UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

DSIC

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

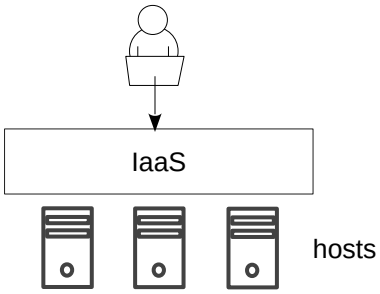
Cloud computing

Laboratorio 1

Contenido

1. Introducción

En este proyecto se implementará un sencillo IaaS, capaz de gestionar los recursos virtuales suministrados por un clúster de hosts.



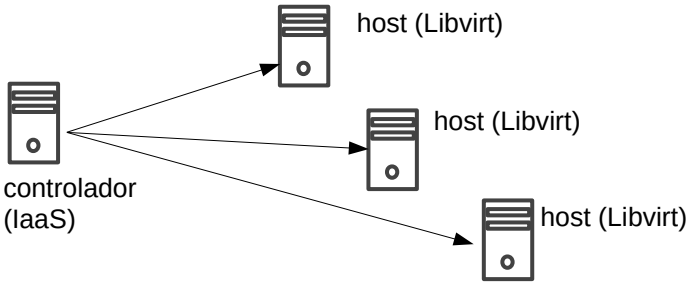
Para ello, el IaaS se irá construyendo de manera incremental a lo largo de dos laboratorios. Los objetivos de este primer laboratorio se listan a continuación:

- Gestión de hosts, utilizando Ansible.
- Gestión de volúmenes virtuales, utilizando NFS.
- Gestión de máquinas virtuales, utilizando Libvirt.

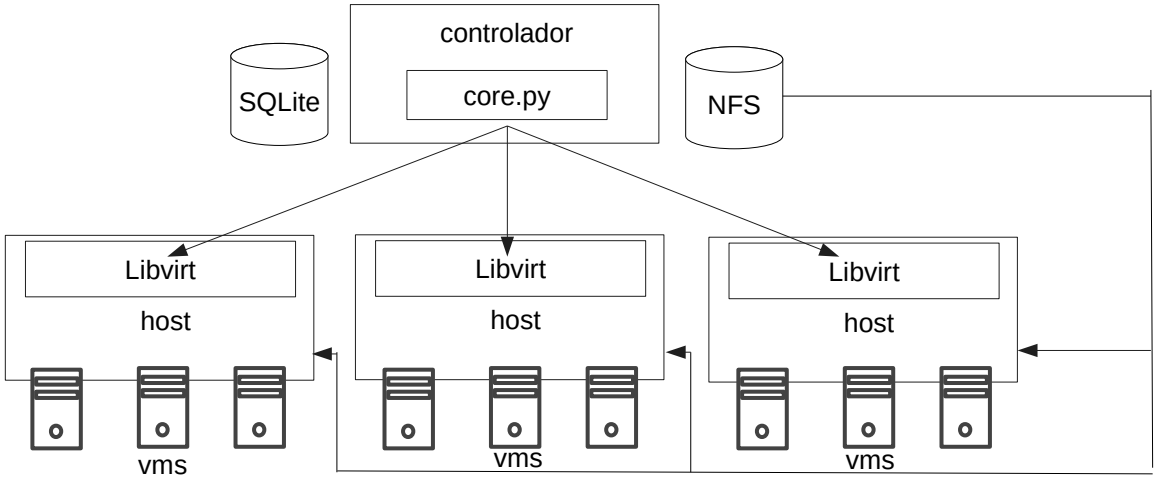
En las siguientes secciones abordaremos todos estos aspectos.

2. Arquitectura del sistema

A nivel global, se dispone de un nodo controlador, que ejecuta el IaaS, y una colección de *hosts* que ejecutan Libvirt. El IaaS gestiona los recursos disponibles en los distintos hosts.



