



Proyecto Final de Ciclo Cloud Privado Dinámico

Moisés Tamaalit Martínez
2º Administración de Sistemas Informáticos en Red

IES Ingeniero de la Cierva
Curso 2024/2025

Índice general

Información de contacto	3
Preámbulo	4
Resumen	5
Introducción	6
Contextualización	6
Justificación	6
Objetivos del Proyecto	6
Alcance	7
Análisis previo	8
Necesidades detectadas	8
Requisitos técnicos y funcionales	8
Herramientas y tecnologías	8
Estudio de viabilidad	8
Metodología	9
Frontend (React + TypeScript)	9
Backend (Node.js + Express)	9
API Proxmox (Python + FastAPI)	9
Base de Datos (MariaDB)	9
Contenedorización	10
Arquitectura del Sistema	11
Visión General	11
Componentes	11
Frontend	11
Backend API	11
API de Proxmox	11
Desarrollo del Proyecto	12
Implementación Frontend	12
Implementación Backend	15
Estructura de la Base de Datos	16
API Proxmox	17
Imágenes de VPS	19
Seguridad	20
Autenticación y Autorización	20
CORS y Seguridad de la API	20
Seguridad en VPS	20
Resultados	21
Interfaz de Usuario	21



Inicio	21
Página de autenticación	22
Panel de control	23
Creación de VPS	24
Conclusiones y Trabajo Futuro	26
Conclusiones	26
Trabajo Futuro	26
Bibliografía	27



Información de contacto

Nombre	Título	Contacto
Moisés Tamaalit Martínez	Alumno	Email: moisestamaalit@gmail.com

Preámbulo

Antes de todo, quisiera agradecer al **IES Ingeniero de la Cierva** por proporcionar los recursos e infraestructura necesarios para desarrollar este proyecto. Su apoyo constante y el acceso a equipamiento profesional han sido fundamentales para alcanzar los objetivos propuestos.

Gracias a **sys4net** por el aprendizaje y la inspiración que me han proporcionado. En este periodo de prácticas, he podido aplicar y ampliar mis conocimientos en administración de sistemas, lo que ha sido crucial para el desarrollo de este proyecto. La experiencia adquirida en un entorno profesional ha sido invaluable y me ha permitido enfrentar desafíos reales, mejorando mis habilidades técnicas y mi capacidad para resolver problemas.

Gracias a Cayetano por acompañarme como tutor en las RMSkills, Spainskills, Worldskills y próximamente Euroskills. Su apoyo y orientación han sido clave para la gran adquisición de conocimientos y habilidades que he desarrollado durante este tiempo.

Gracias a Alejandro, mi tutor de este proyecto, por las ideas y sugerencias que me ha proporcionado, al igual que por su orientación durante el desarrollo del proyecto.

Gracias a todos mis profesores de este ciclo formativo por toda la enseñanza que me han proporcionado durante estos dos años. Cada uno de ellos ha aportado su grano de arena, enseñándome técnicas, herramientas y conocimientos que han sido esenciales para mi formación como administrador de sistemas.

Un último agradecimiento a José Carmelo, quién en grado medio, hace casi cuatro años, despertó mi interés por la informática, por los sistemas alternativos a Windows, por las redes y todo este mundo. Su pasión por la tecnología y la docencia despertó mi motivación y me impulsó a llevarme a donde estoy hoy.



Resumen

Este proyecto presenta el desarrollo de una plataforma de *hosting* VPS que permite desplegar y gestionar servidores virtuales de forma automatizada. La solución implementa una arquitectura de microservicios moderna, usando React, Node.js y Python, ofreciendo una interfaz intuitiva para usuarios sin conocimientos técnicos avanzados.

Los objetivos principales incluyen la automatización del despliegue de VPS, monitorización en tiempo real y gestión simplificada de servidores. Los resultados demuestran una plataforma funcional que cumple estos objetivos, proporcionando una solución segura y escalable.

Introducción

Contextualización

En la actualidad, la demanda de servidores virtuales privados (VPS) ha aumentado significativamente debido a la creciente necesidad de infraestructura *cloud* flexible y escalable. Las empresas y desarrolladores buscan soluciones que les permitan desplegar y gestionar recursos de forma eficiente.

Justificación

La elección de este proyecto se basa en:

- Experiencia previa en administración de sistemas.
- Interés en tecnologías de virtualización.
- Identificación de la necesidad de soluciones más accesibles.
- Oportunidad de aplicar conocimientos del ciclo formativo.
- Oportunidad de aprender sobre desarrollo web y de APIs.

Objetivos del Proyecto

Principales:

- Desarrollar una plataforma completa de gestión de VPS.
- Automatizar el despliegue de servidores virtuales.
- Implementar monitorización en tiempo real.

Secundarios:

- Crear una interfaz de usuario intuitiva.
- Optimizar el rendimiento del sistema.
- Documentar el proceso de desarrollo.



Alcance

El proyecto abarca el desarrollo de una plataforma de *hosting* VPS que incluye:

- Interfaz web para usuarios.
- API para gestión de VPS.
- Integración con Proxmox VE como hipervisor.
- Sistema de autenticación y autorización.
- Monitorización de recursos en tiempo real.

El sistema está diseñado para ser escalable y adaptable a diferentes necesidades, permitiendo la incorporación de nuevas funcionalidades en el futuro.

Análisis previo

Necesidades detectadas

El mercado actual de *hosting* VPS presenta una barrera significativa para usuarios sin experiencia técnica avanzada. Los paneles de control existentes suelen ser complejos y poco intuitivos, dificultando la gestión de servidores virtuales. Además, la automatización del despliegue y configuración de VPS es limitada, requiriendo frecuentemente intervención manual.

Requisitos técnicos y funcionales

Los requisitos técnicos incluyen un servidor Proxmox VE como base de virtualización, mínimo 32GB de RAM y procesador con soporte para virtualización (Intel VT-x o AMD SVM).

