Plataforma de Hosting Académico Basada en Contenedores

Andrés Gómez
Andrea De La Ossa
Sofía Palacio
Dayana Molina
José Sequeda

Introducción

En este proyecto se desarrolla una plataforma de hosting que permite a los usuarios desplegar sus propios sitios web directamente desde un repositorio de GitHub, utilizando Docker como tecnología base para la ejecución. El sistema integra autenticación mediante Roble, ofrece plantillas iniciales dockerizadas y automatiza el proceso de creación, despliegue y administración de cada proyecto.

La plataforma está conformada por tres componentes principales: un panel de control (dashboard) utilizado por el usuario, un servicio backend encargado de la gestión de contenedores, autenticación y comunicación con Roble, y un reverse proxy responsable de enrutar las solicitudes hacia los proyectos activos mediante subdominios locales. Además, se implementan mecanismos de control de recursos y automatización, como apagado de contenedores inactivos y limitación de recursos computacionales.

El objetivo general es ofrecer un entorno funcional donde cada usuario pueda registrar proyectos, desplegarlos en contenedores independientes y acceder a ellos mediante URLs personalizadas, garantizando aislamiento, seguridad y eficiencia en el uso de recursos.

La plataforma de hosting está diseñada bajo una arquitectura modular basada en microservicios. Cada componente cumple una función específica dentro del flujo completo: autenticación, gestión de proyectos, ejecución de contenedores y despliegue del contenido web del usuario. La comunicación entre los servicios se realiza a través de una red interna Docker, y el reverse proxy centraliza el enrutamiento de todas las solicitudes externas.

Descripción de la Arquitectura y Componentes

La plataforma está diseñada siguiendo una arquitectura modular basada en microservicios, donde cada componente del sistema se ejecuta de manera independiente dentro de su propio contenedor Docker. Esta separación permite aislar responsabilidades, facilitar el despliegue, mejorar la escalabilidad y mantener un control claro sobre el flujo completo del sistema. Todos los servicios se encuentran conectados mediante una red interna definida en Docker Compose, y las interacciones externas se canalizan a través de un reverse proxy central.

A nivel global, la arquitectura está compuesta por tres elementos principales: un panel de control o *dashboard*, un servicio backend responsable de la lógica del sistema y la gestión de contenedores, y un reverse proxy que enruta solicitudes y permite el uso de subdominios locales.

A continuación, se describen los componentes principales de la arquitectura:

1. **Dashboard (Frontend)**

   El panel de control del sistema se implementa como una aplicación web desarrollada en React y servida mediante un contenedor Nginx. Desde esta interfaz, los usuarios pueden autenticarse, consultar las plantillas disponibles, registrar nuevos proyectos basados en repositorios de GitHub y visualizar el estado de los despliegues asociados a su cuenta. El dashboard se comunica con el backend mediante solicitudes HTTP, enviando tokens de autenticación cuando es necesario. Todo el tráfico pasa por el reverse proxy, que centraliza el enrutamiento.

2. **Manager (Backend y Gestión de Contenedores)**

   Este módulo, implementado en Flask, constituye el núcleo lógico de la plataforma. Se encarga de la autenticación, comunicación con Roble, administración de proyectos y despliegue automático de contenedores Docker.

   - *Autenticación:* El backend utiliza un cliente dedicado para comunicarse con la API de Roble. Se validan credenciales, se emiten y verifican tokens, se registran usuarios y se obtiene información asociada a los proyectos. Las rutas críticas están protegidas con un decorador que valida los tokens.
   - *Gestión de proyectos:* El backend expone rutas para registrar proyectos, asociarlos al usuario autenticado, consultarlos y actualizar sus atributos. Esta información se almacena de forma remota en Roble.
   - *Despliegue en contenedores:* El proceso de despliegue clona el repositorio del usuario, construye la imagen Docker correspondiente, ejecuta un contenedor

aislado para ese proyecto, establece el puerto interno adecuado y registra el estado final en Roble.

- *Monitoreo y control:* Un monitor de actividad verifica periódicamente la validez del token del Manager y facilita la futura implementación de suspensión de contenedores inactivos o reactivación bajo demanda.

3. **Reverse Proxy**

   El reverse proxy, implementado en Nginx, actúa como puerta de entrada única a la plataforma. Redirige el tráfico al dashboard o al backend según la ruta solicitada e incorpora soporte para subdominios locales. Gestiona un sistema de mapeo dinámico que asocia dominios del tipo `nombreProyecto.nombreUsuario.localhost` con el contenedor correspondiente.

