Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Gestión de Máquinas Virtuales en el IaaS de la ULL

Management of the Virtual Machines in the IaaS of the ULL

Pablo González González

La Laguna, 14 de julio de 2023

D. **Vicente Blanco Pérez**, con N.I.F. 42.171.808-C profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas de la Universidad de La Laguna, como tutor

D. **Juan Carlos Pérez Darias**, con N.I.F. 45.441.625-L profesor Titular de Universidad adscrito al Departamento de Física de la Universidad de La Laguna, como cotutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Gestión de Máquinas Virtuales en el IaaS de la ULL*"

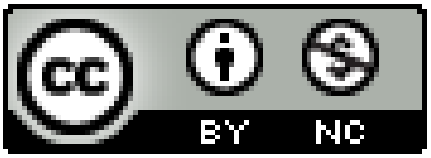
ha sido realizada bajo su dirección por D. **Pablo González González**, con N.I.F. 51.148.320-T.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 14 de julio de 2023

Agradecimientos

A mi familia y amigos, sobre todo a mis padres, que siempre me han apoyado y empujado a conseguir esto.
A mis tutores Vicente y Juan Carlos, que han tenido paciencia y me han guiado y ayudado de la mejor manera posible.

Licencia



Resumen

En este proyecto se ha desarrollado la parte backend de una aplicación web que utiliza el IaaS de la ULL para dar servicio a los usuarios a través de Internet a través de la plataforma oVirt. El IaaS es usado en ámbitos docentes y de investigación debido a la amplia oferta de recursos computacionales.

En la ULL, profesores y alumnos utilizan el IaaS para acceder de forma remota a los recursos, principalmente máquinas virtuales. Estos recursos están gestionados por el STIC. Los profesores necesitan cierto poder de administración sobre los recursos para dotar a todos los alumnos de entornos con las mismas características y configuraciones. Además, los alumnos deben controlar dichas máquinas con facilidad, teniendo capacidad de encender, apagar o borrar, entre otras.

Actualmente, la aplicación web utilizada para acceder al IaaS presenta limitaciones en cuanto a las funcionalidades que debería cubrir. Además, la estructura y stack tecnológico utilizado la convierten en una aplicación difícil de mantener.

The proposed application supports the most common activities performed in the IaaS in a modern and updated manner, both for management and administration by teachers, and for control and usage by students. It is developed in Node.js, which handles user requests and converts them into actions on oVirt. To achieve this, it utilizes the automation tool Ansible.

Palabras clave: IaaS, oVirt, Ansible, Backend, API, Máquina virtual

Abstract

In this project, the backend part of a web application has been developed, which utilizes ULL's IaaS to provide service to users over the Internet through the oVirt platform. IaaS is used in educational and research contexts due to its wide range of computational resources.

At ULL, teachers and students use IaaS to remotely access resources, primarily virtual machines. These resources are managed by the STIC[49]. Teachers require a certain level of administrative power over the resources to provide all students with environments of the same characteristics and configurations. Additionally, students need to easily control these machines, having the ability to power on, power off, or delete them, among other actions.

Currently, the web application used to access the IaaS has deficiencies in terms of the functionalities it should cover. Additionally, the structure and technological stack employed make it a difficult application to maintain.

La aplicación propuesta da soporte a las actividades más comunes que suelen realizarse en el IaaS de una forma moderna y actualizada. Está desarrollada en Node.js, que se encarga de convertir las peticiones de los usuarios en acciones sobre oVirt. Para ello utiliza la herramienta de automatización Ansible.

Keywords: IaaS, oVirt, Ansible, Backend, API, Virtual machine

Índice general

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

En este apartado se abordarán los antecedentes y el estado del arte actual relacionados con el proyecto. Además, se comentará la motivación y los propósitos que dieron lugar a la propuesta de este proyecto, así como los objetivos que se esperan lograr. Finalmente, se describirá la metodología seguida en el desarrollo del proyecto.

1.1. Antecedentes y estado del arte actual

Las plataformas IaaS[13] (Infrastructure as a Service) son servicios en la nube que ofertan a los usuarios infraestructuras de cómputo remoto, proporcionando almacenamiento, procesamiento y comunicaciones a través de Internet. Este tipo de plataformas aportan la virtualización, el almacenamiento, la red y los servidores, por lo que el usuario no tiene que preocuparse por actualizar o mantener esos elementos físicos.

En la actualidad, las empresas usan IaaS principalmente para las siguientes actividades:

- **Alojamiento web**: proporcionan servicios de alojamiento que permiten a las empresas publicar sus sitios y aplicaciones webs en la nube.
- **Almacenamiento y respaldo de datos**: las empresas pueden almacenar y respaldar sus datos de manera eficiente, sin la preocupación de administrar los servidores.
- **Procesamiento y análisis de datos (*big data*)**: permite el análisis y procesamiento de grandes volúmenes de datos al una infraestructura escalable con recursos de computación y almacenamiento bajo demanda.
- **Entornos de desarrollo y pruebas**:las empresas pueden crear una gran cantidad de instancias virtuales, replicar entornos de producción y realizar pruebas exhaustivas.

El módelo IaaS se basa en la nube, característica que lo dota de alta flexibilidad, escalabilidad y accesibilidad, entre otros. Dentro de la nube, se distinguen dos tipos de servicios dependiendo de la forma de acceso sobre los recursos:

- **Nube pública**: es propiedad de proveedores de servicios en la nube, como Amazon Web Service[7], Microsoft Azure[8] o Google Cloud Platform[21]. Estas empresas son responsables de la administración de la infraestructura, el mantenimiento y la seguridad. Ofrecen a los usuarios recursos físicos y lógicos, que son compartidos entre todos.

- **Nube privada**: está gestionada por una organización específica y los recursos están dedicados exclusivamente a esa organización.

La Universidad de La Laguna usa un modelo de nube privada para brindar a los alumnos y profesores los servicios que se exponen en el IaaS. A través del *cloud computing*[10] se ofrece a los usuarios utilidades como el almacenamiento de datos, procesamiento y poder de cómputo, y despliegue de aplicaciones, entre muchas otras. Estas utilidades son utilizadas principalmente en los ámbitos de investigación y docencia.

Para dar soporte a este servicio la ULL utiliza la plataforma de código abierto oVirt[37], que proporciona máquinas virtuales. Especificando en la docencia, estas máquinas virtuales permiten a los alumnos utilizar software específico para asignaturas particulares, como Visual Studio Code[54], así como acceder a entornos relacionados con la administración de sistemas o aplicaciones dedicadas a la robótica, entre otros. Además, también facilita a los estudiantes el acceso remoto y bajo demanda a laboratorios y recursos de asignaturas. Esto crea un entorno homogéneo para todos los estudiantes y facilita la labor del profesorado, ya que las máquinas son preconfiguradas por los profesores de acuerdo a las necesidades de cada asignatura.

Los dos servicios principales ofertados por oVirt son:

- ***Pool* de máquinas**: conjunto de máquinas virtuales que se crean y administran de manera conjunta. En un *pool* de máquinas todos los nodos poseen la mismas configuración, lo que facilita su gestión.

- ***Pool* de recursos**: conjunto de recursos físicos, como servidores y almacenamiento, que también se gestionan de manera conjunta. El objetivo es la agrupación de los recursos para hacer una asignación eficiente a las máquinas virtuales.

Las máquinas virtuales que posee un alumno en particular se muestran en el portal de usuario que se conecta al IaaS de la ULL[22]. Estas máquinas pueden ser visualizadas a través de dos aplicaciones *frontend*: una versión general que muestra todas las máquinas con las acciones posibles de ejecución y configuración, y una versión resumida, conocida como Mis máquinas[23], que lista las máquinas asociadas a un usuario junto con sus direcciones IP. Ambas aplicaciones están conectadas a un *backend* que recibe las solicitudes del usuario y las transforma en las acciones deseadas. Estas acciones se ejecutan mediante el uso de scripts desarrollados en el lenguaje de programación Python[45].

Este proyecto se enfocará tanto en el desarrollo de la aplicación *backend* como en la utilización de Ansible. Se creará una API[5] que se conectará a las aplicaciones *frontend* del portal de usuario que da acceso al IaaS y escuchará las peticiones. A partir de ahí, se seleccionarán y ejecutarán las acciones de Ansible para interactuar con la máquina virtual y satisfacer las solicitudes requeridas.

1.2. Objetivos

En la actualidad, la aplicación utilizada para acceder al IaaS de la ULL presenta ciertas carencias, como la falta de funcionalidades y una experiencia de usuario deficiente, entre otras. Los profesores se encuentran con problemas recurrentes como la necesidad de restablecer la contraseña de los alumnos debido a su pérdida o la preparación de una plantilla configurada para el laboratorio de su asignatura concreta. Por su parte, los alumnos deben poder manejar el estado de las máquinas de una manera renovada y sencilla.

Estos problemas se pretenden solucionar mediante la remodelación de la aplicación de *frontend* (la parte visible con la que el usuario interactúa directamente), lo que implica también la actualización de la parte del *backend*. De esta manera, los objetivos del proyecto se derivan de esta situación: crear una nueva API que cumpla con todas las especificaciones requeridas, brinde soporte a las funcionalidades esperadas en la primera versión y sea escalable. El análisis de los requisitos se detallará en el Capítulo 3.

1.3. Metodología

Para realizar este proyecto, se han seguido las siguientes fases:

- **Análisis de las funcionalidades:** fase de definición del alcance del proyecto y las acciones que se cubrirán.
- **Análisis de herramientas y tecnologías:** investigación y selección de las herramientas y tecnologías a utilizar en base a las necesidades y requisitos del proyecto.
- **Diseño y desarrollo de los playbooks:** programación de las funcionalidades específicas.
- **Diseño y desarrollo de la aplicación *backend* de gestión:** creación de la aplicación que conectará la plataforma oVirt con el portal con el que interactúa el usuario.
- **Testeo y despliegue:** verificación del correcto funcionamiento del conjunto.

Este desarrollo se ha organizado utilizando una herramienta llamada Github Projects[20], que mediante un tablero de tareas permite llevar un seguimiento del trabajo, incluyendo las tareas por realizar, las que están en progreso y las que ya se han completado.

La metodología y ciclo de vida del desarrollo del proyecto son explicados con profundidad en la Sección 2.3.

Capítulo 2

Herramientas, tecnologías y metodología

En este capítulo se introducirán las herramientas y tecnologías que han sido usadas durante el desarrollo de la aplicación. Además, se comentará la metodología usada para dicho desarrollo.

2.1. Herramientas

Antes del comienzo del desarrollo, como se comentó anteriormente, se realizó un estudio y análisis sobre qué tecnologías y herramientas serían usadas en el proyecto. Comenzando con el entorno de desarrollo, se ha elegido el editor de texto Visual Studio Code de Microsoft[33] frente a otras soluciones más potentes como podría ser WebStorm[55] de JetBrains[27] (perteneciente a la suite de IntelliJ[25], pero especializado en JavaScript) o NetBeans[34] de Apache[4] debido, principalmente, a su merecida fama y gran comunidad. Gracias a todos sus usuarios, este editor está en continuo avance y posee un mercado de extensiones que permiten añadir características de una forma muy sencilla, llegando a convertirlo en un programa más cercano a un IDE que a un editor de texto.

Por su parte, para el control de versiones, se ha utilizado el sistema Git[18] en vez de Subversion[50] debido a su carácter distribuido, que permite el trabajo local sin depender del servidor central ni de la conexión a Internet, además de la facilidad para trabajar con ramas que ofrece.

En cuanto a la descripción de la API, se ha utilizado Swagger.io[51], que se trata de un framework para documentar el funcionamiento de la API. Siendo open source, Swagger.io se basa en el estándar de especificación OpenAPI 3.0[36] para ofrecer una interfaz de usuario que ayuda a especificar cómo funcionan los *endpoints*, es decir, qué datos se esperan en qué URL y cuáles devuelven.

Por último, se ha elegido ESLint[17] y Prettier[44] para limpiar y formatear el código, respectivamente. Ambas herramientas son líderes en su campo y conviven muy bien juntas, y sirven de base para crear un código legible y sin errores sintácticos.

