UNIVERSITAT POLITÈCNICA DE VALÈNCIA

CAMPUS D'ALCOI

DSIC

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Cloud computing

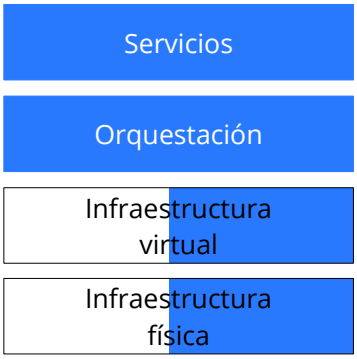
Laboratorio 2

Contenido

1. Introducción

En este laboratorio se diseñará un sencillo IaaS, que haga uso de las primitivas implementadas en el laboratorio 1.

Como ya se ha visto en teoría, para implementar un IaaS es necesario abordar las siguientes capas:



La capa de infraestructura física y virtual ha sido implementada parcialmente en el laboratorio 1, que permite gestionar los recursos virtuales de un único nodo.

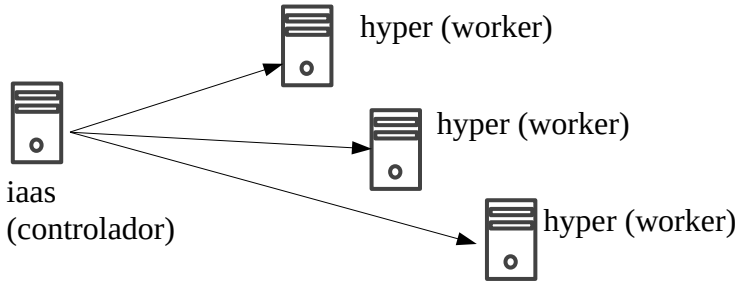
En este laboratorio es necesario implementar las siguientes funcionalidades:

- Extender la capa de *infraestructura física*, para que se puedan gestionar múltiples nodos.
- Extender la capa de *infraestructura virtual*, para que se puedan gestionar recursos virtuales procedentes de diversos nodos.
- Implementar la capa de *orquestación*, que será responsable de:
  - Gestionar y monitorizar los nodos.
  - Gestionar y monitorizar las máquinas virtuales.
  - Planificar la distribución de máquinas virtuales entre los distintos nodos atendiendo a cierta política.
- Implementar la capa de *servicios*, que publicará una API para consumir los recursos del IaaS.

