Reporte técnico: Proyecto final de Sistemas Operativos y Laboratorio

# 1. Información del Proyecto

* **Título del Proyecto:** Despliegue Frontend AWS Zero Downtime
* **Curso/Materia:** Sistemas Operativos
* **Integrantes: Andres Calvo Ariza (**[**andres.calvoa@udea.edu.co**](mailto:andres.calvoa@udea.edu.co)**), Emanuel López Higuita (**[**emanuel.lopezh@udea.edu.co**](mailto:emanuel.lopezh@udea.edu.co)**)**
* **Fecha de Entrega:** 16 Julio 2025

# 2. Introducción

## 2.1. Objetivo del Proyecto

El objetivo principal es resolver el problema de cómo desplegar aplicaciones frontend tipo SPA (Single Page Application) sin necesidad de aplicar una ventana de mantenimiento para bajar servidores y subir cambios. Esto permite despliegues en cualquier momento del día, una característica esencial que hoy ofrecen servicios pagos como Vercel, Netlify o AWS Amplify. Con este proyecto, se busca socializar con la comunidad tecnológica lo fácil que es configurar esta funcionalidad directamente en AWS, evitando sobrecostos asociados a soluciones de terceros.

## 2.2. Motivación y Justificación

La relevancia de este proyecto en el contexto de sistemas operativos radica en cómo se gestionan recursos del sistema y redes de manera eficiente. Al automatizar despliegues con GitHub Actions, se exploran temas como gestión de procesos, sincronización de tareas, comunicación entre servicios distribuidos y optimización de cache. Además, permite comprender cómo servicios en la nube abstraen y exponen recursos del sistema operativo para construir soluciones escalables.

## 2.3. Alcance del Proyecto

Incluye configuración de S3, CloudFront, CloudFront Functions para soporte SPA, automatización con GitHub Actions y manejo de multilenguaje. No se aborda backend ni autentificación.

# 3. Marco Teórico / Conceptos Fundamentales

* **AWS S3:** Almacenamiento estático para servir contenido web.
* **CloudFront:** CDN global que distribuye contenido eficientemente.
* **CloudFront Functions:** Funciones ligeras ejecutadas en Node.js, para manipular requests/responses.
* **GitHub Actions:** Automatización de workflows CI/CD.
* **React + i18next:** SPA multilenguaje con internacionalización dinámica.
* **TypeScript:** Superset de JavaScript que añade tipado estático para mayor robustez.
* **rsbuild:** Compilador y empaquetador web escrito en Rust, diseñado para generar builds rápidos y altamente optimizados con las mejores prácticas de la industria.
* **Cache-Control:** Encabezados HTTP para definir políticas de cacheo en navegador y CDN.

# 4. Diseño e Implementación

## 4.1. Diseño de la Solución

Diagrama

El contenido generado por IA puede ser incorrecto.

Las decisiones de diseño tomadas en este proyecto tuvieron como objetivo construir una arquitectura robusta, escalable y preparada para crecer, evitando soluciones temporales que luego generaran problemas de mantenimiento.

La primera decisión clave fue utilizar un CDN, específicamente Amazon CloudFront, para distribuir los archivos estáticos de la aplicación frontend. Esto no solo permite reducir la latencia y mejorar la experiencia de usuario al servir contenido desde ubicaciones cercanas (edge locations), sino que también protege al bucket S3, ya que el acceso al público se hace exclusivamente a través de CloudFront, dejando S3 como un recurso privado. Esta separación mejora la seguridad y permite aplicar optimizaciones específicas en cada capa.

Un desafío típico en aplicaciones SPA (Single Page Application) es que, al no ser rutas reales en el servidor, sino rutas virtuales manejadas por React en el navegador, los accesos directos a URLs como /about o /dashboard provocan errores si el servidor no sabe devolver siempre el mismo index.html. Una solución básica sería configurar en S3 el index.html como root object, pero esto solo resolvería la raíz (/) y no funcionaría para rutas más complejas. Por este motivo, se decidió implementar una CloudFront Function llamada Spa-Redirection, que detecta cualquier ruta que no corresponda a un archivo real y la redirige al index.html, asegurando que React maneje el ruteo correctamente en el cliente. Esta decisión no solo garantiza el funcionamiento actual, sino que permite escalar la aplicación sin preocuparse por nuevas rutas.