Se propone una arquitectura similar a la siguiente:

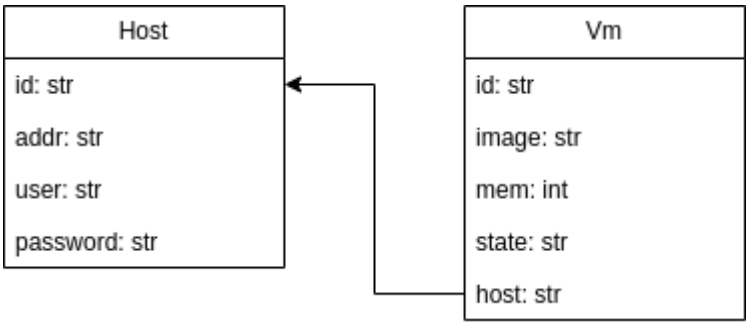


Como se puede observar, el módulo *core.py* se ejecuta en el nodo controlador, e implementa el corazón del sistema. Se comunica con los distintos hosts, para gestionar sus recursos virtuales. Para ello, *core.py* dispone de un espacio de almacenamiento privado, una base de datos SQLite, donde almacenará toda la información sobre los hosts y las máquinas virtuales creadas.

Los hosts deben ejecutar el demonio Libvirt, y serán los responsables de crear las máquinas virtuales, por orden del controlador. Tanto las máquinas virtuales como las imágenes de las que se crean serán almacenadas en un espacio de almacenamiento que reside en el nodo controlador, que recibirá el nombre de repositorio del IaaS y que es compartido por NFS.

3. SQLite

A continuación se presenta el modelo de datos básico que el IaaS debería considerar. Este modelo de datos será ampliado más adelante.



Como se puede observar, el IaaS controla una colección de hosts. Cada host debe poseer una dirección (IP, DNS), un usuario y un password. El usuario y el password deben tener privilegios sudo, porque los utilizaremos para efectuar operaciones sensibles.

Cada máquina virtual se ejecuta en un host, y se crea a partir de una imagen, tiene una cantidad de memoria asignada y se encuentra en un estado (“running”, “stopped”). Por el momento, cada imagen tendrá un nombre, y residirá en el repositorio del IaaS (el directorio compartido por NFS).

Para dialogar con el almacén de datos utilizaremos el módulo *sqlite3* que forma parte de la distribución estándar de Python. A continuación se muestra un código que lo utiliza:

```
import sqlite3

con = sqlite3.connect("iaas.db")
cur = con.cursor()
cur.execute("create table hosts(…)")
cur.execute("insert into hosts values (…)")
rows = cur.fetchall()
for row in rows: print(row[0],row[1],row[2],...)
con.commit()
con.close()
```

Se puede encontrar más información sobre este módulo en el siguiente link:

https://docs.python.org/es/3/library/sqlite3.html

Se recomienda definir una función *init()* en el módulo *core.py*, que incluya el código de inicialización de la base de datos SQLite.

4. Gestión de hosts

El nodo controlador mantiene un control de todos los hosts que forman parte del clúster.

Para ello, el nodo controlador deberá exportar un directorio vía NFS que contendrá todas las imágenes y máquinas virtuales del IaaS, lo denominaremos el repositorio del IaaS. Asumiremos la siguiente estructura básica de directorios:

```
/export/
   |- iaas/
      |- images/
      |- vms/
```

Cuando un nuevo host sea añadido al sistema, será necesario **prepararlo**. Ello implica:

1. Asegurarse de que Libvirt se está ejecutando en su interior.
2. Asegurarse de que el host monta el repositorio del IaaS, por ejemplo en un directorio local */mnt/iaas*

Para garantizar que se cumplen todas las condiciones anteriores, utilizaremos diversos playbooks en Ansible. Se recomienda almacenarlos en un directorio *./plays*:

```
./plays
    |- iaas_init.yaml           # para inicializar el IaaS
    |- iaas_destroy.yaml        # para “desinicializar” el IaaS
    |- host_add.yaml            # para añadir un host al IaaS
    |- host_remove.yaml         # para eliminar un host del IaaS
```

Para trabajar con Ansible desde Python podemos utilizar el paquete *Ansible Runner*:

https://ansible-runner.readthedocs.io/en/stable/

El primer paso consiste en instalar el paquete en un entorno virtual de Python:

```
(venv) > pip install ansible-runner
```

Una vez instalado, se importará el módulo *ansible_runner*.

```
import ansible_runner
```

Es muy sencillo trabajar con este módulo. Únicamente hay que comprender la operación *ansible_runner.interface.run().* Por ejemplo, para ejecutar un módulo addhoc:

```
r = ansible_runner.interface.run(host_pattern="localhost", module="ping")
```

