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

Fecha: 9 de junio de 2025
Proyecto: Cloud Privado Dinámico
Versión 1.0

Índice general

Información de contacto

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

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

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 incluye:

- Sistema de gestión de usuarios
- Panel de control de VPS
- Monitorización básica de recursos
- Automatización de despliegues

No incluye:

- Sistema de facturación
- Soporte para múltiples centros de datos
- Backup automático de VPS
- Sistema de tickets de soporte

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

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.

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.

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.

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.

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 ideal para aplicaciones en tiempo real y Express facilita la creación de rutas y middleware.

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. Además, Proxmox proporciona una API REST que se integra bien con FastAPI.

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

Descripción detallada del desarrollo frontend, incluyendo:

- Estructura de componentes
- Sistema de autenticación
- Panel de control
- Monitorización

Fase 3: Implementación Backend

Desarrollo del servidor Express, incluyendo:

- Sistema de rutas
- Middleware de autenticación
- Integración con base de datos
- API RESTful

Fase 4: API Proxmox

Implementación de la API de Proxmox:

- Endpoints principales
- Sistema de plantillas
- Gestión de recursos
- Monitorización

Resultados

Funcionalidades Implementadas

- Gestión de VPS
- Sistema de usuarios
- Monitorización
- Automatización

Evaluación de Objetivos

Análisis del cumplimiento de objetivos:

- Funcionalidad
- Rendimiento
- Seguridad
- Usabilidad

Seguridad

Autenticación y Autorización

- JWT para tokens de sesión
- Contraseñas hasheadas con bcrypt
- Validación de datos en endpoints
- CORS configurado por seguridad

Seguridad en VPS

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

Conclusiones y Trabajo Futuro

Objetivos Alcanzados

- Plataforma funcional de gestión de VPS
- Sistema de autenticación seguro
- Despliegue automatizado de servidores
- Monitorización en tiempo real
- Interfaz de usuario intuitiva

Trabajo Futuro

- Sistema de copias de seguridad
- Métricas avanzadas
- Soporte para múltiples hipervisores
- API pública para integraciones
- Panel de administración avanzado

Bibliografía

- Documentación oficial de React: https://reactjs.org/
- Documentación de FastAPI: https://fastapi.tiangolo.com/
- Documentación de Proxmox: https://pve.proxmox.com/wiki/
    - Documentación de la API de Proxmox: https://pve.proxmox.com/pve-docs/api-viewer/
- Documentación de Express: https://expressjs.com/
- Documentación de MariaDB: https://mariadb.org/documentation/
- Documentación de shadcn/ui: https://ui.shadcn.com/
- Documentación de Tailwind CSS: https://tailwindcss.com/docs
- Documentación de Docker: https://docs.docker.com/