En cuanto al manejo de cache, se analizaron dos alternativas. La primera era configurar los headers Cache-Control directamente en los objetos del bucket S3, aprovechando la capacidad de S3 de almacenar metadatos HTTP por archivo. Sin embargo, esto habría complicado el workflow de despliegue en GitHub Actions, porque durante cada build y subida a S3 sería necesario agregar un paso para asignar los headers correctos, aumentando la complejidad del pipeline. Por ello, se optó por una CloudFront Function llamada Cache-Headers, que establece dinámicamente los headers según el tipo de archivo: cache largo (max-age=31536000, immutable) para los recursos estáticos como .js y .css, y cache corto (max-age=60, must-revalidate) para index.html. Aunque esta opción introduce un costo adicional (alrededor de 10 centavos de dólar por millón de invocaciones), simplifica el despliegue y garantiza consistencia. Si en el futuro se busca optimizar costos, configurar los headers directamente en S3 sería una mejora recomendable.

## 4.2. Tecnologías y Herramientas

* **Lenguajes:** TypeScript, JavaScript.
* **Frameworks:** React, React Router, i18next.
* **Herramientas:** NodeJS LTS, Git, Rsbuild, GitHub Actions, AWS CLI, VsCode, GitHub.
* **Infraestructura:** S3, CloudFront, CloudFront Functions.

## 4.3. Detalles de Implementación

**Crear llaves de acceso en AWS (IAM):**  
Se generaron llaves con permisos mínimos para que GitHub Actions pudiera autenticarse y tener acceso al bucket S3 y a la distribución CloudFront.

**Crear bucket S3:**  
Se configuró como privado. El acceso al contenido se hace exclusivamente a través de CloudFront, usando Origin Access Control (OAC) para asegurar que solo CloudFront pueda leer los objetos.

**Crear distribución CloudFront:**  
Se configuró apuntando al bucket S3, usando HTTPS, con una política de cache optimizada, y habilitando la asociación de funciones (CloudFront Functions).

**Crear CloudFront Function llamada Spa-Redirection (Viewer Request):**  
Esta función intercepta las rutas del navegador que no corresponden a archivos reales (es decir, no tienen extensión como .js, .css, etc.) y las redirige a /index.html. Esto es fundamental para aplicaciones SPA, porque permite que React maneje el enrutamiento en el cliente.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Crear CloudFront Function llamada Cache-Headers (Viewer Response):**  
Aquí se inyectó código para establecer encabezados Cache-Control distintos según el tipo de archivo:

* Archivos estáticos (.js, .css, imágenes): cache prolongado (max-age=31536000, immutable).
* Archivo index.html: cache corto (max-age=60, must-revalidate), porque es el punto de entrada que enlaza los archivos estáticos.

Interfaz de usuario gráfica, Texto, Aplicación

El contenido generado por IA puede ser incorrecto.

**Compilación con rsbuild:**  
Se utilizó rsbuild, un empaquetador moderno escrito en Rust, que genera builds rápidos y optimizados, añadiendo hashes únicos a los nombres de los archivos (por ejemplo, main.a1b2c3.js). Este hash permite que los navegadores detecten automáticamente cambios y descarguen las nuevas versiones.  
Como index.html mantiene siempre el mismo nombre, es crítico asignarle un cache bajo para evitar problemas de versiones y asegurar que siempre cargue las referencias correctas a los nuevos archivos.

# 5. Pruebas y Evaluación

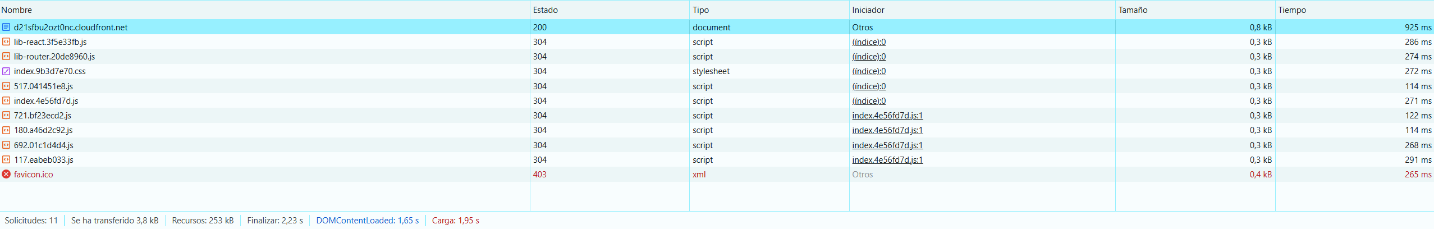
## 5.1. Metodología de Pruebas

Pruebas manuales verificando cambios en CSS, JS, traducciones, comprobando el comportamiento adecuado del cache. Además de esto se recolectan métricas en cuanto al peso de los archivos transferidos en red luego de desplegar un cambio.

