Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Máster Universitario en Máster Universitario en Ingeniería Informática

Trabajo Fin de Máster

Desarrollo de Aplicaciones Globales Usando Técnicas de DevOps en la Plataforma AWS

Autor(a): Manuel Esteban Carrillo Calderón
Tutor(a): Oscar Dieste Tubio

Madrid, Junio 2024

Este Trabajo Fin de Máster se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Máster
Máster Universitario en Máster Universitario en Ingeniería Informática

Título: Desarrollo de Aplicaciones Globales Usando Técnicas de DevOps en la Plataforma AWS

Junio 2024

Autor(a): Manuel Esteban Carrillo Calderón
Tutor(a): Oscar Dieste Tubio
Lenguajes y Sistemas Informáticos e Ingeniería de Software
ETSI Informáticos
Universidad Politécnica de Madrid

Resumen

Los avances en la tecnología informática y la creciente adopción de metodologías ágiles han impulsado a las empresas a adoptar nuevas formas de desarrollo y despliegue de sus aplicaciones. Hoy en día se requiere no solo de una programación robusta, sino también de una infraestructura que permita un despliegue continuo y una gestión eficiente de los recursos.

El propósito principal de este trabajo es explorar el enfoque de implementación de metodologías DevOps y la utilización de los servicios de Amazon Web Services (AWS) para superar los desafíos técnicos y operativos en el desarrollo de un producto software. Teniendo en cuenta que estos desafíos pueden variar según el contexto del producto, aquí se presentan algunos puntos clave que deben abordarse para garantizar una implementación exitosa y un mantenimiento efectivo:

- Desarrollo eficiente: diseñar y desarrollar el software con una estructura que facilite la implementación y mantenimiento. Además, implementar pruebas exhaustivas en todas las etapas del desarrollo para asegurar la calidad del software.
- Escalabilidad y rendimiento: garantizar que el software pueda manejar de manera eficiente un crecimiento en la cantidad de usuarios, la carga de trabajo y los datos sin degradación significativa del rendimiento.
- Gestión de configuración: establecer un sistema de gestión de configuración que permita controlar y gestionar cambios en el software, incluidas versiones, compilaciones y dependencias.
- Monitorización y registro: implementar herramientas y técnicas para monitorear el rendimiento y el comportamiento del software en tiempo real, así como para registrar eventos importantes que faciliten el diagnóstico de problemas.
- Despliegue y actualizaciones: establecer un proceso eficiente para el despliegue del producto y de actualizaciones que garanticen la integridad y la disponibilidad del software en todo momento.
- Compatibilidad e interoperabilidad: asegurar que el software sea compatible con diferentes plataformas, sistemas operativos y tecnologías, y que pueda interoperar con otros sistemas y aplicaciones de manera efectiva.

Con estos puntos se busca abordar tanto las necesidades de desarrollo como los aspectos críticos de operaciones y mantenimiento a lo largo del ciclo de vida del producto; y a su vez presentar una serie de herramientas y metodología de trabajo que puedan servir como base o guía para proyectos de diferentes naturalezas.

Para poder examinar en detalle cada punto, el trabajo presenta un Caso de Estudio

centrado en el desafío de una empresa ficticia que busca desarrollar una plataforma de comercio electrónico para operar a nivel global.

Abstract

Nowadays, advances in information technology and the growing adoption of agile methodologies have driven companies to adopt new ways of developing and deploying applications. It requires not only robust programming, but also an infrastructure that enables continuous deployment and efficient resource management.

The main objective of this Master's Thesis is to explore the DevOps methodologies implementation approach and the use of Amazon Web Services (AWS) to overcome the technical and operational challenges in deploying a software product. Bearing in mind that these challenges may vary depending on the context in which it is developed, here are some key points that need to be addressed to ensure a successful implementation and effective maintenance:

- Efficient development: design and develop the software with an architecture that facilitates implementation and maintenance. Further, implement exhaustive testing at all stages of development to ensure the quality of the software.
- Scalability and performance: ensure that the software can efficiently handle a growing number of users, workload and data without major performance degradation.
- Configuration Management: establish a configuration management system to control and manage changes to the software, including versions, builds and dependencies.
- Monitoring and Logging: implement tools and techniques to monitor software performance and behavior in real time, and to record relevant events to facilitate problem diagnosis.
- Deployment and upgrades: establish an efficient process for product deployment and updates to ensure the integrity and availability of the system over time.
- Compatibility and interoperability: ensure that the software is compatible with different platforms, operating systems and technologies, and that it can interoperate effectively with other systems and applications.

These points seek to address both the development needs and the critical aspects of operations and maintenance throughout the product life cycle; and in turn, provide a series of tools and methodologies that can be used as a basis or guide for projects of different characteristics.

In order to examine each point in detail, the paper will present a Study Case focused on the challenge of a fictitious company seeking to develop an e-commerce platform.

Tabla de contenidos

Capítulo 1

Introducción

Las tecnologías emergentes y las metodologías ágiles han transformado profundamente el desarrollo de aplicaciones y la gestión de infraestructura. Entre estas innovaciones, la combinación de plataformas en la nube, como pueden ser Amazon Web Services (AWS), Microsoft Azure, y Google Cloud Platform (GCP), junto con la metodología DevOps, han destacado por su capacidad para mejorar la eficiencia operativa, reducir los tiempos de despliegue y aumentar la calidad de los productos de software. Estas plataformas en la nube, con sus amplias gamas de servicios escalables y seguros, ofrecen bases robustas para la implementación de soluciones innovadoras. Por su parte, DevOps, con su enfoque en la integración continua y la entrega continua (CI/CD), promueve una cultura de colaboración entre equipos de desarrollo y operaciones, facilitando una mayor agilidad y eficiencia en los procesos de desarrollo de software.

Con esta premisa, este Trabajo de Fin de Máster (TFM) se centra en la aplicación de Amazon Web Services y DevOps en el desarrollo de aplicaciones empresariales y viene principalmente motivado por observar las tendencias actuales en el desarrollo de software y las demandas del mercado. Las empresas tecnológicas, independientemente de su tamaño y sector, se enfrentan a desafíos constantes para mejorar sus productos y servicios. La competencia global y la rápida evolución de la tecnología obligan a las organizaciones a buscar soluciones que les permitan mantenerse a la vanguardia.

En este contexto, la nube de Amazon ofrece una infraestructura flexible y escalable que se adapta a las necesidades cambiantes de las aplicaciones modernas apoyándose en sus servicios que permiten a los desarrolladores centrarse en la innovación, reduciendo el tiempo y los recursos dedicados a la gestión de infraestructura. En particular, la integración de AWS con DevOps se presenta como una solución poderosa para abordar los desafíos en el desarrollo de productos software. Esta combinación no solo optimiza los procesos técnicos, sino que también impulsa una cultura organizacional orientada a la colaboración y la mejora continua.

Debido a esta motivación se ha planteado como objetivo principal:

> Explorar y demostrar cómo la combinación de AWS y DevOps puede transformar el desarrollo de aplicaciones, mejorando la eficiencia, reduciendo los tiempos de

despliegue y aumentando la calidad del software.

Para conseguir este objetivo he aplicado una estrategia de trabajo basada en un enfoque metódico que incluyó varias etapas clave.

- Inicialmente, se definió el Caso de Estudio y se detallaron los requisitos específicos del mismo, lo cual proporcionó una base sólida para guiar el proyecto porque se especifican las funcionalidades y restricciones de lo que se va a desarrollar, asegurando que todas las necesidades y limitaciones sean claramente comprendidas y abordadas desde el inicio.
- Posteriormente, se llevó a cabo una exhaustiva investigación de las tecnologías aplicables, explorando diversas herramientas y servicios que podrían satisfacer los requisitos identificados. Por ejemplo, se evaluaron servicios en la nube como Amazon Web Services, Microsoft Azure y Google Cloud Platform, así como herramientas de automatización y CI/CD como Jenkins, GitLab CI, y CircleCI y también soluciones de contenedorización como Docker o Podman, entre otros.
- Finalmente, las tecnologías seleccionadas fueron puestas en práctica para desarrollar un sistema funcional (operativo en la nube y accesible a través de Internet) integrando los principios de DevOps y los servicios de AWS para lograr una solución que responde a las necesidades del Caso de Estudio.

Durante el desarrollo de este TFM, uno de los desafíos más significativos enfrentados fue la extensa documentación y complejidad de configuración de los servicios en la nube, lo cual dificultó la implementación inicial de algunos servicios. Sin embargo, a través de una investigación exhaustiva y la adopción de soluciones alternativas (workarounds), se lograron superar estos obstáculos para completar el proyecto. Como resultado, se ha obtenido una comprensión profunda de cómo integrar y aplicar AWS y DevOps para transformar el desarrollo de aplicaciones y se ha comprobado de forma práctica como aplicar estas tecnologías aporta eficiencia operativa y calidad al software.

Este TFM está estructurado de la siguiente manera:

- **Capítulo 1**: Introducción, que corresponde con el capítulo actual.
- **Capítulo 2**: Antecedentes. Se presentan los conceptos fundamentales de AWS y DevOps. Se exploran las características y ventajas de AWS como plataforma en la nube y se describen las metodologías y prácticas de DevOps. Además, se discute la sinergia entre estas tecnologías y cómo juntas transforman el desarrollo de software.
- **Capítulo 3**: Objetivos y Metodología. En este capítulo se presentan los objetivos específicos del proyecto y la metodología que se ha seguido para el desarrollo.
- **Capítulo 4**: Caso de Estudio. En este capítulo se presenta el Caso de Estudio específico describiendo la empresa, la aplicación que se desea implementar y los requisitos de desarrollo y operacionales que se deben cumplir.
- **Capítulo 5**: Desarrollo. En esta sección se hace un análisis de las tecnologías que se aplicarán en el desarrollo y se explica como se implementan para cumplir con los requisitos.

- **Capítulo 6**: Resultados. En este capítulo se presenta la arquitectura conseguida y como se relaciona con los requisitos iniciales planteados, además de mostrar mediante capturas de pantalla el despliegue en la práctica. También se hace una estimación del costo de la implementación.
- **Capítulo 7**: Limitaciones e impacto. Este capítulo aborda los problemas encontrados durante la implementación, y cómo estos han afectado el desarrollo del proyecto.
- **Capítulo 8**: Discusión. En este capítulo se discute cómo y en qué medida se han alcanzado los objetivos específicos planteados en el proyecto.
- **Capítulo 9**: Conclusiones. Se presentan unas conclusiones finales en forma de reflexión del trabajo conseguido.

Capítulo 2

Antecedentes

2.1. DevOps

DevOps es una metodología de desarrollo de software que integra las operaciones de desarrollo (Development) y las operaciones de sistemas (Operations) en un flujo de trabajo continuo y colaborativo. El objetivo principal de DevOps es mejorar la eficiencia, la calidad y la velocidad de la entrega de software, promoviendo una cultura de colaboración y comunicación entre los equipos de desarrollo y operaciones. Esta metodología aborda varios aspectos clave del ciclo de vida del software, incluyendo la planificación, el desarrollo, la integración, las pruebas, la entrega y la operación. [1]

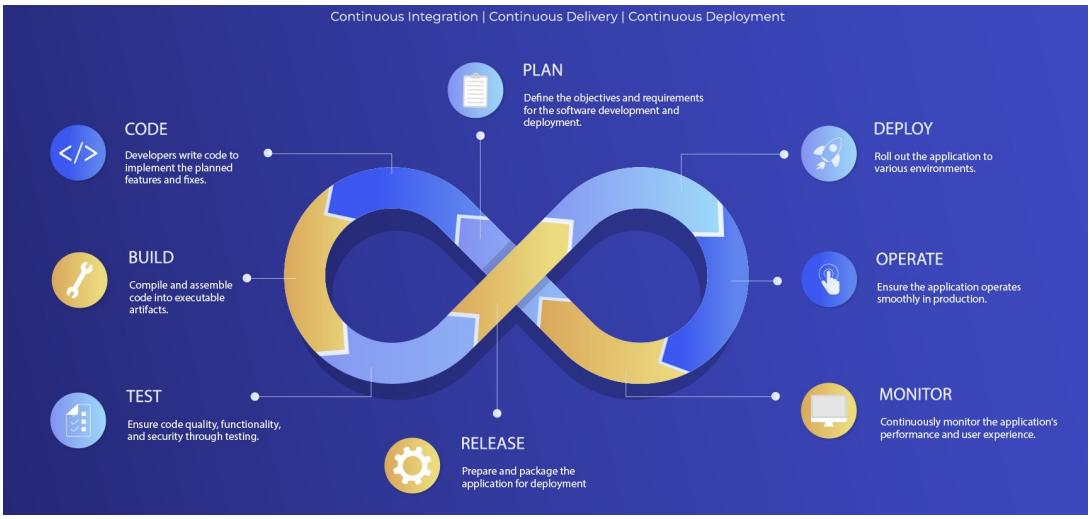

Figura 2.1: Ciclo DevOps

2.1.1. Origen

El término “DevOps” fue acuñado a finales de la década de 2000, surgiendo como una respuesta a los desafíos tradicionales en la entrega de software, donde los equipos de desarrollo y operaciones trabajaban en sitios separados, lo que a menudo resultaba en problemas de comunicación, demoras y errores en la implementación. La filosofía DevOps se basa en prácticas ágiles, que enfatizan la eficiencia y la mejora continua,

y ha evolucionado para incluir una amplia gama de herramientas y técnicas que facilitan la colaboración y la automatización [2].

2.1.2. Principios de DevOps

Los principios fundamentales de DevOps son:

- **Colaboración y Comunicación:** DevOps fomenta una cultura de trabajo colaborativo donde los equipos de desarrollo y operaciones comparten responsabilidades y trabajan juntos hacia objetivos comunes. Esta colaboración reduce los malentendidos y mejora la eficiencia.
- **Automatización:** la automatización es un componente clave de DevOps. Herramientas de automatización se utilizan para realizar tareas repetitivas, como la integración continua (CI), la entrega continua (CD), las pruebas automatizadas y el despliegue. La automatización reduce el error humano y acelera el proceso de desarrollo.
- **Integración Continua y Entrega Continua (CI/CD):** la integración continua implica la fusión frecuente de los cambios de código en un repositorio centralizado, seguido de pruebas automáticas para detectar problemas lo antes posible. La entrega continua extiende este concepto, automatizando la entrega del software a los entornos de producción de manera rápida y confiable.
- **Infraestructura como Código (IaC):** DevOps promueve la gestión de la infraestructura a través de código, lo que permite la configuración y el despliegue automáticos de los entornos de desarrollo, pruebas y producción. IaC mejora la reproducibilidad y facilita la gestión de cambios en la infraestructura.
- **Monitoreo y Registro:** el monitoreo continuo de las aplicaciones y la infraestructura es esencial para detectar y resolver problemas rápidamente. Las prácticas de registro centralizado permiten un análisis detallado de los eventos del sistema, ayudando a mejorar la calidad y el rendimiento del software.
- **Mejora Continua:** DevOps se basa en un ciclo de retroalimentación constante donde los equipos aprenden de cada despliegue y evento operativo, implementando mejoras continuas en sus procesos y herramientas. Esto se traduce en una mayor capacidad de respuesta y adaptación a los cambios en el mercado y en los requisitos del cliente

2.1.3. Beneficios de DevOps

La adopción de DevOps ofrece numerosos beneficios a las organizaciones, entre ellos:

- Mayor velocidad de entrega: al automatizar procesos y promover la colaboración, DevOps acelera el ciclo de desarrollo y despliegue, permitiendo entregas más rápidas y frecuentes.
- Mejora de la calidad del software: la integración continua y las pruebas automatizadas ayudan a detectar y corregir errores de manera temprana, mejorando la calidad del producto final.
- Mayor eficiencia operativa: la automatización reduce la carga de trabajo manual y minimiza los errores, lo que se traduce en una mayor eficiencia operativa.

- Mayor flexibilidad y escalabilidad: las prácticas de IaC y la orquestación de contenedores permiten a las organizaciones escalar sus aplicaciones y recursos de manera flexible y eficiente.
- Reducción de riesgos: el monitoreo continuo y la retroalimentación constante permiten identificar y mitigar riesgos de manera proactiva.

2.2. Cloud computing

El cloud computing, o computación en la nube, es un modelo de provisión de servicios de TI que permite acceder a recursos compartidos y configurables (como redes, servidores, almacenamiento, aplicaciones y servicios) a través de Internet. Este modelo ofrece una serie de ventajas significativas, incluyendo la reducción de costos de infraestructura, la escalabilidad rápida, y la capacidad de acceder a los recursos desde cualquier lugar y en cualquier momento [3].

Dentro del cloud computing, existen varios modelos de servicio, comúnmente referidos como **X as a Service (XaaS)**, donde "X" puede representar diferentes tipos de servicios. Los modelos más comunes se describen en las siguientes subsecciones.

2.2.1. Infrastructure as a Service (IaaS)

IaaS proporciona recursos de infraestructura de TI sobre una base de pago por uso. Estos recursos incluyen máquinas virtuales, almacenamiento, redes y sistemas operativos [4]. Como ejemplos se pueden mencionar AWS EC2, Microsoft Azure Virtual Machines o Google Compute Engine.

Entre las ventajas de estos recuros se destacan:

- Escalabilidad: permite a las empresas escalar sus recursos de manera dinámica según las necesidades.
- Costos Reducidos: elimina la necesidad de invertir en hardware físico y reduce los costos de mantenimiento.
- Flexibilidad: ofrece un control total sobre la infraestructura, permitiendo configuraciones personalizadas.

2.2.2. Platform as a Service (PaaS)

PaaS proporciona una plataforma que incluye infraestructura, servidores, almacenamiento, redes, bases de datos, y herramientas de desarrollo. Permite a los desarrolladores centrarse en la creación de aplicaciones sin preocuparse por la gestión de la infraestructura subyacente [5]. Como ejemplos se tiene Google App Engine, Horizon Cloud de VMware o RedHat.

Ventajas:

- Desarrollo acelerado: facilita el rápido desarrollo y despliegue de aplicaciones.
- Gestión simplificada: reduce la complejidad de la gestión de infraestructura y middleware.

- Colaboración mejorada: ofrece un entorno de desarrollo colaborativo para equipos distribuidos.

2.2.3. Software as a Service (SaaS)

SaaS ofrece aplicaciones de software a través de Internet, eliminando la necesidad de instalación y mantenimiento local. Los usuarios acceden a las aplicaciones a través de navegadores web [6]. Como ejemplos se tiene Google Workspace (anteriormente G Suite) o Microsoft Office 365.

Entre las ventajas se destacan:

- Acceso fácil: las aplicaciones están disponibles desde cualquier dispositivo con conexión a Internet.
- Costos predecibles: generalmente se ofrece un modelo de suscripción, facilitando la previsión de costos.
- Actualizaciones automáticas: los proveedores gestionan las actualizaciones y el mantenimiento del software.

2.2.4. Beneficios del XaaS

Como se ha visto, los distintos modelos de XaaS ofrecen varios benefcios en el desarrollo y utilización de software. De forma general se pueden destacar:

- **Reducción de costos**: al adoptar modelos de XaaS, las empresas pueden reducir significativamente los costos de capital y operativos, ya que no necesitan invertir en hardware físico ni en su mantenimiento.
- **Flexibilidad y escalabilidad**: los servicios en la nube permiten a las empresas escalar sus recursos hacia arriba o hacia abajo según las demandas del negocio, proporcionando una gran flexibilidad operativa.
- **Accesibilidad y colaboración**: los modelos de XaaS facilitan el acceso a aplicaciones y recursos desde cualquier ubicación, promoviendo la colaboración entre equipos distribuidos geográficamente.
- **Innovación rápida**: las empresas pueden innovar rápidamente, aprovechando las últimas tecnologías y servicios sin la necesidad de grandes inversiones iniciales.
- **Mantenimiento y actualización**: los proveedores de servicios en la nube gestionan las actualizaciones y el mantenimiento, asegurando que los usuarios siempre tengan acceso a las versiones más recientes y seguras de las aplicaciones.

2.3. AWS

Amazon Web Services (AWS) es una de las plataformas de computación en la nube más completas y ampliamente adoptadas en el mundo. Lanzada en 2006 por Amazon, AWS ofrece una amplia gama de servicios de infraestructura como servicio (IaaS), plataforma como servicio (PaaS) y software como servicio (SaaS). Su flexibilidad, escalabilidad y seguridad han convertido a AWS en una opción preferida para empresas

de todos los tamaños que buscan transformar y optimizar sus operaciones tecnológicas [7].

2.3.1. Origen

AWS comenzó con servicios básicos como Amazon S3 (Simple Storage Service) y Amazon EC2 (Elastic Compute Cloud), y ha evolucionado para incluir una vasta oferta de más de 200 servicios que cubren computación, almacenamiento, bases de datos, análisis, redes, movilidad, herramientas de desarrollador, herramientas de gestión, IoT, seguridad y aplicaciones empresariales. Esta evolución ha sido impulsada por la demanda creciente de soluciones en la nube que permiten a las empresas innovar más rápidamente y responder ágilmente a los cambios en el mercado [8].

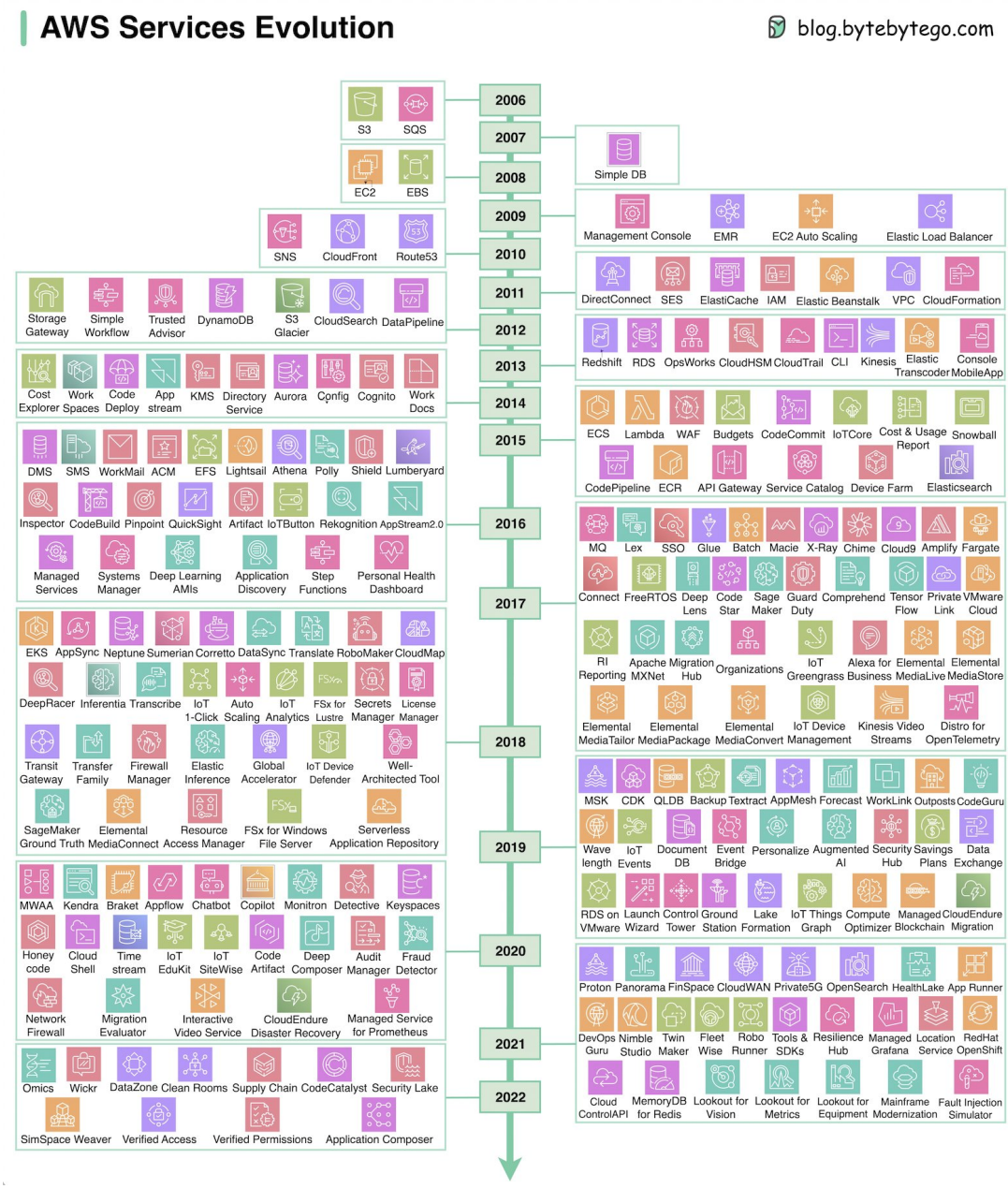

Figura 2.2: Evolución de los servicios de AWS

2.3.2. Casos de usos

Tal y como se mencionó, AWS ofrece una amplia gama de servicios en la nube que se pueden utilizar para diversos casos de uso. A continuación, se presentan algunos de los casos de uso más comunes de AWS junto con algunos servicios aplicables.

- **Almacenamiento web y entrega de contenido:** Amazon S3 (Simple Storage Service) es un servicio de almacenamiento de objetos escalable y duradero que se puede utilizar para almacenar y entregar contenido web, imágenes, videos y otros datos. Amazon CloudFront es una red de entrega de contenido (CDN) global que se puede utilizar para acelerar la entrega de contenido a usuarios de todo el mundo.
- **Cómputo en la nube:** Amazon EC2 (Elastic Compute Cloud) es un servicio de infraestructura como servicio (IaaS) que se puede utilizar para ejecutar instancias de computación en la nube. Amazon Lambda es una plataforma de computación sin servidor que se puede utilizar para ejecutar código sin necesidad de administrar servidores. Amazon ECS (Elastic Container Service) y Amazon EKS (Elastic Kubernetes Service) son servicios para ejecutar contenedores Docker en la nube.
- **Bases de datos:** Amazon RDS (Relational Database Service) es un servicio de bases de datos gestionadas que se puede utilizar para ejecutar bases de datos relacionales como MySQL, PostgreSQL y Oracle. Amazon DynamoDB es una base de datos NoSQL escalable y totalmente gestionada que se puede utilizar para almacenar datos no estructurados.
- **Análisis de datos:** Amazon Redshift es un almacén de datos en la nube que se puede utilizar para almacenar y analizar grandes conjuntos de datos. Amazon QuickSight es un servicio de análisis visual que se puede utilizar para crear y compartir cuadros de mando e informes. Amazon Athena es un servicio de consultas de bases de datos interactivo que se puede utilizar para consultar datos de Amazon S3 y Amazon Redshift.
- **Inteligencia artificial y aprendizaje automático:** Amazon SageMaker es una plataforma de aprendizaje automático totalmente gestionada que se puede utilizar para desarrollar, entrenar y desplegar modelos de aprendizaje automático. Amazon Rekognition es un servicio de inteligencia artificial que se puede utilizar para analizar imágenes y videos. Amazon Polly es un servicio de texto a voz que se puede utilizar para convertir texto en audio.
- **Internet de las cosas (IoT):** Amazon IoT Core es una plataforma de gestión de dispositivos IoT que se puede utilizar para conectar, administrar y proteger dispositivos IoT. Amazon Kinesis es un servicio de transmisión de datos en tiempo real que se puede utilizar para recopilar y procesar datos de dispositivos IoT. Amazon Machine Learning es un servicio de aprendizaje automático que se puede utilizar para analizar datos de dispositivos IoT y generar información.
- **Desarrollo de aplicaciones:** AWS Amplify es una plataforma de desarrollo de aplicaciones móviles que se puede utilizar para crear y desplegar aplicaciones móviles nativas. AWS Serverless Application Model (SAM) es una herramienta para definir y desplegar aplicaciones sin servidor en AWS. Amazon API Gateway es un servicio de gestión de API que se puede utilizar para crear, publicar y

proteger API.

Nota: aunque en la lista anterior se mencionan varios servicios de AWS, los servicios concretos que se usan en este trabajo se describen en el apartado 5.1 del capítulo de Desarrollo.

2.3.3. Beneficios de AWS

AWS ofrece numerosos beneficios que han contribuido a su adopción masiva:

- Escalabilidad y flexibilidad: AWS permite a las empresas escalar sus recursos hacia arriba o hacia abajo según las demandas del negocio, sin necesidad de grandes inversiones iniciales en hardware.
- Costo-Eficiencia: AWS opera bajo un modelo de pago por uso, lo que permite a las empresas optimizar costos al pagar solo por los recursos que realmente utilizan.
- Seguridad: AWS ofrece una infraestructura segura y servicios que cumplen con una amplia gama de certificaciones de cumplimiento. Los usuarios pueden implementar medidas de seguridad robustas para proteger sus datos y aplicaciones.
- Innovación rápida: la amplia gama de servicios y herramientas de AWS permite a las empresas innovar rápidamente, probando y lanzando nuevas aplicaciones y funcionalidades con agilidad.
- Globalización: con una infraestructura global que incluye múltiples regiones y zonas de disponibilidad, AWS permite a las empresas desplegar aplicaciones en cualquier parte del mundo, mejorando la disponibilidad y reduciendo la latencia.

  Una región de AWS es una ubicación geográfica física que agrupa múltiples centros de datos. Cada región de AWS está aislada de las demás, lo que permite a los clientes implementar aplicaciones y almacenar datos en diferentes partes del mundo. Esto ofrece ventajas significativas en términos de cumplimiento normativo, baja latencia y redundancia geográfica. Dentro de cada región, AWS tiene múltiples zonas de disponibilidad (Availability Zones o AZs). Una zona de disponibilidad consiste en uno o más centros de datos distintos, cada uno con alimentación, red y conectividad redundantes y separadas físicamente de los otros centros de datos en la misma zona de disponibilidad. Las zonas de disponibilidad en una región están interconectadas con enlaces de red de alta velocidad y baja latencia [9].

Capítulo 3

Objetivos y Metodología

3.1. Objetivos

El objetivo principal es investigar y demostrar cómo la integración de AWS y DevOps puede revolucionar el desarrollo de aplicaciones, aumentando la eficiencia, reduciendo los tiempos de despliegue y mejorando la calidad del software. Para ello, el Caso de Estudio se ha confeccionado como un escenario complejo, diseñado para explorar diversos aspectos de DevOps y proporcionar una punto inicial para examinar y aplicar múltiples técnicas y herramientas dentro de este marco. Para alcanzar el objetivo general planteado, se han definido los siguientes objetivos específicos para el Caso de Estudio:

O1. Investigar y analizar requisitos

Estudiar detalladamente los requerimientos del Caso de Estudio, identificando las necesidades específicas de la plataforma. Este análisis incluye la evaluación de los costos asociados a la implementación y operación de la plataforma, asegurando que se consideren todos los factores relevantes para una solución eficiente y sostenible.

O2. Comprender el marco de DevOps

Explorar los principios, prácticas y herramientas de DevOps y su aplicabilidad para el desarrollo de la plataforma.

O3. Establecer estrategias DevOps

Diseñar y planificar la implementación de estrategias DevOps específicas que abarquen el desarrollo, despliegue, monitoreo y mantenimiento de la plataforma. Estas estrategias estarán orientadas a optimizar el flujo de trabajo, mejorar la calidad del software y reducir los tiempos de respuesta ante cambios y actualizaciones.

O4. Utilizar servicios AWS

Evaluar y seleccionar los servicios de AWS más adecuados para respaldar los requisitos técnicos y operativos de la plataforma. Esto incluye la identificación de los servicios de computación, almacenamiento, bases de datos y herramientas de gestión que mejor se adapten a las necesidades del proyecto.

O5. Evaluar beneficios y desafíos

Analizar los resultados obtenidos de la implementación de DevOps y AWS, identificando los beneficios logrados y los desafíos enfrentados durante el proceso, proporcionando una visión completa de las oportunidades y limitaciones de estas tecnologías.

O6. Desarrollar una guía práctica

Elaborar en cierta medida una guía práctica para la adopción de servicios de AWS y DevOps por empresas o grupos de desarrollo, detallando los pasos y mejores prácticas para una implementación exitosa.

3.2. Metodología

La metodología seguida en este TFM se ha estructurado en varias fases que abarcan desde la definición de requisitos hasta la implementación y evaluación del sistema. Es importante destacar que esta metodología no se ha llevado a cabo de manera secuencial, sino iterativa, permitiendo ajustes y mejoras continuas a lo largo del proyecto. Este enfoque asegura una cobertura y comprensión de todos los aspectos del proyecto.

A continuación, se describen las fases principales de la metodología:

F1. Definición del Caso de Estudio y requisitos

En esta fase inicial, se definió claramente el Caso de Estudio y se estableció la base sobre la que enfocar el proyecto. Se llevó a cabo también una definición de requisitos, detallando tanto las necesidades de desarrollo como las operativas de la plataforma.

F2. Investigación de tecnologías aplicables

La siguiente fase consistió en una exhaustiva investigación de las tecnologías aplicables, explorando diversas herramientas y tecnologías y también las prácticas y principios de DevOps. Esta investigación incluyó la evaluación de las características y beneficios de servicios de AWS así como de diversas alternativas de terceros existentes en el mercado. El objetivo era identificar las tecnologías que mejor se alinearan con los requisitos del proyecto.

F3. Implementación del Sistema

Con las estrategias de desarrollo definidas, se procedió a la implementación del sistema. Esta fase incluyó el desarrollo y despliegue de la aplicación utilizando los servicios y herramientas seleccionados. Se establecieron pipelines para automatizar el proceso de construcción, prueba y despliegue del software, y también una arquitectura operativa para la plataforma.

F4. Evaluación de resultados y desafíos

Una vez implementado el sistema, se llevó a cabo una evaluación de los beneficios y desafíos encontrados. Se analizaron los resultados obtenidos en forma de retrospectiva de los servicios utilizados y una visión centrada en la arquitectura final del sistema, realizando también un estudio de los costos asociados, asegurando una comprensión clara de los recursos necesarios y los posibles desafíos financieros. También se identificaron y documentaron los desafíos enfrentados durante el proceso, ofreciendo una visión equilibrada de la implementación.

3.3. Plan de trabajo

El enfoque iterativo adoptado en este proyecto permitió una flexibilidad y adaptabilidad cruciales para abordar los desafíos y asegurar el éxito en cada fase. A continuación, se detallan las principales interacciones y actividades realizadas en cada ciclo:

- **Revisiones continuas**: regularmente se revisaron y refinaron los requisitos y las estrategias en función de los hallazgos y desafíos encontrados durante la implementación. Esta revisión continua permitió identificar y ajustar cualquier desalineación con los objetivos del proyecto.
- **Pruebas y ajustes**: se realizaron pruebas continuas de las configuraciones de AWS y los pipelines de CI/CD, ajustando configuraciones y estrategias según los resultados obtenidos.
- **Documentación progresiva**: la guía práctica se desarrolló de manera incremental, documentando cada fase del proyecto y ajustándola según las experiencias y lecciones aprendidas, asegurando de esta manera que todos los aspectos del proyecto estuvieran correctamente registrados y disponibles para futuras referencias.
- **Evaluación y validación**: cada ciclo iterativo concluyó con una fase de evaluación y validación para asegurar que los objetivos específicos fueran alcanzados y que las soluciones implementadas fueran efectivas.

Este enfoque iterativo y flexible permitió adaptarse a los desafíos y aprovechar las oportunidades de mejora a lo largo del proyecto, asegurando un resultado final robusto y alineado con los objetivos establecidos.

Capítulo 4

Caso de Estudio

En el panorama actual del desarrollo de productos software existen desafíos significativos en términos de agilidad, escalabilidad, confiabilidad, eficiencia y mantenibilidad. La integración de prácticas de mantenimiento y operaciones desde las primeras etapas del desarrollo es clave para superar estos desafíos.

Al incorporar el mantenimiento y las operaciones como parte integral del ciclo de vida del software se puede:

- Reducir la complejidad y el costo del mantenimiento a largo plazo.
- Mejorar la confiabilidad y la disponibilidad del software.
- Acelerar la entrega de nuevos productos y funcionalidades.
- Escalar eficientemente la infraestructura para satisfacer las demandas cambiantes.
- Optimizar el uso de recursos.

Este Caso de Estudio tiene como objetivo hacer frente a una situación que se pueda asemejar a un problema real y presentar herramientas y metodologías de trabajo que, en mayor o menor medida, sean aplicables en distintos ámbitos.

Es importante recalcar que se trabaja ante un problema ficticio y que los datos presentados son estimaciones. Sin embargo, es un punto de partida que puede servir para analizar diversos problemas reales: modificando algunos parámetros o aplicando diferentes tecnologías.

4.1. Descripción de la empresa

Dentro de este contexto, se presenta el desafío de una empresa ficticia que tiene el objetivo de desarrollar una plataforma para comercio electrónico y que pretende prestar sus servicios a nivel global. La escala y los requisitos del proyecto son comparables a los de una empresa grande, diseñada para manejar un volumen significativo de clientes y transacciones, similar a empresas establecidas en el mercado como puede ser **Etsy** (https://www.etsy.com). Etsy una plataforma global para la compra y venta de productos únicos y hechos a mano que comenzó como una startup y ha crecido

hasta convertirse en una empresa reconocida mundialmente, manejando un volumen significativo de transacciones y clientes a nivel global .

A continuación, se presenta una estimación general del volumen de clientes, transacciones y datos que se prevé manejar y que se necesitaría restaurar en caso de fallo:

- **Clientes:** Para la empresa ficticia que se plantea, con visión internacional, se podría establecer un rango de clientes entre 100,000 a varios millones. En este Caso de Estudio, se hace una estimación inicial de 500,000 clientes activos en el primer año de operaciones.
- **Transacciones:** Se puede establecer un rango de 1 a 10 transacciones por cliente al mes. Tomando un punto medio de 5 transacciones por cliente al mes, el total sería de 2.5 millones de transacciones al mes para esos 500,000 clientes.
- **Datos a restaurar:** En el peor de los casos, un volumen inicial de datos a restaurar podría estar en el rango de 1 a 5 terabytes, considerando información crítica como bases de datos de clientes, transacciones pasadas, historiales de pedidos, entre otros.
- **Trabajadores:** En un inicio el equipo sería relativamente pequeño, con alrededor de 20 a 30 personas, incluyendo desarrolladores de software, diseñadores, especialistas en marketing, personal de atención al cliente, logística y administración.

4.2. Descripción de la aplicación

La aplicación planteada es una plataforma de comercio electrónico especializada en la venta de productos de pequeños negocios. El objetivo principal es proporcionar un entorno donde los usuarios puedan explorar, descubrir y adquirir una amplia gama de productos de manera fácil y conveniente, atendiendo a sus necesidades e incorporando recomendaciones personalizadas; y a su vez, también permitir a pequeños productores publicar y vender sus productos.

La aplicación permitirá a los usuarios navegar por catálogos de productos, realizar búsquedas personalizadas, agregar artículos al carrito de compras y completar transacciones de pago de forma segura. Además, ofrecerá funcionalidades avanzadas como la gestión de cuentas de usuario, el seguimiento de pedidos en tiempo real, la gestión de devoluciones y reembolsos, y un servicio de atención al cliente eficiente.

Debido a las características de la aplicación se desarrollará con una combinación de los lenguajes de programación:

- **JavaScript con Node.js:** Node.js permite a los desarrolladores utilizar JavaScript tanto en el frontend como en el backend, lo que facilita la unificación del desarrollo y el mantenimiento del código. Esta uniformidad es una de las razones por las cuales grandes plataformas como Netflix utilizan Node.js, que lo emplea para manejar una gran cantidad de conexiones simultáneas, lo cual es esencial para su servicio de streaming en tiempo real [10].
- **Python:** es un lenguaje versátil que se utiliza ampliamente en desarrollo web, análisis de datos y machine learning. Empresas como Instagram y Spotify uti-

lizan Python debido a su sintaxis clara y legible, que facilita tanto el desarrollo como el mantenimiento del código..

En este caso, la combinación de JavaScript con Node.js y Python permite aprovechar las fortalezas de ambos lenguajes. Node.js es ideal para manejar operaciones en tiempo real y alta concurrencia, mientras que Python es perfecto para implementar la lógica de negocio más compleja y análisis de datos avanzados.

Utilizar estos dos lenguajes ofrece flexibilidad y eficiencia, ya que permite elegir la mejor herramienta para cada tarea específica. Esto optimiza el rendimiento y asegura que la aplicación pueda manejar tanto transacciones en tiempo real como algoritmos avanzados de recomendación.

Por otra parte, para llevar a cabo el desarrollo y puesta en operación de la plataforma, será necesario contar con un equipo multidisciplinario compuesto por varios profesionales que deberán trabajar de manera colaborativa e integrada. La complejidad de esta aplicación crea la necesidad de establecer una metodología de trabajo eficaz.

4.3. Requisitos de automatización del desarrollo

Los requisitos de automatización del desarrollo son fundamentales para garantizar la eficiencia, la calidad y la rapidez en la entrega del software. Los requisitos de automatización específicos para este proyecto incluyen:

1. **Gestión de Código Fuente:**
   - **Antecedente:** La colaboración entre desarrolladores y el seguimiento de cambios en el código fuente se vuelven complejos a medida que el proyecto crece.
   - **Objetivo:** Utilizar sistemas de control de versiones y una metodología apropiada para gestionar el código fuente de manera colaborativa y controlada.
2. **Integración y entrega Continua (CI/CD):**
   - **Antecedente:** Ante la complejidad y la frecuencia de cambios en el código fuente, surge la necesidad de automatizar el proceso de integración para garantizar una entrega continua y eficiente del software.
   - **Objetivo:** Automatizar la integración de cambios en el código fuente en un repositorio central, lo que facilita la detección temprana de errores y la entrega de nuevas funcionalidades de forma consistente.
3. **Despliegue Continuo:**
   - **Antecedente:** Con la necesidad de entregar nuevas versiones de software de manera regular, surge la importancia de automatizar el proceso de despliegue para reducir el tiempo de entrega y minimizar los errores humanos.
   - **Objetivo:** Automatizar el despliegue de nuevas versiones del software en entornos de desarrollo, pruebas y producción.
4. **Pruebas Automatizadas:**
   - **Antecedente:** Con el crecimiento del software, se vuelve crucial garantizar su calidad mediante pruebas exhaustivas y repetitivas.