2.1.1. Visual Studio Code

Visual Studio Code es un editor de código fuente gratuito desarrollado por Microsoft y disponible en Windows, Linux y macOs. Se trata del editor de código por excelencia en la actualidad debido a sus características únicas, como pueden ser:

- Soporte para una extensa cantidad de lenguajes de programación
- Intelli-Sense: característica que ayuda a programar mediante la utilización de *snippets* o fragmentos de códigos reutilizables.
- Soporte multiplataforma que permite trabajar en cualquier sistema operativo.
- Mercado de extensiones, tanto gratis como de pago, que aportan funcionalidades distintas o mejoradas, creando un entorno más amplio y potente.
- Herramientas integradas como Git o la *terminal*.

Dentro de esta aplicación se usaron extensiones como Prettier o ESLint, que serán comentadas a continuación.

2.1.2. Github

Github[19] ha sido la herramienta escogida para el control de versiones de este proyecto. Es un portal creado para alojar el codigo de cualquier desarrollador, siempre y cuando deseen que sus proyectos sean de código abierto. Ésta es una de las principales ventajas de la plataforma, ya que cualquier usuario puede buscar y observar cualquier trabajo subido, e incluso contribuir si el propietario lo permite. Por otra parte, Github ofrece una serie de herramientas propias que aportan funcionalidades a los proyectos, como pueden ser la posibilidad de crear una *Wiki*, de crear *Issues* para seguir el desarrollo con sus *bugs* y mejoras, o acompañar al código con un tablero donde reunir las tareas que se realizan durante el desarrollo.

2.1.3. Postman

Postman es principalmente una herramienta para realizar peticiones simuladas a una API, dando la posibilidad de generar colecciones de peticiones para probarlas de una manera rápida y sencilla. Se puede definir el tipo de petición HTTP que se desea realizar (GET, POST, UPDATE...), así como el *body*, los *headers* o incluso *tokens* de autenticación, en caso de ser necesario. Además, permite definir variables y construir un entorno de trabajo para crear una situación lo más cercana posible a la realidad. Por último, todo puede ser exportado en múltiples lenguajes de una manera sencilla.

2.1.4. Jest

Jest es una librería de *testing* de JavaScript desarrollada por Facebook. Entre sus características destacan la facilidad de configuración, su velocidad debida a la

paralelización y priorización de las pruebas más lentas, y la capacidad de hacer una captura o *snapshot* de un momento o estado específico de la aplicación, dando así soporte a la prueba de funcionalidades más complejas.

2.1.5. Swagger

Swagger es un *framework* para diseñar, contruir, documentar y utilizar APIs RESTful, y entre sus funciones se incluye la documentación automatizada, la generación de código y la generación de casos de prueba. Swagger se basa en el estándar de especificación OpenAPI utilizado para crear un lenguaje que se entienda tanto por parte de la máquina como del desarrollador, y que permite describir los *endpoints* disponibles y cómo funcionan, los parámetros necesarios para cada operación o los métodos de autenticación, entre otros. En este caso de siguió la versión 3.0 del estándar.

2.1.6. ESLint

El *linting* consiste en revisar el código en busca de errores en sintaxis, código difícil de entender, malas prácticas o estilos de programación inconsistentes. Estas herramientas forman partes de un grupo de programas que se conoce como herramientas de análisis estático, cuya principal funcionalidad es el análisis de un programa antes de ser ejecutado. Para llevar a cabo esta tarea dentro del proyecto se ha decidido optar por ESLint, una herramienta de código abierto enfocada en realizar dicho proceso en JavaScript.

2.1.7. Prettier

Prettier se trata de una herramienta de formateo automático de código. Se utiliza para mantener una guía de estilo en los proyectos, evitando así la disminución de productividad que puede producir el cambio constante de formas de escribir código.

2.2. Tecnologías

2.2.1. oVirt

oVirt[37] es un sistema de código abierto fundado por la compañía Red Hat basado en el hipervisor KVM[31], que se encarga de crear y ejecutar maquinas virtuales, y la biblioteca libvirt[32] para implementar, mantener y monitorear un conjunto de máquinas virtuales y administrar la infraestructura de la nube.

2.2.2. Ansible

Ansible es una herramienta *open source* de automatización utilizada para tareas como la gestión de la configuración de máquinas, el despliegue de aplicaciones, la orquestación de procesos y otras muchas operaciones relacionadas con la tecnología informática. En este proyecto se ha utilizado Ansible principalmente para la automatización de todas las tareas relacionadas con la gestión de las

máquinas virtuales. El funcionamiento consiste en conectarse de forma remota y ejecutar instrucciones que de otra manera deberían haberse hecho de forma manual.

2.2.3. Node.js

Node.js es un entorno *open source* de tiempo de ejecución de JavaScript. JavaScript[26] es un lenguaje de programación interpretado utilizado principalmente en el ámbito web. A raíz de su crecimiento se creó Node.js para permitir la ejecución de programas escritos en JavaScript fuera del entorno web, es decir, como aplicaciones de escritorio. Su funcionamiento se basa en un modelo de entrada y salida de datos sin bloqueo manejado por eventos. La gran ventaja que tiene es que opera en un solo subproceso, haciéndolo muy liviano y eficiente. Además, cuando una solicitud bloquea la entrada y salida, en vez de esperar al desbloqueo, comienza a procesar la siguiente solicitud y retoma la anterior tan pronto como se quede libre.

2.2.4. NPM

NPM[35] (*Node Package Manager*) es el sistema de gestión de paquetes por defecto para Node.js. Se trata de un repositorio online donde la comunidad publica paquetes de software libre accesibles desde cualquier proyecto que trabaje con Node.js. Además, proporciona las herramientas necesarias para interactuar con dicho repositorio y gestionar las instalación de utilidades, el manejo de dependencias y la publicación de paquetes.

2.2.5. Express.js

Express.js es un *framework* de Node.js diseñado para construir la parte *backend* de una aplicación web. Es una librería rápida y minimalista que proporciona herramientas robustas para desarrollar de una forma escalable. Por defecto, ofrece características como el sistema de enrutamiento, peticiones y respuestas HTTP y la posibilidad de añadir *middlewares*, que son herramientas que aportan funcionalidades concretas ya desarrolladas. Se observa un esquema de su funcionamiento en la Figura 2.1.

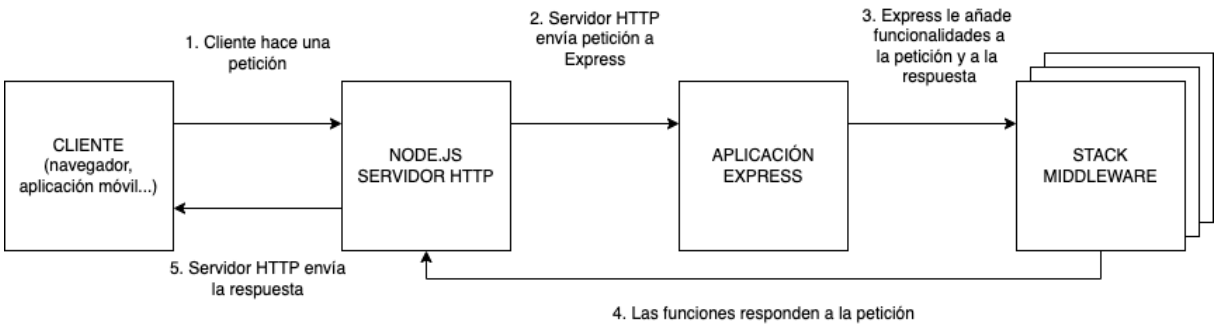


Figura 2.1: Esquema de funcionamiento de Express.js

2.2.6. Jinja2

Jinja2[28] es un motor de plantillas escrito en Python. Se usa junto a Ansible para rellenar las variables de la siguiente manera: Ansible crea un esquema que contendrá todo el diseño del programa, y Jinja2 cargará dicho esquema y realizará la sustitución de las variables en función del contexto de ejecución. Este funcionamiento se puede observar en la Figura 2.2.

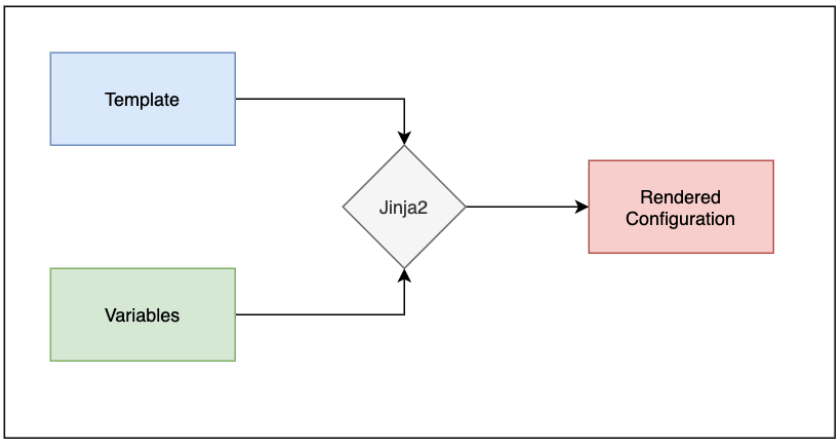


Figura 2.2: Esquema de funcionamiento de Jinja2

2.3. Metodología y ciclo de vida del desarrollo

El ciclo de vida del desarrollo de software o SDLC[48] por sus siglas en inglés *Software Development LifeCycle* es la estructura que contiene los procesos, actividades y tareas relacionadas con el desarrollo y mantenimiento de un producto de software, abarcando la vida completa desde la definición de los requisitos hasta la finalización de de su uso. Las fases en las que se divide un SDLC son:

1. **Fase de planificación:** algunas tareas incluyen actividades como determinación del ámbito del proyecto, realización de estudio de viabilidad, análisis de riesgos asociados, estimación de coste, planificación temporal y asignación de recursos.

2. **Fase de análisis:** se deben reunir todos los detalles que se necesitan para que el producto final cumpla con las expectativas, como son los requisitos, las soluciones existentes y sus alternativas, y las necesidades de los usuarios finales, entre otros.

3. **Fase de diseño:** se busca la mejor solución para los requisitos planteados. Aquí entran la elección de las tecnologías y herramientas de desarrollo, la integración de soluciones ya existentes, etc.

4. **Fase de desarrollo:** consiste en la codificación del producto. Para ello, se dividen los requisitos en tareas de codificación más pequeñas.

5. **Fase de prueba:** se combinan tanto pruebas automáticas como manuales para comprobar el correcto funcionamiento del software. Los análisis de calidad se realizan para comprobar que el software no presenta errores y cumple con los requisitos previamente definidos. Esta fase se suele realizar paralela a la fase de desarrollo.

6. **Despliegue:** una vez terminado el software y comprobado su funcionamiento comienza esta fase, que incluye tareas para llevar el resultado al cliente como son el empaquetado, configuración del entorno e instalación del software.

7. **Mantenimiento:** se supervisa el rendimiento general del producto y se corrigen errores, resuelven problemas de los clientes y administran los nuevos cambios en el software.

Un modelo de ciclo de vida presenta de manera conceptual un SDLC de manera organizada para hacer más sencilla su implementación. Estos modelos presentan las fases del desarrollo en diversos órdenes cronológicos para optimizar el ciclo. Existen diversos modelos, entre los que destacan:

- **Cascada:** dispone de todas las fases de manera secuencial por lo que las nuevas fases dependen del resultado de la fase anterior. Este modelo es muy estricto y proporciona un resultado visible al final de cada fase. Sin embargo, resta flexibilidad al desarrollo ya que hay poco margen a cambio una vez finalizada una fase.

- **Iterativo:** el desarrollo comenzará con un pequeño grupo de requisitos y se mejoran las versiones de manera iterativa a lo largo del tiempo hasta llegar al punto en el que el software final esté preparado para pasar a producción. Esto permite manejar los riesgos y errores de forma sencilla, pero la repetición de los ciclos puede provocar que se cambien los objetivos.

- **Espiral:** combina la repetición de ciclos del modelo iterativo con el flujo secuencial y lineal del modelo de cascada. Se garantizan mejoras graduales mediante la creación de prototipos en cada fase. Es adecuado para proyectos grandes y complejos con cambios frecuentes, pero resulta muy costos para proyectos pequeños con objetivos muy concretos.

