AUTOMATIZACIÓN DEL PROCESO DE DESPLIEGUE DE INFRAESTRUCTURA TECNOLÓGICA, PRUEBAS UNITARIAS Y PRUEBAS DE USUARIO EN ARQUITECTURAS DE RED VIRTUALIZADAS EN EL DEPARTAMENTO DE TECNOLOGÍA DE LA UNIVERSIDAD EL BOSQUE

SERGIO WILBERTH GONZÁLEZ ROZO
SANTIAGO MEJÍA RAMOS

REALIZADO CON LA ASESORÍA DE:

JAIRO HERNÁN GARCÍA TRIANA

JUAN PEDRO MENDOZA RAMÍREZ

UNIVERSIDAD EL BOSQUE
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
SEPTIEMBRE, 2023

UNIVERSIDAD EL BOSQUE
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA

ÁREA DE ÉNFASIS: TELECOMUNICACIONES

AUTOMATIZACIÓN DEL PROCESO DE DESPLIEGUE DE INFRAESTRUCTURA TECNOLÓGICA, PRUEBAS UNITARIAS Y PRUEBAS DE USUARIO EN ARQUITECTURAS DE RED VIRTUALIZADAS EN EL DEPARTAMENTO DE TECNOLOGÍA DE LA UNIVERSIDAD EL BOSQUE

SERGIO WILBERTH GONZÁLEZ ROZO

SANTIAGO MEJÍA RAMOS

REALIZADO CON LA ASESORÍA DE:

JAIRO HERNÁN GARCÍA TRIANA

JUAN PEDRO MENDOZA RAMÍREZ

Página de Aprobación. Inclusión de Acta de grado.

NOTA DE SALVEDAD

Según el artículo 37 del 14 de diciembre de 1989 del acuerdo 017, “La Universidad El Bosque, no se hace responsable de los conceptos emitidos por los investigadores en su trabajo, solo velará por el rigor científico, metodológico y ético del mismo en aras de la búsqueda de la verdad y la justicia”.

DEDICATORIA

Esta obra está dedicada a nosotros, los autores, como testimonio de nuestra perseverancia y dedicación. A través de este proyecto, celebramos nuestra capacidad para superar los desafíos y encontrar el equilibrio entre la vida universitaria, el mundo laboral y nuestras vidas personales. Cada página escrita es un reflejo de nuestro esfuerzo conjunto, de las noches largas de estudio, de los sacrificios y de la pasión que hemos invertido en esta travesía. Esta dedicación es un tributo a nuestra determinación, amistad y valentía. Que esta obra sirva como un recordatorio eterno de nuestra capacidad para alcanzar nuevas alturas cuando trabajamos juntos y nos apoyamos mutuamente en la búsqueda del conocimiento y el crecimiento personal.

AGRADECIMIENTOS

Queremos expresar nuestra profunda gratitud a nuestras familias, cuyo apoyo incondicional nos ha guiado durante los desafiantes cinco años de nuestra carrera universitaria y nuestro camino para ser ingenieros electrónicos. A nuestros amigos y compañeros, en particular a Julio César Hernández, quien ha sido parte integral de nuestro equipo a lo largo de toda la carrera, les agradecemos por su amistad y colaboración inestimables. También extendemos nuestro reconocimiento a nuestro director de proyecto de grado, Jairo García, por su orientación experta y apoyo constante. Por último, a todos aquellos profesores de la facultad que tienen dedicación y vocación para enseñar, sin su guía, sabiduría y pasión por la enseñanza, este logro no habría sido posible.

RESUMEN

Hoy en día, las organizaciones deben enfrentarse a un nuevo entorno económico que incluye un mundo más conectado, tecnologías ampliamente accesibles y negocios disruptivos que están redefiniendo la forma en la que los consumidores interactúan con las marcas a través de experiencias digitales innovadoras. Para lograrlo, las empresas deben empezar una transformación digital, en la que el software sea un componente clave. Por tanto, la capacidad de crear y desarrollar software cuyos criterios sean la velocidad, la eficiencia y la calidad es indispensable para la supervivencia de la empresa. La cultura de DevOps, especialmente en términos de entrega de software continua, da un lineamiento para las prácticas que deben incluirse en el proceso de desarrollo de aplicaciones de una empresa para crear y mantener vigente el software de acuerdo con los requisitos anteriores. Este proyecto se centra en las prácticas y los procesos del ciclo de construcción, prueba, e implementación en la dirección de tecnología de la Universidad El Bosque. Aborda conceptos, definiciones y prácticas, y examina el papel de la configuración, la integración continua, la automatización de pruebas y los procesos de automatización de implementación. También cubre las herramientas necesarias para desarrollar, simplificar, automatizar y gestionar cada práctica de proceso. El resultado principal del proyecto fue lograr la automatización exitosa de estos procesos, lo que resultó en una mejora de los tiempos para finalizar proyectos relacionados con el desarrollo de software. La automatización redujo errores humanos, aumentó la eficiencia y permitió a los desarrolladores ser más proactivos al evitar interrupciones innecesarias. Además, se logró estandarizar los entornos de desarrollo, pruebas y producción para evitar problemas en entornos no homogéneos, de la misma manera mediante la implementación del proyecto se logró permitir a la dirección de Tecnología enfocarse en tareas que no sean repetitivas y mejorar la productividad.

Palabras Clave: Automatización, Cambio tecnológico, Diseño de sistemas, Análisis de redes, Infraestructura tecnológica.

ABSTRACT

Nowadays, organizations must face a new economic environment that includes a more connected world, widely accessible technologies and disruptive businesses that are redefining the way consumers interact with brands through innovative digital experiences. To achieve this, companies must begin a digital transformation, with software as a key component. Therefore, the ability to create and develop software whose criteria are speed, efficiency and quality is indispensable for the survival of the company. The DevOps culture, especially in terms of continuous software delivery, provides a guideline for practices that should be included in a company's application development process to create and maintain current software in accordance with the above requirements. This project focuses on the practices and processes of the build, test, and deployment cycle in the technology department of Universidad El Bosque. It addresses concepts, definitions, and practices, and examines the role of configuration, continuous integration, test automation, and deployment automation processes. It also covers the tools needed to develop, simplify, automate and manage each process practice. The main outcome of the project was to achieve successful automation of these processes, resulting in improved time to completion for software development related projects. Automation reduced human errors, increased efficiency and allowed developers to be more proactive by avoiding unnecessary interruptions. In addition, the development, testing and production environments were standardized to avoid problems in non-homogeneous environments, and the implementation of the project allowed the technology management to focus on non-repetitive tasks and improve productivity.

Keywords: Automation, Technological change, System design, Network analysis, Technological infrastructure.

GLOSARIO DE TÉRMINOS

Alertas del Sistema: Notificaciones automáticas que indican problemas o eventos importantes en el sistema, lo que permite una respuesta rápida. (definición tomada de Glosario Gartner IT) [1]

Amazon CloudFormation: Servicio de AWS que permite definir y desplegar recursos de infraestructura como código en AWS. (definición tomada de Glosario AWS) [2]

Amazon Elastic Container Service (ECS): Servicio de administración de contenedores de AWS que permite ejecutar y escalar contenedores Docker en la nube. (definición tomada de Glosario AWS) [2]

Amazon Fargate: Servicio de AWS que permite ejecutar contenedores Docker sin necesidad de administrar servidores subyacentes. (definición tomada de Glosario AWS) [2]

Amazon S3 (Amazon Simple Storage Service): Servicio de almacenamiento en la nube de AWS que proporciona una forma escalable y segura de almacenar y recuperar datos. (definición tomada de Glosario AWS) [2]

API Gateway: Servicio que permite crear, publicar, mantener y asegurar APIs (Interfaces de Programación de Aplicaciones) para aplicaciones y servicios en la nube. (definición tomada de Glosario Gartner IT) [1]

Artefacto: Elemento o componente generado durante el proceso de desarrollo de software, como un archivo compilado o una imagen de contenedor, que se utiliza en el despliegue y la implementación. (definición tomada de Glosario Gartner IT) [1]

Automatización de Pruebas: Proceso de automatizar la ejecución de pruebas de software para verificar la funcionalidad y calidad del código de manera eficiente. (definición tomada de Glosario Gartner IT) [1]

AWS CodeCommit: Servicio de AWS que proporciona un sistema de control de versiones totalmente administrado para almacenar y gestionar el código fuente de las aplicaciones. (definición tomada de Glosario AWS) [2]

AWS CodePipeline: Porción de un flujo de trabajo de canalización en la que se realizan una o más acciones. (definición tomada de Glosario AWS) [2]

AWS CODE Account: Cuenta de AWS utilizada para almacenar el código fuente de la infraestructura, almacenar el código fuente de las aplicaciones, realizar el compilado de los

artefactos en cada uno de los ambientes correspondientes. (definición tomada de Glosario AWS) [2]

