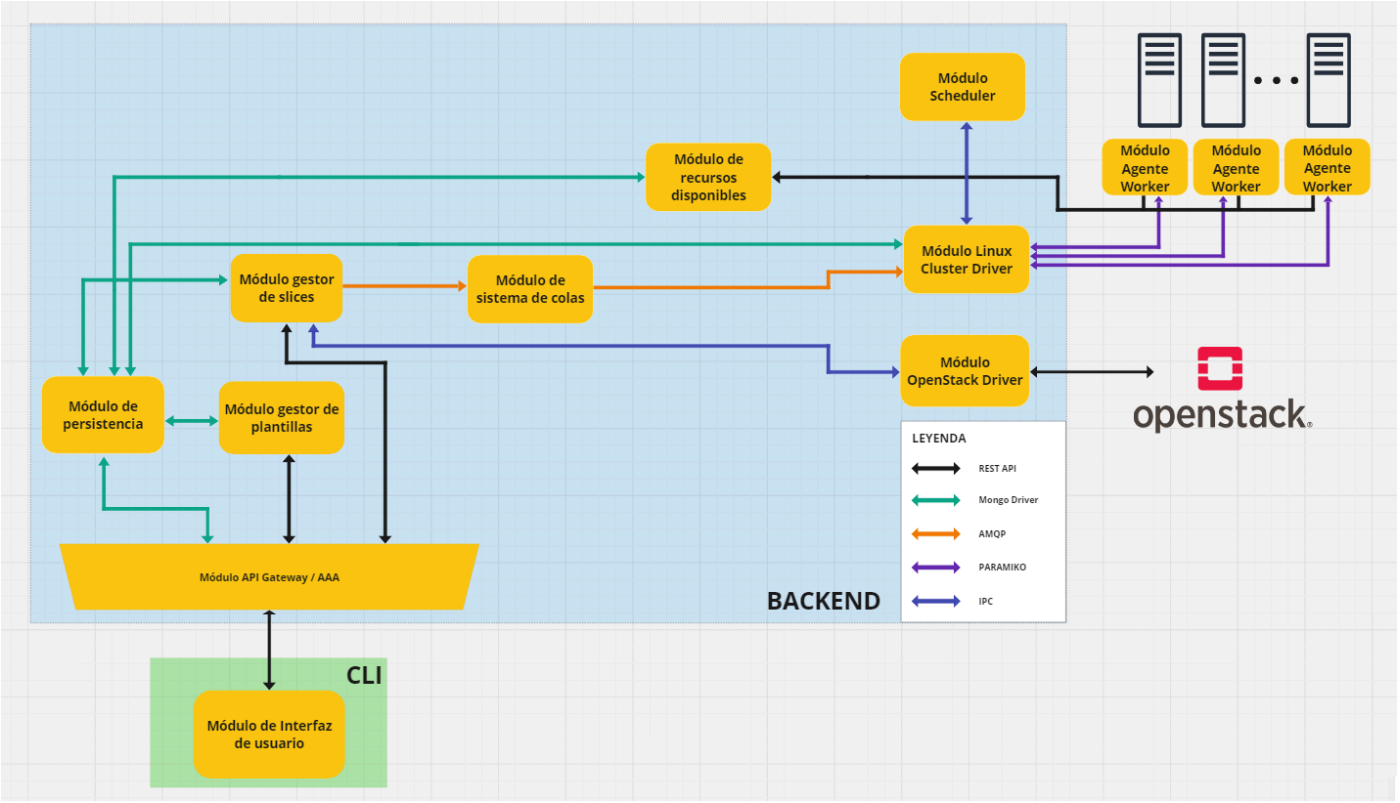
TEL141 - INGENIERÍA DE REDES CLOUD
PRESENTACIÓN FINAL - ARQUITECTURA Y HIGH LEVEL SOFTWARE DESIGN

Código - Nombres y Apellidos:
1 20200334 - Gil Zanabria 2 20195922 - Carlos Pisco 3 20202073 - Rodrigo Barrios

ÍNDICE

1. ARQUITECTURA



2. MÓDULOS DE ARQUITECTURA

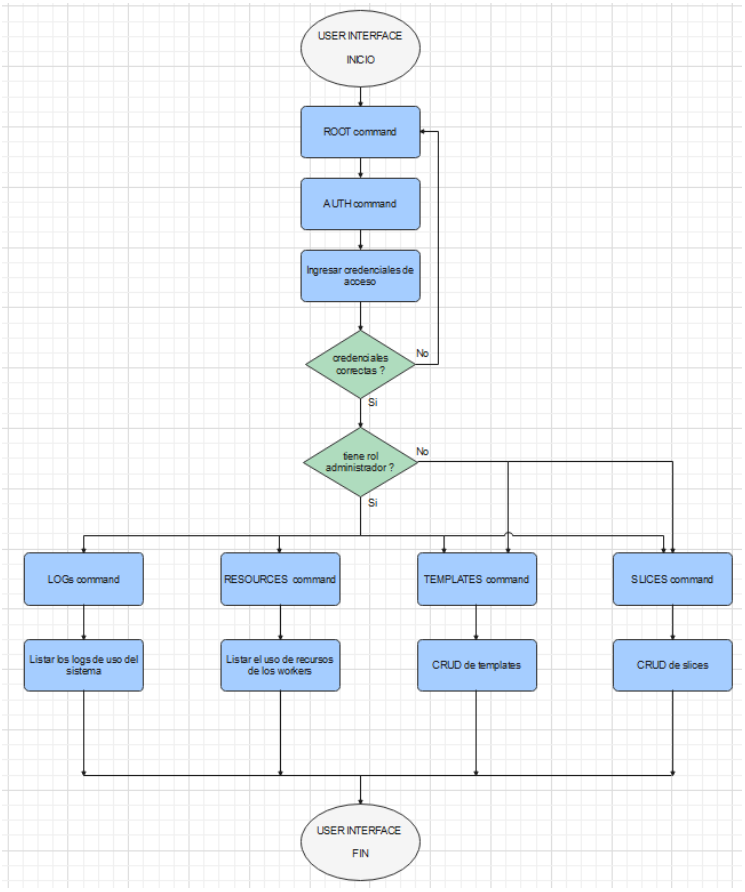
2.1 CLI APP

2.1.1 Módulo de interfaz de usuario

Este módulo proporciona una interfaz de usuario (CLI) intuitiva que los usuarios utilizan para interactuar con el sistema. Otorga al usuario la capacidad de autenticarse, visualizar y gestionar plantillas y slices de VMs, así como el acceso a funciones específicas de administrador en caso esté autorizado a realizarlas. Utiliza un esquema cliente-servidor para la comunicación con el HeadNode a través de REST APIs expuestas.

- Menú para usuarios:
    - Autenticación
    - CRUD Plantillas
    - CRUD Slices
- Opciones adicionales para administrador:
    - Visualización Recursos Disponibles
    - Visualización Logs

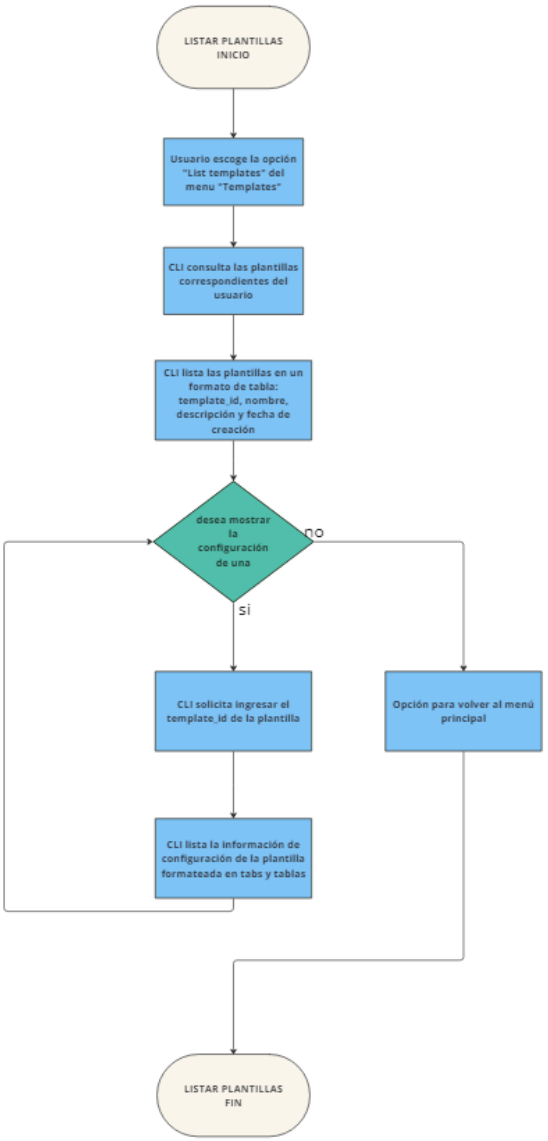
HIGH LEVEL SOFTWARE DESIGN USER INTERFACE



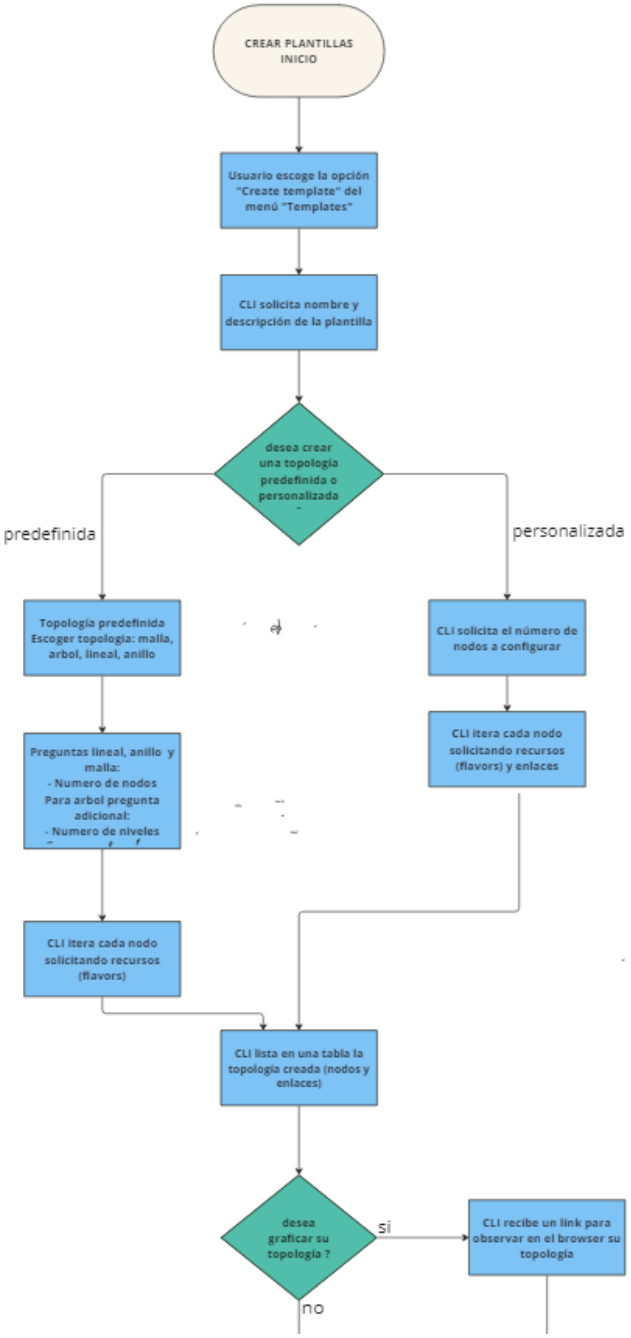
DIAGRAMAS DE FLUJO PARA TEMPLATES COMMAND EN CLI

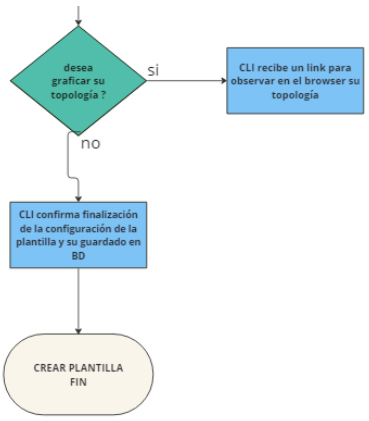
A continuación, se muestran los flujos que se siguen en la ejecución del CLI para las opciones del *Templates* command.