La plataforma debe soportar múltiples usuarios concurrentes y garantizar el aislamiento de recursos.

Funcionalmente, se requiere una interfaz web amigable con el usuario, sistema de autenticación seguro, panel de control intuitivo y monitorización en tiempo real de recursos.

Herramientas y tecnologías

Para desarrollar el proyecto, se ha usado un servidor **HPE Proliant DL360 Gen9** con **80GB de RAM**, 2 procesadores **Intel Xeon E5-2680 v3** y almacenamiento mixto **HDD/SSD**. Este servidor se encuentra alojado en el centro de datos del IES Ingeniero de la Cierva, lo que permite un acceso rápido y seguro a los recursos.

La implementación se basa en React con TypeScript para el *frontend*, proporcionando una experiencia de usuario moderna y *responsive*. El *backend* utiliza Node.js con Express para la API principal, mientras que Python con FastAPI gestiona la interacción con Proxmox. MariaDB sirve como base de datos relacional, y Docker facilita la contenerización de servicios. Proxmox VE actúa como hipervisor base.

La memoria y documentación del proyecto es escrita en LaTeX, lo que permite una presentación profesional y estructurada de la información. Los diagramas se han realizado con Draw.io, una herramienta en línea que facilita la creación de diagramas y esquemas; y dbdiagram.io para el esquema de la base de datos.

Estudio de viabilidad

El análisis de viabilidad indica que el proyecto es técnicamente factible con los recursos disponibles. Los costes se limitan principalmente al hardware necesario para el servidor Proxmox en este entorno.

En un entorno de producción, hay que sumarle el coste de un bloque de IPs, los dominios que fueran a usarse y el mantenimiento del servidor.

El tiempo estimado de desarrollo es de 4 meses, considerando las fases de diseño, implementación y pruebas.

Los riesgos principales incluyen la complejidad de la integración con Proxmox y la necesidad de optimizar el rendimiento del sistema.



Metodología

Frontend (React + TypeScript)

Para el desarrollo del frontend se ha utilizado **React 19** con **TypeScript**, lo que permite crear una interfaz de usuario moderna y escalable.

React es una biblioteca de JavaScript para construir interfaces de usuario, mientras que **TypeScript** es un superconjunto de JavaScript que añade tipado estático, lo que mejora la calidad del código y facilita el mantenimiento a largo plazo.

La ventaja de usar React es su capacidad para construir componentes reutilizables y su ecosistema robusto. TypeScript añade tipado estático, mejorando la mantenibilidad del código y reduciendo errores en tiempo de ejecución.

Se ha implementado **Tailwind CSS** para estilos y **shadcn/ui** para componentes reutilizables. La navegación se gestiona con React Router y la autenticación se realiza mediante JWT.

Tailwind CSS es un framework de CSS que permite crear diseños personalizados de forma rápida y eficiente, mientras que **shadcn/ui** proporciona componentes predefinidos que facilitan el desarrollo de interfaces atractivas y funcionales.

Para desarrollar la interfaz web, dado que soy un administrador de sistemas y no un desarrollador web *frontend*, he utilizado componentes predefinidos de shadcn/ui, que proporcionan una base sólida y estilizada para la interfaz. Esto permite centrarme en la lógica de negocio y la integración con el *backend* sin preocuparme por el diseño desde cero. Me he apoyado en desarrollo con modelos de inteligencia artificial para generar código y mejorar la eficiencia del desarrollo, como **Claude Sonnet 3.5** o **ChatGPT 4** usando GitHub Copilot, si bien he escrito la mayor parte del código.

Backend (Node.js + Express)

Para el desarrollo del *backend* se ha utilizado **Node.js** con **Express**, lo que permite crear una API RESTful eficiente y escalable.

Node.js es un entorno de ejecución de JavaScript del lado del servidor, que permite construir aplicaciones web rápidas y escalables. Express es un framework minimalista para Node.js que facilita la creación de aplicaciones web y APIs.

API Proxmox (Python + FastAPI)

Para interactuar con el hipervisor, decidí hacer una API separada de la principal, utilizando **Python** con **FastAPI**. Esta elección se basa en securizar la interacción con el hipervisor, aprovechando la facilidad de uso de FastAPI para crear APIs RESTful y su excelente rendimiento.

FastAPI es un framework moderno y rápido para construir APIs con Python, que permite definir rutas y manejar peticiones de forma sencilla. Proxmox proporciona una API REST que facilita la gestión de recursos virtuales, lo que permite interactuar con el hipervisor de forma eficiente.

Base de Datos (MariaDB)

Escogí este sistema gestor de bases de datos (SGBD) por mi experiencia con este.



Contenedorización

Para el despliegue de la aplicación me decidí por usar contenedores Docker, lo que permite una fácil portabilidad y escalabilidad. Cada componente del sistema (frontend, backend, API Proxmox) se ejecuta en su propio contenedor, facilitando la gestión y el despliegue en diferentes entornos.

Me he apoyado con Docker Compose para orquestar los contenedores, definiendo servicios y redes necesarios para la aplicación. Esto permite un despliegue sencillo y reproducible en cualquier entorno compatible con Docker.

Arquitectura del Sistema

Visión General

El sistema se compone de varios componentes independientes que trabajan juntos:

- Docker para contenerización.
 - Frontend web desarrollado con React y TypeScript.
 - Backend API en Node.js con Express.
 - API de Proxmox en Python con FastAPI.
- Base de datos MariaDB.
- Sistema de virtualización Proxmox VE.