4. **Plantillas Dockerizadas**

   El sistema incluye tres plantillas listas para su despliegue:

   - Sitio estático en HTML, CSS y JavaScript.
   - Aplicación React.
   - Aplicación Flask.

   Cada plantilla está completamente dockerizada e incluye su propio Dockerfile. El backend ofrece una API que permite mostrar estos templates en el dashboard.

5. **Orquestación con Docker Compose**

   Docker Compose coordina la ejecución de todos los servicios del sistema. Define redes internas, dependencias entre contenedores, puertos expuestos, healthchecks y scripts de inicialización.

6. **Integración con Roble**

   Roble funciona como servicio externo de autenticación y almacenamiento. Permite gestionar usuarios, validar tokens, registrar proyectos y almacenar metadatos relacionados con contenedores y actividad.

1. Flujo de Trabajo del Sistema

El sistema implementa un flujo que conecta el frontend (React/Vite), el backend central (Flask Manager), la infraestructura Docker y la base de datos gestionada por Roble. A continuación se describe el proceso detalladamente.

1.1. 1. Autenticación de usuario

1. El usuario accede al sistema mediante el navegador ingresando a `http://localhost` o un subdominio como `projectX.localhost`.
2. NGINX recibe la solicitud de acceso. Si la ruta es raíz `/`, sirve la aplicación frontend generada por React/Vite.
3. El usuario inicia sesión desde el formulario en el Dashboard.
4. La aplicación envía un `POST /auth/login` que es redirigido por NGINX hacia el Manager Flask.
5. El módulo **Auth** utiliza la API de Roble para validar credenciales y obtener un `accessToken`.
6. El token es devuelto al frontend y almacenado en el *Auth Context*.
7. El frontend envía `/auth/use_token` al Manager para activar el Monitor interno y registrar el token global.

1.2. 2. Navegación en el Dashboard

- El usuario puede acceder a las vistas: **Dashboard, Mis Proyectos, Crear Proyecto**.
- Para cada vista, React realiza solicitudes API como `GET /projects/mine` o `POST /projects/create`.
- Todas las solicitudes son enviadas con el encabezado `Authorization: Bearer <token>`.

1.3. 3. Creación de Proyectos

1. El usuario ingresa el nombre del proyecto y la URL del repositorio.
2. Se realiza un `POST /projects/create` hacia el Manager Flask.
3. El módulo **Projects** valida el token con Roble y obtiene el `user_id`.
4. Se crea un registro en Roble (tabla proyectos) con la información del repositorio.
5. El módulo **Containers** despliega el proyecto en Docker (clonado, build, run).
6. Se registra en Roble el `container_id` y se marca el proyecto como `running`.

1.4. 4. Acceso a Contenedores de Proyectos

- Cada proyecto se ejecuta en un contenedor Docker independiente (ej: `project1:5000`).

- NGINX enruta dominios hacia el contenedor correspondiente: `project1.localhost → project1:5000`.
- El sistema permite detener, reiniciar o consultar estado del contenedor.

1.5. 5. Monitorización en Tiempo Real (SSE)

- El Manager utiliza un **ActivityMonitor** que valida periódicamente el token contra Roble.
- Se puede emitir información mediante eventos SSE con estado activo del sistema.
- Se actualiza `last_access` y métricas de uso asociadas al proyecto.

1.6. 6. Gestión y Optimización de Recursos (Punto clave del sistema)

1. Un módulo interno de **Optimización** consulta periódicamente Roble para obtener:
   - Actividad del contenedor
   - Último acceso del usuario
   - Métricas de uso almacenadas
2. Si un contenedor permanece inactivo por encima del umbral configurado, el sistema ejecuta políticas automáticas:
   - Suspensión automática (`docker stop`)
   - Pausa del contenedor (`docker pause`)
   - Limpieza opcional de imágenes antiguas
3. Tras la acción, el Manager actualiza en Roble el estado del proyecto para reflejar `suspended`, `stopped`, o similar.
4. Esto reduce uso de CPU, RAM y consumo general de infraestructura.

Estrategias de Seguridad y Optimización de Recursos

La plataforma incorpora diversas medidas orientadas a garantizar la seguridad del sistema y a optimizar el uso de recursos durante la ejecución de los proyectos. Estas estrategias abarcan tanto el control de acceso y la protección de datos como el uso eficiente de los contenedores que alojan las aplicaciones de los usuarios. En un entorno donde múltiples proyectos se ejecutan de manera simultánea, resulta fundamental disponer de controles que aseguren el aislamiento, la disponibilidad y la integridad de cada servicio desplegado.