## 5.2. Casos de Prueba y Resultados

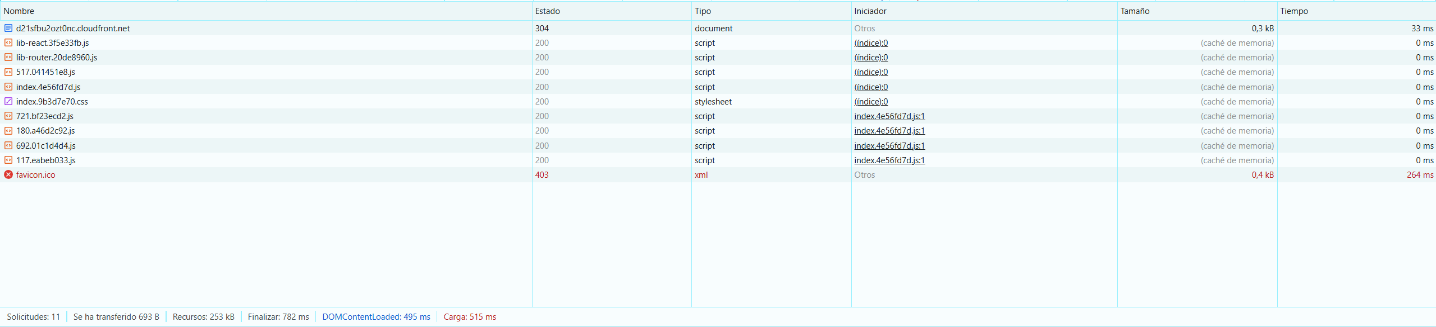
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID Caso de prueba** | **Descripción del caso de prueba** | **Resultado esperado** | **Resultado obtenido** | **Éxito/Fallo** |
| CP-001 | Cambio en archivos CSS | Descarga de nuevo archivo css con los cambios reflejados | Los cambios son reflejados | Éxito |
| CP-002 | Cambio en JS | Nuevo archivo de JS con un hash diferente | Nuevo archivo siendo descargado | Éxito |
| CP-003 | Cambio en traducciones | Las traducciones se actualizan correctamente | Se observan las nuevas llaves de traducción añadidas | Éxito |

## 5.3. Evaluación del Rendimiento



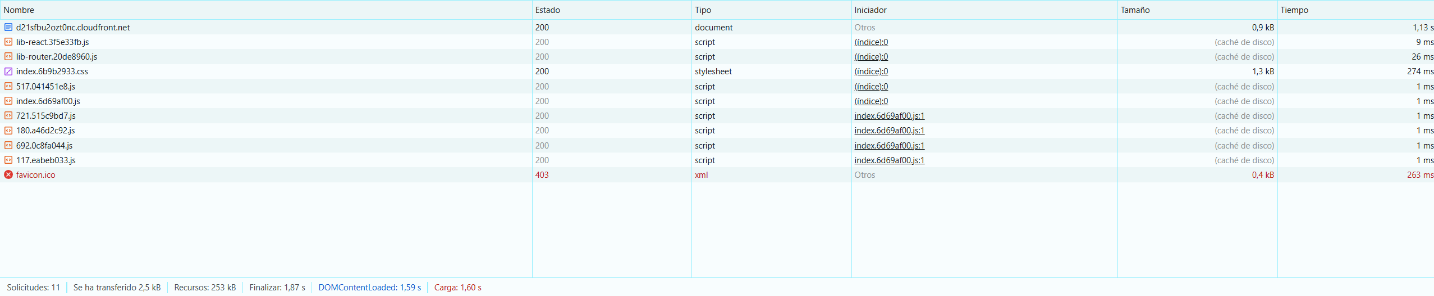
Al entrar por primera vez al ambiente desplegado disponible en: <https://d21sfbu2ozt0nc.cloudfront.net/> observamos que se transfieren por red 3.8KB que luego al ser descomprimidos por el navegador representan un total de 253KB, esto nos brinda un ahorro de 98% gracias a la compresión realizada por la CDN Cloudfront usando algoritmos como gzip.