- **Ágil:** se organizan las fases en varios ciclos. El desarrollo se itera rápidamente a través de las fases y sólo se hacen pequeños cambios progresivos en cada ciclo. Los requisitos, planes y resultados se evalúan constantemente para responder con eficacia a los cambios. Esto permite identificar y abordar problemas antes de que sean muy graves, además de hacer participes del ciclo a las partes interesadas. Sin embargo, esta participación puede hacer que los objetivos cambien drásticamente.

Para el desarrollo de esta aplicación se utilizó un modelo iterativo. En primer lugar, se estudió la aplicación antigua para determinar los problemas, y se planificó el tiempo y coste estimado. En segundo lugar, se analizaron los requisitos y

objetivos que se debían cumplir, y se diseñó una nueva aplicación que los solucionase. A partir de aquí, se iteraron las fases de desarrollo y prueba para cubrir todas los requisitos que se definieron previamente y comprobar su correcto funcionamiento. En paralelo a este flujo se revisaban y comprobaban todas las partes realizadas, como el análisis de requisitos o el diseño de la aplicación, en busca de algún problema. En caso de encontrarlo, se comenzaba una nueva iteración para solucionarlo.

Capítulo 3

Análisis, diseño y desarrollo

3.1. Esquema global de la aplicación

La arquitectura de software es un concepto que se refiere a una planificación basada en modelos, patrones y abstracciones teóricas que se realiza como paso previo a cualquier implementación. Permite planificar el proyecto y elegir el mejor conjunto de herramientas para llevarlo a cabo, y será muy relevante en el ritmo de desarrollo y en el coste.

Para el desarrollo de este proyecto se ha elegido una arquitectura de 3 capas, que se basa en la separación de la funcionalidad en segmentos que representan una subtarea, cada uno perteneciendo a un nivel de abstracción diferente. Cada capa está diseñada para dar servicio a la siguiente capa, como se observa en el gráfico de la Figura 3.1.

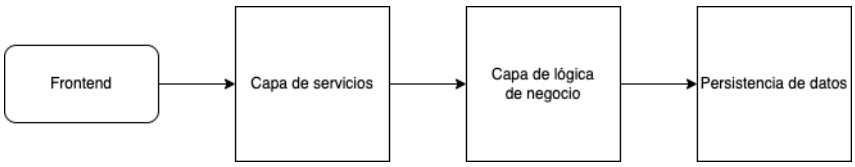


Figura 3.1: Esquema de la arquitectura de 3 capas

Específicamente, esta aplicación está formada por las partes que comprenden Node.js, es decir, el servidor HTTP que recibe las peticiones del cliente (en este caso la aplicación *frontend* que correrá en un navegador).

A continuación, se comunica con la capa de lógica de negocio formada principalmente por Express.js y Ansible. Express.js enruta las peticiones e invoca los procesos de Ansible correspondientes, donde se realizan las gestiones sobre las máquinas.

Por último, la capa de persistencia, constituida por oVirt, se encarga de aplicar las configuraciones de las máquinas virtuales, tanto de creación como de edición y borrado. Esta arquitectura concreta se muestra en la Figura 3.2.

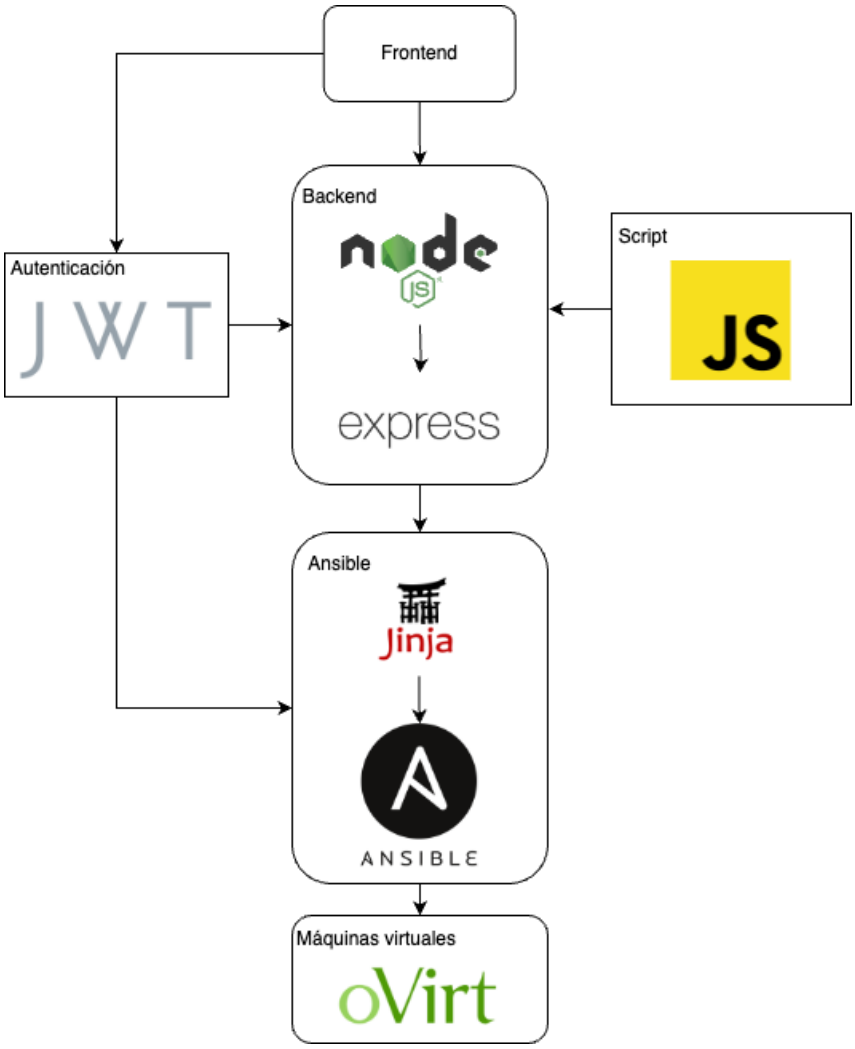


Figura 3.2: Gráfico detallado de la aplicación

3.2. Desarrollo de la aplicación

En esta sección se explica de forma técnica los conceptos tecnológicos involucrados en el desarrollo de la aplicación para entender el funcionamiento concreto de este proyecto expuesto en el Capítulo 4.

3.2.1. Definición de API REST

Una API, de sus siglas en inglés *Application Programming Interface* es un conjunto de reglas y protocolos que permiten interactuar con un sistema o aplicación para obtener datos o ejecutar una función. En otras palabras, una API define cómo comunicarse con la aplicación y utilizar sus funcionalidades, facilitando su integración e interoperabilidad.

Por su parte, REST[47] consiste en un conjunto de principios que definen la API. Estos criterios son:

- Arquitectura cliente-servidor y recursos. Las peticiones se gestionan a través de HTTP y las respuestas se devuelven en formato JSON.

- Comunicación sin estados. No se almacena información entre peticiones y cada una de ellas es independiente.
- Datos que pueden almacenarse en caché para optimizar las interacciones.
- Interfaz uniforme entre los elementos para crear así una comunicación de forma estandarizada.
- Sistema de capas organizado en jerarquías que son totalmente invisibles para el usuario.

Por ello, se pueden extraer las siguientes ventajas de utilizar una API REST:

- **Escalabilidad**: la independencia entre cliente y servidor permite separar responsabilidades, lo que permite un desarrollo aislado, facilita el mantenimiento y la evolución, entre otros. Además, al ser sin estado, cada solicitud contiene toda la información necesaria, por lo que facilita la distribución de cargas y un escalado horizontal.
- **Flexibilidad**: permite realizar una gran variedad de operaciones sobre los recursos, lo que permite a los clientes adaptarse a sus diferentes casos de uso.
- **Evolución gradual**: pueden añadirse nuevas características sin afectar a las aplicaciones clientes existentes.

En este proyecto se ha desarrollado una API REST que ofrece como servicio una forma simplificada de interactuar con oVirt para la administración de máquinas virtuales y recursos.

3.2.2. Administración de máquinas con oVirt

oVirt es una plataforma de virtualización que permite la creación, configuración, monitorización y administración de máquinas virtuales, redes o almacenamiento. Las principales características de esta tecnología son:

- **Administración centralizada**: permite gestionar multiples recursos desde un único punto de control.
- **Agrupación de recursos**: para su gestión eficiente y compartida entre máquinas.
- **Administración de redes**: permite configurar redes virtuales y asignar interfaces de red a las máquinas.
- **Alta disponibilidad**: dispone de funcionalidades de migración en vivo y tolerancia a fallos
- **Portal de autoservicio**: ofrece un portal web para que usuarios finales puedan solicitar y administrar recursos.

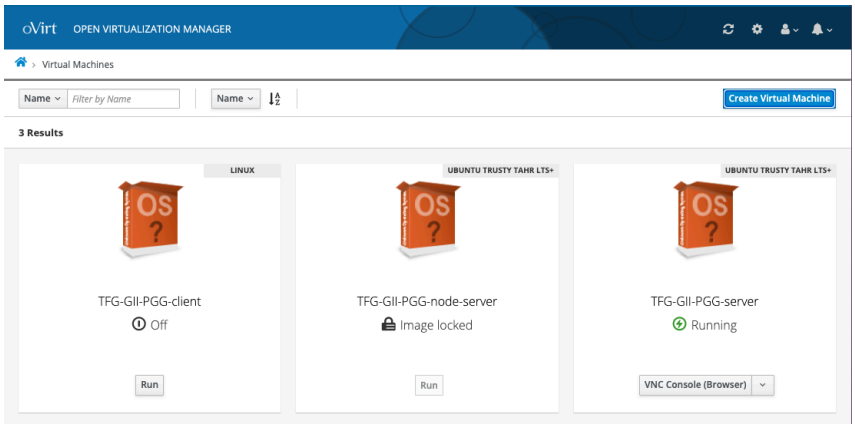


Figura 3.3: Portal de autoservicio del alumno de la ULL

En la Figura 3.3 se puede observar el portal de autoservicio proporcionado por el IaaS de la ULL a los alumnos.

oVirt ofrece una API que permite interactuar con la plataforma y realizar operaciones de administración y configuración a través del protocolo HTTP. Esta API se basa en la estructura comentada anteriormente REST, y utiliza los métodos GET, POST, PUT y DELETE para realizar operaciones sobre los recursos (máquinas virtuales, hosts, almacenamiento...).

Además de las operaciones básicas de CRUD[1], también permite realizar acciones avanzadas, como iniciar o detener máquinas virtuales, migraciones, etc.

En cuanto a seguridad, para la autenticación generalmente se usa un método basado en *tokens*. Para ello, es necesario autenticarse con las credenciales (usuario y contraseña) para obtener el *token*, que luego será incluido en todas las solicitudes para acceder a los recursos protegidos.

Esta API es usada por los módulos de Ansible para gestionar y modificar las máquinas virtuales a través de la aplicación usada por profesores y alumnos.

3.2.3. Ansible

Ansible es la tecnología más importante para el funcionamiento de la aplicación, ya que es la encargada de conectarse con la plataforma oVirt y realizar los cambios que el usuario ejecute. Para ello, apoyándose en el lenguaje de plantillas Jinja2, utiliza las variables que el usuario previamente ha definido para ejecutar una serie de comandos para así persistir las modificaciones en oVirt.

Ansible se conecta a los nodos o máquinas donde va a realizar los cambios y les inserta pequeños programas llamados módulos que permiten realizar tareas en la plataforma. Luego ejecuta estos programas y los retira cuando finaliza la tarea.

En cuanto a la conexión, utiliza el protocolo SSH para conectarse a los nodos y ejecutar las tareas. Una vez conectado, los módulos son transferidos y ejecutados en la máquina remota.

Por último, Ansible utiliza plantillas YAML[56] para automatizar las tareas. YAML

[1]CRUD: del inglés *CREATE, READ, UPDATE, DELETE*. Son las operaciones básicas en las persistencias de datos: crear, leer, actualizar, eliminar.

es un lenguaje de serialización de datos que es comprensible por las personas. Es por ello que es utilizado en el diseño de archivos de configuración.