En primer lugar, la seguridad del sistema se sustenta en el mecanismo de autenticación proporcionado por Roble. Todas las operaciones sensibles, como la creación de proyectos, la consulta de su estado o la ejecución de despliegues, requieren un token válido que identifica al usuario. Este token es verificado desde el backend en cada solicitud, lo que garantiza que solo usuarios autorizados puedan interactuar con sus propios recursos. Para reforzar este proceso, el backend incluye un componente de protección que actúa antes de resolver cada ruta crítica, validando la autenticidad del token y asegurando que la petición provenga de un usuario legítimo. Este enfoque evita accesos indebidos y mantiene un registro claro de las acciones vinculadas a cada identidad.

Un aspecto clave dentro de la seguridad es el aislamiento proporcionado por los contenedores Docker. Cada proyecto se ejecuta dentro de un contenedor independiente, lo que impide cualquier tipo de interferencia o acceso no autorizado entre los proyectos de distintos usuarios. Al mantener entornos separados, se reduce el riesgo de que una aplicación defectuosa o mal configurada afecte el funcionamiento de otras aplicaciones alojadas en la plataforma. Esta separación es fundamental para garantizar un entorno estable en un sistema multiusuario.

El reverse proxy cumple un papel adicional en la protección del sistema. Todos los accesos externos pasan primero por este componente, que redirige las solicitudes hacia el dashboard o el backend según corresponda. Gracias a este diseño, los servicios internos no están expuestos directamente al exterior, lo que disminuye la superficie de ataque. El proxy controla también las cabeceras y los flujos de autenticación, proporcionando una capa adicional de validación y evitando que el cliente interactúe directamente con los puertos internos de la infraestructura.

Otro aspecto relevante dentro de la seguridad es la verificación del código fuente que los usuarios proporcionan. Cuando un repositorio es registrado para desplegar un proyecto, el backend valida que la URL sea correcta, supervisa el proceso de construcción de la imagen y detecta errores en la compilación o en el Dockerfile. Estas comprobaciones permiten identificar anticipadamente repositorios dañados o mal configurados y evitan que una imagen defectuosa llegue a ejecutarse dentro del entorno compartido.

En cuanto a la optimización de los recursos, la plataforma está diseñada para evitar que los contenedores consuman más capacidad de la necesaria. El proceso de despliegue permite incorporar limitaciones de CPU y memoria en cada contenedor, de manera que el uso de hardware por parte de los proyectos permanezca dentro de rangos seguros. Esto evita que una aplicación con alta demanda o con fallos de rendimiento afecte el funcionamiento del sistema global.

Asociado al control de recursos, la plataforma integra un monitor de actividad encargado de registrar el último acceso a cada proyecto. Este componente analiza periódicamente

la actividad de los contenedores con el fin de identificar aquellos que permanecen inactivos por un periodo prolongado. Gracias a esta información, el sistema está preparado para suspender contenedores que ya no están siendo utilizados, lo que contribuye significativamente al ahorro de recursos. De forma complementaria, la arquitectura contempla la posibilidad de reactivar un contenedor cuando un usuario acceda nuevamente a su proyecto, permitiendo una utilización flexible y eficiente de la infraestructura.

La integración con Roble facilita el seguimiento del ciclo de vida de cada proyecto. El backend actualiza el estado del contenedor, el puerto asignado, los registros de acceso y cualquier evento relevante, lo que permite un monitoreo constante y facilita la implementación de políticas de rendimiento o limpieza automática. Adicionalmente, el uso de un proxy permite en el futuro integrar mecanismos de limitación de solicitudes (rate limiting), que ofrecerían una protección adicional contra abusos o cargas excesivas sobre los contenedores individuales.

Participación de los Integrantes del Equipo

- **Andrea De La Ossa** Integración de la autenticación con Roble, flujo de inicio de sesión, conexión dashboard-backend, mejoras visuales del frontend y contribución al funcionamiento del reverse proxy.
- **Sofía Palacio** Desarrollo de dos plantillas dockerizadas, integración con el sistema de despliegue, trabajo en el frontend relacionado con plantillas y elaboración de la documentación formal del proyecto.
- **Andrés Gómez** Desarrollo completo de la plantilla React, integración con Docker, participación en el flujo de inicio de sesión y contribución a las estrategias de optimización del sistema.
- **Dayana Molina** Desarrollo y mejora del frontend, trabajo en la sección de plantillas, colaboración en el diseño del proxy y apoyo en estrategias de optimización y monitoreo.
- **José Sequeda** Construcción del reverse proxy, configuración para subdominios, apoyo en el frontend y participación en la lógica de despliegue dinámico y estrategias de optimización.