A continuación se lista un ejemplo que ejecuta un playbook, definiendo el inventario y el host sobre el que se desea ejecutar:

```
r = ansible_runner.interface.run(host_pattern="test",
   inventory="test ansible_host=cca-024-node2 ansible_port=22",
   playbook="/home/alumno/cloud/lab/lab1/dev/test.yml")
```

A continuación se listan los parámetros más relevantes que acepta esta operación:

- *host_pattern*: el/los target/s de la operación
- *inventory*: el inventario, un string en formato ini, o un diccionario en formato YAML
- *playbook*: el path absoluto del playbook a ejecutar
- *extravars:* un diccionario con las variables a pasar a Ansible
- *module*: el módulo a ejecutar (ej. ping, user, etc.)
- *module_args*: los argumentos del módulo a ejecutar en una línea string

Para obtener más información se recomienda acceder a la documentación u obtener la ayuda en linea:

```
help(ansible_runner.interface.run)
```

A continuación se propone la lista de operaciones a implementar en el módulo *core.py* relacionadas con la gestión de hosts.

`core.init()`

Inicializa el nodo controlador. Ello implica, al menos:

1. Inicializar la base de datos SQLite.
2. Garantizar que el repositorio del IaaS */export/iaas* ha sido exportado por NFS.

Para (2) se recomienda añadir algunas tareas al playbook *./plays/iaas_init.yaml* que se aseguren que NFS está instalado localmente, y que exporta el repositorio */export/iaas*. Serán de utilidad los módulos *apt*, *file*, *lineinfile* y *systemd*. Por ejemplo:

```
- name: Install packages
    apt: name={{ item }} state=present update_cache=yes
    with_items:
    - libvirt-daemon-system
    - libvirt-clients
    - python3-libvirt
    - nfs-kernel-server
- name: Create IaaS folders
    file:
      path: "/export/iaas/{{ item }}"
      state: directory
      owner: nobody
      group: nogroup
      mode: 0777
    with_items:
      - images
      - vms
  - name: Publish IaaS folders
    lineinfile:
      path: /etc/exports
      state: present
      line: "/export/iaas  *(rw,sync,all_squash,no_subtree_check,insecure)"
  - name: Restart NFS
    systemd:
      daemon_reload: yes
      state: started
      name: nfs-kernel-server.service
  - name: Export share
    command: exportfs -a
```

`core.destroy()`

Debería de hacer lo opuesto a *core.init()*. Esta operación destruye el IaaS. Implica mucho trabajo de destrucción, en especial cuando el IaaS tenga múltiples hosts y estos tengan múltiples máquinas virtuales. Ejecutará el playbook *./plays/iaas_destroy.yaml.*

`core.addHost(host:dict)`

Añade un nuevo host al IaaS. Antes será necesario preparar al host. Para ello se utilizará el playbook *./plays/host_add.yaml*. Ya se ha comentado anteriormente qué implica preparar a un host. Serán de utilidad los módulos *apt* (para instalar Libvirt, cliente NFS), *file* (para crear el directorio local de montaje) y *mount* (para montar el repositorio del IaaS en el directorio local).

Una vez el host haya sido correctamente preparado, será necesario actualizar la base de datos.

`core.listHosts(query:str='')`

Lista los hosts que forman parte del IaaS. Bastará con consultar la base de datos local. El parámetro *query* permitirá filtrar los resultados. Para simplificar, se podría aprovechar la misma sintaxis de SQL (aunque no sea muy seguro ;-)).

`core.removeHost(hostId:str)`

Elimina un host del IaaS. Esto implica dejarlo en su estado original. Se hará uso del playbook *./plays/node_remove.yaml*, que efectuará las operaciones opuestas a *./plays/node_add.yaml*.
Además, si el host dispone de máquinas virtuales, habría que eliminarlas previamente.
Si la operación tiene éxito, se actualizará la base de datos convenientemente.

`core.updateHost(hostId:str, data:dict)`

Permite actualizar los datos del host.

5. Gestión de máquinas virtuales

Una vez disponemos de un clúster de hosts con Libvirt, podemos ejecutar máquinas virtuales en su interior.