Colecciones y módulos de Ansible

Un módulo[2] en Ansible es una librería de acciones disponibles en el software, que pueden ser usadas desde una línea de comandos o desde un *playbook*. El usuario puede recurrir a un módulo por tarea o utilizar varios en un mismo *playbook*. Los módulos suelen estar recogidos en colecciones[1].

La mayoría de los módulos en Ansible utilizan los argumentos *clave=valor*, aunque existen algunos que no aceptan argumentos. Además, se encargan de devolver la respuesta en formato JSON[29] (*JavaScript Object Notation*), lo que permite que los módulos puedan ser escritos en cualquier lenguaje de programación.

En este proyecto, la principal colección usada para comunicarse con oVirt fue *ovirt.ovirt*[40]. Esta colección está formada por una seria de módulos que simplifican la interacción de Ansible con la API de oVirt. Da soporte a la automatización de tareas comunes relacionadas con la administración de recursos.

Dentro de la extensa lista de módulos incluidos en *ovirt.ovirt*, los más usados en este desarrollo fueron:

- ***ovirt_auth***[38]: permite autenticarse para acceder a los recursos protegidos. Es usado para recuperar el *token* de autenticación usado en las peticiones y para borrarlo.
- ***ovirt_vm***[43]: utilizado para la administración de máquinas virtuales. Permite crearlas, modificar configuraciones, eliminarlas y cambiarles los estados (iniciar, detener y reiniciar).

Utilizar estos módulos en conjunto con las variables correctas es lo que permite la automatización de las tareas deseadas.

Inventario

El inventario representa el conjunto de máquinas que Ansible va a gestionar automáticamente mediante los playbooks. Además de listar las máquinas, también llamadas *hosts*, este archivo también permite agruparlas y definir variables para cada grupo, lo que simplifica la gestión. Una vez el inventario está definido, se puede iniciar el proceso de gestión de la máquina o grupo de máquinas sobre las que se quiere trabajar.

Playbooks y roles

Un playbook[3] de Ansible es un archivo en formato YAML donde se recogen una seria de tareas de automatización que se desean ejecutar sobre un inventario concreto. Los módulos de Ansible ejecutan tareas, y cuando se combinan dos o más módulos se crea un playbook.

Por su parte, un rol en Ansible es una forma de organizar y reutilizar código. Se trata de una colección de tareas, configuraciones y variables relacionadas que

se agrupan para cumplir un objetivo específico. El propósito de los roles es la reutilización y modularización del código. Además, un rol puede ser compartido o publicado para luego ser utilizado en otros playbooks de distintos proyectos. Por último, permiten la separación de responsabilidades y funciones, lo que facilita el trabajo entre diferentes equipos.

3.3. Servidor DNS

El Sistema de Nombres de Dominio, en inglés DNS[15] o *Domain Name System*, es un conjunto de protocolos y servicios que nació con la finalidad de evitar que los usuarios tuvieran que recordar una secuencia de números poco amigable. Además, la dirección numérica puede cambiar por diferentes razones, pero el nombre se mantiene.

Un servidor DNS es un software que se encarga de traducir los nombres aptos para lectura humana, como puede ser *www.ull.es*, a direcciones IP que son entendidas por máquinas. Por ello, cada dominio debe tener configurado un servidor DNS encargados de traducir el nombre de las máquinas de su dominio en direcciones númericas.

Un servidor DNS puede emplearse para gestionar diferentes zonas. Cada zona es una parte del espacio de nombres DNS que está gestionada de forma independiente. Por su parte, un archivo de zona es un fichero de texto sin formato, almacenado en un servidor DNS, que incluye la representación real de la zona con todos los registros de esta última.

Por último, un registro DNS es una entrada de datos que proporciona información específica. Se utilizan para asociar nombres de recursos con otra información, como direcciones IP. Existen muchos tipos de registros, como puede ser **CNAME**, que establece un alias para un nombre de dominio, **TXT**, que almacena texto arbitrario, o **MX**, que especifica el servidor de correo que debe recibir los correos destinados a un dominio. En esta aplicación, los tipos de registros usados fueron:

- **Registro A (Address)**: asocia un nombre con una dirección IP.
- **Registro NS (Name Server)**: indica los servidores de nombres autorizados para un dominio que se utilizan para responder consultas DNS.
- **Registro SOA (State Of Autority)**: proporciona información fundamental de la zona, como el nombre del servidor primario o el número de serie, que se incrementa cada vez que hay un cambio para dejar constancia.

Capítulo 4

Descripción de la aplicación

En este capítulo se explicará el flujo de uso de la aplicación web que da acceso al IaaS de la ULL, centrándose especialmente en la capa del servidor desarrollada en este proyecto.

4.1. Acceso y uso del IaaS

Para detallar esta sección, se utilizará como *frontend* el proyecto desarrollado por la ex alumna Miriam Núñez García en su Trabajo Fin de Grado titulado *Frontend para la gestión de las Máquinas Virtuales en el IaaS de la ULL*[52], y como *backend* se empleará el trabajo desarrollado para este TFG.

El acceso al portal de gestión de los servicios del IaaS de la ULL requiere estar conectado a la red de la Universidad, ya sea físicamente con un cable, a través de la red WiFi o mediante VPN. Es necesario navegar a la URL *iaas.ull.es*. En esta página se muestra un formulario de inicio de sesión y, a continuación, se redirige a la página principal del *frontend* nombrado en el párrafo anterior, mostrada en la Figura 4.1. En esta vista, se muestran los datos del usuario autenticado y una lista donde se pueden ver las máquinas virtuales asociadas a dicho usuario.

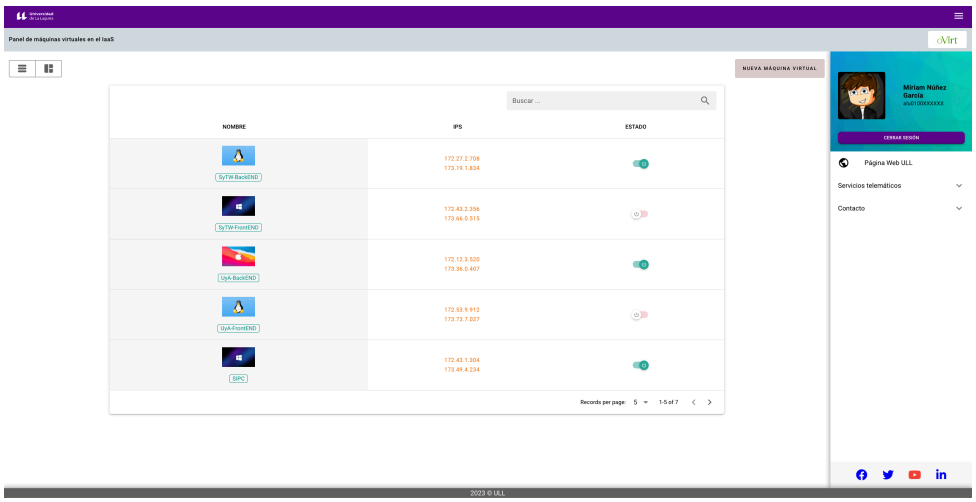


Figura 4.1: Página principal del portal de gestión en el IaaS de la ULL

Como se observa en la Figura 4.1, en la página principal aparece un botón pa-

ra la creación de una nueva máquina virtual que enlaza con un formulario de creación, ilustrado en la Figura 4.2. Se deben completar los campos con los datos específicos (mencionados en la vista de edición de máquina) para la nueva máquina. Cuando se envíe el formulario, el cliente realizará una petición al servidor para realizar los cambios deseados en oVirt.

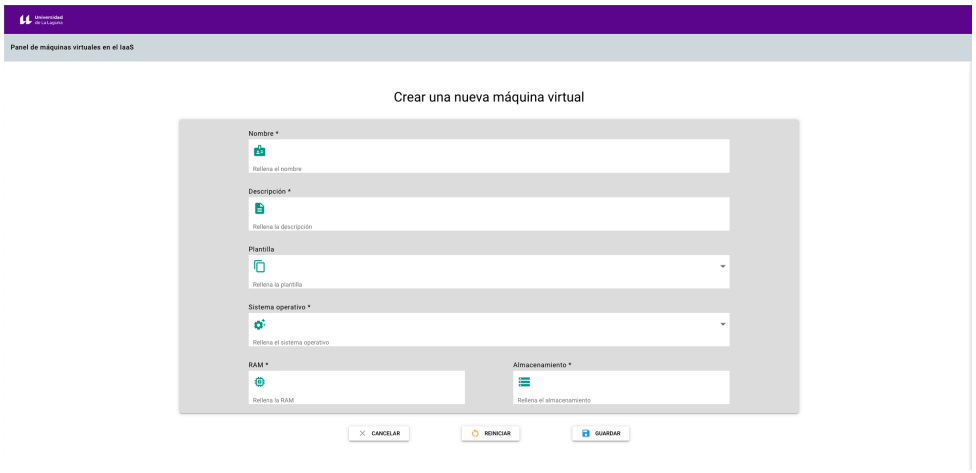


Figura 4.2: Formulario de creación de nueva máquina virtual

Por otra parte, en la tabla principal, junto al nombre de la máquina y el logo del sistema operativo que se ejecuta, se encuentran las columnas de IPs y el estado, que puede ser encendida o apagada, y permite cambiarlo desde esta misma vista. Al hacer clic en un elemento de la tabla, se amplía el detalle de la máquina (Figura 4.3). Es aquí donde aparecen datos generales, como el nombre, la fecha de creación y las IPs asociadas, la plantilla utilizada junto con el sistema operativo, y los recursos asignados, como la RAM y el almacenamiento. Además, se aprecian una serie de botones que se utilizan para realizar acciones sobre la máquina. En la parte superior se encuentran las acciones de encender, abrir la consola en el navegador, suspender, reiniciar y apagar. Por su parte, en la parte inferior se pueden editar los detalles, compartir la máquina con otros usuarios y eliminarla. Cada vez que se ejecute una acción, la aplicación volverá a realizar una llamada al *backend* para efectuar las modificaciones y guardarla en el entorno oVirt.

Al seleccionar la opción de editar, se mostrará un formulario similar al de creación (Figura 4.2), pero con los datos actuales prellenados. Además, al compartir la máquina, se solicitará el nombre de usuario destino.

Este flujo queda ejemplificado en el caso de uso de la Figura 4.4.

4.2. *Endpoints* de la API

Un *endpoint*[16] es una interfaz expuesta por la API para permitir la comunicación con otras aplicaciones. Cada *endpoint* representa un punto de acceso específico en un servidor para realizar una acción concreta, y está formado por la dirección del servidor, el dominio, la ruta y parámetros en caso de ser necesarios.

Para consumir las utilidades de Ansible, el *frontend* tiene que realizar una peti-

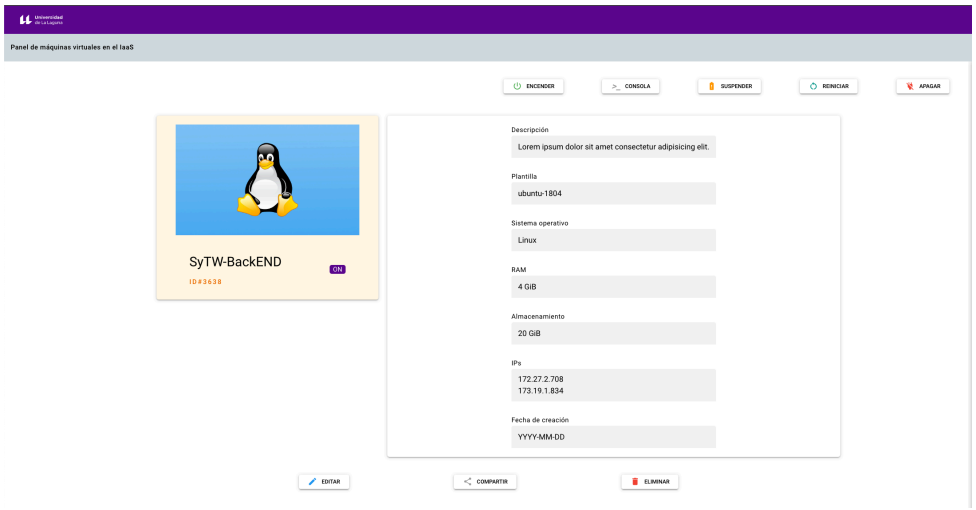


Figura 4.3: Detalles de la máquina virtual

ción HTTP con el método correcto al *endpoint* asociado a la acción deseada. Esto se ilustra en la Figura 4.5, donde se realiza una petición con el método POST al *endpoint* /vm, asociado a la creación de máquinas virtuales. Además, esta petición debe ir acompañada de los parámetros necesarios para realizar la acción de una manera correcta.