- **Objetivo:** Desarrollar suites de pruebas automatizadas (pruebas unitarias, de integración, funcionales y de rendimiento) para validar el comportamiento del software en diferentes niveles.

5. **Análisis Estático de Código:**

   - **Antecedente:** A medida que el código base crece, se hace más difícil detectar problemas de calidad y vulnerabilidades de seguridad manualmente.

   - **Objetivo:** Utilizar herramientas de análisis estático de código para identificar automáticamente posibles problemas de calidad, vulnerabilidades de seguridad y oportunidades de mejora en el código fuente.

4.4. Requisitos de operaciones

Debido a las características de la plataforma, los requisitos operativos que se pretenden cumplir son:

1. **Alta disponibilidad:**

   - **Antecedente:** en un entorno de comercio electrónico, la alta disponibilidad es esencial para garantizar que los usuarios puedan acceder al servicio en todo momento. La interrupción del servicio podría resultar en la pérdida de ventas y la insatisfacción de los usuarios.

   - **Objetivo:** mantener un tiempo de actividad del servicio del 95 % y no más de 15 minutos de tiempo de recuperación en caso de fallo de un servidor o servicio crítico.

2. **Backup y recuperación de datos:**

   - **Antecedente:** la pérdida de datos puede tener consecuencias devastadoras, desde la pérdida de ventas hasta la pérdida de la confianza del usuario. Por lo tanto, es crucial realizar copias de seguridad periódicas de los datos y contar con un plan de recuperación de desastres efectivo para garantizar la disponibilidad continua de la información y la rápida recuperación en caso de fallo.

   - **Objetivo:** realizar copias de seguridad diarias con una ventana de recuperación de datos de máximo 4 horas.

3. **Escalabilidad y Bajos Tiempos de Respuesta:**

   - **Antecedente:** con el crecimiento del negocio y el aumento de la demanda de los usuarios, es necesario que la plataforma de comercio electrónico pueda escalar de manera eficiente para manejar la carga adicional sin degradación en los tiempos de respuesta.

   - **Objetivo:** mantener un tiempo de respuesta promedio del servidor por debajo de 1000 ms, incluso en momentos de alta demanda.

4.5. Aspectos fuera de consideración

Aunque el caso de estudio ha sido diseñado para cubrir una amplia gama de aspectos del desarrollo y operación de la plataforma, hay ciertos aspectos que no se han considerado debido a la magnitud de trabajo que representan:

1. Desarrollo de la aplicación

   El Caso de Estudio se enfoca principalmente en la infraestructura y el proceso de DevOps de integración y entrega continua (CI/CD), pero no incluye el desarrollo de la aplicación en sí misma. No se plantea el diseño y desarrollo de la funcionalidad concreta de la plataforma, como las interfaces de usuario detalladas, la respuesta del servidor, la lógica de negocio específica o la experiencia del usuario (UX).

2. Seguridad

   Aunque se menciona la seguridad en términos generales, no se han abordado en detalle aspectos críticos como las auditorías de seguridad, gestión de identidades, políticas y sistemas para gestionar el acceso a los datos y recursos de la plataforma o estrategias y tecnologías para proteger la plataforma contra amenazas avanzadas.

3. Distribución de contenido

   Un término que se planteó inicialmente pero que al final no ha podido ser abordado fue la distribución de contenido o *edge computing* que se trata de proporcionar una entrega más rápida y eficiente del contenido a los usuarios finales, especialmente donde la proximidad geográfica puede impactar mucho el rendimiento. En este caso se podría utilizar redes de distribución de contenido (CDN) y servicios de computación en el borde para reducir la latencia y mejorar la experiencia del usuario.

4.6. Arquitectura general

En un caso general, sin tener en cuenta servicios en la nube, esta aplicación debería contar con los siguientes elementos para cumplir con los requisitos establecidos:

- Data centers: instalaciones en diversas ubicaciones geográficas para asegurar alta disponibilidad y recuperación ante desastres.
- Servidores físicos:
  - Servidores de aplicaciones ejecutan el backend y los microservicios.
  - Servidores de base de datos dedicados a la gestión y almacenamiento de datos.
  - Servidores que gestionan el almacenamiento de archivos estáticos y backups.
- Redundancia y alta disponibilidad: implementación de múltiples servidores y componentes de red en configuración redundante para evitar puntos únicos de fallo.

- Sistemas de respaldo: sistemas de respaldo de datos y energía (UPS, generadores) para asegurar la continuidad del servicio en caso de fallos.
- Servidores físicos adicionales para distribuir la carga y manejar un mayor número de usuarios y transacciones sin degradar el rendimiento.

Esta arquitectura, aunque robusta y escalable, presenta varias limitaciones y desafíos que pueden afectar su eficiencia y operatividad a largo plazo:

1. Costos iniciales elevados

   Establecer data centers, adquirir servidores, y configurar sistemas de red y almacenamiento requiere una inversión significativa en infraestructura física. Además de los costos iniciales, hay gastos continuos en mantenimiento, actualización de hardware, y operación de los data centers, incluyendo energía y personal especializado.

2. Escalabilidad y flexibilidad limitadas

   La escalabilidad horizontal requiere la adquisición y configuración de nuevos servidores físicos, lo que puede llevar tiempo y no ser suficientemente ágil para responder a picos de demanda inesperados. Para manejar picos de tráfico, es necesario tener una capacidad adicional que puede estar ociosa durante periodos de baja demanda, lo que no es eficiente desde el punto de vista económico.

3. Gestión y monitoreo complejos

   La gestión de una infraestructura física requiere una administración constante y detallada, incluyendo la supervisión de la red, servidores, y sistemas de almacenamiento. Implementar un sistema de monitoreo eficaz para todos los componentes de la infraestructura puede ser complejo y costoso.

4. Redundancia y recuperación ante desastres

   Crear configuraciones redundantes en múltiples ubicaciones geográficas para asegurar alta disponibilidad puede ser extremadamente costoso y complicado de manejar. Implementar y mantener planes de recuperación ante desastres requiere recursos adicionales y puede ser difícil de probar y asegurar su efectividad.

En estas circunstancias, la adopción de servicios en la nube como AWS en lugar de una infraestructura física propia ofrece múltiples ventajas: escalabilidad elástica, reducción de costos, alta disponibilidad, agilidad para innovar y seguridad reforzada. De esta forma se libera a la empresas de las limitaciones del hardware físico, permitiéndole enfocarse en el negocio principal.

Capítulo 5

Desarrollo

En este apartado se describen los pasos seguidos para la puesta en marcha de la plataforma de comercio electrónico teniendo en cuenta los requisitos del Caso de Estudio.

A lo largo del proceso se usan varios servicios de Amazon Web Services con los que se puede interactuar mediante la interfaz gráfica o mediante la interfaz de línea de comando AWS CLI. En la mayoría de los casos se ha seleccionado la interfaz de línea de comandos porque permite trabajar directamente desde la consola de comandos del equipo de desarrollo, proporcionando una forma potente y flexible de administrar los recursos de AWS sin necesidad de utilizar la interfaz web, además de ser una opción más adecuada para la automatización. Sin embargo, en otros casos se usará la interfaz web por la facilidad y rapidez que representa si los comandos a ejecutar son complejos de construir:

```
$ aws codepipeline create-pipeline --pipeline file://pipeline.json
```

Y el contenido del archivo “pipeline.json”:

```
{
    "pipeline": {
        "name": "MyPipeline",
        "roleArn": "arn:aws:iam::123456789012:role/AWS-CodePipeline-
           Service",
        "artifactStore": {
            "type": "S3",
            "location": "my-codepipeline-bucket"
        },
        "stages": [
            {
                "name": "Source",
                "actions": [
                    {
                        "name": "SourceAction",
                        "actionTypeId": {
                            "category": "Source",
```

```
                            "owner": "AWS",
                            "provider": "CodeCommit",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "RepositoryName": "my-repo",
                            "BranchName": "main"
                        },
                        "outputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "inputArtifacts": [],
                        "roleArn": "arn:aws:iam::123456789012:role/
                           AWS-CodePipeline-Service"
                    }
                ]
            },
            {
                "name": "Build",
                "actions": [
                    {
                        "name": "BuildAction",
                        "actionTypeId": {
                            "category": "Build",
                            "owner": "AWS",
                            "provider": "CodeBuild",
                            "version": "1"
                        },
                        "runOrder": 1,
                        "configuration": {
                            "ProjectName": "my-codebuild-project"
                        },
                        "outputArtifacts": [
                            {
                                "name": "BuildArtifact"
                            }
                        ],
                        "inputArtifacts": [
                            {
                                "name": "SourceArtifact"
                            }
                        ],
                        "roleArn": "arn:aws:iam::123456789012:role/
                           AWS-CodePipeline-Service"
                    }
                ]
```

```
64              },
65              {
66                  "name": "Deploy",
67                  "actions": [
68                      {
69                          "name": "DeployAction",
70                          "actionTypeId": {
71                              "category": "Deploy",
72                              "owner": "AWS",
73                              "provider": "CodeDeploy",
74                              "version": "1"
75                          },
76                          "runOrder": 1,
77                          "configuration": {
78                              "ApplicationName": "my-codedeploy-
                                    application",
79                              "DeploymentGroupName": "my-deployment-
                                    group"
80                          },
81                          "outputArtifacts": [],
82                          "inputArtifacts": [
83                              {
84                                  "name": "BuildArtifact"
85                              }
86                          ],
87                          "roleArn": "arn:aws:iam::123456789012:role/
                                AWS-CodePipeline-Service"
88                      }
89                  ]
90              }
91          ],
92          "version": 1
93      }
94  }
```

Por ejemplo, el comando anterior configura un pipeline en AWS CodePipeline que consta de tres etapas: "Source", "Build", y "Deploy". La complejidad de este comando, con múltiples configuraciones detalladas en JSON, hace que sea más sencillo y rápido utilizar la interfaz web de AWS CodePipeline para configurar estos recursos de manera visual, evitando posibles errores en la sintaxis del JSON y facilitando la gestión de los recursos.

Ahora bien, la solución que se va a presentar es una de varias que se pueden adoptar para hacer frente a los objetivos mencionados en el apartado 3.1. Como se verá a lo largo del capítulo, en muchas situaciones se hace referencia a herramientas o tecnologías externas a AWS que son igualmente válidas y se pueden utilizar para la implementación del Caso de Estudio y productos software en general.

5.1. Tecnologías aplicables

5.1.1. Gestión de código fuente

Hay varias opciones tecnológicas para la gestión de código fuente disponibles que ofrecen diferentes características y funcionalidades como son Subversion (SVM), Mercurial o Git. Esta última, Git, es la opción elegida debido a:

- Su naturaleza distribuida, que permite un desarrollo ágil y descentralizado.
- Es rápido, eficiente y ofrece características como ramificación y fusión flexibles.
- Es compatible con una amplia gama de herramientas y servicios.
- Cuenta con una comunidad activa y una abundante documentación.

A su vez, existen varias plataformas y servicios de alojamiento y repositorios Git como GitHub, GitLab, Bitbucket, Azure Repos, entre otros. Debido a que en este trabajo se pretende explorar el entorno de desarrollo y servicios de AWS se usará **AWS CodeCommit** que es la solución que AWS presenta.

La siguiente tabla muestra una breve presentación de las herramientas mencionadas:

Herramienta	**Descripción**	**Sitio web**
AWS CodeCommit	Servicio completamente gestionado en la nube de AWS y totalmente integrado con otros servicios de AWS.	https://aws.amazon.com/codecommit/
GitHub	Plataforma integrada con una amplia gama de herramientas de desarrollo y servicios de terceros.	https://github.com/
Azure Repos	Totalmente integrado con otros servicios de Azure y herramientas de desarrollo de Microsoft.	https://azure.microsoft.com/en-us/services/devops/repos/
GitLab	Plataforma integrada con una amplia gama de herramientas de desarrollo y servicios de terceros.	https://gitlab.com/
Bitbucket	Totalmente integrado con otras herramientas de Atlassian, como Jira y Confluence.	https://bitbucket.org/

Cuadro 5.1: Herramientas de gestión de código fuente

5.1.2. Integración y entrega continua (CI/CD)

La Integración y entrega continua son prácticas de desarrollo de software de DevOps que consisten en automatizar y optimizar el flujo de trabajo de software para que los cambios se puedan implementar de manera rápida y confiable. Esto se logra mediante los siguientes pasos:

1. **Almacenamiento de código y control de versiones**: se almacena el código nuevo en un repositorio central, bajo un sistema de control de versiones, que permite un fácil acceso y seguimiento de los cambios.

2. **Generación automática de builds**: cada vez que se envía un cambio al repositorio se genera automáticamente una build, es decir, una versión en formato ejecutable del código fuente de un software.

3. **Pruebas automatizadas:** se realizan una serie de pruebas automatizadas sobre la build generada para verificar que el código no ha causado efectos secundarios negativos en las funciones existentes.

4. **Detección de errores:** si las pruebas fallan, se notifica a los desarrolladores de inmediato, lo que les permite identificar y corregir los errores en una etapa temprana del proceso de desarrollo.

En este caso se adoptará la solución presentada por AWS llamada **AWS CodePipeline.** AWS CodePipeline es un servicio de integración y entrega continua completamente administrado que permite automatizar las etapas de CI/CD para obtener actualizaciones rápidas y fiables del sistema [11].

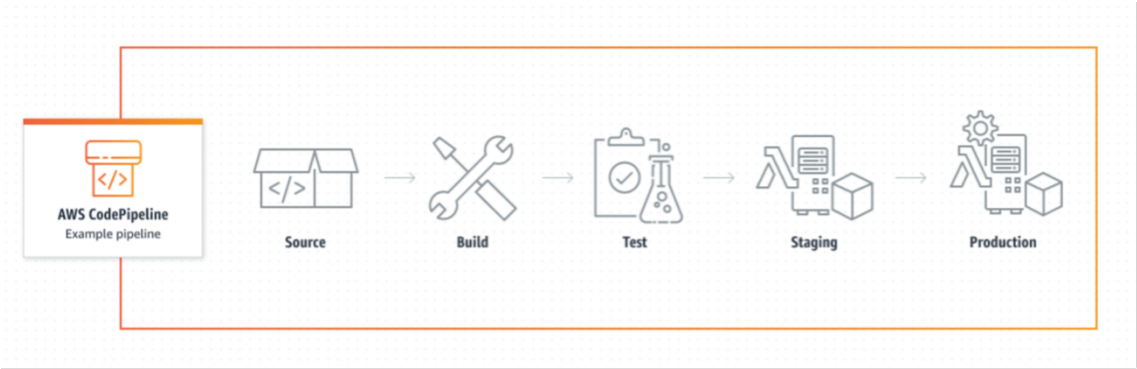

Figura 5.1: Vista general de AWS CodePipeline

Otra solución ofrecida por AWS es **AWS CodeCatalyst** que es una plataforma de DevOps integral que combina la gestión del código, la colaboración y las capacidades de CI/CD en una única interfaz [12]. Es una buena opción para tener una experiencia de usuario simplificada y unificada, sin embargo, es menos flexible y personalizable que AWS CodePipeline y lleva poco tiempo en el mercado, por lo que algunas funcionalidades están limitadas.

A pesar de elegir una solución de AWS, existen varias alternativas de gestión de flujos de trabajo que, dependiendo de la estructura de la plataforma y del equipo de trabajo en otros casos de estudio, se pueden valorar y utilizar. Entre estas alternativas se pueden destacar:

Herramienta	Descripción	Sitio web
Jenkins	Servicio de código abierto compatible con una amplia gama de tecnologías.	https://jenkins.io/
GitHub Actions	Automatiza tareas de desarrollo e implantación de software en GitHub. También se integra con otras herramientas de software.	https://github.com/features/actions
Azure DevOps	Plataforma completa de DevOps que incluye herramientas de gestión de proyectos, flujos de trabajo, repositorios de código e integración continua/entrega continua.	https://azure.microsoft.com/es-es/products/devops
GitLab CI/CD	Solución integrada de CI/CD para proyectos GitLab.	https://docs.gitlab.com/ee/ci/
Bitbucket Pipelines	Solución integrada de CI/CD para proyectos Bitbucket.	https://bitbucket.org/product/features/pipelines
Atlassian Bamboo	Otro servicio de CI/CD de Atlassian. Mientras que Bitbucket Pipelines es una opción estrictamente alojada en la nube, Bamboo ofrece una alternativa autoalojada.	https://www.atlassian.com/es/software/bamboo
Circle CI	Plataforma de CI/CD que admite una amplia gama de lenguajes de programación y herramientas software.	https://circleci.com/
Travis CI	Plataforma de CI/CD popular para proyectos de código abierto.	https://travis-ci.com/

Cuadro 5.2: Herramientas de gestión de flujo de trabajo

5.1.3. Despliegue continuo

El despliegue continuo es la práctica de automatizar el proceso de puesta en marcha de un cambio en un entorno de producción. A menudo se confunde este concepto con el concepto de entrega continua, pero es necesario recalcar que la entrega continua se centra asegurar que cada cambio está listo para la implantación en producción, no se encarga de la puesta en marcha en sí [13].

Todas las herramientas mencionadas en el apartado anterior cuentan con funcionalidad de ayuda en el despliegue continuo, sin embargo, para ayudar a la automatización dentro del entorno AWS se usará la herramienta **AWS CodeDeploy.**

Este servicio es completamente administrado y facilita el despliegue de aplicaciones en diferentes tipos de instancias proporcionadas por AWS y también en instancias locales [14].

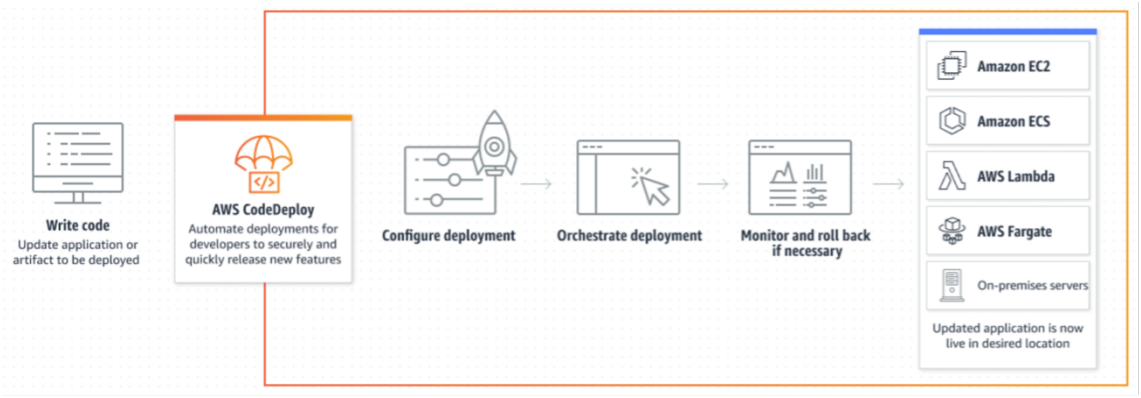

Figura 5.2: Vista general de AWS CodeDeploy

5.1.4. Pruebas automatizadas

El proceso de automatización de pruebas se incluye como parte de la integración continua y para ello es necesario construir suits o baterías de pruebas que se ejecuten cuando se produzca un cambio en el código, para de esta manera poder validar el comportamiento esperado de la aplicación en diferentes situaciones.

Por las características de la aplicación los lenguajes de programación a utilizar son JavaScript y Python, por lo cual, la suit de pruebas se construirá usando Jest como framework JavaScript y para Python PyTest debido a su potencia, facilidad de uso y presentar una solución completa para pruebas unitarias, de integración y de extremo a extremo.

5.1.5. Análisis estático de código

El análisis estático de código en DevOps es esencial para detectar errores, vulnerabilidades y problemas de calidad antes de la compilación. Ayuda a mejorar la calidad del código, garantizar el cumplimiento de estándares y prácticas, y facilita la detección temprana de problemas, lo que contribuye a un desarrollo más eficiente y seguro.

Existen varias herramientas de análisis de código que se pueden utilizar en la realización del trabajo, como pueden ser SonarQube, DeepCode o las soluciones desarrolladas por AWS que son AWS CodeGuru Security y AWS CodeGuru Profiler.

A pesar de que lo ideal para este trabajo sería optar por las herramientas de AWS, en este caso no será así porque estos servicios de momento no presentan una versión madura para el análisis de código Python ni Javascript**.** En su lugar, se analizaron dos alternativas pertenecientes a SonarSource S.A. que son SonarCloud y SonarQube debido a su amplia gama de funcionalidades, soporte para varios lenguajes de programación y facilidad de uso.

Ambas herramientas comparten características similares, con la diferencia que SonarCloud es una solución basada en la nube como SaaS, por lo que no requiere instalación ni mantenimiento, a diferencia de SonarQube que necesita instalarse en un servidor propio lo requiere más tiempo y esfuerzo para configurar. Por este motivo la herramienta seleccionada es SonarCloud.

5.1.6. Alta disponibilidad

La disponibilidad indica el porcentaje de tiempo en que un sistema software está operativo y listo para su uso. Es una relación entre el tiempo de actividad (sistema disponible) y el tiempo total (el tiempo de actividad más el tiempo de inactividad) [15].

$$\text{Disponibilidad} = \frac{\text{Tiempo de actividad esperado}}{\text{Tiempo de actividad esperado + Tiempo de actividad inesperado}}$$

Para medir la disponibilidad se definen tres parámetros:

- Tiempo medio entre errores (MTBF): Tiempo promedio entre el inicio de la operación normal y el siguiente fallo.
- Tiempo medio de reparación (MTTR): Tiempo promedio que tarda el sistema en recuperarse de un fallo.
- Tiempo medio de detección (MTTD): Tiempo promedio que transcurre entre que ocurre un fallo y la reparación (es parte del MTTR).

De esta forma se tiene:

$$\text{Disponibilidad} = \frac{\text{MTBF}}{\text{MTBF + MTTR}}$$

Comprendiendo estas métricas se pueden tomar medidas para mejorar la disponibilidad y garantizar un funcionamiento confiable de la plataforma.

La disponibilidad se puede mejorar reduciendo la frecuencia de fallos (MTBF más largo) o acortando el tiempo de reparación del problema (MTTR más corto).

5.1.6.1. Reducir la frecuencia de fallos

La frecuencia de fallos en el sistema se puede reducir de varias formas, entre las que se encuentra la realización de pruebas funcionales cada vez que se realice un cambio en el sistema. De esta manera se puede minimizar que se introduzcan defectos en la aplicación antes de la puesta en operación. También es necesario analizar el código periódicamente y en cada cambio.

Esto se consigue aplicando las tecnologías descritas de los apartados 5.1.4 y 5.1.5.

5.1.6.2. Reducir el tiempo de corrección

Para reducir el tiempo de corrección de un producto software se usará un enfoque basado en la redundancia de la aplicación. La idea es ejecutar dos o más instancias de la aplicación que se ejecutarán en paralelo para que en el caso dado de que se produzca un fallo en alguna de ellas, exista otra instancia que continúe con el trabajo. De esta manera se consigue asegurar la disponibilidad de la aplicación a pesar de los posibles fallos. Para este cometido se pueden tomar diferentes alternativas de redundancia que pueden ir desde réplicas de procesos software, servidores o instancias de máquinas virtuales duplicadas, o la ejecución del servicio en distintas zonas de disponibilidad.

En algunos casos la redundancia no será suficiente, por ejemplo, cuando el fallo sea debido a un error de codificación que se introdujo en una nueva versión. En estos casos, una solución para asegurar la disponibilidad de la aplicación es volver a una versión estable y funcional previa a la que produce el fallo, es decir, hacer un *rollback* del sistema mientras se trabaja en una solución.

A continuación, se listan los elementos necesarios para implementar un sistema redundante, desplegando y ejecutando el software en distintas zonas de disponibilidad con la posibilidad de *rollback*.

1. **Ejecución del software en un contenedor.** En el contexto del Caso de Estudio de este trabajo se opta por el uso de contenedores por diversos motivos:

   - Se desea una infraestructura más ágil y flexible.
   - Permite implementar y escalar fácilmente la aplicación en diferentes entornos.
   - Proporciona un entorno consistente y portátil.
   - Evitar conflictos de dependencias cuando se tienen varios servicios ejecutándose en un mismo entorno.
   - Permite el aislamiento de recursos para garantizar la confiabilidad de la aplicación.

   La tecnología seleccionada para la contenerización es **Docker**. Docker es una plataforma para desarrollar, implementar y ejecutar aplicaciones utilizando contenedores, de una manera ligera y portable, permitiendo que la aplicación se ejecute de forma consistente en cualquier entorno independientemente de la infraestructura subyacente, además de ser completamente compatible con los servicios en la nube de AWS [16].

   Además de Docker, existen diferentes soluciones en el mercado que se pueden utilizar según las necesidades y requisitos de un proyecto. A continuación se indican algunas de ellas:

Herramienta	Descripción	Sitio web
Podman	Solución de código abierto similar a Docker, con enfoque en la portabilidad y seguridad.	https://docs.podman.io/en/ latest/Introduction.html
Containerd	Interfaz de línea de comandos para administrar contenedores, utilizada como base por otras herramientas.	https://containerd.io/
CRI-O	Implementación de la interfaz Container Runtime Interface (CRI) para Kubernetes, enfocada en la eficiencia y el rendimiento.	https://cri-o.io/
Rocket	Contenedores ligeros y portátiles que no requieren una daemon en ejecución.	https://docs.rs/rocket
LXC	Contenedores de Linux que aíslan completamente los sistemas operativos, ofreciendo mayor seguridad.	https://linuxcontainers.org/
Singularity	Contenedores científicos optimizados para entornos HPC y de investigación.	https://docs.sylabs.io/guides/ 3.5/user-guide/introduction.html
MicroK8s	Distribución ligera de Kubernetes para entornos locales y de baja potencia.	https://microk8s.io/docs/ getting-started

Cuadro 5.3: Herramientas de ejecución de sofware en contenedores

2. **Gestión de contenedores.** Para la gestión de los contenedores se utilizarán dos de los servicios que proporciona Amazon: **Amazon Elastic Container Register (ECR) y Amazon Elastic Container Service (ECS)**. AWS ECR es un registro de contenedores totalmente administrado que ofrece una forma segura y confiable de almacenar, administrar e implementar imágenes de contenedor Docker [17].

   AWS ECS es un servicio de orquestación de contenedores totalmente administrado que se utiliza para desplegar, escalar y gestionar aplicaciones basadas en contenedores en AWS. En pocas palabras, automatiza el ciclo de vida de las aplicaciones en contenedores, desde el despliegue inicial hasta la escalabilidad y la gestión del ciclo de vida [18].

   En el contexto de alta disponibilidad, el uso de un servicio de orquestación de contenedores, y en concreto ECS, es importante por diversos motivos:

   - Automatiza tareas como la creación, configuración y administración de clústeres de contenedores. Esto permite distribuir las cargas de trabajo entre varios nodos, asegurando que la aplicación siga funcionando incluso si un nodo falla. ECS, por ejemplo, automatiza la creación de clústeres, el registro de contenedores y la asignación de recursos, liberando a los equipos de operaciones de tareas manuales y repetitivas.

- Detecta automáticamente fallos en los nodos y reinicia o reemplaza los contenedores afectados, minimizando el tiempo de inactividad y asegurando la continuidad del servicio. ECS, en particular, utiliza mecanismos como el reinicio de contenedores y la reasignación de recursos para garantizar que las aplicaciones se recuperen rápidamente de fallos y reinicios de nodos.
- Monitorea la salud de los contenedores y los reinicia o reemplaza si se detectan problemas, garantizando que solo se ejecuten contenedores sanos y funcionales. ECS proporciona métricas y registros detallados sobre el estado de los contenedores.

Además de AWS ECS, existen diferentes alternativas en el mercado que se pueden utilizar. A continuación se indican algunas de ellas:

Herramienta	**Descripción**	**Sitio web**
Kubernetes (implementación local)	Plataforma de orquestación de contenedores de código abierto, que puede implementarse de forma local en servidores o máquinas virtuales.	https://kubernetes.io/
Google Kubernetes Engine (GKE)	Plataforma de gestión de contenedores basada en Kubernetes, ofrecida por Google Cloud Platform (GCP).	https://cloud.google.com/kubernetes-engine
Microsoft Azure Kubernetes Service (AKS)	Plataforma de gestión de contenedores basada en Kubernetes, ofrecida por Microsoft Azure.	https://learn.microsoft.com/en-us/azure/aks/
Rancher	Plataforma de gestión de contenedores multiplataforma que unifica la gestión de clusters de Kubernetes en diferentes entornos.	https://www.rancher.com/
Nomad	Plataforma de orquestación de contenedores ligera y flexible, desarrollada por HashiCorp.	https://www.nomadproject.io/
Mesos	Plataforma de orquestación de contenedores de código abierto que ofrece un enfoque modular y flexible.	https://mesos.apache.org/
Docker Swarm	Plataforma de orquestación de contenedores nativa de Docker, enfocada en la simplicidad y la integración con Docker.	https://docs.docker.com/swarm

Cuadro 5.4: Herramientas de gestión de contenedores

3. **Distribución de carga de trabajo.** Se necesita incorporar un balanceador de carga para distribuir el tráfico entrante entre las distintas instancias de la aplicación. Con ello se pretende minimizar el riesgo de que un solo servicio se sobrecargue y falle. Además, si una instancia falla, el balanceador de carga auto-

máticamente redirige el tráfico a las instancias restantes, evitando tiempos de inactividad y asegurando la continuidad del servicio. Esta capacidad de detección y recuperación de fallos es esencial para mantener la alta disponibilidad de las aplicaciones críticas. La tecnología de balanceo de carga que se usará es **AWS Application Load Balancer (ALB)**, un servicio administrado por Amazon Web Services que distribuye el tráfico entrante de forma automática entre múltiples destinos en su nube, como instancias EC2, **contenedores en ECS** o aplicaciones web en Elastic Beanstalk. Su objetivo principal es mejorar la alta disponibilidad y la escalabilidad de las aplicaciones web que se ejecutan en el protocolo HTTP/HTTPS [19].

Además de AWS ALB existen diferentes soluciones en el mercado que se pueden utilizar. A continuación se indican algunas de ellas:

Herramienta	Descripción	Sitio web
Google Cloud Load Balancing	Solución de balanceo de carga completa que ofrece balanceadores de capa 4 (Network) y capa 7 (HTTP/S) para diferentes escenarios de aplicaciones.	https://cloud.google.com/load-balancing/
Azure Load Balancer	Servicio de balanceo de carga de Microsoft Azure que proporciona balanceadores de capa 4 (Basic) y capa 7 (Application) para distribuir el tráfico de aplicaciones web.	https://learn.microsoft.com/en-us/azure/load-balancer/
HAProxy	Balanceador de carga de código abierto y altamente personalizable, popular por su rendimiento y flexibilidad.	https://haproxy.org/
Traefik	Proxy inverso y balanceador de carga moderno, compatible con microservicios y arquitecturas en contenedores.	https://traefik.io/
NGINX	Servidor web y proxy inverso de alto rendimiento, que también puede utilizarse para balanceo de carga básico.	https://nginx.org/
Artillery	Herramienta de pruebas de rendimiento escalable utilizada para simular cargas de trabajo y evaluar el rendimiento de los balanceadores de carga.	https://artillery.io/

Cuadro 5.5: Herramientas de distribución de carga de trabajo

4. **Despliegue.** El despliegue de la aplicación se realizará mediante la acción combinada de AWS CodeDeploy y AWS ECS. Al combinar estos dos servicios, se obtienen una serie de beneficios en el despliegue de contenedores:

a) Despliegues más rápidas y confiables: CodeDeploy permite realizar despliegues blue/green con ECS, lo que significa que se puede actualizar la aplicación sin necesidad de interrumpir el servicio a los usuarios, teniendo por un lado un servicio activo (blue) y otro inactivo (green). Al generarse un cambio de versión se actualiza el entorno green, se valida el funcionamiento y se traspasa el tráfico del entorno blue al entorno green. En caso de que una implementación falle, CodeDeploy revierte automáticamente a la versión de software anterior.

b) Mayor seguridad: CodeDeploy permite definir permisos específicos para cada usuario o rol, lo que ayuda a controlar quién puede realizar despliegues en la aplicación.

c) Mayor visibilidad y control: CodeDeploy permite monitorizar el estado del despliegue en tiempo real y además genera un registro de auditoría detallado de cada despliegue, lo que permite rastrear quién, cuándo y qué cambios se hicieron en la aplicación.

Además de los servicios de AWS, como alternativas en el mercado destacan:

Herramienta	**Descripción**	**Sitio web**
Docker Compose	Herramienta de orquestación local para definir y ejecutar aplicaciones multicontenedor.	https://docs.docker.com/compose/
Ansible	Herramienta de automatización de código abierto que puede utilizarse para implementar y gestionar aplicaciones en contenedores.	https://www.ansible.com/
Terraform	Infraestructura como código (IaC) para definir y provisionar recursos de infraestructura en la nube, incluyendo contenedores.	https://www.terraform.io/
Jenkins	Servidor de integración continua y entrega continua (CI/CD) popular para automatizar la construcción, pruebas e implementación de aplicaciones, incluyendo aplicaciones en contenedores.	https://www.jenkins.io/

Cuadro 5.6: Herramientas de despliegue de contenedores

Por otra parte, es importante asegurar que un contenedor es saludable o si está presentando algún fallo. Para ello, AWS Application Load Balancer ofrece varias opciones de chequeo de salud para monitorear el estado y la disponibilidad de sus contenedores, asegurando que el tráfico se dirija solo a los recursos saludables:

1. Chequeo HTTP/HTTPS: Envía una solicitud HTTP/HTTPS a un puerto específico en los contenedores para verificar si responden correctamente.

2. Chequeo TCP: Envía un paquete TCP a un puerto específico en sus contenedores para verificar si están conectados y aceptando conexiones.
3. Chequeo de latencia: Mide el tiempo que tarda una solicitud HTTP/HTTPS en completarse, lo que proporciona información sobre el rendimiento del contenedor.

5.1.7. Escalabilidad

La escalabilidad se refiere a la capacidad de un sistema para adaptarse y manejar un aumento o disminución en la carga de trabajo sin comprometer su rendimiento ni funcionalidad [20]. Este atributo es crítico para determinar el éxito de cualquier proyecto de software, ya que no solo influye en la funcionalidad del sistema en cuestión, sino también en la viabilidad del negocio.

Para afrontar este desafío, se va a usar la solución de escalado automático que AWS proporciona: **AWS Auto Scaling**. Este servicio permite el escalado automático de recursos en la nube, es decir, aumenta o disminuye automáticamente la cantidad de recursos informáticos que utiliza la aplicación en función de la demanda [21].

Se ha seleccionado este servicio debido a la potencia que presenta al poder configurar el tipo de escalabilidad deseado – horizontal añadiendo más instancias de la aplicación, o vertical añadiendo más recursos (CPU, memoria, etc.) – y también porque se integra perfectamente con el resto de tecnologías utilizadas para cumplir el resto de requisitos del sistema.

En este caso, tabajará en conjunto con el servicio AWS ECS (descrito en el apartado 5.1.6) para aumentar o disminuir el número de instancias de la aplicación dependiendo de métricas como el uso de CPU, memoria o red. A su vez, las métricas se obtendrán a partir de **AWS CloudWatch**, el servicio de monitoreo y observabilidad ofrecido por Amazon. Este servicio funciona como un repositorio central para recopilar y visualizar métricas, registros y eventos de recursos de AWS [22].

5.1.8. Backup y recuperación de datos

El backup y la recuperación de datos constituyen estrategias fundamentales para garantizar la integridad, confiabilidad y accesibilidad de la información ante cualquier eventualidad que pueda comprometerla.

Por una parte, el backup o copia de seguridad, consiste en la creación de duplicados de los datos que se conservan en ubicaciones separadas y seguras, sirviendo como "redes de seguridad" en caso de que los datos originales se pierdan, corrompan o sean inaccesibles.

Por otra parte, la recuperación de datos es el proceso de restaurar información perdida o dañada. Este proceso implica identificar los datos perdidos, localizar las copias de seguridad correspondientes y utilizar herramientas y técnicas para extraer y restaurar la información de manera precisa y completa.

Para el almacenamiento de datos la plataforma web utilizará:

- **AWS DynamoDB**: servicio de base de datos NoSQL completamente gestionado que ofrece un rendimiento rápido, predecible y escalable. Proporciona acceso rápido y de baja latencia a grandes conjuntos de datos como puede ser, en este

caso, la información de los usuarios, productos o las transacciones realizadas [23].

DynamoDB se basa en una arquitectura distribuida y redundante que utiliza múltiples nodos para almacenar y procesar datos. Esto proporciona una alta disponibilidad y rendimiento. Por otra parte, DynamoDB se escala automáticamente para adaptarse a la cantidad de datos que almacena y a las solicitudes que realiza por lo que no se necesita aprovisionar o administrar infraestructura manualmente.

- **AWS Simple Storage Service (S3)**: servicio de almacenamiento en la nube ofrecido por Amazon Web Services que proporciona almacenamiento de objetos para una amplia gama de datos. En el caso específico de la aplicación que se está desarrollando aporta una solución eficaz para la entrega de contenido estático. Es útil para almacenar y entregar las imágenes u otros archivos estáticos del sitio web de forma rápida y eficiente, mejorando así el rendimiento y la experiencia del usuario [24].

  Dentro de las características de este servicio destacan:

  - Escalabilidad: se puede almacenar desde unos pocos bytes hasta petabytes de datos. S3 se escala automáticamente para adaptarse a las necesidades de la aplicación, sin necesidad de aprovisionar capacidad adicional.
  - Durabilidad: Los datos almacenados en S3 se replican en múltiples centros de datos en todo el mundo para garantizar su disponibilidad y durabilidad.
  - Seguridad: S3 ofrece una amplia gama de funciones de seguridad para proteger los datos, como cifrado de datos en reposo y en tránsito, control de acceso basado en roles y listas de control de acceso (ACL).
  - Accesibilidad: S3 ofrece una variedad de interfaces para acceder a los datos, incluyendo la consola web, la API de REST y el SDK de AWS.

Se seleccionan estos servicios debido a la perfecta integración con el resto de servicios de Amazon Web Services utilizados, sin embargo, en el mercado existen diversas soluciones como CosmoDB, MongoDB, Cassandra, Google Cloud SQL, Google Cloud Storage, Microsoft Azure Blob Storage, entre muchos otros.

Ahora bien, el objetivo principal de esta sección es cumplir con el requisito de respaldo y recuperación de datos, para lo cual existen diversas soluciones que pueden ir desde hacer un volcado manual de la base de datos y su consiguiente almacenamiento interno y restauración, hasta el uso de herramientas más avanzadas y diseñadas para automatizar este proceso. En este trabajo se utilizará el servicio **AWS Backup**, un servicio de copia de seguridad y recuperación totalmente administrado que permite centralizar y automatizar los backups [25].

A pesar de que las opciones de backup nativas de Amazon DynamoDB y Amazon S3 ofrecen soluciones para proteger los datos, se usará AWS Backup por los siguientes motivos:

1. Protección centralizada y unificada: AWS Backup permite centralizar la administración de las copias de seguridad de diversos recursos de AWS, incluyendo DynamoDB, S3 en un único servicio. Esto simplifica la gestión y el monitoreo de

las copias de seguridad, ya que no se necesita utilizar diferentes herramientas para cada servicio.

2. Automatización y escalabilidad: AWS Backup ofrece funciones avanzadas de automatización que permiten programar la creación de copias de seguridad, establecer políticas de retención y configurar alertas.

3. Recuperación rápida y sencilla: AWS Backup permite restaurar datos de copias de seguridad de forma rápida y sencilla, ya sea en el mismo recurso o en uno nuevo. Además, ofrece opciones de restauración granular que permiten restaurar objetos específicos o versiones anteriores de un objeto.

5.2. Gestión de código fuente

El punto de partida para la implementación de la plataforma es generar el código base del sitio web y alojarlo en el servicio de control de repositorios git AWS CodeCommit.

El primer paso es configurar el cliente AWS CLI para poder usar los comandos AWS desde la terminal del equipo de desarrollo. Para esto, se puede seguir la guía que Amazon proporciona en su web [26]. En resumen, consiste en descargar la última versión el módulo AWS CLI y configurar las credenciales del usuario que usará los servicios.

A continuación, para crear el repositorio dentro de la plataforma de AWS es necesario ejecutar el siguiente comando en la terminal:

```
$ aws codecommit create-repository --repository-name "repo\_name"
```

En este caso, como *repo_name* se ha utilizado **“tebimart”** (nombre pensado para la plataforma de e-commerce). Para verificar que se ha creado se puede consultar la interfaz web de AWS o ejecutar el comando:

```
$ aws codecommit list-repositories
```

```
esteban@mackbook:~ $ aws codecommit list-repositories
{
    "repositories": [
        {
            "repositoryName": "tebimart",
            "repositoryId": "f3c9af69-36ad-46ae-8171-63753b748706"
        }
    ]
}
```

Figura 5.3: Lista de repositorios por comando

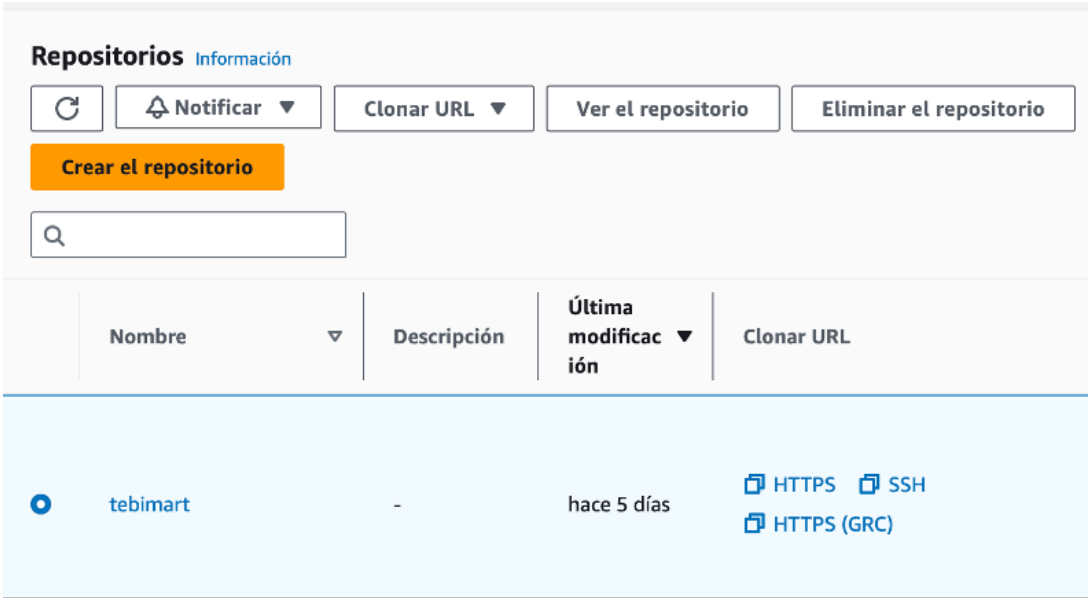

Figura 5.4: Lista de repositorios en interfaz web

Una vez se ha creado el repositorio, se prepara una versión inicial de la plataforma web con un código bastante simple que permite arrancar un servidor con node.js y visualizar vía web una interfaz con una serie de productos estáticos. Este código se usa como base para implementar sobre él los requisitos del producto. La siguiente imagen muestra la web inicial a la que da acceso el código base.

