ACE HOSTING

Servicio de Hosting Seguro, Rápido y Automatizado

Proyecto de Desarrollo

CFGS Administración de Sistemas Informáticos y en Red

Proyecto realizado por Cristian Cano Andrade y Albert Rodriguez Alvarez

Tutor Proyecto: Jordi Mimó Gómez

Domingo 17 de Mayo de 2026

ÍNDICE

ÍNDICE....2
Licencia....3
Resumen....3
Abstract....3
1. Presentación del proyecto....4
1.1 Introducción....4
1.2 Contexto....4
1.3 Justificación....5
1.4 Objetivos....6
2. Estrategia y planificación....7
2.1 Estrategia de desarrollo y viabilidad....7
2.2 Metodología de trabajo....8
2.3 Planificación....9
3. Análisis....10
3.1 Casos de uso....10
3.2 Requisitos funcionales....13
3.3 Requisitos no funcionales....14
3.4 Análisis de alternativas tecnológicas....14
4. Diseño....16
4.1 Arquitectura del sistema....16
4.2 Modelo de datos....18
4.3 Diseño de interfaz....20
5. Desarrollo....26
5.1 Estructura del proyecto....26
5.2 Implementación de funcionalidades....27
5.2.1 Autenticación y registro de clientes....27
5.2.2 Catálogo de servicios y contratación con Stripe....27
5.2.3 Despliegue automatizado de contenedores por cliente....28
5.2.4 Clúster de alta disponibilidad (DRBD + Pacemaker)....29
5.2.5 Sistema de monitorización (Prometheus + Grafana + Loki)....30
5.2.6 Bastión SSH (Warpgate)....30
5.2.7 Servicio de correo electrónico....31
5.2.8 Sistema de tickets de soporte....31
5.2.9 Panel de administración....31
5.3 Pruebas....32
6. Conclusiones....33
6.1 Conclusiones generales....33
6.2 Consecución de objetivos....34
6.3 Valoración de la metodología y la planificación....35
6.4 Visión de futuro....35
7. Glosario....36
8. Bibliografía....38
9. Anexos....39
Anexo A — Manual de instalación y despliegue....39
Anexo B — Manual de usuario....39
Anexo C — Documentación de la API de Webhooks de Stripe....40
Anexo D — Documentación progresiva del desarrollo del proyecto....41
Anexo E — Declaración de uso de inteligencia artificial....41

Licencia

Resumen

El sector del alojamiento web exige soluciones que combinen automatización, alta disponibilidad y un control centralizado de los servicios contratados por los clientes. En este proyecto se ha desarrollado AceHosting, un panel de control web que permite gestionar de forma automatizada servicios de hosting a través de una infraestructura de alta disponibilidad. El sistema se ha construido sobre Laravel 12 como framework backend, PostgreSQL como sistema gestor de base de datos y Docker Compose para la contenedorización de los servicios. La infraestructura se apoya en un clúster de dos nodos con DRBD, Pacemaker y Corosync que garantiza la replicación de almacenamiento y la conmutación automática ante fallos. El panel de control integra pasarela de pago Stripe para la contratación de servicios, despliegue automatizado de contenedores por cliente, sistema de tickets de soporte, servidor de correo electrónico con Roundcube, monitorización mediante Prometheus, Grafana y Loki, y un bastión SSH con Warpgate. El resultado es una plataforma funcional que demuestra la viabilidad técnica de unificar en un solo sistema las capacidades de un proveedor de hosting profesional con garantías de alta disponibilidad.

Palabras clave: hosting, alta disponibilidad, Laravel, DRBD, Pacemaker, Docker, Stripe, monitorización, despliegue automatizado

---

Abstract

The web hosting sector demands solutions that combine automation, high availability, and centralised control over customer services. This project has developed AceHosting, a web-based control panel that enables automated management of hosting services through a high-availability infrastructure. The system is built on Laravel 12 as the backend framework, PostgreSQL as the database management system, and Docker Compose for service containerisation. The infrastructure relies on a two-node cluster with DRBD, Pacemaker, and Corosync to ensure storage replication and automatic failover. The control panel integrates the Stripe payment gateway for service subscriptions, automated per-client container deployment, a support ticket system, an email server with Roundcube, monitoring through Prometheus, Grafana, and Loki, and an SSH bastion with Warpgate. The result is a functional platform that demonstrates the technical feasibility of unifying in a single system the capabilities of a professional hosting provider with high-availability guarantees.

Keywords: hosting, high availability, Laravel, DRBD, Pacemaker, Docker, Stripe, monitoring, automated deployment

---

1. Presentación del proyecto

1.1 Introducción

El alojamiento web es un servicio fundamental en la infraestructura de internet actual. Cualquier sitio web necesita un espacio en un servidor donde residir, y los proveedores de hosting ofrecen precisamente ese servicio, complementado con herramientas que permiten a sus clientes gestionar sus contratos, dominios, correo electrónico y otros recursos asociados.

En este proyecto se propone el desarrollo de AceHosting, un panel de control web orientado a proveedores de hosting que deseen ofrecer servicios de alojamiento a sus clientes con garantías de alta disponibilidad. El sistema permite gestionar el ciclo de vida completo de un cliente de hosting: desde su registro y contratación de servicios hasta el despliegue automatizado de sus contenedores, la monitorización de sus recursos y la atención al cliente mediante un sistema de tickets.

La plataforma se ha desarrollado utilizando Laravel 12 como framework backend, PostgreSQL como base de datos y Docker para la contenedorización de servicios, tanto de la propia plataforma como de los entornos de cada cliente. La infraestructura subyacente se apoya en un clúster de alta disponibilidad formado por dos servidores físicos con DRBD, Pacemaker y Corosync, que garantizan la replicación del almacenamiento y la continuidad del servicio ante fallos de hardware.

A lo largo de esta memoria se describe el planteamiento del proyecto, el análisis de las necesidades detectadas, las decisiones tecnológicas adoptadas, el diseño de la arquitectura, el desarrollo de las funcionalidades principales y las conclusiones obtenidas tras su implementación.

1.2 Contexto

El sector del alojamiento web es uno de los pilares de la economía digital. Miles de empresas y particulares necesitan un espacio en internet donde publicar sus sitios web, alojar sus aplicaciones o gestionar su correo electrónico. Los proveedores de hosting compiten en aspectos como el precio, la facilidad de uso, el rendimiento y, cada vez más, la disponibilidad del servicio.

En la actualidad, las soluciones más extendidas para la gestión de hosting son paneles de control comerciales como cPanel, Plesk o DirectAdmin. Estas herramientas ofrecen una amplia gama de funcionalidades, pero presentan algunas limitaciones importantes. Por un lado, son productos cerrados y de pago, cuyo coste puede ser elevado para pequeños proveedores. Por otro lado, su arquitectura monolítica dificulta la personalización y la integración con infraestructuras de alta disponibilidad.

En el lado de las soluciones open source, existen alternativas como VestaCP, HestiaCP o ISPConfig, que ofrecen funcionalidades básicas de gestión de hosting. Sin embargo, estas herramientas suelen carecer de soporte nativo para despliegues automatizados basados en contenedores o para infraestructuras de alta disponibilidad con conmutación automática ante fallos.

La alta disponibilidad es un factor crítico en el sector del hosting. Un proveedor que no garantice la continuidad de sus servicios corre el riesgo de perder clientes y reputación. Tradicionalmente, garantizar la alta disponibilidad requería inversiones significativas en hardware redundante y configuraciones complejas. En los últimos años, la madurez de tecnologías como DRBD (Distributed Replicated Block Device) y Pacemaker ha facilitado la construcción de clústeres de alta disponibilidad sobre hardware estándar.

En este contexto, resulta interesante explorar el desarrollo de una plataforma de hosting propia que integre, desde su diseño, la automatización del despliegue de servicios, un panel de control funcional para clientes y administradores, y una infraestructura de alta disponibilidad como base subyacente.

1.3 Justificación

Teniendo en cuenta las limitaciones de las soluciones existentes en el mercado del hosting, resulta pertinente plantear el desarrollo de un sistema propio que aborde de forma integral las necesidades de un proveedor de servicios de alojamiento web.

El principal problema que se quiere resolver es la falta de una solución de código abierto que combine un panel de control funcional con una infraestructura de alta disponibilidad basada en clúster. Las soluciones comerciales existentes son costosas y cerradas, mientras que las alternativas open source carecen de integración nativa con tecnologías de alta disponibilidad y despliegue automatizado basado en contenedores.

El valor que aporta este proyecto es doble. Por un lado, se obtiene una plataforma funcional que permite a un proveedor de hosting gestionar sus clientes, servicios, pagos y monitorización desde un único panel de control. Por otro lado, la plataforma está diseñada desde el inicio sobre una infraestructura de alta disponibilidad, lo que garantiza la continuidad del servicio incluso ante fallos de hardware.

Además, este proyecto permite aplicar de forma práctica los conocimientos adquiridos a lo largo del ciclo formativo de ASIR en áreas fundamentales como la administración de sistemas operativos (Linux), la gestión de bases de datos (PostgreSQL), la administración de redes (Netplan, DNS con Bind9, VLANs), la virtualización y contenedores (Docker), la implantación de aplicaciones web (Laravel, Nginx), la seguridad perimetral (Warpgate, certificados SSL) y la monitorización de infraestructuras (Prometheus, Grafana, Loki).

Por estos motivos, el desarrollo de AceHosting constituye un proyecto adecuado tanto desde el punto de vista técnico como práctico, ya que aborda una necesidad real del sector del hosting y permite consolidar las competencias propias del ciclo formativo.

1.4 Objetivos

El proyecto se centra en el desarrollo de una plataforma de hosting que permita a un proveedor gestionar sus servicios de forma automatizada, ofreciendo a sus clientes un panel de control funcional y garantizando la continuidad del servicio mediante una infraestructura de alta disponibilidad.

Objetivo general

Desarrollar un panel de control web para la gestión automatizada de servicios de hosting, desplegado sobre una infraestructura de alta disponibilidad basada en un clúster de dos nodos con DRBD y Pacemaker.

Objetivos específicos

- Implementar un sistema de autenticación y registro que permita a los clientes acceder al panel de control y gestionar sus datos personales.
- Diseñar un catálogo de servicios contratables a través de una pasarela de pago integrada (Stripe), con modalidad de suscripción mensual o anual.
- Desarrollar un sistema de despliegue automatizado que, tras la contratación de un servicio, cree y configure los contenedores Docker del cliente con su propio Nginx, base de datos y servicios de correo.
- Construir un clúster de alta disponibilidad con DRBD, Pacemaker y Corosync que garantice la replicación del almacenamiento y la conmutación automática ante fallos de un nodo.
- Integrar un sistema de monitorización con Prometheus, Grafana y Loki para la supervisión de los recursos del servidor y de los contenedores de los clientes.
- Implementar un sistema de tickets de soporte que permita a los clientes comunicarse con el administrador y realizar un seguimiento de sus incidencias.
- Configurar un servidor de correo electrónico completo con gestión de cuentas por cliente y acceso web a través de Roundcube.
- Establecer un bastión SSH con Warpgate que centralice y asegure el acceso a los servidores de la infraestructura.

Estos objetivos permitirán obtener una plataforma funcional que integre todas las capacidades necesarias para operar como un proveedor de hosting profesional, con especial énfasis en la automatización, la alta disponibilidad y la monitorización del sistema.

---

2. Estrategia y planificación

2.1 Estrategia de desarrollo y viabilidad

El proyecto se plantea como el desarrollo completo de una plataforma de hosting desde cero, sin reutilizar código existente de otros paneles de control. Este enfoque permite adaptar cada componente a las necesidades específicas del proyecto y tomar decisiones técnicas libres de condicionantes previos.

Punto de partida

El proyecto parte sin ningún software previo que pueda servir como base. Sin embargo, se dispone del diseño de base de datos elaborado durante la fase de estudio de PostgreSQL, que constituye el punto de partida para el modelo de datos del sistema. También se cuenta con prototipos de interfaz desarrollados durante la fase de diseño web, que servirán como referencia para las vistas de la aplicación Laravel.

La infraestructura sobre la que se desplegará la plataforma está formada por dos servidores físicos con Ubuntu Server 24.04, conectados a través de una red interna dedicada para DRBD y Corosync. Esta infraestructura se ha configurado siguiendo las guías elaboradas durante el proyecto.

Estrategia de desarrollo

Se plantea un desarrollo por fases, organizado en grandes bloques de trabajo que abarcan tanto la configuración de la infraestructura como el desarrollo del software.

En primer lugar se ha abordado la configuración de la infraestructura base: instalación de los servidores, configuración de la red, despliegue del clúster DRBD+Pacemaker y puesta en marcha de los servicios base (Docker, Nginx, Bind9). Esta fase inicial es crítica porque todo el desarrollo posterior depende de una infraestructura estable.

A continuación se ha desarrollado el núcleo del panel de control, comenzando por los elementos fundamentales: autenticación de usuarios, gestión del catálogo de servicios e integración con la pasarela de pago Stripe. Una vez que estos componentes básicos han estado operativos, se han ido añadiendo funcionalidades adicionales como el despliegue automatizado de contenedores, el sistema de tickets, la gestión de correo electrónico y la monitorización.