Componentes

Frontend

Desarrollado con React 19 y TypeScript, utiliza:

- Tailwind CSS para estilos
- Shadcn/ui para componentes de interfaz
- React Router para navegación
- JWT para autenticación

Backend API

Implementado en Node.js con Express:

- Express para el servidor web
- JWT para autenticación
- MariaDB para persistencia de datos
- Bcrypt para hash de contraseñas

API de Proxmox

Desarrollada en Python con FastAPI:

- FastAPI para la API REST
- Proxmox API Client para interacción con Proxmox
- Pydantic para validación de datos

Desarrollo del Proyecto

Implementación Frontend

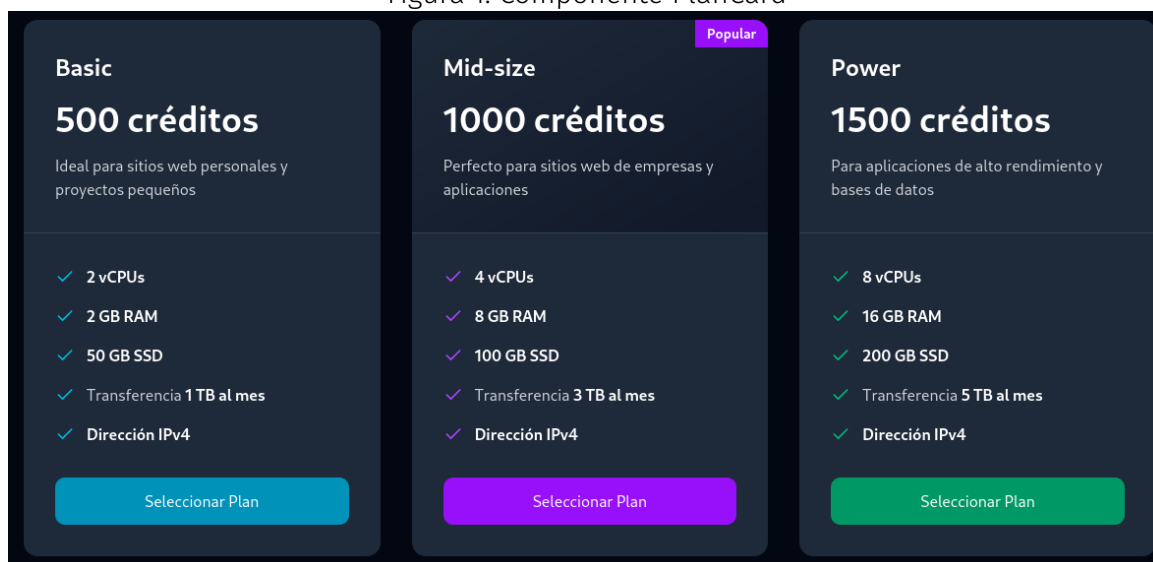
Para poder programar con React 19 y TypeScript, he utilizado **npm** como gestor de paquetes con Vite como entorno de desarrollo. Para preparar el entorno, previamente instalé Node.js y npm en mi máquina. Luego, creé un nuevo proyecto con el siguiente comando:

```
npm create vite@latest frontend --template react-ts
```

Para la interfaz de usuario quería que se viera moderna y profesional, por ello me he apoyado en componentes de shadcn/ui, que sin mucho código aportan un diseño atractivo. Para demostrar la utilidad de los componentes, miren el siguiente ejemplo.

Esta sección de la página principal, que muestra los planes de VPS disponibles, se ha implementado utilizando un componente personalizado PlanCard, que facilita la reutilización del código:

Figura 1: Componente PlanCard



Aquí se muestra el código para mostrar los planes:

```
<section className="py-16">
  <div className="container mx-auto px-4">
    <div className="max-w-6xl mx-auto">
      <div className="grid md:grid-cols-3 gap-8">
        {plans.map((plan) => {
          const style = planStyles[plan.name as keyof typeof planStyles]
          return (
            <PlanCard
              key={plan.id}
              plan={plan}
              style={style}
              onSelect={handleSelectPlan}
            />
          )
        })}
      </div>
    </div>
  </div>
</section>
```

Como se puede apreciar, solo se llama al componente en lugar de todo el código necesario para mostrarlo.

Todos los datos de los planes se obtienen de la base de datos a través del *backend*, lo que permite una fácil actualización y gestión de los planes disponibles.

Para el desarrollo del frontend, he usado el servidor integrado de Vite, el cual se lanza con el siguiente comando:

```
npm run dev
```

Esto inicia un servidor de desarrollo que permite ver los cambios en tiempo real mientras se desarrolla la aplicación.

De cara al despliegue en producción, he utilizado Docker para hacer una imagen de la aplicación con un *build multi-stage*, lo que permite crear un contenedor para construir el fichero final de frontend, mientras que el segundo contenedor (nginx) sirve el contenido estático generado por el primer contenedor. Más detalles en la sección de Docker.

Todo el código se encuentra en el Anexo A.

El siguiente diagrama de flujo muestra las acciones que se pueden realizar desde el frontal:



El cliente tendrá que registrarse para realizar cualquier acción distinta de ver los planes. Una vez registrado, podrá adquirir un VPS, editar su perfil, ver los detalles de su VPS, etc.

Implementación Backend

El servidor backend no está expuesto al usuario directamente, se accede por un proxy inverso configurado en el frontend.

Para empezar con el desarrollo del backend, he creado un nuevo proyecto de Node.js con Express utilizando el siguiente comando:

```
npm init -y
npm install express cors dotenv mariadb bcrypt jsonwebtoken
```

A partir de aquí, he creado un fichero *server.js* que contiene la configuración básica del servidor Express para su ejecución. Este fichero es el punto de entrada de la aplicación y se encarga de configurar las rutas, middleware y la conexión a la base de datos; escritas en otros ficheros.