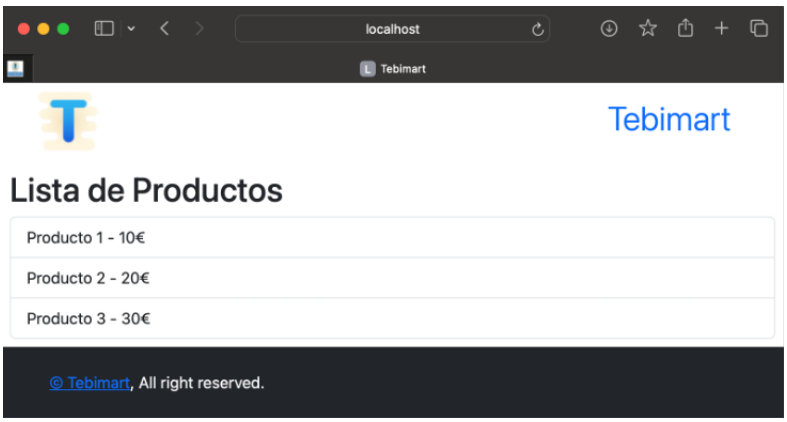

Figura 5.5: Página principal de la plataforma

El código de la plataforma se aloja en el repositorio previamente creado haciendo uso de los comandos Git habituales:

- git clone: para tener una copia del repositorio local del repositorio remoto.
- git add: para añadir los ficheros necesarios.
- git commit: para confirmar los cambios.
- git push: para enviar los commits locales al repositorio remoto.

En este primer paso se han creado dos ramas en el repositorio: una para tener la versión estable y de producción de la plataforma “main”, y otra usada para la fase de desarrollo “develop”.

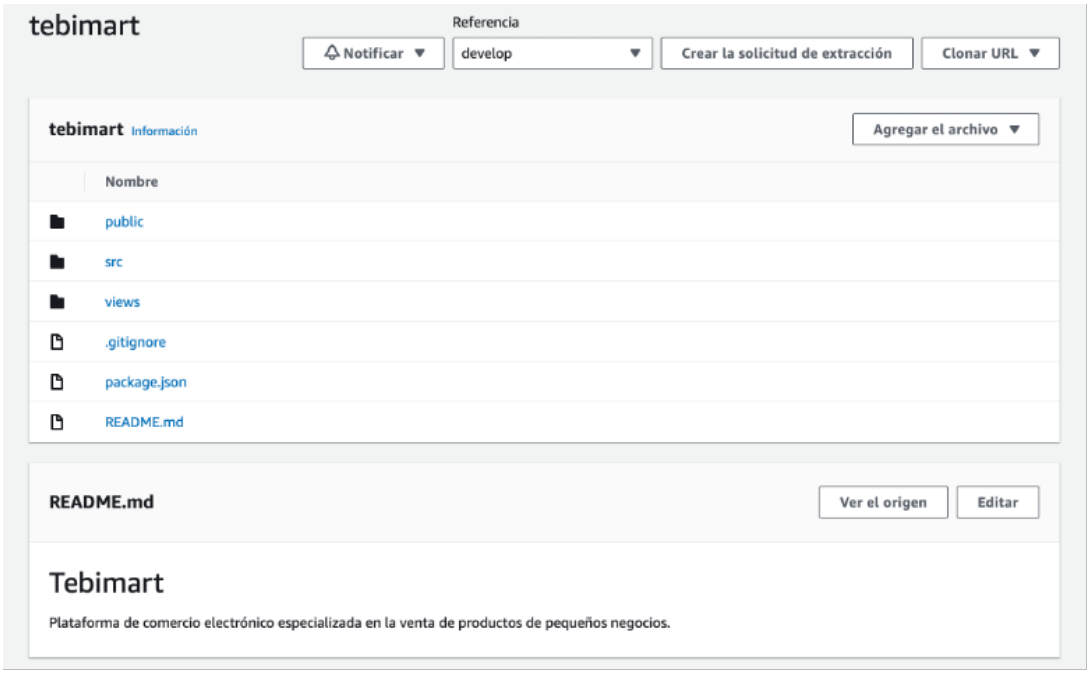

Figura 5.6: Vista de la rama ’develop’ del repositorio

5.3. Integración, entrega y despliegue continuo (CI/CD)

La implementación de la integración y entrega continua, sobre el código base preparado se lleva a cabo con el servicio AWS CodePipeline. Con este servicio se va a preparar un “pipeline” o flujo de trabajo que se ejecute cuando se produzca un cambio en el código de la aplicación.

5.3.1. Flujo de trabajo

La siguiente imagen muestra el flujo que se seguirá a través del pipeline:

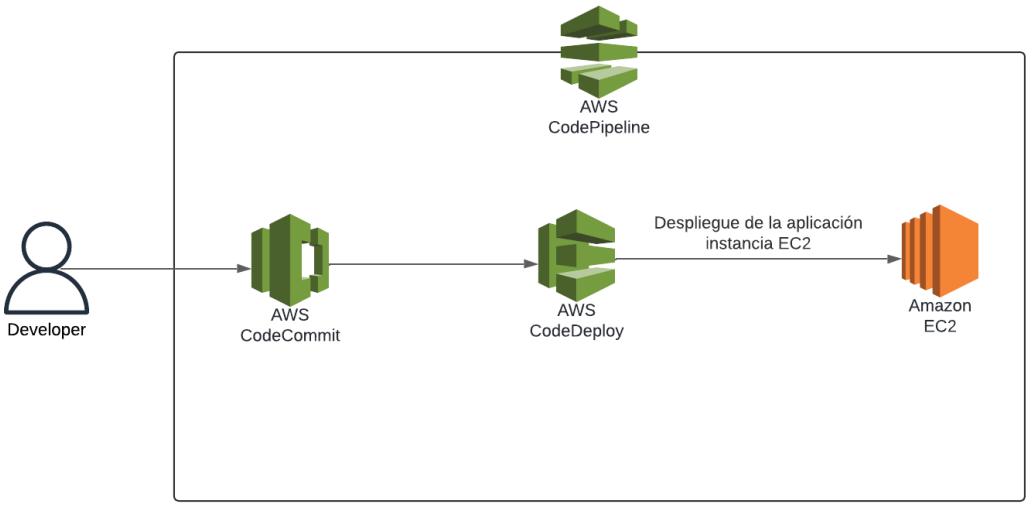

Figura 5.7: Flujo de trabajo inicial con AWS CodePipeline

Como se observa, el flujo es sencillo por ahora: se inicia cuando el desarrollador realiza un cambio y actualiza el repositorio de AWS CodeCommit y finaliza con el despliegue de la aplicación realizado por el servicio AWS CodeDeploy en una instancia de tipo EC2.

En este caso, el uso de EC2 con AWS CodeDeploy ofrece varias ventajas en términos de sencillez y facilidad de uso:

- Integración con CodeDeploy: el proceso de despliegue de aplicaciones en instancias EC2 no presenta complejidad con AWS CodeDeploy.
- Sencillez de configuración: EC2 proporciona instancias virtuales escalables y personalizables que pueden ejecutar aplicaciones web y servicios, lo que facilita la implementación de diferentes tipos de aplicaciones.

5.3.1.1. Implementación

La construcción del pipeline se ha realizado tomando como base la guía descrita por el equipo de Amazon en su web [27]. Para construir el flujo es necesario primero configurar por separado cada componente, y luego asociar todos usando AWS CodePipeline. Los componentes que se necesitan configurar son:

1. Instancia AWS EC2
2. Servicio de despliegue AWS CodeDeploy

5.3.1.2. Configuración de instancia EC2

Una instancia EC2 es esencialmente una máquina virtual en la nube que permite a los usuarios ejecutar aplicaciones y procesos computacionales en el entorno de AWS [28]. En este caso la puesta en marcha de la aplicación se ha hecho a través de la consola de comandos con los siguientes pasos:

1. Crear un par de claves (Key Pair) para la instancia. Este par de claves se utiliza para autenticar y acceder de forma segura a las instancias a través de SSH o RDP, proporcionando un control de acceso estricto.

```
$ aws ec2 create-key-pair \
    --key-name ec2-tebimart_KeyPair \
    --query 'KeyMaterial' \
    --output text > MyKeyPair.pem
```

   El comando anterior genera una clave pública y una clave privada. La clave pública se guarda en AWS y la clave privada se guarda en el dispositivo que ejecuta el comando.

2. Crear grupo de seguridad para la instancia. El grupo de seguridad permite controlar el tráfico de red que entra y sale de la instancia EC2. Con los siguiente comandos se crea el grupo de seguridad y se especifican las reglas que se deben cumplir. Aquí se permite el tráfico de HTTP y HTTPS estándar y se habilita también el puerto 3000 que es el usado por el servidor de la aplicación:

```
$ aws ec2 create-security-group \
    --group-name ec2-tebimart_SG \
    --description "Security group for EC2-tebimart instance"
```

```
$ aws ec2 authorize-security-group-ingress \
    --group-id $SG_ID \
    --ip-permissions \
    IpProtocol=tcp,FromPort=80,ToPort=80,IpRanges='[{CidrIp=0.0.0.0/0,Description="HTTP
        from anywhere"}]' \
    IpProtocol=tcp,FromPort=443,ToPort=443,IpRanges='[{CidrIp=0.0.0.0/0,Description="
        HTTPS from anywhere"}]' \
    IpProtocol=tcp,FromPort=3000,ToPort=3000,IpRanges='[{CidrIp=0.0.0.0/0,Description="
        Tebimart app traffic"}]'
```

3. Lanzar instancia EC2. Se utiliza el siguiente comando para poner en marcha la instancia:

```
$ aws ec2 run-instances \
    --image-id ami-02d7fd1c2af6eead0 \
    --count 1 \
    --instance-type t2.micro \
    --key-name ec2-tebimart_KeyPair \
    --security-group-ids ec2-tebimart_SG_ID
```

   –image-id ami-02d7fd1c2af6eead0: Esta opción especifica el ID de la imagen de máquina de Amazon (AMI) que se utiliza para lanzar la instancia. En concreto el id corresponde a una máquina con un sistema operativo propietario de Amazon y basado en Linux incluido con el nivel gratuito de AWS, suficiente por el momento.

   –count 1: Número de instancias a lanzar. En este caso, solo 1.

   –instance-type t2.micro: La t2.micro es una micro instancia de nivel libre con recursos limitados de CPU y memoria. Por el momento es suficiente con este tipo de instancia.

–key-name ec2-tebimart_KeyPair: Par de claves creado para acceder a la instancia.

–security-group-ids ec2-tebimar_SG_ID: Esta opción especifica el ID del grupo de seguridad creado. Para consultar el id del grupo de seguridad se usa el comando:

```
$ aws ec2 describe-security-groups \
    --group-names ec2-tebimart_SG \
    --query 'SecurityGroups[*].GroupId'
```

4. Instalar agente CodeDeploy. Es imprescindible instalar un agente de CodeDeploy en la instancia para que el servicio AWS CodeDeploy pueda acceder a la máquina y desplegar el software. Para esto se ha seguido paso por paso la guía disponible en la web de Amazon [29].

5.3.1.3. Configuración de AWS CodeDeploy

Para configurar el servicio AWS CodeDeploy se necesita:

1. Crear un rol CodeDeploy. Se necesita crear un rol especial que permita al servicio AWS CodeDeploy realizar acciones a nombre del usuario propietario de la cuenta. El rol se puede crear mediante el comando:

```
$ aws create-role \
    --role-name CodeDeployRole \
    --assume-role-policy-document file://AWSCodeDeployRole.json
```

Debido a que es necesario tener el documento .json de la política AWSCodeDeployRole (que implica abrir la sesión en la consola, buscar el código y descargarlo u obtener su identificador), este proceso de creación se ha hecho a través de la interfaz web dentro del panel de IAM que resulta inmediato.

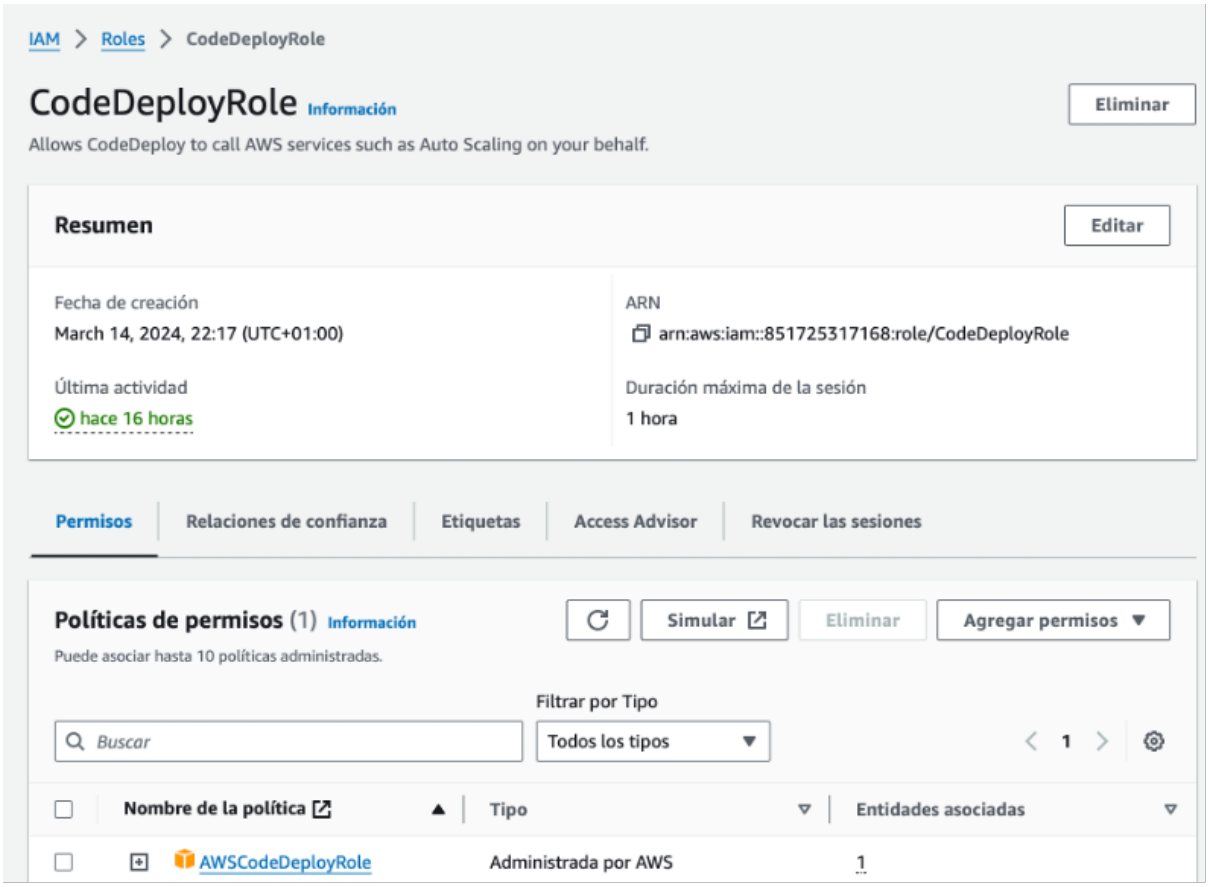

Figura 5.8: Detalles del CodeDeploy role creado

2. Crear una aplicación en CodeDeploy. Una aplicación es un recurso que contendrá el software de la plataforma. Para crear la aplicación se ejecuta el comando:

```
$ aws deploy create-application \
    --application-name tebimart-app \
    --compute-platform Server
```

   En el comando se indica el nombre de la aplicación y la plataforma en la que se va a desplegar. En este caso la plataforma es “Server” que se corresponde con instancias de tipo EC2.

3. Configurar el archivo de especificación de la aplicación (appspec.file). Este archivo administra dónde y cómo se realiza el despliegue. Se debe incluir junto al código fuente en el repositorio CodeCommit.

tebimart / appspec.yml Info

```
1  version: 0.0
2  Resources:
3    - TargetService:
4        Type: AWS::ECS::Service
5        Properties:
6          TaskDefinition: "arn:aws:ecs:us-east-1:851725317168:task-definition/tebimart-task-def"
7          LoadBalancerInfo:
8            ContainerName: "tebimart-ecs-container"
9            ContainerPort: 3000
10         PlatformVersion: "LATEST"
```

Figura 5.9: Archivo de especificación de la aplicación

Lo que se indica es el directorio de instalación de la aplicación, los permisos y las acciones a ejecutarse (hooks). Adicionalmente, hay que incluir en el repositorio varios scripts que son necesarios para el deploy:

- before_install.sh: instalar las librerías necesarias para la ejecución de la aplicación.
- after_install.sh: prepara la aplicación para ser ejecutada, por ejemplo, instalando las dependencias que se necesiten.
- application_start.sh, application_stop.sh, validate_service.sh: ejecutan el arranque, parada y comprobación de la aplicación, respectivamente.

4. Crear un grupo de despliegue en CodeDeploy. Este grupo define la configuración del despliegue. Se usa el siguiente comando AWS:

```
$ aws deploy create-deployment-group \
    --application-name tebimart-app \
    --deployment-group-name TebimartDeploymentGroup \
    --service-role-arn arn:aws:iam::user_id:role/CodeDeployRole \
    --deployment-style deploymentType=IN_PLACE, deploymentOption=WITHOUT_TRAFFIC_CONTROL
        \
    --ec2-tag-filters Key=Name,Value=Ec2-tebimart, Type=KEY_AND_VALUE
```

Se configura el grupo de despliegue para la aplicación CodeDeploy usando el rol creado en el paso 1 y seleccionando la instancia en la que se cargará la aplicación.

5.3.1.4. Configuración de AWS CodePipeline

La configuración del pipeline se puede realizar utilizando el siguiente comando:

```
$ aws codepipeline create-pipeline \
    --cli-input-json file://PipelineConfiguration.json
```

La configuración del archivo json puede resultar compleja debido a la gran cantidad de parámetros que hay que indicar, por ese motivo, la configuración y creación del pipeline se ha realizado con la interfaz gráfica de AWS. En concreto hay 4 pasos para la creación del pipeline:

1. Configuración de la canalización
2. Configuración del origen del código

3. Configuración de la compilación
4. Configuración del despliegue

Configuración de la canalización

El primer paso consiste en indicar la configuración inicial de la canalización, esto es, indicar el nombre, el tipo, el modo de ejecución y asignar un rol para el servicio. La siguiente imagen muestra la configuración asignada, con una descripción de cada parámetro de la configuración:

Figura 5.10: Configuración inicial del pipeline

Configuración del origen del código

En este paso se indica el proveedor del código fuente (AWS CodeCommit), la rama que se va a monitorear, el método trigger de la canalización y el formato del artefacto de salida:

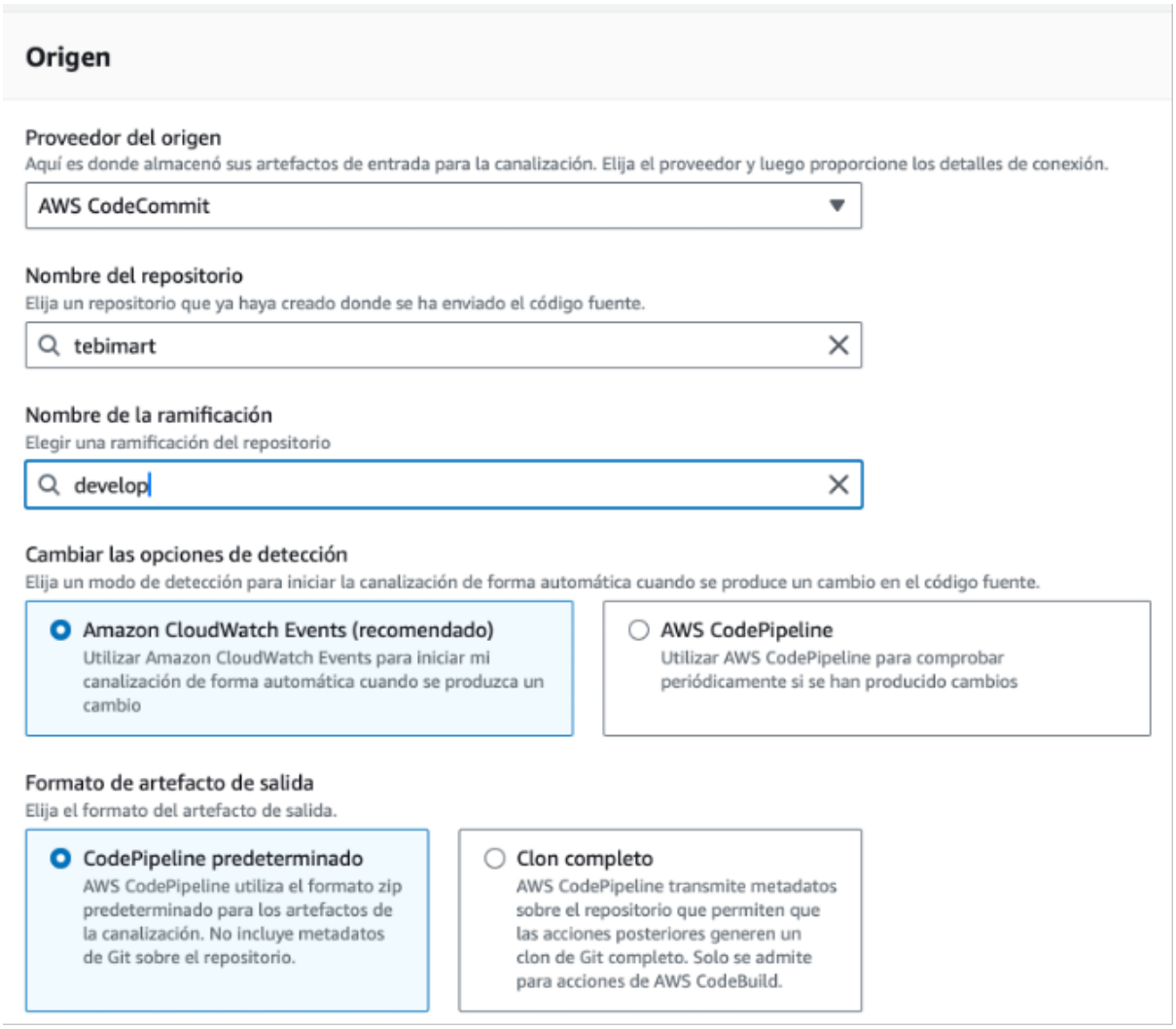

Figura 5.11: Configuración del origen del código

Configuración de la compilación

El código actual de la aplicación no requiere compilación, por lo que se salta esta etapa hasta más adelante (la configuración de la canalización se puede editar en cualquier momento). Sin embargo, hay que destacar que aquí se puede utilizar como servicio de compilación herramientas como AWS CodeBuild o Jenkins.

Configuración del despliegue

En esta última etapa se indica que el despliegue se realizará por medio del servicio AWS CodeDeploy configurado anteriormente.

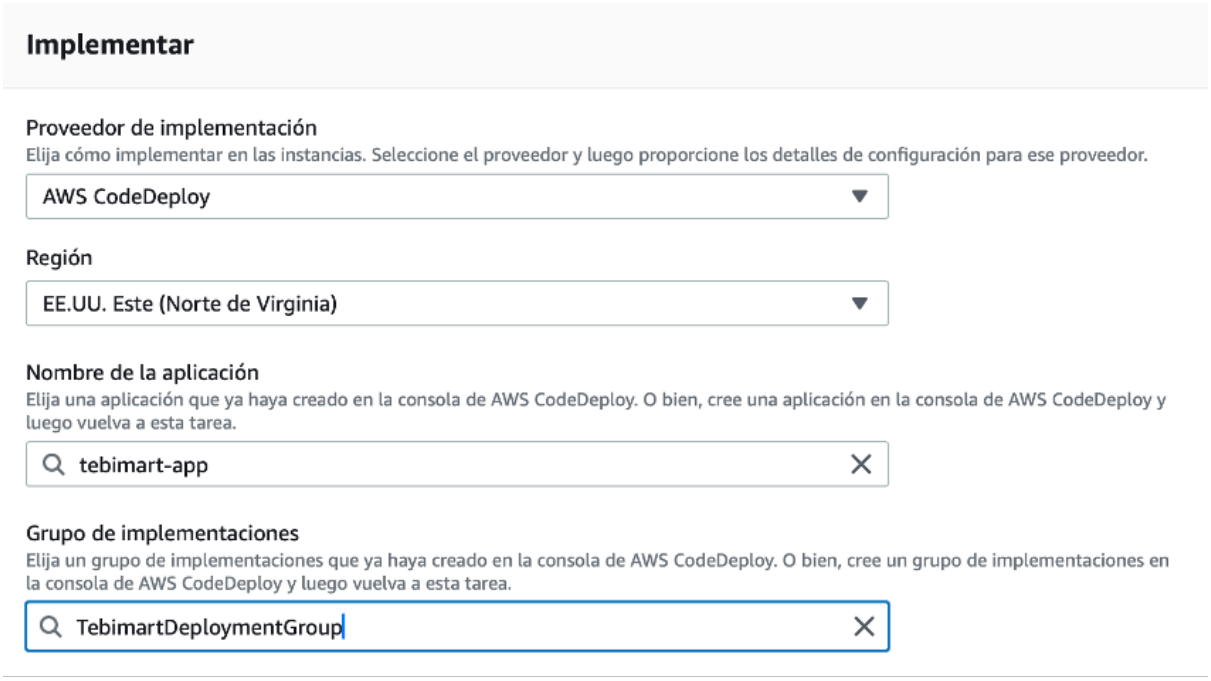

Figura 5.12: Configuración del servicio de despliegue

Con estos pasos se tiene listo el pipeline que se ejecuta por primera vez cuando se crea y las próximas veces cuando se produce un cambio en el código o bajo algún evento definido en AWS CloudWatch.

5.3.2. Resultados de la configuración

Mediante comandos de AWS CLI se puede monitorizar el estado del flujo de trabajo, pero debido a que la salida es en texto plano y confusa en cierto modo (véase figura 5.15) se prefiere comprobar el estado a través de la interfaz web. La figura 5.13 y figura 5.14 muestran el proceso de ejecución del pipeline y a su vez el estado exitoso de la toma de código y despliegue. En este punto cabe mencionar que la primera vez el despliegue no fue exitoso debido a un fallo en los scripts de despliegue mencionados en el apartado 5.3.1.3. La figura 5.17 muestra el historial de despliegues donde está reflejada esta situación.

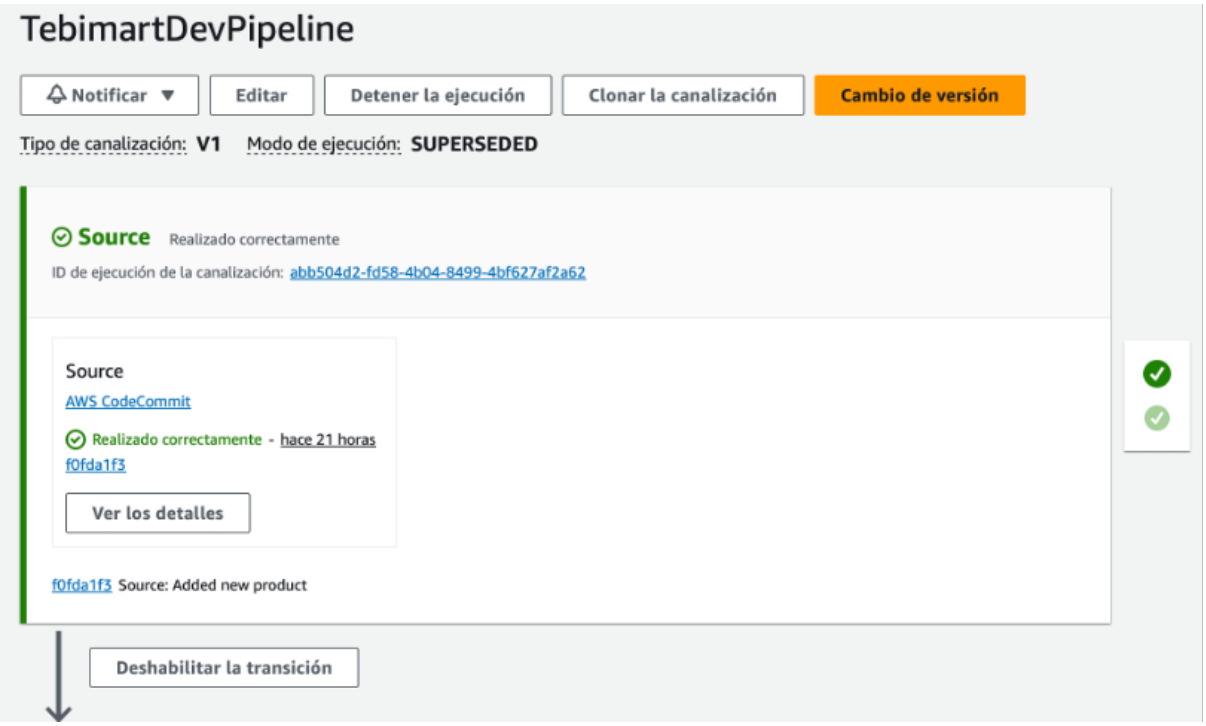

Figura 5.13: CodePipeline - captura de código

Figura 5.14: CodePipeline - despliegue de software

Figura 5.15: Listado de ejecuciones de CodePipeline por la línea de comandos

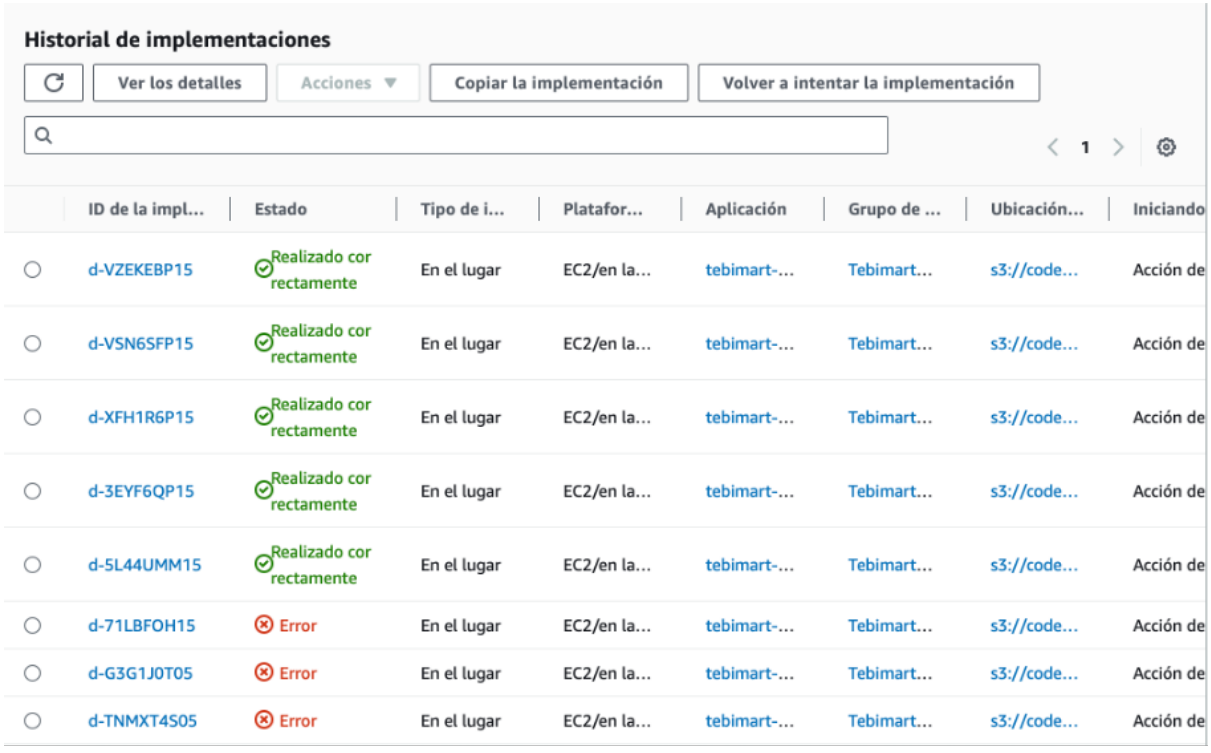

Figura 5.16: Listado de ejecuciones de CodePipeline por interfaz web

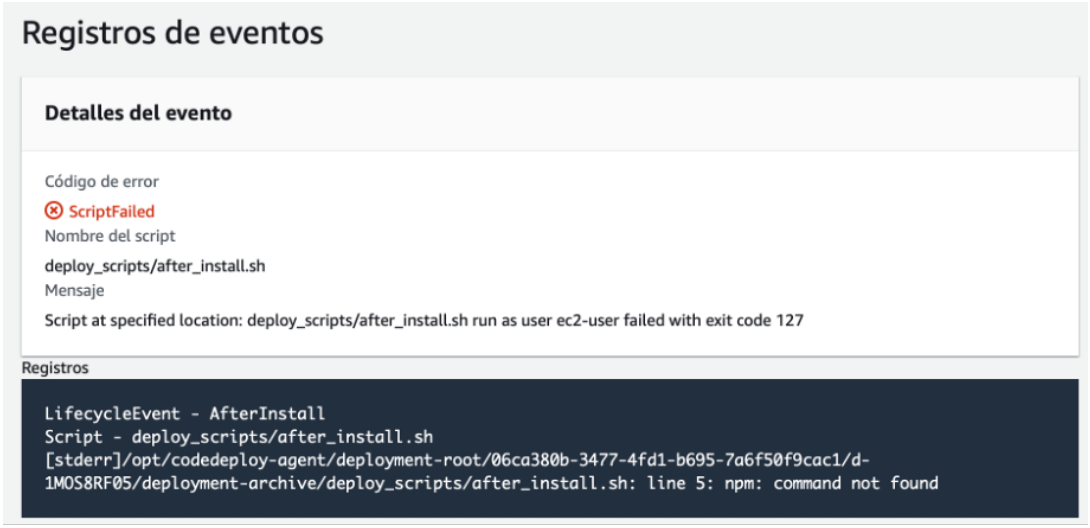

Figura 5.17: Registro de un evento fallido

A partir de ahora, el software se desplegará y ejecutará en la instancia EC2 creada, estando accesible a través de la dirección IP de dicha instancia bajo el puerto 3000. Para comprobar esto se han realizado algunos cambios en la interfaz y se ha accedido a la plataforma desde un dispositivo móvil:

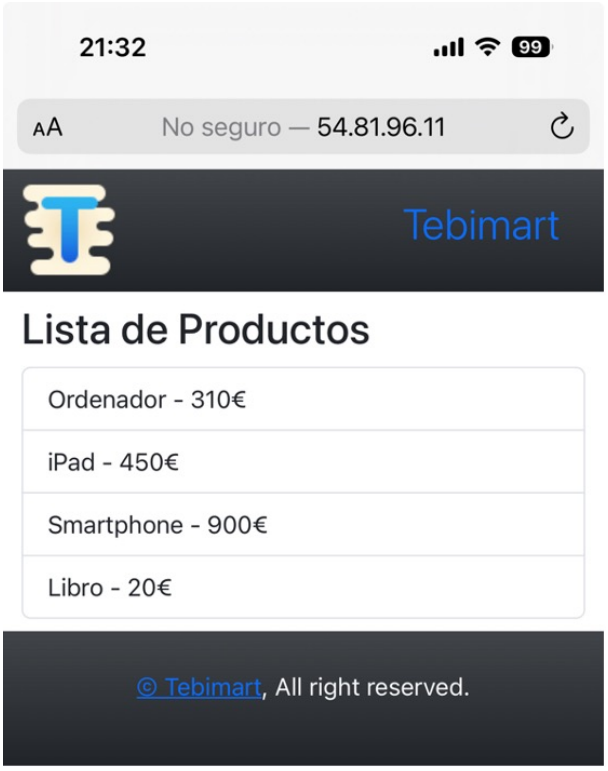

Figura 5.18: Vista de la plataforma desde un smartphone

5.4. Análisis de código y pruebas automatizadas

Como parte esencial de los requisitos de automatización del desarrollo es necesario incluir en el flujo de trabajo la capacidad de que se realicen pruebas automáticas sobre el código existente y también que se realice un análisis estático del código. Teniendo en cuenta las tecnologías seleccionadas para esta tarea (comentadas en el apartado 5.1) el flujo será el siguiente:

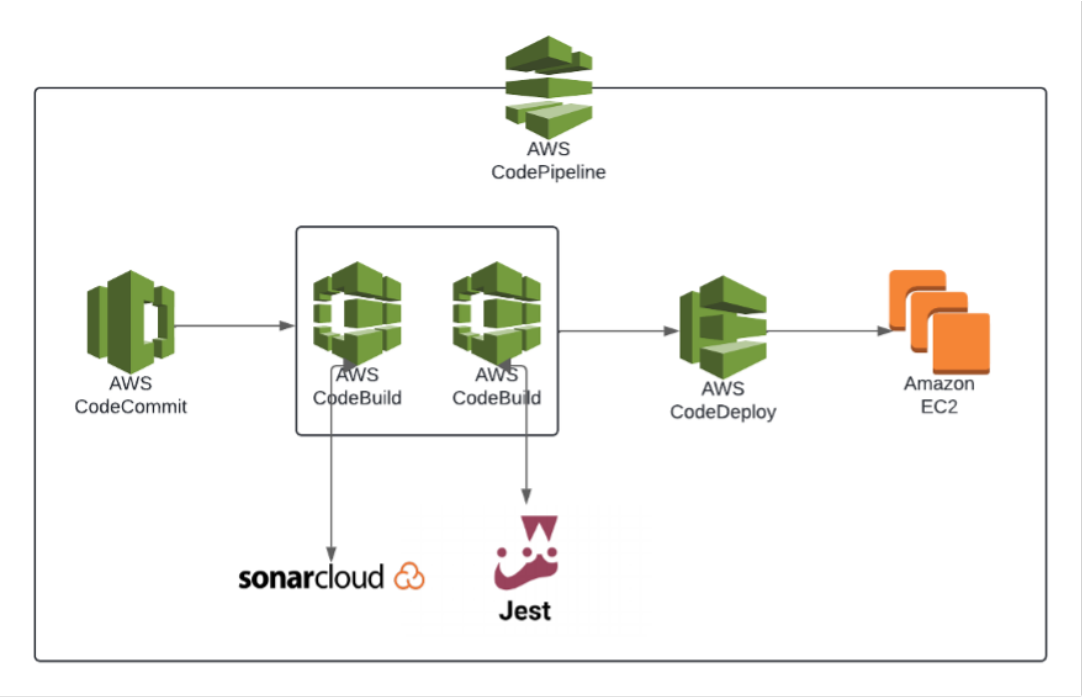

Figura 5.19: Flujo de trabajo AWS integrando revisión de código

5.4.1. Integración de pruebas automatizadas en CodeBuild

La integración de pruebas automatizadas en el flujo de trabajo consta de dos partes que son la generación del código de pruebas y la creación de un proyecto de AWS CodeBuild que ejecute dichas pruebas.