LISTAR PLANTILLAS

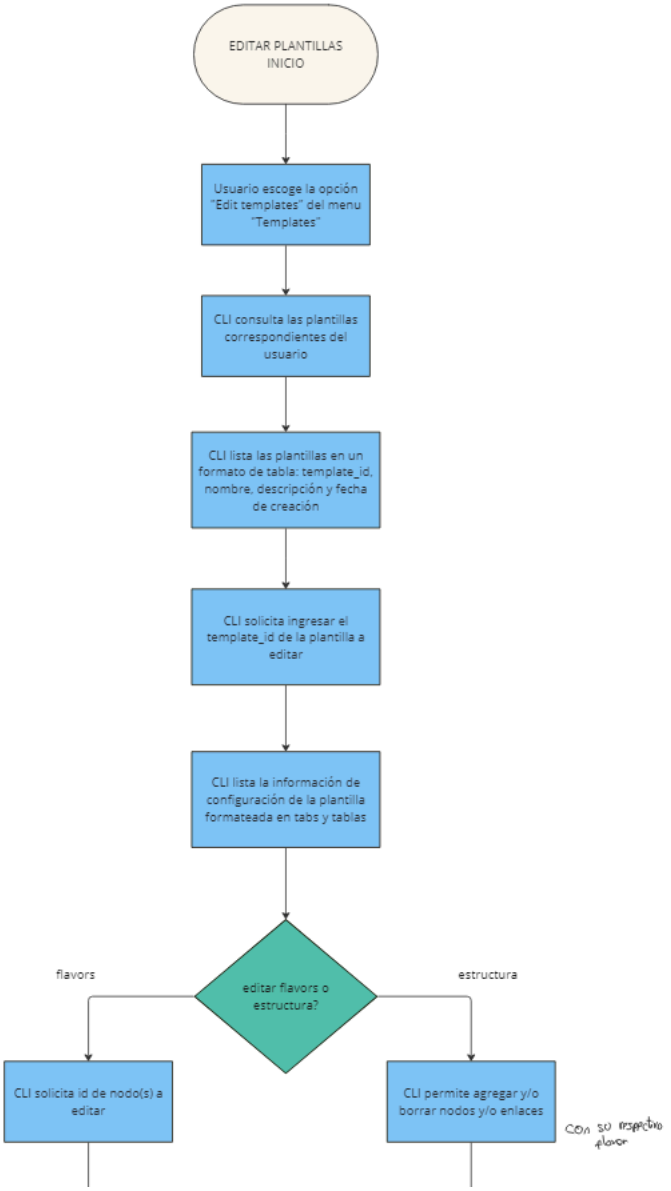


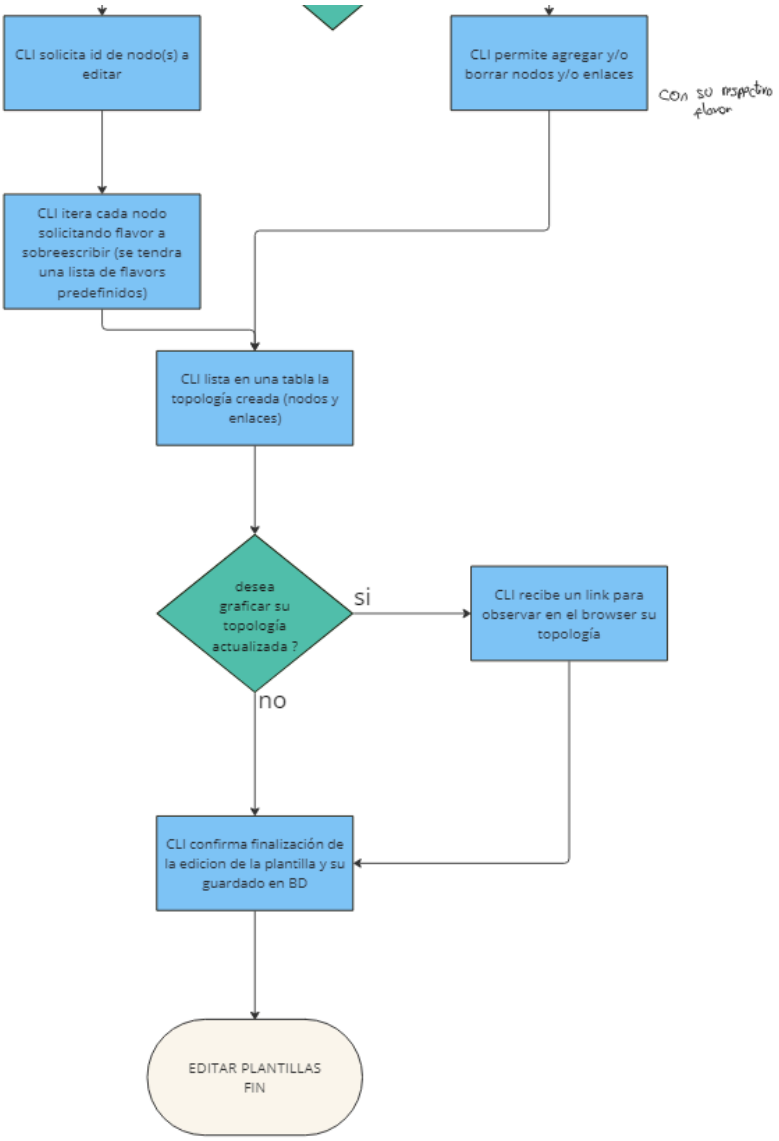
CREAR PLANTILLAS



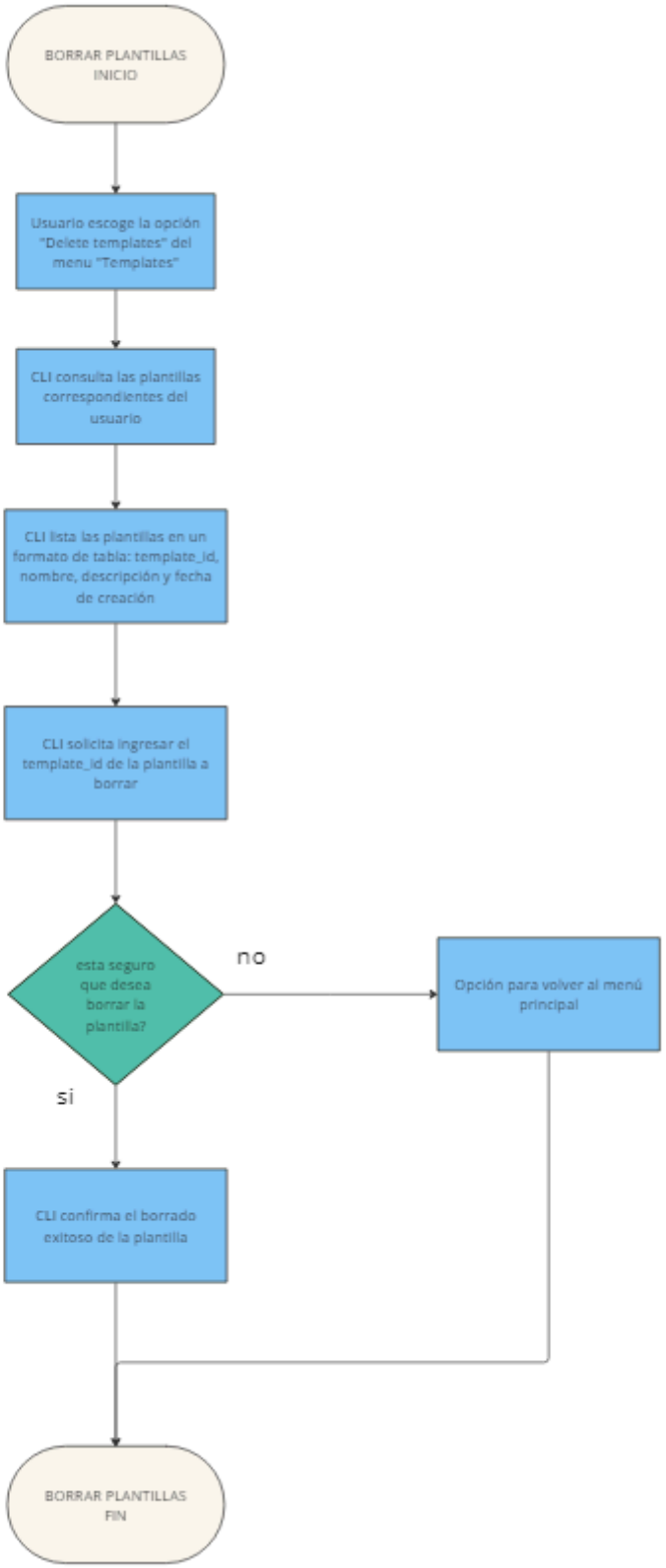


EDITAR PLANTILLAS

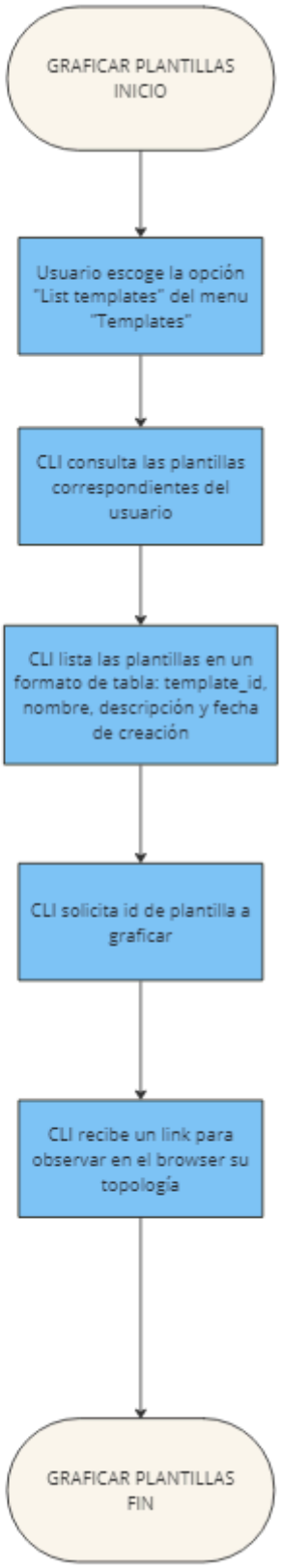



CLI solicita id de nodo(s) a editar
CLI permite agregar y/o borrar nodos y/o enlaces
con su respectivo flavor
CLI itera cada nodo solicitando flavor a sobreescribir (se tendra una lista de flavors predefinidos)
CLI lista en una tabla la topología creada (nodos y enlaces)
desea graficar su topología actualizada ?
si
CLI recibe un link para observar en el browser su topología
no
CLI confirma finalización de la edicion de la plantilla y su guardado en BD
EDITAR PLANTILLAS
FIN