AWS Dev Account: Cuenta de AWS utilizada específicamente para el desarrollo y pruebas de aplicaciones antes de su implementación en entornos de producción(definición tomada de Glosario AWS) [2]

AWS Production Account: Cuenta de AWS utilizada para implementar y alojar aplicaciones y servicios en entornos de producción. (definición tomada de Glosario AWS) [2]

AWS QA Account: Cuenta de AWS utilizada para realizar pruebas de calidad y aseguramiento de calidad de aplicaciones antes de su implementación en entornos de producción. (definición tomada de Glosario AWS) [2]

Backend: La parte de una aplicación que se encarga del procesamiento y almacenamiento de datos, generalmente no visible para el usuario final. (definición tomada de Glosario Gartner IT) [1]

Bucket de S3: Contenedor de nivel superior en Amazon S3 que almacena objetos, como archivos y datos, y se utiliza para organizar y administrar los recursos en S3. (definición tomada de Glosario AWS) [2]

CLI (Interfaz de Línea de Comando): Método de interactuar con una computadora o programa mediante comandos de texto escritos en lugar de una interfaz gráfica de usuario (GUI). (definición tomada de Glosario Gartner IT) [1]

Cluster de ECS: En Amazon Elastic Container Service (ECS), un cluster es un grupo de contenedores que se ejecutan en instancias de EC2 y que están administrados por el servicio ECS. (definición tomada de Glosario AWS) [2]

CodeBuild: Servicio de integración continua totalmente gestionado que compila el código fuente, ejecuta pruebas y produce paquetes de software listos para su implementación. (definición tomada de Glosario AWS) [2]

CodeDeploy: Servicio de AWS encargado de desplegar código. (definición tomada de Glosario AWS) [2]

Control de Versiones: Sistema que rastrea los cambios realizados en el código fuente o en archivos y permite la colaboración entre diferentes desarrolladores al mantener un historial de versiones. (definición tomada de Glosario Gartner IT) [1]

Cuenta de AWS: Entorno de Amazon Web Services (AWS) para acceder y gestionar recursos en la nube. (definición tomada de Glosario AWS) [2]

DevOps: Filosofía y conjunto de prácticas que enfatizan la colaboración y comunicación entre los equipos de desarrollo de software (Dev) y operaciones de TI (Ops) para acelerar la entrega de software. (definición tomada de Glosario Gartner IT) [1]

Ejecución de Pruebas de Usuario: Proceso de ejecución de pruebas diseñadas para evaluar la funcionalidad de una aplicación desde la perspectiva de un usuario final. (definición tomada de Glosario Gartner IT) [1]

Ejecución de Pruebas Unitarias: Proceso de ejecución de pruebas diseñadas para verificar la funcionalidad de componentes de software individuales o unidades. (definición tomada de Glosario Gartner IT) [1]

ELB (Elastic Load Balancer): Servicio de AWS que distribuye el tráfico de red entre recursos para mejorar la disponibilidad. (definición tomada de Glosario AWS) [2]

ELB Listener: Se refiere a una configuración en Elastic Load Balancer (ELB) de AWS que especifica cómo se deben dirigir las solicitudes de tráfico entrante a los destinos apropiados, como instancias de Amazon EC2. (definición tomada de Glosario AWS) [2]

Entrega Continua de Software: Práctica que implica la automatización de la construcción, pruebas y despliegue de software para permitir entregas rápidas y consistentes de nuevas funcionalidades. (definición tomada de Glosario Gartner IT) [1]

Elastic Container Registry (ECR): Servicio de AWS que proporciona almacenamiento y administración de imágenes de contenedores Docker. (definición tomada de Glosario AWS) [2]

Frontend: La parte de una aplicación que interactúa directamente con el usuario final y presenta la interfaz de usuario. (definición tomada de Glosario Gartner IT) [1]

Gestión de Configuración: Proceso de controlar y gestionar los cambios en los componentes de software y hardware para mantener la integridad y la trazabilidad. (definición tomada de Glosario Gartner IT) [1]

Git: Sistema de control de versiones. (definición tomada de Glosario Gartner IT) [1]

Git Push: Comando para enviar cambios locales a un repositorio remoto. (definición tomada de Glosario Gartner IT) [1]

Git Merge: Operación para combinar ramas en Git. (definición tomada de Glosario Gartner IT) [1]

IaC (Infraestructura como Código): Enfoque que trata la infraestructura de TI como código, permitiendo la automatización y gestión de recursos mediante scripts y configuraciones. (definición tomada de Glosario Gartner IT) [1]

IaaS (Infraestructura como Servicio): Modelo de entrega de servicios de infraestructura de TI, como servidores virtuales, almacenamiento y redes, a través de la nube. (definición tomada de Glosario Gartner IT) [1]

IAM: Siglas de "Identity and Access Management", se refiere a la gestión de identidades y control de acceso, permitiendo administrar quién tiene permiso para acceder a recursos y servicios específicos. (definición tomada de Glosario Gartner IT) [1]

IDE (Integrated Development Environment): Entorno de desarrollo integrado que proporciona herramientas y funciones para programar y desarrollar software de manera eficiente. (definición tomada de Glosario Gartner IT) [1]

Infraestructura Tecnológica: Conjunto de componentes físicos y virtuales, como servidores, redes y almacenamiento, necesarios para respaldar las operaciones de TI de una organización. (definición tomada de Glosario Gartner IT) [1]

Integración Continua: Práctica que implica la integración constante de cambios de código en un repositorio compartido, seguida de pruebas automáticas para mantener la calidad del software. (definición tomada de Glosario Gartner IT) [1]

Manual de Usuario: Documento que proporciona instrucciones detalladas sobre cómo utilizar una aplicación, incluyendo pasos para realizar diferentes tareas. (definición tomada de Glosario Gartner IT) [1]

Métricas de Desempeño: Indicadores cuantitativos utilizados para evaluar el rendimiento y la eficiencia de los procesos de entrega continua de software. (definición tomada de Glosario Gartner IT) [1]

Microservicio: Pequeña unidad de software independiente que realiza una función específica. (definición tomada de Glosario Gartner IT) [1]

PaaS (Plataforma como Servicio): Modelo de entrega de una plataforma de desarrollo y despliegue de aplicaciones, incluyendo herramientas y servicios, a través de la nube. (definición tomada de Glosario Gartner IT) [1]

Parameter Store: Servicio de gestión de configuración que almacena y recupera datos de configuración y parámetros de forma segura para su uso en aplicaciones y sistemas. (definición tomada de Glosario Gartner IT) [1]

Pipeline: Conjunto automatizado de pasos para la construcción, prueba y despliegue continuo de software. (definición tomada de Glosario AWS) [2]

Pipeline de Despliegue: Conjunto de pasos y acciones automatizadas que permiten el despliegue continuo de una aplicación en diferentes entornos. (definición tomada de Glosario AWS) [2]

Pruebas de Calidad: Evaluación sistemática de un software para asegurar que cumple con los estándares de calidad, detectar errores y garantizar su funcionamiento correcto. (definición tomada de Glosario Gartner IT) [1]

Rama de Repositorio: Línea de desarrollo independiente en un sistema de control de versiones como Git. (definición tomada de Glosario Gartner IT) [1]

Región de AWS: Área geográfica con centros de datos de Amazon Web Services (AWS) que ofrecen servicios en la nube. (definición tomada de Glosario AWS) [2]

Repositorio de Infraestructura: Almacenamiento de código y configuración que permite versionar y gestionar la infraestructura como código (IaC) utilizada para implementar y configurar aplicaciones. (definición tomada de Glosario Gartner IT) [1]

SaaS (Software como Servicio): Modelo de distribución de software en el que las aplicaciones se alojan en la nube y se proporcionan a los usuarios a través de Internet. (definición tomada de Glosario Gartner IT) [1]

Security Group de AWS: Es un conjunto de reglas de firewall que controla el tráfico de red entrante y saliente para las instancias de Amazon EC2 en una VPC de AWS. (definición tomada de Glosario AWS) [2]

SDN: Siglas de "Software-Defined Networking," se refiere a una tecnología que permite gestionar y controlar de manera centralizada las redes de comunicación, separando el plano de control del plano de datos. (definición tomada de Glosario Gartner IT) [1]

SDLC: Siglas de "Software Development Life Cycle" (Ciclo de Vida del Desarrollo de Software), se refiere a un conjunto de procesos y fases utilizados para planificar, diseñar, desarrollar, probar y mantener un software a lo largo de su vida útil. (definición tomada de Glosario Gartner IT) [1]

Stack de CloudFormation: Conjunto de recursos y configuraciones definidos en un archivo de plantilla de AWS CloudFormation que se crean y gestionan como una sola unidad. (definición tomada de Glosario AWS) [2]