4.2.1. */vm*

Descripción

El endpoint */vm* permite crear una nueva máquina virtual asociada al usuario que está conectado al servicio IaaS.

Método HTTP

POST

Parámetros de solicitud

- ***nodes***: lista de nodos que se desean crear. Cada nodo sigue el modelo de datos que se muestra en la Figura 4.6. Estos datos serán recogidos en un formulario en la parte *frontend* y enviados en el cuerpo de la petición de la forma que se muestra. Todos los campos que aparecen son obligatorios.

4.2.2. */vm/state*

Descripción

En */vm/state* se exponen las funcionalidades relacionadas con el estado interno de la máquina.

Método HTTP

PUT

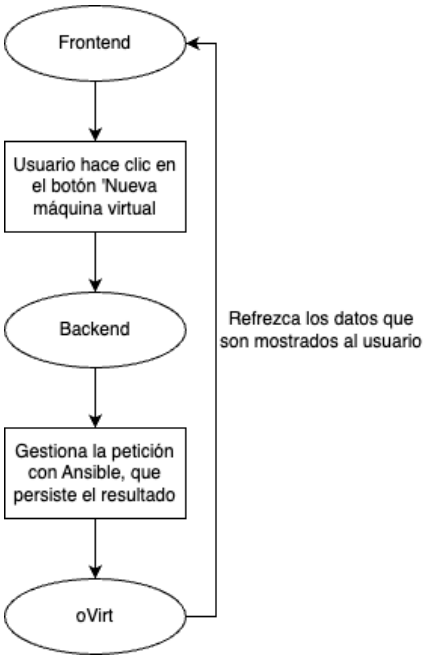


Figura 4.4: Caso de uso de creación de una nueva máquina virtual

Acciones

- Encender
- Apagar
- Reiniciar
- Resetear: esta acción pretende dejar la máquina en el estado inicial. En este caso, el modelo del parámetro *nodes* debe ser el complejo, como se observa en la Figura 4.6.

Parámetros de solicitud

- ***actionType***: define el tipo de acción. Las acciones y sus valores asociados se observan en la Tabla 4.1.
- ***nodes***: lista de nodos que se desean borrar. Además del modelo de datos mostrado en la Figura 4.6, este campo puede aparecer de una manera simplificada como una lista de cadenas de caracteres (*strings*) con los nombres de las máquinas objetivos. Esta forma se muestra en la Figura 4.7. Cabe destacar que este modelo sólo es válido para ciertos endpoints. Para comprobar la definición exacta de la API revisar este documento[6].

4.2.3. */vm/config*

Descripción

Este endpoint se utiliza para cambiar configuraciones de las máquinas.

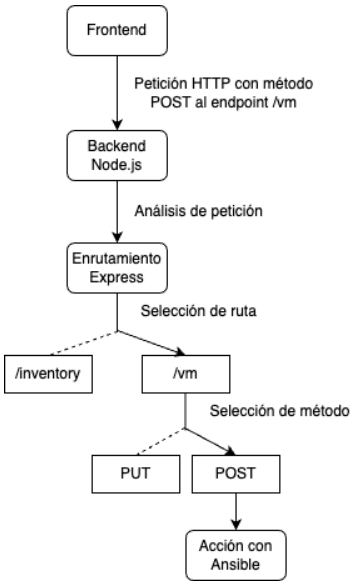


Figura 4.5: Ejemplo de petición a la API

```
{
  "nodes": [
    {
      "name": "frontend",
      "cluster": "Cluster-Rojo",
      "distro": "debian-11-cloudinit",
      "prefix": "LPP",
      "ip": "192.168.0.1"
    }
  ]
}
```

Figura 4.6: Modelo de datos complejo del parámetro **nodes**

Método HTTP

PUT

Acciones

- Resetear contraseña del usuario: la próxima vez que el usuario objetivo definido en el parámetro *username* inicie sesión, se le pedirá que introduzca una nueva contraseña. Esto es útil para situaciones en las que un alumno olvida su contraseña y el profesor pretende recuperarla. Requiere de los parámetros *targetUsername* e *ip* de la máquina.

Acción	actionType
Encender	*start*
Apagar	*stop*
Reiniciar	*restart*
Resetear	*reset*

Tabla 4.1: Tabla de acciones para el endpoint /vm/state

Figura 4.7: Modelo de datos simple del parámetro **nodes** como lista de *strings*

- Añadir usuario a la máquina actual: sirve para darle permisos de acceso a un usuario (*targetUsername* en las máquinas listadas en su forma sencilla4.7 en *nodes*.

Parámetros de solicitud

- ***actionType***: define el tipo de acción. Las acciones y sus valores asociados se observan en la Tabla 4.2.
- ***ip***: se trata de la dirección IP de la máquina objetivo.
- ***targetUsername***: es el nombre de usuario sobre el que se desea realizar la acción.

Acción	actionType
Resetear contraseña	*resetPassword*
Añadir usuario	*addUser*

Tabla 4.2: Tabla de acciones para el endpoint /vm/config

4.2.4. */vm/remove*

Descripción

El endpoint */vm* permite borrar una o varias máquina virtuales que pertenezcan al usuario que ejecuta la acción.

Método HTTP

PUT

Parámetros de solicitud

- ***nodes***: lista de nodos simplificados que se desean borrar siguiendo el modelo 4.7.

4.2.5. */inventory*

Descripción

El endpoint */inventory* se encarga de extraer de la plataforma oVirt la lista de las máquinas virtuales asociadas a un usuario concreto junto con algunos datos generales de interés como las direcciones IPs o el estado de la máquina (encendida, apagada...).

Método HTTP

GET

Resultado

En caso de ejecutarse correctamente, se creará un archivo llamado *hosts-username*, donde *username* será sustituido por el usuario propietario. En este fichero aparecerá el inventario de máquinas del usuario transformado al formato deseado por Ansible.

4.3. Roles de Ansible

En esta sección se van a mostrar y detallar los roles utilizados en los *playbooks* de esta aplicación. Todos los roles tienen la misma estructura de directorios, ilustrada en la Figura 4.8. El fichero donde es definida la funcionalidad concreta se llama *main.yml*, dentro de la carpeta *tasks*, y es el único utilizado en este proyecto. En él aparece una o varias tareas de Ansible escritas en lenguaje YAML[56] y utilizando variables de entorno definidas en el *playbook* donde son utilizadas mediante el lenguaje de platillas Jinja2[28].

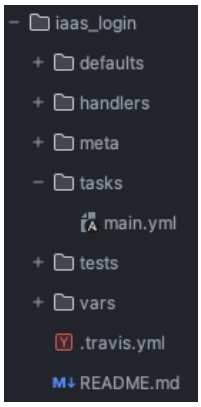


Figura 4.8: Estructura de directorios de un rol

4.3.1. *iaas_login*

El rol ***iaas_login*** (Figura 4.9) se encarga de realizar el inicio de sesión en la plataforma oVirt. Para ello utiliza el módulo *ovirt_auth* de la colección *ovirt.ovirt*.

```
---
- name: Login to IaaS
  ovirt_auth:
    url: https://iaas.ull.es/ovirt-engine/api
    insecure: yes
    username: "{{ ovirt_username }}"
    password: "{{ ovirt_password }}"
    headers:
      filter: true
  debugger: never
```

Figura 4.9: Código del rol *iaas_login*

Los paramétros necesarios para hacer el *login* son:

- ***url*:** URL de la API contra la que se autentica, en este caso la del *IaaS*.
- ***insecure*:** un valor *booleano* (verdadero o falso) que indica si se debe comprobar el certificado TLS[53].
- ***username*:** nombre del usuario
- ***password*:** contraseña del usuario
- ***headers*:** cabeceras HTTP que se desean añadir a la petición. En este caso, aparece la opción *filter* a *true*, lo que indica que se desea eliminar información confidencial o innecesaria en la respuesta.
- ***debugger*:** al tener el valor *never* indica que no se quiere mostrar mensajes de depuración durante la ejecución.

4.3.2. *iaas_logout*

Este rol (Figura 4.10) es usado para limpiar el *token* de autenticación creado al realizar un inicio de sesión en *oVirt* mediante el rol *iaas_login*4.3.1. Se utiliza el mismo módulo *ovirt_auth* que en el *login* definiendo los parámetros *ovirt_auth* con el valor del *token* y *state* con *absent* para indicar la eliminación del primero.

```
---
- name: Cleanup Iaas auth token
  ovirt_auth:
    ovirt_auth: "{{ ovirt_auth }}"
    state: absent
```

Figura 4.10: Código del rol *iaas_logout*

4.3.3. *vm_create*

vm_create (Figura 4.11) se encarga de crear una máquina virtual en la plataforma *oVirt* mediante el uso del módulo *ovirt_vm*, el cual también pertenece a la colección *ovirt.ovirt*. Los parámetros son:

- ***auth:*** *token* de autenticación contra *oVirt*.
- ***cluster:*** nombre del clúster[1] al que se debe asignar la máquina virtual.
- ***name:*** nombre de la máquina.
- ***cpu_cores:*** número de núcleos de CPU asignados.
- ***cpu_sockets:*** número de sockets de CPU asignados.

[1]Clúster: conjunto de recursos infromáticos que incluye hosts físicos, almacenamiento y redes virtuales.

- ***memory:*** capacidad de almacenamiento.
- ***template:*** plantilla base utilizada para crear la máquina. Estas plantillas están preconfiguradas y permiten crear varias máquinas virtuales con las mismas características.
- ***nics:*** define la configuración de las interfaces red.
- ***state:*** estado de la máquina. Usarlo con valor *present* sirve para que *oVirt* comprueba si la máquina existe y sólo crearla en caso negativo.
- ***wait:*** indica si se desea que Ansible espere a que se termine esta tarea para seguir con su flujo.

Por último, la funcionalidad *with_items* en Ansible se utiliza para iterar sobre una lista de elementos. En cada iteración, el elemento seleccionado se guarda en la variable *item*, y sus propiedades se pueden acceder mediante *item.propiedad*. En el contexto específico de este caso, Ansible recorre la lista *nodes* que contiene el conjunto de máquinas virtuales que se desean crear.

```yaml
---
- name: Create a VM
  ovirt_vm:
    auth: "{{ ovirt_auth }}"
    cluster: "{{ item.cluster }}"
    name: "{{ item.fullName }}"
    cpu_cores: 1
    cpu_sockets: 1
    memory: 1GiB
    template: "{{ item.distro }}"
    nics: "{{ node_nics }}"
    state: present
    wait: yes
  with_items: "{{ nodes }}"
```

Figura 4.11: Código del rol *vm_create*

4.3.4. *vm_remove*

Este rol (Figura 4.12) tiene la responsabilidad de iterar sobre una lista de elementos llamada *nodes* y eliminar cada uno de ellos. Para lograr esto, utiliza el módulo *ovirt_vm*, pasando diferentes parámetros. El campo *auth* se utiliza para proporcionar el token de autenticación, el campo *name* se utiliza para especificar el nombre de la máquina y el campo *state* se establece en *absent*, lo que indica a *oVirt* que si la máquina existe, debe ser eliminada.

```yaml
---
- name: Stop and Remove a VM
  ovirt_vm:
    auth: "{{ ovirt_auth }}"
    name: "{{ item.fullName }}"
    state: absent
  with_items: "{{ nodes }}"
```

Figura 4.12: Código del rol *vm_remove*

4.3.5. *vm_start*, *vm_stop* y *vm_restart*

Estas tres tareas funcionan de manera similar al rol *vm_remove*,explicado en la Sección 4.3.4, con la única diferencia en el parámetro *state*. Cada uno de estos roles tiene un estado específico: ***running*** para encenderla, ***stopped*** para apagarla y **reboot** para reiniciarla.