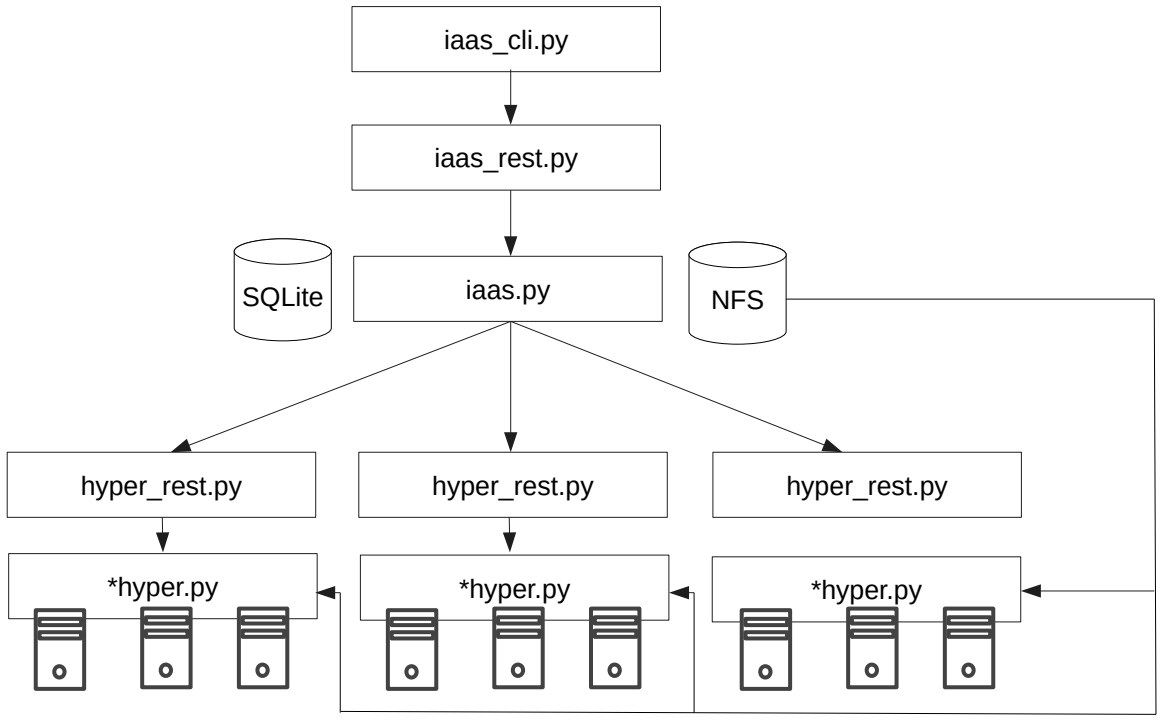
En la sección 2 se presenta la arquitectura del sistema que se pretende implementar. En la sección 3 se describe el modelo de datos que implementará el IaaS, así como su API y los comandos CLI que permitirán gestionarlo. En la sección 4 se discute cómo efectuar la gestión de nodos. En la sección 5 se presenta la gestión de imágenes. Por último, en la sección 6 se presenta la problemática de la gestión de máquinas virtuales.

2. Arquitectura del sistema

A nivel global, se dispone de un nodo controlador, que ejecuta el *iaas*, y una colección de nodos worker, que ejecutan *hyper*, implementado en el laboratorio 1. El *iaas* gestiona los recursos publicados por todos los *hyper*.



Para ello, se propone una arquitectura similar a la siguiente:



Como se puede observar, será necesario implementar los siguientes módulos nuevos:

- *iaas.py*: implementa la funcionalidad del IaaS.
- *iaas_rest.py*: publica la funcionalidad del IaaS como un servicio RESTful.
- *iaas_cli.py*: herramienta CLI para acceder a la funcionalidad publicada por el servicio RESTful.

Además, será necesario modificar algunos aspectos del módulo *hyper.py* (por eso aparece un *), por ejemplo para permitir la conexión entre máquinas virtuales pertenencientes a distintos nodos, o para acceder a las imágenes disponibles en un directorio compartido por NFS.

3. El modelo

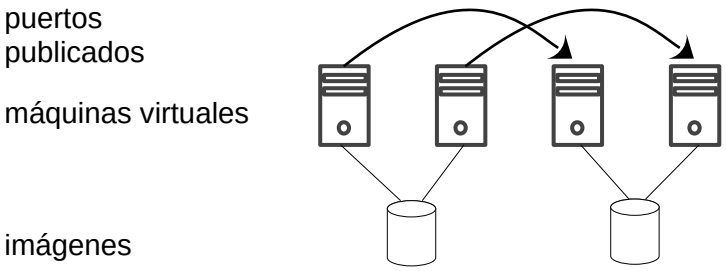
El IaaS a implementar será muy sencillo, y permitirá gestionar los siguientes recursos:

- **Nodos**: los distintos nodos físicos que conforman la infraestructura física.
- **Imágenes**: las imágenes de disco a partir de las que se podrán crear máquinas virtuales.
- **Máquinas virtuales**: las máquinas virtuales que solicita el cliente, y que pueden ejecutarse en cualquier nodo físico.