5.4.1.1. Generar código de pruebas

En el caso de la aplicación, cuyo código es bastante simple y mayormente escrito en Javascript, las pruebas se codifican utilizando el framework de pruebas unitarias Jest. Sin embargo, independientemente del lenguaje de programación o framework de pruebas utilizado, es necesario:

1. Actualizar el repositorio para añadir los ficheros de pruebas. Se crea un nuevo directorio llamado “tests” donde se incluye el código de las pruebas. Para el caso de la app se generan dos archivos sencillos que comprueban la correcta configuración del middleware y las rutas de la aplicación.
2. Configurar el archivo de especificación de build (buildspec_test.json). Este archivo define el proceso que se llevará a cabo en el proyecto de CodeBuild. Se incluye junto al código fuente en el repositorio CodeCommit:

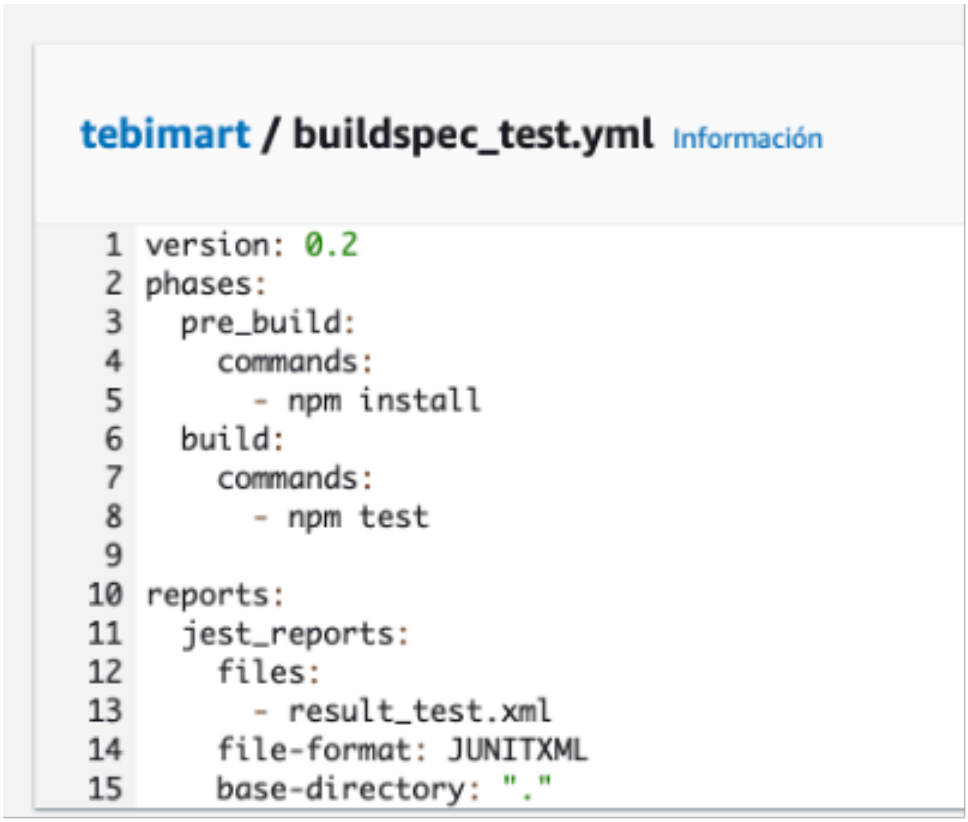

Figura 5.20: Archivo buildspec_test.json

En el archivo se indican los comando a realizar para ejecutar las pruebas. En el caso de la aplicación: instalar las dependencias necesarias con *npm install* y ejecutar las pruebas con *npm test*. También se indica el formato del reporte de pruebas que se generará.

5.4.1.2. Configuración de AWS CodeBuild

El proyecto de AWS CodeBuild encargado de las pruebas se configura con los siguientes pasos:

1. Creación de un rol CodeBuild: Se necesita crear un rol especial que permita al servicio AWS CodeBuild realizar acciones a nombre del usuario propietario de la cuenta. El rol se puede crear mediante el comando:

```
$ aws iam create-role \
    --role-name codebuild-TebimartTest-service-role \
    --assume-role-policy-document file://CodeBuildRolePolicy.json
```

El archivo de políticas .json se construye dando los permisos necesarios para asumir el rol de CodeBuild:

```
1 {
2     "Version": "2012-10-17",
3     "Statement": [
```

```
4           {
5               "Effect": "Allow",
6               "Principal": {
7                   "Service": "codebuild.amazonaws.com"
8               },
9               "Action": "sts:AssumeRole"
10          }
11      ]
12  }
```

2. Creación del proyecto CodeBuild. Para crear el proyecto de CodeBuild se utiliza el siguiente comando:

```
$ aws codebuild create-project \
    --cli-input-json file://CodeBuild_TebimartTest.json
```

   El archive .json en este caso se construye siguiendo la guía disponible en la documentación de AWS [30]. Entre otras cosas, en el archivo .json se especifica el nombre del proyecto, de dónde se obtendrá el código en el que trabajar (CodePipeline), el archivo de build (buildspec_test.json), el rol y las especificaciones de la máquina en la que se ejecutará el proceso.

3. Asociar una política al rol. Es necesario asociar una política al rol para que sea capaz de acceder al proyecto de CodeBuild creado. Esto se realiza con el comando:

```
$ aws iam create-policy \
    --policy-name CodeBuildBasePolicy-TebimartTest-us-east-1 \
    --policy-document file://CodeBuildBasePolicy-TebimartTest.json
```

```
$ aws iam attach-role-policy \
    --role-name codebuild-TebimartTest-service-role \
    --policy-arn arn:aws:iam::123456789012:policy/CodeBuildBasePolicy-TebimartTest-us-
        east-1
```

5.4.2. Integración de SonarCloud en CodeBuild

SonarCloud al ser una herramienta de "terceros" no tiene una integración directa con el resto de los servicios AWS, sin embargo, se pueden utilizar sus servicios a través de un proyecto CodeBuild. *Grosso modo*, lo que se hace aquí es crear un proyecto CodeBuild que ejecuta una serie de instrucciones para usar la herramienta *sonar-scanner* que se comunica con el servidor de SonarCloud.

5.4.2.1. Crear un proyecto en SonarCloud

El primer paso consiste en crear un proyecto en SonarCloud que se asociará con el repositorio de AWS CodeCommit. Como se comentó al inicio, no existe una integración directa con AWS, por lo que en este caso el proyecto de SonarCloud se asocia a un repositorio de GitHub que actúa como bypass de código. La configuración del

proyecto se ha llevado a cabo siguiendo la guía definida en la documentación de SonarCloud [31][32]. Al finalizar se tiene:

- Una “organización” SonarCloud.
- Un proyecto en GitHub que actuará como bypass.
- Un token de acceso al proyecto SonarCloud.

5.4.2.2. Modificar repositorio AWS CodeCommit

Es necesario modificar el reposito de AWS CodeCommit para añadir el archivo de configuración de build "buildspec.yml"que define el proceso que se llevará a cabo en el proyecto CodeBuild:

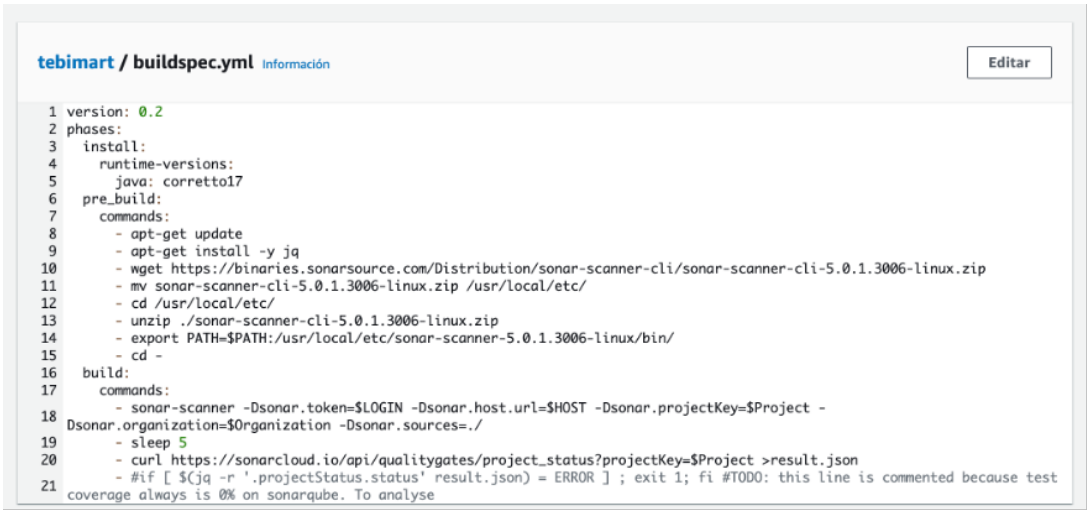

Figura 5.21: Archivo buildspec.json

Las instrucciones de las fases “install” y “pre_build” indican las dependencias necesarias que se tienen que instalar en el entorno de build, que son JDK (en este caso la versión que proporciona Amazon: corretto17) y la última versión de la herramienta sonar-scanner. En la fase “build” se ejecuta la herramienta de escaneo de código con el comando:

```
$ sonar-scanner \
    -Dsonar.token=$LOGIN \
    -Dsonar.host.url=$HOST \
    -Dsonar.projectKey=$Project \
    -Dsonar.organization=$Organization \
    -Dsonar.sources=./
```

El comando necesita cuatro parámetros que son el token de acceso, la URL del servidor, el proyecto y la organización sobre la que se trabaja. Estos parámetros se obtienen al crear el proyecto de SonarCloud (5.4.2.1). En este punto se presenta un problema de seguridad que hay que tener en cuenta y es que los parámetros en esta instrucción van en texto plano y en específico el token de acceso puede dar paso a un acceso no autorizado si cae en malas manos. Para esta situación AWS proporciona un servicio que permite almacenar y administrar de forma segura variables como

contraseñas, claves API, tokens de autenticación y otros datos confidenciales: AWS Secrets Manager [33]. En el caso de la aplicación se ha optado por no utilizar este servicio debido a que la aplicación no pone en riesgo ningún dato sensible y para no caer en gastos innecesarios (el servicio es de pago a partir del primer mes de uso). En su lugar, los parámetros se configuran como variables de entorno al crear el proyecto de AWS CodeBuild.

5.4.2.3. Configuración de AWS CodeBuild

El proyecto de AWS CodeBuild encargado de las pruebas se configura con los siguientes pasos, en esencia idénticos al proyecto creado para las pruebas:

1. Creación de un rol CodeBuild. Se necesita crear un rol especial que permita al servicio AWS CodeBuild realizar acciones a nombre del usuario propietario de la cuenta. El rol se crea mediante el comando:

```
$ aws iam create-role \
    --role-name codebuild-TebimartReview-service-role \
    --assume-role-policy-document file://CodeBuildRolePolicy.json
```

2. Creación del proyecto CodeBuild. Para crear el proyecto de CodeBuild se utiliza el siguiente comando:

```
$ aws codebuild create-project \
    --cli-input-json file://CodeBuild_TebimartReview.json
```

   El archivo .json en este caso se construye siguiendo la guía disponible en la documentación de AWS [30]. Entre otras cosas, en el archivo .json se especifica el nombre del proyecto, de dónde se obtendrá el código en el que trabajar (CodePipeline), el archivo de build (buildspec.json), el rol y las especificaciones de la máquina en la que se ejecutará el proceso.

3. Asociar una política al rol. Es necesario asociar una política al rol para que sea capaz de acceder al proyecto de CodeBuild creado. Esto se realiza con los comandos:

```
$ aws iam create-policy \
    --policy-name CodeBuildBasePolicy-TebimartReview-us-east-1 \
    --policy-document file://CodeBuildBasePolicy-TebimartReview.json
```

```
$ aws iam attach-role-policy \
    --role-name codebuild-TebimartReview-service-role \
    --policy-arn arn:aws:iam::123456789012:policy/CodeBuildBasePolicy-TebimartReview-us-
        east-1
```

5.4.3. Integración de CodeBuild en CodePipeline

Con los dos proyectos de AWS CodeBuild TebimartReview y TebimarTest creados se modifica el pipeline para incluir una fase nueva “Review” entre la toma de código y el despliegue. Esto se realiza mediante la interfaz web debido a su sencillez, retomando la fase que se dejó pendiente en el apartado 5.3.1.4.

En el pipeline se agrega la nueva etapa Review y se añaden dos acciones, cada una correspondiente con los proyectos CodeBuild:

1. Acción CodeReview

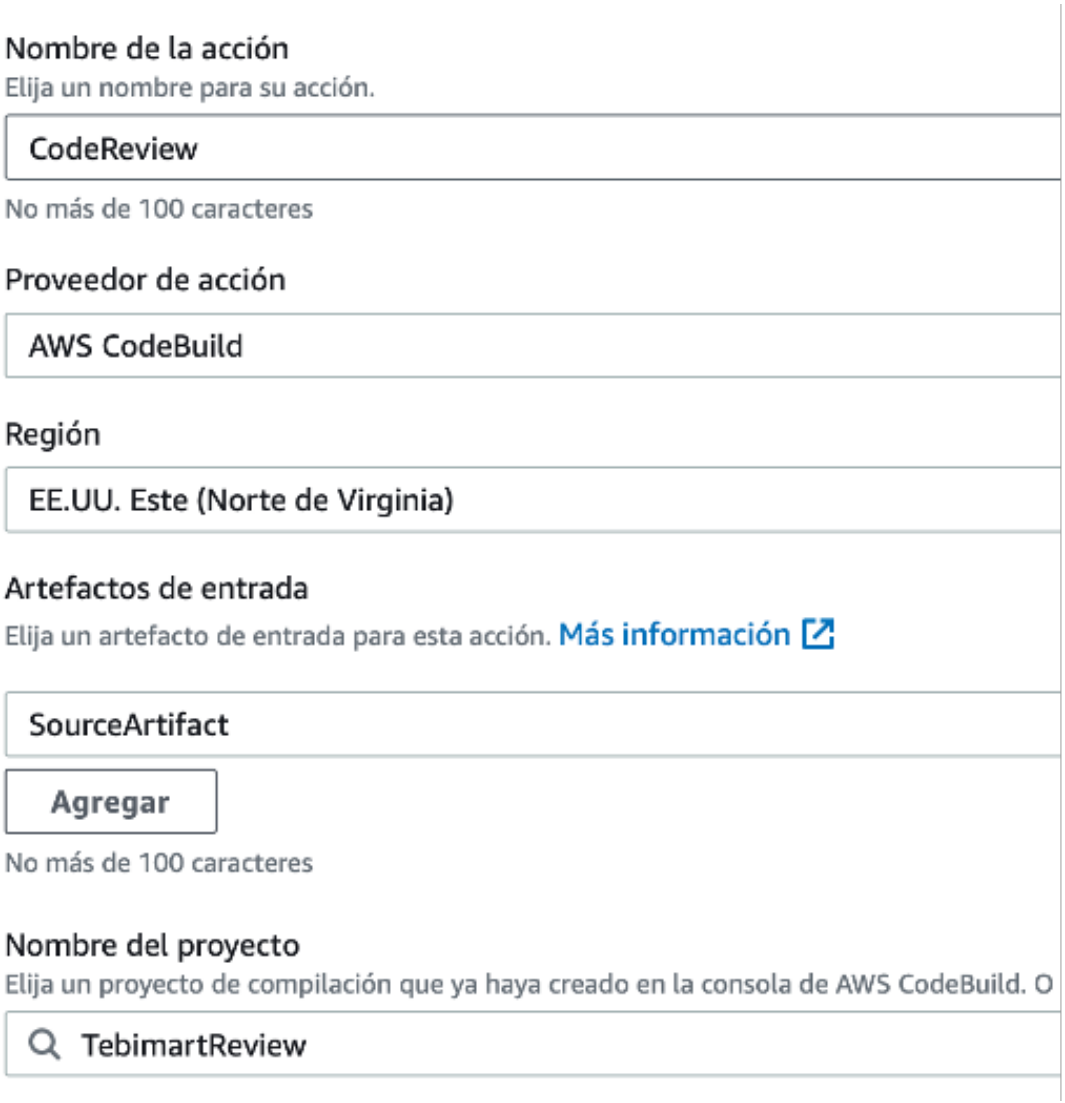

Figura 5.22: Creación de acción CodeReview en CodePipeline

2. Acción CodeTest

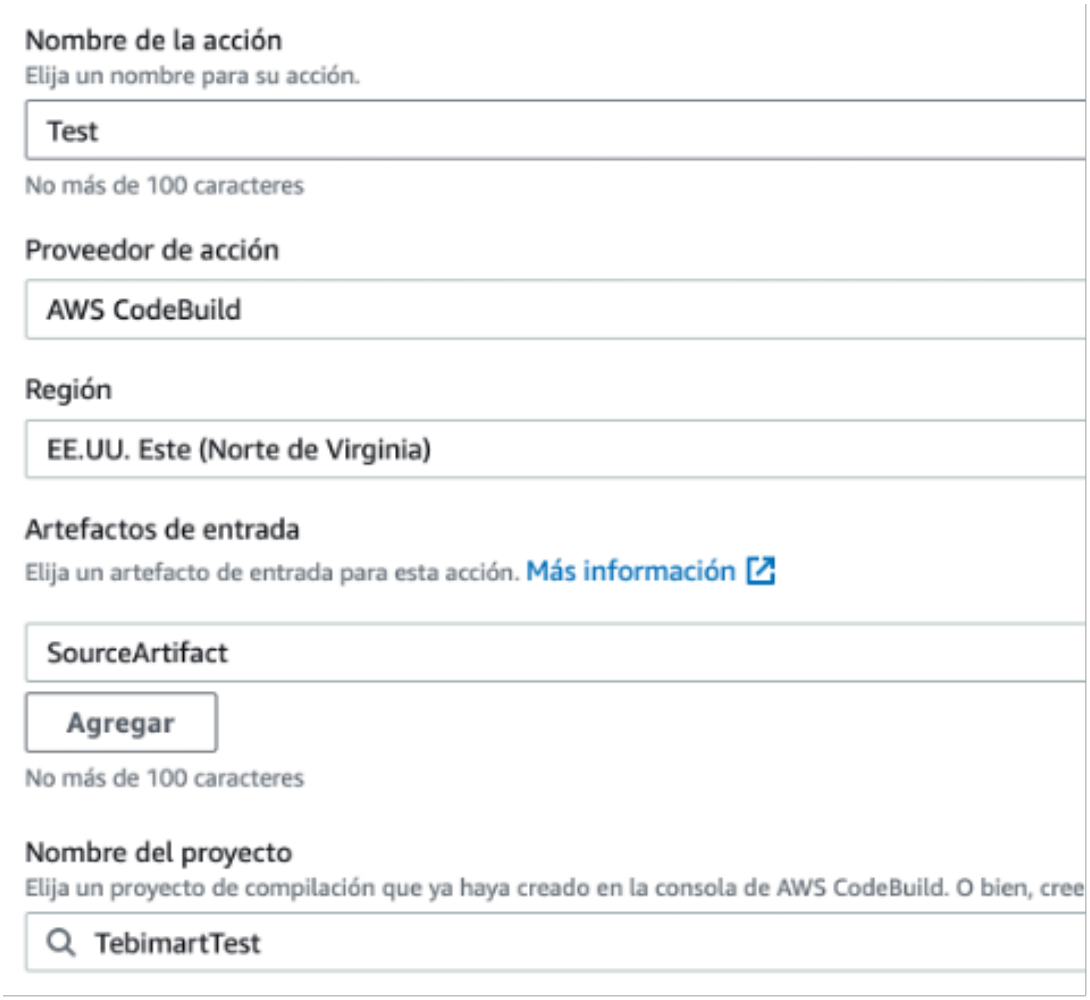

Figura 5.23: Creación de acción CodeTest en CodePipeline

Con la nueva fase creada, cuando se produzca un cambio en el código se ejecutarán las acciones de análisis estático y pruebas a cargo del servicio AWS CodeBuild. Con esto se consigue que el nuevo software únicamente se despliegue cuando el código no presente errores y las pruebas sean exitosas. La siguiente imagen muestra una fase de Review fallida en la que no se continúa a la fase de despliegue:

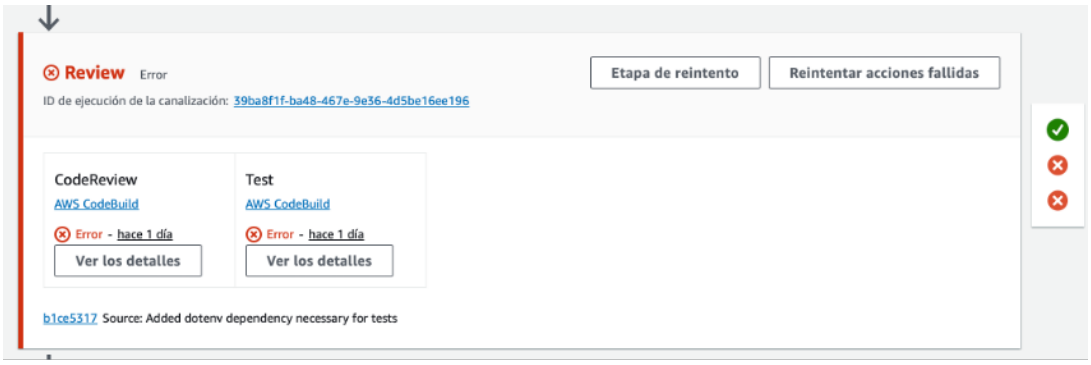

Figura 5.24: CodePipeline - Revisión de código fallida

5.5. Alta disponibilidad: uso de contenedores en CI/CD

Hasta este punto se tiene ya creado un pipeline que toma los últimos cambios de la aplicación, realiza pruebas sobre el nuevo código y distribuye el software a una máquina EC2. Esta puede ser una opción perfectamente válida y a ser considerada por equipos de desarrollo que consideren que su aplicación es simple, con requisitos que no impliquen una alta disponibilidad, escalabilidad ni portabilidad, o si los recursos técnicos son limitados.

Ahora bien, como se indicó en el apartado 5.1.6.2, en este trabajo se va a implementar el uso de contenedores en el proceso de integración y entrega continua.

5.5.1. Flujo de trabajo

La siguiente imagen muestra el flujo que se seguirá a través del pipeline:

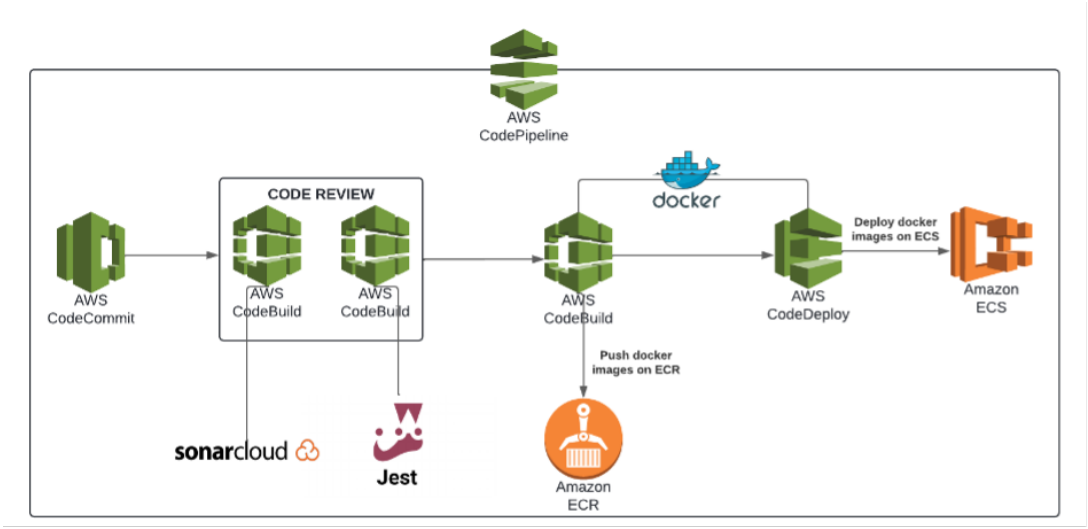

Figura 5.25: Flujo de trabajo con contenedores en AWS CodePipeline

Al flujo de trabajo inicial ahora se suma una fase más de build en la que se crea una imagen Docker y se almacena en el servicio de registro de contenedores AWS ECR.

Por otra parte, se modifica la fase de despliegue en la que AWS CodeDeploy va a trabajar a la par con Amazon Elastic Container Service.

5.5.2. Implementación del pipeline

La implementación del flujo descrito se ha llevado a cabo siguiendo varias de las guías que proporciona Amazon para la creación de sus recursos. Los servicios que se necesitan configurar son:

1. Amazon Elastic Container Register
2. AWS CodeBuild
3. Amazon Elastic Container Service
4. AWS CodeDeploy

5. AWS CodePipeline

5.5.2.1. Configuración de AWS ECR

La creación del repositorio ECR es bastante sencilla y se realiza con el comando:

```
$ aws ecr create-repository --repository-name tebimart-ecr-repo
```

Una vez creado el repositorio ya se pueden guardar imágenes Docker en él y se hará de forma automática con el servicio AWS CodeBuild cuando se ejecute el pipeline. Sin embargo, en este punto se va a crear una imagen Docker local y se almacenará en el repositorio para que el servicio ECS no de problemas de ejecución cuando se configure más adelante.

Para generar la imagen y guardar en el repositorio se ejecutan los comandos:

- Crear imagen:

```
$ docker build -t tebimart-image tebimart/
```

- Etiquetar la imagen con la URI del repositorio ECR creado:

```
$ docker tag tebimart-image:latest 123456789012.dkr.ecr.us-east-1.amazonaws.com/tebimart
    -ecr-repo
```

- Obtener token de autenticación y autenticarse en el repositorio ECR para poder subir imágenes:

```
 $ aws ecr get-login-password --region us-east-1 | docker login --username AWS --
     password-stdin
```

- Enviar la imagen a AWS ECR:

```
$ docker push \
    123456789012.dkr.ecr.us-east-1.amazonaws.com/tebimart-ecr-repo
```

5.5.2.2. Configuración de AWS CodeBuild

Es necesario configurar un nuevo proyecto AWS CodeBuild que se encargue de construir la imagen Docker con el código de la aplicación y enviarla al repositorio AWS ECR. Los pasos para la creación del proyecto CodeBuild son idénticos a los proyectos creados en los apartados 5.4.1.2 y 5.4.2.3. El nuevo proyecto lleva el nombre de "TebimartDocker". Un paso adicional que destacar aquí es que al "role" creado para este proyecto se le debe dar permiso para realizar acciones en el servicio AWS ECR. Para asignarle los permisos se utiliza el comando:

```
$ aws iam attach-role-policy \
    --role-name codebuild-TebimartDocker-service-role \
    --policy-arn arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryPowerUser
```

Por otra parte, es necesario modificar el repositorio AWS CodeCommit para añadir el archivo de especificación "buildspec_docker.yml". El archivo tiene la siguiente forma:

tebimart / buildspec_docker.yml Información — Editar

```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws --version
      - aws ecr get-login-password --region $AWS_DEFAULT_REGION | docker login --username AWS --password-stdin
851725317168.dkr.ecr.us-east-1.amazonaws.com
      - REPOSITORY_URI=851725317168.dkr.ecr.us-east-1.amazonaws.com/tebimart-ecr-repo
      - COMMIT_HASH=$(echo $CODEBUILD_RESOLVED_SOURCE_VERSION | cut -c 1-7)
      - IMAGE_TAG=${COMMIT_HASH:=latest}
  build:
    commands:
      - echo Build started on `date`
      - echo Building the Docker image...
      - docker build -t $REPOSITORY_URI:latest .
      - docker tag $REPOSITORY_URI:latest $REPOSITORY_URI:$IMAGE_TAG
  post_build:
    commands:
      - echo Build completed on `date`
      - echo Pushing the Docker images...
      - docker push $REPOSITORY_URI:latest
      - docker push $REPOSITORY_URI:$IMAGE_TAG
      - echo Writing image definitions file...
      - printf '[{"name":"tebimart-ecs-container","imageUri":"%s"}]' $REPOSITORY_URI:$IMAGE_TAG > imagedefinitions.json
artifacts:
    files: imagedefinitions.json
```

Figura 5.26: Archivo buildspec_docker.yml

El archivo de especificación contiene las fases:

1. Pre_build:
   - Se inicia sesión en Amazon ECR.
   - Se establece el URI del repositorio en la imagen de ECR y se añade una tag con el hash del commit de git.
2. Build:
   - Se crea la imagen de Docker y se etiqueta la imagen como latest.
3. Post_build:
   - Se envía la imagen al repositorio de ECR.
   - Se crea el archivo "imagedefinitions.json" con el nombre del contenedor del servicio de AWS ECS, la imagen y la etiqueta. Esta información se utiliza para que el servicio de AWS ECS despliegue correctamente la versión software.

También es necesario añadir el archivo Dockerfile que contiene las instrucciones para crear la imagen de Docker. Inicialmente el archivo tenía el siguiente contenido:

```
FROM --platform=linux/amd64 node:18-alpine
WORKDIR /app
COPY . .
RUN yarn install
COPY . .
CMD ["node", "src/server.js"]
EXPOSE 3000
```

Las primeras ejecuciones funcionaron sin problema, sin embargo, en un momento se presentó el error que se muestra en la figura 5.27.

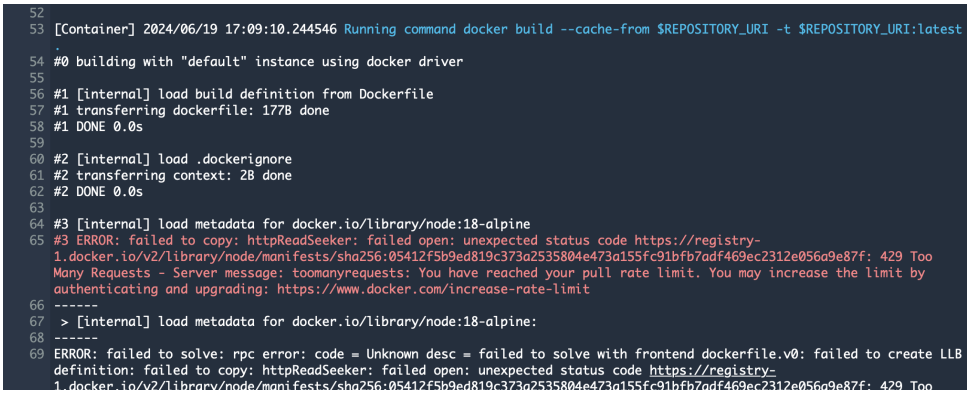

Figura 5.27: Construcción de imagen docker fallida

Este error viene dado cuando se intenta extraer una imagen del repositorio público de Docker Hub después de haber alcanzado el límite de extracción de Docker. Docker Hub utiliza direcciones IP para autenticar a los usuarios, y los límites de extracción se basan en direcciones IP individuales. Para evitar alcanzar el límite de Docker Hub la solución aplicada ha sido guardar la imagen base en un repositorio de AWS ECR y cargar la imagen de dicho repositorio, siguiendo la solución planteada en esta entrada de StackOverflow: https://stackoverflow.com/questions/65058619/finding-rate-limit-of-docker-loggged-in-user:

```
FROM --platform=linux/amd64 851725317168.dkr.ecr.us-east-1.amazonaws.com/node-18-alpine:
    latest
WORKDIR /app
COPY . .
RUN yarn install
COPY . .
CMD ["node", "src/server.js"]
EXPOSE 3000
```

5.5.2.3. Configuración de AWS ECS

Para la configuración del servicio Amazon ECS se han seguido varias de las guías que AWS proporciona en su web [34][35][36]. En la lista de abajo se indican los pasos seguidos, aunque para información más detallada se aconseja mirar las guías:

1. Crear rol de ejecución ECS. Este rol concede permisos necesarios al servicio ECS para realizar tareas relacionadas con el lanzamiento y la gestión de la aplicación en contenedores:

```
$ aws iam create-role \
    --role-name ecsTaskExecutionRole \
    --assume-role-policy-document file://ecs-tasks-trust-policy.json
```

```
$ aws iam attach-role-policy \
    --role-name ecsTaskExecutionRole \
    --policy-arn arn:aws:iam::aws:policy/service-role/AmazonECSTaskExecutionRolePolicy
```

El contenido del archive ecs-tasks-trusy-policy.json es:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "ecs-tasks.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

2. Crear “task definition”. Una “task definition” es una plantilla que describe cómo se ejecutará la aplicación dentro del servicio de ECS y qué configuraciones utilizar al ejecutar las imágenes de contenedor, como la cantidad de CPU y memoria que necesita el contenedor [37]. Se crea mediante el siguiente comando:

```
$ aws ecs register-task-definition \
    --cli-input-json file://tebimart-ecs-task-definition.json \
    --region us-east-1
```

El archivo *tebimart-ecs-task-definition.json* presenta el siguiente formato:

```
{
    "family": "tebimart-task-def",
    "containerDefinitions": [
        {
            "name": "tebimart-ecs-container",
            "image": "<user-id>.dkr.ecr.us-east-1.amazonaws.com/
               tebimart-ecr-repo:latest",
            "cpu": 0,
            "portMappings": [
                {
                    "name": "tebimart-ecs-container-80-tcp",
                    "containerPort": 80,
                    "hostPort": 80,
                    "protocol": "tcp",
                    "appProtocol": "http"
                },
                {
                    "name": "tebimart-ecs-container-3000-tcp",
                    "containerPort": 3000,
                    "hostPort": 3000,
                    "protocol": "tcp",
                    "appProtocol": "http"
```

```
                }
            ],
            "logConfiguration": {
                "logDriver": "awslogs",
                "options": {
                    "awslogs-create-group": "true",
                    "awslogs-group": "/aws/ecs/TebimartAppLog",
                    "awslogs-region": "us-east-1",
                    "awslogs-stream-prefix": "TEBIMART"
                }
            },
            "essential": true,
            "environment": [
                {
                    "name": "AWS_ACCESS_KEY_ID",
                    "value": "<clave de acceso>"
                },
                {
                    "name": "AWS_SECRET_ACCESS_KEY",
                    "value": "<clave secreta de acceso>"
                }
            ],
            "environmentFiles": [],
            "mountPoints": [],
            "volumesFrom": [],
            "systemControls": []
        }
    ],
    "executionRoleArn": "arn:aws:iam::851725317168:role/
       ecsTaskExecutionRole",
    "networkMode": "awsvpc",
    "requiresCompatibilities": [
        "FARGATE"
    ],
    "cpu": "512",
    "memory": "1024",
    "runtimePlatform": {
        "cpuArchitecture": "X86_64",
        "operatingSystemFamily": "LINUX"
    }
}
```

Entre otros parámetros, se pueden destacar las variables de entorno especificadas que se incluyen para poder hacer uso del SDK de AWS, y también la configuración de los Logs en donde se especifica el grupo de AWS CloudWatch en el que se escribirán logs de la aplicación útil para depuración. Para que la aplicación no presente problemas durante el despliegue es necesario crear previamente el grupo para los logs mediante el siguiente comando:

```
$ aws logs create-log-group --log-group-name /aws/ecs/TebimartAppLog
```

3. Crear Application Load Balancer (ALB). Un ALB es un servicio que distribuye el tráfico de red entrante entre los contenedores que ejecutan la aplicación [38]. Es necesario configurar este servicio para tener la funcionalidad blue/green descrita anteriormente. El siguiente comando crea el ALB:

```
$ aws elbv2 create-load-balancer \
     --name tebimart-alb \
     --subnets subnet-abcd1234 subnet-abcd5678 \
     --security-groups sg-abcd1234 \
     --region us-east-1
```

   Es necesario tener de antemano un grupo de seguridad (ver segundo punto del apartado 4.2.2.1) y conocer los identificadores de las subredes que se van a utilizar. Las subredes se pueden consultar usando el siguiente comando:

```
$ aws ec2 describe-subnets --region us-east-1
```

4. Crear ALB Target Groups. Se necesitan configurar dos grupos de destino para que dirijan el tráfico a la tarea establecida en el servicio ECS. Se crean con el siguiente comando:

```
$ aws elbv2 create-target-group \
     --name tebimart-target-1 \
     --protocol HTTP \
     --port 80 \
     --target-type ip \
     --vpc-id vpc-abcd1234 \
     --region us-east-1
```

```
$ aws elbv2 create-target-group \
     --name tebimart-target-2 \
     --protocol HTTP \
     --port 80 \
     --target-type ip \
     --vpc-id vpc-abcd1234 \
     --region us-east-1
```

   Para ejecutar este comando es necesario tener configurada una VPC (Virtual Private Cloud) bajo la que se ejecutará el contenedor. Se puede consultar la lista de VCPs que proporciona AWS por defecto (primer comando) o crear una nueva VCP (segundo comando):

```
$ aws ec2 describe-vpcs --output text
```

```
$ aws ec2 create-vpc \
    --cidr-block 10.0.0.0/24 \
    --query Vpc.VpcId --output text
```

5. Crear ALB Listener. Un listener actúa como punto de entrada de tráfico, escuchando en los puertos configurados. En este caso se utiliza para distribuir el tráfico al grupo de destino creado. Se crea mediante el comando:

```
$ aws elbv2 create-listener \
    --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer
        /app/tebimart-alb/44ea1f6ba1430f1f \
    --protocol HTTP --port 80 \
    --default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east
        -1:123456789012:targetgroup/tebimart-target-1/6736a1f1d8ea1eda \
    --region us-east-1
```

6. Crear ECS Clúster. El clúster se encarga de gestionar los recursos usados por los contenedores ECS. Se crea usando el comando:

```
$ aws ecs create-cluster \
     --cluster-name tebimart-ecs-cluster \
     --region us-east-1
```

7. Crear ECS Service. Un servicio define cuántas instancias de la aplicación (basadas en la "task definition") deben ejecutarse; orquesta el despliegue de los contenedores y monitoriza el estado de los recursos disponibles en el clúster. Este servicio también define las reglas para la conexión en red y la ubicación de las tareas en el hardware.

   Es importante indicar que una vez que el servicio se crea, empieza el despliegue de la aplicación, por lo tanto, es necesario tener todo correctamente configurado para que no se produzcan fallos.