Después, he creado diferentes directorios para organizar el código:

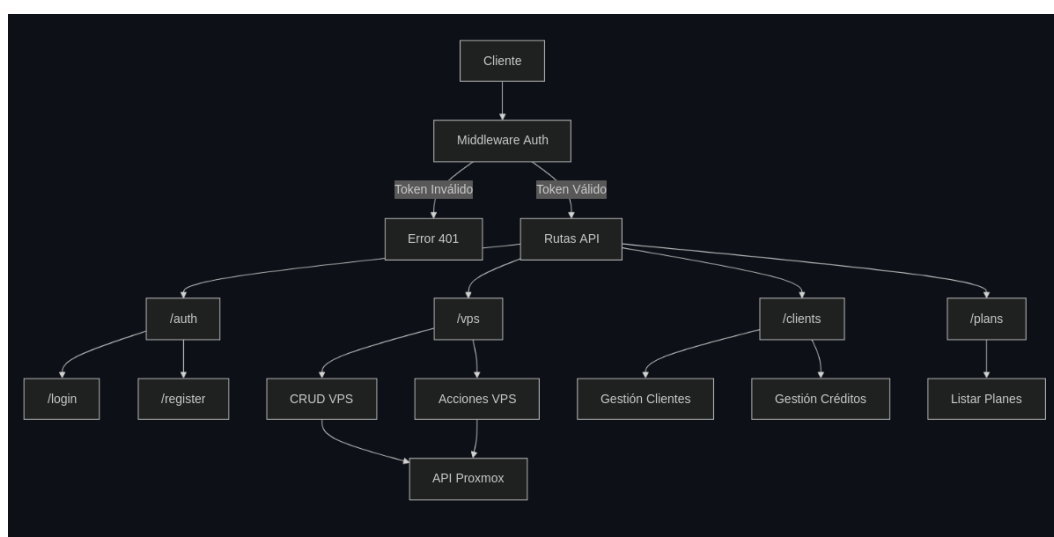
```
mkdir routes controllers config middleware
```

Como sus nombres rezan, cada directorio tiene una función específica:

- **routes:** Contiene los archivos de rutas que definen los endpoints de la API.
- **controllers:** Contiene la lógica de negocio para cada endpoint, separando la lógica de las rutas.
- **config:** Contiene la configuración de la base de datos y otros parámetros del servidor.
- **middleware:** Contiene la lógica de autenticación a través de JWT.

Los datos de conexión a BBDD, en el directorio *config*, los aporta Docker por medio de variables de entorno, lo que permite una fácil configuración y despliegue en diferentes entornos.

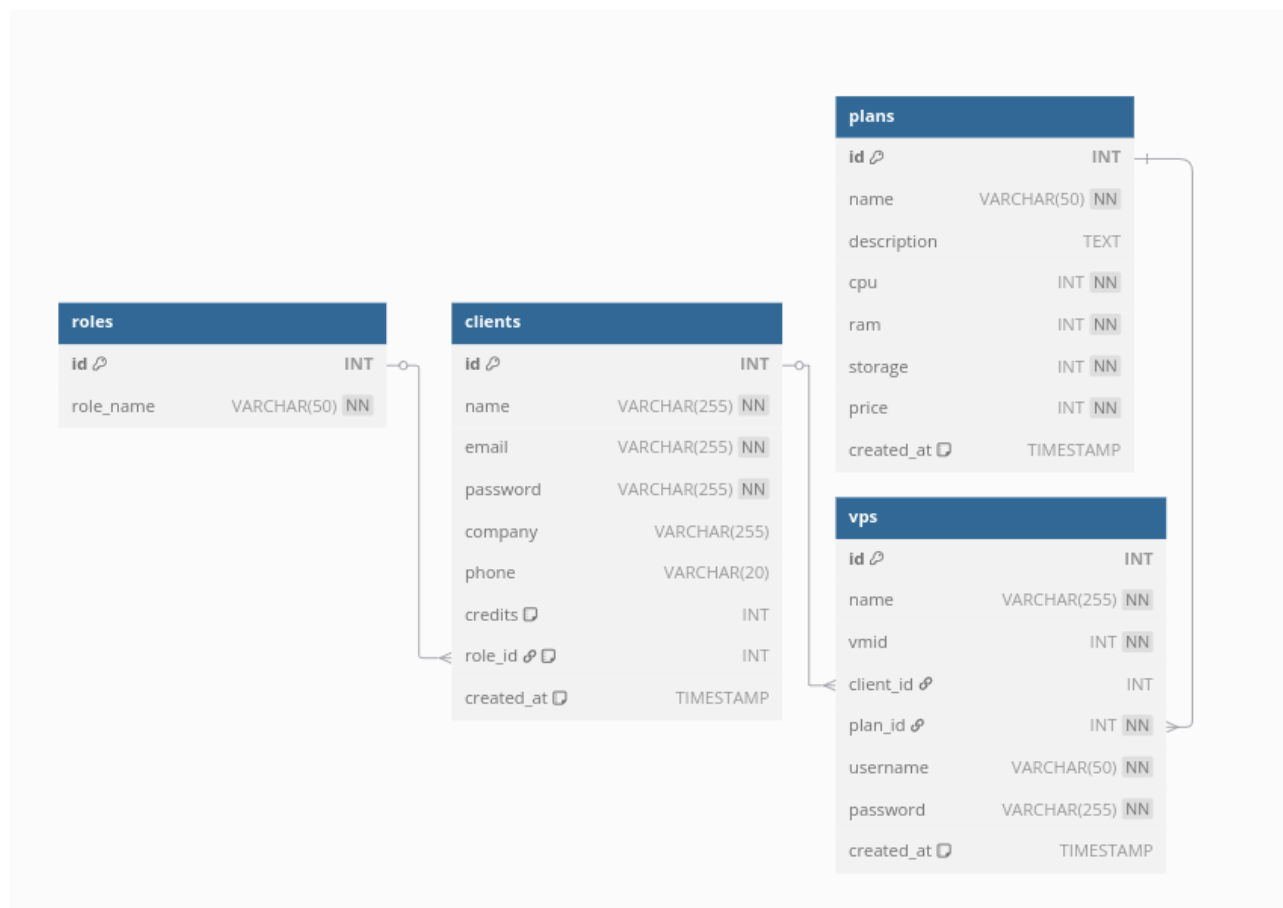
Adjunto un diagrama de flujo que muestra las acciones que se pueden realizar desde el backend:



Todas las acciones implican interacción con la base de datos, ya sea para crear un usuario, autenticarlo, obtener los planes disponibles o gestionar los VPS del usuario.