4.3.6. *vm_init*

vm_init (Figura 4.13) se utiliza para personalizar la configuración de una máquina virtual en *oVirt* utilizando Cloud-Init[11]. Cloud-Init es una herramienta ampliamente utilizada en entornos de nube y virtualización que permite inyectar fácilmente datos, como configuraciones de red, usuarios, contraseñas y scripts de inicialización durante el proceso de inicio de una máquina virtual.

```yaml
---
- name: Updating VM
  ovirt_vm:
    auth: "{{ ovirt_auth }}"
    name: "{{ item.fullName }}"
    cluster: "{{ item.cluster }}"
    state: running
    cloud_init_nics:
      - nic_name: ens3
        nic_boot_protocol: dhcp
        nic_on_boot: True
    cloud_init:
      host_name: "{{ item.fullName }}"
      user_name: ansible
      root_password: "{{ ansible_password }}"

      authorized_ssh_keys: "{{ ssh_keys }}"
      custom_script: # too long for a screenshot
    wait: yes
  with_items: "{{ nodes }}"
```

Figura 4.13: Código del rol *vm_init*

Además de las opciones mencionadas anteriormente para parametrizar el módulo *ovirt_vm*, se han agregado dos campos adicionales:

- ***cloud_init_nics:*** configura una interfaz de red llamada *ens3* con el protocolo DHCP[14] para así asignarle una IP dinámica a la máquina.
- ***cloud_init:*** añade un usuario llamado *ansible* y una clave SSH que permita al servidor desde el que se crea acceder a la máquina en un futuro para realizar

acciones, como restablecer la contraseña de un usuario. Además, se añaden una serie de *scripts* para añadir al usuario *ansible* al fichero *sudoers* y darle así privilegios de ejecución como administrador.

4.4. *Playbooks* de Ansible

Los *playbooks* desarrollados para este proyecto siguen la misma estructura. Esta se ve reflejada en la Figura 4.14, donde se aprecia el código del *playbook* con nombre *start_vm*.

```yaml
---
- name: Start VMs.
  hosts: localhost
  gather_facts: false

  # Variables
  vars_files:
    - ../files/pabloTFG.yaml
    - ../files/credentials.yaml

  # Pre-tasks
  pre_tasks:
    - name: Login to IaaS
      include_role:
        name: iaas_login

  # Tasks
  tasks:
    - name: Start VM
      include_role:
        name: vm_start

  # Post-tasks
  post_tasks:
    - name: Logout from IaaS
      include_role:
        name: iaas_logout
```

Figura 4.14: Código del *playbook* para encender una máquina

En la primera parte del archivo se especifica el **nombre** del *playbook* y los ***hosts***, también llamado inventario, en los que se ejecutarán las tareas y configuraciones definidas. En general, para esta aplicación, el inventario se establecerá como *localhost* ya que las tareas serán realizadas sobre las máquinas remotas por el servidor utilizando los módulos de Ansible que gestionan oVirt, aunque existe un caso concreto en el playbook *reset_password* donde Ansible tiene que acceder a la máquina virtual para ejecutar un comando. En este último caso, el inventario será la dirección IP o grupo al que pertenezca esa máquina.

Siguiendo con la estructura del archivo aparece la sección de variables ***vars_files***, donde se especifican los archivos externos donde aparecen variables y se definen las constantes globales para ese *playbook* en particular. Para que Ansible comprenda los archivos, éstos deben estar en formato YAML o JSON, con su correspondiente sintaxis y extensión de archivo. Por su parte, las constantes definidas dentro del *playbook* deben seguir la sintaxis del mismo, es decir, YAML. Este comportamiento se puede observar en la sección ***vars*** del *playbook* ***create_vm*** ilustrada en la Figura 4.15.

```yaml
---
- name: Create VMs.
  hosts: localhost
  gather_facts: false

  # Variables
  vars_files:
    - ../files/pabloTFG.yaml
    - ../files/credentials.yaml

  vars:
    node_nics:
      - name: nic1
        profile_name: DOC1
```

Figura 4.15: Sección *vars* del *playbook create_vm*

A continuación, aparece el apartado ***pre_tasks***, que se refieren a las tareas que se ejecutan antes del *playbook* inicial, generalmente para efectuar configuraciones previas o preparar el entorno. En este caso se realiza el inicio de sesión en oVirt, que devuelve el *token* de autenticación.

Las ***tasks*** en Ansible son las acciones específicas que se realizarán sobre los *hosts* del inventario. En la Figura 4.14 se observa como se invoca al rol con nombre *vm_start*. Con esto se consigue que se ejecuten todos los módulos que estén definidos en dicho rol, completando así el objetivo del playbook.

Por último, se ejecutan las tareas definidas en la sección de ***post_tasks***, relacionadas con la limpieza o generación de informes, entre otras. El rol *iaas_logout* ejecutado en esta parte se encarga de borrar el *token* de autenticación de oVirt generado anteriormente.

En la Tabla 4.3 se observa un resumen de los *playbooks* disponibles, con su inventario y los roles que utiliza.

4.5. Funcionalidad de Mis Máquinas

Como se comenta en la Sección 1.1, la aplicación de Mis Máquinas es un caso particular y distinto al comentado anteriormente. Existe un portal que ofrece un listado resumido de las máquinas virtuales asociadas a un usuario, junto a infor-

Nombre del *playbook*	Inventario	Roles
create_vm	IP del servidor (localhost)	iaas_login vm_create vm_init iaas_logout
remove_vm	IP del servidor (localhost)	iaas_login vm_remove iaas_logout
reset_password	IP del cliente	iaas_login vm_start reset_password vm_stop iaas_logout
reset_vm	IP del servidor (localhost)	iaas_login vm_remove vm_create vm_init iaas_logout
restart_vm	IP del servidor (localhost)	iaas_login vm_restart iaas_logout
start_vm	IP del servidor (localhost)	iaas_login vm_start iaas_logout
stop_vm	IP del servidor (localhost)	iaas_login vm_stop iaas_logout

Tabla 4.3: Tabla de *playbooks*

mación como el sistema operativo, el estado de la máquina y las interfaces de red con las direcciones IP. Esta vista es mostrada en la Figura 4.16.

Las direcciones IP de las máquinas virtuales son asignadas a través de un servidor DHCP que implementa la ULL para esta finalidad. Al no tener acceso a este servidor, es necesario encontrar una manera alternativa para construir el listado de máquinas. Por ello, en este proyecto se utiliza un servidor DNS que servirá de almacenamiento de registros donde se asocian el nombre de una máquina con su dirección IP mediante registros del tipo A. El flujo para recuperar los datos es detallado en el apartado 4.5.

BIND9 es el software elegido para la gestión de todo lo relacionado con el servidor DNS. BIND9 utiliza una herramienta llamada *named* para implementar y ejecutar el servidor. *named* se ejecuta en segundo plano y se encarga de recibir consultas de resolución de nombre DNS y devolver respuestas adecuadas, además

Nombre de la máquina	Sistema operativo	Estado	Direcciones IP
TFG-GII-PGG-test-memoria-2	Otro sistema operativo	Encendida	**nic1**: *10.6.129.244, fe80::21a:4aff:fe4d:dd68*
TFG-GII-PGG-test-memoria	Otro sistema operativo	Encendida	**nic1**: *10.6.130.9, fe80::21a:4aff:fe4d:dd3b*
TFG-GII-PGG-node-server	Ubuntu	Encendida	**nic1**: *10.6.130.22, fe80::21a:4aff:fe97:5197*
TFG-GII-PGG-server	Ubuntu	Encendida	**nic1**: *10.6.131.86, fe80::21a:4aff:fe97:7f20*
TFG-GII-PGG-client	Linux	Encendida	**nic1**: *10.6.129.88, fe80::21a:4aff:fe97:7f3c*

Figura 4.16: Portal Mis Máquinas

de administrar las zonas DNS y realizar las transferencias de zonas[2], entre otras.

Esta utilidad lee la configuración definida en el fichero *named.conf* (generalmente en la ruta */etc/bind*), donde se especifican las zonas, registros y otras opciones relacionadas con el funcionamiento del servidor DNS. La configuración utilizada para este proyecto se aprecia en la Figura 4.17.

```
usuario@ubuntu:~$ cat /etc/bind/named.conf
// This is the primary configuration file for the BIND DNS server named.
//
// Please read /usr/share/doc/bind9/README.Debian.gz for information on the
// structure of BIND configuration files in Debian, *BEFORE* you customize
// this configuration file.
//
// If you are just adding zones, please do that in /etc/bind/named.conf.local

include "/etc/bind/named.conf.options";
include "/etc/bind/named.conf.local";
include "/etc/bind/named.conf.default-zones";
include "/etc/bind/rndc.key";

controls {
        inet 127.0.0.1 allow { localhost; } keys { "rndc-key"; };
};

logging {
        channel default_file {
                file "/var/log/named.log" size 10m;
                severity info;
                print-time yes;
                print-severity yes;
                print-category yes;
        };
        category default{ default_file; };
};
```

Figura 4.17: Fichero de configuración *named.conf*

Dentro de este fichero se incluyen diversos archivos de configuración. Entre ellos, el archivo *named.conf.local*, mostrado en la Figura 4.18, donde se definen las zonas y sus características.

En cada definición se detalla cuál es el fichero de zona. Por ejemplo, todos los registros de la zona *alu0100887037.ull.lan*, creada específicamente para guardar los recursos de ese usuario, aparecen en el fichero con el mismo nombre, representado en la Figura 4.19.

En los registros de recursos, delimitados por un comentario, se pueden observar el nombre de la máquina virtual, el tipo de registro y la dirección IP.

[2]Transferencia de zona: proceso mediante el cual se replica o se transfiere la información de una zona DNS

```
usuario@ubuntu:~$ cat /etc/bind/named.conf.local
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "midominio.local" {
        type master;
        file "/var/cache/bind/db.midominio.local";
        allow-update { 0/0; };
};

zone "ull.lan" {
        type master;
        file "/var/cache/bind/db.ull.lan";
        allow-update { 0/0; };
};

zone "alu0100887037.ull.lan" {
        type master;
        file "/etc/bind/zones/alu0100887037.ull.lan";
        allow-update { 0/0; };
};
```

Figura 4.18: Fichero de configuración *named.conf.local*

Funcionalidad de la aplicación

Una vez configurado el servidor DNS, para almacenar el inventario se crea una zona para cada usuario donde cada registro representa una máquina virtual, con su nombre y su dirección IP. Esta zona será actualizada automáticamente a través de la herramienta *ddclient*[12], que se utiliza para añadir o modificar una entrada en un servidor DNS remoto.

