:token Parámetros: query Resultado (JSON): [host]
`POST`	`/iaas/hosts`	Crea un host. Cabeceras: Authorization:token Contenido (JSON): host Resultado (JSON): host
`PUT`	`/iaas/hosts/<hostId>`	Actualiza un host. Cabeceras: Authorization:token Contenido (JSON): data
`DELETE`	`/iaas/hosts/<hostId>`	Elimina un host Cabeceras: Authorization:token
Recurso users		
`GET`	`/iaas/users`	Lista los usuarios. Cabeceras: Authorization:token Parámetros: query Resultado (JSON): [user]
`POST`	`/iaas/users`	Crea un usuario. Cabeceras: Authorization:token Contenido (JSON): user Resultado (JSON): user
`PUT`	`/iaas/users/<userId>`	Actualiza un usuario. Cabeceras: Authorization:token Contenido (JSON): data
`DELETE`	`/iaas/users/<userId>`	Elimina un usuario. Cabeceras: Authorization:token

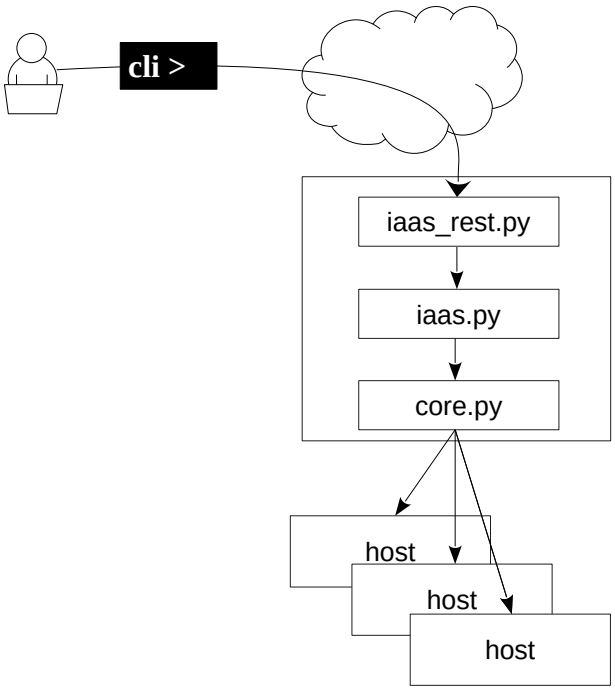
Recurso imágenes		
GET	/iaas/images	Lista las imágenes. Cabeceras: Authorization:token Parámetros: query Resultado (JSON): [image]
POST	/iaas/images	Crea una imagen. Cabeceras: Authorization:token Contenido (JSON): image Resultado (JSON): image
PUT	/iaas/images/<imgId>	Actualiza una imagen. Cabeceras: Authorization:token Contenido (JSON): data
DELETE	/iaas/images/<imgId>	Elimina una imagen. Cabeceras: Authorization:token
GET	/iaas/images/<imgId>/shares	Lista los permisos sobre una imagen. Cabeceras: Authorization:token Resultado (JSON): [perm]
POST	/iaas/images/<imgId>/shares	Crea un permiso sobre una imagen. Cabeceras: Authorization:token Contenido (JSON): {user}
DELETE	/iaas/images/<imgId>/shares/<userId>	Elimina un permiso sobre una imagen. Cabeceras: Authorization:token
Recurso vms		
GET	/iaas/vms	Lista las vms. Cabeceras: Authorization:token Parámetros: query Resultado (JSON): [vm]
POST	/iaas/vms	Crea una vm. Cabeceras: Authorization:token Contenido (JSON): vm Resultado (JSON): vm
PUT	/iaas/vms/<vmId>	Actualiza una vm. Cabeceras: Authorization:token Contenido (JSON): data
DELETE	/iaas/vms/<vmId>	Elimina una vm. Cabeceras: Authorization:token
GET	/iaas/vms/<vmId>/shares	Lista los permisos sobre una vm. Cabeceras: Authorization:token Resultado (JSON): [perm]
POST	/iaas/vms/<vmId>/shares	Crea un permiso sobre una vm. Cabeceras: Authorization:token Contenido (JSON): {user}
DELETE	/iaas/vms/<vmId>/shares/<userId>	Elimina un permiso sobre una vm. Cabeceras: Authorization:token
POST	/iaas/vms/<vmId>/snapshots	Crea una imagen a partir de una máquina virtual. Cabeceras: Authorization:token Contenido (JSON): image Resultado (JSON): image

La API será implementada en el módulo *iaas_rest.py*, utilizando *Flask*, dando lugar a la siguiente arquitectura.



8. El CLI

Por último, el servicio RESTful será accedido por un CLI, que suministrará una interfaz basada en consola, y que hará uso de la librería *requests.*



A continuación se muestra un posible listado de opciones disponibles a través de esta herramienta:

```
Usage: python cli.py
Welcome!
cli> help
Available commands:
    help
    exit
    login <email> <password>
    host ls [<query>]
    host add <addr> <user> <password>
    host rm <hostId>
    user list [<query>]
    user add <email> <password>
    user rm <userId>
    image ls [<query>]
    image add <url> <name> <desc>
    image rm <imgId>
    image shares <imgId>
    image share <imgId> <userId>
    image unshare <imgId> <userId>
    vm ls [<query]
    vm add <image> [<mem>]
    vm start <vmId>
    vm stop <vmId>
    vm rm <vmId>
    vm save <vmId> <imgName> <imgDesc>
    vm shares <vmId>
    vm share <vmId> <userId>
    vm unshare <vmId> <userId>
```