Las funcionalidades se han desarrollado de forma incremental, validando cada bloque antes de pasar al siguiente. Esto ha permitido mantener una versión funcional del sistema en todo momento y detectar errores de forma temprana.

Viabilidad del proyecto

El proyecto se considera viable dentro del tiempo disponible por varias razones. En primer lugar, las funcionalidades están bien delimitadas y se ha definido un orden de desarrollo claro que prioriza los componentes esenciales. En segundo lugar, se dispone de documentación exhaustiva y bien estructurada sobre cada tecnología utilizada. En tercer lugar, la experiencia previa adquirida durante el ciclo formativo en cada una de las áreas implicadas reduce la curva de aprendizaje.

Las principales dificultades previstas se concentran en la configuración del clúster de alta disponibilidad, especialmente en la integración de Pacemaker con los servicios Docker y en la gestión de la conmutación automática de los contenedores de clientes. Para mitigar este riesgo, se ha priorizado la configuración y validación del clúster en las primeras fases del proyecto, dedicando el tiempo necesario para garantizar su correcto funcionamiento.

2.2 Metodología de trabajo

Para el desarrollo del proyecto se ha seguido una metodología de trabajo progresiva, centrada en la implementación de funcionalidades de forma incremental y en la validación continua del sistema.

Enfoque de trabajo

El proyecto se ha dividido en grandes bloques de trabajo que se corresponden con las fases de la planificación. Dentro de cada bloque, las tareas se han organizado de forma que las funcionalidades más críticas se han desarrollado primero, permitiendo disponer de una versión funcional del sistema desde etapas tempranas.

Se ha priorizado en todo momento la estabilidad de la infraestructura base antes de añadir nuevas funcionalidades, ya que cualquier problema en los cimientos del sistema habría comprometido el desarrollo del resto del proyecto.

Organización del trabajo

El trabajo se ha organizado en tareas concretas asignadas a cada miembro del equipo en función de su área de especialización. Las tareas se han gestionado mediante un tablero Trello donde se registraban las funcionalidades pendientes, en desarrollo y completadas, lo que ha permitido hacer un seguimiento visual del progreso del proyecto.

El código fuente se ha gestionado mediante un repositorio en GitLab, lo que ha permitido mantener un control de versiones, registrar el historial de cambios y trabajar de forma coordinada sin conflictos.

Herramientas de seguimiento

- **ClickUp:** Herramienta para el seguimiento de las tareas, con funciones como diagrama de Gantt o Kanban.

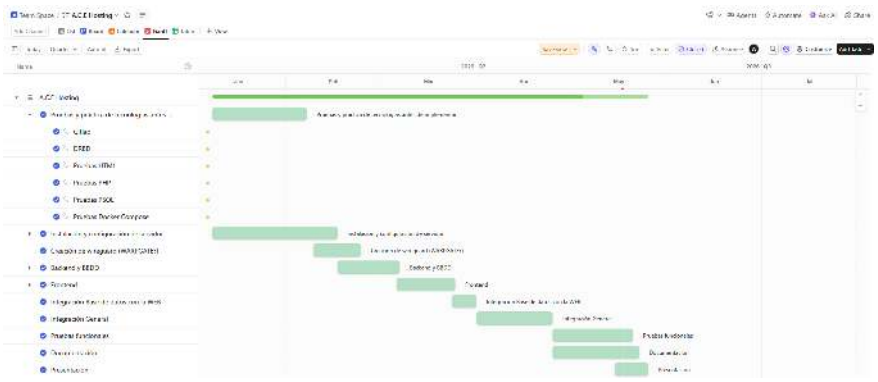


Figura 1: Diagrama de Gantt en ClickUp con seguimiento de tareas y subtareas.

- **GitLab:** para el control de versiones y la gestión del código fuente. Enlace
- **Git:** como sistema de control de versiones distribuido (CLI).

2.3 Planificación

El proyecto se ha organizado en seis fases que han permitido avanzar de forma progresiva desde la definición inicial hasta una plataforma funcional completa.

Fase 1 · Análisis y diseño inicial

En esta fase se ha definido el alcance del proyecto, se ha elaborado el diseño de la base de datos y se ha realizado el diseño básico de la interfaz de usuario. También se ha llevado a cabo el estudio de alternativas tecnológicas para cada componente del sistema. El resultado de esta fase ha sido una definición clara del proyecto y una base sólida para el desarrollo.

Fase 2 · Configuración de la infraestructura

Se ha instalado y configurado la infraestructura base del proyecto: los servidores físicos con Ubuntu Server 24.04, la configuración de red con Netplan, el clúster de alta disponibilidad con DRBD y Pacemaker, Docker y Docker Compose, el servidor DNS Bind9, el bastión SSH Warpgate y los certificados SSL. El objetivo ha sido disponer de una infraestructura estable y lista para alojar la aplicación.

Fase 3 · Desarrollo del panel de control

Se ha implementado el núcleo del panel de control con Laravel: sistema de autenticación y registro de clientes, catálogo de servicios, integración con la pasarela de pago Stripe y panel de administración básico. El objetivo ha sido disponer de una primera versión funcional de la aplicación web.

Fase 4 · Funcionalidades avanzadas

Se han añadido las funcionalidades más complejas: sistema de despliegue automatizado de contenedores por cliente, sistema de tickets de soporte, gestión de cuentas de correo electrónico e integración con Roundcube. También se ha implementado la monitorización con Prometheus, Grafana y Loki, y se ha configurado el dashboard de administración con las métricas del sistema.

Fase 5 · Pruebas e integración

Se ha verificado el funcionamiento de todas las funcionalidades del sistema, con especial atención a las pruebas de integración entre componentes y a las pruebas de alta disponibilidad (conmutación por fallo del nodo primario del clúster). Se han corregido los errores detectados durante las pruebas.

Fase 6 · Documentación y mejoras finales

Se han realizado los ajustes finales, mejoras visuales en la interfaz y la elaboración de la documentación del proyecto, incluyendo esta memoria, el manual de instalación y el manual de usuario.

Distribución temporal

Fase	Duración estimada
Fase 1 · Análisis y diseño inicial	4 semanas
Fase 2 · Configuración de la infraestructura	6 semanas
Fase 3 · Desarrollo del panel de control	8 semanas
Fase 4 · Funcionalidades avanzadas	6 semanas
Fase 5 · Pruebas e integración	4 semanas
Fase 6 · Documentación y mejoras finales	2 semanas

Esta planificación ha sido orientativa y se ha ajustado durante el desarrollo en función del progreso real y de las dificultades encontradas, especialmente en la configuración del clúster de alta disponibilidad que requirió más tiempo del inicialmente estimado.

---

3. Análisis

3.1 Casos de uso

El sistema tiene tres tipos de actores principales: el **cliente**, que contrata servicios y gestiona sus recursos; el **administrador**, que supervisa el sistema y gestiona a los clientes; y el **sistema**, que ejecuta procesos automatizados como el despliegue de contenedores o el envío de webhooks. A continuación se documentan los casos de uso más representativos del sistema.

CU-01 · Registrarse como cliente

Actor: Cliente.

Descripción: Un usuario no registrado crea una cuenta en la plataforma para poder contratar servicios.

Flujo principal:

1. El usuario accede a la página de registro.
2. Introduce sus datos personales: nombre, correo electrónico y contraseña.
3. El sistema valida que el correo no esté ya registrado.
4. El sistema crea la cuenta y redirige al usuario al panel de cliente.

Flujo alternativo:

- Si el correo ya está registrado, el sistema muestra un mensaje de error y solicita otro.
- Si los campos obligatorios están vacíos, el sistema muestra un mensaje de error.

Resultado: El cliente queda registrado y puede acceder al panel de control.

CU-02 · Contratar un servicio

Actor: Cliente.

Descripción: El cliente selecciona un servicio del catálogo, elige la modalidad de pago y completa la contratación a través de la pasarela de pago Stripe.

Flujo principal:

1. El cliente accede al catálogo de servicios desde el panel de control.
2. Selecciona el servicio deseado (hosting web, VPS, dominio, etc.).
3. Elige la modalidad de pago (mensual o anual).
4. El sistema redirige al cliente a la pasarela de pago de Stripe Checkout.
5. El cliente introduce sus datos de pago y completa la transacción.
6. Stripe envía un webhook al sistema confirmando el pago.
7. El sistema crea el contrato del cliente con estado "pagado" y encola el despliegue.

Flujo alternativo:

- Si el pago es rechazado, Stripe informa al cliente y el sistema no crea el contrato.
- Si el webhook de Stripe no se recibe correctamente, el sistema reintenta el procesamiento.

Resultado: El servicio queda contratado y pendiente de despliegue.

CU-03 · Desplegar el entorno del cliente (sistema)

Actor: Sistema (automático).

Descripción: Tras la contratación de un servicio, el sistema despliega automáticamente los contenedores Docker del cliente.

Flujo principal:

1. El sistema detecta un contrato en estado "pagado".
2. El administrador (o el cliente, según configuración) sube un archivo ZIP con el contenido del sitio web.
3. El sistema crea una carpeta para el cliente en el almacenamiento compartido (DRBD).
4. El sistema genera un archivo docker-compose.yml personalizado para el cliente.
5. El sistema genera la configuración de Nginx para el subdominio del cliente.
6. El sistema ejecuta `docker compose up -d` en el directorio del cliente.
7. El sistema actualiza el estado del contrato a "activo".

Flujo alternativo:

- Si el despliegue falla, el sistema registra el error y notifica al administrador.
- Si el almacenamiento DRBD no está disponible, el despliegue se pospone hasta que el clúster esté operativo.

Resultado: El cliente dispone de su entorno de hosting en funcionamiento.

CU-04 · Abrir un ticket de soporte

Actor: Cliente.

Descripción: El cliente abre un ticket de soporte para comunicar una incidencia o realizar una consulta al administrador.

Flujo principal:

1. El cliente accede a la sección de tickets del panel de control.
2. Pulsa el botón de nuevo ticket.
3. Introduce el asunto, la prioridad y el mensaje.
4. Opcionalmente, asocia el ticket a un contrato concreto.
5. El sistema registra el ticket con estado "abierto" y lo muestra en el panel del cliente y del administrador.
6. El administrador responde al ticket; el cliente recibe la respuesta en el panel.

Flujo alternativo:

- Si el cliente intenta enviar un ticket sin asunto, el sistema muestra un mensaje de error.

Resultado: El ticket queda registrado y visible para el cliente y el administrador.

CU-05 · Supervisar el sistema (administrador)

Actor: Administrador.

Descripción: El administrador accede al panel de control para supervisar el estado del sistema, consultar métricas y gestionar clientes.

Flujo principal:

1. El administrador inicia sesión con su cuenta.
2. Accede al panel de administración.
3. Visualiza los indicadores del sistema: número de clientes activos, servicios contratados, uso de recursos.
4. Consulta los dashboards de Grafana incrustados con métricas detalladas de CPU, RAM, disco y red.
5. Puede cambiar el estado de un cliente (activo, suspendido, inactivo).

Resultado: El administrador obtiene una visión general del estado del sistema y puede tomar decisiones informadas.

3.2 Requisitos funcionales

La plataforma debe cubrir las necesidades de gestión de un proveedor de hosting. A continuación se presentan los requisitos funcionales identificados:

ID	Requisito	Prioridad
RF01	El sistema permitirá a los clientes registrarse con nombre, correo y contraseña	Alta
RF02	El sistema permitirá iniciar y cerrar sesión	Alta
RF03	El sistema mostrará un catálogo de servicios contratables	Alta
RF04	El sistema permitirá contratar servicios a través de Stripe Checkout	Alta
RF05	El sistema procesará webhooks de Stripe para confirmar pagos	Alta
RF06	El sistema gestionará suscripciones mediante Stripe Billing Portal	Alta
RF07	El sistema desplegará automáticamente contenedores Docker por cliente	Alta
RF08	El sistema generará configuración dinámica de Nginx por subdominio	Alta
RF09	El sistema permitirá gestionar tickets de soporte (crear, responder, cerrar)	Alta
RF10	El administrador podrá consultar y modificar el estado de los clientes	Alta
RF11	El sistema permitirá crear y eliminar cuentas de correo electrónico por cliente	Media
RF12	El sistema integrará un cliente web de correo (Roundcube)	Media
RF13	El sistema mostrará métricas de monitorización en el panel de administración	Media
RF14	El sistema registrará logs centralizados con Loki	Media
RF15	El administrador podrá cambiar la prioridad y el estado de los tickets	Media
RF16	El sistema garantizará la alta disponibilidad del almacenamiento con DRBD	Alta
RF17	El sistema conmutará automáticamente los servicios ante fallo del nodo primario	Alta
RF18	El sistema permitirá el acceso SSH seguro mediante Warpgate	Alta
RF19	El sistema gestionará la resolución DNS interna con Bind9	Alta
RF20	El sistema renovará automáticamente los certificados SSL locales	Media

Los requisitos de prioridad alta constituyen el núcleo funcional de la plataforma y deben estar implementados en la versión final del proyecto. Los de prioridad media se han desarrollado en función del tiempo disponible y del avance del proyecto.