   En este caso el servicio se configura de tal manera que se desplieguen dos instancias de la aplicación (dos contenedores), como respaldo en caso de fallo.

```
$ aws ecs create-service \
     --cli-input-json file://service-tebimart.json \
     --region us-east-1
```

   El archivo service-tebimart.json presenta el siguiente formato:

```
{
    "cluster": "tebimart-ecs-cluster",
    "serviceName": "tebimart-ecs-service",
    "taskDefinition": "tebimart-task-def",
    "loadBalancers": [
        {
            "targetGroupArn": "arn:aws:elasticloadbalancing:us-
               east-1:123456789012:targetgroup/tebimart-target-1
               /6736a1f1d8ea1eda",
            "containerName": "tebimart-ecs-container",
            "containerPort": 3000
        }
    ],
    "launchType": "FARGATE",
    "schedulingStrategy": "REPLICA",
    "deploymentController": {
        "type": "CODE_DEPLOY"
    },
```

```
    "platformVersion": "LATEST",
    "networkConfiguration": {
       "awsvpcConfiguration": {
          "assignPublicIp": "ENABLED",
          "securityGroups": [ "sg-9042787840sad23" ],
          "subnets": [ "subnet-03556706cb4501891w",
                       "subnet-123533s45db4332891a" ]
       }
    },
    "desiredCount": 2
}
```

5.5.2.4. Configuración de AWS CodeDeploy

Para configurar el servicio AWS CodeDeploy se necesita:

1. Crear un rol CodeDeploy ECS. Se necesita crear un rol especial que permita al servicio AWS CodeDeploy realizar acciones en el servicio ECS a nombre del usuario propietario de la cuenta. El rol se crea mediante los comandos:

```
$ aws iam create-role \
      --role-name CodeDeployRoleECS \
      --assume-role-policy-document file://codedeploy-trust-policy.json
```

```
$ aws iam attach-role-policy \
      --role-name CodeDeployRoleECS \
      --policy-arn arn:aws::iam::aws:policy/AWSCodeDeployRoleForECS
```

```
$ aws iam attach-role-policy \
      --role-name CodeDeployRoleECS \
      --policy-arn arn:aws::iam::aws:policy/AWSCodeDeployRoleForECSLimited
```

   El archivo codedeploy-trust-policy.json tiene la siguiente política de confianza:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": ["codedeploy.amazonaws.com"]
            },
            "Action": "sts:AssumeRole"
        }
    ]
}
```

2. Crear una aplicación en CodeDeploy. Para crear la aplicación se ejecuta el comando:

```
$ aws deploy create-application \
    --application-name tebimart-app-ecs \
    --compute-platform ECS
```

   En el comando se indica el nombre de la aplicación y la plataforma en la que se va a desplegar.

3. Configurar el archivo de especificación de la aplicación (appspec.file). Este archivo administra dónde y cómo se realiza el despliegue. Se debe incluir junto al código fuente en el repositorio CodeCommit.

tebimart / appspec.yml Información

```
1  version: 0.0
2  Resources:
3    - TargetService:
4        Type: AWS::ECS::Service
5        Properties:
6          TaskDefinition: "arn:aws:ecs:us-east-1:851725317168:task-definition/tebimart-task-def:3"
7          LoadBalancerInfo:
8            ContainerName: "tebimart-ecs-container"
9            ContainerPort: 3000
10         PlatformVersion: "LATEST"
```

Figura 5.28: Archivo de especificación CodeDeploy ECS

   Lo que se indica es el arn del "task definition" creado, y el contenedor ECS de la aplicación.

4. Crear un grupo de despliegue en CodeDeploy. Este grupo define la configuración del despliegue. Se usa el siguiente comando AWS:

```
$ aws deploy create-deployment-group \
    --application-name tebimart-app-ecs \
    --deployment-group-name TebimartDeploymentGroupECS \
    --service-role-arn arn:aws:iam::user_id:role/CodeDeployRoleECS \
    --cli-input-json file://TebimartDeploymentGroupECS.json
```

   El archivo TebimartDeploymentGroupECS.json se configura con los recursos creados y lleva el siguiente contenido:

```
1 {
2     "deploymentConfigName": "CodeDeployDefault.ECSAllAtOnce",
3     "ec2TagFilters": [],
4     "onPremisesInstanceTagFilters": [],
5     "autoScalingGroups": [],
6     "triggerConfigurations": [],
7     "alarmConfiguration": {
8         "enabled": false,
9         "ignorePollAlarmFailure": false,
```

```
        "alarms": []
    },
    "deploymentStyle": {
        "deploymentType": "BLUE_GREEN",
        "deploymentOption": "WITH_TRAFFIC_CONTROL"
    },
    "outdatedInstancesStrategy": "UPDATE",
    "blueGreenDeploymentConfiguration": {
        "terminateBlueInstancesOnDeploymentSuccess": {
            "action": "TERMINATE",
            "terminationWaitTimeInMinutes": 1
        },
        "deploymentReadyOption": {
            "actionOnTimeout": "CONTINUE_DEPLOYMENT",
            "waitTimeInMinutes": 0
        }
    },
    "loadBalancerInfo": {
        "targetGroupPairInfoList": [
            {
                "targetGroups": [
                    {
                        "name": "tebimart-target-1"
                    },
                    {
                        "name": "tebimart-target-2"
                    }
                ],
                "prodTrafficRoute": {
                    "listenerArns": [
                        "arn:aws:elasticloadbalancing:us-east-1:
                           851725317168:listener/app/tebimart-
                           alb/44ea1f6ba1430f1f/c56db393e931398e
                           "
                    ]
                }
            }
        ]
    },
    "ecsServices": [
        {
            "serviceName": "tebimart-ecs-service",
            "clusterName": "tebimart-ecs-cluster"
        }
    ],
    "terminationHookEnabled": false
}
```

Como se trató en el apartado 5.1.6.2, en el caso de que sea necesario hacer un

rollback de la aplicación, se puede realizar con el siguiente comando:

```
$ aws deploy create-deployment \
    --application-name tebimart-app \
    --deployment-group-name TebimartDeploymentGroup \
    --revision s3://codepipeline-us-east-1-190290071478/TebimartDevPipeline/SourceArti/FnfR5oO
        ?eTag=814522657ededf5831535684e143x-2
```

En este caso es esencial conocer el identificador de revisión que se quiere desplegar.

5.5.2.5. Configuración de AWS CodePipeline

Es necesario modificar el Pipeline para añadir el nuevo servicio CodeBuild que genera la imagen Docker, y sustuir el servicio CodeDeploy EC2 por el servicio CodeDeploy ECS. Esto se realiza de manera sencilla a través de la interfaz gráfica modificando las fases necesarias dentro del pipeline configurado inicialmente.

Nueva fase de Build:

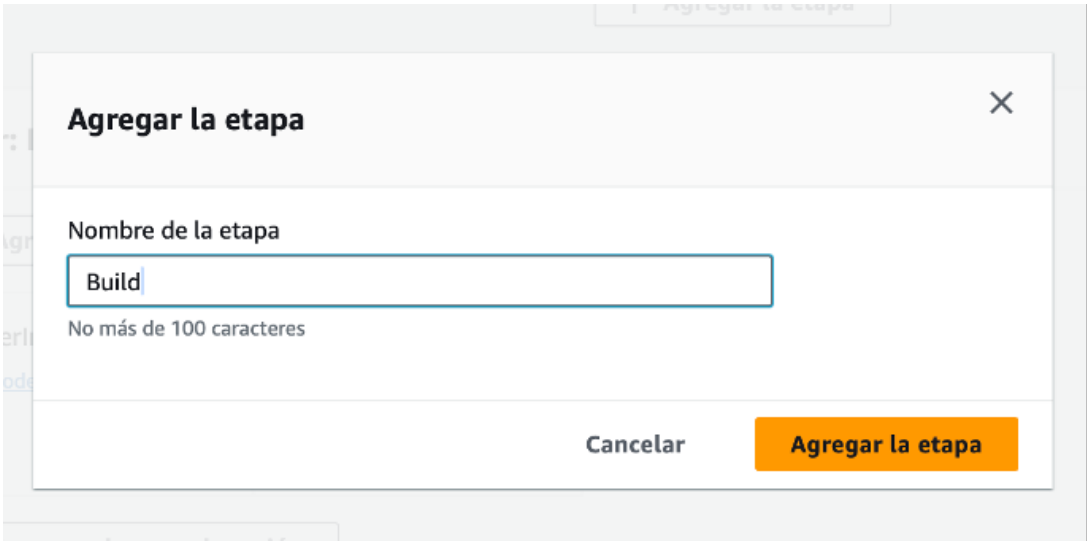

Figura 5.29: Creación de la etapa Build en CodePipeline

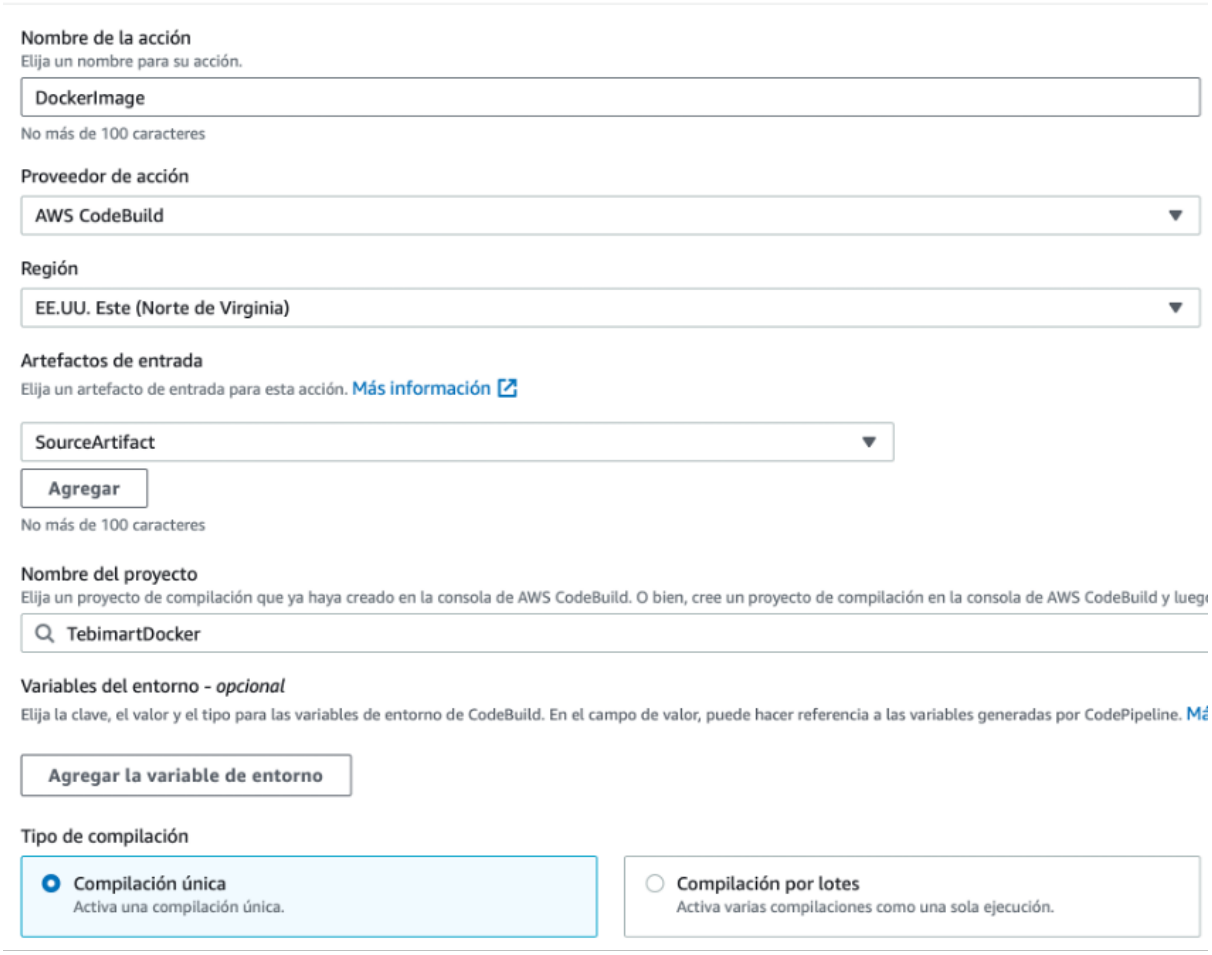

Figura 5.30: Configuración de la acción DockerImage dentro de la etapa Build

Nueva fase DeployECS:

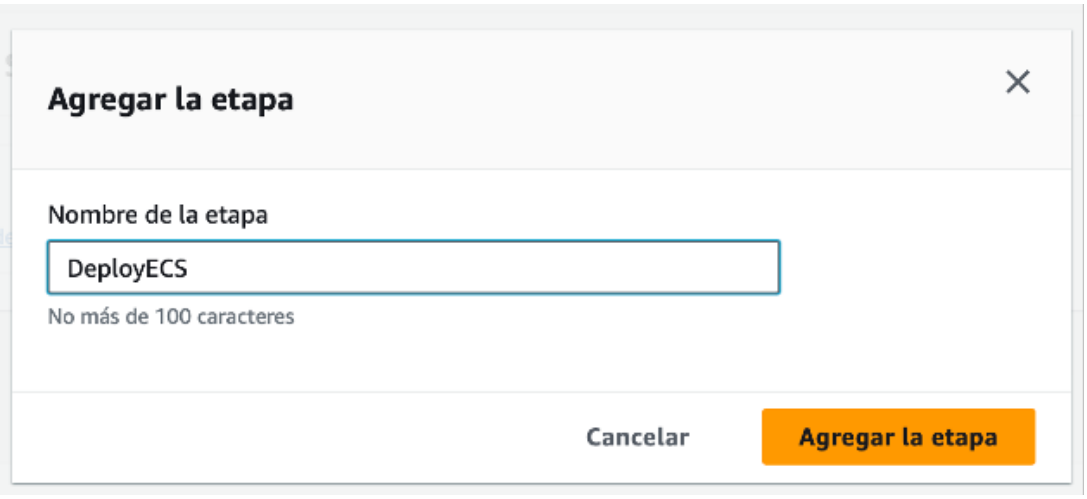

Figura 5.31: Creación de la estapa DeployECS en CodePipeline

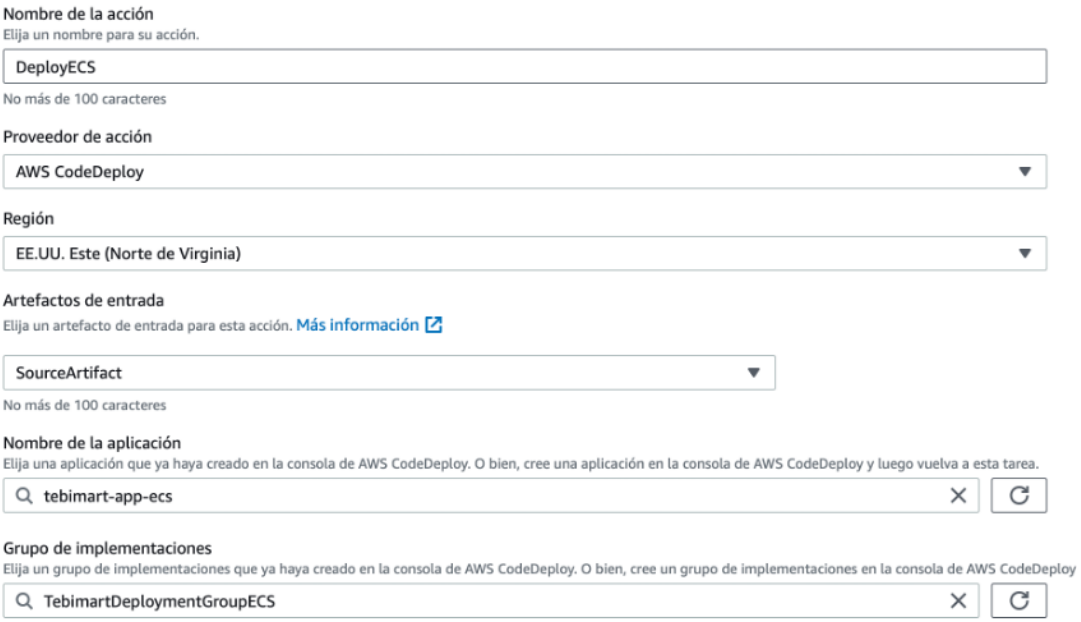

Figura 5.32: Configuración de la acción DeployECS dentro de la etapa DeployECS

5.5.2.6. Resultados de la configuración

A partir de ahora, el software se desplegará y ejecutará a través de contenedores orquestados por el servicio AWS ECS, estando accesible a través de la dirección DNS que proporciona el balanceador de carga creado. Para comprobar el correcto funcionamiento, se ha realizado un cambio sencillo en la plataforma, modificando el precio del producto "Libro2" de 20€ a 40€. Las siguientes imágenes muestran la situación inicial, el proceso seguido por el pipeline, el estado del despliegue en CodeDeploy y la situación final:

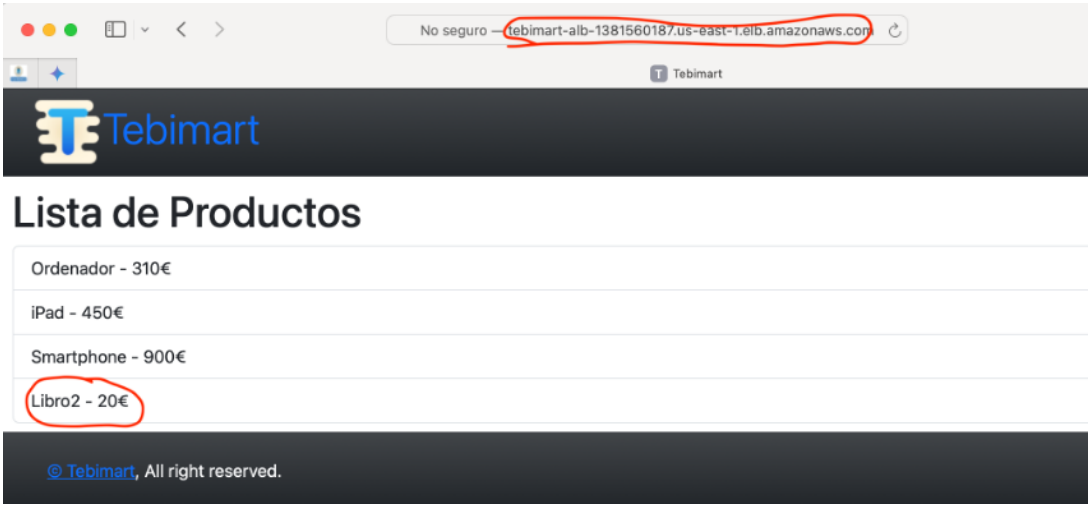

Figura 5.33: Situación inicial de la plataforma

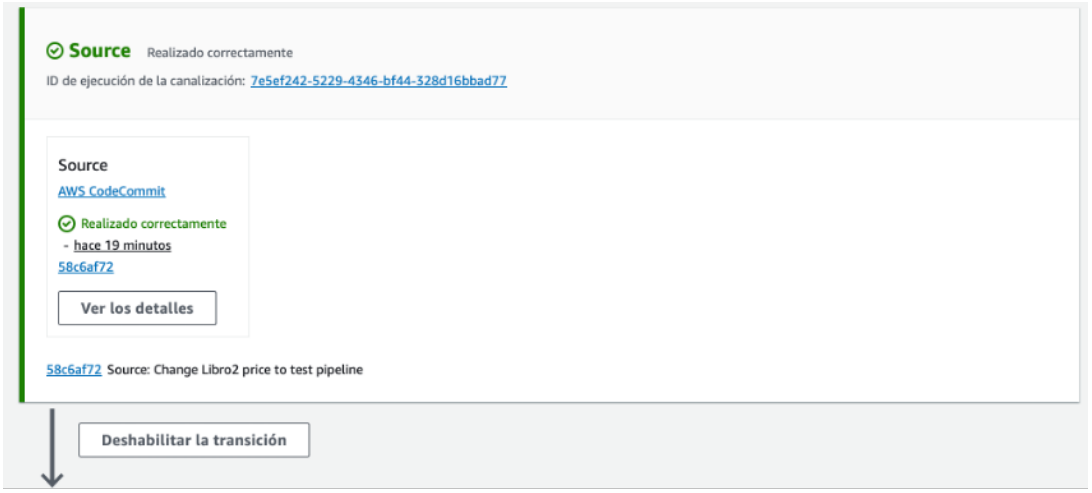

Figura 5.34: Flujo de trabajo - Source

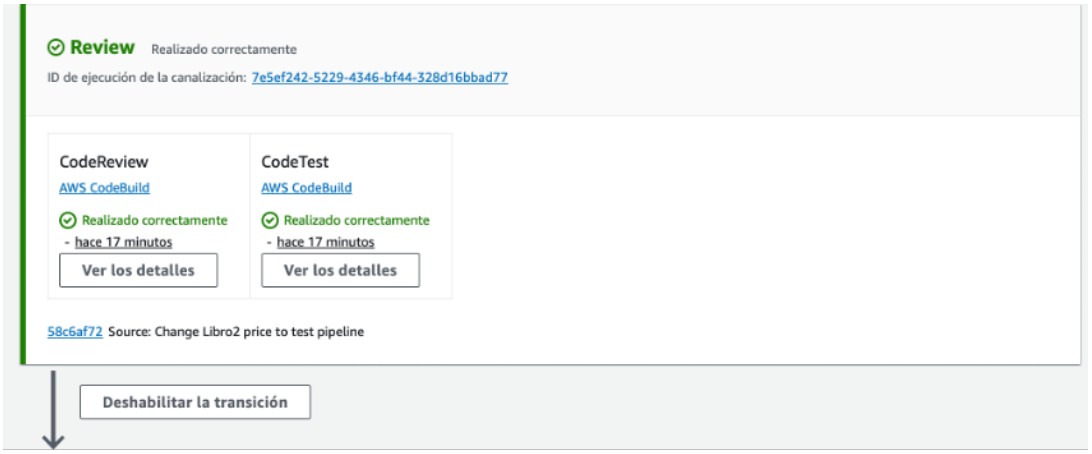

Figura 5.35: Flujo de trabajo - Review

Figura 5.36: Flujo de trabajo - Build

Figura 5.37: Flujo de trabajo - Deploy

Mediante la interfaz de AWS se puede comprobar fácilmente el estado de despliegue realizado por el servicio CodeDeploy. La siguiente muestra como una vez el cambio de software se ha hecho efectivo, el tráfico se deja de enviar a la versión antigua de software y ahora es redirigido a la nueva.

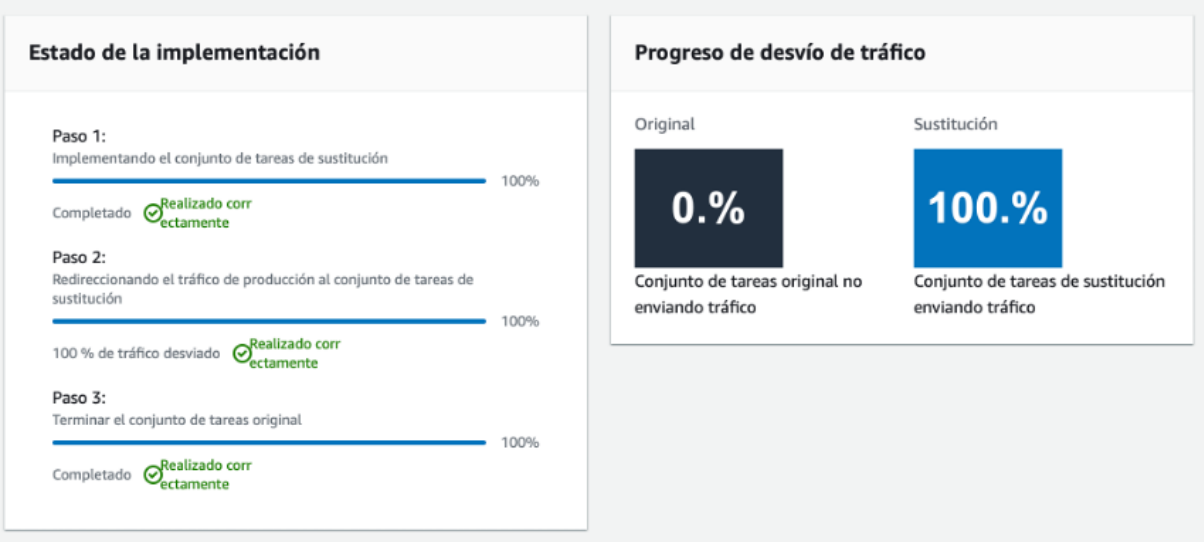

Figura 5.38: Estado del despliegue por interfaz gráfica

El estado del despliegue también se puede mirar mediante comando de AWS CLI.

```
esteban@mackbook:~ $ aws deploy get-deployment-group \
    --application-name tebimart-app-ecs --deployment-group-name TebimartDeploymentGroupECS
{
    "deploymentGroupInfo": {
        "applicationName": "tebimart-app-ecs",
        "deploymentGroupId": "e84e49fa-bea6-48cc-86a8-654d952d45d9",
        "deploymentGroupName": "TebimartDeploymentGroupECS",
        "deploymentConfigName": "CodeDeployDefault.ECSAllAtOnce",
        "ec2TagFilters": [],
        "onPremisesInstanceTagFilters": [],
        "autoScalingGroups": [],
        "serviceRoleArn": "arn:aws:iam::851725317168:role/CodeDeployRoleECS",
        "targetRevision": {
            "revisionType": "S3",
            "s3Location": {
                "bucket": "codepipeline-us-east-1-190290071478",
                "key": "TebimartDevPipeline/SourceArti/wrWP6ag",
                "bundleType": "zip",
                "eTag": "befcc1b717f765fb780a994278639193-1"
            }
        },
        "triggerConfigurations": [],
        "alarmConfiguration": {
            "enabled": false,
            "ignorePollAlarmFailure": false,
            "alarms": []
        },
        "deploymentStyle": {
            "deploymentType": "BLUE_GREEN",
            "deploymentOption": "WITH_TRAFFIC_CONTROL"
        },
        "outdatedInstancesStrategy": "UPDATE",
        "blueGreenDeploymentConfiguration": {
            "terminateBlueInstancesOnDeploymentSuccess": {
                "action": "TERMINATE",
                "terminationWaitTimeInMinutes": 0
            },
            "deploymentReadyOption": {
                "actionOnTimeout": "CONTINUE_DEPLOYMENT",
                "waitTimeInMinutes": 0
            }
        },
```

Figura 5.39: Estado del despliegue mediante línea de comandos

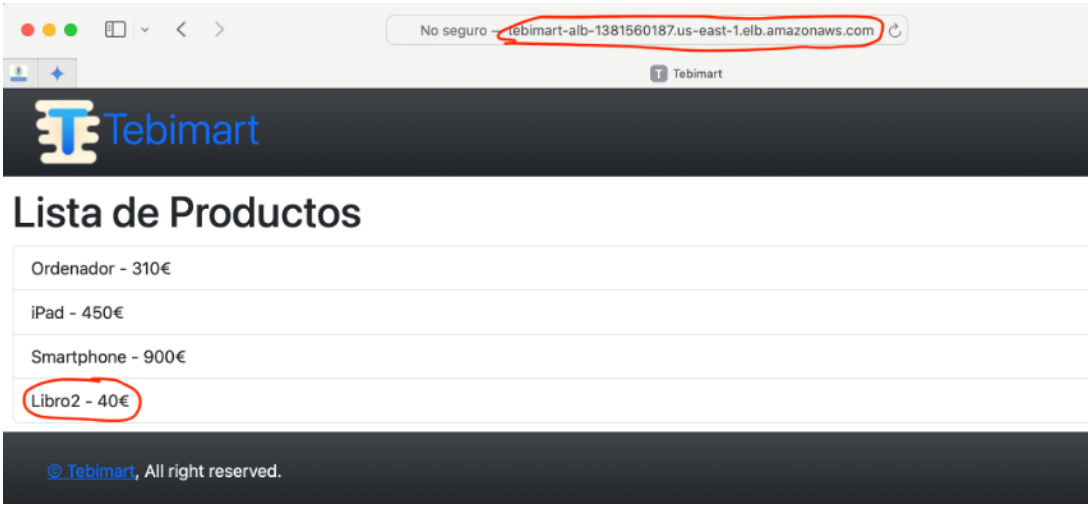

Figura 5.40: Situación final de la plataforma

5.5.3. Health check

El balanceador de carga creado ofrece la posibilidad de definir una configuración de verificación de estado (health check) para un target group. En este caso, se configuran de la siguiente manera para los grupos objetivos creados:

```
$ aws elbv2 modify-target-group \
    --target-group-arn arn:aws:elasticloadbalancing:us-east-1:user-id:targetgroup/tebimart-
        target-1/s3d8gge57d82e137 \
    --health-check-protocol HTTP \
    --health-check-port traffic-port \
    --health-check-enabled true \
    --health-check-intervalSeconds 30 \
    --health-check-timeoutSeconds 5 \
    --healthy-threshold-count 5 \
    --unhealthy-threshold-count 2 \
    --health-check-path / \
    --matcher HttpCode="200" \
```

```
$ aws elbv2 modify-target-group \
    --target-group-arn arn:aws:elasticloadbalancing:us-east-1:user-id:targetgroup/tebimart-
        target-2/p3d8tge57d91e139 \
    --health-check-protocol HTTP \
    --health-check-port traffic-port \
    --health-check-enabled true \
    --health-check-intervalSeconds 30 \
    --health-check-timeoutSeconds 5 \
    --healthy-threshold-count 5 \
    --unhealthy-threshold-count 2 \
    --health-check-path / \
    --matcher HttpCode="200" \
```

Con esta configuración, el ALB enviará periódicamente solicitudes HTTP a la ruta raíz de las instancias desplegadas (usando el mismo puerto que el tráfico) y espera una respuesta exitosa (código HTTP 200) en 5 segundos. Si se fallan 2 verificaciones de estado consecutivas, se marcará como no saludable y se detendrá el tráfico hacia esa instancia.

5.6. Escalabilidad

La escalabilidad en el sistema se implementa dentro del servicio creado en AWS ECS. Como se indicó en el apartado 5.5.2.3, en el servicio se configura el número de instancias de la aplicación que se van a ejecutar, pero además se puede incluir una política de auto-escalado para que incremente o decremente el número de instancias dependiendo de diversas métricas. En el caso concreto de este trabajo, se utilizará una métrica basada el número de solicitudes realizadas a la plataforma.

Según la descripción dada en el apartado 4.1, la plataforma recibirá alrededor de 2.5 millones de visitas mensuales. Si suponemos que estas visitas se distribuyen uniformemente a lo largo del mes y durante las 24 horas del día, esto equivaldría a aproximadamente 83 mil visitas por día y 4 mil visitas por hora. Con estos datos se puede configurar una política de escalado que incremente el número de instancias cuando se supere en un 30 % el número de solicitudes estimado por minuto, es decir, se configurará un umbral de aproximadamente 85 RPM (Request Per Minute).

A continuación, se indican los pasos para que se desplieguen un máximo de 4 ins-

tancias, distribuidas en dos zonas de disponibilidad,cuando el número de solicitudes en la aplicación supere las 85 solicitudes por minuto. Para esto se utilizará la interfaz gráfica de AWS que permite una configuración inmediata, en lugar de utilizar AWS CLI por la complejidad que representa:

1. Establecer número de instancias máximas y mínimas que se requieren.

   En la consola de AWS se selecciona el servicio y se aplica la opción Update. A continuación, en la sección *Service auto scaling* se especifica el número máximo y mínimo de instancias.

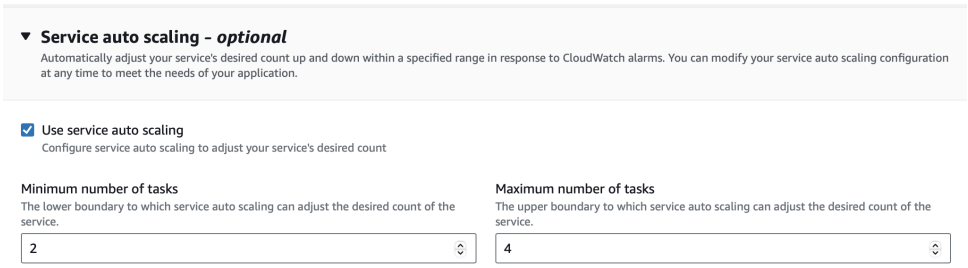

Figura 5.41: Configuración de número de instancias para auto-escalado

2. Establecer política de escalado.

   El siguiente paso es establecer la política de escalado, donde se especifica la métrica y el valor umbral. Dentro de esta política también se indica un periodo de “enfriamiento”, que se refiere a un intervalo de tiempo durante el cual AWS Auto Scaling se abstiene de agregar nueva capacidad después del lanzamiento o la terminación de una instancia. Este periodo de enfriamiento se implementa para evitar que se reaccione a picos transitorios en la demanda y cause acciones de escalado innecesarias. El periodo de enfriamiento se ha configurado en 5 minutos.

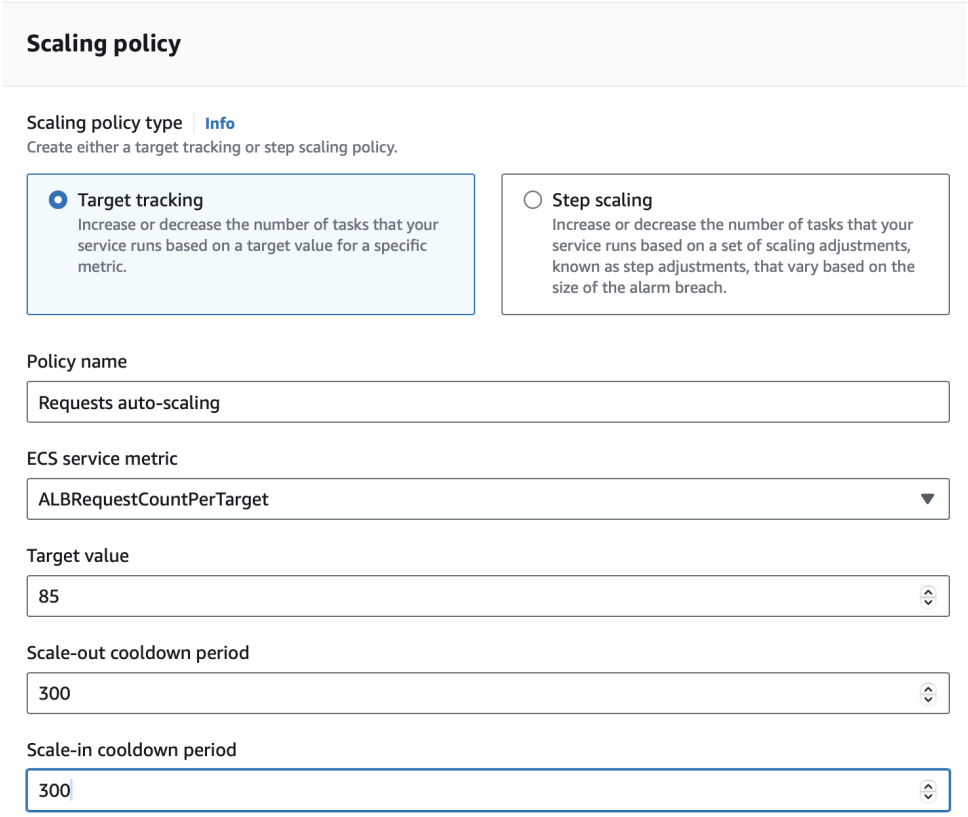

Figura 5.42: Política de auto-escalado

5.7. Backup y recuperación de datos

Es necesario implementar en la aplicación la integración con los servicios AWS DynamoDB y AWS S3 a usar como base de datos de la plataforma. La comunicación con estos servicios, se hará a través del Software Develpment Kit (SDK) que proporciona Amazon para acceder a sus productos.

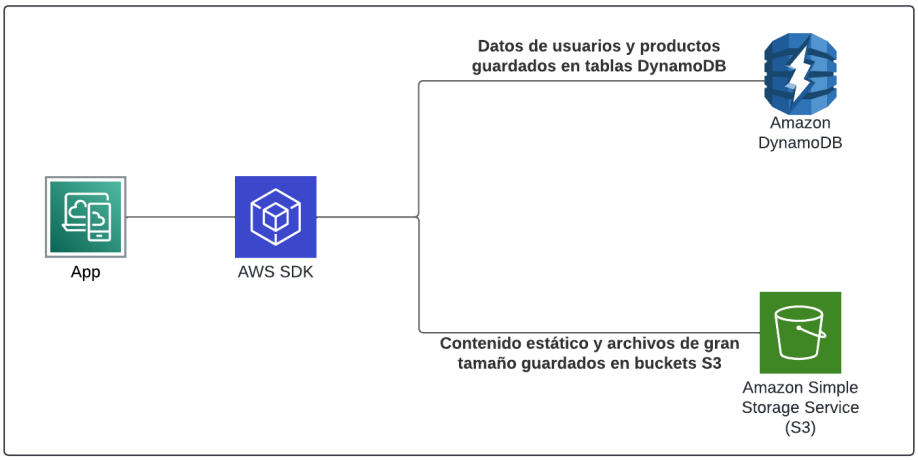

Figura 5.43: Bases de datos de la aplicación

En los siguientes sub-apartados se detalla la configuración de cada servicio y seguidamente también la configuración de AWS Backup como herramienta para el respaldo de los datos.

5.7.1. Configuración de AWS DynamoDB

A diferencia de las bases de datos relacionales tradicionales, DynamoDB no utiliza filas y columnas, sino que se basa en un modelo de clave-valor. Cada elemento de datos se almacena como un par: una clave única que identifica al elemento y un valor que contiene los datos en sí.

Los componentes principales de DynamoDB son:

- Tablas: almacenan los datos en pares clave-valor. Cada tabla tiene un nombre único y puede contener hasta 255 atributos.
- Ítems: son los elementos individuales que se almacenan en las tablas. Cada ítem tiene una clave primaria y puede contener hasta 40 atributos.
- Atributos: son las propiedades de un ítem, como el nombre, la descripción o el precio.
- Claves primarias: identifican de forma única a cada ítem en una tabla. Pueden ser simples o compuestas.
- Índices secundarios: facilitan la consulta de datos en atributos que no son la clave primaria.
- Capacidad de lectura y escritura: determina la cantidad de solicitudes de lectura y escritura que puedes realizar por segundo.
- Unidades de lectura (RU) y unidades de escritura (WU): métricas que se utilizan para medir la capacidad de una tabla.

Para que sea más sencillo de entender el uso de este servicio se va a indicar con un ejemplo donde se guarda información sobre los productos disponibles en la plataforma y se cargan los datos en la aplicación:

1. Crear una tabla. El primer paso es crear la tabla o tablas que se van a utilizar para almacenar los datos. Esto se realiza con el comando:

```
$ aws dynamodb create-table \
    --table-name Products \
    --attribute-definitions AttributeName=productId,AttributeType=S AttributeName=
        price,AttributeType=N \
    --key-schema AttributeName=productId,KeyType=HASH AttributeName=price,KeyType=
        RANGE \
    --table-class STANDARD \
    --billing-mode PAY_PER_REQUEST
```

   Aquí se crea una tabla llamada “Products” que guardará elementos con “productId” como clave primaria, “price” como índice secundario. La clase “STANDARD” y “PAY_PER_REQUEST” representan el tipo de tabla que Amazon tendrá en cuenta para la facturación, en este caso se usa una clase estándar y se pagará por las unidades de lectura y escritura que se usen (se puede consultar información más detallada en la documentación de DynamoDB [39]).

2. Guardar datos de productos. Una vez creada la tabla se pueden guardar la información sobre los productos en ella. En esta ocasión se hace a través de línea de comandos, aunque también se podría hacer usando el SDK de AWS.

```
$ aws dynamodb put-item \
    --table-name Products \
    --item '{
        "productId": {"S": "Teclado mecanico"},
        "price": {"N": "50"},
        "currency": {"S": "eur"},
        "description: {"S": "Teclado mecanico para juegos con cable,
            retroiluminacion LED para Windows, Mac, Linux, Google OS"}
        }'
```

   Con este comando se guarda en la tabla "Products" la información de un "Teclado mecánico". Aquí hay dos cosas importantes a destacar:

   a) La información del item va en formato JSON, por lo que en el caso de tener varios productos, se puede hacer referencia a un fichero externo. *–item file://item.json*

   b) Al guardar la información del producto se almacenan atributos adicionales que no se declararon al crear la tabla. DynamoDB interpreta correctamente esta acción al tratarse de una BBDD noSQL y guarda el producto con los nuevos atributos.

3. Una vez se tiene guardada la información sobre los productos en la BBDD, se puede acceder a ellos a través de la aplicación utilizando el SDK . Para esto es necesario importar las dependencias de DynamoDB y utilizar la función *Scan*. A continuación, un ejemplo para JavaScript:

```
1   // Importar dependencias
2   import { DynamoDBClient, ScanCommand } from "@aws-sdk/client-
        dynamodb";
3   import { DynamoDBDocumentClient } from "@aws-sdk/lib-dynamodb";
4   ...
5   const client = new DynamoDBClient({});
6   const docClient = DynamoDBDocumentClient.from(client);
7   ...
8   // Path to home page
9   app.get("/", async (req, res) => {
10      const products = [];
11      ...
12      try {
13          const command = new ScanCommand({TableName: "Products"});
14
15          const response = await docClient.send(command);
16
17          for (const item of response.Items) {
18              const { productId, price: priceValue, currency,
                    description} = item;
19              ...
20          }
21
22      } catch (error) {
23          console.error(error);
```

```
24                 res.status(500).send('Error scanning DynamoDB table');
25             }
26         ...
27         }
```

En la documentación disponible en la web de Amazon se encuentran indicaciones detalladas de cómo realizar acciones de DynamoDB con el SDK de AWS [40].

La siguiente imagen muestra el aspecto de la aplicación al cargar los datos de los productos desde AWS DynamoDB:

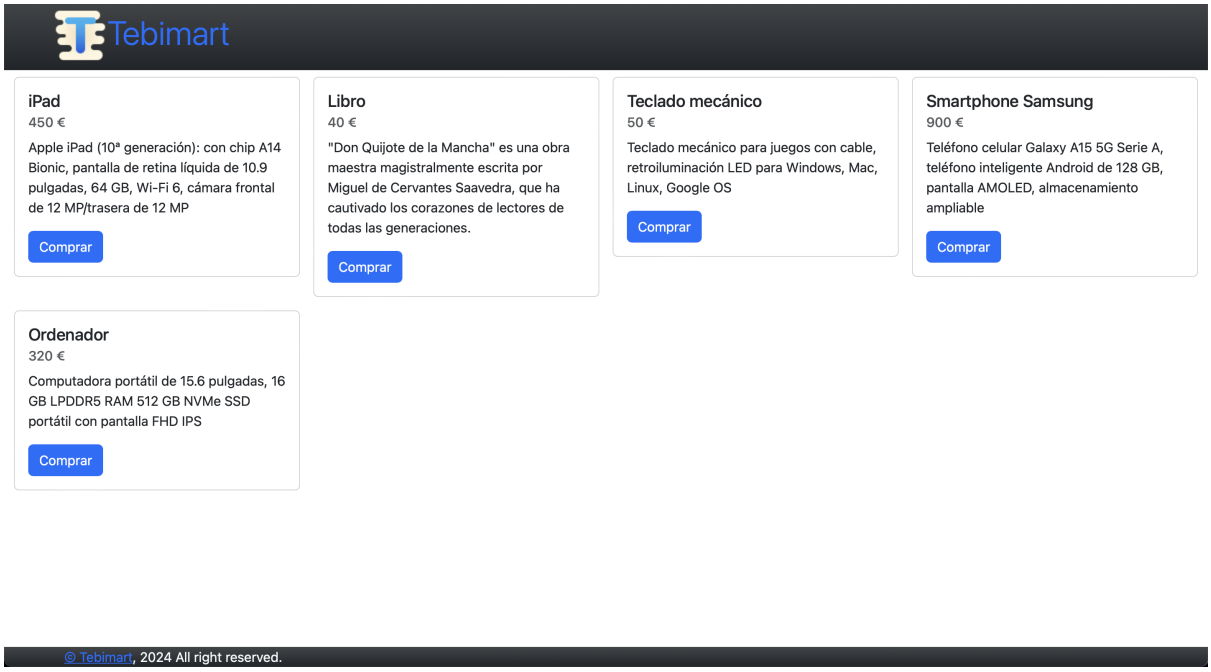

Figura 5.44: Vista de la plataforma cargando datos desde AWS DynamoDB

Nota: durante la primera ejecución del nuevo código que toma los datos de DynamoDB surgió un problema en la fase de Test porque el rol asignado a AWS CodeBuild no tenía los permisos necesarios para acceder al servicio DynamoDB al ejecutar las pruebas como se muestra en el siguiente reporte:

```
> tebimart@1.0.0 test
> node -r dotenv/config --experimental-vm-modules node_modules/jest/bin/jest.js --detectOpenHandles --coverage=100 --
forceExit

(node:87) ExperimentalWarning: VM Modules is an experimental feature and might change at any time
(Use `node --trace-warnings ...` to show where the warning was created)
PASS tests/app.middleware_setup.test.js
  ✓ app uses helmet middleware (3 ms)
  ✓ app sets view engine to ejs (1 ms)
  ✓ app uses public directory for static assets

  console.error
    AccessDeniedException: User: arn:aws:sts::851725317168:assumed-role/codebuild-TebimartReview-service-
role/AWSCodeBuild-fa18dca8-7973-4a6a-93f5-8ba8eff7bdbf is not authorized to perform: dynamodb:Scan on resource:
arn:aws:dynamodb:us-east-1:851725317168:table/Products because no identity-based policy allows the dynamodb:Scan action
        at throwDefaultError (/codebuild/output/src1043591440/src/node_modules/@smithy/smithy-client/dist-
cjs/index.js:839:20)
        at /codebuild/output/src1043591440/src/node_modules/@smithy/smithy-client/dist-cjs/index.js:848:5
        at de_CommandError (/codebuild/output/src1043591440/src/node_modules/@aws-sdk/client-dynamodb/dist-
cjs/index.js:2230:14)
        at processTicksAndRejections (node:internal/process/task_queues:95:5)
        at /codebuild/output/src1043591440/src/node_modules/@smithy/middleware-serde/dist-cjs/index.js:35:20
        at /codebuild/output/src1043591440/src/node_modules/@smithy/core/dist-cjs/index.js:165:18
        at /codebuild/output/src1043591440/src/node_modules/@smithy/middleware-retry/dist-cjs/index.js:320:38
        at /codebuild/output/src1043591440/src/node_modules/@aws-sdk/middleware-logger/dist-cjs/index.js:34:22
        at /codebuild/output/src1043591440/src/src/app.js:22:26 {
      '$fault': 'client',
      '$metadata': {
        httpStatusCode: 400,
        requestId: '8SMN7ZFFMC4B0B0JAPOIG5SSM7VV4KQNSO5AEMVJF66Q9ASUAAJG',
        extendedRequestId: undefined,
```

Figura 5.45: Log de pruebas - acceso denegado a DynamoDB

Para solucionar este problema fue necesario añadir una política al rol de test que permita realizar acciones con tablas DynamoDB:

```
$ aws iam attach-role-policy \
    --role-name codebuild-TebimartTest-service-role \
    --policy-arn arn:aws:iam::aws:policy/AmazonDynamoDBReadOnlyAccess
```

5.7.2. Configuración de AWS Simple Storage Service (S3)

El servicio AWS S3 está compuesto por "Buckets" que es el contenedor para los datos, y "Objetos" que son los archivos individuales dentro de un bucket.

El bucket se configura con el comando:

```
$ aws s3api create-bucket \
    --bucket tebimart-products \
    --region us-east-1
```

Una vez creado el bucket ya se pueden almacenar objetos. Como ejemplo, con el siguiente comando se guarda el objeto iPad.jpg en el bucket con la clave "iPad" (la clave se usa para obtener el objeto cuando se necesite):

```
$ aws s3api put-object \
    --bucket tebimart-products \
    --key "iPad" \
    --body /Users/esteban/Documents/UPM/TFM/fotos-productos/iPad.jpg
```

Cuando se crea el bucket de S3, se establece automáticamente un "Bloqueo de Acceso Público" que es una función de seguridad de AWS S3 que permite evitar que cualquier persona de internet acceda a los objetos dentro del bucket. En este caso se va a deshabilitar este bloqueo porque se considera que las imágenes de los productos no

son información sensible, además de que se que quiere que estén públicas para que cualquier persona pueda visualizarlas a través de la plataforma. Para deshabilitar el bloqueo se utiliza el siguiente comando:

```
$ aws s3api delete-public-access-block --bucket tebimart-products
```

Para controlar de una manera más específica que la aplicación pueda acceder al bucket y realizar acciones sobre los elementos, se configura una política en la que se define que cualquier usuario pueda leer y obtener los elementos del bucket, restringiendo el resto de acciones, como por ejemplo, eliminar elementos. La politica se configura con el siguiente comando:

```
$ aws s3api put-bucket-policy \
    --bucket tebimart-products \
    --policy file://s3-policy.json
```

El archivo JSON de la politica tiene el siguiente contenido:

```
{
    "Version": "2012-10-17",
    "Id": "Policy1716218557741",
    "Statement": [
        {
            "Sid": "Stmt1716218415270",
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "s3:GetObject",
                "s3:PutObject",
                "s3:PutObjectAcl"
            ],
            "Resource": "arn:aws:s3:::tebimart-products/*"
        }
    ]
}
```

Por último, hay que habilitar el versionado del bucket S3, esto es importante para que AWS Backup pueda realizar copias de seguridad de los objetos. Esto se debe a que AWS Backup depende de los identificadores de versión únicos (Version IDs) de S3 para diferenciar entre diferentes versiones del mismo objeto. El siguiente comando habilita el versionado en el bucket:

```
$ aws s3api put-bucket-versioning \
    --bucket tebimart-products \
    --versioning-configuration MFADelete=Disabled,Status=Enabled
```

La siguiente imagen muestra el aspecto de la aplicación al cargar las imágenes de los productos desde AWS S3:

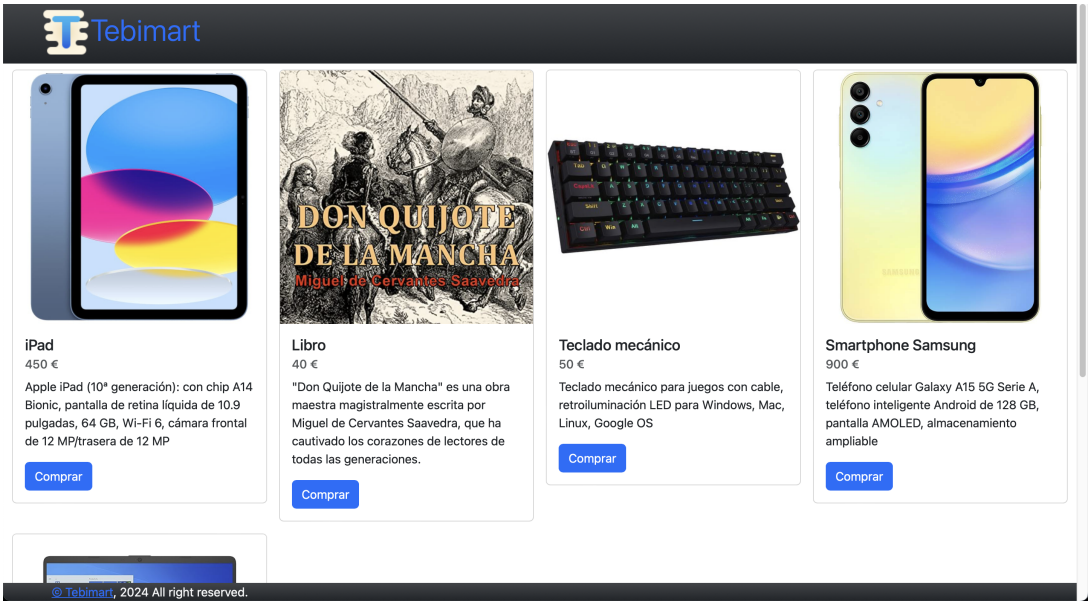

Figura 5.46: Vista de la plataforma cargando imagenes desde AWS S3

5.7.3. Configuración de AWS Backup

En el servicio AWS Backup se va a configurar un "Plan de Backup", que es una política que define cómo y cuándo se deben crear copias de seguridad de los recursos de AWS. Entre las reglas del plan de backup se especifica:

- Recursos AWS que se deben respaldar: En este caso DynamoDB y S3.
- Frecuencia de backup: se pueden crear copias de seguridad a diario, semanalmente, mensualmente o con cualquier otra frecuencia.
- Ventana de backup: intervalo de tiempo durante el cual se deben crear las copias de seguridad. Esto es útil para evitar interrupciones en las operaciones durante las horas pico.
- Opciones de retención: indica cuánto tiempo se deben conservar las copias de seguridad.

El Plan de Backup se va a crear a través de la interfaz web de AWS por la facilidad que presenta en contraposición a usar la línea de comandos, que requiere la ejecución de varios comandos para una implementación.

Un paso previo a la configuración del Plan de Trabajo es crear lo que se llama "Backup Vault", que es un contenedor lógico dentro del servicio AWS Backup que almacena las copias de seguridad. Para esto se utiliza el siguiente comando:

```
$ aws backup create-backup-vault --backup-vault-name tebimart-backup-vault
```

Las imágenes de abajo muestran como se configura el Plan de Backup a través de la consola web.

El plan de backup llevará el nombre de "tebimart-backup-plan", y se ejecutará todos los días a las 00:30 horas y terminando en un plazo de seis horas. El backup se

permanecerá en el sistema por unos 35 días, después se eliminará.

Los recursos para los que se hará el backup son todas las tablas DynamoDB y el bucket “tebimart-products” de AWS S3.

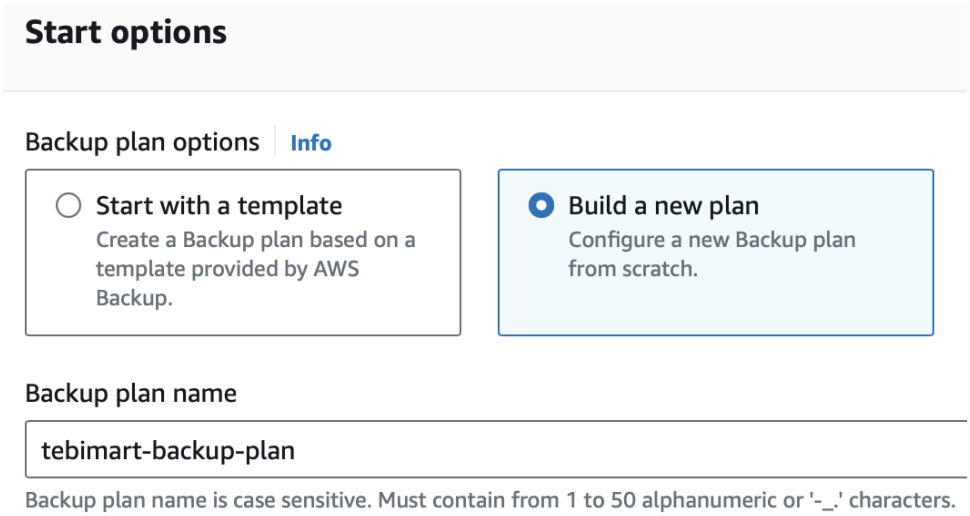

Figura 5.47: AWS Backup - Creación de nuevo backup

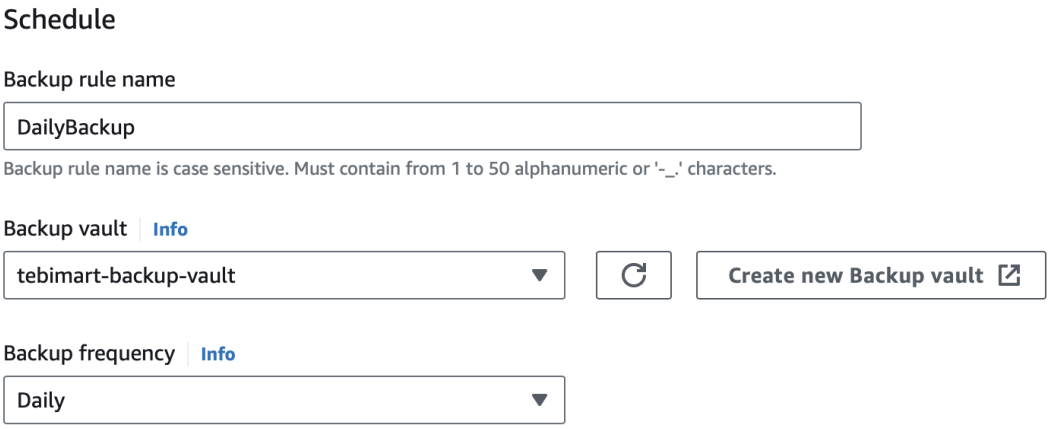

Figura 5.48: AWS Backup - Frecuencia de backup

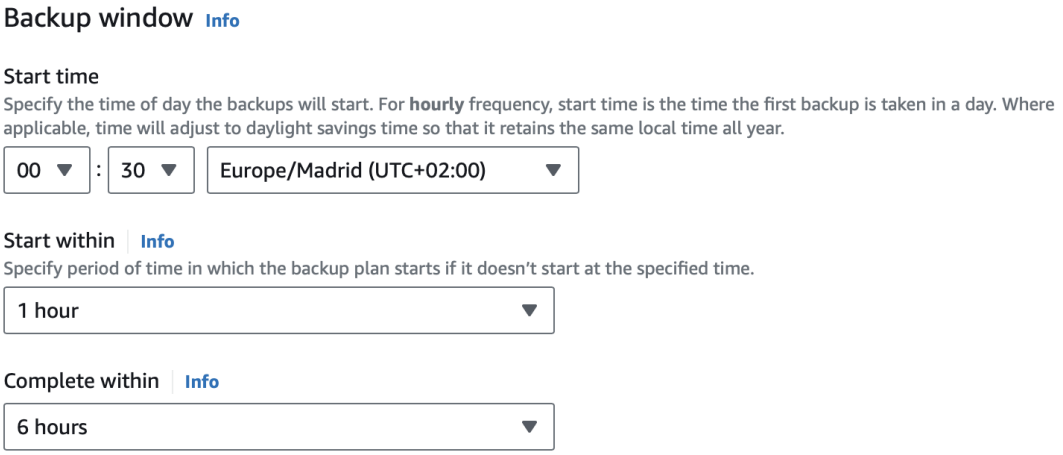

Figura 5.49: AWS Backup - Período de ejecución del backup

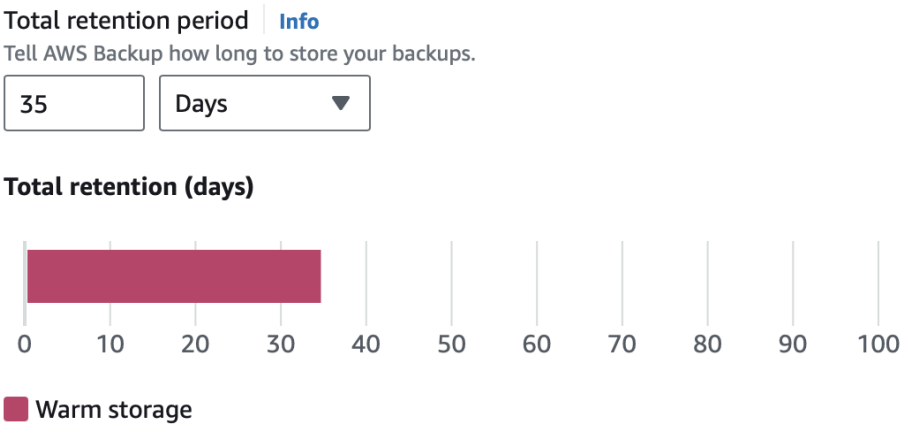

Figura 5.50: AWS Backup - Periodo de retención del backup

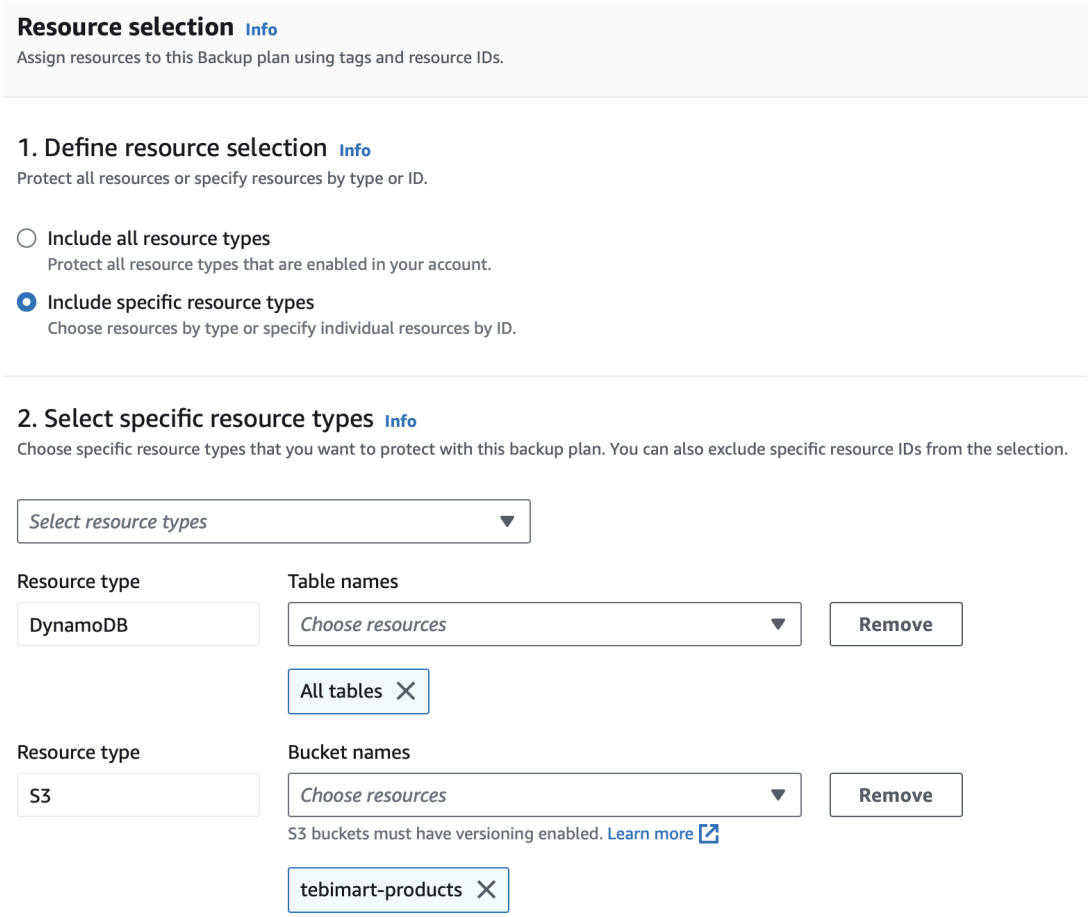

Figura 5.51: AWS Backup - Asignación de recursos DynamoDB y S3

Se puede consultar la lista de backups del sistema con el comando:

```
$ aws backup list-backup-jobs
```

El comando muestra la información necesaria de todos los backups del sistema. Este dato también se puede consultar por la interfaz web, que resulta más visual:

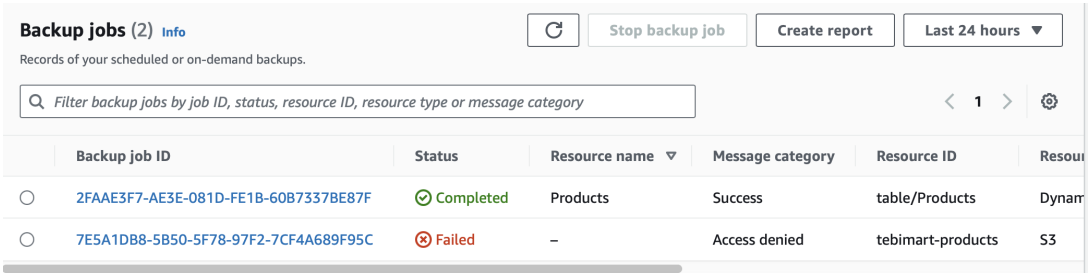

Figura 5.52: Ejemplo de estado de Backups en el sistema

En la figura anterior se muestra el estado de los backups para los recursos AWS DynamoDB y AWS S3. Como se puede observar, el backup del recurso S3 ha fallado

debido a los permisos. Esto se debe a que cuando se crea el Backup Plan, a su vez se crea un rol de IAM para que el servicio AWS Backup pueda operar sobre los distintos recursos, y este rol por defecto no tiene permiso para acceder al servicio S3. Para solucionarlo es necesario añadir dos políticas ejecutando los comandos:

```
$ aws iam attach-role-policy \
    --role-name AWSBackupDefaultServiceRole \
    --policy-arn arn:aws:iam::aws:policy/AWSBackupServiceRolePolicyForS3Backup

$ aws iam attach-role-policy \
    --role-name AWSBackupDefaultServiceRole \
    --policy-arn arn:aws:iam::aws:policy/AWSBackupServiceRolePolicyForS3Restore
```

Ahora, en las nuevas copias de seguridad realizadas, se observa que no existe ningún problema:

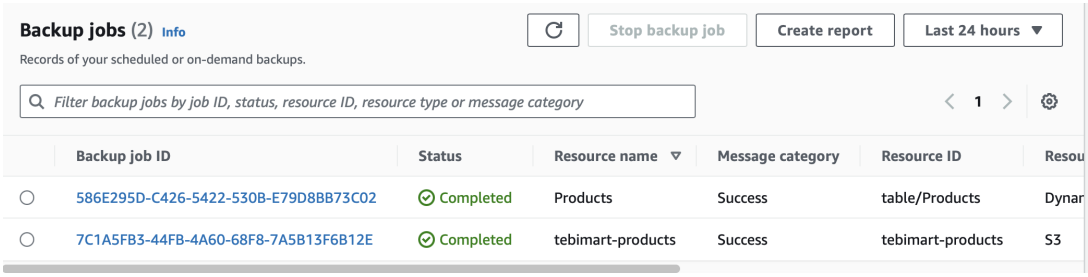

Figura 5.53: Ejemplo de estado de Backups en el sistema

5.7.4. Recuperación de datos

La recuperación de los datos se realiza de forma manual. Para ilustrar la forma de recuperación de manera práctica, se va a crear una situación ficticia en la que "por error" se borran dos elementos de las bases de datos y se quiere restaurar el estado anterior.

1. **Situación inicial**

   El comando devuelve información sobre la tabla *Products*, entre otras cosas, indicando el tamaño de la tabla, el número de elementos en ella o el ID:

   ```
   $ aws dynamodb scan --table-name Products
   ```

   Resultado:

   ```
   "Items": [
       {
           ...
           "productId": {"S": "iPad"}
       },
       {
           ...
           "productId": {"S": "Libro"}
       },
   ```

```
10      {
11          ...
12          "productId": {"S": "Teclado mecanico"}
13      },
14      {
15          ...
16          "productId": {"S": "Smartphone Samsung"}
17      },
18      {
19          ...
20          "productId": {"S": "Ordenador"}
21      }
22  ],
23  "Count": 5,
24  "ScannedCount": 5,
```

La información devuelta se corresponde con los productos de la figura 5.44

Por otra parte, para consultar la información de AWS S3 se usa el comando:

```
$ aws s3 ls tebimart-products
```

Resultado:

```
2024-05-20 18:00:00      87690 Libro
2024-05-20 17:59:15      39863 Ordenador
2024-05-20 18:01:15      47898 Smartphone Samsung
2024-05-20 18:00:18      24016 Teclado mecanico
2024-05-20 17:59:48      51645 iPad
```

La información devuelta se corresponde con las imágenes presentes en la figura 5.46

2. **Situación de desastre**

   Para generar la situación de desastre se van a borrar los productos Libro y iPad tanto de DynamoDB como de S3, con los siguientes comandos:

```
# Borrar datos de AWS DynamoDB
$ aws dynamodb delete-item  \
    --table-name Products \
    --key '{"productId": {"S": "Libro"}, "price": {"N": "40"}}'

$ aws dynamodb delete-item  \
    --table-name Products \
    --key '{"productId": {"S": "iPad"}, "price": {"N": "450"}}'

# Borrar datos de AWS S3
$ aws s3api delete-object --bucket tebimart-products --key Libro
$ aws s3api delete-object --bucket tebimart-products --key iPad
```

Al listar nuevamente los elementos usando los comandos del punto 1 se obtiene:

- DynamoDB:

```
"Items": [
    {
        ...
        "productId": {"S": "Teclado mecanico"}
    },
    {
        ...
        "productId": {"S": "Smartphone Samsung"}
    },
    {
        ...
        "productId": {"S": "Ordenador"}
    }
],
"Count": 3,
"ScannedCount": 3,
```

- S3:

```
2024-05-20 17:59:15      39863 Ordenador
2024-05-20 18:01:15      47898 Smartphone Samsung
2024-05-20 18:00:18      24016 Teclado mecanico
```

En la aplicación Web:

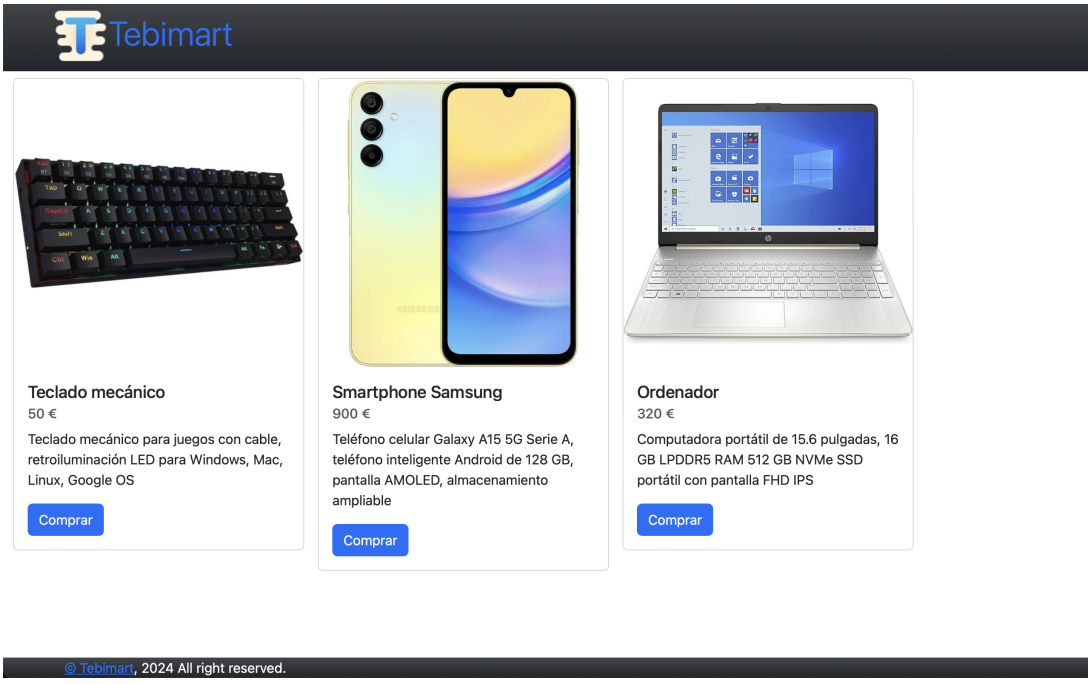

Figura 5.54: Vista de la aplicación al eliminar dos elementos de la base de datos

3. **Recuperación de datos**

La recuperación de datos se realiza en 3 pasos:

a) Obtener el identificador de punto de recuperación de los recursos que se desean recuperar, esto es el campo "RecoveryPointArn" devuelto por el siguiente comando:

```
$ aws backup list-backup-jobs
```

b) Obtener el conjunto de pares clave-valor de metadatos. Contiene la información necesaria para restaurar un punto de recuperación. Para ello se ejecuta el comando:

```
$ aws backup get-recovery-point-restore-metadata \
    --backup-vault-name tebimart-backup-vault
    --recovery-point-arn <valor obtenido en el paso anterior>
```

El servicio DynamoDB necesita la información de los siguientes campos [41]:

```
1 targetTableName
2 encryptionType
3 kmsMasterKeyArn
4 aws:backup:request-id
```

El servicio S3 necesita la información de los siguientes campos [42]:

```
1 DestinationBucketName
2 NewBucket
3 Encrypted
4 CreationToken
5 EncryptionType
6 aws:backup:request-id
```

c) Iniciar el trabajo de recuperación con el comando:

```
$ aws backup start-restore-job \
    --recovery-point-arn <paso 1> \
    --metadata file://metadatos.json(paso 2) \
    --iam-role-arn <role de backup creado durante la config. de AWS Backup>
```

Se puede supervisar el progreso de la tarea de restauración mediante el comando:

```
$ aws backup describe-restore-job \
    --restore-job-id <value>
```

Es importante señalar que cuando se utiliza AWS Backup para restaurar una copia de seguridad se crea un nuevo recurso con la copia de seguridad que se está restaurando. Esto sirve para evitar que la actividad de restauración destruya los recursos existentes. En este caso de ejemplo se han creado dos nuevos recursos, "ProductsRestore" y "tebimart-products-restore", que se corresponden con los datos restaurados de la tabla de DynamoDB y del bucket S3, respectivamente.

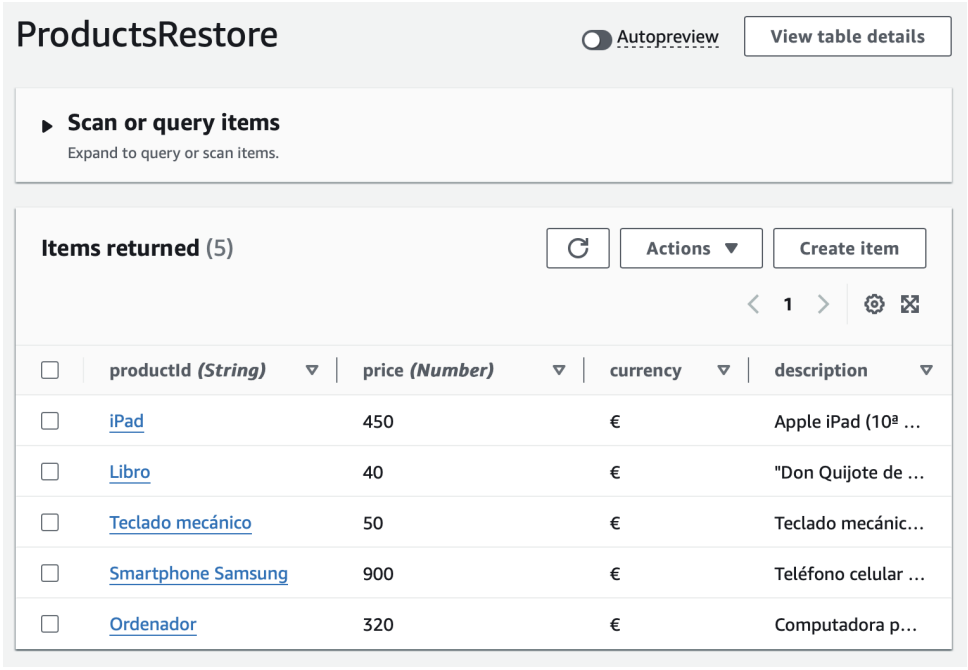

Figura 5.55: Nueva tabla DynamoDB con datos restaurados (interfaz web)

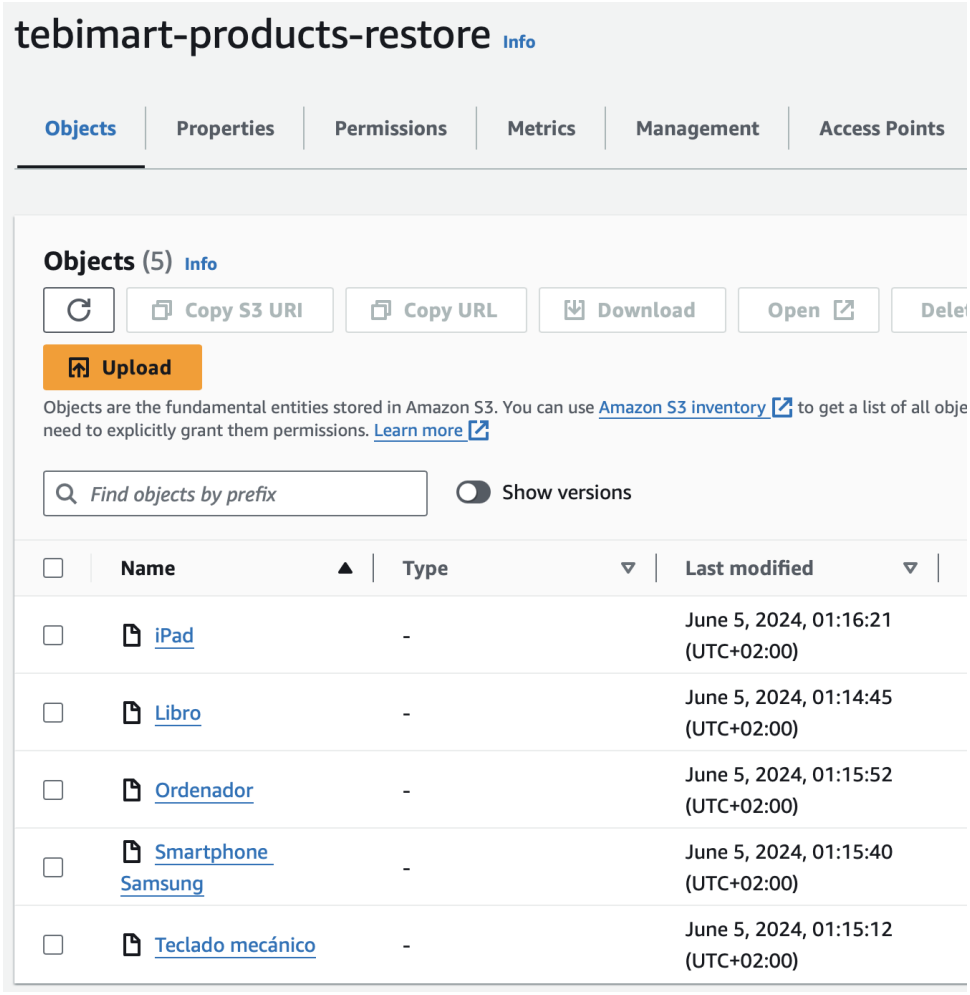

Figura 5.56: Nuevo bucket con datos restaurados (interfaz web)

Capítulo 6

Resultados

6.1. Arquitectura del sistema

En esta sección se presenta la arquitectura del sistema resultante tras la implementación de AWS y DevOps para el desarrollo de aplicaciones. Los resultados se estructuran en torno a los requisitos específicos del Caso de Estudio.

6.1.1. Arquitectura de desarrollo

La siguiente imagen muestra la arquitectura de desarrollo de la plataforma, con la que se implementa el flujo de trabajo de integración y entrega continua (CI/CD) utilizando varios servicios de AWS:

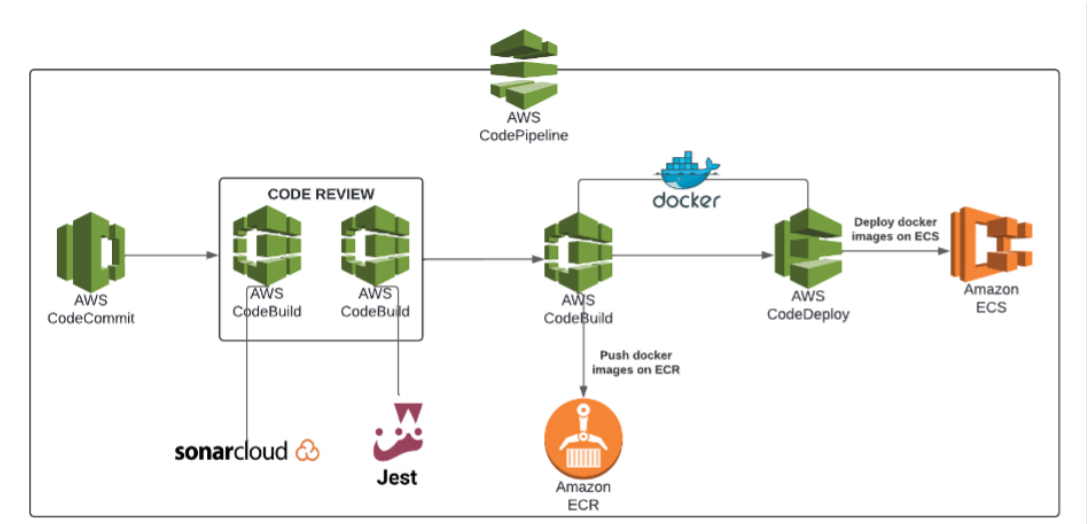

Figura 6.1: Arquitectura de desarrollo del sistema

Los componentes involucrados son:

1. AWS CodeCommit:
   - Función: actúa como repositorio de control de versiones donde los desarrolladores suben y gestionan el código fuente de la aplicación.

- Acción: los desarrolladores realizan cambios en el código y los envían al repositorio de CodeCommit.

2. AWS CodePipeline:
   - Función: orquesta todo el flujo de trabajo de CI/CD, automatizando los procesos de build, prueba y despliegue.
   - Acción: una vez que se detecta un nuevo commit en CodeCommit, CodePipeline se activa y comienza el proceso.
3. AWS CodeBuild:
   - Función: servicio de construcción que compila el código, ejecuta pruebas y produce artefactos de construcción.
   - Acción: con CodeBuild se realizan varias tareas:
     - Primera etapa: en la fase de "Code Review" se ejecutan pruebas software y se analiza la calidad del código con SonarCloud.
     - Segunda etapa: se construye la imagen Docker a partir del código fuente.
4. Docker:
   - Función: plataforma para desarrollar y ejecutar aplicaciones dentro de contenedores.
   - Acción: se utiliza en CodeBuild para construir una imagen de la aplicación y preparala para ser desplegada.
5. Amazon ECR (Elastic Container Registry):
   - Función: registro de contenedores que almacena y administra imágenes Docker.
   - Acción: las imágenes Docker construidas son subidas a Amazon ECR, donde se almacenan de manera segura.
6. AWS CodeDeploy:
   - Función: automatiza el proceso de despliegue de aplicaciones en servicios como Amazon ECS.
   - Acción: obtiene las imágenes Docker de ECR y las despliega en Amazon ECS.
7. Amazon ECS (Elastic Container Service):
   - Función: servicio de orquestación de contenedores que permite ejecutar y escalar aplicaciones Docker en un clúster administrado.
   - Acción: las imágenes Docker son desplegadas en ECS Fargate [43], donde se ejecutan como tareas dentro de los servicios configurados en el clúster.

Esta arquitectura asegura que los cambios en el código fuente se integren y despliegen de manera continua, garantizando la calidad y la disponibilidad de la aplicación a través de la automatización de los procesos de CI/CD.

6.1.2. Arquitectura operacional

La figura 6.2 corresponde a la arquitectura de operación del sistema. En ella se muestra como se gestionan las instancias y los recursos para asegurar la disponibilidad, el balanceo de carga, la escalabilidad y el almacenamiento seguro de los datos:

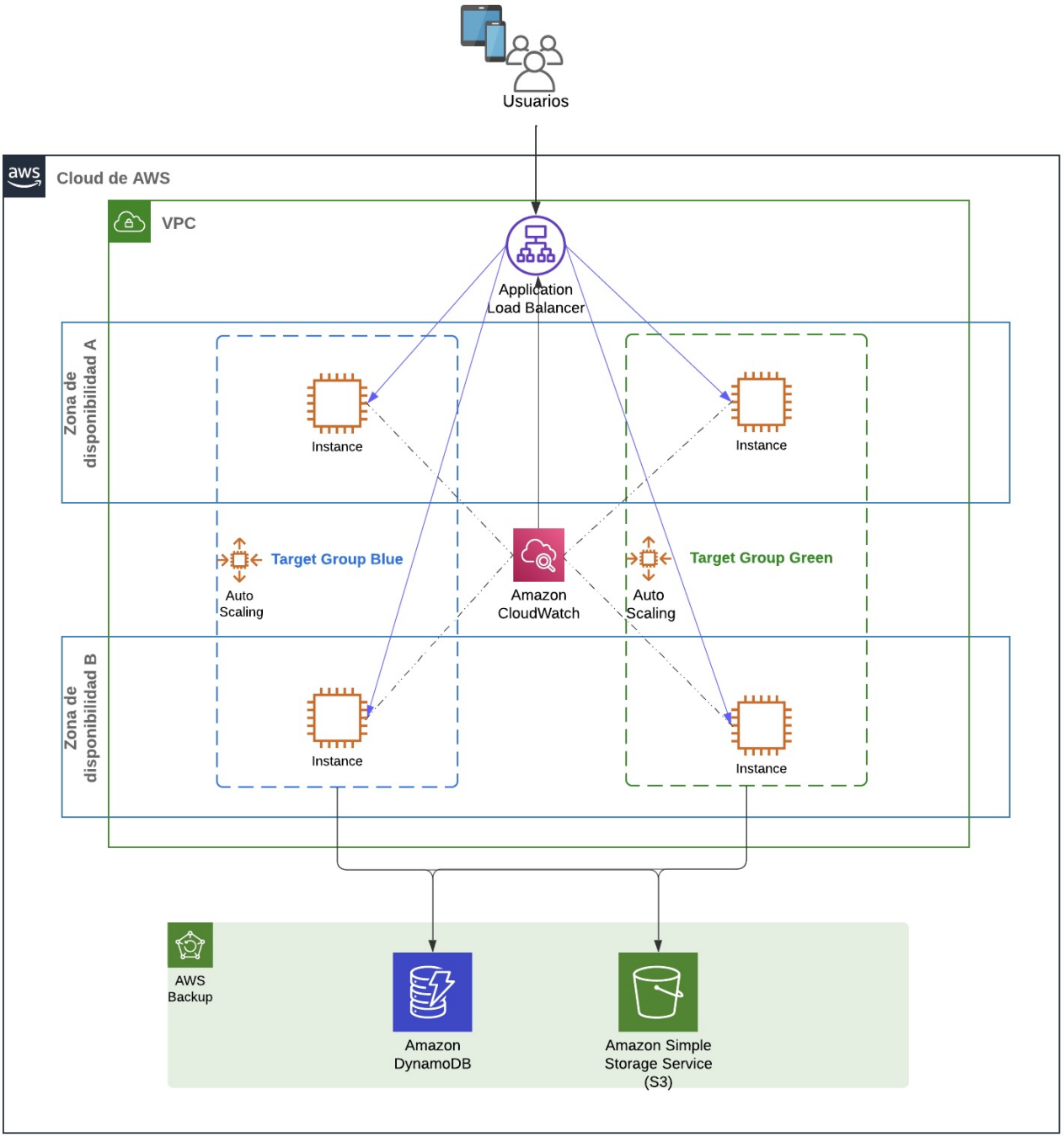

Figura 6.2: Arquitectura operacional del sistema

A continuación, se describe el flujo de operación y los componentes involucrados:

1. Virtual Private Cloud (VPC):
   - Función: proporciona un entorno de red virtual aislado dentro de la nube de AWS.
   - Acción: VPC crea una red privada virtual dentro de la infraestructura de AWS, aislada de la red pública de Internet y otras VPCs.
2. Application Load Balancer (ALB):
   - Función: distribuye las solicitudes entrantes entre varias instancias de la aplicación para equilibrar la carga y aumentar la disponibilidad.
   - Acción: recibe las solicitudes de los usuarios y las distribuye a las instancias en los distintos grupos de destino (Target Groups).
3. Instancias en Zonas de Disponibilidad:
   - Función: las instancias que ejecutan la aplicación están distribuidas en dos zonas de disponibilidad.
   - Acción: por defecto se despliegan dos instancias de la aplicación en cada zona de disponibilidad para asegurar la alta disponibilidad y tolerancia a fallos.
4. Target Groups (Blue y Green):
   - Función: conjuntos de instancias que reciben el tráfico distribuido por el ALB. Tal y como se describe en el punto relativo al despliegue en el apartado 5.1.6.2, se tiene un grupo activo (blue) y otro inactivo (green) .
   - Acción:
     - Target Group Blue: maneja la porción del tráfico activa y tiene configuraciones de autoescalado para ajustar dinámicamente el número de instancias según la demanda.
     - Target Group Green: grupo al que se redirige el tráfico cuando existe una actualización del sistema. También tiene configuraciones de autoescalado para ajustar el número de instancias según la demanda.
5. Auto Scaling:
   - Función: ajusta automáticamente el número de instancias en los grupos de destino según las métricas de uso proporcionadas por AWS Cloud Watch.
   - Acción: escala hacia arriba o hacia abajo el número de instancias para manejar la carga de manera eficiente.
6. Amazon CloudWatch:
   - Función: servicio de monitoreo y gestión de recursos que proporciona datos y métricas de rendimiento en tiempo real.
   - Acción: monitorea las instancias y genera alarmas y acciones automáticas según sea necesario, aportando información al ALB y al servicio de autoescalado para mantener la "salud" del sistema.
7. Servicios de Almacenamiento y Base de Datos:

- Función: proveen almacenamiento persistente y base de datos para la aplicación.
- Acción:
  - Amazon DynamoDB: almacena en tablas los datos de productos, usuarios, historial de compras etc.
  - Amazon Simple Storage Service (S3): alamcena contenido estático de la aplicación.

8. AWS Backup:
   - Función: servicio de gestión de copias de seguridad que permite crear y restaurar copias de seguridad de los datos y aplicaciones.
   - Acción: realiza copias de seguridad diarias de los datos almacenados en DynamoDB y S3 para garantizar la recuperación ante desastres.

Con esta arquitectura se consigue garantizar que la aplicación sea altamente disponible, escalable y este monitoreada continuamente para mantener un rendimiento óptimo y la resiliencia del sistema.

6.2. Análisis de costes

En el contexto de la implementación de soluciones en la nube, uno de los aspectos más críticos a considerar es el análisis de costos. La arquitectura presentada, que hace uso de múltiples servicios de AWS, ofrece una solución robusta y escalable para el desarrollo y operación de aplicaciones, sin embargo, para asegurar la viabilidad económica de esta implementación es fundamental desglosar y entender los costos asociados a cada componente de la arquitectura.

En este apartado cada uno de los componentes será analizado individualmente para determinar su impacto en el costo total de la solución. Se considerarán factores como el uso de recursos, la cantidad de datos procesados y almacenados, y las tarifas específicas de cada servicio. Una vez desglosados estos costos, se sumarán para proporcionar una visión global del costo total de la arquitectura.

Es importante señalar que, aunque se intenta ser lo más realista posible, los datos de uso son hipotéticos y en un entorno real van a variar considerablemente. Además, en este TFM se ha intentado maximizar el uso de la Capa Gratuita de AWS. Esta Capa Gratuida (también conocida como “Free Tier”) es un programa ofrecido por Amazon que permite a nuevos clientes probar varios servicios de AWS de forma gratuita durante un período de 12 meses [44]. Por otra parte, no se ha tenido en cuenta el costo asociado al uso de servicios externos a AWS seleccionados porque son open-source y gratuitos.

Para obtener una estimación de los costos asociados a cada servicio AWS se usa la **Calculadora de Costos** que proporciona Amazon en su web [45].

6.2.1. Arquitectura de desarrollo

1. AWS CodeCommit

- Costo del servicio: $1 por usuario activo al mes (5 usuarios gratuitos)
- Uso: 20 personas en el equipo de desarrollo
- Costo para la plataforma:

  5 usuarios x 0,00 USD = 0,00 USD

  15 usuarios x 1,00 USD = 15,00 USD

  Costo total de la capa: 0,00 USD + 15,00 USD = **15,00 USD costo por usuario mensual**

2. AWS CodeBuild
   - Costo del servicio: $0,005 por minuto de compilación
   - Uso: se tienen tres servicio CodeBuild, encargados de la revisión de código, ejecución de pruebas y dockerización. Se estima que el tiempo total de uso de CodeBuild será de 5 minutos y que se realizarán unas 500 compilaciones por mes.
   - Costo para la plataforma:

     500 compilaciones al mes x 5 minutos = 2500,00 minutos facturados (mensual)

     2500,00 minutos x 0,005 USD = 12,50 USD

     Costo de AWS CodeBuild (mensual): 12.50 USD

3. AWS CodePipeline
   - Costo del servicio: $1,00 por canalización activa al mes
   - Uso: se estima tener en principio dos canalizaciones por mes. Una para la rama de desarrollo, y otra para la rama operacional.
   - Costo para la plataforma:

     2 canalizaciones activas - 1 canalización activa gratuita cada mes = 1 total de canalizaciones activas al mes

     1 activar CodePipelines x 1,00 USD al mes = 1,00 USD

     Costo de CodePipeline (mensual): 1.00 USD

4. AWS CodeDeploy
   - Costo del servicio: no se aplica ningún cargo adicional por las implementaciones de código en Amazon ECS mediante AWS CodeDeploy.
   - Costo para la plataforma: **0 USD**

5. AWS Elastic Container Registry (ECR)
   - Costo del servicio: $0,10 por GB almacenado al mes
   - Uso: en un mes en el que se realizan 500 compilaciones se estima tener 40Gb de datos almacenados en el repositorio. Cada imagen generada pesa unos 80Mb aproximadamente.

- Costo para la plataforma:

  40 GB al mes x 0,10 USD = 4,00 USD

  Precios de Elastic Container Registry (mensual): 4.00 USD

6. AWS Elastic Container Service (ECS)

   - Costo del servicio: por CPU virtual por hora 0,012144 USD; por GB por hora 0,0013335 USD.
   - Uso: 2 instancias de la aplicación que estarán activas las 24 horas.
   - Costo para la plataforma:

     60,83 tareas x 0,50 CPU virtuales x 24 horas x 0,04048 USD por hora = 29,55 USD por las horas de vCPU

     60,83 tareas x 1,00 GB x 24 horas x 0,004445 USD por GB por hora = 6,49 USD por las horas de GB

     20 GB - 20 GB (sin cargo adicional) = 0,00 Almacenamiento efímero facturable por tarea (en GB)

     29,55 USD por las horas de vCPU + 6,49 USD por las horas de GB = 36,04 USD total

     Costo de Fargate (mensual): 36.04 USD

6.2.2. Arquitectura operacional

1. AWS Application Load Balancer (ALB)

   - Costo del servicio: 0,0225 USD por hora de balanceador de carga de aplicaciones (u hora parcial)
   - Uso: 1 balanceador de carga ejecutándose las 24 horas
   - Costo para la plataforma:

     1 balanceadores de carga x 0,0225 USD por hora x 730 horas en un mes = 16,43 USD

     Cargos fijos por hora del Balanceador de carga de aplicaciones (mensual): 16.43 USD

2. AWS CloudWatch

   - Costo del servicio: métricas de supervisión básicas (métricas enviadas desde los servicios de AWS de forma predeterminada) gratuitas.
   - Costo para la plataforma: **0 USD**

3. VPC

   - Costo del servicio: $0,01 por hora.
   - Uso: una VPC con dos subredes

- Costo para la plataforma:

  1 Puntos de enlace de la VPC x 2 ENI por punto de enlace de la VPC x 730 horas en un mes x 0,01 USD = 14,60 USD (costo mensual correspondiente al punto de conexión ENI)

  Costo mensual correspondiente a los puntos de conexión de la interfaz: 14,60 USD

4. AWS DynamoDB

   - Costo del servicio: unidades de solicitud de escritura (WRU) 1,25 USD por millón de unidades de solicitud de escritura. Unidades de solicitud de lectura (RRU) 0,25 USD por millón de unidades de solicitud de lectura

   - Uso: como se indicó en el apartado de desarrollo, el servicio DynamoDB se configura con el modo de pago por uso. En este modo no se cobra por la cantidad de datos almacenados sino por el número de solicitudes de lectura y escritura que se realizan.

     Actualizar la información de un producto o subir un nuevo producto consume aproximadamente 2 unidades de escritura. Obtener datos de un producto consume aprox 4 unidades de lectura.

     Para el caso de estudio se estima tener en línea 1 millón de productos activos por mes, 500.000 usarios activos y realizarse 2.5 millones de transacciones de compra y venta de productos. Esto se traduce en 8 millones de unidades de escritura por mes.

     Por otra parte, la información almacenada se listará alrededor de 120 millones de veces por mes, lo que se traduce a 240 millones de unidades de lectura por mes.

   - Costo para la plataforma:

     Número de escrituras: 8 millones por mes * 1000000 multiplicador = 8000000 por mes

     0,0009765625 Tamaño promedio del elemento en KB / 1 KB = 0,000976563 unidades de solicitud de escritura no redondeadas necesarias por elemento

     RoundUp (0.000976563) = 1 unidades de solicitud de escritura necesarias por elemento 8.000.000 número de escrituras x 1 parte estándar x 1 unidades de solicitud de escritura para escrituras estándar x 1 unidades de solicitud de escritura necesarias por elemento = 8.000.000,00 unidades de solicitud de escritura para escrituras estándar

     8.000.000 número de escrituras x 0 parte transaccional x 2 unidades de solicitud de escritura para escritura transaccional x 1 unidades de solicitud de escritura necesarias por elemento = 0,00 unidades de solicitud de escritura para escritura transaccional

     8.000.000,00 unidades de solicitud de escritura para escrituras estándar + 0,00 unidades de solicitud de escritura para escritura transaccional = 8.000.000,00 unidades totales de solicitud de escritura

8.000.000,00 unidades totales de solicitud de escritura x 0,00000125 USD = 10,00 USD costo de solicitud de escritura

Costo mensual de escritura (mensual): 10.00 USD

Número de lecturas: 240 millones por mes * 1000000 multiplicador = 240000000 por mes

0,0009765625 Tamaño promedio del elemento en KB / 4 KB = 0,000244141 unidades de solicitud de lectura no redondeadas necesarias por elemento

RoundUp (0.000244141) = 1 unidades de solicitud de lectura necesarias por elemento 240.000.000 número de lecturas x 1 parte coherente posterior x 0.5 unidades de solicitud de lectura para lecturas coherentes posteriores x 1 unidades de solicitud de lectura necesarias por elemento = 120.000.000,00 unidades de solicitud de lectura para lecturas coherentes posteriores

240.000.000 número de lecturas x 0 parte coherente fuerte x 1 unidades de solicitud de lectura para lecturas coherentes fuertes x 1 unidades de solicitud de lectura necesarias por elemento = 0,00 unidades de solicitud de lectura para lecturas coherentes fuertes

240.000.000 número de lecturas x 0 parte transaccional x 2 unidades de solicitud de lectura para lecturas transaccionales x 1 unidades de solicitud de lectura necesarias por elemento = 0,00 unidades de solicitud de lectura para lecturas transaccionales

120.000.000,00 unidades de solicitud de lectura para lecturas coherentes posteriores + 0,00 unidades de solicitud de lectura para lecturas coherentes fuertes + 0,00 unidades de solicitud de lectura para lecturas transaccionales = 120.000.000,00

Unidades totales de solicitud de lectura 120.000.000,00

Unidades totales de solicitud de lectura x 0,00000025 USD = 30,00 USD costo de solicitud de lectura

Costo de lectura mensual (mensual): 30.00 USD

5. AWS S3

   - Costo del servicio: Primeros 50 TB/mes a 0,023 USD por GB; siguientes 450 TB/mes a 0,022 USD por GB; más de 500 TB/mes a 0,021 USD por GB.
   - Uso: se almacenarán alrededor de 500GB de datos estáticos y se estima que de media se realizarán 4 millones de accesos a la base de datos por mes.
   - Costo para la plataforma:

     500 GB x 0,023 USD = 11,50 USD Costo total de la capa = 11,50 USD (coste de almacenamiento en S3 Estándar)

     4.000.000 Solicitudes PUT para almacenamiento de S3 Standard x 0,000005 USD por solicitud = 20,00 USD (coste de solicitudes PUT en S3 Estándar)

     4.000.000 Solicitudes GET en un mes x 0,0000004 USD por solicitud = 1,60 USD (coste de solicitudes GET en S3 Estándar)

11,50 USD + 1,60 USD + 20,00 USD = 33,10 USD (Total de almacenamiento de S3 Standard, solicitudes de datos, coste de S3 Select)

Coste de S3 Estándar (mensual): 33.10 USD

6. AWS Backup
   - Costo del servicio: 0,05 USD por GB al mes para S3; 0,05 USD por GB al mes para DynamoDB.
   - Uso: 100 GB para las tablas DynamoDB y 500GB para S3.
   - Costo para la plataforma:

     100GB x 0,05 USD = 5 USD DynamoDB

     500GB x 0,05 USD = 25 USD S3

     Coste de AWS Backup (mensual): 30 USD

6.2.3. Visión general

La siguiente tabla muestra una visión más general del coste de cada servicio.

Servicio	**Costo mensual**
AWS CodeCommit	$15,00
AWS CodeBuild	$12,50
AWS CodePipeline	$1,00
AWS CodeDeploy	$0,00
AWS ECR	$4,00
AWS ECS	$36,04
AWS ALB	$16,43
AWS CloudWatch	$0,00
AWS VPC	$14,60
AWS DynamoDB	$40,00
AWS S3	$33,10
AWS Backup	$30,00

Cuadro 6.1: Costo mensual de los servicios AWS

Costo total al mes = $202,67

Costo total anual = $2.432,04

Ahora bien, al costo total hay que añadirle los impuestos que pueden aplicarse a los clientes de Amazon Web Services. En el caso concreto de España se aplica el 21 % de IVA tal y como se recoge en la Tabla de tipos de IVA de Amazon [46].

	Sin taxes	**Con taxes (21 %)**
Mensual	$202,67	$245,23
Anual	$2.432,04	$2.942,77

Cuadro 6.2: Costo mensual y anual de la arquitectura

6.3. Sistema en operación

En esta sección se va a visualizar el funcionamiento del sistema siguiendo un enfoque práctico: para mostrar el proceso de integración y despliegue continuo se modificará el código actual y se seguirán paso a paso las acciones que se realizan desde que se hace el cambio hasta que se pone en operación en la nueva versión de software.

Situación inicial

La situación inicial de la plataforma se ve reflejada en la siguiente figura que presenta la página principal de *Tebimart* con los productos disponibles.

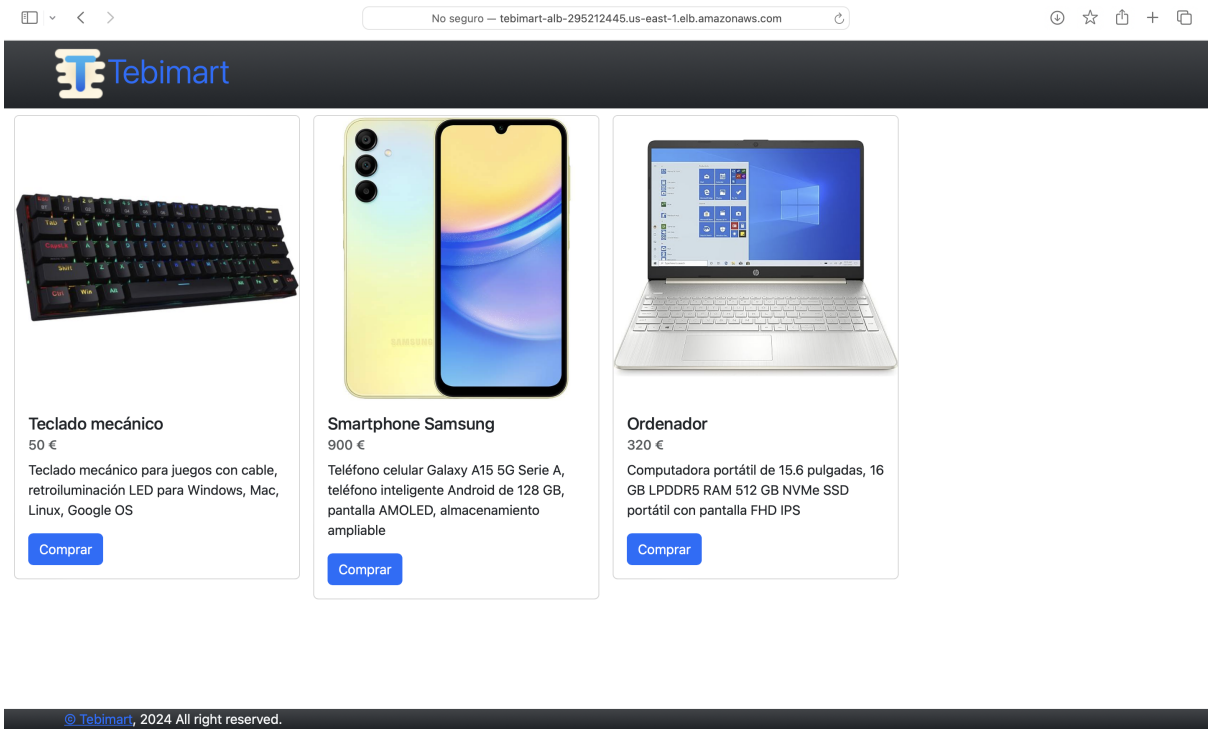

Figura 6.3: Página principal de la plataforma

Cambio de color

Se va a realizar un cambio bastante simple en el código que modificará el color de fondo de los productos, pasando de blanco a un color azul:

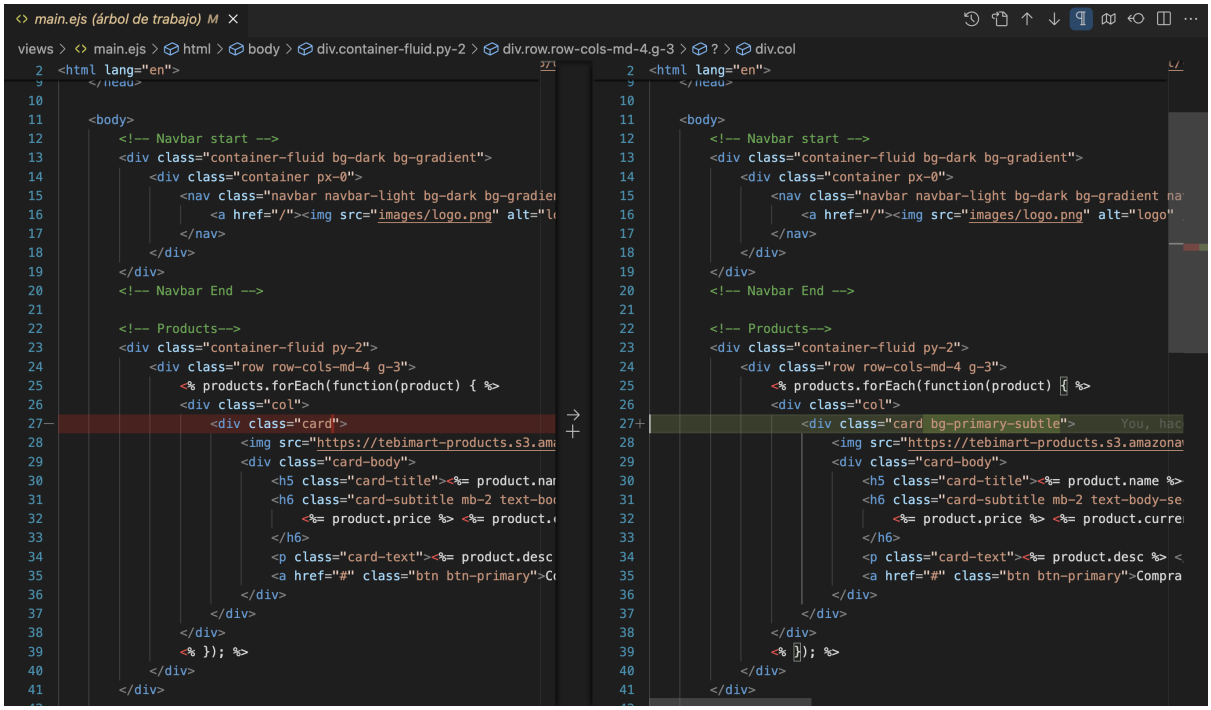

Figura 6.4: Cambio de color en la descripción de productos

Se hace un commit del cambio para iniciar el proceso:

```
esteban@mackbook:~/Documents/UPM/TFM/tebimart $ git status
On branch develop
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   views/main.ejs

