Proyecto Final

Curso de Sistemas Operativos y Laboratorio

Despliegue de Frontend con Zero Downtime usando AWS S3, CloudFront y React con RsBuild

Miembros del equipo

Andres Felipe Calvo Ariza
Emanuel López Higuta

Resumen

Se propone diseñar e implementar una **pipeline de despliegue continuo** para una aplicación frontend desarrollada en React y compilada con RsBuild, que garantice **zero downtime** utilizando AWS S3 y CloudFront. La solución aprovechará versionado de assets, encabezados de cache-control, **estrategias de TTL jerárquicos (tiered TTLs)** y estrategias de invalidación de CDN, asegurando que los usuarios nunca experimenten interrupciones durante actualizaciones de la aplicación.

Introducción

- **¿Cuál es la necesidad y/o problema que aborda el desafío seleccionado?** Durante el ciclo de vida de una aplicación web, cada despliegue de nuevas versiones puede ocasionar **tiempo de inactividad** o inconsistencias en el navegador de los usuarios debido a la caché. Esto impacta negativamente la experiencia de usuario y la percepción de calidad.

- **¿Por qué es importante el desarrollo de este desafío en el contexto tecnológico actual?** Con el crecimiento de aplicaciones SPA (Single Page Application) y la adopción masiva de CDNs, es clave asegurar **disponibilidad continua** y **coherencia de contenido**. Implementar zero downtime en despliegues frontend es una práctica crítica en **DevOps** y se basa en conceptos fundamentales de sistemas operativos como operaciones atómicas, manejo de procesos y protocolos de coherencia de caché.

Antecedentes o marco teórico

- **Sistemas de archivos y operaciones atómicas**: en SO, el renombrado atómico (rename) garantiza consistencia; en S3 se replica mediante versionado en rutas y alias.
- **Caché y coherencia**: los protocolos MESI/MOESI en CPUs (cache invalidation) equivalen a invalidaciones de objetos en edge nodes de CloudFront.
- **Administración de procesos y señales**: el ciclo de vida de instancias y health-checks en AWS remite a creación, monitoreo y finalización ordenada de procesos en el kernel.
- **Namespaces y aislamiento**: versionado de archivos con hashes actúa como namespaces que evitan colisiones en caché.

Relación con temas del curso de Sistemas Operativos:

- **Protocolos de coherencia de caché**: los esquemas MESI/MOESI en CPUs invalidan líneas en caches distribuidos; análogo a las invalidaciones de objetos en edge nodes de CloudFront para asegurar consistencia.
- **Planificación y scheduling**: el scheduler del SO asigna procesos a CPUs basado en afinidad; en la CDN, el DNS geográfico y CloudFront equilibran tráfico entre edge nodes.
- **Sincronización y concurrencia**: mecanismos de locking y semáforos en SO garantizan acceso coordinado; en AWS, las invalidaciones de caché y versionado coordinan actualizaciones sin inconsistencias.
- **Gestión de políticas de tiempo de vida (TTL)**: al igual que TTL en entradas de TLB o caches de nivel L1/L2, los headers `Cache-Control` dictan la expiración y refresco de recursos en navegadores.

Objetivos (principal y específicos)

Objetivo principal Implementar un mecanismo de despliegue continuo para frontend React con RsBuild en AWS que garantice zero downtime mediante S3, CloudFront y políticas de cache-control.

Objetivos específicos

1. Configurar bucket de S3 con rutas versionadas para assets.
2. Establecer distribución de CloudFront con políticas de origen y health-checks.
3. Implementar encabezados HTTP `Cache-Control` adecuados para assets immutables y dinámicos, definiendo **tiered TTLs** en CloudFront (Default TTL, Min TTL, Max TTL) y comportamientos diferenciados por path (e.g. HTML vs. CSS/JS/images).
4. Automatizar invalidaciones de CloudFront tras cada despliegue.
5. Integrar el compilador RsBuild en un pipeline CI/CD (GitHub Actions o AWS CodePipeline).
6. Documentar y validar la experiencia de usuario sin interrupciones.

Metodología

Herramientas y tecnologías

- **Frontend**: React y RsBuild para bundling y optimización.
- **Infraestructura en la nube**: AWS S3 (almacenamiento de assets), CloudFront (CDN).
- **CI/CD**: GitHub Actions o AWS CodePipeline para automatizar despliegues.
- **Control de versiones**: Git y branch strategy (git-flow).
- **Monitoreo**: CloudWatch para health-check y métricas de latencia.

Actividades

1. **Investigación**: estudio de best practices de zero downtime y caché en AWS, incluyendo el blog de tiered TTLs en CloudFront y S3.
2. **Preparación del entorno**: creación de bucket S3 y configuración básica de distribución CloudFront con comportamientos diferenciados.
3. **Implementación de build**: integración de RsBuild en el proyecto React.
4. **Versionado de assets**: configuración de nombres con hashes.
5. **Políticas de cache-control y TTL jerárquicos**: definir headers en S3 y configurar comportamientos en CloudFront con tiered TTLs para recursos estáticos (JS/CSS/images) y dinámicos (HTML).

6. **Invalidación CDN**: scripting de invalidaciones tras despliegue.
7. **Automatización**: pipeline CI/CD que ejecute build, versionado, upload y invalidación.
8. **Pruebas**: despliegues de prueba y verificación de zero downtime y coherencia de caché.
9. **Documentación**: manual de operación y resultados de latencia, disponibilidad y aciertos de TTL.

Cronograma

Actividad	Semana 1	Semana 2	Semana 3	Semana 4
Investigación y diseño	x			
Preparación de AWS		x		
Integración RsBuild y React		x		
Versionado y cache-control			x	
Scripts de invalidación CDN			x	
CI/CD Automation			x	
Pruebas de despliegue				x
Documentación y entrega final				x

Referencias

1. AWS Documentation: *Hosting a Single Page App in Amazon S3* y *Amazon CloudFront Developer Guide*.
2. AWS Networking & Content Delivery Blog: *Host Single Page Applications (SPA) with Tiered TTLs on CloudFront and S3*.
3. ReactJS: *Building and Bundling for Production*.
4. RsBuild: *Optimizing React Builds with RsBuild*.
5. Tanenbaum, A.: *Sistemas Operativos: Diseño e Implementación* (conceptos de rename y cache-coherencia).