Stackset de CloudFormation: Conjunto de stacks de AWS CloudFormation que se implementan en varias cuentas y regiones de AWS de manera coordinada y centralizada. (definición tomada de Glosario AWS) [2]

Subnet de AWS: Es una subdivisión de una VPC en la nube de AWS. Cada subnet puede estar asociada a una zona de disponibilidad y contiene un rango de direcciones IP de la VPC. (definición tomada de Glosario AWS) [2]

Task Definition: En Amazon Elastic Container Service (ECS), una tarea (Task) es una instancia de ejecución de un contenedor dentro de un cluster. La definición de tarea (Task Definition) especifica los parámetros y configuraciones para ejecutar una o más instancias del mismo contenedor en un cluster de ECS. (definición tomada de Glosario Gartner IT) [1]

TestNG: Herramienta de automatización de pruebas de software utilizada para realizar pruebas unitarias y de usuario en aplicaciones. (definición tomada de Glosario Gartner IT) [1]

Time to Market: El tiempo que lleva desde el inicio del desarrollo de un producto hasta su lanzamiento en el mercado. (definición tomada de Glosario Gartner IT) [1]

Transformación Digital: Proceso de integración de tecnologías digitales en todos los aspectos de una organización para mejorar la forma en que opera y ofrece valor a sus clientes. (definición tomada de Glosario Gartner IT) [1]

Uptime: Tiempo durante el cual un sistema o servicio está operativo y disponible para su uso. (definición tomada de Glosario Gartner IT) [1]

Virtualización: Tecnología que permite crear múltiples instancias virtuales de hardware o software en una sola máquina física. (definición tomada de Glosario Gartner IT) [1]

VPC: Siglas de "Virtual Private Cloud," se refiere a una red virtual aislada en la nube de AWS donde los usuarios pueden lanzar recursos de AWS, como instancias de EC2 y bases de datos RDS. (definición tomada de Glosario AWS) [2]

Zona de Disponibilidad: Localización física dentro de una región de nube que contiene uno o más centros de datos independientes con energía y redes redundantes para garantizar la alta disponibilidad de los servicios en la nube. (definición tomada de Glosario Gartner IT) [1]

LISTA DE ABREVIATURAS

AWS	Amazon Web Services
ECS	Elastic Container Service
ECR	Elastic Container Registry
ELB	Elastic Load Balancer
CI	Continuous Integration
CD	Continuous Deployment
QA	Quality Assurance
CLI	Command Line Interface
IDE |	Integrated Development Environment (Entorno de desarrollo integrado)
IaC	Infraestructura como Código
IaaS	Infraestructura como Servicio
PaaS	Plataforma como Servicio
SaaS	Software como Servicio
DevOps	Desarrollo y Operaciones
YAML	YAML Ain't Markup Language (Lenguaje YAML)
API	Interfaz de Programación de Aplicaciones
VPC	Virtual Private Cloud (Nube Privada Virtual)
DNS	Domain Name System (Sistema de Nombres de Dominio)
SQL	Structured Query Language (Lenguaje de Consulta Estructurado)
JSON	JavaScript Object Notation (Notación de Objetos de JavaScript)
REST	Representational State Transfer (Transferencia de Estado Representacional)
TI	Tecnologías de la Información
URL	Uniform Resource Locator (Localizador de Recursos Uniforme)
SSL	Secure Sockets Layer (Capa de Conexiones Seguras)
SSH	Secure Shell (Shell Seguro)
SDK	Software Development Kit (Kit de Desarrollo de Software)
VCS	Version Control System (Sistema de Control de Versiones)
EC2	Elastic Compute Cloud (Máquina Virtual de AWS)
RDS	Relational Database Service (Servicio de Bases de Datos Relacionales)
S3	Simple Storage Service (Servicio de Almacenamiento Simple)

TABLA DE CONTENIDOS

Pág.

LISTA DE TABLAS

Tabla Pág.

LISTA DE FIGURAS

Figura Pág.

1. INTRODUCCIÓN

Posiblemente la construcción de este documento se realizó utilizando una aplicación tecnológica, pudo haber sido descargado o almacenado en algún tipo de plataforma que hoy día es de común uso, como lo son los gestores de almacenamiento o aplicaciones de texto. De la misma manera funcionan los procesos tecnológicos en los que está hoy día soportada la industria, una persona que quiera estudiar una nueva carrera universitaria, seguramente ingresará a una página web donde solicitará que lo contacten o activará un proceso en una institución que finalmente desencadenara en el ingreso de esta persona a la vida académica. Detrás de estas aplicaciones o plataformas, existe una infraestructura tecnológica que los soporta, así como un conjunto de personas que crean el código fuente de la aplicación, prueban las aplicaciones, y crean los recursos tecnológicos necesarios para poder acceder a esta aplicación desde un navegador web o desde algún otro medio. Todas las aplicaciones tecnológicas presentes en la red de internet basan su funcionamiento en el concepto de telecomunicaciones, acceder a cierto recurso desde una ubicación diferente a donde se encuentra el recurso mediante la utilización de redes de datos y de infraestructura tecnológica.

El proceso de creación de aplicaciones en la Universidad El Bosque hoy día es dependiente de la intervención del personal de arquitectura tecnológica de la universidad, así como del personal que realiza las pruebas a las aplicaciones. Mediante la automatización de procesos, las dependencias de intervenciones manuales se pueden reducir y hasta eliminar, a lo largo del documento se describe cómo se optimizó el procedimiento de creación de recursos de infraestructura, se automatizó la técnica de realización de pruebas y se realizó la entrega de un proceso que puede ser replicado, el cual requiere de una intervención mínima en la dirección de tecnología de la Universidad El Bosque. El proyecto basó su creación y funcionamiento en el conocimiento recibido en la carrera de ingeniería electrónica, en áreas como programación, implementación de buenas prácticas en seguridad (versionado y DRP), telecomunicaciones, teleinformática y redes de datos.

Mediante la utilización de plantillas creadas de infraestructura como código, la puesta en marcha de un microservicio se minimizó a la ejecución de un comando en la línea de código, proceso que anteriormente se realizaba mediante la intervención del personal de tecnología y requería de un manual de procedimiento. Estas mismas plantillas realizan una creación homogénea de los ambientes en los cuales se almacena la infraestructura de un microservicio. De igual manera las plantillas incluyen la infraestructura y el código necesario para automatizar la realización de pruebas y el despliegue de las aplicaciones a la infraestructura tecnológica mediante la utilización de CloudFormation, una herramienta de Amazon Web Services que permite la creación de infraestructura basado en una plantilla, proveedor de infraestructura de nube que utiliza la universidad. De esta manera, se da una mejora en los tiempos de puesta en marcha de nuevas aplicaciones o de nuevas características en las aplicaciones ya existentes, así como, se disminuye la probabilidad de inserción de errores humanos y generación de fallas en la infraestructura tecnológica que soporta las aplicaciones de cara a los usuarios.

2. PLANTEAMIENTO DEL PROBLEMA

2.1 Antecedentes y Estado del Arte

2.1.1 Nivel Internacional

A lo largo de los últimos años, empresas de todo el mundo han decidido invertir en la automatización de procesos, acoplándose a las prácticas de automatización eficiente. La automatización de la creación y despliegue de infraestructura tecnológica revela un panorama en constante evolución y crecimiento. En un mundo cada vez más digital y competitivo, las organizaciones de todo el mundo están buscando formas de agilizar y optimizar sus procesos de desarrollo de software, despliegue de infraestructura y pruebas para mejorar la eficiencia y la calidad de sus productos y servicios.

En este contexto, se ha observado un crecimiento significativo en la adopción de prácticas de DevOps (Desarrollo y Operaciones) a nivel internacional. DevOps se ha convertido en un enfoque fundamental para la automatización de procesos, la colaboración entre equipos de desarrollo y operaciones, y la entrega continua de software. Empresas líderes a nivel global, como Amazon, Google y Netflix, han demostrado el éxito de la implementación de DevOps en la optimización de sus operaciones y la reducción de los tiempos de desarrollo.[3]

Además, según [4] se han desarrollado numerosas herramientas y plataformas en todo el mundo para facilitar la automatización de procesos, la gestión de infraestructura en la nube y las pruebas automatizadas. Estas herramientas abarcan desde soluciones de administración de la nube, como AWS, Azure y Google Cloud, hasta plataformas de automatización de pruebas como Selenium y Jenkins.

En cuanto a la virtualización y las arquitecturas de red, tecnologías como la virtualización de servidores y contenedores (por ejemplo, Docker) han revolucionado la forma en que se despliegan y administran los recursos de infraestructura. Las arquitecturas de red definidas por software (SDN) también han ganado relevancia, permitiendo la gestión y el despliegue flexibles de redes virtuales. [5] A nivel

internacional, la automatización de procesos, la entrega continua de software y la optimización de la infraestructura tecnológica son tendencias dominantes en el desarrollo de software y las operaciones de TI. La adopción de estas prácticas y tecnologías está en constante crecimiento, lo que subraya la importancia y la relevancia de proyectos como el que se propone en este trabajo de grado.