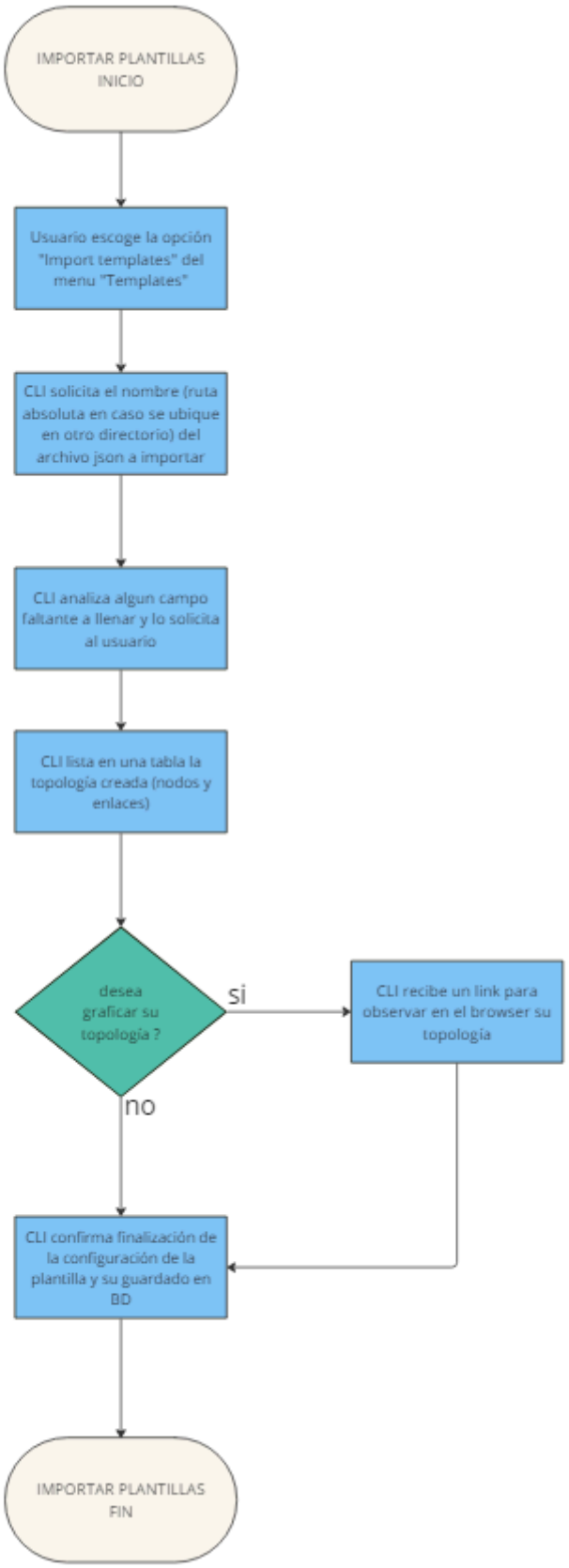
BORRAR PLANTILLAS



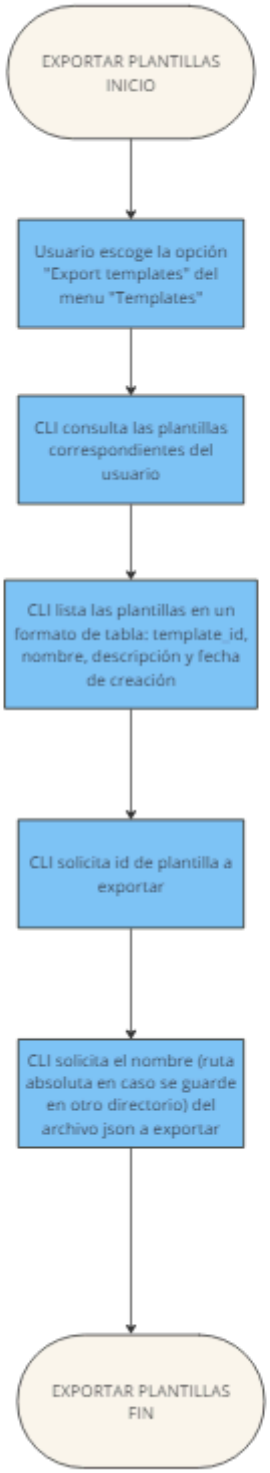
GRAFICAR PLANTILLAS



IMPORTAR PLANTILLAS



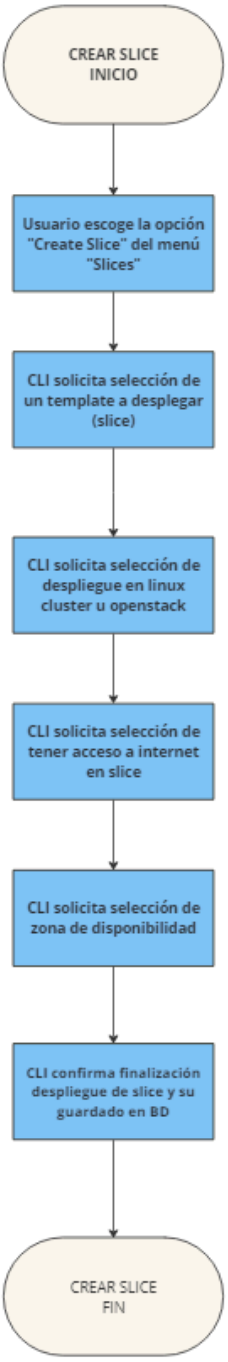
EXPORTAR PLANTILLAS



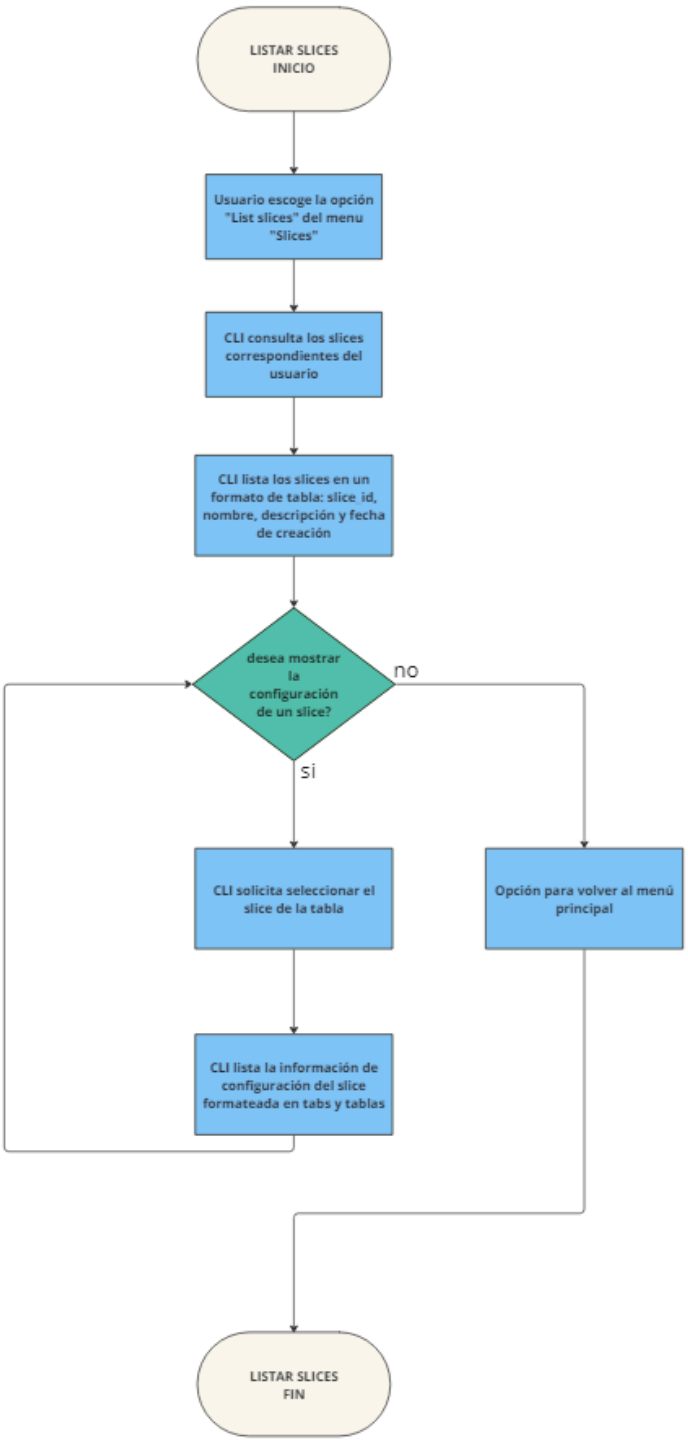
DIAGRAMAS DE FLUJO PARA SLICES COMMAND EN CLI

A continuación, se muestran los flujos que se siguen en la ejecución del CLI para las opciones del *Slices* command.

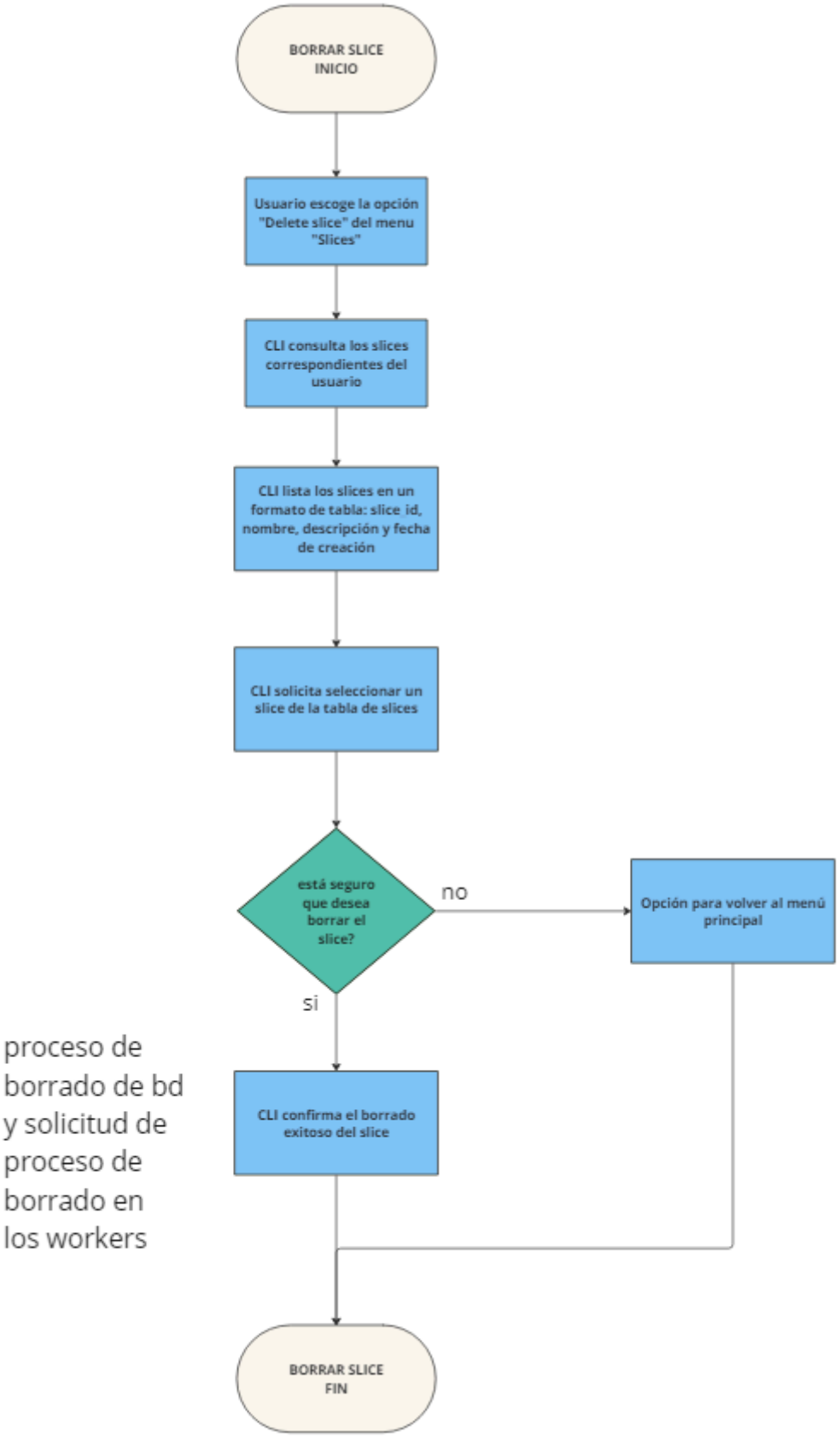
CREAR SLICES



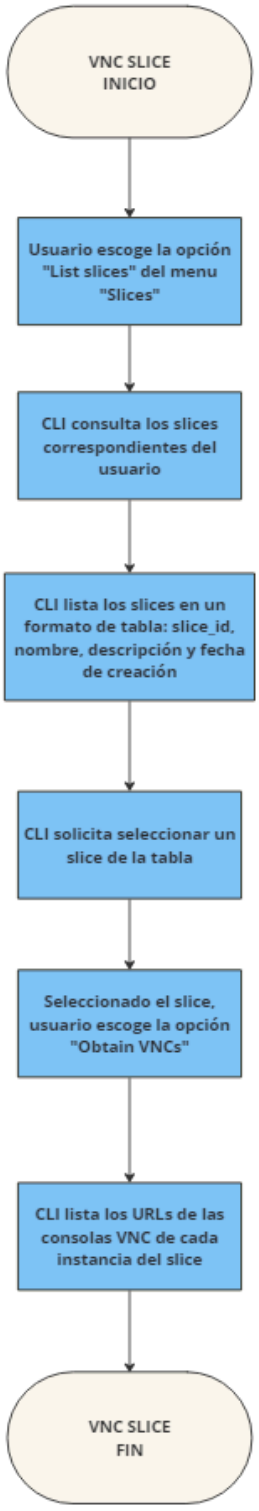
LISTAR SLICES



BORRAR SLICE



OBTENER VNCs URLs DE SLICE



2.2 BACK END MODULES

2.2.1 Módulo Api gateway / AAA

Este módulo actúa como punto de entrada para todas las solicitudes de API que provienen del CLI. Su principal función es recibir, dirigir, controlar y proteger el tráfico hacia y desde las API que se encuentran en el backend, ofreciendo una capa de seguridad. Además, se encarga de autenticar a los usuarios y verificar sus permisos para acceder a diferentes funciones y recursos dentro del sistema mediante el envío de una cabecera X-API-Key. Se hace uso de json web tokens para manejar la sesión de cada usuario. En la base de datos (acceso mediante módulo de persistencia), se cuenta con una colección de usuarios con roles predefinidos. Cada usuario cuenta con credenciales (username, password), un id y un rol.

APIs API GATEWAY / AAA

Método	Ruta	Acción
POST	/login	Iniciar sesión y autenticar al usuario con JWT
GET	/graph/{template_id}	Obtener un gráfico de topología específico por ID
ANY (GET,POST,PUT,DELETE)	/templateservice/*path	Redirigir todas las solicitudes al servicio templateservice
ANY (GET,POST,PUT,DELETE)	/sliceservice/*path	Redirigir todas las solicitudes al servicio sliceservice

2.2.2 Módulo gestor de plantillas

Este módulo gestiona las plantillas de slices de VMs, que sirven como modelos predefinidos para la creación de nuevos slices. Almacena información sobre las configuraciones de las plantillas y facilita su uso en la creación de nuevos slices. Para este caso de creación de plantillas, se toma en consideración el id del usuario, nombre de la plantilla, descripción de la plantilla, tipo de topología (lineal, malla, árbol general, árbol binario, anillo, estrella y personalizado) y timestamp de creación. Adicionalmente, se guardan los enlaces y nodos. Cada nodo se especifica su flavor, imagen y reglas de seguridad. A continuación, se muestran las distintas funciones generales y endpoints que ofrece el módulo gestor de plantillas.

Configuración y Conexión a MongoDB

- La aplicación comienza importando las bibliotecas necesarias, Flask para crear la API y pymongo para interactuar con MongoDB. Luego, se configura la conexión a MongoDB utilizando MongoClient de pymongo, conectándose a un servidor local en el puerto 27017.

Endpoint para Listar Plantillas

- El endpoint /templates permite listar todas las plantillas. Según el rol del usuario (user o administrator), se filtran las plantillas por user_id o se devuelven todas. Los campos seleccionados incluyen name, description, created_at, topology_type y _id.

Endpoint para Buscar una Plantilla por ID

- El endpoint /templates/<string:template_id> permite buscar una plantilla específica por su ID. Si se encuentra la plantilla, se devuelve como una respuesta JSON.

Endpoint para Crear una Nueva Plantilla

- El endpoint /templates con el método POST permite crear una nueva plantilla. Los datos de la nueva plantilla se reciben en el cuerpo de la solicitud en formato JSON y se insertan en la colección de plantillas. Si la operación es exitosa, se devuelve el ID de la nueva plantilla creada.

Endpoint para Editar una Plantilla por ID

- El endpoint /templates/<template_id> con el método PUT permite editar una plantilla específica por su ID. Los datos actualizados de la plantilla se reciben en el cuerpo de la solicitud en formato JSON y se actualizan en la base de datos.

Endpoint para Eliminar una Plantilla por ID

- El endpoint /templates/<template_id> con el método DELETE permite eliminar una plantilla específica por su ID. Si la plantilla se elimina correctamente, se devuelve un mensaje de éxito.

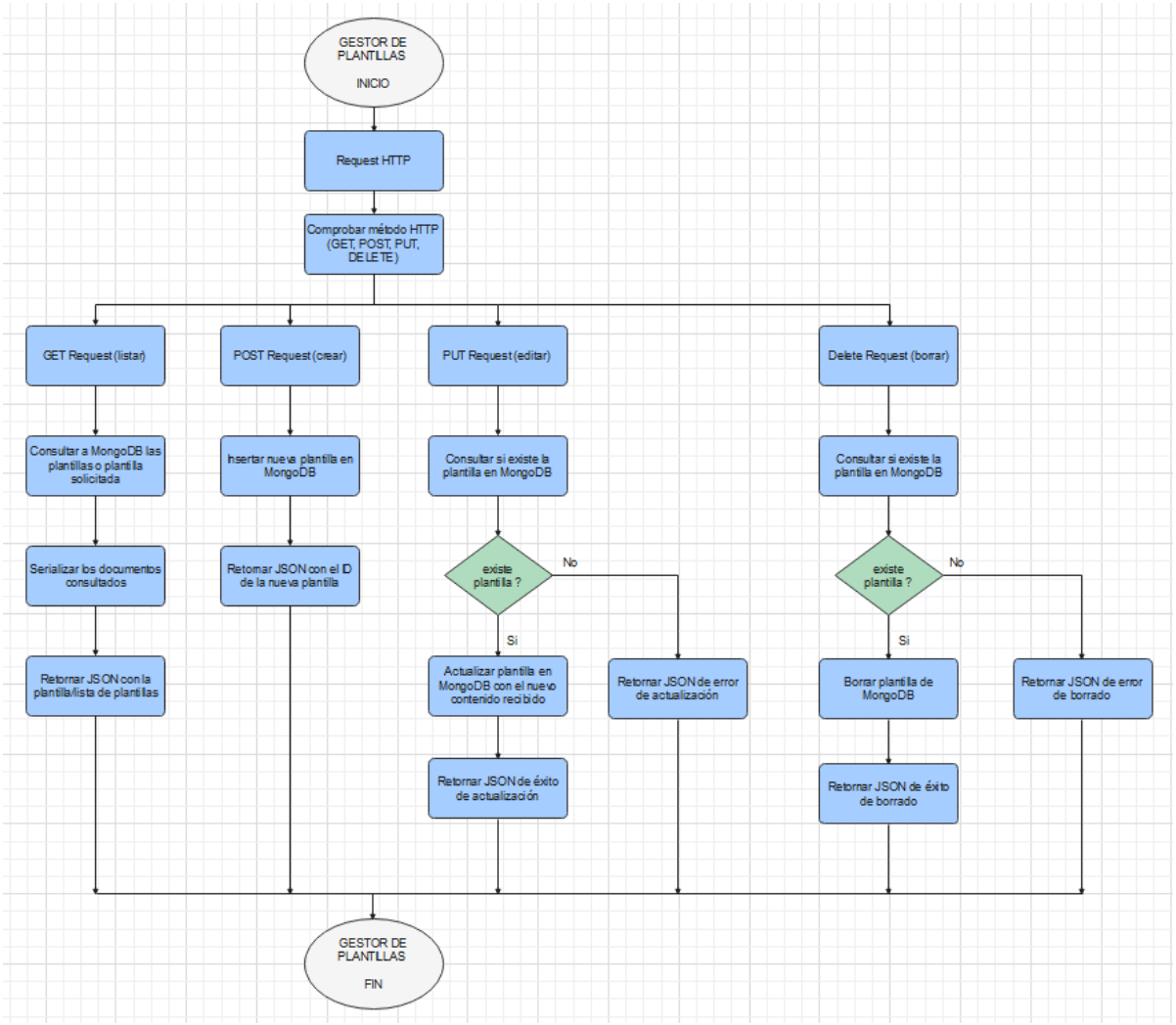
Endpoints para Listar Sabores, Imágenes y Zonas de Disponibilidad

- Se definen endpoints para listar sabores (/templates/flavors), imágenes (/templates/images) y zonas de disponibilidad (/templates/avz). Estos endpoints devuelven los documentos de las respectivas colecciones en formato JSON.