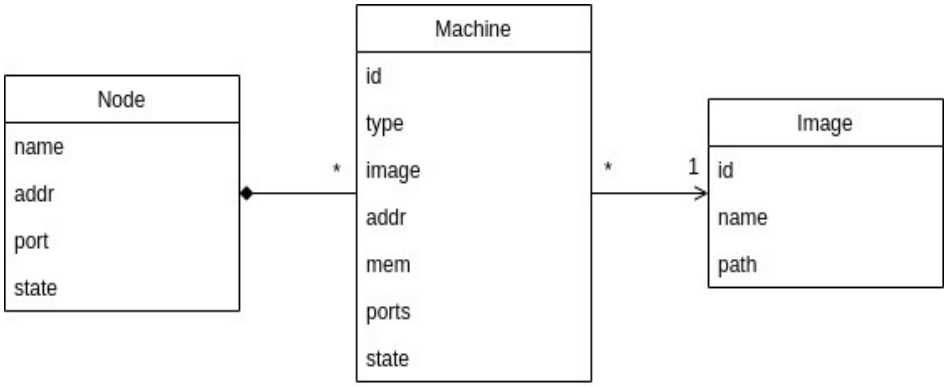
Cada **máquina virtual** se crea a partir de una **imagen**. El sistema dispone de una colección de imágenes predefinidas. Podrán registrarse nuevas imágenes en el sistema.

Las máquinas virtuales podrán comunicarse con el exterior. Además, una máquina virtual podrá publicar uno o más puertos en el host en el que se ejecuta. De este modo, cualquier otra máquina virtual o cliente externo podrá acceder a ella.

Por tanto, el esquema que el cliente posee en mente sería el siguiente:

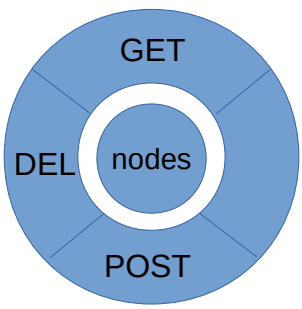


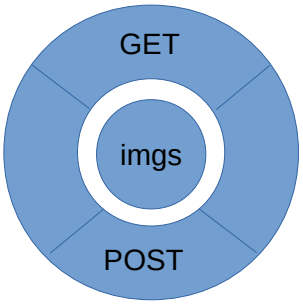
A nivel interno, el modelo de datos con el que podría trabajar el IaaS sería parecido al siguiente:

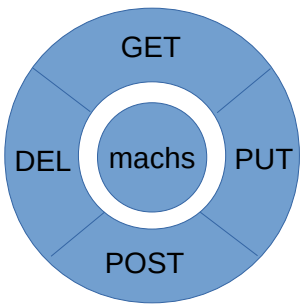


Por tanto, la API RESTful que publicará el módulo *iaas_rest.py* podrá gestionar tres tipos de recursos principales:

- */iaas/nodes*: los nodos
- */iaas/images*: las imágenes
- */iaas/machines*: las máquinas







A continuación se resume la API del servicio RESTful que deberá publicar:

Método	URL	Comentarios
Recurso nodes		
GET	/iaas/nodes	Recupera los nodos Parámetros: query Resultado (JSON): [node]
GET	/iaas/nodes/<name>	Recupera el nodo con nombre <name> Resultado (JSON): node
POST	/iaas/nodes	Añade un nuevo nodo Contenido (JSON): node Resultado (JSON): node
DELETE	/iaas/nodes/<name>	Elimina el nodo con nombre <name>
Recurso images		
GET	/iaas/images	Recupera las imágenes Parámetros: query Resultado (JSON): [image]
POST	/iaas/images	Añade una nueva imagen Contenido (JSON): image Resultado (JSON): image
Recurso machines		
GET	/iaas/machines	Recupera las máquinas virtuales Parámetros: query Resultado (JSON): [machine]
GET	/iaas/machines/<mid>	Recupera la máquina virtual con id <mid> Resultado (JSON): machine
POST	/iaas/machines	Crea una nueva máquina virtual Contenido (JSON): machine Resultado (JSON): machine
PUT	/iaas/machines/<mid>	Actualiza el estado de la máquina con id <mid> Contenido (JSON): {“state”:<estado>”} Resultado (JSON): machine
DELETE	/iaas/machines/<mid>	Elimina la máquina con id <mid>