Estructura de la Base de Datos

La base de datos se ha diseñado para almacenar la información necesaria para gestionar los VPS y los usuarios. La estructura incluye las siguientes tablas:



En el Anexo A se podrá ver el código SQL para crear las tablas y sus relaciones.

API Proxmox

Este es el núcleo del proyecto, puesto que esta API es la que interactúa directamente con el hipervisor Proxmox para gestionar los VPS, lo que permite versatilidad para usarla en otros proyectos o incluso para crear un panel de control personalizado.

Para empezar con el desarrollo de la API de Proxmox, he creado un nuevo proyecto de Python con FastAPI en un entorno virtual utilizando los siguientes comandos:

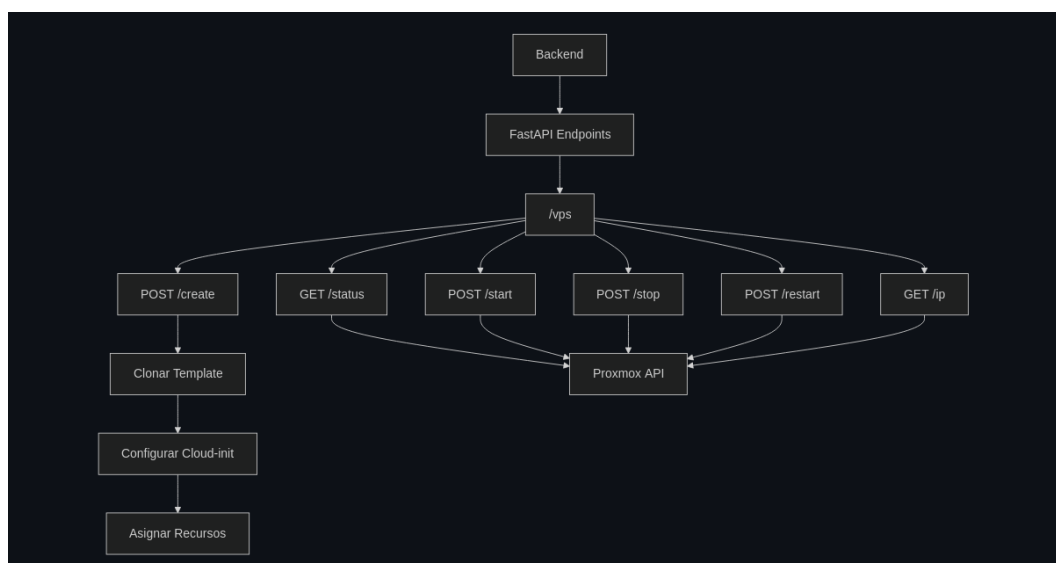
```
python -m venv .venv
source .venv/bin/activate
pip install fastapi uvicorn proxmoxer pydantic
```

He utilizado la librería **proxmoxer** para interactuar con la API de Proxmox, así evito tener que lidiar con las peticiones HTTP directamente. Por otro lado, uvicorn es un servidor ASGI que permite ejecutar aplicaciones FastAPI de forma eficiente y el que está a la escucha de las peticiones.

Para este proyecto, dentro del directorio **app** he creado cuatro directorios principales y un fichero **main.py** que es el punto de entrada de la aplicación.

- **routes:** Contiene los archivos de rutas que definen los endpoints de la API y la lógica de negocio asociada a cada uno.
- **models:** Contiene los modelos de datos utilizados por Pydantic para validar las entradas y salidas de la API.
- **services:** Contiene funciones auxiliares que no necesariamente son endpoints, pero que se utilizan en la lógica de negocio de las rutas,
- **config:** Contiene la cadena de conexión a Proxmox y otras configuraciones necesarias para la API.

Adjunto un diagrama de flujo que muestra las acciones que se pueden realizar desde la API de Proxmox:





En esta API no se valida el origen de la petición ni hay autenticación de usuarios, ya que se supone que solo se va a acceder desde el backend de la aplicación y solo es accesible desde la red interna de Docker. Sin embargo, se valida la entrada de datos en cada endpoint utilizando Pydantic, lo que garantiza que los datos enviados a Proxmox sean correctos y evitar errores.

Cuando se crea una máquina virtual, lo que hace esta API es clonar una plantilla de Proxmox previamente creada, que contiene el sistema operativo y las configuraciones básicas necesarias para el VPS. Para aumentar el almacenamiento se aumenta el tamaño del disco del clon, y de forma automática se incrementan los tamaños de las particiones y del sistema de archivos, para que el usuario pueda usar todo el espacio disponible sin necesidad de realizar ninguna acción adicional. Esto es así por ser una imagen cloud, que ya viene preparada para ser utilizada en entornos de virtualización.



Imágenes de VPS

Para crear las imágenes de los VPS, he utilizado scripts de Bash que se ejecutan en el hipervisor Proxmox. Estos scripts descargan las imágenes de los sistemas operativos desde sus fuentes oficiales y las preparan para ser utilizadas como plantillas.

He creado dos scripts, uno para AlmaLinux y otro para Ubuntu, que realizan las siguientes acciones:

- Descargan una imagen para cloud del sistema operativo desde su repositorio oficial.
- Descomprimen la imagen y la convierten al formato necesario para Proxmox.
- Configuran la imagen para que sea compatible con Proxmox, incluyendo la instalación de `qemu-guest-agent` y la configuración de `cloud-init`.