2.1.2 Nivel Nacional

En Colombia, se está reconociendo la importancia de la automatización de procesos, la entrega continua de software y la virtualización de recursos tecnológicos como elementos clave para mejorar la competitividad y la calidad en el desarrollo de software. La automatización de estos procesos está experimentando un crecimiento significativo, reflejando la necesidad de las organizaciones locales de mejorar la eficiencia en el desarrollo de software y la gestión de recursos tecnológicos. [6]

En Colombia, la adopción de prácticas de DevOps (Desarrollo y Operaciones) ha ido en aumento en diversas industrias, incluyendo el sector financiero, de telecomunicaciones y de salud. Empresas colombianas líderes han comenzado a implementar DevOps para automatizar la entrega de software y reducir los tiempos de desarrollo, lo que ha resultado en mejoras en la calidad de los productos y una mayor competitividad en el mercado. [7]

En el ámbito de la virtualización y la infraestructura en la nube y según los datos entregados, Colombia ha experimentado un crecimiento constante en la adopción de servicios de nube como Amazon Web Services (AWS), Microsoft Azure y Google Cloud Platform (GCP). [8] Esto ha impulsado la necesidad de herramientas y soluciones que permitan la gestión eficiente de la infraestructura en la nube y la virtualización de recursos.

En cuanto a la automatización de pruebas, se ha observado un interés creciente en Colombia por la implementación de pruebas automatizadas en el proceso de desarrollo de software. Esto se debe a la búsqueda de reducir los errores y garantizar

la calidad de las aplicaciones. Se han llevado a cabo iniciativas de capacitación y formación en el país para fortalecer las habilidades en este ámbito [6].

2.2 Descripción y Formulación del Problema

Los tiempos de despliegue y puesta en marcha de nuevas características en las aplicaciones tecnológicas son dependientes de la disponibilidad del recurso humano involucrado en el proceso, si no hay personal disponible para realizar pruebas (analista de pruebas), la puesta en marcha se retrasa, si no hay personal para realizar el despliegue (arquitecto de infraestructura), la puesta en marcha se retrasa. Al ser un proceso dependiente de la intervención humana, es susceptible a la inserción de fallas que no se ven reflejadas inmediatamente y pueden derivar en una mala imagen para el usuario final, así como la presencia de errores y fallas en la infraestructura que soporta la aplicación. La ralentización en el proceso de despliegue de aplicaciones y puesta en marcha de nuevas características puede afectar de manera directa la capacidad de la universidad para reaccionar a los cambios del mercado, de igual manera limita la posibilidad de la universidad para mantenerse a la vanguardia en relación a los procesos tecnológicos que hoy en día son la base de la mayoría de industrias.

2.3 Justificación

La creación, despliegue y pruebas de infraestructura tecnológica es delegada a una o varias personas dedicadas solo a esta tarea en la dirección de tecnología de la Universidad El Bosque. Al ser un proceso manual y repetitivo, implica un alto riesgo de inserción de errores humanos, adicionalmente el proceso ágil de desarrollo se puede ver detenido a la espera de disponibilidad del recurso humano para la realización del despliegue. Al automatizar los procesos de despliegue y prueba, la intervención humana en el proceso se ve disminuida notablemente en comparación a los procesos manuales, adicionalmente los tiempos totales de desarrollo de software se ven reducidos. La creación de infraestructura tecnológica contempla la creación de servidores, redes, servicios y demás recursos necesarios para el almacenamiento del artefacto generado en el proceso de desarrollo de software.

En el proceso de desarrollo convencional, el desarrollador modifica el código, lo compila y genera un objeto llamado artefacto, el artefacto es enviado a las personas encargadas del

despliegue o implementación, generalmente el equipo de infraestructura tecnológica es quien realiza la creación de la infraestructura y entrega el servicio funcional al equipo de pruebas. El equipo de pruebas ingresa de forma manual, prueba las funcionalidades del servicio desplegado y autoriza el despliegue a producción (acceso de usuarios reales a los recursos productivos). El desarrollador también tiene a su cargo realizar pruebas unitarias del código, las cuales consisten en verificar cada una de las funciones incluidas en el código de manera aislada. Al automatizar los tres procesos anteriores, pruebas unitarias, pruebas de usuario y despliegue de infraestructura; se disminuye el riesgo de un error humano en el proceso, adicionalmente las tareas repetitivas son delegadas a una máquina que puede funcionar en cualquier momento del día, lo cual representa una optimización en el proceso de desarrollo y puesta en producción del código disminuyendo el "time to market" de una nueva funcionalidad, lo cual representa un ahorro económico respecto a recursos humanos destinados a la realización de la tarea. [4]

Este proyecto representa una oportunidad para optimizar y mejorar los procesos existentes relacionados con infraestructura tecnológica en la dirección de tecnología de la Universidad El Bosque, introduciendo sistemas y procesos delegados a maquinas, y disminuyendo las fallas y errores asociados al mismo. Según una investigación, las organizaciones que han adoptado prácticas de DevOps, independientemente de su tamaño o industria, han experimentado notables mejoras. Estos beneficios incluyen una disminución de 5 veces en la tasa de fracaso de cambios en producción, un aumento significativo en la frecuencia de despliegues de código (hasta 46 veces más), una reducción drástica del intervalo de tiempo entre la implementación de cambios en el código fuente y su despliegue en el producto final (440 veces más corto), y una disminución de 170 veces en el tiempo medio de reparación de errores desplegados en producción. [9]

2.4 Objetivos

2.4.1 Objetivo General

Automatizar el proceso de despliegue de infraestructura tecnológica, pruebas unitarias y pruebas de usuario en arquitecturas de red virtualizadas en el Departamento de Tecnología de la Universidad El Bosque.

2.4.2 Objetivos Específicos

1. Realizar el levantamiento de información del proceso y automatizar el proceso de despliegue de la infraestructura tecnológica requerida en una arquitectura de red virtualizada.
2. Automatizar el proceso de realización de pruebas unitarias de código.
3. Automatizar el proceso de realización de pruebas de usuario realizadas al código.

2.5 Alcance y Limitaciones del Proyecto

Alcance

Realizar la entrega de 3 procesos de infraestructura automatizados tanto ambiente de pruebas, ambiente de desarrollo como producción:

- Proceso de despliegue de artefactos.
- Proceso de creación de recursos de infraestructura.
- Proceso de pruebas al código.

Limitaciones

El proyecto estará limitado a la implementación de la infraestructura necesaria sobre la cual un microservicio esté soportado que esté a cargo del área de tecnología de la Universidad El Bosque

Las limitaciones de recursos de tecnología o infraestructura se definieron en la etapa de diseño del proyecto, quedando plasmadas en la arquitectura propuesta en la Figura 14 y en los requerimientos restrictivos del proyecto.

3. MARCO DE REFERENCIA

3.1 Marco Teórico o Conceptual

Transformación digital

Uno de los mayores desafíos que enfrentan las empresas es seguir siendo relevantes y rentables en un mercado cada vez más globalizado y competitivo. Entonces, para que cada uno de ellos responda a estos desafíos, se hace necesaria la transformación digital, independientemente de su tamaño o sector económico. Esta evolución de la empresa desde su modalidad actual y tradicional, cualquiera que sea su forma, a un estado conocido como negocio digital, donde es posible crear valor para los clientes a través de sus nuevas experiencias con productos o servicios, y nuevos modelos de negocio basados en fuentes y esquemas de generación de ingresos diferentes a los actuales, es el principal objetivo de muchas organizaciones. Un negocio es considerado digital cuando ha implementado nuevos diseños de negocio eliminando las barreras y las diferencias entre el mundo físico y el digital. [7]

Esto significa que una empresa es considerada digital utiliza las tecnologías de la información (TI) para adoptar prácticas nuevas e innovadoras para crear no sólo nuevas formas de generar riqueza sino también implementar mecanismos destinados a simplificar y automatizar los procesos creados para satisfacer las necesidades y superar las crecientes y cambiantes expectativas de los clientes [7]. Esto implica que la implementación de la estrategia empresarial y el crecimiento de la actividad económica de estas empresas ahora dependen totalmente de las TI. En vista de esto, las empresas ahora necesitan una nueva función de TI que esté equipada para ofrecer soluciones de TI que puedan satisfacer las expectativas de valor de los clientes y al mismo tiempo permitir a la empresa adaptarse rápidamente a las condiciones cambiantes del mercado, los requisitos legales y las demandas de los clientes. En concreto, una nueva función de TI que pueda implementar software con niveles adecuados de calidad cuando sea necesario en plazos que puedan medirse en horas, como máximo en días, y no en semanas.