Por último, el módulo *iaas_cli.py* suministrará una interfaz CLI basada en consola, que consumirá el servicio RESTful. A continuación se muestra un posible listado de opciones disponibles a través de esta herramienta:

```
Usage: python iaas_cli.py <command> [options]
Commands:
    node add <name> <addr:port>    # each node listens to different port
    node rm <name>
    node ls
    img add <imageName> <file>   # add image from file
    img ls
    vm add <imgName> [-t <type>] [-m <mem>] [-n <node> {-p <nodePort>:<vmPort>}]
    vm start <machId>
    vm stop <machId>
    vm rm <machId>
    vm ls {field=value}
```

4. Gestión de nodos

Los comandos disponibles para gestionar nodos desde la perspectiva del usuario son los siguientes:

```
node add <name> <addr:port>    # each node listens to different port
node rm <name>
node ls
```

Al añadir un nuevo nodo se deben suministrar los parámetros <name> y <addr:port>. Esto posibilitará arrancar múltiples nodos en el mismo host.

Cuando se añade un nuevo nodo al IaaS es necesario instalar *hyper* y todas sus dependencias (paquetes como libvirt, etc.), inicializar el nodo de proceso y finalmente arrancar *hyper*.

Se trata por tanto de una secuencia de pasos que se deben de ejecutar en una máquina remota. Se usará para ello Ansible desde Python. En concreto, se utilizará 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 ejempo, 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=beta.alc.upv.es ansible_port=10022",
   playbook="/home/alumno/cloud/lab/lab2/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 formati ini, o un diccionario en formato YAML
- *playbook*: el path absoluto del playbook a ejecutar
- *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 obtener la ayuda en linea:

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

A continuación se lista una guía rápida de implementación de las operaciones del módulo *iaas.py* relacionadas con la gestión de nodos.

`iaas.addNode(node)`

Se ejecuta un playbook */playbooks/addNode.yml* sobre el nodo que instala y configura *hyper*

1. Instalar con el módulo *apt* todos los paquetes requeridos en el nodo de proceso, por ejemplo: *libvirt-daemon-system*, *libvirt-clients*, *rpcbind, nfs-common.*

   ```
   - name: Install dependecies.
     apt: name={{ item }} state=present update_cache=yes
     with_items:
     - libvirt-daemon-system
     - libvirt-clients
       ....
   ```

2. Copiar todos los scripts de hyper al nodo, usando el módulo *copy*. Por ejemplo:

   ```
   - name: Copy files
     copy:
       src: "/source/path/to/hyper/"
       dest: "/destination/path/to/hyper/"
   ```

3. Ejecutar *hyper* en modo demonio (arrancar el servicio RESTful *hyper_rest.py*). Para ejecutar en modo demonio se puede usar start-stop-daemon, daemon, o bien nohup, Por ejemplo:

   ```
   - name: Start daemon
     shell: nohup myexeprogram arg1 arg2 &
   ```

Se inserta la info del nuevo nodo en SQLite.

`iaas.listNodes()`

Se listan los nodos, recuperando la info de SQLite.

`iaas.removeNode(name)`

Se elimina el nodo. Podría ejecutarse un playbook */playbooks/removeNode.yml* sobre el nodo que se elimina, que haga las operaciones inversas a */playbooks/addNode.yml* y dejar el nodo “limpio”.