Para trabar con Libvirt desde Python utilizaremos el módulo *libvirt-python*. La documentación sobre esta API la podemos encontrar en los siguientes links:

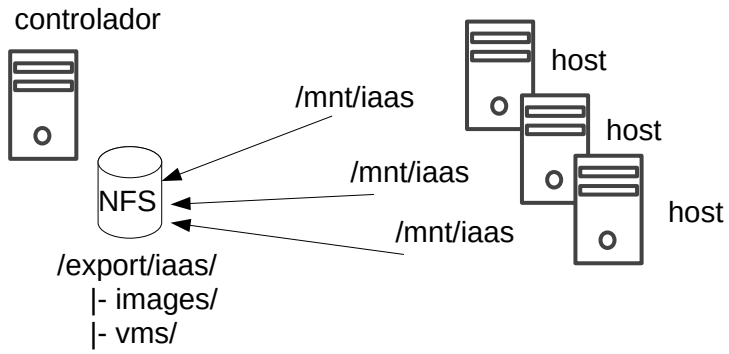
https://libvirt-python.readthedocs.io/
https://libvirt.gitlab.io/libvirt-appdev-guide-python/index.html

A continuación se muestra un ejemplo de código que se conecta a una máquina remota utilizando SSH:

https://wiki.libvirt.org/SSHSetup.html

```
con = libvirt.open("qemu+ssh://alumno@cca-xxx-node1/system")
doms = con.listAllDomains()
for dom in doms: print(dom.ID())
dom = con.lookupByName(name)
dom = con.createXML(template, 0)  # crea y arranca una máquina virtual
dom.shutdown()
dom.destroy()
```

Como se ha comentado anteriormente, el nodo controlador publica vía NFS un repositorio donde se encuentran todas las imágenes y máquinas virtuales. Los distintos hosts montan este repositorio. La siguiente figura ilustra esta disposición.



Cuando se añade una nueva máquina virtual al sistema, es necesario especificar la imagen a partir de la cual se creará. Esta imagen debe estar disponible en el repositorio */export/iaas/images*. Entonces, se creará una copia de la imagen en el repositorio */export/iaas/vms*, será el disco virtual de la máquina virtual. Después, se instruirá a un host para que cree una máquina virtual a partir de dicho disco virtual.

La máquina virtual se debe de crear a partir de una especificación en .xml. Para generar esta especificación se recomienda crear en el nodo controlador una plantilla *./template.xml* con ciertas constantes (e.g. ID, MEM, DISK). Estas constantes serán posteriormente reemplazadas por valores concretos a la hora de generar dicha especificación.

A continuación se propone la lista de operaciones a implementar en el módulo *core.py* relacionadas con la gestión de máquinas virtuales.

`core.addVm(vm:dict)`

Crea una nueva máquina virtual en un host.
Se selecciona un host en el que se debe ejecutar la máquina virtual.
Se copia la imagen definida en vm[“image”] del directorio */export/iaas/images* al directorio */export/iaas/vms* con un nombre igual a vm[“id”].
Se lee la plantilla *./template.xml* y se modifican las configuraciones específicas de la máquina virtual (el id, la mem, el disk, etc.)
Se crea el dominio con *libvirt.defineXML()*.
Se guardan los cambios en la base de datos.

`core.startVm(vmId:str)`

Arranca la máquina virtual especificada.
Se busca el dominio con libvirt.lookupByName().
Se arranca el dominio con libvirt.create().
Se guardan los cambios en la base de datos.

`core.stopVm(vmId:str)`

Para la máquina virtual especificada.
Se busca el dominio con libvirt.lookupByName().
Se para el dominio con libvirt.shutdown().
Se guardan los cambios en la base de datos.

`core.removeVm(vmId:str)`

Elimina la máquina virtual especificada.
Se busca el dominio con libvirt.lookupByName().
Se para el dominio con libvirt.undefine().
Se elimina el disco duro virtual de la máquina virtual de */export/iaas/vms*.
Se guardan los cambios en la base de datos.

`core.listVms(query:str='')`

Lista las máquinas virtuales que forman parte del IaaS. Bastará con consultar la base de datos local. El parámetro *query* permitirá filtrar los resultados. Para simplificar, se podría aprovechar la misma sintaxis de SQL (aunque no sea muy seguro ;-)).