Luego cuando ingresamos una segunda vez podemos observar lo siguiente:



Con un total transferido por red de 696 Bytes, representa un ahorro del 81% con respecto a los 3.8KB gracias a tener en cache de memoria los archivos.

Finalmente ejecutando un caso de prueba, como lo fue modificar un archivo de CSS luego de tener la aplicación en cache, observamos lo siguiente:

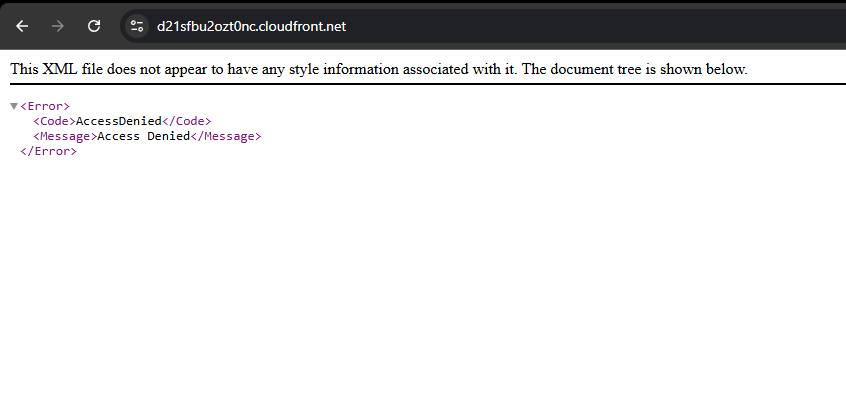


Se puede apreciar como solo es descargado el archivo CSS y los demás archivos JS son obtenidos de la cache.

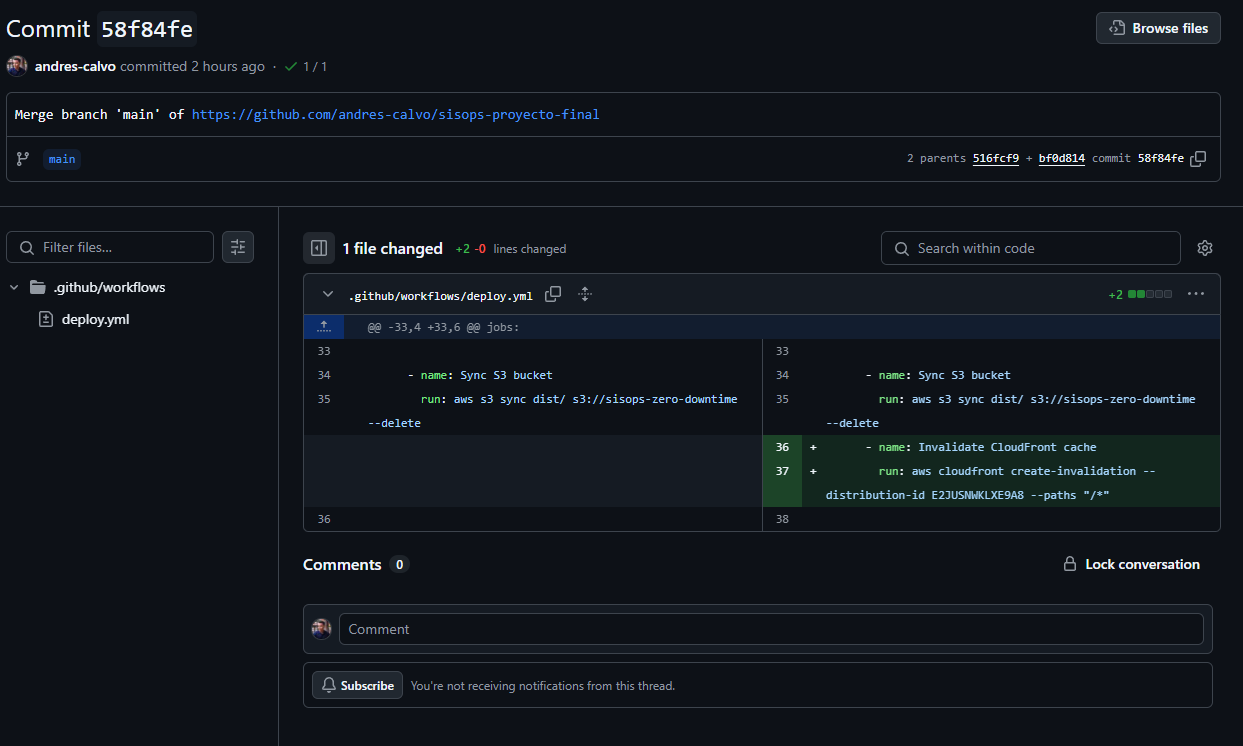
## 5.4. Problemas Encontrados y Soluciones