Ejecución

El módulo se ejecuta en el puerto 6666 en modo de depuración, lo que permite el desarrollo y la prueba interactiva del sistema.

HIGH LEVEL SOFTWARE DESIGN GESTOR DE PLANTILLAS



APIs GESTOR DE PLANTILLAS

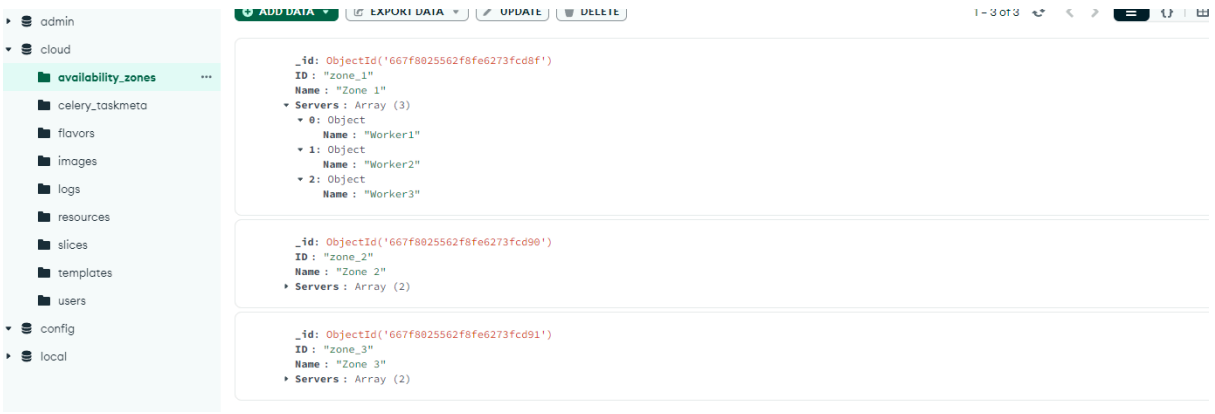
Método	Ruta	Acción
GET	/templates	Listar todas las plantillas pertenecientes a un usuario
GET	/templates/{template_id}	Listar una plantilla por ID
POST	/templates	Crear una nueva plantilla
PUT	/templates/{template_id}	Editar una plantilla por ID

DELETE	/templates/{template_id}	Eliminar una plantilla por ID
GET	/templates/flavors	Listar todos los flavors
GET	/templates/images	Listar todas las imágenes
GET	/templates/avz	Listar todas las zonas de disponibilidad

2.2.3 Módulo de persistencia

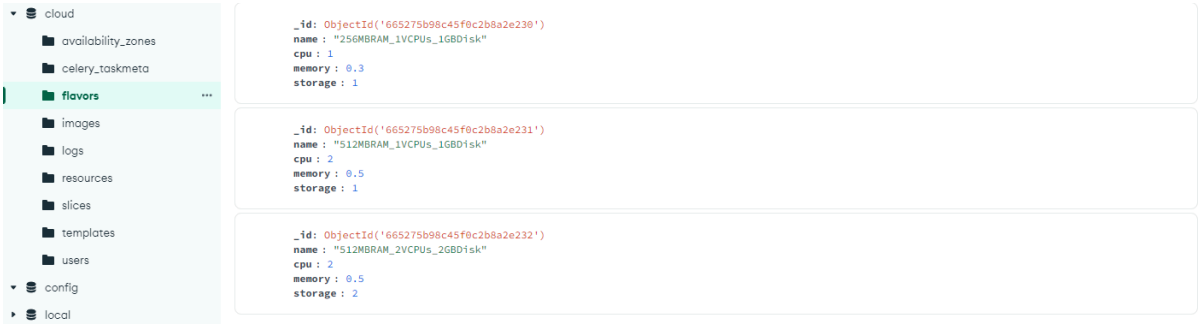
Este módulo proporciona funciones para el almacenamiento persistente de datos importantes del sistema: usuarios, configuración de plantillas y slices, registros de eventos, etc. En particular, para el proyecto se utiliza MongoDB, una base de datos no relacional. Esto se realiza con el objetivo de evitar conflictos al realizar cambios a los atributos de objetos. Se utilizará el driver de MongoDB en los módulos que necesiten conexión con la base de datos. Las colecciones de datos que se tienen son las siguientes:

- availability_zones: contiene las zonas de disponibilidad preestablecidas



- celery_taskmeta: guarda información los tasks generados para la gestión asíncrona.

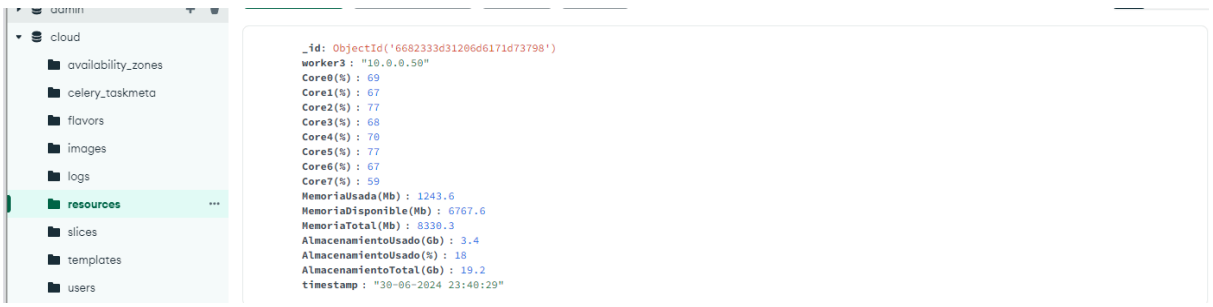
- flavors: contiene los flavors preestablecidos



- images: contiene información de las imágenes preestablecidas

- logs: guarda logs del despliegue y borrado en Linux Cluster

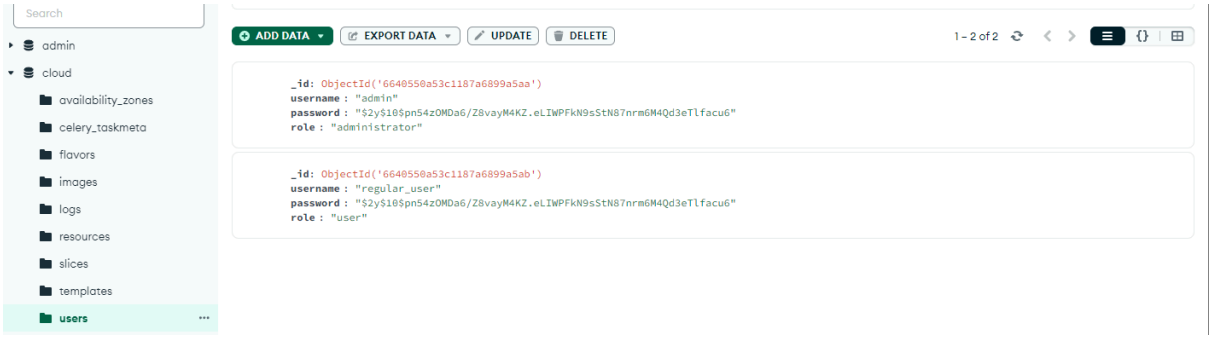
- resources: guarda las información recolectada de los recursos (CPU, RAM, DISK) de los Workers



- slices: guarda los slices creados por cada usuario

- templates: guarda las plantillas creadas por cada usuario

- users: contiene los usuarios existentes en nuestro sistema



2.2.4 Módulo Gestor de Slices

El gestor de slices es una herramienta para la gestión de slices en entornos de cloud computing, proporcionando funcionalidades completas para la creación, listado, búsqueda y eliminación de slices, así como la capacidad de gestionar tareas asíncronas a través de Celery. Este módulo administra la asignación de recursos, la gestión de topologías de red y cualquier otra configuración relacionada con las slices. Para el caso de creación de slice se toma en consideración el id del usuario, el id de la plantilla creada anteriormente, así como su nombre y descripción. Adicionalmente, se escoge el ambiente con el cual desplegarlo (linux u openstack). A continuación, una descripción de las funcionalidades y componentes:

Componentes del Módulo

- Flask: Es el framework web utilizado para crear la API RESTful que permite interactuar con el sistema.
- MongoDB: Base de datos NoSQL utilizada para almacenar información relacionada con las plantillas y los slices.
- Celery: Sistema de gestión de tareas distribuidas que se utiliza para ejecutar tareas asíncronas, como la implementación y eliminación de slices en diferentes entornos.
- Pymongo: Biblioteca de Python utilizada para interactuar con MongoDB.
- BSON ObjectId: Identificadores únicos utilizados por MongoDB.
- linux_cluster y openstack_driver: Módulos importados que proporcionan funcionalidades específicas para la implementación en clústeres Linux y entornos OpenStack.

Funcionalidades del Módulo

Serialización de Documentos

Las funciones serialize_document y serialize_template se utilizan para convertir los documentos de MongoDB en un formato que pueda ser fácilmente enviado como respuesta JSON. Estas funciones convierten los ObjectId en cadenas de texto y ajustan los nombres de los campos según sea necesario.

Gestión de Plantillas para despliegue

Obtener Plantilla por ID: La función obtenerTemplateById recupera una plantilla específica de la base de datos utilizando su ObjectId. Si la plantilla tiene un campo _id, este se elimina antes de devolver la plantilla.

Endpoints de la API

Crear Slice (/slices, método POST)

- Recibe información de un nuevo slice a través de una solicitud POST.
- Recupera la plantilla correspondiente desde la base de datos.
- Inserta el nuevo slice en la colección slices.
- Dependiendo del tipo de implementación (openstack o linux), llama al módulo correspondiente (openstack_driver o linux_cluster) para iniciar el proceso de despliegue.

Listar Slices (/slices, método GET):

- Recupera y lista todos los slices disponibles en la base de datos.
- Filtra los slices según el rol del usuario (user o administrador), verificando los encabezados de la solicitud.

Buscar Slice por ID (/slices/<string:slice_id>, método GET):

- Recupera un slice específico utilizando su ObjectId.
- Serializa el slice y lo devuelve en la respuesta JSON.

Buscar VNC de Slice (/slices/vnc/<string:slice_id>, método GET):

- Recupera la información de un slice específico.

- Obtiene los enlaces VNC asociados al proyecto del slice desde OpenStack y los devuelve en la respuesta.

Eliminar Slice por ID (/slices/<string:slice_id>, método DELETE):

- Elimina un slice específico utilizando su ObjectId.
- Dependiendo del tipo de implementación (openstack o linux), llama al módulo correspondiente para eliminar el slice.

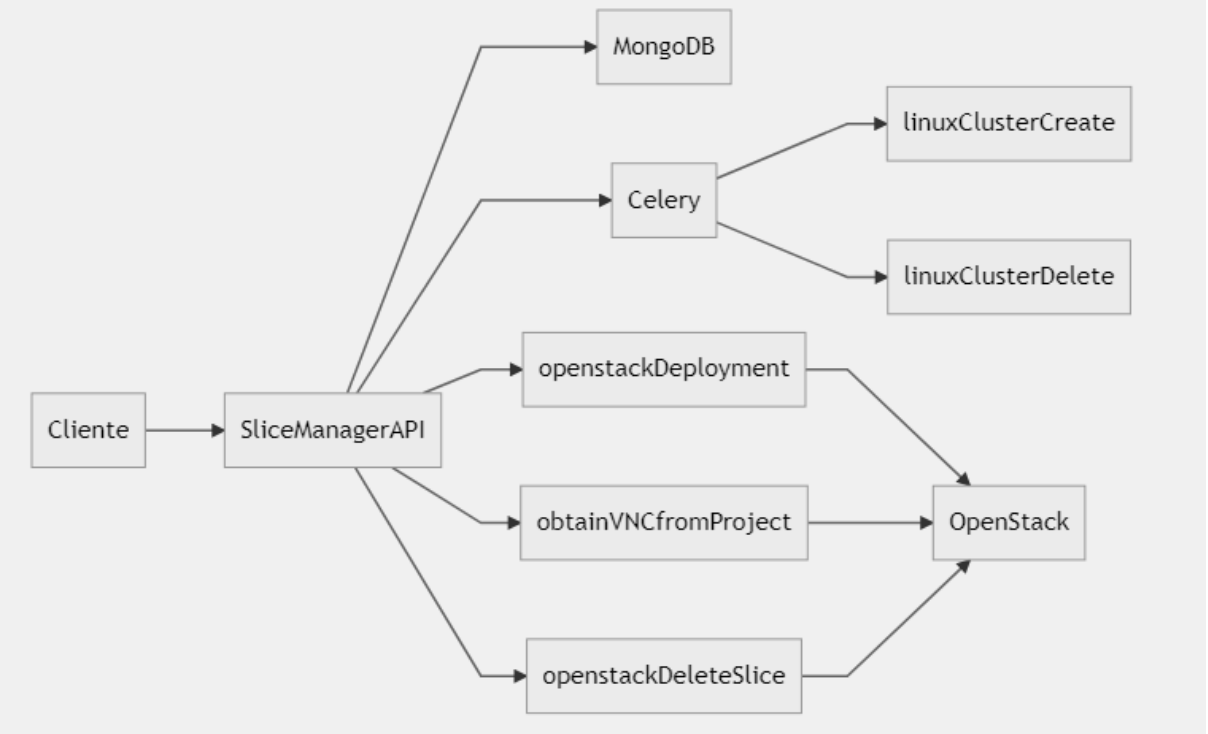
Obtener Resultado de Tarea (/tasks/<task_id>, método GET):

- Recupera el estado de una tarea asíncrona utilizando su task_id.
- Devuelve el estado actual de la tarea y, si está disponible, el resultado exitoso o el mensaje de error correspondiente.

Ejecución

El módulo se ejecuta en el puerto 9999 en modo de depuración, lo que permite el desarrollo y la prueba interactiva del sistema.