La elección de estos sistemas operativos se basa en su popularidad y estabilidad, siempre quedando el sistema abierto a incorporar otros sistemas operativos en el futuro.

Seguridad

Autenticación y Autorización

Para garantizar la seguridad de la plataforma, se ha implementado un sistema de autenticación y autorización robusto. Los usuarios deben registrarse y autenticarse para acceder a las funcionalidades del sistema.

El sistema utiliza JWT (JSON Web Tokens) para gestionar las sesiones de usuario. Al iniciar sesión, se genera un token que se envía al cliente y se almacena en el almacenamiento local del navegador. Este token se incluye en las cabeceras de las peticiones a la API para autenticar al usuario.

JWT es un estándar abierto (RFC 7519) que define un formato compacto y autónomo para transmitir información entre partes como un objeto JSON. Este token se firma digitalmente, lo que garantiza su integridad y autenticidad. La información que contiene el token incluye el ID del usuario, la fecha de expiración y otros datos relevantes. En mi caso incluye el rol del usuario.

Las contraseñas de los usuarios se almacenan de forma segura utilizando bcrypt, un algoritmo de hash que protege las contraseñas contra ataques de fuerza bruta. Además, se valida la entrada de datos en todos los endpoints para evitar inyecciones SQL y otros ataques comunes.

CORS y Seguridad de la API

Para proteger la API de accesos no autorizados, se ha configurado CORS (Cross-Origin Resource Sharing) en el servidor Express. Esto permite que solo las solicitudes desde el dominio del frontend sean aceptadas, bloqueando cualquier intento de acceso desde otros orígenes.

CORS es un mecanismo de seguridad que permite o restringe el acceso a recursos web desde diferentes dominios. En este caso, se ha configurado para permitir solo solicitudes desde el dominio del frontend, lo que evita ataques de tipo CSRF (Cross-Site Request Forgery).

Seguridad en VPS

Los VPS creados a través de la plataforma cuentan con varias medidas de seguridad preconfiguradas para proteger los recursos del usuario:

- Aislamiento de recursos.
- Redes virtuales privadas.
- Firewall configurado por defecto.
- Actualización automática de plantillas.

Resultados

Al finalizar el proyecto, se ha conseguido una plataforma funcional de gestión de VPS que cumple con los objetivos propuestos. La solución permite a los usuarios desplegar y gestionar servidores virtuales de forma sencilla y segura, sin necesidad de conocimientos técnicos avanzados.

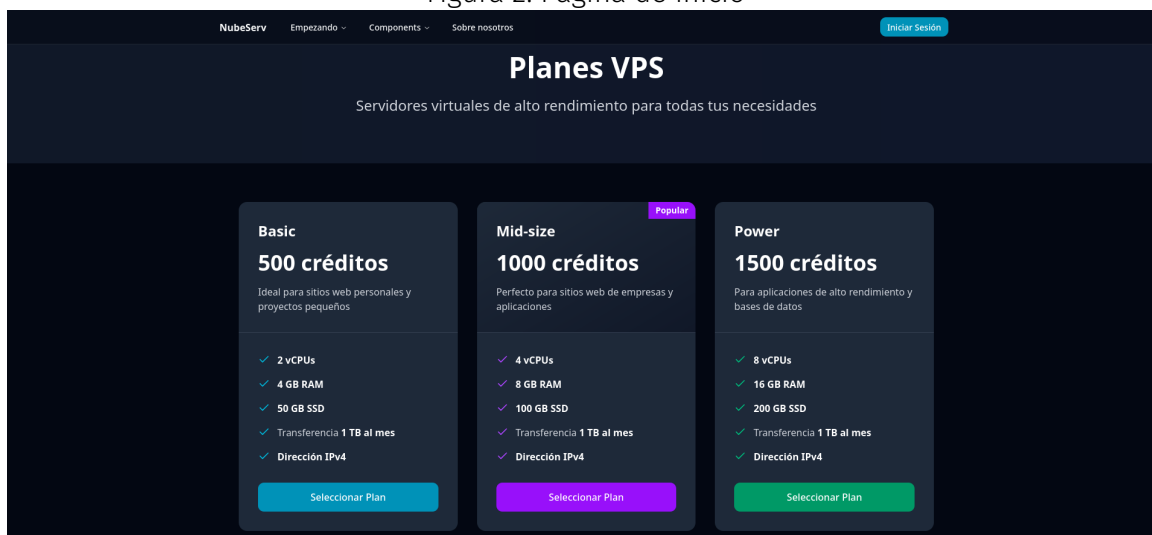
Interfaz de Usuario

La interfaz de usuario es intuitiva y fácil de usar, permitiendo a los usuarios gestionar sus VPS de forma eficiente.

Inicio

La página de inicio muestra los planes disponibles y permite a los usuarios registrarse o iniciar sesión.

Figura 2: Página de inicio



Página de autenticación

La página de autenticación permite a los usuarios registrarse o iniciar sesión en la plataforma. El registro requiere un nombre de usuario, correo electrónico y contraseña, mientras que el inicio de sesión solicita solo el correo electrónico y la contraseña.

Figura 3: Formulario de inicio de sesión

The screenshot shows the login form on the NubeServ website. The header includes the NubeServ logo and navigation links: Empezando, Components, and Sobre nosotros. A blue button labeled 'Iniciar Sesión' is in the top right. The main content area features a dark blue box with the title 'Iniciar Sesión'. Below the title is a link: '¿No tienes cuenta? Regístrate'. The form contains two input fields: 'Correo electrónico' and 'Contraseña'. At the bottom of the box is a blue button labeled 'Iniciar Sesión'.