Describa los problemas o errores significativos encontrados durante el desarrollo y las pruebas, y cómo los resolvió.

* **Problema:** El archivo index.html quedaba cacheado en navegador, obligando a limpiar manualmente para ver cambios.  
  **Solución:** Se creó la CloudFront Function Cache-Headers para asignar max-age=60, must-revalidate solo a index.html, permitiendo revalidación automática y evitando versiones obsoletas.
* **Problema:** CloudFront accede literalmente al bucket S3; rutas como / intentaban buscar carpetas inexistentes en S3.  
  **Solución:** Se creó la CloudFront Function Spa-Redirection para mapear cualquier ruta que no pertenezca a un archivo a index.html. Esto es esencial en SPA, donde el enrutado lo maneja React, no el servidor.



* **Problema:** Aunque el build generaba archivos nuevos con hash, CloudFront seguía sirviendo versiones viejas.  
  **Solución:** Se ajustó el pipeline GitHub Actions para incluir un paso de invalidación de CloudFront tras el despliegue, notificando al CDN sobre el contenido nuevo para actualizarlo inmediatamente.



# 6. Conclusiones

El proyecto alcanzó su objetivo principal: implementar un despliegue frontend SPA en AWS con zero downtime, utilizando S3, CloudFront, GitHub Actions y rsbuild.

Se lograron solucionar problemas clave como el manejo de rutas dinámicas, la configuración de cache y la automatización del despliegue, cumpliendo ampliamente con lo planeado.

Durante el desarrollo, se aplicaron conceptos de sistemas operativos como concurrencia en pipelines, sincronización de tareas y gestión eficiente de recursos distribuidos.

También se profundizó en cómo los sistemas en la nube abstraen operaciones de bajo nivel para garantizar alta disponibilidad.

El proyecto fue exitoso al demostrar que es posible replicar funcionalidades avanzadas de servicios pagos usando AWS de forma controlada y optimizada.

En resumen, los objetivos iniciales se cumplieron, dejando además un aprendizaje valioso sobre automatización, optimización y arquitectura moderna en la nube.

## 7. Referencias

Rsbuild: <https://rsbuild.rs/guide/start/>

Cloudfront + S3:

<https://docs.aws.amazon.com/whitepapers/latest/build-static-websites-aws/controlling-how-long-amazon-s3-content-is-cached-by-amazon-cloudfront.html>

<https://aws.amazon.com/es/blogs/aws-spanish/como-alojar-tu-sitio-web-estatico-en-amazon-s3-y-amazon-cloudfront/>

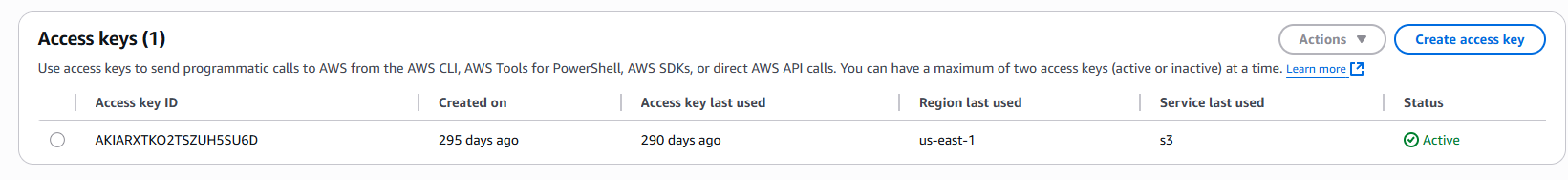
React: <https://es.react.dev/>

Typescript: <https://www.typescriptlang.org/>

Github Actions: <https://docs.github.com/es/actions>

## 8. Anexos

**Creación de llaves IAM en AWS**

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

**Creación de Bucket de S3**Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.Interfaz de usuario gráfica, Aplicación, Word