3.3 Requisitos no funcionales

Además de las funcionalidades descritas, la plataforma debe cumplir una serie de condiciones de calidad que afectan a su comportamiento general.

Disponibilidad: El sistema debe garantizar un tiempo de actividad mínimo del 99,9% gracias a la arquitectura de alta disponibilidad del clúster. La conmutación por fallo del nodo primario al secundario debe producirse en menos de 120 segundos.

Rendimiento: La aplicación web debe responder a las acciones del usuario en menos de dos segundos en condiciones normales de uso. Los contenedores de los clientes deben desplegarse en menos de cinco minutos desde la confirmación del pago.

Seguridad: Las contraseñas se almacenarán cifradas mediante bcrypt. El acceso a las funciones de administración estará restringido al usuario administrador. Las comunicaciones con Stripe se realizarán mediante HTTPS con verificación de firmas de webhook. El acceso SSH a los servidores se realizará exclusivamente a través del bastión Warpgate, sin exponer el servicio SSH directamente.

Usabilidad: La interfaz debe ser intuitiva y permitir a un cliente realizar las operaciones principales (registro, contratación, consulta de tickets) en pocos pasos, sin necesidad de formación previa.

Compatibilidad: La aplicación debe funcionar correctamente en los navegadores más habituales (Chrome, Firefox, Edge) y ser accesible desde dispositivos de escritorio.

Mantenibilidad: El código estará organizado siguiendo el patrón MVC de Laravel, con separación clara entre modelos, vistas y controladores, lo que facilitará la incorporación de nuevas funcionalidades en el futuro.

Monitorización: El sistema debe registrar y visualizar métricas de rendimiento de los servidores y los contenedores, permitiendo al administrador detectar anomalías de forma temprana.

3.4 Análisis de alternativas tecnológicas

Las tecnologías utilizadas en este proyecto no se han elegido de forma arbitraria, sino tras valorar distintas opciones para cada decisión relevante. A continuación se describe el proceso de análisis seguido en cada caso.

Framework backend

Para el desarrollo del servidor se valoraron Laravel (PHP), Django (Python) y Express (Node.js). Django ofrece una estructura sólida y bien documentada, pero el equipo no disponía de experiencia previa en Python. Express es ligero y flexible, pero su modelo asíncrono puede resultar complejo para un proyecto con operaciones síncronas como el despliegue de contenedores. Laravel es el framework PHP más utilizado en la actualidad, cuenta con una comunidad muy activa, una documentación exhaustiva y un ecosistema de herramientas

integradas (Eloquent ORM, sistema de colas, autenticación, migraciones de base de datos). Además, el equipo ya tenía experiencia previa con PHP.

Se eligió **Laravel 12** por su madurez, su ecosistema completo y la experiencia previa del equipo, lo que ha permitido reducir el tiempo de desarrollo y centrarse en las funcionalidades específicas del proyecto.

Sistema gestor de base de datos

Se valoraron PostgreSQL, MySQL y SQLite. SQLite es ligera pero no adecuada para entornos multiusuario concurrentes. MySQL es un sistema consolidado con gran presencia en el alojamiento web, pero PostgreSQL ofrece ventajas significativas en proyectos que requieren alta disponibilidad y consultas complejas, como soporte nativo de transacciones avanzadas, índices parciales, el tipo de datos JSONB (utilizado en este proyecto para almacenar configuraciones flexibles de servicios) y un sistema de replicación más robusto.

Se eligió **PostgreSQL 15** por su fiabilidad, su soporte de datos JSONB y su idoneidad para entornos de alta disponibilidad donde la integridad de los datos es crítica.

Motor de contenedores

Para la contenedorización de servicios se valoraron Docker Compose y Kubernetes. Kubernetes es una plataforma muy potente para la orquestación de contenedores a gran escala, pero su curva de aprendizaje es elevada y su complejidad operativa resulta excesiva para el alcance de este proyecto. Docker Compose permite definir y ejecutar aplicaciones multicontenedor de forma sencilla, con una sintaxis clara y un ecosistema maduro.

Se eligió **Docker Compose** por su simplicidad, su idoneidad para el tamaño del proyecto y la experiencia previa del equipo con esta tecnología.

Tecnología de alta disponibilidad

Para garantizar la alta disponibilidad del almacenamiento se valoraron DRBD, GlusterFS y Ceph. GlusterFS y Ceph son sistemas de almacenamiento distribuido muy potentes, pero su configuración y operación son complejas y requieren recursos adicionales. DRBD es una tecnología de replicación de bloque a nivel de kernel, más ligera y con una integración estrecha con Pacemaker/Corosync, lo que permite construir un clúster de alta disponibilidad con dos nodos de forma eficiente.

Se eligió **DRBD 8.4** combinado con **Pacemaker 2.1.6** y **Corosync 3**, por ser una solución madura, bien documentada y adecuada para un clúster de dos nodos con almacenamiento replicado.

Sistema de monitorización

Para la monitorización se valoraron Prometheus + Grafana, Zabbix y Nagios. Nagios es un clásico en la monitorización de infraestructuras, pero su configuración es compleja y su interfaz está desactualizada. Zabbix ofrece una solución todo-en-uno funcional, pero su modelo de recopilación de datos basado en agente es menos flexible. Prometheus, combinado con Grafana, ofrece un modelo de recopilación basado en extracción (pull) muy potente, un ecosistema de exportadores amplio (cAdvisor para contenedores, node_exporter para servidores) y dashboards altamente personalizables en Grafana.

Se eligió **Prometheus + Grafana + Loki** por su flexibilidad, su integración nativa con entornos Docker y la calidad de sus dashboards de visualización.

Bastión SSH

Para el acceso seguro a los servidores se valoraron Warpgate, Teleport y el SSH estándar con túneles inversos. El SSH estándar requiere exponer el puerto 22 o configurar túneles, lo que aumenta la superficie de ataque. Teleport es una solución empresarial muy completa, pero su licencia comunitaria tiene limitaciones. Warpgate es una solución ligera, de código abierto y con soporte nativo para SSH, MySQL y PostgreSQL, lo que permite centralizar todo el acceso a la infraestructura desde una única herramienta con interfaz web.

Se eligió **Warpgate** por su ligereza, su facilidad de configuración y su capacidad para actuar como bastión tanto para SSH como para conexiones a bases de datos.

Pasarela de pago

Para la integración de pagos se valoraron Stripe y PayPal. PayPal ofrece una solución ampliamente extendida, pero su API es menos flexible y su proceso de integración es más farragoso. Stripe ofrece una API moderna y bien documentada, con productos específicos para suscripciones (Stripe Checkout, Billing Portal, webhooks) que encajan perfectamente con el modelo de negocio de un proveedor de hosting.

Se eligió **Stripe** por la calidad de su API, su soporte nativo para suscripciones y la facilidad de integración con Laravel.

---

4. Diseño

4.1 Arquitectura del sistema

La plataforma AceHosting sigue una arquitectura multicapa que separa claramente los diferentes componentes del sistema. La arquitectura se organiza en torno a un clúster de alta disponibilidad de dos nodos, sobre el que se ejecutan tanto el panel de control como los servicios de infraestructura y los contenedores de los clientes.

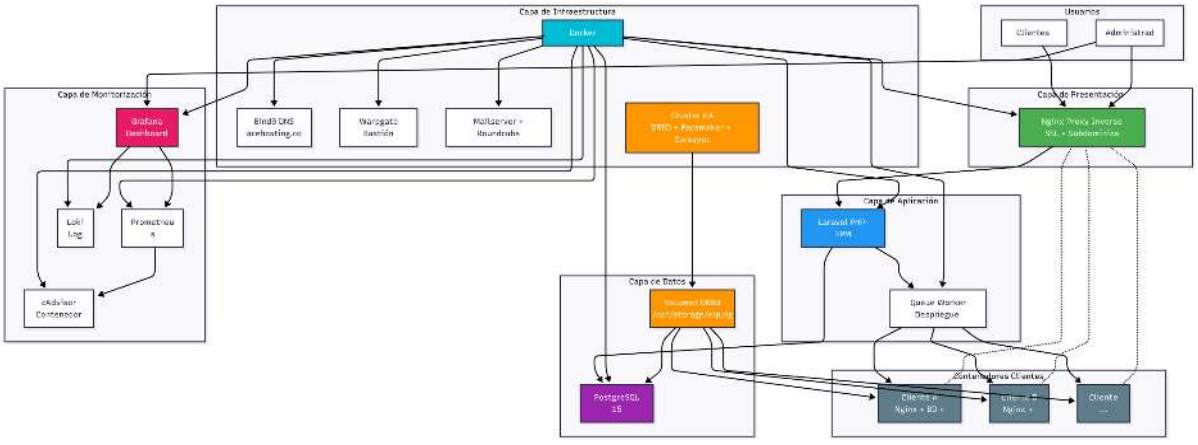


Figura 2: Diagrama de la arquitectura general del sistema AceHosting

Capa de presentación

La capa de presentación está formada por las vistas Blade del panel de control Laravel, servidas a través de Nginx. El usuario accede a la plataforma mediante un navegador web y, en función de su rol (cliente o administrador), ve un panel adaptado a sus necesidades. El administrador tiene acceso adicional a dashboards de Grafana incrustados en el panel.

Capa de aplicación

La capa de aplicación está compuesta por el panel de control Laravel, que se ejecuta en un contenedor PHP-FPM. Esta capa gestiona la lógica de negocio: autenticación, contratación de servicios, procesamiento de pagos, gestión de tickets y administración de clientes. También incluye un trabajador de colas (queue worker) que procesa tareas asíncronas como el despliegue de contenedores.

Capa de datos

La capa de datos está formada por PostgreSQL 15, que almacena toda la información del sistema: clientes, servicios, contratos, tickets, facturas y pagos. La base de datos se ejecuta en un contenedor Docker y su almacenamiento reside en el volumen DRBD replicado entre los dos nodos del clúster.

Capa de infraestructura

La capa de infraestructura incluye todos los servicios base sobre los que se apoya la plataforma:

- **Clúster DRBD + Pacemaker + Corosync:** proporciona almacenamiento replicado y alta disponibilidad con conmutación automática.
- **Docker y Docker Compose:** ejecutan todos los servicios del sistema y los contenedores de los clientes.
- **Nginx:** actúa como proxy inverso para el panel de control y genera configuración dinámica para los subdominios de los clientes.
- **Bind9:** sirve como servidor DNS interno para la resolución de nombres en el dominio acehosting.com.
- **Warpgate:** actúa como bastión SSH para el acceso seguro a los servidores.

Capa de monitorización

La capa de monitorización está formada por Prometheus (recopilación de métricas), cAdvisor (métricas de contenedores), Loki (agregación de logs) y Grafana (visualización y dashboards).

Capa de comunicaciones

Todas las comunicaciones entre los diferentes componentes del sistema se realizan a través de redes Docker internas. La red `ace-network` (172.18.0.0/16) conecta los servicios internos de la plataforma, mientras que la red `ace_proxy_network` permite la comunicación entre el proxy Nginx principal y los contenedores de los clientes.

4.2 Modelo de datos

La base de datos de AceHosting almacena la información necesaria para gestionar clientes, servicios, contratos, pagos y tickets. A continuación se describen las entidades principales y sus relaciones.

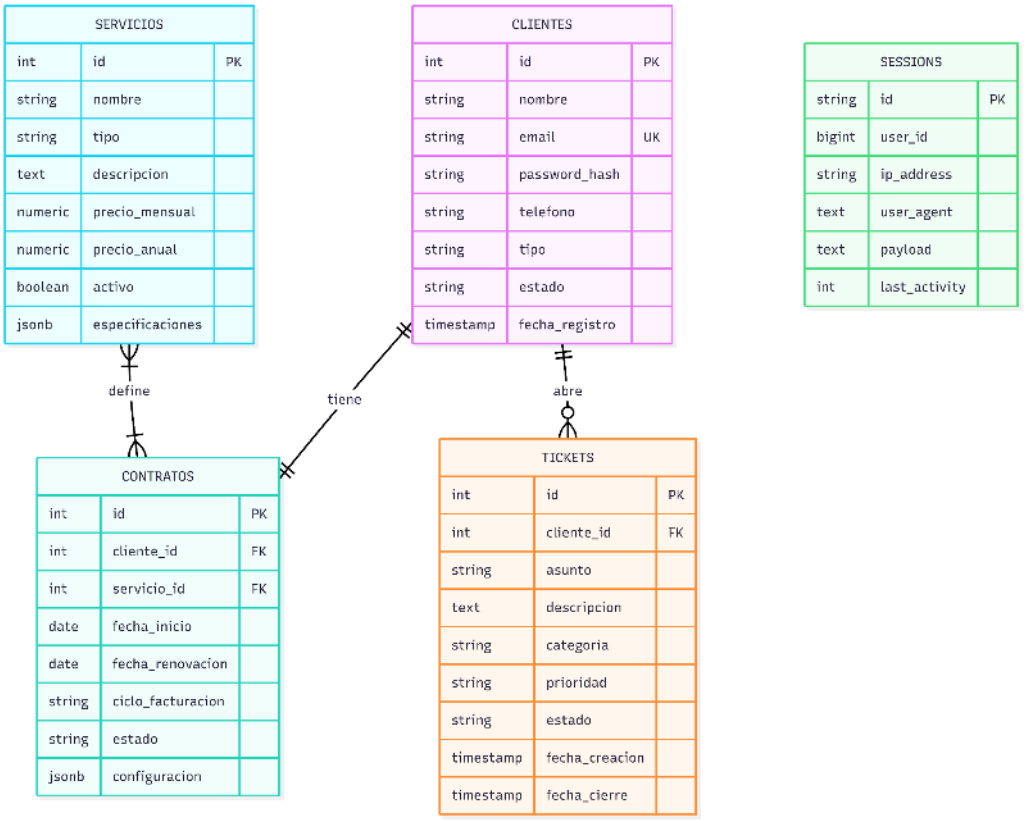


Figura 3: Diagrama entidad-relación de la base de datos del sistema

Por ejemplo, tabla clientes una línea que vaya a la tabla contratos y que se especifique la relación entre ambas, y la cardinalidad. Tabla rectangulo, relación es rombo. y asi con toda la base de datos, tienes el backup_ace.sql en la carpeta prueba_laravel.