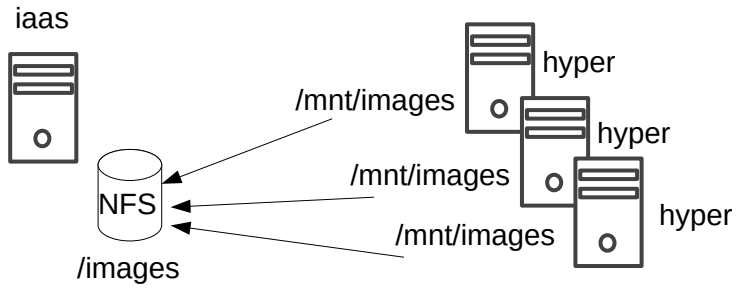
En las secciones siguientes ser verá que es necesario efectuar más labores de inicialización al añadir un nuevo nodo al sistema (por ejemplo, inicialización relativa a las imágenes, o inicialización relativa a la red).

5. Gestión de imágenes

Los comandos disponibles para gestionar imágenes desde la perspectiva del usuario son los siguientes:

```
img add <imageName> <file>   # add image from file
img ls
```

Para gestionar las imágenes, el nodo de control (*iaas.py*) publicará un espacio de almacenamiento virtualizado. Para ello, se propone compartir un directorio a través de NFS. Este directorio compartido será montado por los distintos nodos de proceso donde se ejecuta *hyper.py*.



Nodo de control
Para ello, en el nodo de control (donde se ejecuta iaas.py) será necesario instalar el servidor NFS:

```
> sudo apt install nfs-kernel-server
```

En el fichero */etc/exports* del nodo de control se añadirá la siguiente línea:

```
/images *(rw,sync,all_squash,no_subtree_check,insecure)
```

A continuación, se cambian los permisos del directorio y se exporta el directorio compartido:

```
> sudo chown nobody:nogroup /images
> sudo exportfs -a
> sudo systemctl restart nfs-kernel-server
```

Cuando se añada una nueva imagen al IaaS, se guardará en este directorio.

Nodos de proceso
Los nodos de proceso donde se ejecuta *hyper.py* deberán de montar este directorio compartido, para poder efectuar copias de las imágenes en local y arrancar las máquinas virtuales.

Para montar el directorio compartido:

```
> sudo mkdir /mnt/images
> sudo mount -t nfs <ip-nodo-control>:/images /mnt/images
```

Lo lógico es que esta acción se efectuara una única vez al añadir el nodo de proceso al sistema. Es decir, sería interesante añadir nuevas tareas al playbook */playbooks/addNode.yml.* Por ejemplo:

```
- name: Create mountable dir
  file: path=/mnt/images state=directory mode=777 owner=root group=root

- name: set mountpoints
  mount: name=/mnt/images src=<ip-nodo-control>:/images fstype=nfs
         opts=defaults dump=0 passno=2 state=mounted
```

A continuación se lista una guía rápida de implementación de las operaciones del módulo *iaas.py* que gestionan las imágenes.

`iaas.addImage(img)`
Se genera un id para la imagen con *uuid.uuid4()* o *time.time_ns()*.
Se obtiene el fichero que contiene la imagen y se copia en el export */images*
Se inserta la info de la nueva imagen en SQLite.

`iaas.listImages()`
Se listan las imágenes, recuperando la info de SQLite.

6. Gestión de máquinas virtuales

Los comandos disponibles para gestionar máquinas virtuales desde la perspectiva del usuario son los siguientes:

```
vm add <imgName> [-t <type>] [-m <mem>] [-n <node> {-p <nodePort>:<vmPort>}]
vm start <machId>
vm stop <machId>
vm rm <machId>
vm ls {field=value}
```

En este apartado se comentarán algunos aspectos importantes para la gestión de máquinas virtuales:

- Interconexión de máquinas virtuales.
- Planificación de máquinas virtuales.
- Monitorización de máquinas virtuales.