En respuesta a esto, han surgido metodologías ágiles de desarrollo de software como Scrum y Extreme Programming (XP), que se han implementado con éxito y están ganando

popularidad rápidamente. Según una encuesta realizada entre julio y noviembre de 2015 en los cinco continentes, el 95% de las empresas ha institucionalizado prácticas ágiles de desarrollo de software, y el 62% de ellas lo hace para acelerar la entrega de aplicaciones empresariales. [10] La misma encuesta revela que, en promedio, sólo el 30% de las prácticas que se consideran esenciales para acortar el tiempo entre una idea y el software utilizable que la hace realidad se utilizan realmente. Esto sugiere que las organizaciones se han concentrado en gran medida en diseñar y poner en práctica procesos y prácticas como Scrum y Extreme Programming [10]. Esto enfatiza el desarrollo de software, dejando de lado las actividades requeridas para hacer una implementación en producción.

DevOps

Las metodologías y habilidades necesarias para implementar software de alta calidad en producción a los niveles requeridos de velocidad y eficiencia se derivan del movimiento DevOps. A continuación se presentan algunas definiciones:

"DevOps representa un cambio en la cultura de TI, foco en la rápida entrega de servicios de TI a través de la adopción de prácticas Agiles y Lean. Enfatiza en las personas (y cultura), busca mejorar la colaboración entre los equipos de operaciones y desarrollo". [11]

"DevOps es un modelo cultural y operacional que fomenta la colaboración para permitir áreas de TI de alto desempeño para lograr las metas del negocio". [12]

En marcos y cuerpos de conocimiento como Lean, Teoría de Restricciones, El Sistema de Producción Toyota y el Movimiento Toyota Kata, numerosos movimientos filosóficos y de gestión se han unido para formar DevOps. [13] Como resultado de esta convergencia ha surgido un conjunto de prácticas técnicas, arquitectónicas y culturales para abordar las cuestiones de velocidad, calidad y eficiencia en la entrega de servicios y aplicaciones con el software como componente central.

Importancia de DevOps

A diferencia de las empresas donde las prácticas de DevOps son incipientes o inexistentes, según una investigación realizada durante más de 4 años, organizaciones de todos los tamaños, industrias y tipos de tecnologías de software que han adoptado las prácticas de DevOps logran mejores resultados, cómo los presentados a continuación [14]:

- Los cambios implementados en producción tienen una tasa de fracaso 5 veces menor.
- Mayor cantidad de despliegues de código, hasta 46 veces más frecuente.
- El intervalo de tiempo entre la adición de un cambio al código fuente y la implementación de ese cambio en el producto final es 440 veces más corto.
- El tiempo medio de reparación de un error desplegado en producción se reduce 170 veces.

Otra investigación afirma que el desempeño de TI y las prácticas de DevOps influyen en el desempeño organizacional a través de evidencia cuantitativa. Según este estudio, las empresas con departamentos de TI sólidos tienen el doble de probabilidades de superar sus objetivos de rentabilidad, participación de mercado y productividad. [15]

Practicas DevOps

Según una investigación, existen 24 capacidades críticas que mejoran estadísticamente los procesos de entrega de software, lo que a su vez afecta el desempeño organizacional. Los han dividido en las siguientes 5 categorías [14]:

Categoría 1: Capacidades de entrega continua de software

1. Todos los artefactos de producción deben tener control de versión.
2. Crear un sistema de despliegue automatizado.
3. Implementar integración continua.
4. Emplear técnicas de desarrollo "basadas en troncales".
5. Configurar pruebas automáticas.
6. Ejecutar pruebas de gestión de datos.
7. Incorporar aspectos de seguridad en las fases de diseño y pruebas.

8. Iniciar un proceso continuo de entrega de software.

Categoría 2: Capacidades de arquitectura

9. Emplear arquitecturas de acoplamiento flexible.
10. Permitir que los grupos desarrollen sus propias arquitecturas.

Categoría 3: Capacidades de producto y procesos

11. Recopilar y poner en práctica los comentarios de los clientes.
12. Mantener el flujo de trabajo visible en todo momento durante el ciclo.
13. Trabajar con tamaños pequeños o por lotes.
14. Habilitar y alentar al equipo a realizar experimentos.

Categoría 4: Capacidades de gestión Lean y monitorización

15. Establecer un procedimiento sencillo para la aprobación de modificaciones.
16. Estar atento a la infraestructura y las aplicaciones para tomar decisiones comerciales.
17. Vigilar de forma proactiva la funcionalidad de los servicios y aplicaciones.
18. Gestionar y establecer límites para el trabajo en progreso (Work-InProgress).
19. Visualizar el trabajo para evaluar la calidad e informar al equipo.

Categoría 5: Capacidades culturales

20. Implementar una cultura generativa.
21. Apoyar y fomentar el aprendizaje del equipo.
22. Fomentar el trabajo en equipo y facilitarlo.
23. Ofrecer a las personas los recursos y herramientas que necesitan para que su trabajo tenga sentido.
24. Fomentar un liderazgo transformador.

Ciclos de vida de desarrollo de software

La creación, el mantenimiento, la mejora y la evolución de software o aplicaciones requieren entidades o productos complejos. Por este motivo se requieren métodos estructurados para su creación. En inglés y por sus siglas The Software Development Life Cycle (SDLC), estas técnicas se denominan Modelos de Ciclo de Vida de Desarrollo de Software. Existen muchos modelos, pero los más populares y ampliamente aplicados son los modelos Cascada, Modelo V, Prototipado, Espiral, Iterativo, Incremental y Ágil [7].

En general, cada modelo considera las etapas de planificación, análisis, diseño, desarrollo, implementación y mantenimiento. De acuerdo a lo anterior, la transformación digital requiere prácticas y modelos de ingeniería de software que permitan que las aplicaciones de software necesarias satisfagan rápidamente las necesidades comerciales cambiantes. Por esta razón, se propone el modelo ágil como modelo de ciclo de vida de desarrollo de software, el cual incluye las siguientes fases o etapas [16]:

- Concepción
- Incepción
- Construcción / Iteraciones
- Transición
- Producción
- Retiro

A continuación, se muestra de forma gráfica el ciclo de vida de desarrollo de software ágil que muestra un enfoque flexible y colaborativo para la creación de software que se basa en la entrega incremental y continua de productos de software funcionales. A diferencia de los enfoques tradicionales de desarrollo de software, como el modelo en cascada, que se basan en una planificación y ejecución rígidas, el desarrollo ágil se adapta a las cambiantes necesidades del cliente y al entorno empresarial.

Figura 1. Ciclo de entrega ágil[17]

Entrega continua de software

Además de escribir código que cumpla con los requisitos del negocio y del cliente, es necesario llevar a cabo un conjunto diferente de tareas en los procesos relacionados con la construcción, prueba, implementación y lanzamiento a producción del software para lograr la implementación del software con los criterios requeridos de rapidez, calidad y eficiencia. La entrega continua se refiere a estos procedimientos que se planifican, se llevan a cabo y están tan completamente automatizados como sea posible. Este es un método de ingeniería de software en el que los equipos producen continuamente software con valor comercial en iteraciones rápidas, al tiempo que garantizan que el software pueda lanzarse de manera confiable en cualquier momento. Cuando se cumplen los requisitos antes mencionados, una organización ha implementado la entrega continua[18]:

- En cualquier punto del ciclo de vida, el software siempre está preparado para ser desplegado.

- En lugar de trabajar en nuevas funciones, el equipo da mayor prioridad a tener el software siempre listo para su implementación.
- Cualquier persona puede saber en cualquier momento de forma rápida y automática si un sistema está preparado para la implementación de software después de un cambio.
- Cualquier versión del software se puede instalar instantáneamente en cualquier entorno con solo hacer clic en un botón.

La entrega continua se refiere a un conjunto de principios y técnicas diseñadas para disminuir los costos, reducir los plazos y mitigar los riesgos relacionados con la introducción de nuevas características o la corrección de fallos en el software destinado a los usuarios. [19] Es importante destacar que, con la creciente adopción de metodologías ágiles, el desarrollo de software se lleva a cabo de forma gradual y progresiva. Por lo tanto, los mayores esfuerzos se concentran después de la primera implementación y puesta en producción, no solo para mantener el software en funcionamiento, sino también para su evolución y mejora continua. En este contexto, estas prácticas se vuelven especialmente relevantes y valiosas para el éxito del proyecto.