Clientes (tabla `clientes`)

Almacena los datos de cada cliente registrado en la plataforma. Incluye nombre, correo electrónico, contraseña cifrada, estado (activo, inactivo, suspendido) y fecha de registro. Es la entidad central del sistema, ya que todas las demás entidades están vinculadas a un cliente.

Servicios (tabla `servicios`)

Define los tipos de servicio que la plataforma ofrece a sus clientes: hosting web, VPS, dominio, certificado SSL, copias de seguridad, etc. Cada servicio tiene un nombre, una descripción, un precio y unas especificaciones técnicas almacenadas en formato JSONB, lo que permite una gran flexibilidad para definir características variables. También almacena los identificadores de los precios en Stripe necesarios para la integración con la pasarela de pago (price_id mensual y price_id anual).

Contratos (tabla `contratos`)

Vincula a un cliente con un servicio contratado. Almacena el estado del contrato (pagado, configurando, creando, activo, cancelado_pendiente, cancelado), la configuración específica del contrato en formato JSONB y el identificador de la suscripción en Stripe. El estado del contrato sigue una máquina de estados definida: desde "pagado" tras confirmarse el pago, pasando por "configurando" y "creando" durante el despliegue, hasta "activo" cuando el entorno del cliente está operativo. La cancelación puede ser "cancelado_pendiente" (hasta fin del período facturado) y finalmente "cancelado".

Tickets (tabla `tickets`)

Almacena las solicitudes de soporte realizadas por los clientes. Cada ticket tiene un asunto, un estado (abierto, en curso, resuelto, cerrado), una prioridad (baja, media, alta, crítica) y puede estar asociado opcionalmente a un contrato concreto. Los mensajes de cada ticket se almacenan en una tabla relacionada `mensajes_ticket`.

Facturas y pagos (tablas `facturas`, `factura_contratos`, `pagos`)

Gestionan la información económica de la plataforma. Las facturas se generan automáticamente al confirmarse un pago y se vinculan con los contratos correspondientes. Los pagos registran las transacciones realizadas a través de Stripe.

Métricas y backups (tablas `metricas`, `backups`)

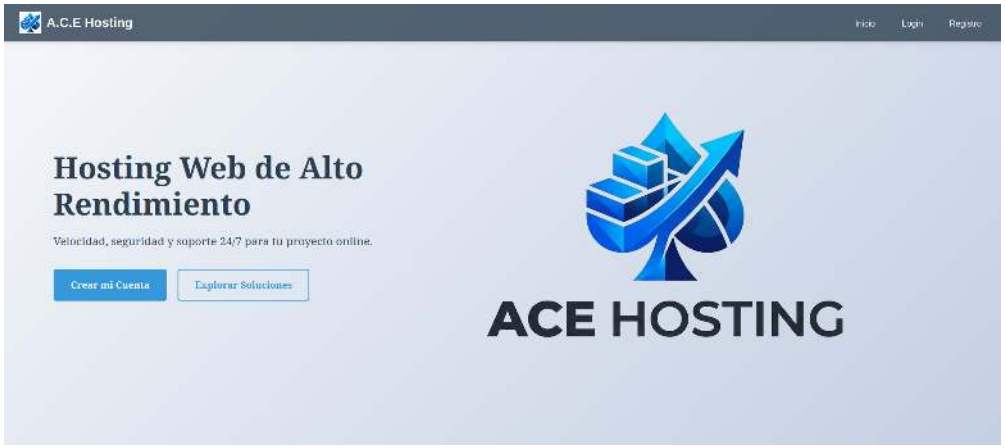
Almacenan información histórica para la monitorización y la gestión de copias de seguridad. Las métricas registran el consumo de recursos (CPU, RAM, disco, red) de los contenedores de los clientes. Los backups registran las copias de seguridad realizadas de los datos de los clientes.

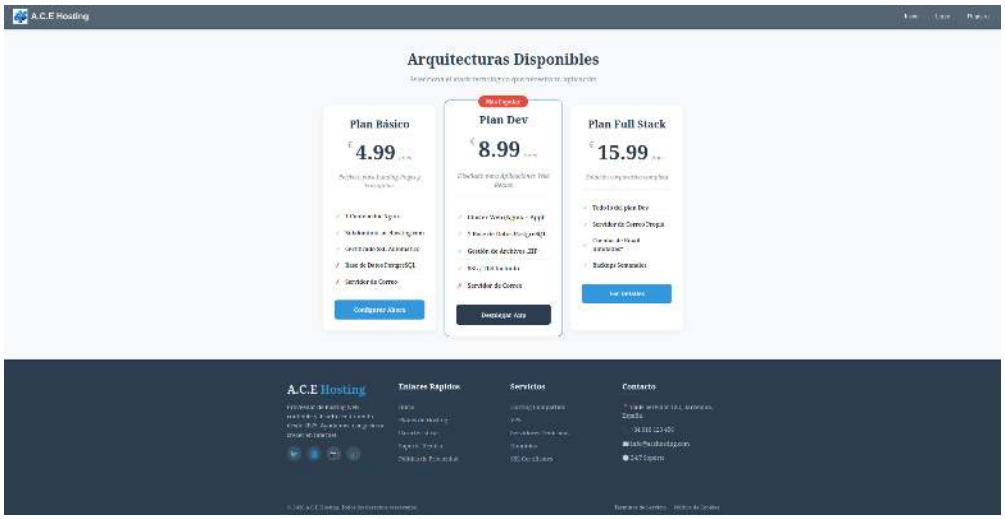
Las relaciones entre tablas son las siguientes:

- Un cliente puede tener un solo contrato, cada contrato pertenece a un único cliente.
- Un contrato se asocia a un único servicio, aunque un servicio puede estar contratado por múltiples clientes.
- Un cliente puede abrir múltiples tickets, y cada ticket pertenece a un solo cliente.
- Una factura puede incluir múltiples contratos, y un contrato puede aparecer en múltiples facturas. ******
- Un pago se corresponde con una factura.*****
- Las métricas y los backups se asocian a contenedores, que a su vez se vinculan a contratos.

4.3 Diseño de interfaz

La interfaz del panel de control AceHosting está diseñada para ser funcional y clara, priorizando que el usuario pueda realizar las operaciones principales con el menor número de pasos posible. Se ha utilizado Bootstrap para garantizar que la interfaz sea accesible desde distintos dispositivos.



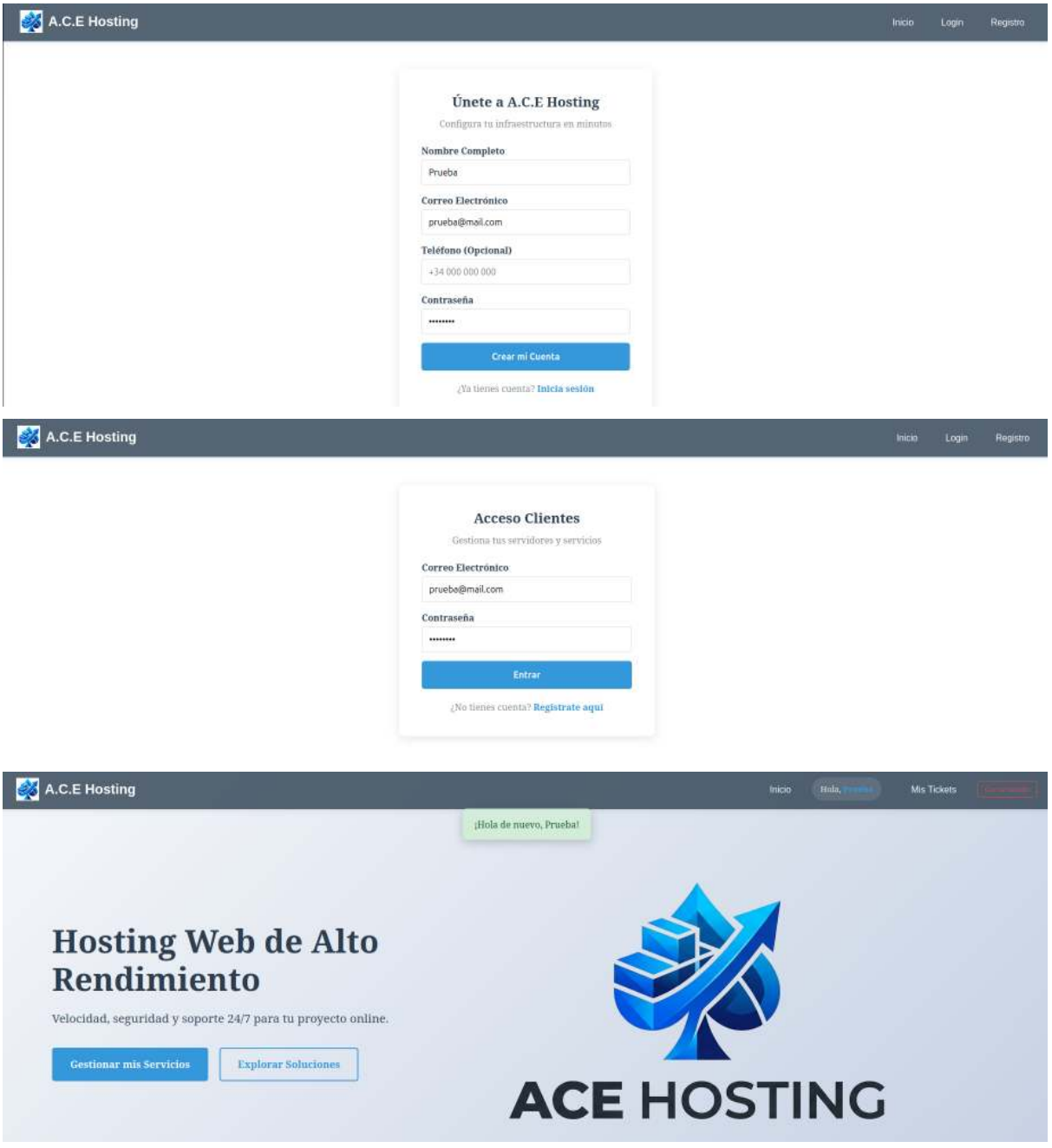


Figuras 4,5 y 6: Página de inicio de AceHosting

Las pantallas principales del sistema son las siguientes:

Página de inicio: Muestra una presentación de la plataforma con los servicios destacados, los planes de precios y una llamada a la acción para registrarse o iniciar sesión. Es la puerta de entrada a la plataforma para los visitantes no registrados.

Pantalla de registro y login: Formularios para crear una nueva cuenta o acceder a una existente. El registro solicita nombre, correo electrónico y contraseña. Tras el registro, el cliente es redirigido al panel de control.



Figuras 7,8 y 9: Pantallas de registro e inicio de sesión

Panel de cliente (catálogo de servicios): Es la pantalla principal del cliente una vez ha iniciado sesión. Muestra el catálogo de servicios disponibles con sus precios y una descripción de cada uno. Desde aquí el cliente puede contratar nuevos servicios. Si el cliente ya tiene servicios contratados, la vista cambia para mostrar sus contratos activos.