6.1 Interconexión de máquinas virtuales

Por defecto, las máquinas virtuales que se ejecutan en un host no son visibles desde fuera. Esta circunstancia impide que máquinas virtuales que se ejecutan en distintos hosts puedan comunicarse entre sí. Para posibilitar la comunicación entre máquinas virtuales, permitiremos que una máquina virtual pueda publicar un puerto en el host en el que se ejecuta. Entonces se habilitará una redirección del puerto del host al puerto de la máquina virtual. Para ello, al crear la máquina virtual es obligatorio especificar el host en el que se debe ejecutar, así como el puerto del host que se redirigirá y el puerto de la máquina virtual al que se redirigirá.

Por ejemplo, desde el punto de vista del usuario, para crear una máquina virtual de tipo *libvirt* o *docker*, en el nodo *node1* y que redirija el puerto 8080 del host al puerto 80 de la máquina virtual se usarían respectivamente los siguientes comandos:

```
> vm add debian -t libvirt -n node1 -p 8080:80
> vm add nginx -t docker -n node1 -p 8080:80
```

En Docker es sencillo implementar esta redirección de puertos. Basta con utilizar la opción -p, por ejemplo:

```
> docker run -p 8080:80 nginx
```

Para obtener los mismos resultados utilizando la librería Python de Docker usaríamos el siguiente código:

```
import docker
client = docker.from_env()
client.containers.run("nginx", ports={"80/tcp":8080})
```

Para implementar redirección de puertos en Libvirt podemos usar múltiples herramientas. Por ejemplo podemos usar *pwncat*:

https://pwncat.org/

Primero la instalaremos:

```
(venv) > pip install pwncat
```

Para crear una redirección del puerto 8080 local al puerto 80 de la máquina 172.17.0.2 podemos usar el siguiente comando:

```
(venv) > pwncat -L 0.0.0.0:8080 172.17.0.2 80
```

Para obtener los mismos resultados en Python, tendremos que ejecutar un proceso externo. Podemos hacerlo por ejemplo usando el módulo *subprocess*.

```
import subprocess
p = subprocess.Popen(['pwncat', '-L', '0.0.0.0:8080', '172.17.0.2', '80'])
```

Cuando deseemos finalizar la redirección bastaría con matar el proceso con:

```
p.kill()
```

Con esta estrategia, nuestro IaaS debe efectuar un seguimiento de qué puertos se publican en qué nodos. Si una máquina virtual desea publicar en un puerto ya ocupado se debe rechazar la operación.

6.2 Planificación de máquinas virtuales

Cuando el consumidor solicita la creación de una máquina virtual, puede opcionalmente especificar el nodo de proceso donde desea que se ejecute con la opción *-n*. Es obligatorio si además necesita publicar un puerto. Por ejemplo:

```
> vm add debian -t libvirt -n node1
> vm add nginx -t docker -n node2 -p 8080:80
```

Si no se especifica el nodo el IaaS debe decidir cuál es el mejor nodo de proceso donde podría arrancarla. A este proceso se le denomina **planificación**.

El IaaS puede implantar distintas estrategias de planificación. Por ejemplo, podría escoger el nodo de proceso que posea menos máquinas virtuales en ejecución. Otra opción sería adquirir medidas de la carga de cada nodo de proceso, y actuar en base a ello.

Se deja al alumno que escoja cuál es la mejor estrategia.

6.3 Monitorización de máquinas virtuales

Otra cuestión importante es que el IaaS conozca en todo momento el estado de los nodos de proceso y de las máquinas virtuales que corren en su interior. Para ello, el IaaS puede ejecutar de manera periódica procesos que comprueben si están vivos.

Si una máquina virtual falla, el IaaS podría marcarlo como fallido. Si el IaaS es “inteligente” podría volver a arrancar la máquina virtual de nuevo.

Si un nodo de proceso falla, entonces todas las máquinas virtuales ejecutándose en el nodo deben de ser marcadas como fallidas, y podría replanificarse su ejecución en nuevos nodos. El nodo fallido debería de ser marcado como “no disponible” y ninguna otra máquina virtual debería de ejecutarse en él.

Se deja al alumno escoger la mejor manera de implementar estas cuestiones.