En la actualidad, el proceso de despliegue y liberación de software en cualquier etapa de su ciclo de vida se considera deseable y factible cuando cumple con ciertos criterios: ser frecuente, predecible, rápido, de bajo riesgo y económico. Existen ejemplos de casos exitosos, prácticas recomendadas documentadas y herramientas de software disponibles para orquestar y automatizar las diversas actividades relacionadas con la Entrega Continua. Los fundamentos en los cuales se basa la entrega continua de software se resumen a continuación [20]:

1. Establecer un proceso repetible y confiable para la liberación de software: Mediante la repetición constante de procesos, se logra una entrega de resultados más predecible y confiable. Sin embargo, para alcanzar este objetivo, es esencial combinarlo con la automatización y el control de versiones.

2. Automatizar todo lo que sea posible: La obtención de resultados coherentes y predecibles, así como la capacidad de desplegar y liberar software con un simple clic, solo es viable al delegar tareas repetitivas a las máquinas. La intervención humana, en lugar de facilitar, puede aumentar el riesgo de errores.

3. Gestionar todo dentro de un sistema de control de versiones es esencial: Esto incluye cualquier elemento necesario para construir, desplegar, probar y liberar una aplicación, como scripts de prueba, casos automatizados de prueba y scripts para configuraciones de red, servidores web, bases de datos y aplicaciones, entre otros.

4. Integrar la calidad en cada etapa es una premisa fundamental en la entrega continua: La mayoría de las prácticas y procesos están diseñados para detectar defectos lo más temprano posible durante el ciclo de despliegue y liberación. Se busca identificar problemas antes de que los artefactos sean incorporados al control de versiones. Se parte del principio de que cuanto antes se encuentren los errores, menor será el costo de su corrección. Esto implica que las pruebas no son una fase independiente y que la responsabilidad de realizar pruebas recae en todos los miembros del equipo, no solo en el equipo de pruebas.

5. "Completado significa liberado": Una nueva funcionalidad o corrección solo se considera completada cuando se ha implementado o liberado, al menos en un entorno similar al de producción. La verdadera finalización de una característica solo ocurre cuando llega a manos de los usuarios del software. Aunque hay situaciones en las que no es apropiado desplegar en el entorno de producción, se acepta el despliegue en un entorno equivalente cuando sea necesario.

6. La responsabilidad es compartida por todos en el proceso de entrega: La colaboración de todos los involucrados en la entrega de software es esencial. La entrega exitosa es un logro colectivo, y el fracaso no recae en individuos, sino en el equipo en su conjunto. La entrega de software no se basa en una única persona o equipo funcional, lo que subraya la importancia del trabajo en equipo desde el principio.

7. Búsqueda continua de mejoras: La primera vez que una aplicación se implementa y libera en producción marca el comienzo de su ciclo de vida. Con certeza, la aplicación evolucionará y se adaptará continuamente a las necesidades del negocio. La corrección de errores será esencial para mantener su funcionamiento óptimo. A medida que se realicen más despliegues y liberaciones, el proceso en sí debe evolucionar para mejorar su eficiencia y efectividad en todas las etapas.

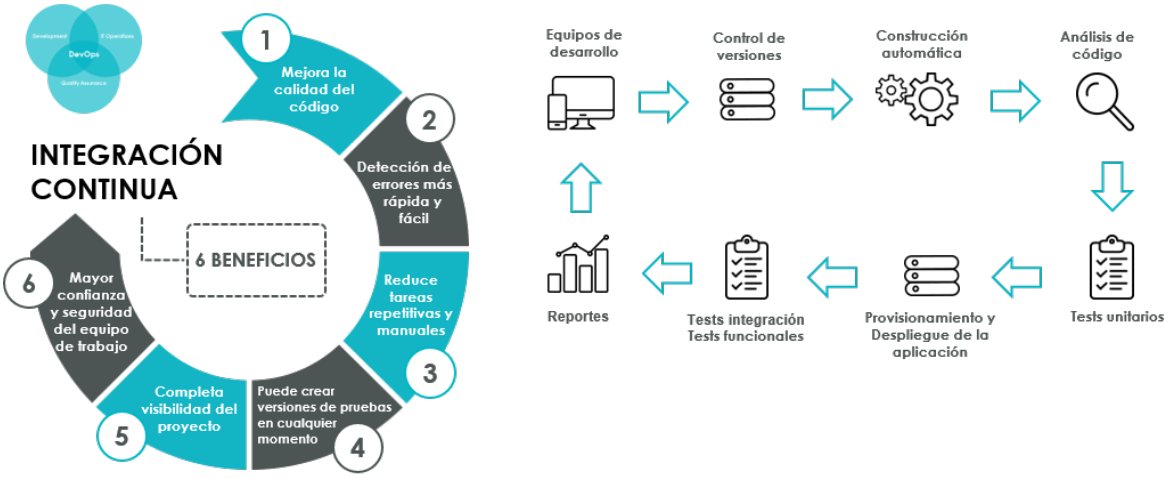


Figura 2. Beneficios de la integración continua y la entrega continua de software[21]

Importancia de la entrega continua de software

Uno de los aspectos más significativos de la Entrega Continua de software es su capacidad para establecer un proceso de liberación de software en producción que sea sistemático, confiable y anticipado. [20] Este enfoque conlleva una notable disminución en los tiempos de entrega, lo que, a su vez, facilita una entrega más ágil de nuevas características o correcciones de errores a los usuarios finales, cumpliendo con los estándares de calidad requeridos.

La Entrega Continua de software desempeña un papel fundamental en el contexto empresarial actual, donde la rapidez de adaptación a las cambiantes demandas del mercado es esencial. Al establecer un proceso repetible y predecible, las organizaciones pueden responder de manera más ágil a las necesidades de los usuarios y a las oportunidades emergentes. Esta agilidad en la entrega de software permite que las empresas mantengan su

ventaja competitiva al proporcionar de manera constante nuevas funcionalidades y mejoras a sus clientes, manteniendo así una relación de confianza y satisfacción. [20]

Además de acelerar la entrega de software, la Entrega Continua también contribuye significativamente a la calidad del producto final. Al automatizar pruebas, revisar constantemente el código y garantizar la integridad de las versiones, se minimiza la posibilidad de errores y fallos en producción. Esto se traduce en una experiencia de usuario más confiable y satisfactoria, lo que, a su vez, mejora la reputación de la empresa y su posición en el mercado. En resumen, la Entrega Continua de software no solo impulsa la eficiencia en el desarrollo, sino que también eleva la calidad y la competitividad de las organizaciones en un entorno empresarial cada vez más dinámico y competitivo. [22]

Gestión de la configuración

La gestión de la configuración es un componente crítico que abarca más allá del control de versiones, aunque a menudo se les asocie. Es esencialmente un proceso integral que se concentra en las interdependencias entre todos los artefactos relacionados con el software, así como su almacenamiento, recuperación, identificación única y modificación. Esta gestión tiene un conjunto de actividades clave [23]:

1. Identificación de líneas base: En este proceso, se establecen las líneas base de configuración para cada artefacto. Estas líneas de base representan versiones particulares de cosas, como configuraciones, versiones de código o documentación.

2. Control de cambios: La gestión de la configuración implica la regulación y documentación de todos los cambios realizados en los artefactos o ítems de configuración. Esto garantiza que cualquier modificación sea rastreable y reversible, lo cual es crucial para mantener la integridad del sistema.

3. Construcción y especificaciones: Se incluye la capacidad de construir artefactos o proporcionar especificaciones para su construcción a partir del sistema de gestión de la

configuración. Esto es esencial para garantizar que se puedan producir versiones consistentes y reproducibles de los artefactos.

4. Integridad de las líneas base: Mantener la integridad de las líneas base es fundamental para asegurar que las versiones y configuraciones acordadas se mantengan consistentes y sin corrupción.

5. Proveer información precisa: La gestión de la configuración también tiene como objetivo proporcionar información precisa sobre el estado y los datos de configuración actual a los desarrolladores, usuarios y clientes. Esto garantiza que todos tengan una comprensión clara de los componentes de software y de infraestructura.

Importancia de la gestión de la configuración

El papel de la gestión de la configuración es particularmente importante y va más allá del simple seguimiento de los cambios de versión. La gestión de la configuración busca lograr dos objetivos principales. Primero, la capacidad de automatizar completamente la replicación de entornos, asegurando que cada entorno creado de esta manera tenga exactamente la misma configuración. [19] El segundo es la trazabilidad, que le permite identificar versiones en cualquier entorno de forma rápida y precisa, así como contrastar versiones anteriores para encontrar diferencias. De estos objetivos fundamentales se derivan importantes ventajas en varios ámbitos, entre ellos:

1. Recuperación de desastres: La capacidad de reconstruir rápidamente entornos en caso de eventos catastróficos o fallos graves es esencial para la recuperación de servicios de manera eficiente y efectiva.

2. Auditoría: Para demostrar la integridad del proceso de entrega de software, es necesario proporcionar información sobre los componentes, sus versiones y su historial. Esto es fundamental para garantizar la calidad y la transparencia en la entrega.