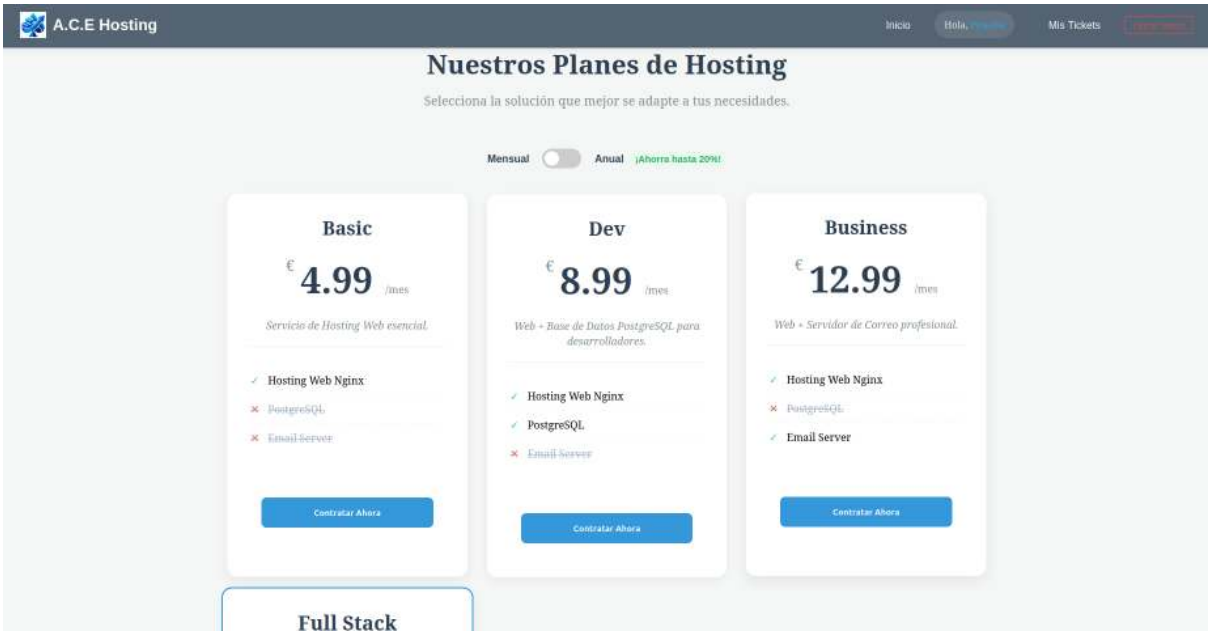


Figura 10: Catálogo de servicios para clientes

Pantalla de checkout: Muestra el resumen del servicio que se va a contratar tras redirigir al cliente a la pasarela de pago de Stripe Checkout. Tras el pago, el cliente es redirigido de vuelta al panel donde el cliente verá el panel de.

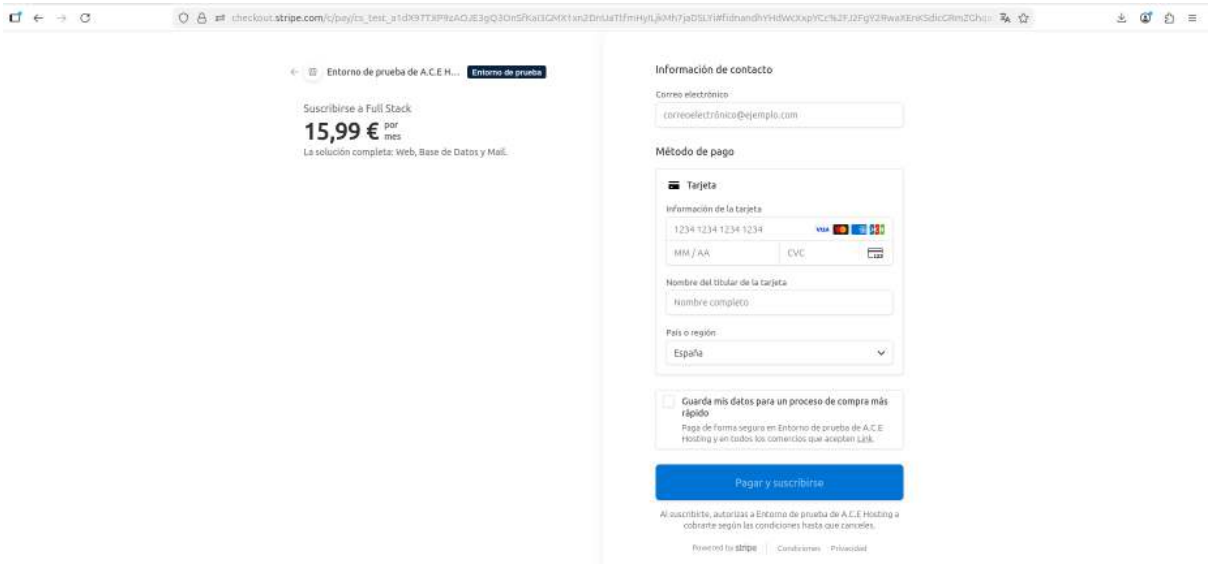


Figura 11: Pantalla de pasarela de pago de Stripe

Panel del cliente (confirmación): pantalla donde se configuran los datos sobre los servicios contratados como el nombre del dominio, el usuario principal del mail y además se añaden los archivos necesarios para crear la web y la base de datos si es que están contratados.

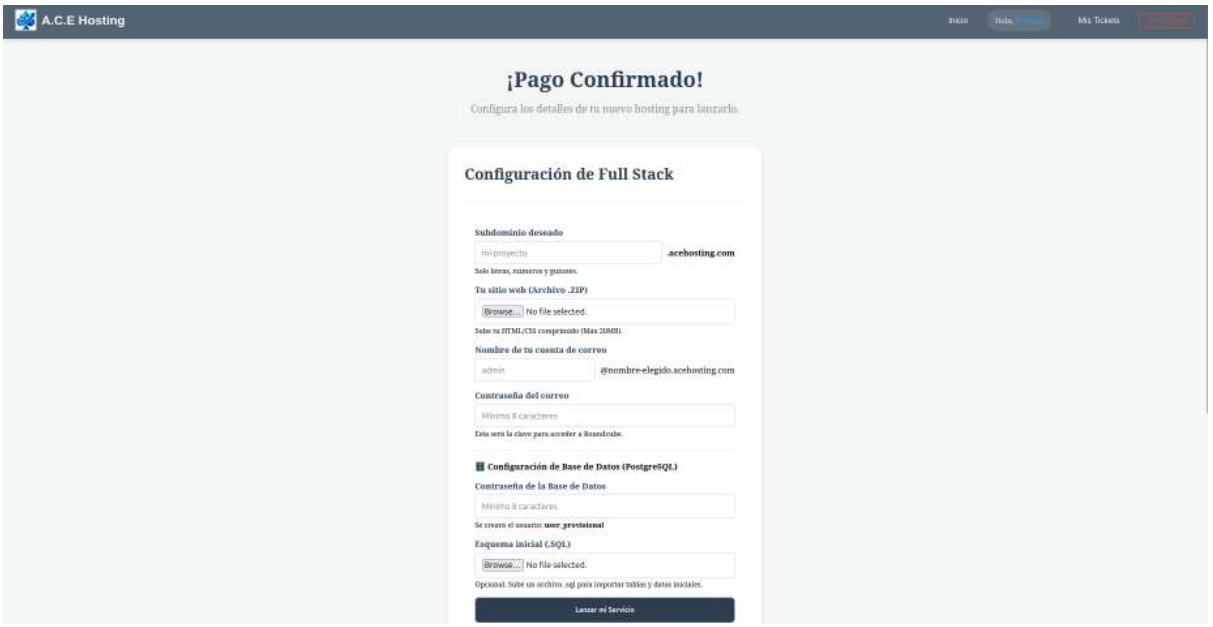


Figura 12: Pantalla de configuración de servicios

Panel del cliente (levantamiento de servicios): Una vez enviado el formulario de la pantalla anterior, accedemos brevemente a una nueva pantalla que nos indica que se está levantando el entorno del cliente, esta pantalla solo aparece durante unos segundos.

Figura 13: Pantalla de espera para creación del entorno del cliente

Panel del cliente (Servicios activos): Esta pantalla se muestra una vez se han creado los contenedores de cada servicio, y mostrará el estado de éstos, así como botones para acceder mediante la web a cada servicio, y además da la posibilidad de crear y eliminar cuentas de correo.

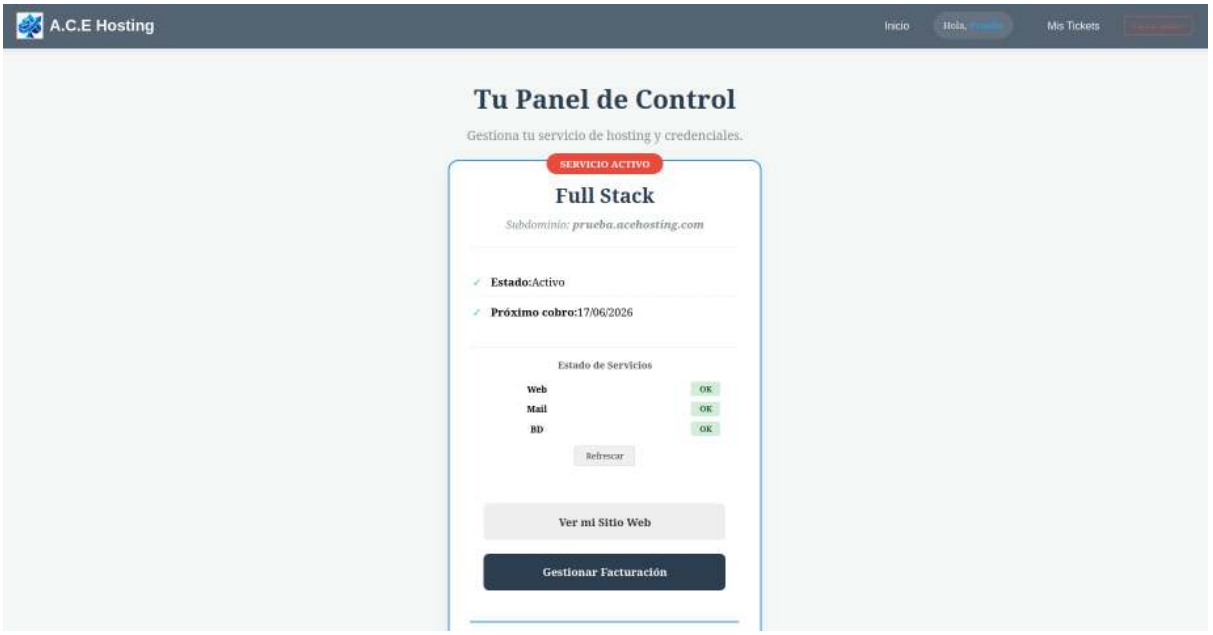


Figura 14: Pantalla de servicios activos del cliente.

Panel de administración: Accesible únicamente para el administrador. Muestra indicadores del sistema: número de clientes activos, servicios contratados, ingresos y estado de los servidores. Incluye dashboards de Grafana incrustados que muestran métricas detalladas de CPU, RAM, disco y red de los servidores y los contenedores.

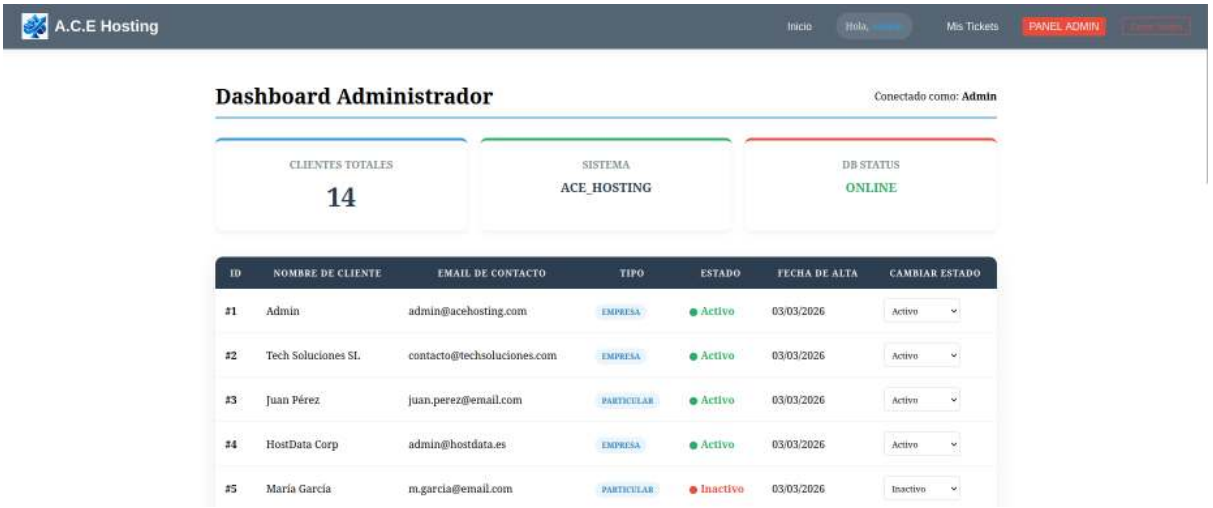


Figura 15: Panel de administración con dashboards de Grafana

Sección de tickets: Permite al cliente crear nuevos tickets de soporte, consultar el estado de los existentes y ver el historial de respuestas del administrador. El administrador puede cambiar la prioridad y el estado de los tickets desde su panel.

Figura 16: Sistema de tickets de soporte

La navegación entre pantallas sigue una estructura clara: desde el panel principal se accede a las distintas secciones a través de un menú superior que se adapta al rol del usuario autenticado. Los clientes ven opciones limitadas a servicios, tickets y su perfil. El administrador ve opciones adicionales para gestión de clientes, configuración del sistema y acceso a monitorización.