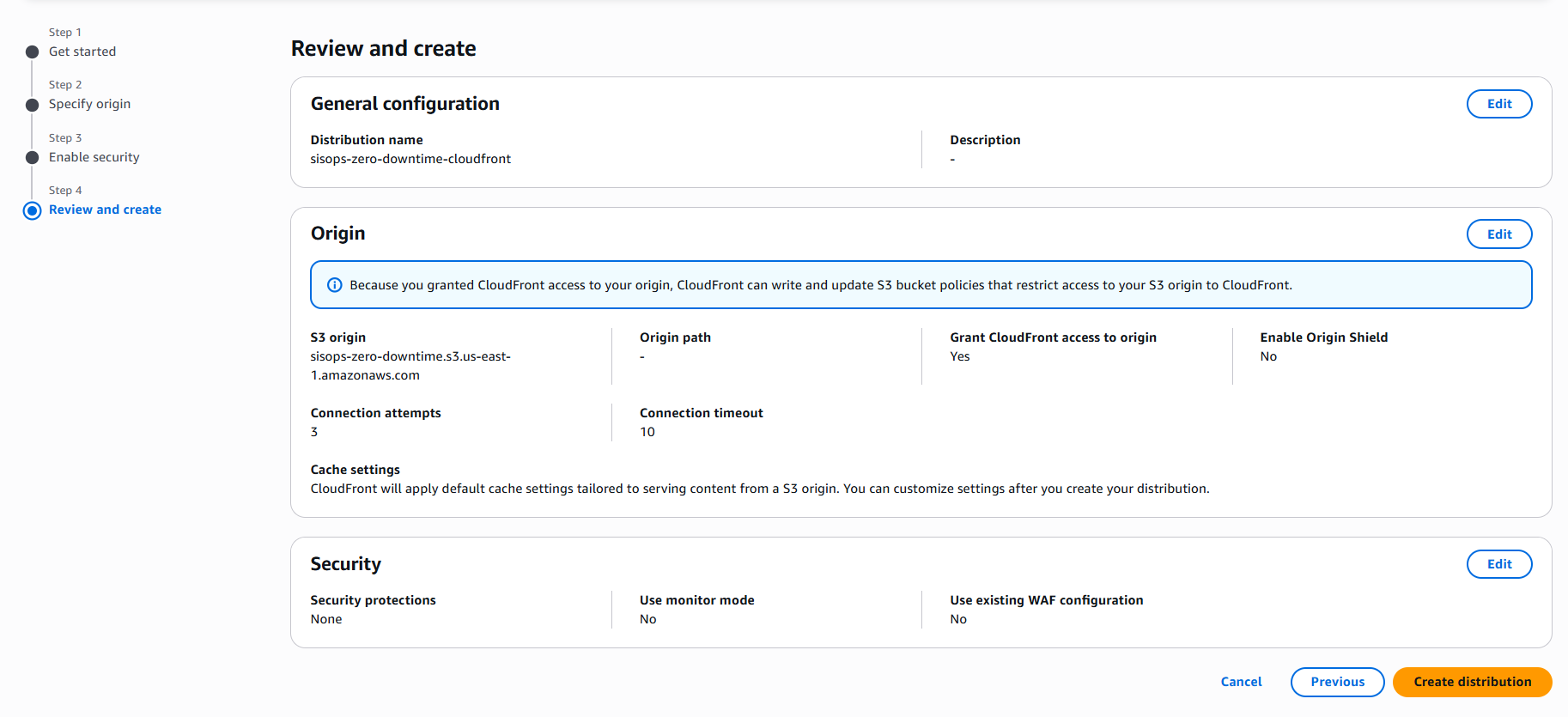
El contenido generado por IA puede ser incorrecto.

**Creación de Cloudfront**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

**Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.**

Implementación de pipeline de CI/CD con github Actions: <https://github.com/andres-calvo/sisops-proyecto-final/blob/main/.github/workflows/deploy.yml>

Texto

El contenido generado por IA puede ser incorrecto.