HIGH LEVEL SOFTWARE DESIGN GESTOR DE SLICES



APIs GESTOR DE SLICES

Método	**Ruta**	**Acción**
POST	/slices	Crear un nuevo slice (despliegue)
GET	/slices	Listar todos los slices
GET	/slices/{slice_id}	Buscar un slice por ID
GET	/slices/vnc/{slice_id}	Obtener enlaces VNC de un slice
DELETE	/slices/{slice_id}	Eliminar un slice por ID
GET	/tasks/{task_id}	Obtener el resultado de una tarea por ID (uso de celery)

2.2.5 Módulo de sistema de colas

Este módulo asigna una posición a los planes de despliegue o borrado utilizando Linux Cluster. Al tener múltiples usuarios usando de manera concurrente el sistema, se ha identificado la posibilidad de ´race conditions´ al intentar crear diferentes slices utilizando los mismos recursos en la misma zona de disponibilidad. Para prevenir esto, el módulo de sistema de colas utiliza un sistema de colas que proporciona un orden secuencial a las solicitudes recibidas de despliegue. Esto se logra utilizando el broker de mensajería de código abierto, RabbitMQ, así como la librería de python celery.
Celery es una biblioteca para Python que permite la ejecución de tareas en segundo plano, permitiendo el manejo de trabajos asincrónicos y distribuidos. En este caso, se utiliza para gestionar las tareas de despliegue y eliminación de slices de máquinas virtuales (VMs) en un entorno de Linux Cluster. A continuación, se describe cómo funciona Celery en la configuración, utilizando MongoDB como backend y RabbitMQ como broker.

Componentes Principales:

1. **Broker (RabbitMQ)**:
    - El broker es responsable de recibir, encolar y distribuir las tareas a los workers. En este caso, RabbitMQ actúa como intermediario, gestionando la cola de mensajes donde las tareas son enviadas.
    - RabbitMQ garantiza que las tareas sean recibidas por los workers de Celery y permite manejar múltiples tareas concurrentemente.
2. **Backend (MongoDB)**:
    - El backend almacena los resultados de las tareas y su estado. En este configuración, MongoDB se utiliza para almacenar el estado de cada tarea, sus resultados y los logs generados durante su ejecución.
    - Esto permite consultar el estado y el resultado de cualquier tarea en cualquier momento.
3. **Workers**:
    - Los workers son los encargados de ejecutar las tareas encoladas por RabbitMQ. En este caso, hay un worker que procesa las tareas de despliegue y eliminación de VM slices.
    - Cada worker se suscribe a la cola de RabbitMQ, toma las tareas y las ejecuta de manera asincrónica.
4. **Aplicación Flask**:
    - La aplicación Flask (slice manager) expone los endpoints, permitiendo recibir solicitudes para desplegar y eliminar slices.
    - Flask también se encarga de iniciar las tareas y devolver el estado de las mismas a los usuarios.

Escalabilidad y Concurrencia de solicitudes:

- **Múltiples Workers**: Aunque actualmente se tiene un solo worker, se puede escalar horizontalmente añadiendo más workers. Cada nuevo worker se suscribe a la cola de RabbitMQ y puede procesar tareas concurrentemente, mejorando el rendimiento y la capacidad de manejar múltiples solicitudes simultáneas.

- **Concurrencia**: RabbitMQ puede manejar múltiples tareas en cola simultáneamente, y Celery distribuirá estas tareas entre los workers disponibles. Esto permite que múltiples despliegues de VM slices se manejen de manera concurrente sin bloquear la aplicación Flask o causar tiempos de espera largos para los usuarios.

2.2.6 Módulo de scheduler (VM placement): diseño teórico

En el contexto de la gestión de recursos en una infraestructura de virtualización, el principal requerimiento para el Scheduler es el ahorro de energía. Además, se prioriza la concentración de la carga en un servidor para optimizar el consumo de energía y minimizar el número de servidores activos.

Diferentes empresas implementan estrategias de gestión de energía para optimizar el uso de sus servidores que alojan diversas aplicaciones. Una tecnología clave en este contexto es el **Intelligent Platform Management Interface (IPMI)**, un conjunto de especificaciones estandarizadas que facilitan la gestión fuera de banda (Out of Band Management). El **Baseboard Management Controller (BMC)**, un chip ubicado en la placa base del servidor, utiliza IPMI para operar independientemente del sistema operativo. Este controlador recopila información de sensores como temperatura, voltaje y velocidad de los ventiladores, y proporciona control de energía, permitiendo encender, apagar y reiniciar el dispositivo, además de ofrecer acceso a otros recursos del equipo.

Vendor	BMC's Name	BMC's Siglas
HP	Integrated Lights Out	iLO
Dell	Dell Remote Access Control	DRAC
IBM	Remote Supervisor Adapter	RSA
Cisco	Integrated Management Controller	IMC

Además, se utiliza **IPMITool**, una herramienta de código abierto que permite interactuar con dispositivos compatibles con IPMI. Esta herramienta permite supervisar y configurar estos dispositivos mediante comandos específicos. Junto con **IPMILan**, una interfaz lógica ubicada en el BMC del servidor, es posible gestionar estos dispositivos de manera remota a través de una red Ethernet.

Para garantizar la seguridad de estos dispositivos, se recomienda utilizar IPMITool y otras herramientas relacionadas en redes privadas, minimizando así los riesgos asociados con el acceso no autorizado y protegiendo la infraestructura crítica de la empresa.

Otra consideración tomada por el servidor es que no puede estar con cargas de CPU altas durante un largo periodo de tiempo, ya que esto afecta su rendimiento y tiempo de respuesta a las solicitudes. Por ello, se define un umbral de consumo de recursos para delimitar el uso máximo en cada servidor. En este proyecto, los flavors correspondientes a las VMs a desplegar presentan tres recursos a considerar (CPU, RAM, Disco), donde el recurso decisivo para la toma de decisiones en el despliegue fue la CPU.

Parámetros para la Asignación de VMs

CPU (Unidad Central de Procesamiento):

- **Dificultad de Actualización: Alta**. La actualización de la CPU es compleja, implicando la verificación de compatibilidad con la placa base y posibles actualizaciones de BIOS y sistemas de enfriamiento.
- **Razón**: Las CPUs tienen limitaciones en cuanto a sockets y chipsets, y su instalación requiere una transferencia de calor adecuada y una conexión segura.

RAM (Memoria de Acceso Aleatorio):

- **Dificultad de Actualización: Moderada**. La RAM es relativamente fácil de actualizar, requiriendo solo la instalación de módulos adicionales en las ranuras disponibles.
- **Razón**: La compatibilidad de la RAM se limita a tipo (DDR3, DDR4, etc.) y velocidad, y muchas placas base tienen ranuras adicionales que facilitan la expansión.

Disco (Almacenamiento, HDD o SSD):

- **Dificultad de Actualización: Baja**. La adición o reemplazo de discos es sencillo y común en la mayoría de los sistemas.
- **Razón**: Los discos utilizan interfaces estándar y las cajas de computadoras suelen ofrecer espacio adicional para nuevas unidades.
    -

Por Qué la CPU es el Parámetro Principal en la Toma de Decisiones del Scheduler

En la toma de decisiones del scheduler para la asignación de VMs a zonas de disponibilidad, la **CPU** se convierte en el criterio más crítico por varias razones:

1. **Restricciones de Compatibilidad y Escalabilidad:** La CPU tiene restricciones más estrictas en términos de compatibilidad con la placa base y otros componentes del sistema. La actualización o modificación de la CPU puede requerir cambios significativos en la infraestructura, lo cual implica un costo y una complejidad mayor en comparación con la RAM o el disco.
2. **Impacto en el Rendimiento:** La CPU es el componente central que maneja las tareas de procesamiento. Las decisiones sobre la asignación de VMs basadas en la CPU aseguran que se cumplan las demandas de procesamiento, que es esencial

para el rendimiento general de las aplicaciones. El rendimiento de una VM está directamente relacionado con la capacidad de la CPU disponible en el worker.

3. **Capacidad de Recursos y Balanceo de Carga:** La capacidad de CPU limita el número de VMs que se pueden ejecutar simultáneamente en un worker. Asegurar que una VM tenga acceso a suficientes recursos de CPU evita la sobrecarga de un worker y garantiza que el balanceo de carga sea eficiente. La gestión de la CPU también permite la implementación de estrategias de escalabilidad vertical, donde la capacidad del worker puede ser ampliada o reducida con base en la carga de trabajo.
4. **Planificación a Largo Plazo:** Dado que la actualización de la CPU es más complicada y costosa, los schedulers priorizan la evaluación de recursos de CPU para garantizar una asignación eficiente y sostenible de las VMs a lo largo del tiempo. Esto ayuda a planificar mejor la capacidad a largo plazo y a minimizar el riesgo de cuellos de botella.

Para evaluar la asignación de una VM en un determinado worker se evaluará la carga presente en todos los cores presentes en los tres workers.
Ejemplo de los recursos obtenidos de cada worker:


```
_id: ObjectId('667f86915ac30dcfcf0305d7')
worker1 : "10.0.0.30"
Core0(%) : 92
Core1(%) : 92
Core2(%) : 81
Core3(%) : 94
Core4(%) : 84
Core5(%) : 74
Core6(%) : 72
Core7(%) : 93
MemoriaUsada(Mb) : 1080.8
MemoriaDisponible(Mb) : 6932
MemoriaTotal(Mb) : 8330.4
AlmacenamientoUsado(Gb) : 3.2
AlmacenamientoUsado(%) : 17
AlmacenamientoTotal(Gb) : 19.2
timestamp : "28-06-2024 22:59:13"
```


Se suman los porcentajes de uso obtenidos en cada core, posteriormente al haber realizado lo mismo en las 1000 mediciones se le suma la carga demandada por la VM a desplegar, ese porcentaje total obtenido se evaluará con el umbral definido del total de uso de CPU, para poder obtener el número de veces que dicha carga total supera el umbral, la que finalmente delimitara si dicha VM con dicha carga se desplegará en dicho worker o pasará a evaluar con las mismas condiciones en el siguiente worker.

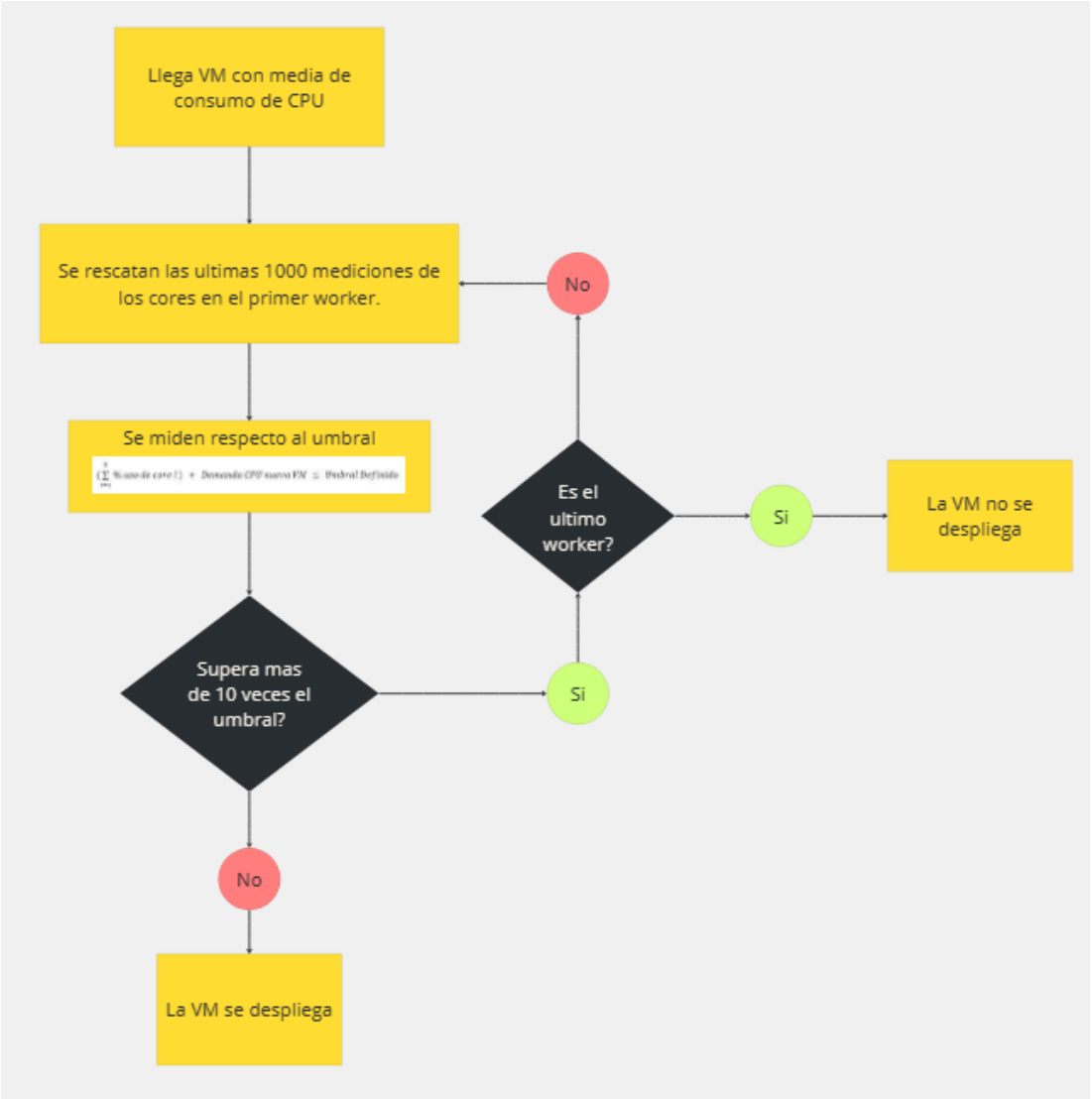
$$\left(\sum_{i=1}^{8} \%\ uso\ de\ core\ i\right) + Demanda\ CPU\ nueva\ VM \leq Umbral\ Definido$$

Consideraciones:

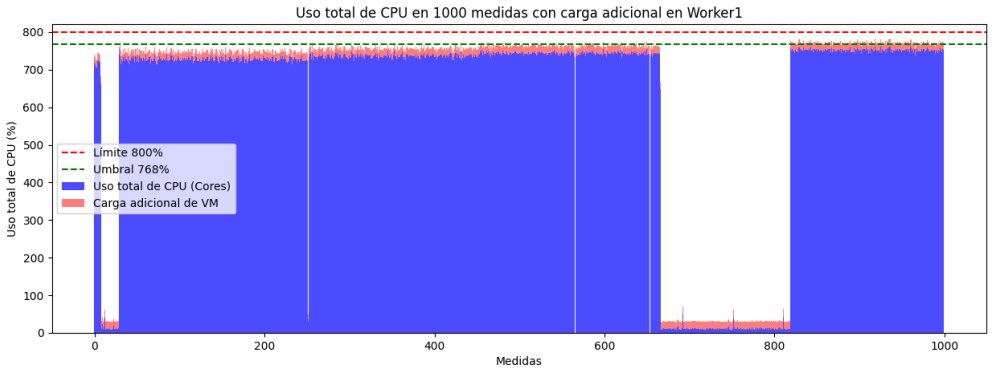
- El umbral definido es del 96% para cada core, y considerando los 8 cores presentes en cada worker el umbral sería de 768%.
- Las demandas de CPU consideradas para fines de simplicidad en este proyecto presentan el mismo valor.
- El número de veces aceptado para la carga de sobrepasar el umbral es de 1/100 dándonos un total de 1000 para lo considerado en este modelamiento.