Figura 4: Formulario de registro

The screenshot shows the registration form on the NubeServ website. The header is identical to the login page. The main content area features a dark blue box with the title 'Crear Cuenta'. Below the title is a link: '¿Ya tienes cuenta? Inicia sesión'. The form contains five input fields: 'Nombre completo', 'Correo electrónico', 'Contraseña', 'Empresa (opcional)', and 'Teléfono (opcional)'. At the bottom of the box is a blue button labeled 'Crear Cuenta'.

Panel de control

El panel de control muestra los VPS del usuario, permitiendo crear, editar y eliminar servidores virtuales. También se pueden ver los detalles de cada VPS, incluyendo el estado, recursos asignados y acciones disponibles.

Figura 5: Panel de control - Sección de VPS

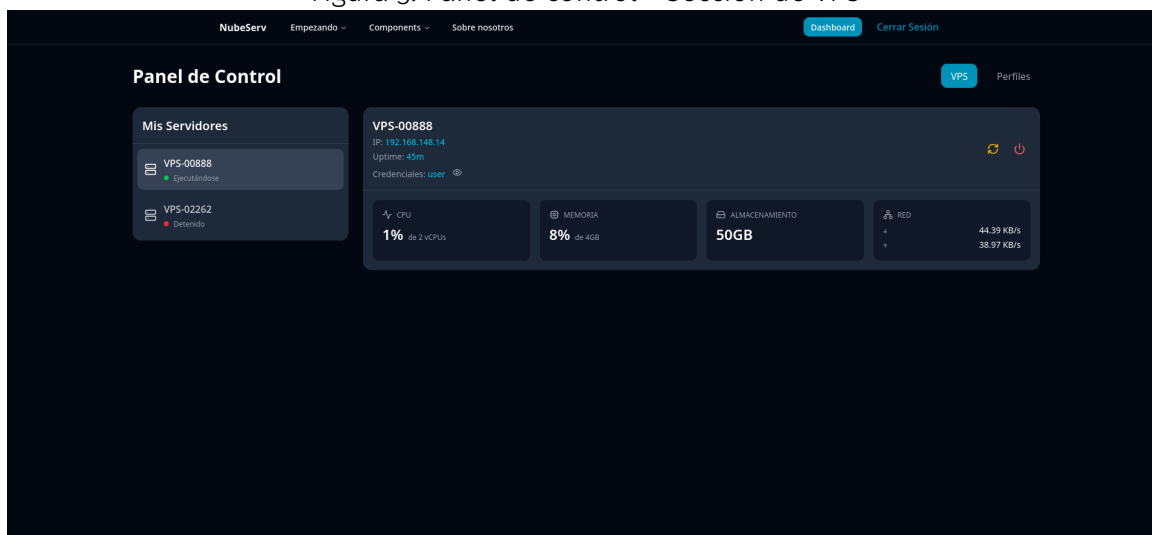
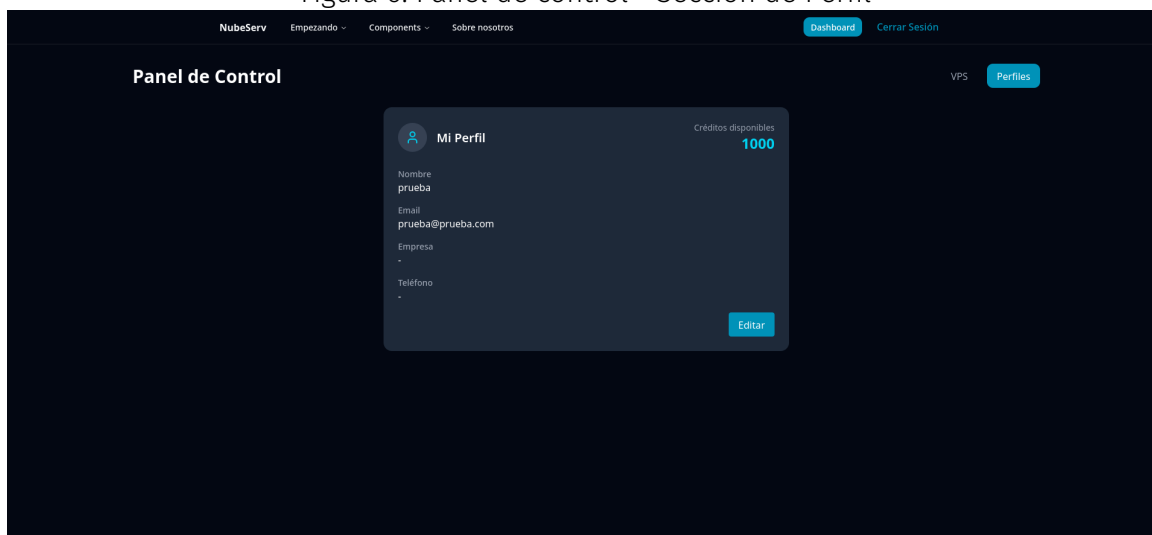


Figura 6: Panel de control - Sección de Perfil



Creación de VPS


La creación de un VPS se realiza a través de un formulario que permite seleccionar el plan deseado. Una vez enviado el formulario, se crea el VPS en Proxmox y se muestra en el panel de control del usuario.