---

5. Desarrollo

5.1 Estructura del proyecto

El proyecto AceHosting está compuesto por varios subproyectos que se organizan de la siguiente manera:

```
/opt/storage/elpuig/var/www/acehosting/      # Raíz del proyecto
├── src/                                      # Código fuente de
Laravel
│   ├── app/
│   │   ├── Http/Controllers/                 # Controladores
│   │   ├── Models/                           # Modelos Eloquent
│   │   └── Jobs/                             # Trabajos de cola
│   ├── database/migrations/                  # Migraciones de BD
│   ├── resources/views/                      # Vistas Blade
│   ├── routes/web.php                        # Rutas web
│   └── docker-compose.yml                    # Servicios de la
plataforma
├── monitoring/                               # Configuración de
monitorización
│   ├── prometheus/
│   ├── grafana/dashboards_json/
│   └── loki/
├── clientes/                                 # Contenedores de
clientes
│   └── {subdominio}/                         # Un directorio por
cliente
│       ├── docker-compose.yml
│       └── nginx/
├── nginx/                                    # Configuración de Nginx
│   └── sites-enabled/                        # Virtual hosts dinámicos
├── notes/                                    # Documentación técnica
├── setup.sh                                  # Script de
automatización
└── docker-compose.yml                        # Servicios principales
```

La carpeta `src/` contiene todo el código fuente de Laravel, organizado según la estructura estándar del framework. Los controladores gestionan la lógica de las rutas, los modelos definen las entidades de la base de datos y las migraciones mantienen el esquema de la base de datos versionado. Las vistas Blade se almacenan en `resources/views/` y los jobs de cola en `app/Jobs/`.

**Las vistas blade son lo que coloquialmente definimos como páginas web, es decir, cada vista blade es una página distinta dentro de la web.

La carpeta `monitoring/` contiene los archivos de configuración de Prometheus (prometheus.yml), las definiciones de dashboards de Grafana en formato JSON y la configuración de Loki para la agregación de logs.

La carpeta `clientes/` se crea dinámicamente durante el despliegue de cada nuevo cliente. Contiene un subdirectorio por cada cliente con su propio docker-compose.yml y la configuración de Nginx para su subdominio.

El script `setup.sh` automatiza la configuración inicial del sistema, incluyendo la generación de certificados SSL, la instalación de dependencias, la configuración de la base de datos y el despliegue de los servicios.

5.2 Implementación de funcionalidades

5.2.1 Autenticación y registro de clientes

El sistema de autenticación se ha implementado de forma personalizada para adaptarse a la estructura de datos del proyecto. A diferencia de una instalación estándar de Laravel que utiliza la tabla `users`, AceHosting utiliza la tabla `clientes` para almacenar los datos de los clientes. El modelo `Cliente` extiende la clase `Authenticatable` de Laravel, lo que permite utilizar los mecanismos nativos de autenticación del framework (guard, sesión, protección de rutas).

El registro de clientes valida que el correo electrónico no esté ya registrado y almacena la contraseña cifrada mediante el algoritmo bcrypt proporcionado por Laravel. La validación de los datos del formulario se realiza tanto en el frontend como en el servidor, garantizando la integridad de los datos incluso si las validaciones del cliente son eludidas.

Se ha implementado una distinción clara entre el rol de cliente y el de administrador. El administrador se identifica por tener el ID 1, y las rutas sensibles del panel de administración comprueban esta condición antes de ejecutarse.

5.2.2 Catálogo de servicios y contratación con Stripe

El catálogo de servicios se ha implementado como una vista Blade que consulta la tabla `servicios` y muestra cada servicio con su nombre, descripción y precio. Cada servicio tiene asociados dos identificadores de precio en Stripe (uno para modalidad mensual y otro para anual), lo que permite ofrecer diferentes modalidades de contratación.

El proceso de contratación sigue los siguientes pasos:

1. El cliente selecciona un servicio y una modalidad de pago desde el catálogo.
2. El sistema crea una sesión de Stripe Checkout con los parámetros del servicio seleccionado, incluyendo el identificador del precio en Stripe, la modalidad de suscripción y las URLs de retorno.
3. El cliente es redirigido a la página de pago de Stripe, donde introduce sus datos de pago de forma segura.
4. Una vez completado el pago, Stripe redirige al cliente de vuelta a la plataforma.
5. Stripe envía un webhook al servidor para confirmar el evento `checkout.session.completed`.
6. El webhook procesa el evento, crea un nuevo contrato en la base de datos con estado "pagado" y registra el identificador de la suscripción en Stripe para gestionar futuros pagos.

La integración con el Billing Portal de Stripe permite a los clientes gestionar su método de pago, actualizar la modalidad de suscripción o cancelar su plan sin necesidad de contactar con el administrador.

5.2.3 Despliegue automatizado de contenedores por cliente

Una de las funcionalidades más relevantes del proyecto es el despliegue automatizado del entorno de hosting de cada cliente. El proceso se inicia cuando un contrato alcanza el estado "pagado" y el cliente sube un archivo ZIP con el contenido del sitio web, si cuenta con el servicio de base de datos también sube un archivo .sql y si cuenta con el servicio de mail, introduce los datos para la cuenta principal del mismo.

El sistema realiza las siguientes operaciones:

1. **Creación del directorio del cliente:** se crea una carpeta en `/opt/storage/elpuig/var/www/acehosting/clientes/{subdominio}/`, que reside sobre el almacenamiento DRBD replicado.
2. **Extracción del contenido:** los archivos ZIP y .sql se descomprimen/colocan en el directorio del cliente.
3. **Generación de docker-compose.yml:** se genera dinámicamente un archivo `docker-compose.yml` personalizado para el cliente, que puede incluir:
    - Un contenedor Nginx para servir el sitio web.
    - Un contenedor PostgreSQL opcional si el cliente necesita base de datos.
    - Un contenedor Mailserver opcional si el cliente contrata correo electrónico.
    - Un contenedor Adminer opcional para gestión web de la base de datos.
    - Un contenedor Roundcube opcional para webmail.
4. **Generación de configuración de Nginx:** se crea un archivo de configuración de Nginx para el subdominio del cliente (`{subdominio}.acehosting.com`), que actúa como proxy inverso hacia el contenedor Nginx del cliente.
5. **Despliegue de los contenedores:** se ejecuta de forma automática mediante un worker `docker compose up -d` en el directorio del cliente, lo que inicia todos los contenedores definidos.
6. **Actualización del estado:** el contrato se actualiza a estado "activo".

Este proceso se ejecuta a través de un job de cola de Laravel (`DesplegarDocker`), lo que permite procesar los despliegues de forma asíncrona sin bloquear la respuesta al usuario.

5.2.4 Clúster de alta disponibilidad (DRBD + Pacemaker)

El clúster de alta disponibilidad es la columna vertebral de la infraestructura de AceHosting. Está formado por dos nodos, `serveis1` (192.168.15.170) y `serveis2` (192.168.15.171) conectados a través de una red dedicada para DRBD (10.0.3.0/24).

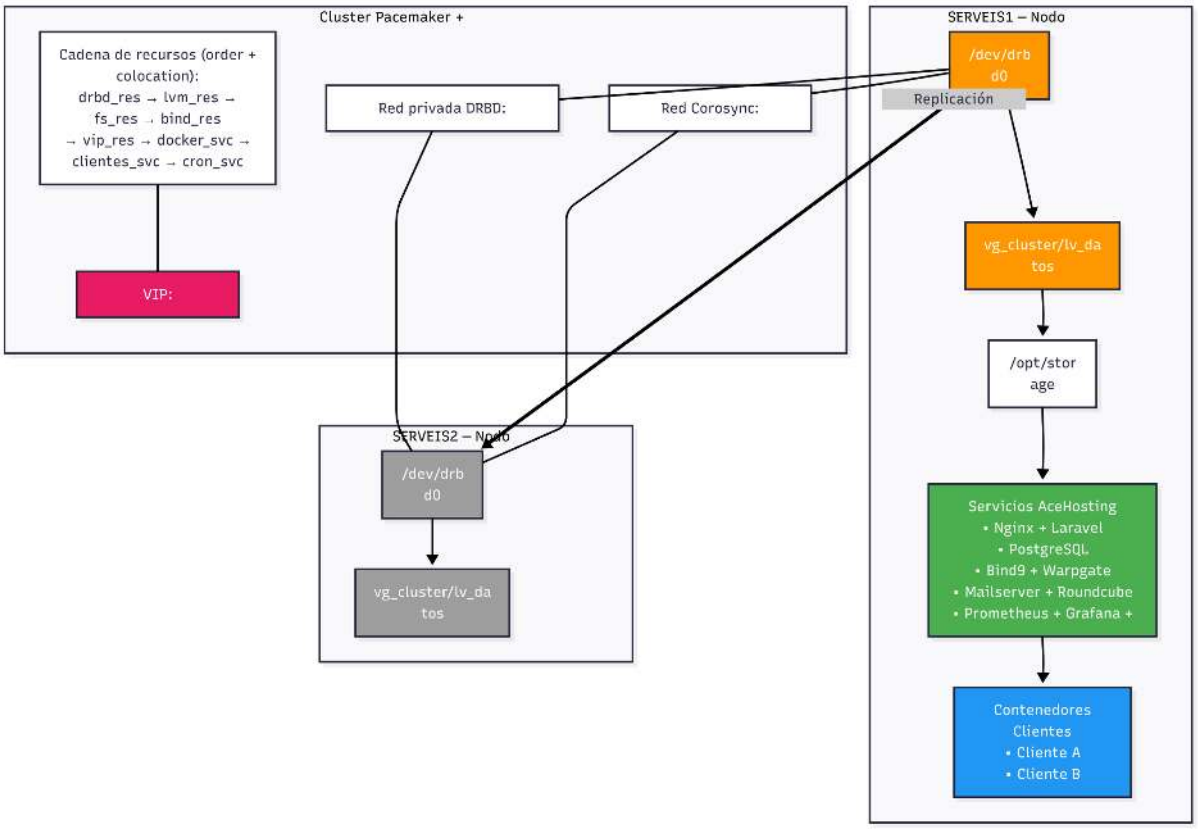


Figura 17: Arquitectura del clúster de alta disponibilidad

DRBD (Distributed Replicated Block Device) proporciona replicación de bloque a nivel de kernel entre los dos nodos. El dispositivo `/dev/drbd0` replica los datos del volumen físico `/dev/sdb1` entre ambos servidores. Sobre DRBD se ha creado un volumen LVM (`vg_cluster/lv_datos`) que se monta como sistema de archivos ext4 en `/opt/storage/elpuig`.

Pacemaker gestiona los recursos del clúster y asegura que los servicios críticos se ejecuten siempre en el nodo activo. Los recursos definidos en el clúster son:

- `drbd_res`: recurso DRBD configurado como clon promovible, lo que permite que un nodo actúe como primario (activo) y el otro como secundario (pasivo).
- `lvm_res`: activa el volumen LVM sobre el dispositivo DRBD.
- `fs_res`: monta el sistema de archivos ext4 en `/opt/storage/elpuig`.
- `bind_res`: crea un bind mount de `/opt/storage/elpuig` a la ubicación esperada por los servicios.
- `vip_res`: dirección IP virtual (192.168.15.220) que flota entre los dos nodos.
- `docker_svc`: servicio Docker, asegurando que está en ejecución en el nodo activo.

Las restricciones de orden (`pcs constraint order`) garantizan que los recursos se inicien en la secuencia correcta: primero DRBD, luego LVM, luego el montaje del sistema de archivos y finalmente Docker y los contenedores de los clientes.

Ante un fallo del nodo primario, Pacemaker detecta la pérdida de comunicación a través de Corosync y promueve el nodo secundario a primario, montando el sistema de archivos, iniciando Docker y arrancando los contenedores de los clientes. Este proceso se completa en menos de 120 segundos (Este tiempo está definido por el hardware que estamos utilizando, en el caso de contar con un hardware profesional con más recursos, este tiempo se ve considerablemente reducido).

5.2.5 Sistema de monitorización (Prometheus + Grafana + Loki)

El sistema de monitorización se ha implementado para proporcionar visibilidad sobre el estado de la infraestructura y de los contenedores de los clientes.

Prometheus se encarga de la recopilación de métricas. Está configurado para recolectar datos de cAdvisor, que expone métricas detalladas de todos los contenedores Docker en ejecución: uso de CPU, memoria, ancho de red y operaciones de disco. El intervalo de recopilación se ha configurado a 15 segundos, lo que proporciona una granularidad suficiente para detectar anomalías sin generar un volumen excesivo de datos.

Loki se encarga de la agregación de logs. Todos los contenedores del sistema están configurados para enviar sus logs a Loki a través del driver de logging de Docker para Loki. Esto centraliza los logs de todos los servicios (Laravel, Nginx, PostgreSQL, contenedores de clientes) en un único punto de consulta.

Grafana proporciona la capa de visualización. Se han configurado dashboards predefinidos para mostrar:

- Métricas de CPU, memoria, disco y red de los servidores.
- Métricas de los contenedores de los clientes.
- Logs del sistema filtrados por servicio.
- Estado del clúster de alta disponibilidad.

Los dashboards de Grafana se han incrustado en el panel de administración de AceHosting mediante iframes, lo que permite al administrador supervisar el estado del sistema sin salir de la aplicación.