Cabe resaltar que para escalar lo modelado en este proyecto se deberían considerar cargas variables para cada VM, esto implicaría considerar diferentes umbrales dependiendo de las aplicaciones a usarse en cada VM y la disponibilidad requerida.

El flujo a seguir es de la siguiente forma:

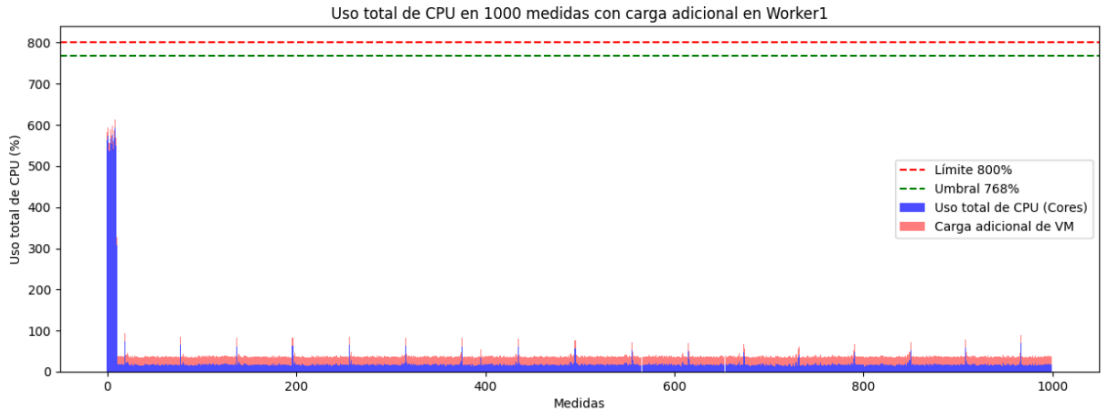


Los resultados obtenidos al sumar todos los porcentajes de los cores en cada una de las mediciones añadiendo la media de la demanda requerida por la VM a intentar desplegar (carga 20%) se vería de la siguiente forma:




```
(myvenv) ubuntu@HeadNode:~/piton$ /home/ubuntu/piton/myvenv/bin/python /home/ubuntu/piton/promedios.py
Gráfico guardado como 'total_usage_with_vm.png'
Ratio de valores que pasan el umbral de 768%: 0.16
```

En este caso se tiene un total de 160 valores que superaron el umbral definido, lo cual es superior al 0.01 definido inicialmente, como resultado la VM no se desplegará en dicho worker y pasaría a evaluarse el siguiente worker de la misma forma.
Caso contrario donde los cores no presentan cargas elevadas es el siguiente:



En este caso el número de veces superado el umbral fue 0 por lo que la VM puede ser desplegada correctamente.

```
ubuntu@HeadNode:~/owo/piton$ /home/ubuntu/owo/piton/myvenv/bin/python /home/ubuntu/owo/piton/promedios.py
Gráfico guardado como 'total_usage_with_vm.png'
Ratio de valores que pasan el umbral de 768%: 0.00
```

Cabe resaltar que para este caso se están tomando todos los cores como uno solo, es por eso que se presentan dos escenarios que pueden resultar confusos al examinar este modelamiento:

- Si todos los cores presentan una carga elevada y solo uno de ellos no está

siendo usado, el umbral podría verse superado, entonces se estaría desperdiciando un core de dicha máquina.
- Si todos los cores presentan una carga elevada, pero al sumarse presentan una carga menor al umbral la demanda de la VM en cuestión sería aceptada pero podría no 'entrar' en ningún core realizando la suma.

En este caso entra en juego el kernel scheduler de Linux el cual puede administrar las cargas de los procesos para poder administrar de mejor manera el uso de cada core, y así esto se solucionaría:

- El linux scheduler podría mostrar por un instante 7 cores saturados y uno 'vacío', pero su misión es poder equilibrar dichas cargas con los recursos que posee y de esa forma se podría equilibrar la carga en los demás cores en las posteriores mediciones.
- El linux scheduler se encargaría de recibir la carga dicha VM y para poder asignarla a un core movería la carga de otros cores con fin de hacer espacio para esta nueva VM con su demanda requerida.

Finalmente recalcar que para el requerimiento establecido (Ahorro de Energía) se dejan de lado otros requerimientos como resiliencia, en caso el server donde estamos concentrando las cargas pierda conexión a su fuente de alimentación todos las VMs/Slices desplegados se verían afectados, y los usuarios no tendrían conexión hasta volver a levantar el servidor. Por esa razón sería una mejor idea iniciar con dos servers prendidos y emplear balanceo de carga a manera de no arriesgar todo a un server que puede experimentar fallos en el futuro.

2.2.7 Módulo de recursos disponibles

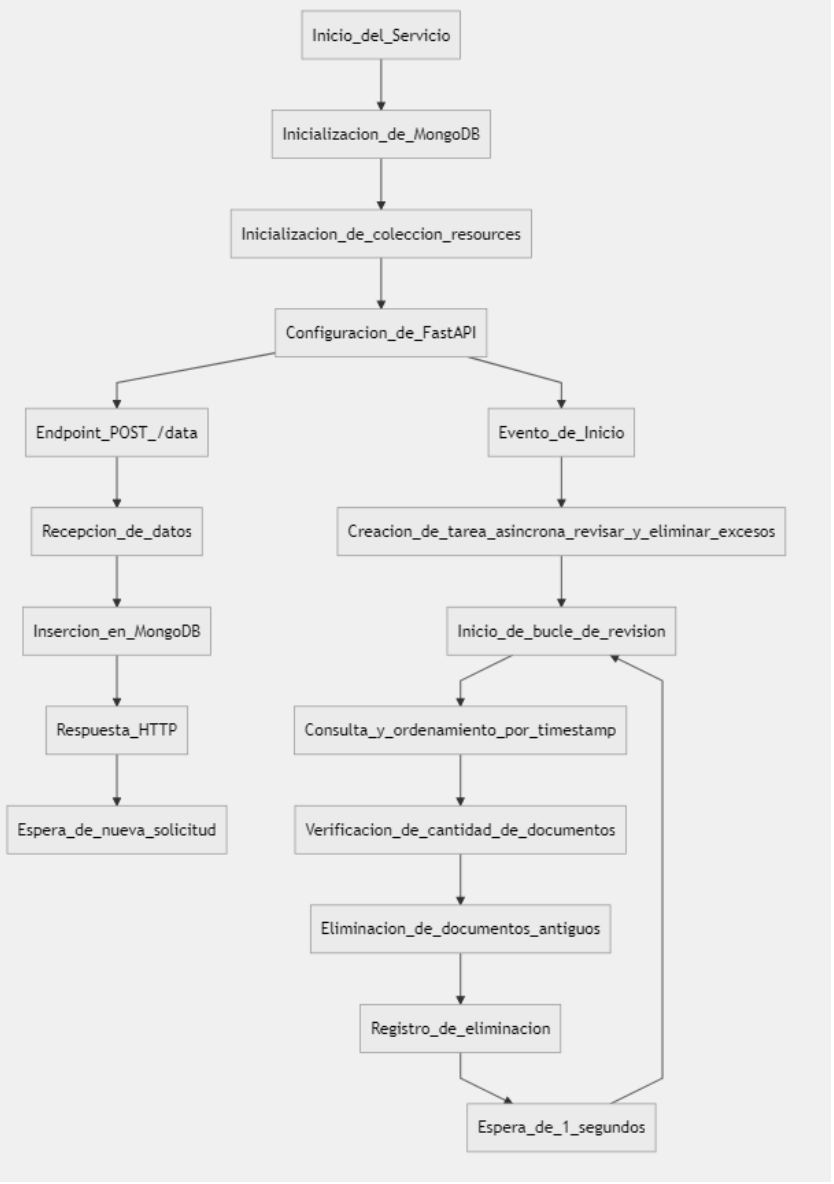
Este módulo consta en la extracción de los recursos (CPU, RAM y Disco) en cada uno de los workers, esta data es enviada a un endpoint expuesta en el headnode para posteriormente ser procesada y guardada en base de datos.
Para el caso de CPU se examina el uso de cada uno de los 8 cores cada segundo, esto permite tener un historial que será usado para la evaluación realizada por el VM Placement Teórico.
Para los casos de RAM y Disco se envía información sobre el total, usado y disponible.
Adicionalmente se añade la etiqueta para identificar cada worker, y también el timestamp correspondiente.
En el headnode se recibe dicha información en formato json, se guarda en la base de datos y se procede a verificar si existen más de 1000 muestras registradas por worker, de ser así se eliminan las más antiguas filtrando por el timestamp y de esa forma quedan las más recientes para ser usadas por el VM Placement y sus reglas implementadas.

APIs RECURSOS DISPONIBLES

Método	Ruta	Acción
POST	/data	Recibe datos JSON y los inserta en la colección

		resources de MongoDB. Además, ejecuta un proceso asincrónico para revisar y eliminar documentos excesivos quédense con los 1000 más actuales fijándose en el timestamp.

HIGH LEVEL SOFTWARE DESIGN RECEPCIÓN Y GESTIÓN RECURSOS DISPONIBLES



2.2.8 Módulo Linux Cluster Driver

El módulo Linux Cluster Driver es responsable de proporcionar funciones de integración con una plataforma de gestión de despliegue de slices de máquinas virtuales (VMs) en un entorno de Linux Cluster. Este entorno consta de un servidor headnode y tres workers. El módulo se integra con Celery para manejar tareas asincrónicas, y utiliza MongoDB para almacenar la configuración y los logs de las operaciones. A continuación se detalla su funcionalidad específica.

- def assign_nodes_to_workers(num_nodes, workers):
  Asigna nodos a los workers utilizando un algoritmo round-robin, balanceando la carga entre los workers disponibles.

- def execute_on_headnode(script):
  Ejecuta scripts en el headnode

- def execute_on_worker(worker_address, script, username, password):
  Conecta a un worker usando SSH y ejecuta el script proporcionado

- @shared_task(bind=True)
  def create(self, slice_id):
  Esta tarea maneja todo el proceso de creación de un VM slice, incluyendo:
  Inicialización de logs.
  Recuperación de la información de la slice desde MongoDB.
  Configuración y ejecución de scripts en el headnode y los workers.
  Actualización de MongoDB con los detalles del despliegue.

- @shared_task(bind=True)
  def delete(self, slice_id):
  Esta tarea maneja el proceso de eliminación de un VM slice, incluyendo:
  Inicialización de logs.
  Recuperación de la información de la slice desde MongoDB.
  Ejecución de scripts de eliminación en el headnode y los workers.
  Eliminación de la información de la slice de MongoDB.

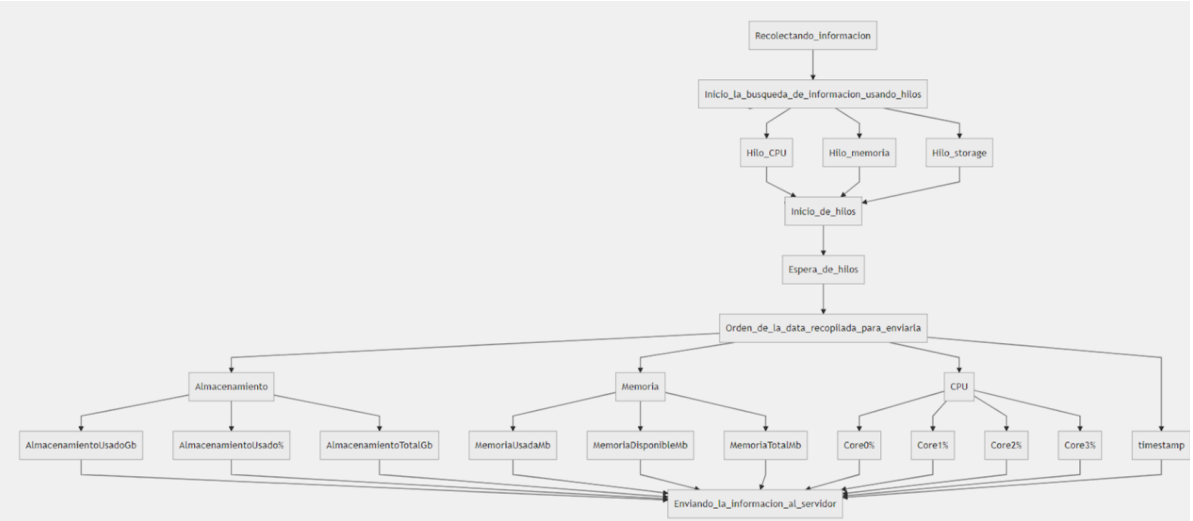
- Las últimas dos funciones utilizan la anotación `@shared_task` debido a que están diseñadas para ejecutarse como tareas asincrónicas gestionadas por Celery.

2.2.9 Módulos de agente worker

Este módulo consiste en un agente en cada worker que actúa como un intermediario entre el módulo de recursos disponibles y el Linux Cluster Driver con el hardware del servidor. Una de sus funciones es recopilar información sobre el uso de recursos del servidor (CPU, memoria y almacenamiento) y proporcionar estos datos al módulo de recursos disponibles.

Además, el agente ejecuta las órdenes del módulo de Linux Cluster Driver para el despliegue de recursos, asegurando que se asignen los recursos de manera eficiente y conforme a las políticas establecidas. También, ejecuta las órdenes para el borrado del slice utilizando Linux Cluster.

HIGH LEVEL SOFTWARE DESIGN AGENTE WORKER - RECURSOS



2.2.10 Módulo OpenStack Driver

Este módulo proporciona funciones de integración con una plataforma de Cloud Provider, permitiendo al sistema interactuar con los recursos de dicho proveedor, como la creación y gestión de instancias de VMs, a través de su API. En particular, en el proyecto se usará OpenStack. Para el despliegue y borrado de slices, se han integrado funciones que utilizan las APIs de los siguientes módulos de OpenStack: Nova, Neutron, Keystone y Glance. Para el VM placement, se ha habilitado la configuración default de OpenStack en su módulo Placement. Luego, para la interacción y verificación de asignación de recursos, se recurre a la interfaz gráfica Horizon.

Este módulo tiene las siguientes funciones principales que usan las funciones primarias de nuestro OpenstackSDK.py (llamadas directas a las APIs):

Función openstackDeployment
Esta función se encarga de desplegar una topología de slice en OpenStack. El flujo general de esta función es el siguiente:

1. Obtener el nombre y descripción del proyecto:

    - project_name = slice_json["name"]
    - project_description = slice_json["description"]

2. Autenticación como administrador:

    - admin_token = authenticate_admin()

3. Creación del proyecto en OpenStack:

    - project = crear_proyecto(admin_token, DOMAIN_ID, project_name, project_description)
    - Obtiene el project_id del proyecto creado.