Figura 7: Formulario de creación de VPS

Tu Saldo

1000 créditos


Sistema Operativo



Ubuntu

Ubuntu 24.04

LTS Release



AlmaLinux 9

Enterprise Linux

Plan Seleccionado

Basic

500 créditos/mes

2 vCPUs
4 GB RAM
50 GB SSD
Transferencia infinita

Mid-size

1000 créditos/mes

4 vCPUs
8 GB RAM
100 GB SSD
Transferencia infinita

Power

1500 créditos/mes

8 vCPUs
16 GB RAM
200 GB SSD
Transferencia infinita
Necesitas 500 créditos más para este plan

Crear VPS

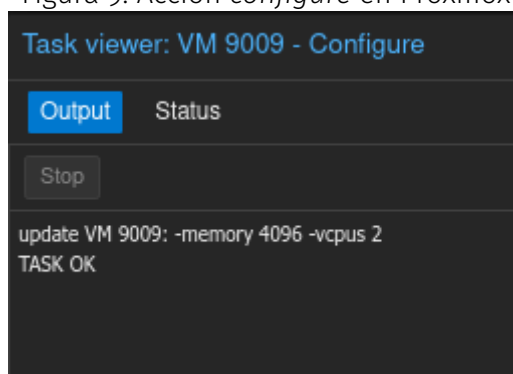
En Proxmox genera las siguientes acciones, de forma automatizada por una llamada de la API de Express (la de la web) hacia la API de Proxmox (la de Python):

Figura 8: Acciones generadas en Proxmox

Tasks Cluster log				
Start Time ↓	End Time	Node	User name	Description
Jun 10 18:28:32	Jun 10 18:56:34	PVE-POTE...	root@pam	VM/CT 9008 - Console
Jun 10 18:27:18	Jun 10 18:27:23	PVE-POTE...	api-user@pve	VM 9009 - Start
Jun 10 18:27:09	Jun 10 18:27:10	PVE-POTE...	api-user@pve	VM/CT 9009 - Resize
Jun 10 18:27:09	Jun 10 18:27:09	PVE-POTE...	api-user@pve	VM 9009 - Configure
Jun 10 18:27:05	Jun 10 18:27:09	PVE-POTE...	api-user@pve	VM 9001 - Clone

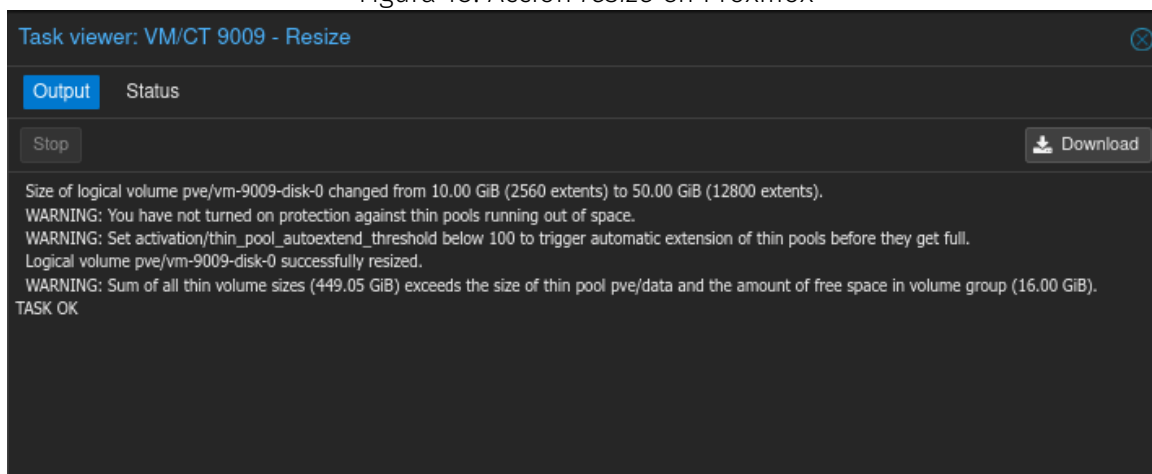
La acción *configure* asigna la RAM y el número de núcleos:

Figura 9: Acción *configure* en Proxmox



La acción *resize* aumenta el tamaño del disco:

Figura 10: Acción *resize* en Proxmox



Conclusiones y Trabajo Futuro

Conclusiones

Con este proyecto, he logrado desarrollar una plataforma de *hosting* VPS que permite a los usuarios desplegar y gestionar servidores virtuales de forma sencilla y segura.

Se ha desplegado con un método fácil de usar, lo que facilita el desarrollo de la aplicación y su publicación.

Ha sido gracias a soluciones de código abierto que he podido desarrollar esta plataforma con facilidad y rapidez, como **proxmoxer** para interactuar con Proxmox, **FastAPI** para la API de Proxmox, y **shadcn/ui** para los componentes del frontend.

Trabajo Futuro

De cara al futuro, y si hubiera más tiempo, se podrían implementar las siguientes mejoras:

- **Soporte para más sistemas operativos:** Ampliar la plataforma para incluir otros sistemas operativos como Debian, Fedora, Windows Server, etc., lo que aumentaría la versatilidad de la solución.
- **Sistema de copias de seguridad:** Con Proxmox Backup Server se puede implementar muy rápidamente un sistema robusto de copias de seguridad.
- **Métricas avanzadas con histórico:** Ahora mismo solo se muestran las métricas actuales de CPU, RAM y disco, pero se podría implementar un sistema de monitorización más avanzado que permita ver el histórico de uso de recursos y alertas con Zabbix.
- **Soporte para múltiples hipervisores:** Ampliar la plataforma para soportar otros hipervisores como vSphere, KVM, Nutanix, etc., lo que aumentaría la versatilidad de la solución.
- **Personalización del VPS:** Posibilidad de que el usuario inserte su usuario, contraseña y clave SSH en el formulario de creación.



Bibliografía

- **React:** <https://reactjs.org/>
- **FastAPI:** <https://fastapi.tiangolo.com/>
- **Proxmox:** <https://pve.proxmox.com/wiki/>
 - **Proxmox VE API Documentation:** <https://pve.proxmox.com/pve-docs/api-viewer/>
- **Express - Node.js web application framework:** <https://expressjs.com/>
- **Documentation - MariaDB.org:** <https://mariadb.org/documentation/>
- **Build your Component Library - shadcn/ui:** <https://ui.shadcn.com/>
- **Get started with Tailwind CSS:** <https://tailwindcss.com/docs>
- **Docker Docs:** <https://docs.docker.com/>
- Manhas Mansi. How To Use An .env File In Docker Compose.
<https://www.warp.dev/terminus/docker-compose-env-file>