3. Calidad: La reducción de los tiempos necesarios para configurar entornos permite dedicar más tiempo a construir calidad en cada componente y realizar pruebas exhaustivas con una mayor cobertura. Esto contribuye directamente a la mejora de la calidad del software.

4. Velocidad en corrección de defectos: En caso de descubrir errores o vulnerabilidades de seguridad, la capacidad de reproducir entornos de forma automática permite implementar rápidamente nuevas versiones del software en producción. Esto agiliza la respuesta a problemas y la corrección de defectos.

Integración continua

La Integración Continua es una práctica fundamental en DevOps que se enfoca en la colaboración constante y la integración regular del código generado por los miembros de un equipo de desarrollo. Como su nombre lo sugiere, la característica distintiva de la Integración Continua es la frecuencia constante de estas integraciones, con la mayoría de los desarrolladores realizando al menos una integración diaria. Esto resulta en múltiples integraciones a lo largo del día.[24]

Cada integración se somete a un proceso de construcción automática, conocido como "Build", que incluye pruebas destinadas a identificar errores de integración tan pronto como sea posible. Esta práctica tiene como objetivo principal asegurar que el software se mantenga en un estado operativo y estable de manera constante. Se logra garantizando que el código desarrollado por cada miembro del equipo sea compatible con el de sus compañeros. [25]

La Integración Continua significa que los desarrolladores mantienen su trabajo en progreso constantemente integrado con el de los demás.[7] Esto permite detectar y resolver conflictos de integración y errores en una etapa temprana del desarrollo, evitando que se acumulen y se conviertan en problemas más difíciles de abordar. Además, asegura que el software siempre esté en un estado funcional, lo que facilita el desarrollo de nuevas características y la corrección de errores de manera más ágil. En resumen, la Integración Continua es una práctica esencial que promueve la colaboración eficiente y la calidad constante en el proceso de desarrollo de software en el contexto de DevOps.[22]

Construcción automática (build): La comprensión completa del concepto de Integración Continua requiere abordar el concepto de Construcción Automática. El término "compilación automática" se refiere a una variedad de tareas que incluyen compilación, prueba, inspección, implementación y otras actividades relacionadas. [25] No es sólo una construcción sencilla. La tarea fundamental de AutoBuild es garantizar que el software funcione como una única unidad cohesiva unificando el código. En adelante, en este contexto, los términos "construcción" y "construcción automática" se emplearán indistintamente.

Pruebas: Como se mencionó previamente en la definición de Construcción Automática, este proceso puede incluir pruebas de diversos tipos, como pruebas unitarias, de componentes, de sistema, de rendimiento, de seguridad y otras. [21].

Ciclo de integración continua: El Ciclo de Integración Continua representa una parte esencial de la práctica de Integración Continua en DevOps. Este proceso se centra en la colaboración constante y la integración regular del código generado por los miembros del equipo de desarrollo.[25] Se puede entender el Ciclo de Integración Continua como un flujo continuo de actividades que se describen a continuación [7]:

1. El proceso comienza cuando se necesita realizar modificaciones en el código existente por diversas razones, como la incorporación de nuevas características, la corrección de errores observados en el entorno de producción o la optimización del código para reducir la deuda técnica.

2. Para abordar estos cambios, el desarrollador encargado crea una copia del código actual, que se conoce como "código integrado," en su entorno de desarrollo local.

3. A continuación, el desarrollador procede a realizar todas las acciones necesarias para lograr los objetivos del desarrollo, que pueden incluir la incorporación de cambios en el código existente y la actualización de las pruebas automatizadas correspondientes.

4. Posteriormente, se lleva a cabo una construcción automática en la máquina de desarrollo del programador. Si la compilación se realiza sin errores, se generan los ejecutables y las pruebas automatizadas se ejecutan exitosamente, se considera que la construcción ha sido satisfactoria.

5. Después de lograr una construcción automática exitosa en su máquina local, el desarrollador realiza una acción adicional. En este punto, vuelve a tomar una copia del código del repositorio de versiones. Esta acción es necesaria debido a que podría haber habido cambios en el repositorio desde que se hizo la primera copia en el paso 1. Luego, el desarrollador incorpora los cambios que realizó en el paso 3 y ejecuta nuevamente una construcción automática en su máquina local, siguiendo el mismo proceso del paso 4. Si esta construcción se ejecuta sin problemas, el desarrollador está listo para proponer (hacer un "Commit" o "Check-In") sus cambios al repositorio. Sin embargo, si los cambios realizados en el paso 3 generan errores en la construcción automática, el desarrollador debe trabajar en corregir el código hasta que los cambios funcionen correctamente con la copia más reciente del repositorio de control de versiones antes de proponer sus modificaciones.

6. Después de que el desarrollador propone sus cambios ("Commit" o "Check-In"), se procede a realizar una nueva construcción automática (Build). Sin embargo, esta vez, la construcción no se lleva a cabo en la máquina local del desarrollador, sino en una máquina designada como la "Máquina de Integración." Para esto, se toma el código fuente del repositorio, que ahora contiene los cambios propuestos ("Commit" o "Check-In"). Si la construcción automática es exitosa, se considera que los cambios se han completado de manera exitosa. No obstante, si surgen errores durante la construcción, el desarrollador debe trabajar en la corrección del código hasta obtener el resultado deseado en la construcción.

Automatizar el proceso de integración continua

El proceso previamente detallado es una secuencia altamente automatizada que facilita la colaboración simultánea de varios miembros del equipo de desarrollo. Su finalidad es prevenir conflictos entre desarrolladores y detectar errores mediante la detección temprana de cambios incompatibles entre sí, lo que a su vez evita la realización de construcciones

automáticas cuando se identifican tales incompatibilidades. A continuación en la figura 3, se presentan las partes para un sistema automatizado de integración continua, evidenciando el proceso de infraestructura que ocurre desde que el desarrollador realiza cambios los cambios hasta que estos se compilan, se integran a la base de datos, ejecuta las pruebas y se despliega en el ambiente de producción.

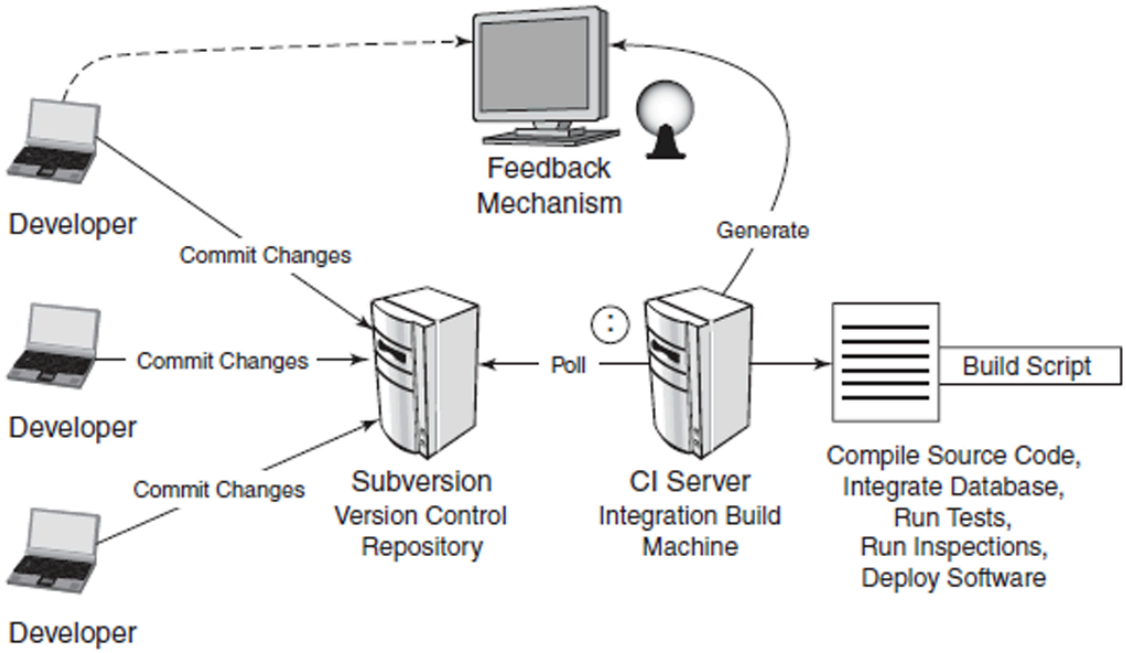


Figura 3. Componentes del Sistema Automático de Integración Continua[21]

Al automatizar este proceso, el equipo de desarrollo tiene la tarea de hacer commit a las modificaciones realizadas, es decir, subir sus cambios al repositorio de control de versiones. Este repositorio es monitoreado de manera continua por un servidor de integración que verifica si se han realizado cambios en el código fuente. Cuando el servidor identifica que hubo algún cambio, hace una copia del código nuevo y ejecuta un script de construcción automatizada que lleva a cabo la integración del software. Posteriormente, notifica a los desarrolladores acerca de los resultados obtenidos. [7]