5.2.6 Bastión SSH (Warpgate)

Warpgate se ha configurado como bastión SSH para centralizar y asegurar el acceso a los servidores de la infraestructura. El servicio Warpgate escucha en los siguientes puertos:

- Puerto 2222 para conexiones SSH.
- Puerto 8888 para la interfaz de administración web.
- Puerto 33306 para conexiones MySQL/PostgreSQL proxyficadas.

La configuración de Warpgate permite definir objetivos (targets) que son los servidores a los que se puede acceder a través del bastión. Cada objetivo tiene asociados usuarios y roles que determinan quién puede acceder y con qué credenciales. Para la autenticación se utilizan claves SSH, y Warpgate registra todas las sesiones para su posterior auditoría.

5.2.7 Servicio de correo electrónico

El servicio de correo electrónico se ha implementado utilizando docker-mailserver, que proporciona un servidor de correo completo con Postfix (SMTP), Dovecot (IMAP/POP3) y soporte para múltiples dominios.

La gestión de cuentas de correo se ha integrado en el panel de control de AceHosting. Los clientes pueden crear y eliminar cuentas de correo para sus dominios desde la interfaz web. El controlador `CorreoController` ejecuta comandos Docker para añadir o eliminar cuentas en el contenedor mailserver, lo que permite gestionar el correo electrónico sin necesidad de acceso directo al servidor.

El acceso web al correo se proporciona a través de Roundcube, que se ha configurado como un contenedor independiente conectado al servidor de correo interno.

5.2.8 Sistema de tickets de soporte

El sistema de tickets permite a los clientes comunicarse con el administrador para reportar incidencias o realizar consultas. Se ha implementado con las siguientes funcionalidades:

- **Creación de tickets:** el cliente introduce un asunto, selecciona la prioridad y escribe el mensaje inicial.
- **Gestión de estados:** los tickets avanzan por los estados "pendiente", "en curso", "solucionado" y "cerrado". El administrador puede cambiar el estado en cualquier momento.
- **Relación con contrato:** Cada ticket está asociado con el contrato del cliente lo que facilita el trabajo al administrador a la hora de identificar problemas.
- **Visibilidad:** los clientes ven únicamente sus propios tickets, mientras que el administrador ve todos los tickets del sistema.

5.2.9 Panel de administración

El panel de administración es la interfaz desde la que el administrador supervisa y gestiona toda la plataforma. Sus funcionalidades principales son:

- **Visión general del sistema:** indicadores del número de clientes activos y estado de los servidores.
- **Gestión de clientes:** permite consultar la lista de clientes registrados y cambiar su estado (activo, suspendido, inactivo).
- **Monitorización:** dashboards de Grafana incrustados con métricas detalladas del sistema.
- **Gestión de tickets (otro panel):** permite ver todos los tickets del sistema, cambiar su prioridad y estado.

El panel de administración está protegido y solo es accesible para el usuario con ID 1, que se identifica como administrador durante la instalación del sistema.

5.3 Pruebas

Las pruebas realizadas se han centrado en verificar el correcto funcionamiento de las funcionalidades principales y de la integración entre los distintos componentes del sistema.

Pruebas funcionales

Se han realizado pruebas funcionales sobre cada una de las funcionalidades del panel de control: registro e inicio de sesión de clientes, visualización del catálogo de servicios, proceso de contratación con Stripe (Solo en modo prueba), creación y respuesta de tickets, y gestión de cuentas de correo.

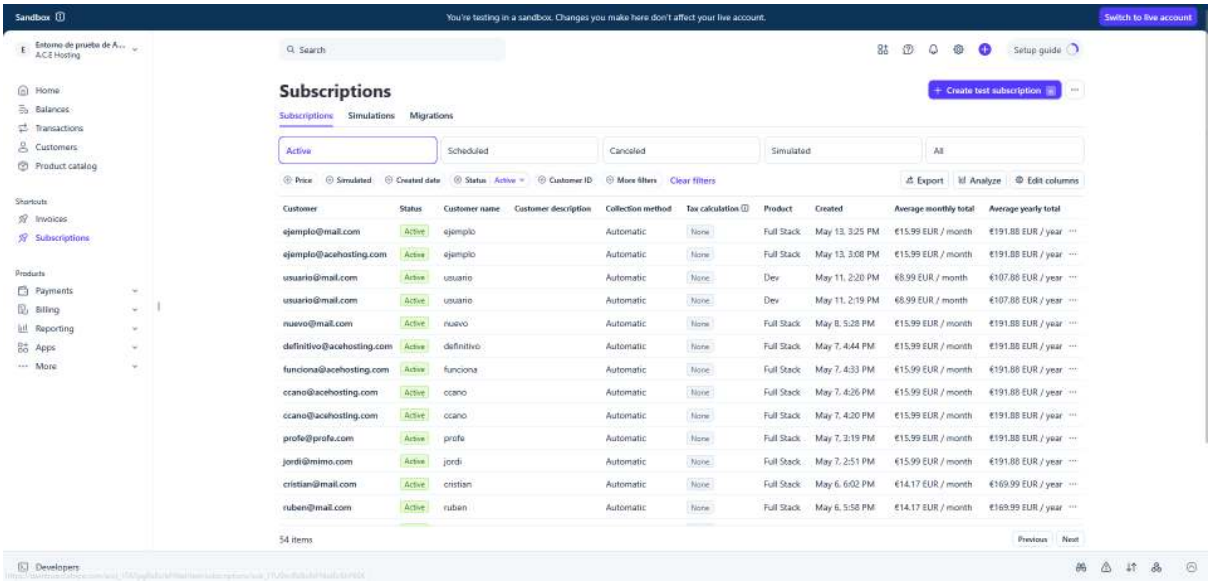


Figura 18: Panel que muestra las suscripciones en Stripe.

Las pruebas más relevantes han sido las relacionadas con el proceso de contratación y despliegue automatizado. Se verificó que el flujo completo funciona correctamente: desde que el cliente selecciona un servicio y completa el pago en Stripe, hasta que los contenedores del cliente se despliegan automáticamente y su sitio web es accesible a través del subdominio asignado.

Durante estas pruebas se detectó un problema en el que los webhooks de Stripe no se procesaban correctamente cuando llegaban antes de que la sesión de checkout hubiera finalizado en el frontend. Se resolvió implementando una cola de procesamiento de webhooks que garantiza que los eventos se procesen en el orden correcto y que no se pierdan notificaciones.

Pruebas de integración

Se verificó la comunicación entre todos los componentes del sistema: el panel de control Laravel con la base de datos PostgreSQL, el panel con el servicio de colas para el despliegue de contenedores, la integración con las APIs de Stripe, y la comunicación entre el proxy Nginx principal y los contenedores de los clientes.

Pruebas de alta disponibilidad

Se realizaron pruebas de conmutación por fallo del clúster DRBD+Pacemaker. Se simuló la caída del nodo primario (`serveis1`) y se verificó que el nodo secundario (`serveis2`) asumía correctamente el rol de primario, montaba el sistema de archivos, iniciaba los servicios Docker y arrancaba los contenedores de los clientes. El tiempo de conmutación se mantuvo dentro del límite de 120 segundos en todas las pruebas realizadas.

También se verificó que, tras restaurar el nodo fallido, este se reincorporaba correctamente al clúster como secundario y que la replicación DRBD se sincronizaba sin pérdida de datos.

Pruebas de seguridad

Se verificó que las rutas de administración no son accesibles para usuarios no autenticados o para clientes sin rol de administrador. También se comprobó que los webhooks de Stripe incluyen la verificación de firma, garantizando que solo Stripe puede enviar eventos válidos al sistema.

---

6. Conclusiones

6.1 Conclusiones generales

El resultado del proyecto es una plataforma de hosting funcional que integra todas las capacidades necesarias para operar como un proveedor de servicios de alojamiento web profesional. El panel de control permite gestionar clientes, servicios, pagos, tickets de soporte y monitorización desde una interfaz unificada, mientras que la infraestructura subyacente garantiza la continuidad del servicio mediante un clúster de alta disponibilidad.

A nivel de aprendizaje, el proyecto ha permitido consolidar y aplicar de forma práctica los conocimientos adquiridos a lo largo del ciclo formativo de ASIX en áreas muy diversas. La administración de sistemas Linux se ha puesto en práctica en la configuración de los servidores y del clúster de alta disponibilidad. La gestión de bases de datos se ha aplicado en el diseño e implementación del modelo de datos con PostgreSQL. Los conocimientos de redes han sido fundamentales para la configuración de las interfaces de red, el servidor DNS y las redes Docker. El desarrollo de aplicaciones web con Laravel ha permitido construir el panel de control, y la administración de servicios de internet se ha reflejado en la configuración del servidor de correo y del proxy Nginx.

Una de las partes que más nos ha aportado del proyecto ha sido la integración de todas estas tecnologías en un sistema centralizado y consistente. Los principales desafíos han sido la configuración del clúster de alta disponibilidad, especialmente la coordinación entre DRBD, Pacemaker y los servicios Docker, que requirió un esfuerzo significativo de investigación y pruebas y también la automatización en la puesta en marcha de los contenedores de los clientes, ya que hasta dar con la solución de usar un worker, no eramos capaces de hacerlo funcionar.

Como aspecto mejorable, la interfaz de usuario del panel de control es funcional pero podría beneficiarse de un diseño más cuidado. Las decisiones de diseño se tomaron priorizando la funcionalidad, lo que ha dado como resultado una aplicación que funciona correctamente pero cuyo aspecto visual es mejorable en términos de experiencia de usuario.

6.2 Consecución de objetivos

El objetivo general del proyecto, desarrollar un panel de control web para la gestión automatizada de servicios de hosting sobre una infraestructura de alta disponibilidad, se ha alcanzado satisfactoriamente. La plataforma es funcional y cubre las necesidades planteadas al inicio del proyecto.

Respecto a los objetivos específicos:

Objetivo	**Estado**	**Observaciones**
Sistema de autenticación y registro	Completado	Los clientes pueden registrarse e iniciar sesión de forma segura
Catálogo de servicios con Stripe	Completado	La contratación y el pago funcionan correctamente con Stripe Checkout
Despliegue automatizado de contenedores	Completado	El sistema genera y despliega contenedores Docker por cliente
Clúster de alta disponibilidad	Completado	DRBD + Pacemaker garantiza la conmutación automática
Sistema de monitorización	Completado	Prometheus, Grafana y Loki proporcionan métricas y logs
Sistema de tickets	Completado	Los clientes pueden crear tickets y el administrador gestionarlos
Servicio de correo electrónico	Completado	docker-mailserver + Roundcube operativos con gestión desde el panel
Bastión SSH con Warpgate	Completado	Acceso SSH centralizado y seguro a los servidores

Todos los objetivos planteados se han cumplido de forma completa. La integración con Stripe para suscripciones mensuales y anuales funciona de forma fiable, y el sistema de webhooks procesa correctamente los eventos de pago, actualización y cancelación de suscripciones. El despliegue automatizado de contenedores ha quedado operativo, permitiendo que al confirmarse un pago se genere y ejecute todo el entorno del cliente de forma autónoma.

6.3 Valoración de la metodología y la planificación

La planificación inicial se ha seguido en términos generales, aunque con algunas desviaciones en las fases intermedias. La fase de configuración de la infraestructura requirió más tiempo del previsto, principalmente por la complejidad de la configuración del clúster de alta disponibilidad. Los problemas encontrados durante la integración de Pacemaker con los servicios Docker no se habían anticipado en la planificación inicial y requirieron un esfuerzo adicional de investigación y pruebas.

Como consecuencia, la fase de funcionalidades avanzadas se vio ligeramente comprimida, aunque esto no afectó al cumplimiento de los objetivos gracias a la priorización de las funcionalidades más críticas.

La metodología de trabajo por fases ha resultado adecuada para este proyecto. El enfoque de construir primero la infraestructura base antes de desarrollar el software ha sido acertado, ya que ha permitido detectar y resolver problemas de configuración sin afectar al desarrollo del panel de control. En una planificación futura sería recomendable reservar un margen de tiempo adicional para las fases de mayor complejidad técnica.

6.4 Visión de futuro

La plataforma AceHosting tiene un recorrido amplio de mejora y evolución. Las líneas de trabajo más inmediatas son las siguientes:

En el ámbito del panel de control, la mejora más evidente es el trabajo en el diseño visual de la interfaz, que actualmente es funcional pero mejorable en términos de experiencia de usuario. También sería interesante añadir un sistema de facturación automatizada que genere y envíe facturas a los clientes de forma periódica, una funcionalidad que actualmente está soportada a nivel de base de datos pero pendiente de implementar en la interfaz.

En el ámbito de la infraestructura, una evolución natural sería ampliar el clúster a tres o más nodos para lograr una tolerancia a fallos aún mayor. También podría explorarse la migración a orquestadores de contenedores como Kubernetes si el número de clientes creciera significativamente, aunque esto implicaría un esfuerzo de adaptación considerable.

En el ámbito de la monitorización, sería interesante añadir alertas automáticas que notifiquen al administrador cuando se detecten anomalías en las métricas de los servidores o de los contenedores de los clientes, así como un sistema de autoescalado que ajuste los recursos de los contenedores en función de la demanda.