4. Asignación de roles:

    - Asigna el rol de administrador al usuario administrador en el nuevo proyecto.
    - Obtiene un token del proyecto.
    - Obtiene el user_id del usuario por user_name.
    - Asigna el rol de miembro al usuario en el proyecto.

5. Creación de la topología de slice:

    - Llama a create_slice_topology(slice_json["topology"], project_token, project_id) para crear la topología de red y desplegar las instancias virtuales (VMs) según el JSON de la topología.

Función openstackDeleteSlice
Esta función se encarga de eliminar una topología de slice en OpenStack y el proyecto asociado. El flujo general es el siguiente:

1. Autenticación como administrador:

    - admin_token = authenticate_admin()

2. Autenticación del proyecto y obtención del token del proyecto:

    - project_token = authenticate_project(admin_token, project_name)

3. Obtención del ID del proyecto:

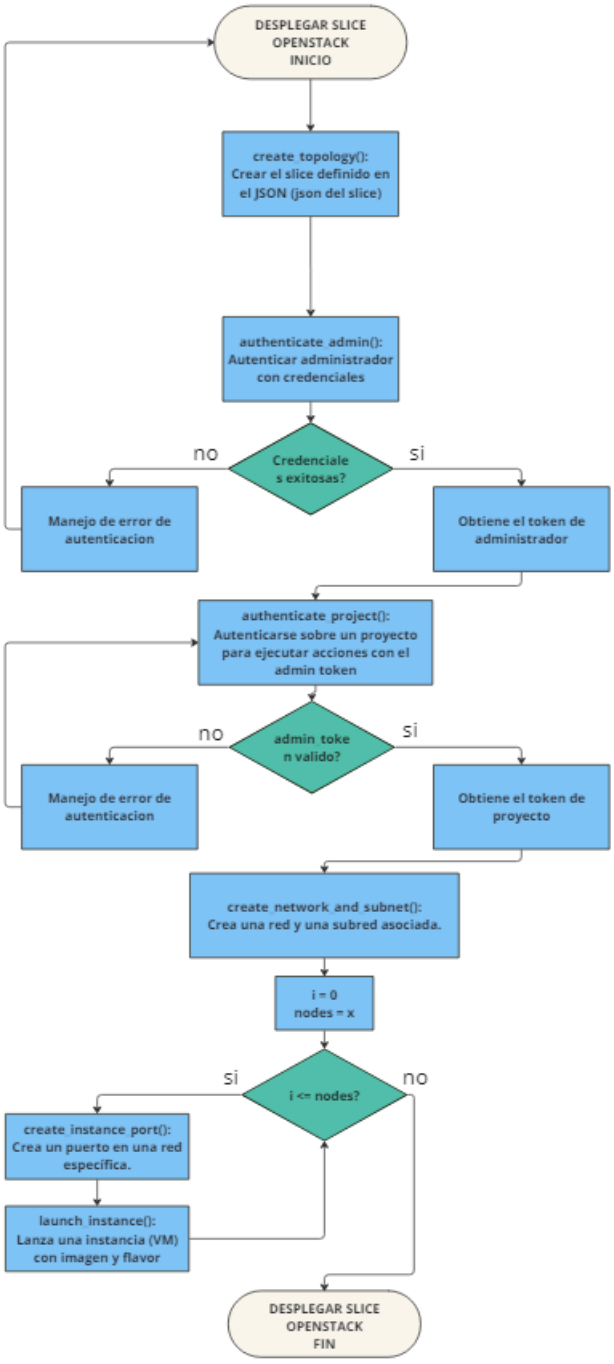
    - project_id = obtenerIdProyecto(project_token, project_name)
4. Eliminación de la topología de slice:

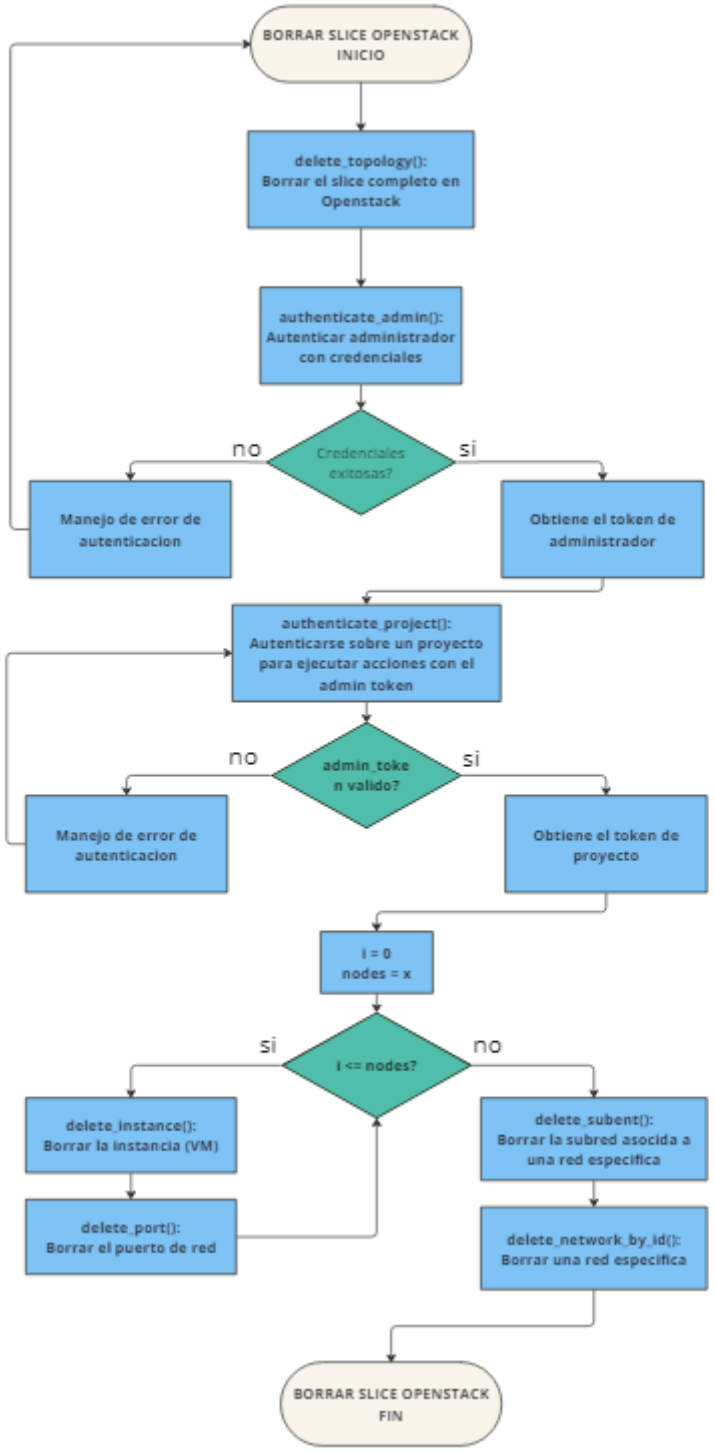
    - Llama a delete_slice_topology(project_token, project_id) para eliminar todas las instancias, puertos, subredes y la red asociada a la topología.

5. Eliminación del proyecto:

- Llama a deleteProject(admin_token, project_id) para eliminar el proyecto en OpenStack.

HIGH LEVEL SOFTWARE DESIGN OPENSTACK DRIVER




BORRAR SLICE OPENSTACK INICIO
delete_topology(): Borrar el slice completo en Openstack
authenticate_admin(): Autenticar administrador con credenciales
Credenciales exitosas?
no
si
Manejo de error de autenticacion
Obtiene el token de administrador
authenticate_project(): Autenticarse sobre un proyecto para ejecutar acciones con el admin token
admin_token valido?
no
si
Manejo de error de autenticacion
Obtiene el token de proyecto
i = 0
nodes = x
i <= nodes?
si
no
delete_instance(): Borrar la instancia (VM)
delete_port(): Borrar el puerto de red
delete_subent(): Borrar la subred asocida a una red especifica
delete_network_by_id(): Borrar una red especifica
BORRAR SLICE OPENSTACK FIN

3. FLUJO DE CASOS DE USO

ROLES DE USUARIO

Administrador

El rol de administrador corresponde al usuario encargado de monitorear y gestionar la infraestructura de nube y sus recursos. Este rol posee facultades para listar (con información detallada), crear y borrar los slices de VMs de todos los usuarios clientes que estén utilizando la infraestructura. Asimismo, puede observar y gestionar el consumo de recursos de la infraestructura de nube, pudiendo identificar sobrecargas en algún servidor, realizar troubleshooting a través de logs.

Cliente

El rol de cliente corresponde al usuario que hace uso de los recursos computacionales ofrecidos a través de la nube privada. Este rol posee facultades para listar, crear, editar y borrar sus slices de VMs, ya sea, se encuentren guardados como plantillas o estén ejecutándose. Dentro de las topologías definidas presenta las de tipo árbol, anillo, bus, malla y lineal de donde podrá partir para poder editar/crear su propia topología de slice. También puede elegir dentro de las zonas de disponibilidad disponibles para poder crear su slice, siempre y cuando se cuente con los recursos necesarios para la creación.

3.1 AUTENTICACIÓN

1) El usuario inicia el CLI e ingresa sus credenciales (username y password) para solicitar autenticación en nuestro sistema y acceder a las funciones que ofrece la interfaz.

2) La solicitud de autenticación se envía en formato json hacia el endpoint de autenticación (/login) que expone el módulo API GATEWAY / AAA.

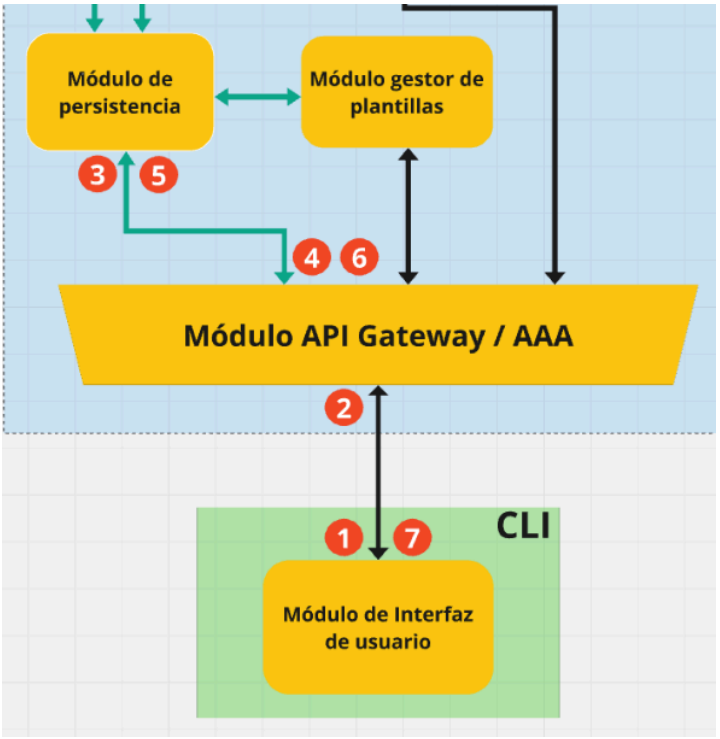
3) Este módulo recibe la solicitud de autenticación y se encarga de verificar la existencia del usuario en la BD mediante un query al módulo de persistencia (MongoDB), utilizando las credenciales recibidas en la solicitud de autenticación.

4) De haber confirmado la existencia del usuario utilizando el campo username, el módulo de persistencia devuelve el hash de la contraseña del usuario almacenada en la BD. Una vez recibido el hash almacenado, el módulo API GATEWAY / AAA realiza una comparación con la contraseña recibida mediante la solicitud de autenticación. De ser equivalentes a la misma contraseña, se confirma la autenticación del usuario y el módulo genera un token de sesión para dicho usuario.

5) El token de sesión es guardado en base de datos; específicamente, en el documento correspondiente al usuario. Además, se realiza un query a BD solicitando los siguientes datos del usuario: uuid, username, rol y token de sesión.

6) Los cuatro datos solicitados son devueltos al módulo API GATEWAY / AAA para su posterior redireccionamiento al CLI del usuario.

7) El CLI recibe una respuesta en formato json del módulo API GATEWAY / AAA indicando la autenticación exitosa e incluyendo los datos anteriormente solicitados de: uuid, username, rol y token de sesión. Dependiendo del rol, el usuario tendrá acceso a determinadas opciones de CRUD que ofrece el CLI.



3.2 GESTIÓN DE PLANTILLAS

1) El usuario realiza una operación CRUD de plantillas desde el menú que ofrece el CLI (e.g. creación de plantilla). Se necesita estar autenticado previamente.

2) La solicitud, en formato json, de la operación CRUD es dirigida al módulo API GATEWAY / AAA. Este json contiene el método http, el nombre de la acción a realizar (una acción CRUD), los datos de la operación CRUD realizada por el usuario y la cabecera X-API-Key. Es este último parámetro el que permite la autenticación y autorización de los usuarios.

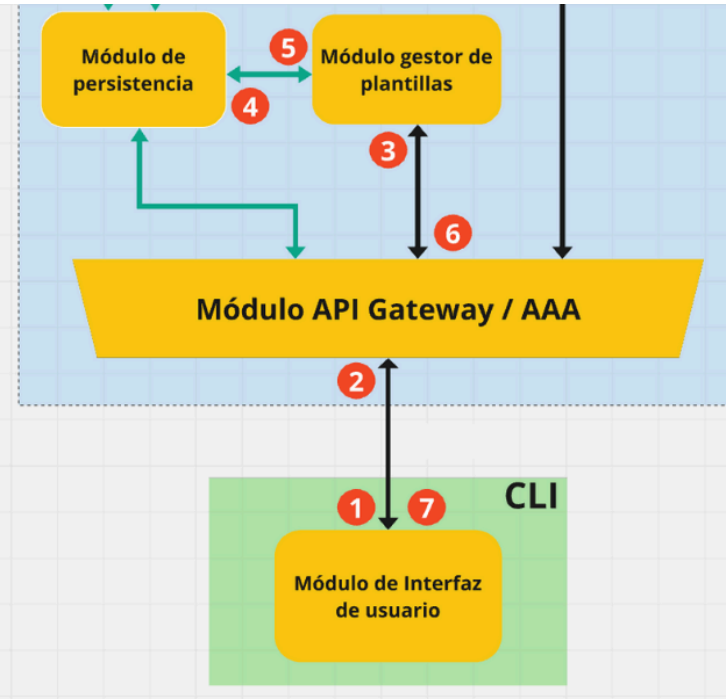
3) El módulo API GATEWAY / AAA reenvía la solicitud al módulo gestor de plantillas. En particular, a la ruta del endpoint que corresponde según la indicación de la acción CRUD del json bajo el método dado por el mismo.