no changes added to commit (use "git add" and/or "git commit -a")
esteban@mackbook:~/Documents/UPM/TFM/tebimart $ git add views/main.ejs
esteban@mackbook:~/Documents/UPM/TFM/tebimart $ git commit -m "Change products description color"
[develop 356fd7e] Change products description color
 1 file changed, 1 insertion(+), 1 deletion(-)
esteban@mackbook:~/Documents/UPM/TFM/tebimart $ git push origin develop
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 383 bytes | 383.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Validating objects: 100%
To https://git-codecommit.us-east-1.amazonaws.com/v1/repos/tebimart
   5131c5d..356fd7e  develop -> develop
esteban@mackbook:~/Documents/UPM/TFM/tebimart $
```

Figura 6.5: Commit y push de local al repositorio remoto

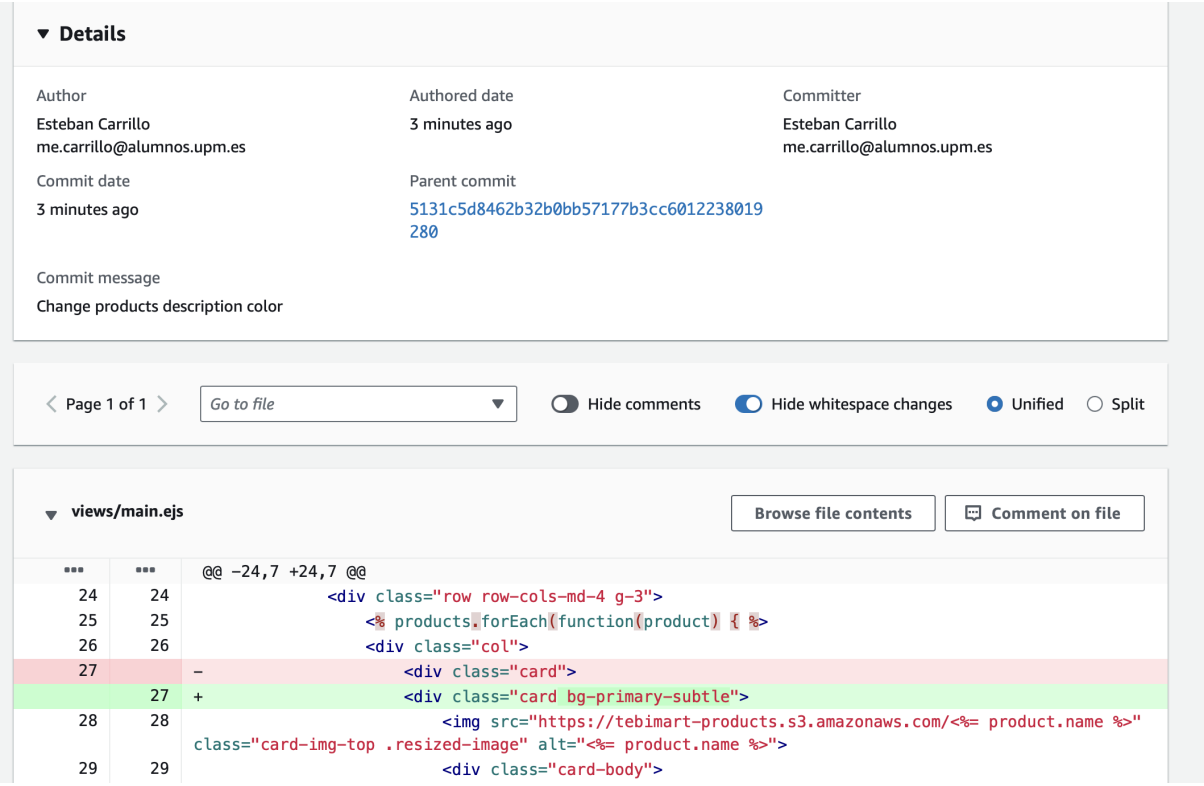

Figura 6.6: Vista de los cambios en AWS CodeCommit

Inicia el proceso

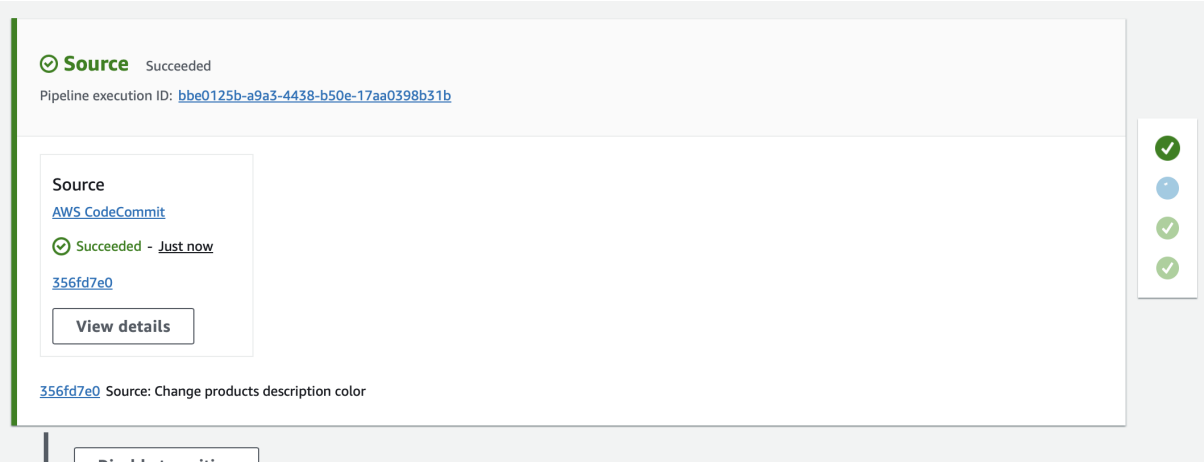

Figura 6.7: AWS CodePipeline - Fase de captura de código completada

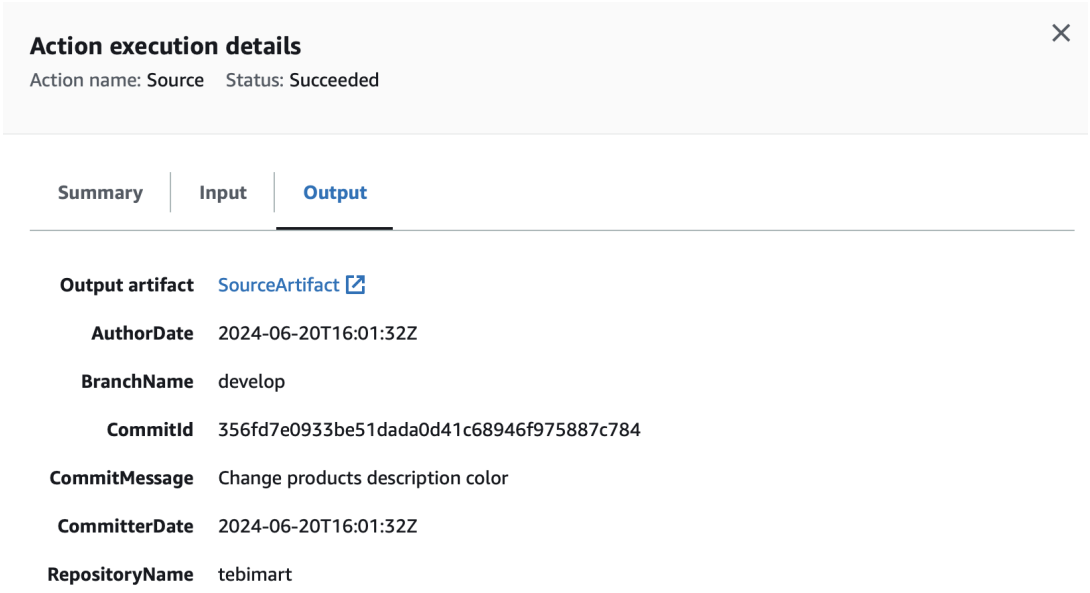

Figura 6.8: Detalles de la captura de código

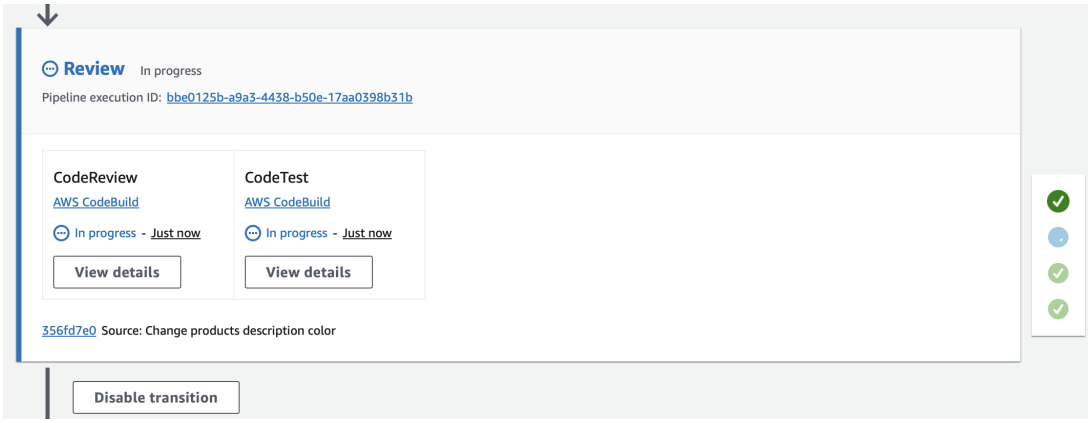

Figura 6.9: AWS CodePipeline - Fase de review

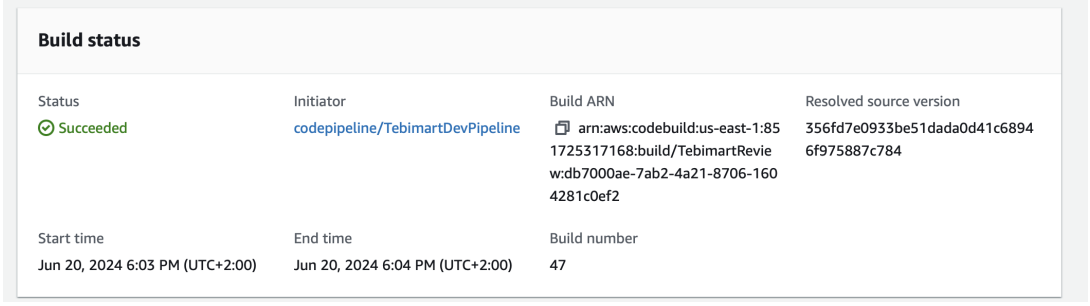

Figura 6.10: AWS CodeBuild - Fase de análisis de código exitosa

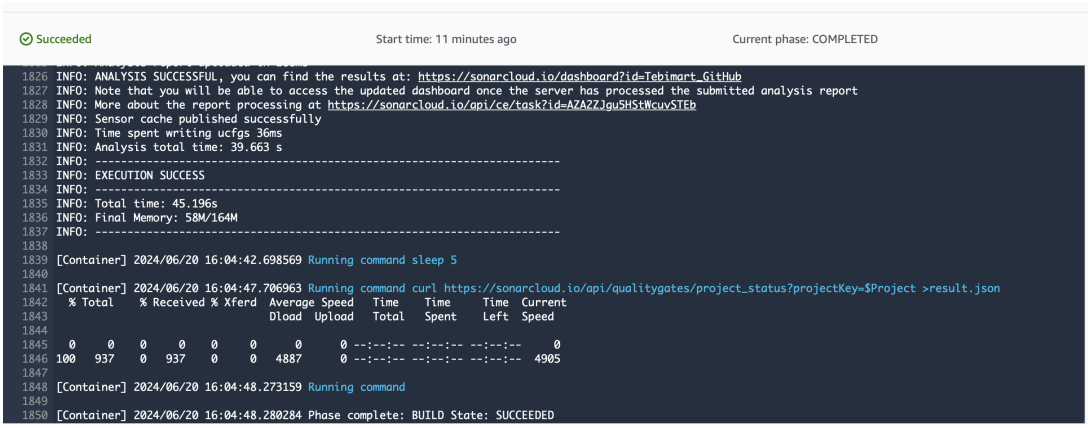

Figura 6.11: Log de la fase de análisis de código

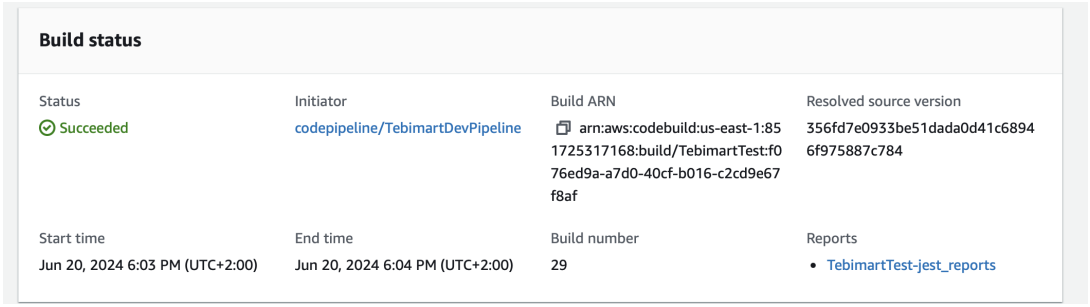

Figura 6.12: AWS CodeBuild - Fase de ejecución de pruebas exitosa

Figura 6.13: Log de la ejecución de pruebas

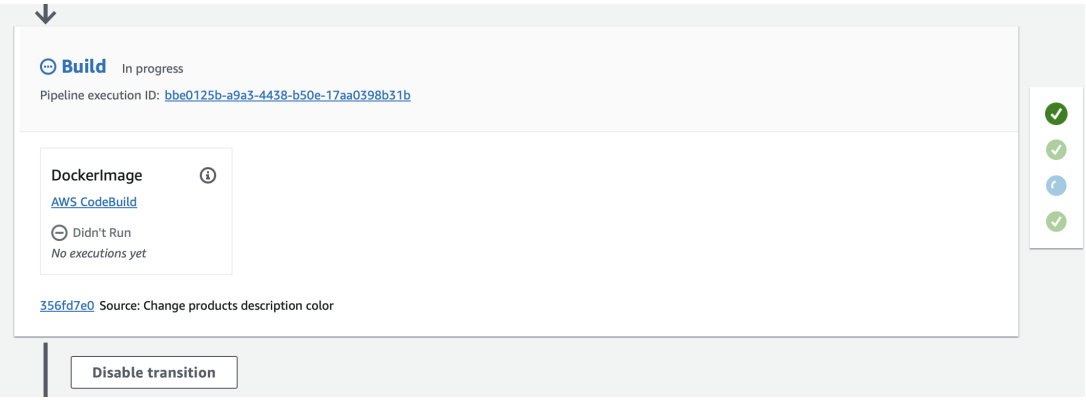

Figura 6.14: AWS CodePipeline - Fase de creación de imagen Docker

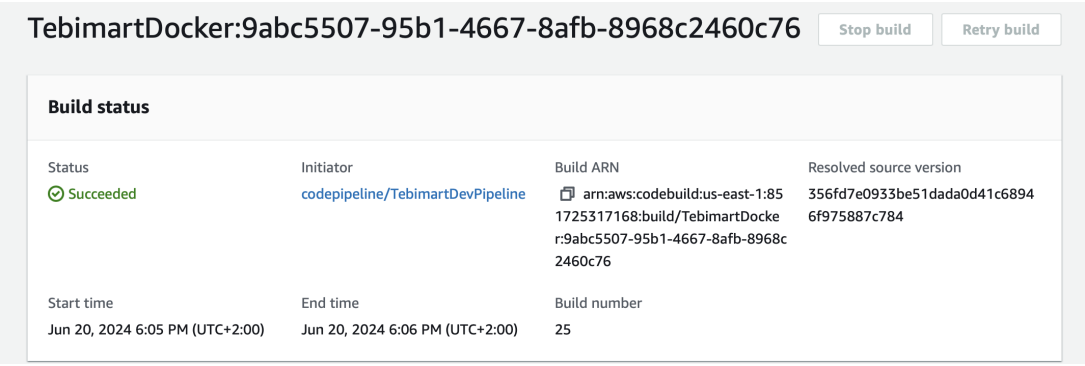

Figura 6.15: AWS CodeBuild - Imagen docker generada correctamente

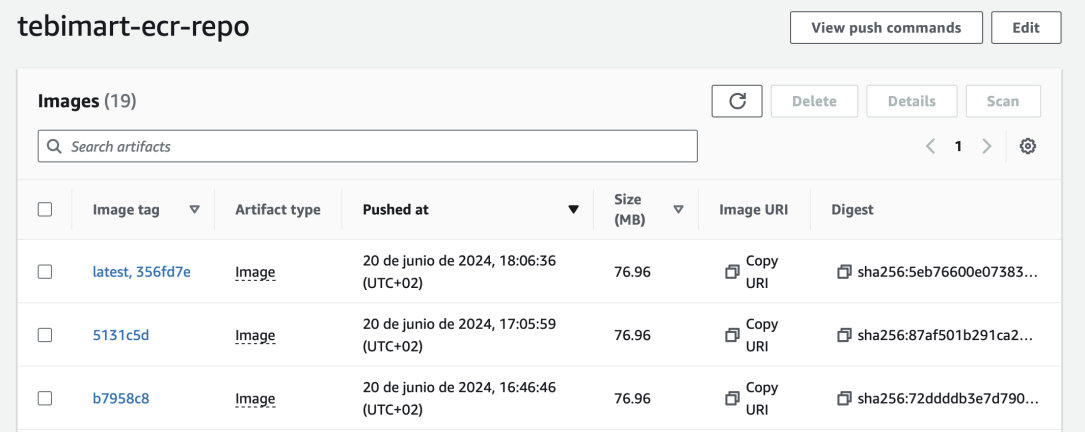

Figura 6.16: Nueva imagen docker guardada en AWS ECR

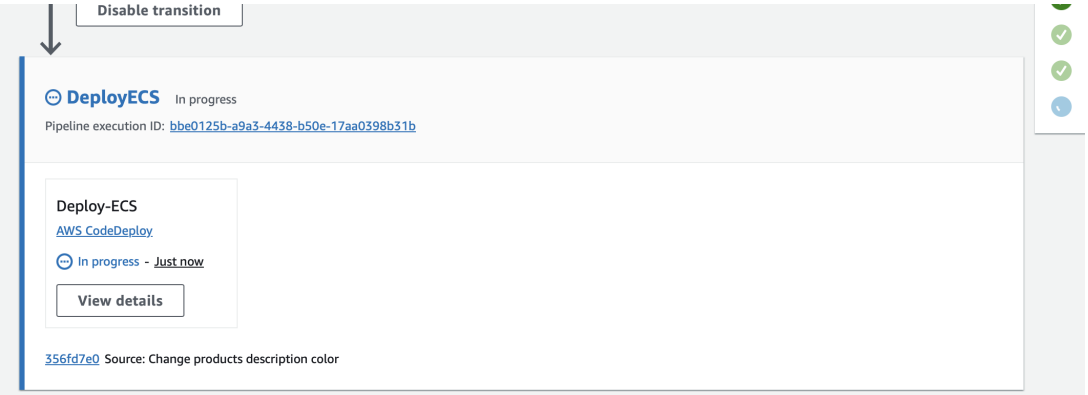

Figura 6.17: AWS CodePipeline - Fase de despliegue

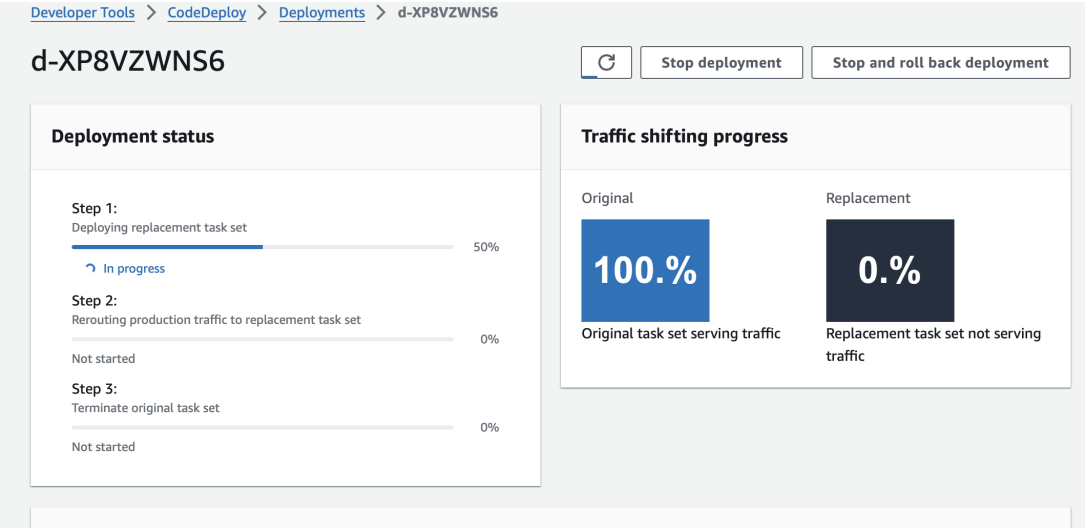

Figura 6.18: AWS CodeDeploy - Cambio de versión de software en proceso

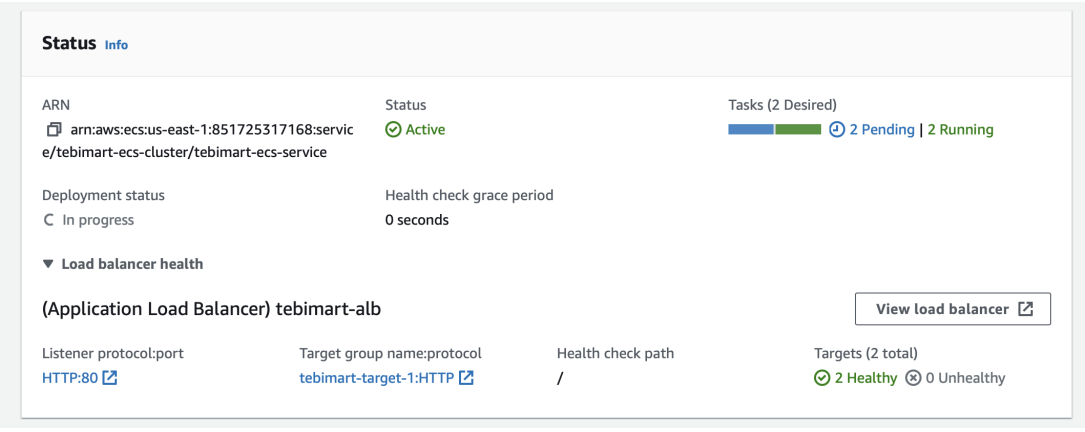

Figura 6.19: Estado del despliegue en AWS ECS

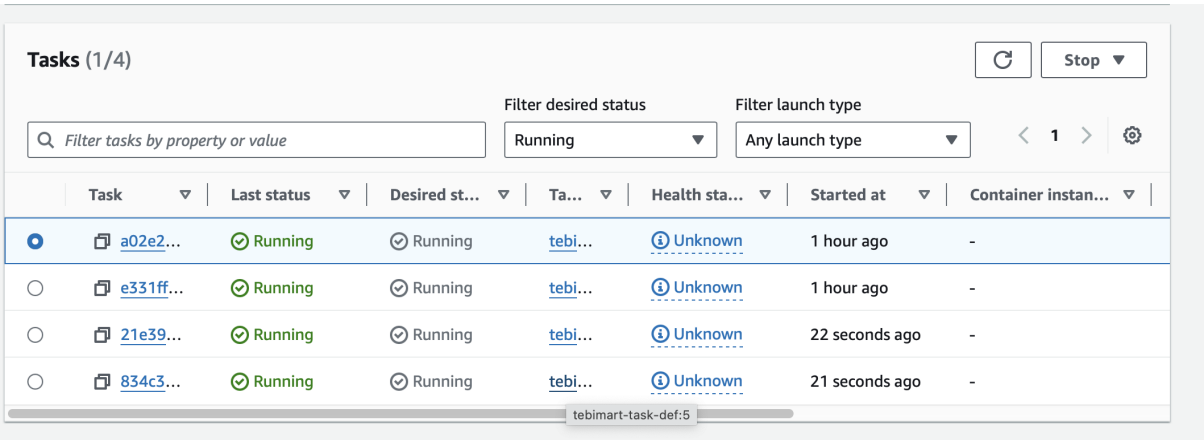

Figura 6.20: AWS ECS - Tareas desplegándose

Esta figura 6.20 muestra como se lanzan dos nuevas instancias de la aplicación con la nueva versión de software mientras las otras dos antiguas se siguen ejecutando. Cuando las nuevas instancias alcanzan un estado estable, las antiguas dejan de ejecutarse, como se muestra en la figura 6.23

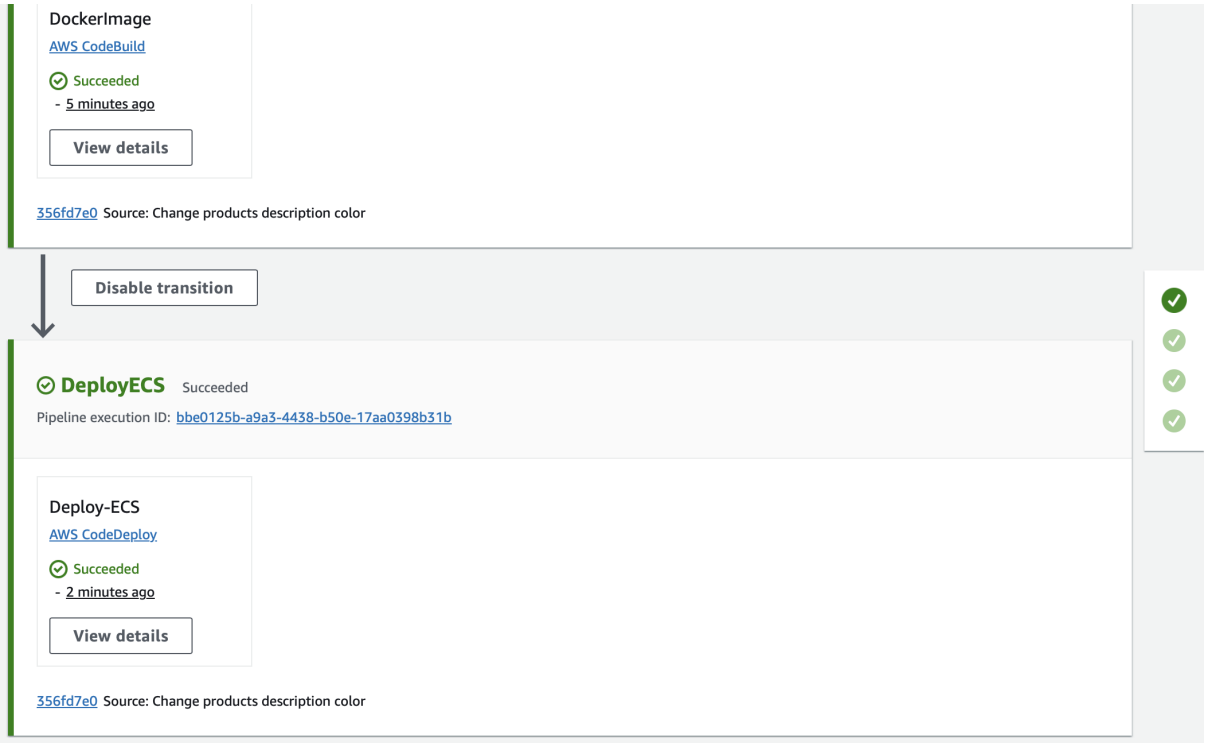

Figura 6.21: AWS CodePipeline - Despliegue completado

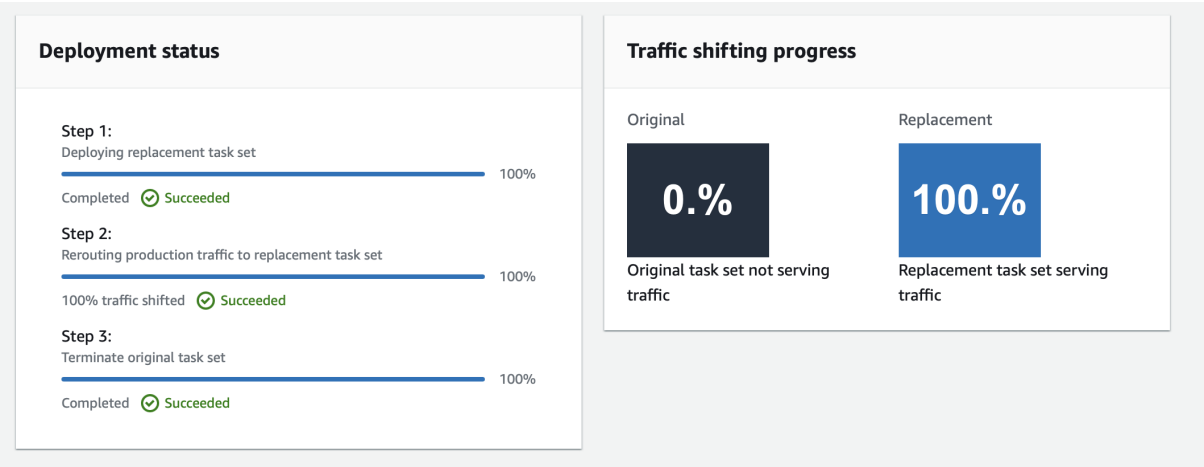

Figura 6.22: AWS CodeDeploy - Despliegue de software completado

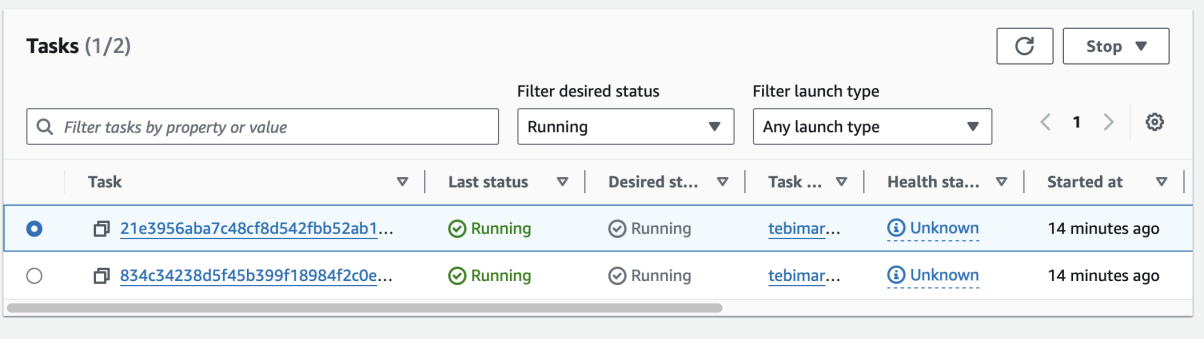

Figura 6.23: AWS ECS - Tareas finales ejecutándose

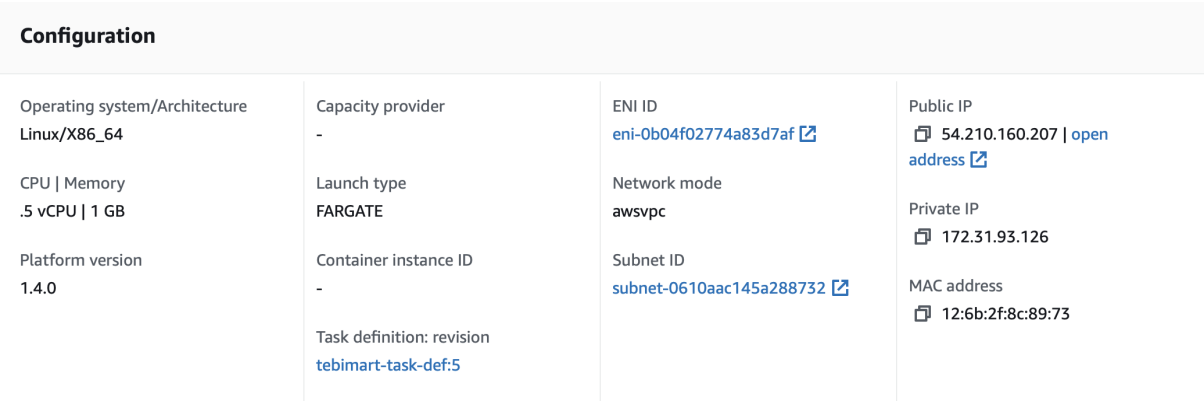

Figura 6.24: Configuración de la primera tarea/instancia en ejecución

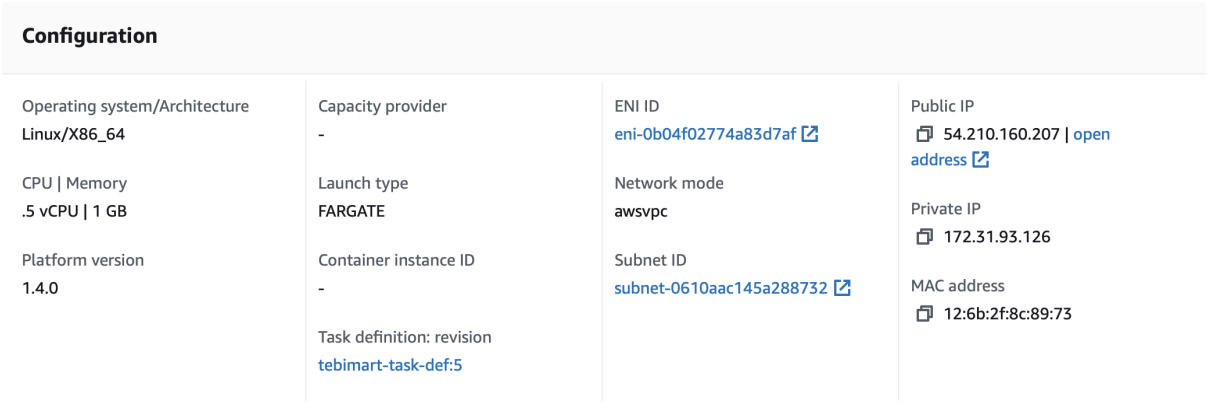

Figura 6.25: Configuración de la segunda tarea/instancia en ejecución

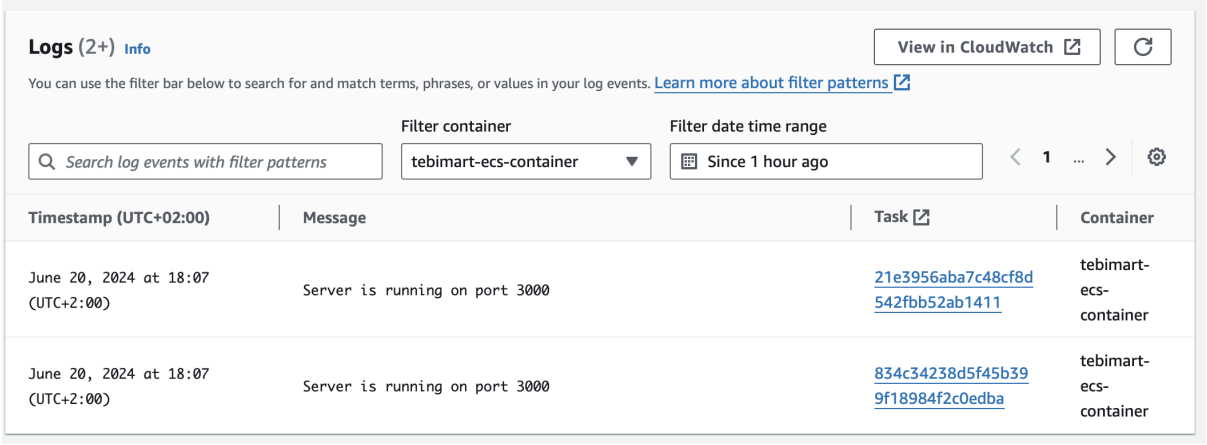

Figura 6.26: AWS ECS - Logs de las tareas

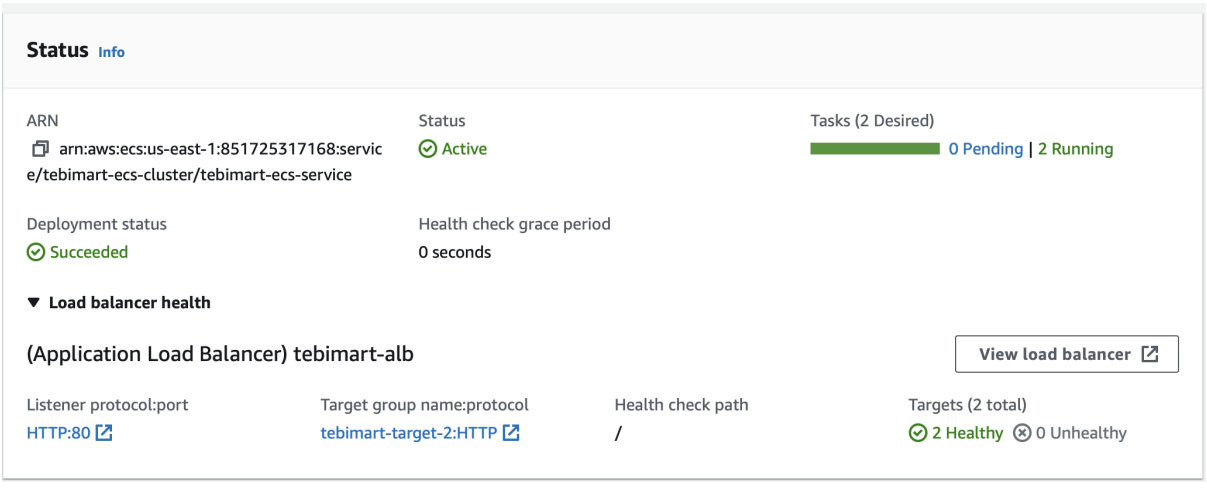

Figura 6.27: AWS ECS - Situación final del despliegue

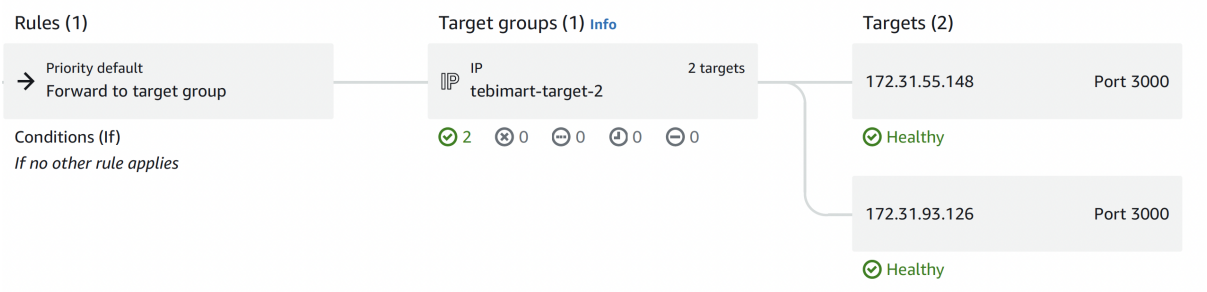

Figura 6.28: Balanceador de carga distribuyendo el tráfico a las dos tareas desplegadas

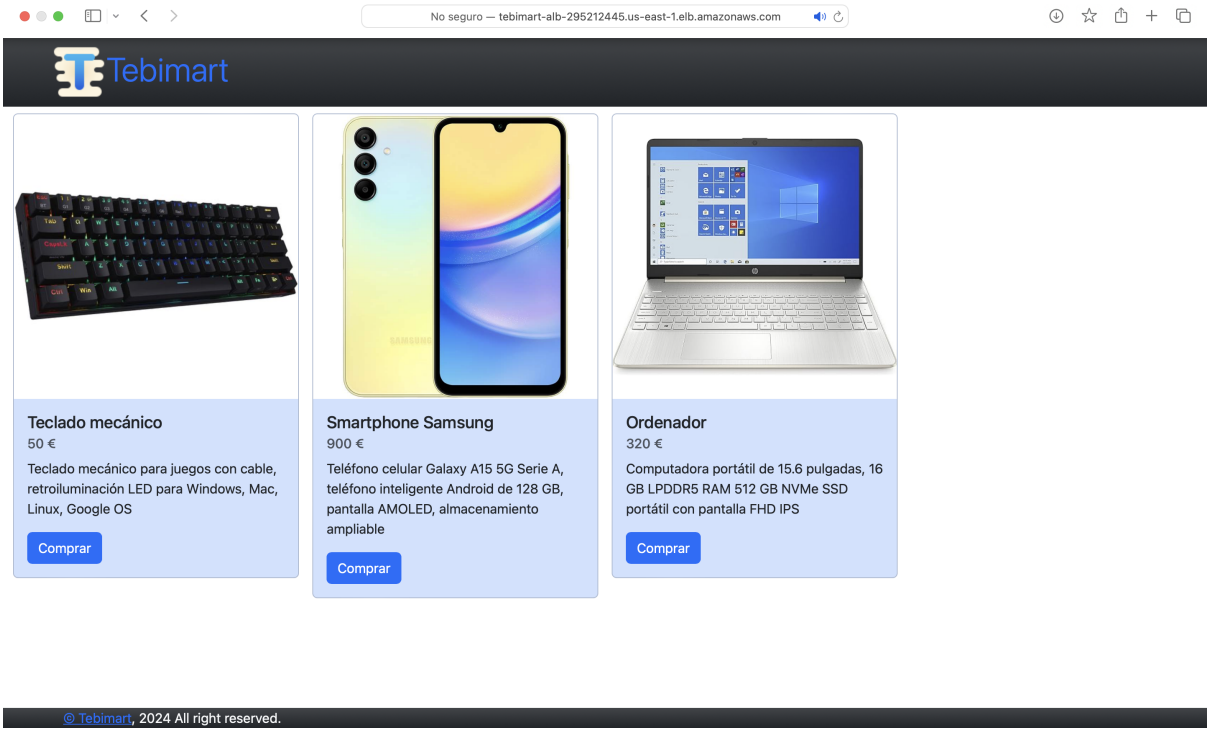

Figura 6.29: Página principal de la plataforma después del cambio de software

Como se puede observar el proceso de integración y despliegue continuo se ha realizado correctamente en este caso, pasando de la situación inicial que se muestra en la figura 6.3 a la situación final con el cambio de color efectuado en la figura 6.29. Tal y como se ha configurado, todo el proceso se ha realizado de forma automática sin intervención humana en ningún momento, salvo cuando se hizo el cambio de código. Si se compara la hora en la que se inicia el proceso (16:01:32Z como se muestra en la figura 6.8) con la hora en la que las tareas finales están puestas en operación (16:07:00Z como se muestra en la figura 6.26) se tiene que el tiempo tomado efectuar todo el proceso ha sido de aproximadamente 6 minutos.

Capítulo 7

Limitaciones e impacto

Durante el desarrollo de este Trabajo Final de Máster, se han encontrado diversas limitaciones que han influido en la implementación y resultados del proyecto. En las siguientes secciones se detallan las principales limitaciones identificadas y su impacto.

7.1. Limitaciones

Extensa documentación y dificultad de análisis en la plataforma AWS

La plataforma AWS ofrece una amplia y completa documentación sobre sus productos y servicios, disponible en diversos formatos y canales:

- Sitio web de AWS: el recurso central de documentación de AWS es su sitio web, donde se encuentran guías detalladas, tutoriales, ejemplos de código y referencias de API para todos los productos y servicios de AWS.
- Guías para principiantes: AWS ofrece guías específicas para principiantes como ayuda a familiarizarse con la plataforma y sus servicios básicos.
- Documentación de productos individuales: cada producto o servicio de AWS tiene su propia página de documentación dedicada, que contiene información detallada sobre sus características, configuración, uso y mejores prácticas.
- Tutoriales: AWS ofrece una amplia colección de tutoriales paso a paso que guían a través de tareas comunes y escenarios específicos de uso.
- Ejemplos de código: AWS proporciona ejemplos de código en diversos lenguajes de programación que muestran cómo utilizar los servicios de AWS.
- Referencias de API: para desarrolladores, AWS ofrece referencias de API completas que describen en detalle las funciones y parámetros de cada servicio.
- Foro de AWS: el foro de AWS es una comunidad en línea donde puede hacer preguntas y obtener ayuda de otros usuarios y expertos de AWSs.
- Cursos de capacitación: AWS ofrece una variedad de cursos de capacitación en línea para ayudar a aprender a usar sus productos y servicios.

El conjunto total es una extensa documentación que está en continuo crecimiento. En un principio se puede ver como algo positivo ya que se puede encontrar todo lo que se necesita para la configuración de cada uno de los servicios. Sin embargo, la gran cantidad de información disponible puede resultar abrumadora y dificultar el análisis detallado y la comprensión de todas las características y configuraciones necesarias.

Para poner un ejemplo, uno de los varios casos en el que esta limitación se evidenció fue durante el proceso de despliegue de la aplicación en una instancia EC2. Se dió la situación de que el desplligue no se realizaba correctamente debido a fallos en la configuración de instancia que necesitaba unos permisos especiales para ejecutar la aplicación. En la página principal de la documentación no se indicaba explícitamente, pero se hacía referencia a otras páginas en las que sí y la búsqueda de esa información entre toda la documentación consumío bastante tiempo.

Incidencias

A lo largo del desarrollo del TFM se presentaron incidencias con algunos de los servicios utilizados que no se pudieron resolver por completo porque la búsqueda de una solución estaba retrasando demasiado el resto de tareas. Al tratarse de incidencias menores y que no aportaban valor significativo para el TFM se decidió implementar medidas alternativas para no poner en riesgo la completa ejecución del proyecto. Entre las incidencias presentadas están:

1. Falta de generación de reportes de test en AWS CodeBuild
    - Descripción: se tiene un problema en el servicio AWS CodeBuild cuando se ejecutan los test de la aplicación que implica que no se generen reportes de test de manera automática. Esta limitación dificulta la evaluación del estado y la calidad del código después de cada construcción, ya que no se dispone de un resumen detallado de los resultados de las pruebas y no se puede automatizar por completo el flujo de trabajo.
    - Posible causa: falta de configuración en algún punto del servicio o buildspec. Es necesario analizar más en profundidad la documentación de AWS para encontrar la solución.
    - Solución alternativa: al no poder obtener los resultados de los test, en el flujo de trabajo se omite el chequeo de estado y se da por bueno siempre y cuando todos los tests se ejecuten.
2. Sonarqube no tiene en cuenta los test creados
    - Descripción: la herramienta SonarCloud utilizada para la inspección continua de la calidad del código, no tiene en cuenta los test creados para el cálculo de cobertura de pruebas. Esto limita la capacidad de análisis integral de la calidad del código.
    - Posible causa: falta de configuración en algún punto del servicio SonarCloud. Es necesario analizar más en profundidad el servicio para encontrar la solución.
    - Solución alternativa: debido a que la cobertura de pruebas es siempre 0 % en el resultado devuelto por SonarCloud se ha comentado la línea que hace este chequeo en el fichero *buildspec* del servicio AWS CodeBuild.

Modelo de pago

Aunque para este proyecto se intentó utilizar la capa gratuita de AWS para minimizar los costos, hubieron casos en los que se incurrieron en cargos adicionales por el uso de servicios que no estaban incluidos en dicha capa.

Este modelo de pago, basado en el uso, presenta un desafío adicional al requerir una monitorización y gestión constante de los recursos utilizados para evitar costos imprevistos que puedan afectar a una empresa.

7.2. Impacto

Las limitaciones mencionadas anteriormente han tenido un impacto significativo en el proyecto, tanto en términos de tiempo como de recursos.

La necesidad de revisar y comprender una gran cantidad de documentación ha retrasado en cierta medida el proceso de implementación y búsqueda de soluciones para los problemas. Además, en algunas ocasiones la configuración final de los recursos fueron resultado de múltiples iteraciones de prueba, error y reflexión de los pasos realizados, incrementando el tiempo dedicado a la fase de "Desarrollo" del proyecto.

Por otra parte, las incidencias encontradas relativas a las acciones de "Review" y "Test" afectan la capacidad de monitorizar y evaluar el estado del software de manera efectiva. Esto provoca una disminución en la eficiencia del ciclo de desarrollo ya que no se reflejan adecuadamente los resultados reales.

7.3. Reflexión sobre las limitaciones

A pesar de las limitaciones y los desafíos encontrados, la experiencia adquirida en la resolución de estos problemas ha sido invaluable. Las dificultades enfrentadas proporcionaron una comprensión más profunda de los servicios de AWS y las herramientas DevOps, y destacan puntos clave a tener en cuente en futuros proyectos:

- Curva de aprendizaje: la adopción de nuevas tecnologías y prácticas requiere tiempo y capacitación para los equipos de desarrollo y operaciones. Se resalta la importancia de una documentación clara y accesible.
- Integración de distintas herramientas: la integración de varias herramientas y tecnologías en un proyecto puede presentar desafíos de compatibilidad y requerir configuraciones adicionales.
- Gestión de costos: aunque el modelo de pago por uso ofrece ventajas, es esencial monitorear y gestionar los costos continuamente para evitar gastos inesperados.

En general, estos desafíos destacan en cierta medida la necesidad de un enfoque proactivo para la gestión y resolución de problemas en plataformas complejas.

A pesar de que el impacto global de la adopción de AWS y DevOps ha sido positivo, es importante seguir trabajando en las limitaciones identificadas, sobre todo en las incidencias abiertas que dejan un trabajo pendiente a analizar y resolver en un futuro cercano.

Capítulo 8

Discusión

En este capítulo se discute cómo y en qué medida se han alcanzado los objetivos específicos planteados en el proyecto. El objetivo principal era investigar y demostrar cómo la integración de AWS y DevOps puede revolucionar el desarrollo de aplicaciones, aumentando la eficiencia, reduciendo los tiempos de despliegue y mejorando la calidad del software. Las siguientes secciones analizan cada objetivo específico y su grado de cumplimiento.

8.1. O1. Investigar y analizar requisitos

Este objetivo ha sido fundamental para establecer una base sólida para el proyecto. Se llevó a cabo una especificación y análisis detallado de los requisitos del Caso de Estudio, identificando las necesidades específicas de una plataforma de comercio electrónico. Este análisis incluyó la selección de tecnologías aplicables para cumplir con cada requisito y así mismo la evaluación de los costos asociados a la implementación y operación de la plataforma.

Logros:

- Identificación clara de los requisitos funcionales y no funcionales de la plataforma.
- Evaluación de las tecnologías disponibles para cumplir con los requisitos.
- Evaluación de los costos asociados a la implementación y operación de la plataforma.

Este objetivo se ha alcanzado en su totalidad, proporcionando un marco claro para el desarrollo y la implementación de la plataforma.

8.2. O2. Comprender el Marco de DevOps

La comprensión del marco de DevOps fue esencial para el éxito del proyecto. Se exploraron los principios, prácticas y herramientas de DevOps, evaluando su aplicabilidad para el desarrollo de la plataforma. Esto incluyó la revisión de prácticas de integración continua (CI), entrega continua (CD), infraestructura como código (IaC) y monitoreo continuo.

Logros:

- Profunda comprensión de los principios y prácticas de DevOps.
- Identificación de herramientas de CI/CD clave como AWS CodePipeline, AWS CodeBuild y AWS CodeDeploy para su implementación en la plataforma.

Este objetivo se ha alcanzado satisfactoriamente, proporcionando la base teórica y práctica necesaria para implementar DevOps en el proyecto.

8.3. O3. Establecer Estrategias DevOps

El diseño y la planificación de estrategias DevOps específicas fueron cruciales para el desarrollo, despliegue, monitoreo y mantenimiento de la plataforma. Se formularon estrategias para optimizar el flujo de trabajo, mejorar la calidad del software y reducir los tiempos de respuesta ante cambios, conflictos y actualizaciones.

Logros:

- Implementación de pipelines de CI/CD para automatizar el proceso de desarrollo y despliegue.
- Definición de prácticas de monitoreo continuo utilizando servicios de AWS como CloudWatch.
- Estrategias de IaC implementadas mediante AWS CLI para gestionar la infraestructura de manera eficiente.

Este objetivo se ha alcanzado de manera efectiva, estableciendo un flujo de trabajo optimizado y eficiente para el desarrollo y mantenimiento de la plataforma.

8.4. O4. Utilizar servicios AWS

La evaluación y selección de los servicios de AWS más adecuados fue esencial para respaldar los requisitos técnicos y operativos de la plataforma. Se identificaron y utilizaron servicios de computación, almacenamiento, bases de datos, respaldo y herramientas de gestión que mejor se adaptaran a las necesidades del proyecto.

Logros:

- Selección y configuración de servicios clave de Amazon como AWS CodePipeline, AWS CodeBuild, AWS CodeDeploy, AWS ECS, AWS ECR.
- Uso eficiente de servicios de AWS para asegurar la escalabilidad, respaldo y alta disponibilidad de la plataforma: AWS AutoScaling, AWS DynamoDB, AWS S3, AWS Backup, AWS Application Load Balancer.
- Integración de múltiples servicios de AWS junto con otros servicios para crear una infraestructura robusta y flexible: AWS CodeBuild + SonarCloud, AWS CodeDepoy + AWS ECS + AWS ECR + Docker.

Este objetivo se ha alcanzado con éxito, asegurando que la infraestructura de la plataforma esté bien soportada por los servicios de AWS.

8.5. O5. Evaluar beneficios y desafíos

El análisis de los resultados obtenidos de la implementación de DevOps y AWS fue crucial para identificar los beneficios logrados y los desafíos enfrentados durante el proceso. Esto proporcionó una visión completa de las oportunidades y limitaciones de estas tecnologías.

Logros:

- Identificación de beneficios significativos como la mejora en la eficiencia operativa y aumento de la calidad del software al establecer procesos de desarrollo, pruebas y despliegue totalmente automatizados.
- Reconocimiento de desafíos como la complejidad de la configuración inicial y la necesidad de formación continua.
- Documentación de lecciones aprendidas y estrategias para superar los desafíos enfrentados.

Este objetivo se ha alcanzado ampliamente, proporcionando una evaluación detallada y útil para futuros proyectos de la implementación de DevOps y tecnologías en la nube como AWS.

8.6. O6. Desarrollar una Guía Práctica

Elaborar una guía práctica para la adopción de servicios de AWS y DevOps fue uno de los objetivos finales del proyecto. Esta guía detalla los pasos y mejores prácticas para una implementación exitosa, basada en las experiencias y lecciones aprendidas durante el proyecto.

Logros:

- Creación de una guía detallada que incluye configuraciones, mejores prácticas y estrategias para la implementación de DevOps y AWS.
- Provisión de un recurso valioso para otras empresas o grupos de desarrollo que deseen adoptar estas tecnologías.
- Inclusión de ejemplos prácticos y casos de uso basados en el proyecto.

Este objetivo se ha alcanzado de manera integral, proporcionando un recurso práctico y útil para la adopción de DevOps y AWS.

Capítulo 9

Conclusiones

La realización de este Trabajo de fin de Máster ha constituido una experiencia enriquecedora que me ha permitido profundizar en parte del mundo de la tecnología en la nube. A través de la investigación y el análisis realizados, se ha logrado entender y aplicar conceptos avanzados de DevOps y los servicios de AWS, los cuales ofrecen una amplia gama de ventajas para la creación de plataformas eficientes y escalables.

El proyecto se inició con un análisis meticuloso de los requisitos del Caso de Estudio, identificando las necesidades específicas y las tecnologías que ayudarían en la implementación. En este punto también se analizaron varios servicios para dar soporte a los requisitos técnicos y operativos, por lo que se puede decir que este análisis inicial sirvió como piedra angular, definiendo el marco de trabajo y estableciendo una base sobre la cual desarrollar el proyecto.

En segundo lugar, se ha comprendido el marco de DevOps mediante la exploración de sus principios, prácticas y herramientas. Este entendimiento permitió apreciar la aplicabilidad de DevOps para el desarrollo de la plataforma, destacando su importancia en la integración continua y la entrega continua (CI/CD). Se formularon estrategias DevOps específicas para garantizar el desarrollo, despliegue y mantenimiento continuos de la plataforma. Estas estrategias no solo facilitaron la automatización de procesos, sino que también presentan una manera en la que mejorar la eficiencia y la confiabilidad operativa.

El resultado tangible de este proyecto ha sido el diseño e implementación de la arquitectura necesaria para el desarrollo y la capacidad operativa de una plataforma de comercio electrónico. Esto incluye la configuración de la infraestructura en la nube, la implementación de pipelines de CI/CD siguiendo técnicas DevOps, y la utilización de servicios de AWS para asegurar una operación eficiente de la plataforma: se ha establecido una infraestructura robusta que permite la gestión automatizada del código fuente, integración, despligue y monitoreo continuo de la plataforma. Además, se han implementado estrategias de alta disponibilidad, escalabilidad y recuperación de desastres, garantizando que la plataforma pueda soportar un alto volumen de tráfico y mantener la operatividad incluso en situaciones adversas.

Por último, cabe mencionar que el análisis de los resultados obtenidos de la implementación de DevOps y AWS permitió identificar tanto los beneficios logrados como los desafíos enfrentados durante el proceso. Sin embargo, también se identificaron

desafíos, como la necesidad de una curva de aprendizaje inicial y la gestión de costos asociados con los servicios en la nube.

Considero que la experiencia adquirida a lo largo de este proyecto me ha preparado para enfrentar futuros desafíos en el ámbito del desarrollo y la operación de plataformas tecnológicas, consolidando mis conocimientos y habilidades en un campo en constante evolución en el que las empresas y grupos de desarrollo deberían poner más énfasis para mejorar sus resultados.

Bibliografía

[1] Altexsoft. (2023) Devops principles, practices, and devops engineer role. [Online]. Available: https://www.altexsoft.com/blog/devops-principles-practices-and-devops-engineer-role/

[2] NetApp, "¿qué es devops?" *NetApp*, 2024. [Online]. Available: https://www.netapp.com/es/devops-solutions/what-is-devops/

[3] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[4] A. W. Services. (2024) What is iaas? [Online]. Available: https://aws.amazon.com/what-is-iaas/

[5] G. C. Platform. (2024) App engine documentation. [Online]. Available: https://cloud.google.com/learn/what-is-paas?hl=es

[6] FYCCorp. (2024) Xaas: Anything as a service (everything as a service) [spanish]. [Online]. Available: https://www.fyccorp.com/articulo-xaas:-anything-as-a-service-o-todo-como-un-servicio

[7] A. W. Services, "What is aws?" https://aws.amazon.com/es/what-is-aws/, 2024.

[8] A. XU. (2023) Ep74: The evolution of aws services. https://blog.bytebytego.com/p/ep74-the-evolution-of-aws-services.

[9] A. W. Services. (2024) Regions, availability zones, and local zones. [Online]. Available: https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Concepts.RegionsAndAvailabilityZones.html

[10] Technource. (2023) Why apps like netflix use node.js [blog post]. [Online]. Available: https://www.technource.com/blog/why-app-like-netflix-uses-nodejs/#:~:text=High%20Performance%30.

[11] A. W. Services. (2024) Amazon web services codepipeline. [Online]. Available: https://aws.amazon.com/es/codepipeline/

[12] ——. (2024) Amazon codecatalyst: Servicio de desarrollo unificado. [Online]. Available: https://aws.amazon.com/es/codecatalyst/

[13] IBM. (2024) ¿qué es el despliegue continuo? [Online]. Available: https://www.ibm.com/es-es/topics/continuous-deployment

[14] A. W. Services. (2024) Aws codedeploy: Automated code deployment. [Online]. Available: https://aws.amazon.com/codedeploy/

[15] ——. (2024) Understanding availability. [Online]. Available: https://docs.aws.amazon.com/pdfs/whitepapers/latest/availability-and-beyond-improving-resilience/availability-and-beyond-improving-resilience.pdf#understanding-availability

[16] I. Docker. (2024) Docker. [Online]. Available: https://www.docker.com

[17] A. W. Services. (2024) Amazon elastic container registry (ecr). [Online]. Available: https://aws.amazon.com/ecr/?nc1=h_ls

[18] ——. (2024) Amazon elastic container service (ecs). [Online]. Available: https://aws.amazon.com/es/ecs/

[19] ——. (2024) Application load balancers (elbs) - guide. [Online]. Available: https://docs.aws.amazon.com/pdfs/elasticloadbalancing/latest/application/elb-ag.pdf

[20] Kodigo, "Escalabilidad en proyectos de software: ¿por qué es crucial para el éxito a largo plazo?" *Kodigo*, 2024. [Online]. Available: https://kodigo.org/escalabilidad-en-proyectos-de-software-por-que-es-crucial-para-el-exito-a-largo-plazo/

[21] A. W. Services. (2024) Amazon elastic auto scaling. [Online]. Available: https://aws.amazon.com/es/autoscaling/

[22] ——. (2024) Amazon cloudwatch. [Online]. Available: https://aws.amazon.com/es/cloudwatch/

[23] ——. (2024) Amazon dynamodb. [Online]. Available: https://aws.amazon.com/es/dynamodb/

[24] ——. (2024) Amazon s3: Servicio de almacenamiento de objetos escalable. [Online]. Available: https://aws.amazon.com/es/s3/

[25] ——. (2024) Amazon backup: Servicio centralizado de copia de seguridad en la nube. [Online]. Available: https://aws.amazon.com/es/backup/

[26] ——, "Aws cli user guide," 2024. [Online]. Available: https://docs.aws.amazon.com/cli/latest/userguide/

[27] ——. (2024) Welcome to aws codepipeline. [Online]. Available: https://docs.aws.amazon.com/codepipeline/latest/userguide/welcome.html

[28] R. B. Rahul Saini, "An introduction to aws ec2 elastic compute cloud," https://annals-csis.org/Volume_24/drp/pdf/4.pdf, 2020, aWS Resource.

[29] A. W. Services. (2024) Installing, updating, and verifying the codedeploy agent. [Online]. Available: https://docs.aws.amazon.com/codedeploy/latest/userguide/codedeploy-agent-operations-install.html

[30] ——. Create a build project (aws cli). [Online]. Available: https://docs.aws.amazon.com/codebuild/latest/userguide/create-project-cli.html

[31] SonarSource. (2024) Set up sonarcloud with github. [Online]. Available: https://docs.sonarsource.com/sonarcloud/getting-started/github/

[32] ——. (2024) Sonarscanner cli and sonarcloud. [Online]. Available: https://docs.sonarsource.com/sonarcloud/advanced-setup/ci-based-analysis/sonarscanner-cli/

[33] A. W. Services. (2024) Aws secrets manager. [Online]. Available: https://aws.amazon.com/es/secrets-manager/

[34] ——. (2024) Creating a blue/green deployment with amazon ecs. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/create-blue-green.html

[35] ——. (2024) Creating an ecs deployment pipeline with codepipeline. [Online]. Available: https://docs.aws.amazon.com/codepipeline/latest/userguide/ecs-cd-pipeline.html

[36] ——. (2024) Tutorial: Deploying an amazon ecs application with codepipeline, amazon ecr, and aws codedeploy. [Online]. Available: https://docs.aws.amazon.com/codepipeline/latest/userguide/tutorials-ecs-ecr-codedeploy.html

[37] ——. (2024) Task definitions. [Online]. Available: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task_definitions.html

[38] ——. (2024) Application load balancing. Amazon Elastic Load Balancing Application Layer Guide. [Online]. Available: https://docs.aws.amazon.com/elasticloadbalancing/latest/application/introduction.html

[39] ——. (2024) Table classes. Amazon DynamoDB Developer Guide. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/WorkingWithTables.tableclasses.html

[40] ——. (2024) Getting started with amazon dynamodb. [Online]. Available: https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/GettingStarted.html

[41] ——. (2024) Restoring a dynamodb table using aws backup. [Online]. Available: https://docs.aws.amazon.com/aws-backup/latest/devguide/restoring-dynamodb.html

[42] ——. (2024) Restoring s3 objects from backups. [Online]. Available: https://docs.aws.amazon.com/aws-backup/latest/devguide/restoring-s3.html

[43] ——. (2024) Aws fargate para amazon ecs. [Online]. Available: https://docs.aws.amazon.com/es_es/AmazonECS/latest/developerguide/AWS_Fargate.html

[44] “Aws free tier,” 2024. [Online]. Available: https://aws.amazon.com/free/

[45] “Aws calculator,” 2024. [Online]. Available: https://calculator.aws/

[46] A. W. Services. (2024) Emea value added tax (vat) tables. [Online]. Available: https://aws.amazon.com/es/tax-help/emea-vat-tables/