Por último, la plataforma podría evolucionar hacia un modelo multiadministrador, permitiendo que varios operadores gestionen el sistema con distintos niveles de permisos, lo que la haría adecuada para proveedores de hosting con equipos técnicos de mayor tamaño.

---

7. Glosario

A continuación se definen los principales términos técnicos y acrónimos utilizados en esta memoria.

API (Application Programming Interface): Interfaz que permite la comunicación entre dos sistemas de software mediante un conjunto de reglas y definiciones predefinidas.

Bastión SSH: Servidor que actúa como punto de entrada único para acceder a otros servidores de una infraestructura, mejorando la seguridad al centralizar y auditar todos los accesos.

bcrypt: Algoritmo de cifrado utilizado para almacenar contraseñas de forma segura, transformándolas en un valor irreversible.

Bind9: Servidor DNS de código abierto ampliamente utilizado, que permite la resolución de nombres de dominio en redes IP.

cAdvisor: Agente de Google que expone métricas detalladas de los contenedores Docker en ejecución.

Checkout (Stripe Checkout): Producto de Stripe que proporciona una página de pago alojada y lista para usar, sin necesidad de desarrollar la interfaz de pago.

Clúster: Conjunto de servidores que trabajan de forma coordinada para proporcionar alta disponibilidad, balanceo de carga o capacidad de proceso.

Corosync: Sistema de comunicación entre nodos de un clúster que proporciona detección de fallos y mensajería fiable.

CRUD (Create, Read, Update, Delete): Conjunto de las cuatro operaciones básicas que pueden realizarse sobre los datos de un sistema: crear, leer, actualizar y eliminar.

DNS (Domain Name System): Sistema que traduce nombres de dominio legibles por humanos (como acehosting.com) a direcciones IP numéricas.

DRBD (Distributed Replicated Block Device): Tecnología de replicación de bloques a nivel de kernel para Linux, que permite mantener dos dispositivos de bloque sincronizados entre servidores.

Eloquent ORM: Mapeador objeto-relacional incluido en Laravel que permite interactuar con la base de datos utilizando objetos PHP en lugar de consultas SQL directas.

FPM (FastCGI Process Manager): Gestor de procesos FastCGI para PHP, utilizado en entornos de producción para servir aplicaciones PHP a través de Nginx.

Grafana: Plataforma de código abierto para la visualización y análisis de métricas, que permite crear dashboards personalizados a partir de múltiples fuentes de datos.

HA (High Availability): Alta disponibilidad. Capacidad de un sistema para mantenerse operativo y accesible incluso ante fallos de algunos de sus componentes.

JSONB: Tipo de datos de PostgreSQL que almacena datos JSON en formato binario, permitiendo consultas eficientes sobre su contenido.

Laravel: Framework de desarrollo web para PHP que sigue el patrón MVC, conocido por su sintaxis elegante y su ecosistema de herramientas integradas.

Loki: Sistema de agregación de logs de código abierto, desarrollado por Grafana Labs, diseñado para ser eficiente y fácil de integrar con entornos Docker.

MVC (Model-View-Controller): Patrón de arquitectura de software que separa la lógica de negocio (Modelo), la interfaz de usuario (Vista) y la lógica de control (Controlador).

Nginx: Servidor web y proxy inverso de alto rendimiento, ampliamente utilizado por su eficiencia y su capacidad para gestionar múltiples conexiones concurrentes.

ORM (Object-Relational Mapping): Técnica que permite interactuar con una base de datos relacional utilizando objetos del lenguaje de programación.

Pacemaker: Gestor de recursos de clúster que garantiza la máxima disponibilidad de los servicios monitorizando los nodos y gestionando los recursos del clúster.

PostgreSQL: Sistema de gestión de bases de datos relacionales de código abierto, conocido por su robustez, su cumplimiento de estándares SQL y sus funcionalidades avanzadas.

Prometheus: Sistema de monitorización y alerta de código abierto, diseñado para recopilar métricas mediante un modelo de extracción (pull) desde los servicios monitorizados.

Proxy inverso: Servidor intermedio que recibe las peticiones de los clientes y las redirige al servidor de destino, proporcionando características adicionales como balanceo de carga, terminación SSL y caché.

Roundcube: Cliente de correo web de código abierto, con interfaz moderna y soporte para IMAP y SMTP.

Stripe: Plataforma de procesamiento de pagos en línea que proporciona APIs para integrar pagos, suscripciones y facturación en aplicaciones web.

VIP (Virtual IP): Dirección IP virtual que flota entre los nodos de un clúster, permitiendo que los clientes accedan al servicio siempre a través de la misma dirección IP independientemente del nodo activo.

Warpgate: Bastión SSH/MySQL/PostgreSQL de código abierto, con interfaz web para la gestión de usuarios y objetivos.

Webhook: Mecanismo de comunicación entre sistemas que permite a un servicio enviar notificaciones automáticas a otro servicio cuando ocurre un evento determinado.

---

8. Bibliografía

A continuación se listan las fuentes consultadas durante el desarrollo del proyecto, ordenadas por número de referencia según el orden en que aparecen citadas en la memoria.

[0] IsardVDI (2025). https://www.isardvdi.com/

[1] Laravel LLC. (2025). *Laravel 12.x Documentation*. Laravel. https://laravel.com/docs/12.x

[2] The PostgreSQL Global Development Group. (2024). *PostgreSQL 15 Documentation*. PostgreSQL. https://www.postgresql.org/docs/15/

[3] Stripe Inc. (2025). *Stripe API Reference*. Stripe. https://stripe.com/docs/api

[4] Docker Inc. (2024). *Docker Compose Overview*. Docker Documentation. https://docs.docker.com/compose/

[5] LINBIT. (2024). *DRBD User's Guide*. LINBIT. https://docs.linbit.com/docs/users-guide-8.4/

[6] ClusterLabs. (2024). *Pacemaker Explained*. ClusterLabs. https://www.clusterlabs.org/pacemaker/doc/

[7] ClusterLabs. (2024). *Corosync Documentation*. ClusterLabs. https://corosync.github.io/corosync/

[8] Prometheus Authors. (2025). *Prometheus Documentation*. Prometheus. https://prometheus.io/docs/

[9] Grafana Labs. (2025). *Grafana Documentation*. Grafana. https://grafana.com/docs/grafana/

[10] Grafana Labs. (2025). *Loki Documentation*. Grafana. https://grafana.com/docs/loki/

[11] Google. (2024). *cAdvisor Documentation*. GitHub. https://github.com/google/cadvisor

[12] Warpgate contributors. (2025). *Warpgate Documentation*. GitHub. https://github.com/warp-tech/warpgate

[13] Internet Systems Consortium. (2024). *BIND 9 Documentation*. ISC. https://bind9.readthedocs.io/

[14] docker-mailserver contributors. (2025). *Docker Mailserver Documentation*. GitHub. https://docker-mailserver.github.io/docker-mailserver/

[15] Roundcube contributors. (2024). *Roundcube Webmail Documentation*. Roundcube. https://roundcube.net/about/

[16] Netplan contributors. (2024). *Netplan Documentation*. Ubuntu. https://netplan.readthedocs.io/en/stable/

[17] Bootstrap contributors. (2024). *Bootstrap 5 Documentation*. Bootstrap. https://getbootstrap.com/docs/5.3/

[18] Canonical Ltd. (2024). *Ubuntu Server 24.04 Documentation*. Ubuntu. https://ubuntu.com/server/docs

9. Anexos

Anexo A — Manual de instalación y despliegue

Este anexo describe los pasos necesarios para instalar y configurar la plataforma AceHosting en un entorno nuevo. Se asume que se dispone de dos servidores con Ubuntu Server 24.04 y conectividad de red entre ellos.

Requisitos previos

- Dos servidores con Ubuntu Server 24.04 instalado.
- Al menos 4 GB de RAM en cada nodo.
- Disco duro adicional para DRBD (al menos 20 GB).
- Conectividad de red dedicada para DRBD y Corosync.
- Docker y Docker Compose instalados.
- Git instalado.

Pasos de instalación

1. Clonar el repositorio del proyecto en el directorio `/opt/storage/elpuig/var/www/acehosting/`.
2. Ejecutar el script `setup.sh` que automatiza la configuración inicial: generación de certificados SSL, creación de la red Docker `ace_proxy_network`, instalación del driver de Loki para Docker y creación de los directorios de datos.
3. Configurar el archivo `.env` de Laravel con los parámetros de conexión a la base de datos, las claves de API de Stripe y la configuración del servidor de correo.
4. Ejecutar las migraciones de Laravel para crear las tablas en la base de datos: `php artisan migrate`.
5. Configurar el clúster de alta disponibilidad siguiendo las guías incluidas en el directorio `notes/`.
6. Iniciar todos los servicios con `docker compose up -d`.
7. Verificar que todos los servicios están operativos accediendo al panel de control a través del dominio configurado.

Para más detalle sobre cada paso, consultar la documentación técnica incluida en el directorio `notes/` del repositorio.

Anexo B — Manual de usuario

Este anexo describe el funcionamiento de la plataforma desde el punto de vista del usuario final.

Registro e inicio de sesión

Para acceder al panel de control, el cliente debe registrarse en la plataforma a través de la página de registro, introduciendo su nombre, correo electrónico y contraseña. Una vez registrado, puede iniciar sesión con su correo y contraseña desde la página de login.

Contratación de servicios

Desde el catálogo de servicios, el cliente puede consultar los planes disponibles y seleccionar el que mejor se adapte a sus necesidades. Tras seleccionar un servicio y la modalidad de pago (mensual o anual), el cliente es redirigido a la pasarela de pago de Stripe. Una vez completado el pago, el servicio se activa automáticamente.

Despliegue del sitio web

Tras la contratación, el cliente debe proporcionar el contenido de su sitio web (a través del administrador) en un archivo ZIP, y si es necesario debe proporcionar los datos/archivos necesarios para otros servicios contratados. El sistema desplegará automáticamente los contenedores necesarios y el sitio web será accesible a través de un subdominio de acehosting.com.

Gestión de tickets

El cliente puede abrir tickets de soporte desde la sección correspondiente del panel de control. Para crear un ticket, debe introducir un asunto, seleccionar la prioridad y escribir el mensaje. El administrador podrá ver el ticket y cambiar su estado.

Correo electrónico

Si el cliente ha contratado servicios de correo electrónico, puede gestionar sus cuentas de correo desde el panel de control: crear nuevas cuentas, eliminar existentes o acceder a Roundcube para consultar su correo desde el navegador.

Anexo C — Documentación de la API de Webhooks de Stripe

La plataforma AceHosting expone un punto final (endpoint) de webhook para recibir notificaciones de Stripe. El endpoint está configurado en la ruta `/webhook/stripe` de la aplicación Laravel y requiere verificación de firma para garantizar que las peticiones provienen legítimamente de Stripe.

Eventos gestionados

Evento de Stripe	**Acción del sistema**
`checkout.session.completed`	Crear un nuevo contrato con estado "pagado"
`invoice.paid`	Actualizar el estado de la suscripción y registrar el pago
`customer.subscription.updated`	Actualizar la modalidad de suscripción en el contrato
`customer.subscription.deleted`	Marcar el contrato como "cancelado"

Formato de las peticiones

Stripe envía las peticiones en formato JSON con una estructura estándar que incluye el tipo de evento, el identificador del objeto relacionado y los datos del evento. Laravel procesa estas peticiones y ejecuta la lógica de negocio correspondiente en función del tipo de evento recibido.

Verificación de firma

Todas las peticiones recibidas incluyen una cabecera `Stripe-Signature` que el sistema verifica utilizando el `webhook_secret` configurado en el archivo `.env`. Si la firma no es válida, la petición es rechazada con un código de error 401.

Anexo D — Documentación progresiva del desarrollo del proyecto

Aquí se encuentran todas las notas en las que se han ido añadiendo las modificaciones, problemas encontrados y sus soluciones, o simplemente guías paso a paso de todo lo que se ha ido realizando durante el transcurso del proyecto.

https://gitlab.com/puig_ca/Proyecto/-/tree/27ee9d65fe108287c9d408a134a0af7542b237d1/docs/guides/prueba_laravel/notes

Anexo E — Declaración de uso de inteligencia artificial

Durante la elaboración de esta memoria se han utilizado herramientas de inteligencia artificial como apoyo en la redacción y organización de los contenidos. En todo momento, el contenido refleja las decisiones técnicas reales tomadas durante el desarrollo del proyecto, la infraestructura realmente configurada y las funcionalidades efectivamente implementadas y verificadas.

Asimismo, se ha hecho uso de la IA en ciertas partes del desarrollo del proyecto, sobre todo en fases de investigación y aprendizaje sobre el uso de algunas tecnologías que se han utilizado.

La IA se ha utilizado para mejorar la fluidez del texto, estructurar algunos apartados y generar borradores iniciales que posteriormente han sido revisados, adaptados y completados manualmente para asegurar su precisión y adecuación al trabajo realizado.

Los autores asumen la responsabilidad completa sobre el contenido de este documento.