Al crear una máquina, con el rol *vm_init*4.3.6 se le añade a través de Cloud-Init un fichero llamado *ddclient.conf*, ilustrado en la Figura 4.20. Este fichero sirve para configurar la herramienta *ddclient*. En él se define el protocolo que se va a usar (en este caso *nsupdate*), la dirección IP y puerto del servidor, la contraseña necesaria para acutalizar la zona en el servidor DNS, la interfaz de red sobre la que se va

```
usuario@ubuntu:~$ cat /etc/bind/zones/alu0100887037.ull.lan
$TTL 86400
@       IN      SOA     ns1.alu0100887037.ull.lan. admin.alu0100887037.ull.lan. (
                        2023070301 ; número de serie
                        3h ; tiempo de actualización
                        1h ; tiempo de reintento
                        1w ; tiempo de expiración
                        1d ; tiempo mínimo en caché
                        )
        IN      NS      ns1.alu0100887037.ull.lan.

; Registros de recursos (RR) para la subzona
ns1     IN      A       10.6.131.86
TFG-GII-PGG-node-server IN      A       10.6.130.22
TFG-GII-PGG-test-memoria        IN      A       10.6.130.9
TFG-GII-PGG-client      IN      A       10.6.129.88
```

Figura 4.19: Fichero de zona para *alu0100887037.ull.lan*

a actuar, la zona a la que se va añadir y el nombre de la máquina. Esto creará o modificará una entrada en la zona destino cada vez que la interfaz de red de la máquina cambie.

```
- path: /etc/ddclient.conf
  permissions: '0644'
  content: |
    # Configuration file for ddclient generated by debconf
    # /etc/ddclient.conf
    protocol=nsupdate
    server={{ server }} {{ port }}
    password=/etc/bind/rndc.key
    use=if
    if=ens3
    zone={{ username }}.ull.lan
    {{ item.fullName }}
```

Figura 4.20: Fichero *ddclient.conf*

Finalmente, para conseguir el inventario de un usuario se realiza una transferencia de zona desde el servidor DNS hacia la máquina donde se encuentra la aplicación desarrollada en este proyecto. El resultado de la transferencia se vuelca en un fichero temporal (Figura 4.21). Este fichero es transformado al formato que espera Ansible y guardado en un archivo llamado *hosts-username*, dando como resultado el inventario final que será usado por los playbooks que lo necesiten para conectarse a las máquinas de un determinado usuario.

```
usuario@ubuntu:~/app$ cat files/inventory.txt

; <<>> DiG 9.16.1-Ubuntu <<>> @10.6.131.86 -t axfr alu0100887037.ull.lan
; (1 server found)
;; global options: +cmd
alu0100887037.ull.lan.  86400   IN      SOA     ns1.alu0100887037.ull.lan.
alu0100887037.ull.lan.  86400   IN      NS      ns1.alu0100887037.ull.lan.
ns1.alu0100887037.ull.lan. 86400 IN     A       10.6.131.86
TFG-GII-PGG-client.alu0100887037.ull.lan. 86400 IN A 10.6.129.88
TFG-GII-PGG-node-server.alu0100887037.ull.lan. 86400 IN A 10.6.130.22
TFG-GII-PGG-test-memoria.alu0100887037.ull.lan. 86400 IN A 10.6.130.9
alu0100887037.ull.lan.  86400   IN      SOA     ns1.alu0100887037.ull.lan.
;; Query time: 0 msec
;; SERVER: 10.6.131.86#53(10.6.131.86)
;; WHEN: Sun Jul 09 21:48:07 UTC 2023
;; XFR size: 7 records (messages 1, bytes 306)
```

Figura 4.21: Fichero con el contenido de la transferencia de zona

Capítulo 5

Resultados y problemas

5.1. Resultados

A continuación se ejemplifican casos de uso del funcionamiento de la aplicación. Las pruebas se realizan con la aplicación expuesta en el puerto 8080 de una máquina virtual del IaaS con dirección IP 10.6.130.22 y las peticiones son simuladas mediante Postman. Además, las credenciales necesarias para hacer *login* están guardadas como variables de entorno.

5.1.1. Crear una máquina virtual

Para crear una máquina virtual, se hace una petición HTTP con el método POST al endpoint */vm*, como se muestra en la Figura 5.1.

Figura 5.1: Petición en Postman para crear una máquina virtual

El *body request* de la petición (Figura 5.2) debe estar compuesto por una lista de nodos a crear con los campos: nombres de la máquina (*name*), prefijo del nombre (*prefix*), el *cluster* de recursos (*cluster*) y la plantilla que servirá para definir la configuración inicial, como sistema operativo o aplicaciones preinstaladas (*distro*).

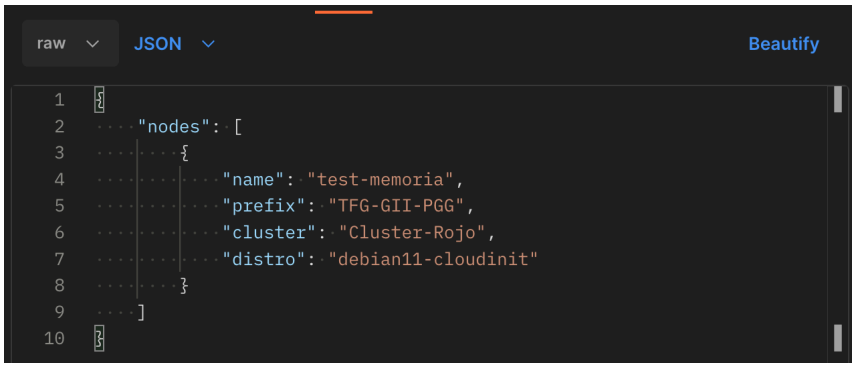


Figura 5.2: *Body request* de la petición de crear máquina virtual

En caso exitoso, el resultado de este playbook es la creación de una máquina virtual que puede ser observada en el portal de usuario que se conecta con el IaaS como en la Figura 5.3.

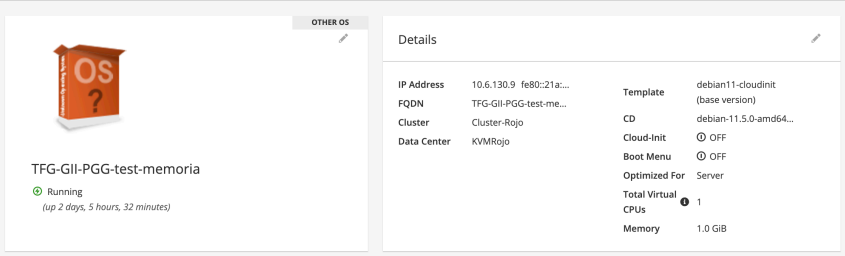


Figura 5.3: Portal de usuario de acceso al IaaS

5.1.2. Encender varias máquinas virtuales

Para encender varias máquinas virtuales, se realiza una petición HTTP con el método PUT al endpoint */vm/state*. Esta petición debe estar acompañada con un *body request* que contenga un campo *actionType* con la acción que se quiere realizar (*start* en este caso), una lista o *array* de los nombres de las máquinas que se pretende encender y el *cluster* al que se desea agregar. Esto queda representado en la Figura 5.4.

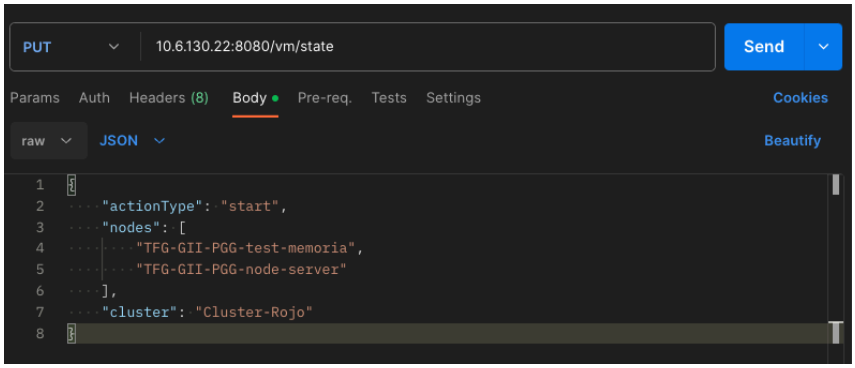


Figura 5.4: Petición en Postman para encender varias máquinas virtuales

Esto se traduce en el servidor como la ejecución del playbook *start_vm*. En él, primero se hace un inicio de sesión contra oVirt para recuperar el *token* de autorización. Luego, se ejecutan las acciones de encendido de las máquinas objetivo y, por último, se borra el *token*. Durante este flujo, Ansible informa del estado de las acciones mediante la salida en consola mostrada en la Figura 5.5.

5.1.3. Recuperar el inventario de un usuario

Para recuperar el inventario de un usuario se realiza una petición HTTP con el método GET al endpoint */inventory/username*, donde *username* es el nombre del usuario destino. En la Figura 5.6 se ve la petición para obtener la lista de máquinas asociadas al usuario alu0100887037.

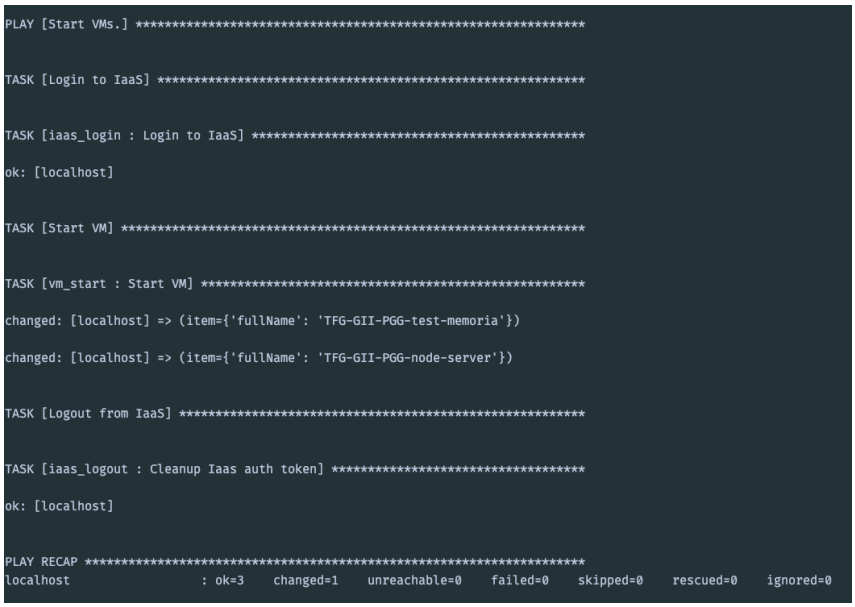


Figura 5.5: Salida por console de la ejecución del playbook *start_vm*

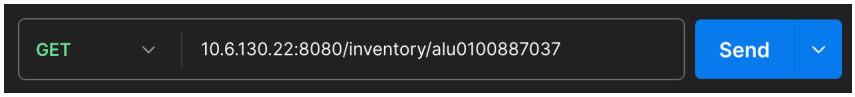


Figura 5.6: Petición para obtener el inventario de un usuario

Tras esta solicitud, el servidor ejecuta el flujo descrito en la Sección 4.5. Como resultado, se crea un fichero de inventario con el nombre *hosts-username* como el que se muestra en la Figura 5.7.

```
usuario@ubuntu:~/app$ cat ansible/inventory/hosts-alu0100887037
[ns1.alu0100887037.ull.lan]
10.6.131.86

[TFG-GII-PGG-client.alu0100887037.ull.lan]
10.6.129.88

[TFG-GII-PGG-node-server.alu0100887037.ull.lan]
10.6.130.22

[TFG-GII-PGG-test-memoria.alu0100887037.ull.lan]
10.6.130.9
```

Figura 5.7: Inventario del usuario *alu0100887037*

5.2. Problemas

5.2.1. Versiones de las tecnologías

El primer problema encontrado ha sido las versiones de las tecnologías. El sistema operativo que se utilizó para desarrollar el proyecto es macOS, propio de la

compañía Apple. A pesar de estar basado en Unix y, por lo tanto, ser similar y compatible con Linux, existen ciertas inconsistencias a la hora de instalar y actualizar paquetes. Es por ello que surgieron problemas a la hora de ejecutar playbooks de Ansible que recurrían a módulos concretos, donde las versiones de las librerías estaban instaladas correctamente, pero Ansible era incapaz de encontrarlas. La solución fue complicada de encontrar, pero el problema residía en ciertas variables de entorno globales del sistema. Una vez se modificaron, todo funcionó correctamente.

5.2.2. Falta de documentación

Ansible es una tecnología desarrollada por Red Hat, una enorme empresa. Por ello, posee una amplía comunidad que comparte problemas y soluciones en foros en internet. Sin embargo, concretamente la plataforma oVirt y su módulo de Ansible son más pequeñas, y la información que se encuentra publicada es muy poca e incluso nula. Por esta razón, cuando ha surgido alguna duda o problema respecto a la gestión de máquinas se ha tenido que recurrir a la documentación oficial de oVirt que, pese a ser muy extensa y bien detallada, hay casos concretos que no se consideran en ella.

5.2.3. Servidor DNS

El principal problema encontrado durante el desarrollo fue cómo conseguir la funcionalidad de Mis Máquinas. Al no tener acceso a la aplicación que está siendo utilizada, no se pudo saber de qué forma se consiguen los datos actualmente.

Primero, se intentó mediante módulos de Ansible como *ovirt-nics-info*[39], pero el mejor resultado conseguido fue la lista de nombres de las máquinas virtuales asociadas a un usuario, faltando las direcciones IPs de éstas. Para recoger información interna de la máquina a través de Ansible, primero debe tener la dirección IP definida en el inventario, por lo que este método quedó descartado.