4) El módulo gestor de plantillas recibe la solicitud de operación CRUD redirigida por el módulo API GATEWAY / AAA. Este gestor lista/modifica datos de la BD del módulo de persistencia según los datos de operación CRUD recibidos de la solicitud.

5) El módulo de persistencia devuelve al módulo gestor de plantillas la data solicitada o con la confirmación de data modificada según sea el caso.

6) El gestor de plantillas responde, al módulo API GATEWAY / AAA, la solicitud de operación CRUD con los datos obtenidos (listado) o confirmación de modificación realizada (creado, borrado, actualización) en formato json.

7) Finalmente, el módulo API GATEWAY / AAA redirecciona la respuesta del gestor de plantillas al CLI. Este último realiza el formateo del json para mostrar el mensaje en pantalla.



3.3 DESPLIEGUE DE SLICES

3.3.1 DESPLIEGUE LINUX CLUSTER

1) El usuario selecciona desde su CLI la plantilla que desea desplegar. Para lograrlo, deberá listar las plantillas existentes y cada plantilla tendrá un template_id que las identifica de forma única. Mediante la opción “crear slice” del CRUD de slices, escoge la plantilla a desplegar de la tabla de plantillas creadas por el usuario.

2) Se envía la solicitud de despliegue de slice en formato json hacia el módulo API GATEWAY / AAA. Esta solicitud contiene como parámetro el template_id de la plantilla, un parámetro para indicar el tipo despliegue (en este caso Linux Cluster) y la indicación del método http (POST) del endpoint que expone el gestor de slices.

3) El módulo API GATEWAY / AAA redirecciona la solicitud de despliegue de slice hacia el endpoint (POST method con parámetro template_id y tipo de despliegue) que expone el gestor de slices.

4) El módulo gestor de slices verifica la existencia de la plantilla con el módulo de persistencia utilizando el template_id para realizar el query.

5) De no existir, se envía una respuesta de resultado “error” y mensaje de nula existencia hacia el API GATEWAY / AAA, redirigido al CLI que realizó la solicitud. Caso contrario, de existir la plantilla, el módulo gestor de slices recoge sus parámetros de configuración para iniciar el proceso de despliegue según el tipo indicado por el parámetro anteriormente enviado.

6) De existir, el módulo gestor de slices ejecuta el despliegue del slice como una tarea asíncrona con Celery. Envía esta tarea hacia el módulo de sistema de colas. Este módulo actúa como un broker de solicitudes, es decir, un sistema de colas que proporciona un orden secuencial de todas las peticiones que se dirigen hacia el módulo de Linux Cluster Driver. Asimismo, envía una respuesta a la petición del cliente con el id de la tarea que está esperando en la cola y un mensaje de la inicialización del despliegue. De esta manera, se evitan los cuellos de botella en el servidor flask.

7) El módulo de Linux Cluster Driver recibe la petición de despliegue de manera secuencial. En esta petición se encuentra el id de la plantilla a desplegar.

8) Se consulta la base de datos para obtener información relevante de la plantilla. como la cantidad de nodos que la conforman.

9) En el driver del cluster de servidores se recibirá la solicitud correspondiente a la información de la plantilla.

10) La cantidad de nodos se envía al módulo de scheduler para que tome la decisión sobre el despliegue en los workers (round robin implementado).

11) El módulo Linux Cluster driver obtiene una respuesta del scheduler, con los nodos asignados a los workers. Con esta información, ejecuta script de manera local para empezar el despliegue.

12) El módulo Linux Cluster ejecuta scripts se encuentran en los workers de manera remota para el despliegue de las VMs en los mismos.

13) A partir de la ejecución de estos scripts, el Linux Cluster driver recibe una respuesta con la información de las VMs desplegadas, como el proceso, puerto vnc expuesto.

14) Este linux driver guarda en base de datos la información de configuración del slice desplegado. Asimismo, guarda el estado final de la tarea realizada.

15) Desde el cli (postman), se envía una solicitud al endpoint en el slice manager que actualiza sobre el estado de la tarea de despliegue del slice.

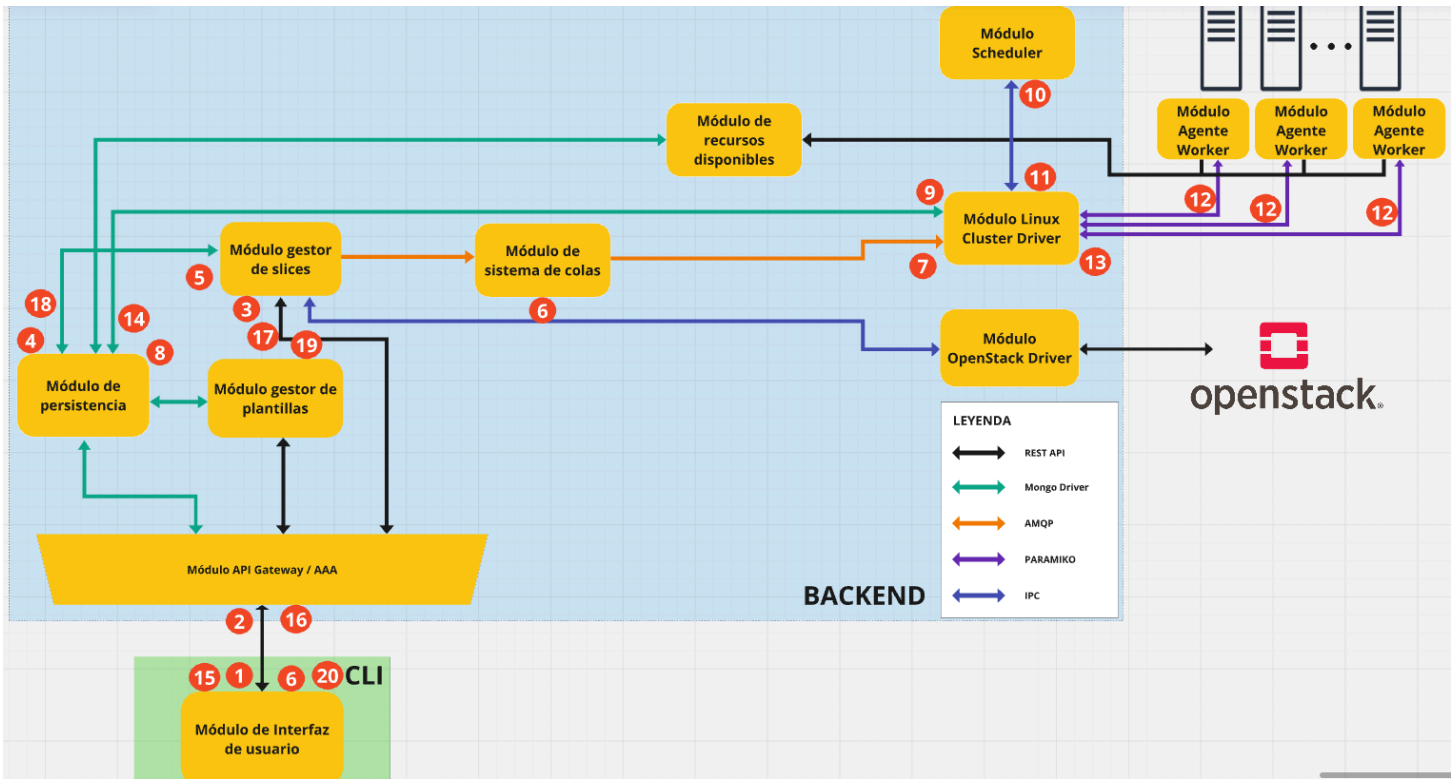
16) El módulo API GATEWAY / AAA redirecciona la solicitud del estado de la tarea hacia el endpoint (GET method con parámetro task_id) que expone el gestor de slices.

17) El módulo gesto de slices recibe la solicitud y verifica el estado de la tarea en base de datos.

18) En base de datos, en la colección celery_taskmeta se tienen los documentos guardados que representan las tareas.

19) Se recibe esta tarea con su respectivo estado y se reenvía al cliente que realizó la solicitud.

20) Se recibe el estado de la tarea de despliegue del slice respectivo.



3.3.2 DESPLIEGUE OPENSTACK

1) El usuario selecciona desde su CLI la plantilla que desea desplegar. Para lograrlo, deberá listar las plantillas existentes y cada plantilla tendrá un template_id que las identifica de forma única. Mediante la opción “crear slice” del CRUD de slices, escoge la plantilla a desplegar de la tabla de plantillas creadas por el usuario.

2) Se envía la solicitud de despliegue de slice en formato json hacia el módulo API GATEWAY / AAA. Esta solicitud contiene como parámetro el template_id de la plantilla, un parámetro para indicar el tipo despliegue (en este caso Openstack) y la indicación del método http (POST) del endpoint que expone el gestor de slices.

3) El módulo API GATEWAY / AAA redirecciona la solicitud de despliegue de slice hacia el endpoint (POST method con parámetro template_id y tipo de despliegue) que expone el gestor de slices.

4) El módulo gestor de slices verifica la existencia de la plantilla con el módulo de persistencia utilizando el template_id para realizar el query.

5) De no existir, se envía una respuesta de resultado “error” y mensaje de nula existencia hacia el API GATEWAY / AAA, redirigido al CLI que realizó la solicitud. Caso contrario, de existir la plantilla, el módulo gestor de slices recoge sus parámetros de configuración para iniciar el proceso de despliegue según el tipo indicado por el parámetro anteriormente enviado.

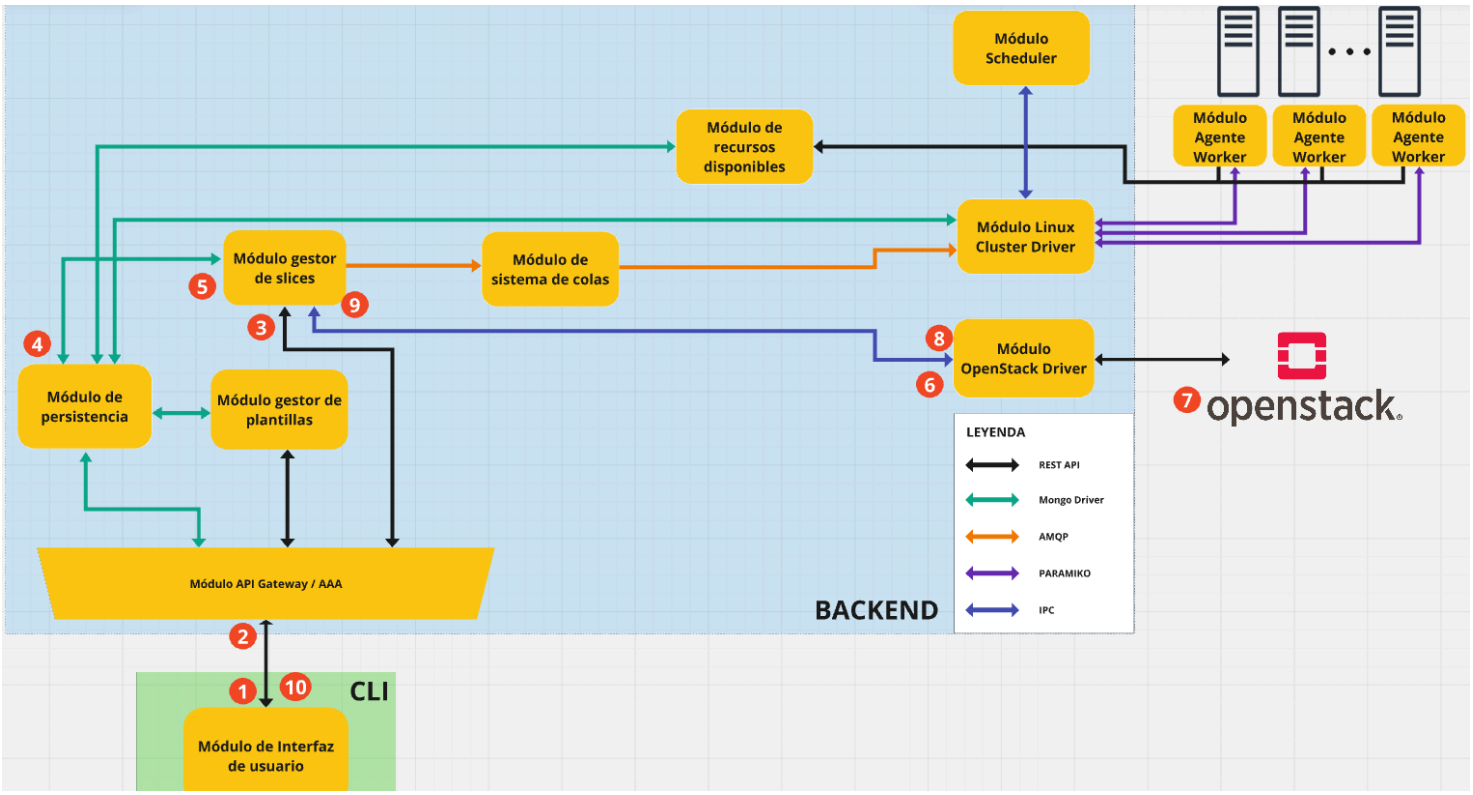
6) El módulo gestor de slices invoca a la función *openstackDeployment(new_slice, user_name)* del módulo OpenStack Driver para pasarle como parámetros la variable de configuración del slice y el nombre del usuario dueño del slice a desplegar.

7) La función *openstackDeployment(new_slice, user_name)* utiliza las APIs que expone OpenStack para realizar el despliegue del slice. El orden de despliegue es: creación de redes, creación de subredes, creación de puertos y creación de instancias.

8) La función *openstackDeployment* termina el despliegue y devuelve la confirmación de la misma hacia el gestor de slices.

9) El gestor de slices recibe la confirmación de despliegue en OpenStack y prepara el retorno de la respuesta hacia el CLI del usuario. Asimismo, guarda en base de datos la información de configuración del slice desplegado.

10) El CLI del usuario recibe un json de confirmación de despliegue del slice y le retorna el id del mismo. Puede verificar su creación mediante el listado de slices y el acceso a Horizon usando sus credenciales. Además, puede solicitar las URLs de acceso VNC hacia las consolas de las VMs desplegadas por OpenStack.


Módulo Scheduler
Módulo de recursos disponibles
Módulo Agente Worker
Módulo Agente Worker
Módulo Agente Worker
Módulo Linux Cluster Driver
Módulo gestor de slices
Módulo de sistema de colas
Módulo de persistencia
Módulo gestor de plantillas
Módulo OpenStack Driver
openstack
Módulo API Gateway / AAA
BACKEND
LEYENDA
REST API
Mongo Driver
AMQP
PARAMIKO
IPC
CLI
Módulo de Interfaz de usuario