Luego, se intentó mediante agentes como QEMU Guest Agent[46] recoger los datos de la máquina, pero para poder ejecutar el agente en la máquina remota se necesita realizar un primer acceso para abrir el socket de comunicación que usa QEMU para enviar la información, haciendo inviable la automatización de este paso.

Tras estos fallos y como opción final se decidió usar un servidor DNS donde cada vez que se inicialice una interfaz de red dentro de una máquina virtual, ésta última se encarga de enviar una consulta al servidor para escribir una nueva entrada.

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

El servicio IaaS de la Universidad de La Laguna ofrece a profesores y alumnos una potente herramienta, tanto para la investigación como para la docencia. En el ámbito docente, este modelo en la nube ha permitido a los profesores preparar entornos de prácticas homogéneos donde todos los estudiantes pueden experimentar con tecnologías y herramientas específicas de una manera sencilla y sin limitaciones. Los profesores pueden configurar máquinas virtuales con software preinstalados mediante plantillas que facilita la instalación y uso del entorno a los estudiantes. Por su parte, los alumnos pueden acceder a los recursos ofrecidos a través de Internet y con alta disponibilidad, por lo que resulta más accesible la práctica y el desarrollo de las asignaturas.

Este proyecto se ha centrando en la creación de una aplicación *backend* que comunique de forma eficiente la parte gráfica del portal de servicio utilizado por los usuarios con los servicios de virtualización ofrecida por el IaaS de la ULL a través de oVirt. Para ello, mediante la herramienta Ansible se consiguió conectar las peticiones realizadas por el usuario con el las funcionalidades aportadas por el entorno oVirt de una manera automática y desatendida.

Este proyecto brinda a los usuarios la posibilidad de crear, resetear y borrar máquinas bajo demanda, además de realizar cambios de estado como encendido, apagado y reinicio. Además, permite hacer algunos ajustes de configuración de las máquinas ya creadas de manera remota, como la recuperación de la contraseña. Por último, ofrece a los usuarios una lista resumida de todas las máquinas virtuales que posee, junto con su información mas relevante.

Por su parte, la aplicación ha sido diseñada para ser mantenible y escalable, dado que tiene una estructura bien definida y organizada que permite la incorporación de nuevas funcionalidades de manera sencilla sin necesidad de modificar nada de lo realizado anteriormente.

6.2. Líneas futuras

6.2.1. Añadir seguridad a las llamadas mediante JWT

JWT[30], de sus siglas en ingles *JSON Web Token*, es un estándar abierto basado en JSON para la creación de *tokens* de acceso que permiten la propagación de identidad y privilegios. Se utiliza para transmitir información de forma segura.

Un ejemplo de uso en esta aplicación puede ser la autorización por roles. Cuando un usuario se conecte al portal de acceso al servicio IaaS se debería crear un JWT con los permisos del rol al que pertenezca, como puede ser administrador, profesor o alumno. Este *token*, que está encriptado en *Base64*[9], es enviado al *backend* junto a cada petición y traducido en el servidor, donde se comprueba que el usuario tiene permisos para realizar la petición.

6.2.2. Ampliar funcionalidades

En este proyecto se desarrollaron las funcionalidades comentadas en la Subsección 4.4. Para ampliar el catálogo de acciones se podría desarrollar un playbook para añadir un usuario a una máquina, donde el propietario de la máquina le daría permisos de administrador al usuario destino. Esta característica se puede implementar mediante el módulo de Ansible *ovirt_permission*[41].

Por otro lado, también sería útil añadir la posibilidad de clonar máquinas, donde se crease una máquina nueva con el estado actual de la original. Para ello haría falta guardar el estado actual en una plantilla de creación de máquinas y crear una nueva a partir de dicha plantilla. Para almacenar el estado de la máquina se puede recurrir al módulo *ovirt_snapshot*[42].

6.2.3. Mejorar el control de errores, respuestas de la API y *testing*

Para mejorar el control de errores de la aplicación se podría utilizar la salida con mensaje de error del comando de Ansible. Ésta debería ser analizada por una función para desechar las partes que no aportan valor y crear un mensaje final que se comprensible por el usuario. Éste puede ser enviado en la respuesta de la API, haciendo así que los mensajes de error sean más flexibles para dar más información sobre la ejecución de la acción.

Por otra parte, existe la posibilidad de implementar *testing* unitario a nivel de Ansible. Cada rol creado puede ser testado de manera aislada para asegurarse que, aunque se realicen cambios, el funcionamiento es el esperado con cada iteración.

Capítulo 7

Summary and Conclusions

7.1. Conclusions

The IaaS service of the University of La Laguna offers teachers and students a powerful tool for both research and teaching. In the educational field, this cloud model has enabled professors to create homogeneous practice environments where all students can easily and without limitations experiment with specific technologies and tools. Teachers can configure virtual machines with pre-installed software using templates, which simplifies the installation and usage of the environment for students. On the other hand, students can access the resources offered via the Internet with high availability, making practical exercises and subject development more accessible.

The main objective of this project is the creation of a backend application that efficiently communicates the graphical part of the service portal with the virtualization services offered by ULL's IaaS through oVirt. To achieve this, Ansible tool was used to connect user requests with the functionalities provided by the oVirt environment in an automated and unattended manner.

This project provides users with the ability to create, reset, and delete machines on demand, as well as perform state changes such as powering on, powering off, and restarting. Additionally, it allows for remote configuration adjustments of already created machines, such as password recovery. Lastly, it offers users a summarized list of all the virtual machines they own, along with their most relevant information.

On the other hand, the application has been designed to be maintainable and scalable, as it has a well-defined and organized structure that allows for easy incorporation of new functionalities without the need to modify anything previously implemented.

7.2. Future work

7.2.1. Adding security to the calls using JWT

JWT (JSON Web Token), is an open standard based on JSON for creating access tokens that allow the propagation of identity and privileges. It is used to securely

transmit information.

An example of usage in this application can be role-based authorization. When a user logs into the portal to access the IaaS service, a JWT should be created with the permissions of their role, such as administrator, professor, or student. This token, which is encrypted in *base64*[9], is sent to the *backend* along with each request and decoded on the server, where it is checked that the user has permissions to perform the request.

7.2.2. Expanding functionalities

In this project, the functionalities mentioned in Subsection 4.4 were developed. To expand the catalog of actions, a playbook could be developed to add a user to a machine, where the machine owner would grant administrative permissions to the target user. This feature can be implemented using the Ansible module *ovirt_permission*[41].

On the other hand, it would also be useful to add the possibility of cloning machines, where a new machine would be created with the current state of the original one. To do this, it would be necessary to save the current state in a machine creation template and create a new machine based on that template. To store the machine's state, the *ovirt_snapshot* module[42] can be used.

7.2.3. Improve error handling, API responses, and *testing*

To improve error control in the application, the error output of the Ansible command could be used. This output should be analyzed by a function to discard parts that do not provide value and create a final message that is understandable to the user. This message can be sent in the API response, making error messages more flexible to provide more information about the execution of the action.

In addition, there is the possibility of implementing unit testing at the Ansible level. Each created role can be tested in isolation to ensure that, even with changes, the functionality is as expected with each iteration.

Capítulo 8

Presupuesto

Siguiendo el Régimen General de la Seguridad Social[24], la base de cotización de contingencias comunes de la categoría profesional *Ingenieros y Licenciados. Personal de alta dirección no incluido en el artículo 1.3.c) del Estatuto de los Trabajadores* es 1759,50 euros/mes. Trabajando un total de 160 horas al mes da un coste de 11 euros las hora.

Este proyecto tuvo una duración de 8 semanas, que suman un total de 320 horas trabajadas. El presupuesto para está aplicación es de **3520 euros**.

Bibliografía

[1] Lista de colecciones de ansible. https://docs.ansible.com/ansible/latest/collections/index.html#list-of-collections.

[2] Lista de módulos de ansible. https://docs.ansible.com/ansible/2.9/modules/list_of_all_modules.html.

[3] Qué es un playbook de ansible. https://www.redhat.com/es/topics/automation/what-is-an-ansible-playbook.

[4] Apache. https://httpd.apache.org/.

[5] Qué es una api. https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces.

[6] Definición de la api en openapi 3.0.0. https://app.swaggerhub.com/apis/PabloGonzalezGonzalez/iaas-ansible-backend/1.0.0.

[7] Amazon web service (aws). https://aws.amazon.com/es/.

[8] Qué es microsoft azure. https://azure.microsoft.com/en-us.

[9] Definición de base64. https://en.wikipedia.org/wiki/Base64.

[10] Qué es *cloud computing*. https://www.ibm.com/es-es/cloud/learn/cloud-computing-gbl/.

[11] Cloud-init. https://cloud-init.io/.

[12] Herramienta ddclient. https://ddclient.net/.

[13] Definición de iaas. https://en.wikipedia.org/wiki/Infrastructure_as_a_service.

[14] Dhcp. https://learn.microsoft.com/es-es/windows-server/networking/technologies/dhcp/dhcp-top.

[15] Qué es un servidor dns. https://aws.amazon.com/es/route53/what-is-dns/.

[16] Definición de *endpoint*. https://es.wikipedia.org/wiki/Punto_final_de_comunicación.

[17] Eslint. https://eslint.org/.

[18] Git. https://git-scm.com/.

[19] Github. https://github.com/.

[20] Github projects. https://github.com/features/issues/.

[21] Qué es google cloud platform. https://cloud.google.com/.

[22] Iaas de la ull. https://iaas.ull.es/.

[23] Portal de mis máquinas del iaas de la ull. https://iaas.ull.es/ovirtadmin/usuarios/mismaquinas.

[24] Cotización de un ingeniero informático. https://www.seg-social.es/wps/portal/wss/internet/Trabajadores/CotizacionRecaudacionTrabajadores/36537#36538.

[25] Intellij. https://www.jetbrains.com/es-es/idea/.

[26] Qué es javascript. https://developer.mozilla.org/en-US/docs/Web/JavaScript.

[27] Jetbrains. https://www.jetbrains.com/.

[28] Jinja2. https://pypi.org/project/Jinja2/.

[29] Json. https://www.json.org/json-es.html.

[30] Qué es jwt. https://es.wikipedia.org/wiki/JSON_Web_Token.

[31] Kvm. https://www.redhat.com/es/topics/virtualization/what-is-KVM.

[32] libvirt. https://libvirt.org/.

[33] Microsoft. https://www.microsoft.com/es-es.

[34] Netbeans. https://netbeans.apache.org/.

[35] Npm. https://www.npmjs.com/.

[36] Openapi 3.0. https://swagger.io/specification/.

[37] ovirt. https://www.ovirt.org/.

[38] Módulo *ovirt_auth*. https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_auth_module.html.

[39] Módulo de *ovirt-nics-info*. https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_nic_info_module.html.

[40] Colección *ovirt.ovirt*. https://galaxy.ansible.com/ovirt/ovirt?extIdCarryOver=true&sc_cid=701f2000001OH7YAAW.

[41] Módulo *ovirt_permission*. https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_permission_module.html.

[42] Módulo *ovirt-snapshot*. https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_snapshot_module.html.

[43] Módulo *ovirt_vm*. https://docs.ansible.com/ansible/latest/collections/ovirt/ovirt/ovirt_vm_module.html.

[44] Prettier. https://prettier.io/.

[45] Python. https://www.python.org/.

[46] Qemu guest agent. https://qemu-project.gitlab.io/qemu/interop/qemu-ga.html.

[47] Qué es rest. https://www.redhat.com/es/topics/api/what-is-a-rest-api.

[48] Artículo sobre qué es sdlc. https://aws.amazon.com/es/what-is/sdlc/.

[49] Sitio web del stic de la ull. https://www.ull.es/servicios/stic/.

[50] Subversion. https://subversion.apache.org/.

[51] Swagger.io. https://swagger.io/.

[52] Trabajo fin de grado de miriam núñez garcía, titulado frontend para la gestión de las máquinas virtuales en el iaas de la ull. https://riull.ull.es/xmlui/handle/915/24739.

[53] Definición de certificado *SSL/TLS*. https://aws.amazon.com/es/what-is/ssl-certificate/.

[54] Visual studio code. https://code.visualstudio.com/.

[55] Webstorm. https://www.jetbrains.com/es-es/webstorm/.

[56] Yaml. https://yaml.org/.