Java

El lenguaje de programación Java es un lenguaje de alto nivel ampliamente utilizado en el desarrollo de aplicaciones de software en una variedad de dominios. Fue desarrollado por Sun Microsystems a mediados de la década de 1990 y se ha convertido en uno de los

lenguajes más populares y versátiles en el mundo de la programación. Java se caracteriza por su portabilidad, sintaxis limpia y estructurada, y la robustez de sus aplicaciones. [26]

Uno de los aspectos más notables de Java es su capacidad de ser ejecutado en múltiples plataformas sin necesidad de modificaciones significativas en el código fuente. Esto se logra gracias a la máquina virtual Java (JVM), que actúa como una capa de abstracción entre el código Java y el sistema operativo. Como resultado, las aplicaciones Java pueden ser escritas una vez y ejecutarse en cualquier plataforma que disponga de una JVM compatible. [26]

Java se destaca por su enfoque en la orientación a objetos. Utiliza una sintaxis clara y coherente para definir clases y objetos, lo que facilita la creación de aplicaciones modularizadas y reutilizables. La herencia, la encapsulación y el polimorfismo son conceptos fundamentales en la programación orientada a objetos en Java. [26]

La biblioteca estándar de Java, conocida como la API de Java, proporciona una amplia gama de clases y métodos predefinidos que abordan una variedad de necesidades de desarrollo. Esto incluye manipulación de archivos, operaciones de red, manejo de excepciones, gráficos, acceso a bases de datos y más. [26] La API de Java es una parte integral de la plataforma y permite a los desarrolladores crear aplicaciones poderosas de manera eficiente.

Java es un lenguaje utilizado en diversas áreas, desde aplicaciones de escritorio hasta desarrollo web, aplicaciones móviles, sistemas integrados y soluciones empresariales cómo las pruebas automatizadas. [27] La adopción de Java en la comunidad de desarrollo se ha traducido en una amplia cantidad de bibliotecas y marcos de trabajo específicos para tareas y sectores particulares. [28]

Java se ha mantenido relevante y actualizado a lo largo de los años, con nuevas versiones que introducen características y mejoras significativas. La introducción de módulos en Java 9, por ejemplo, mejoró la modularidad de las aplicaciones, mientras que Java 11 introdujo un enfoque de desarrollo a largo plazo (LTS) que brinda estabilidad y soporte a largo plazo a los proyectos.

Java es un lenguaje que se ha convertido en una base fundamental para la industria de la tecnología, especialmente en el ámbito empresarial. Empresas de todo el mundo confían en aplicaciones Java para impulsar sus operaciones diarias y brindar servicios a sus clientes. [27] Esto se debe en parte a la seguridad y estabilidad de Java, que lo convierten en una elección segura para entornos críticos y sistemas de alto rendimiento.

Pruebas automatizadas usando Java

Java es ampliamente reconocido en el campo de la automatización de pruebas debido a su versatilidad y a las numerosas bibliotecas y herramientas disponibles para este propósito. [29] En la automatización de pruebas, los equipos de QA utilizan Java para crear scripts que permiten verificar la funcionalidad y el rendimiento de aplicaciones y sistemas de software de manera automatizada.

Una de las ventajas clave de Java es la disponibilidad de bibliotecas y frameworks diseñados específicamente para la automatización de pruebas. [29] Por ejemplo, TestNG es un framework popular que facilita la creación y ejecución de pruebas unitarias y de integración en aplicaciones Java. [28] Además, bibliotecas como Selenium WebDriver son usualmente utilizadas para automatizar pruebas en navegadores web, lo que es esencial para las pruebas de aplicaciones web.

La facilidad de uso de Java es otro factor que lo convierte en una opción atractiva para la automatización de pruebas. Su sintaxis clara y legible permite a los desarrolladores y profesionales de pruebas crear scripts de manera eficiente y mantenerlos con facilidad a lo largo del tiempo. [30] Esto fomenta la colaboración entre equipos de desarrollo y pruebas al utilizar un lenguaje común.

Java se integra de manera efectiva con una variedad de herramientas de automatización de pruebas y entornos de desarrollo integrado (IDE). [29] Los desarrolladores pueden utilizar IDEs como Eclipse junto con bibliotecas y plugins diseñados para la automatización de pruebas. Esta integración simplifica la creación y ejecución de pruebas automatizadas. Para [30] una característica importante de Java es su capacidad de ser ejecutado en múltiples

plataformas y sistemas operativos. Esto significa que los scripts de prueba escritos en Java son portables y pueden ejecutarse en diferentes entornos sin problemas. Esto es especialmente beneficioso cuando se prueba aplicaciones en entornos variados.

Repositorio Maven

El repositorio Maven es una parte esencial en el contexto del sistema de gestión de proyectos Maven, especialmente utilizado en el desarrollo de software basado en Java. [31] Se trata de un componente centralizado y organizado donde se almacenan y gestionan los artefactos necesarios para la construcción y compilación de proyectos Java. Estos artefactos incluyen bibliotecas de código, JAR (Java Archive) y plugins que se utilizan en el desarrollo de proyectos Maven. La gestión centralizada de estas dependencias es uno de los aspectos clave del repositorio Maven, ya que evita la necesidad de descargar manualmente cada componente de software y automatiza este proceso. [32] Además, el sistema de resolución de dependencias de Maven garantiza que las versiones de las bibliotecas y plugins sean compatibles entre sí, lo que previene conflictos y garantiza la coherencia de las aplicaciones. [31]

Los desarrolladores pueden configurar Maven para utilizar el repositorio central de Maven o cualquier otro repositorio personalizado que contenga bibliotecas específicas. Esta flexibilidad permite el acceso a una amplia variedad de componentes de software según las necesidades del proyecto. Maven se integra en el ciclo de vida de desarrollo de proyectos, lo que significa que se utiliza para tareas clave como compilación, empaquetado, pruebas y despliegue de aplicaciones. [32] Usando este repositorio, cada artefacto en el repositorio Maven está asociado con una versión específica, lo que facilita el control de versiones y garantiza que las aplicaciones mantengan la coherencia en el tiempo.

El uso de pruebas en Maven

En Maven, las pruebas unitarias se realizan típicamente como parte del ciclo de construcción y pruebas del proyecto. Para ello, Maven utiliza complementos específicos que se encargan de la ejecución de las pruebas unitarias. [33] Uno de los complementos más comunes para

este propósito es TestNG. Este complemento se configura en el archivo de configuración de Maven (pom.xml) y permite definir cuáles son las clases de pruebas y dónde se encuentran.

Maven facilita la ejecución de pruebas unitarias. A menudo, se utilizan marcos de pruebas como JUnit o TestNG para escribir y diseñar las pruebas unitarias. Estos marcos proporcionan las estructuras y métodos necesarios para definir y ejecutar pruebas de manera efectiva. [34]

La integración de Maven y TestNG permite automatizar la ejecución de pruebas unitarias como parte del proceso de construcción y pruebas del proyecto. Cuando se ejecuta el comando de construcción de Maven, este se encarga de compilar el código fuente, incluir las dependencias necesarias, como bibliotecas de pruebas, y luego ejecutar las pruebas definidas. [34] El resultado de estas pruebas se presenta en el informe de construcción de TestNG, lo que facilita la identificación de problemas o errores en el código.

TestNG

TestNG, que significa "Test Next Generation", es un marco de pruebas automatizadas ampliamente utilizado en el desarrollo de software Java. Este marco fue diseñado para abordar las limitaciones de JUnit, otro marco de pruebas de Java, y ha ganado popularidad debido a su flexibilidad y capacidades avanzadas para la automatización de pruebas. [34] Una de las características clave de TestNG es su capacidad para admitir diferentes tipos de pruebas, lo que lo hace adecuado tanto para pruebas unitarias como para pruebas de integración y pruebas de usuario. Esto lo convierte en una herramienta versátil que puede cubrir múltiples aspectos de la calidad del software en un proyecto.

TestNG también ofrece la capacidad de agrupar pruebas relacionadas, lo que facilita la organización y la ejecución selectiva de grupos específicos de pruebas. [34] Este framework también admite la definición de dependencias entre pruebas, lo que garantiza que las pruebas se ejecuten en el orden adecuado y que las pruebas subsecuentes no se ejecuten si una prueba anterior falla. La generación de informes detallados es otra característica destacada de TestNG. [28] Después de la ejecución de las pruebas, TestNG genera informes que muestran

los resultados de cada prueba, incluidas las pruebas exitosas y fallidas, las excepciones lanzadas y el tiempo que llevó ejecutar cada prueba. Estos informes son útiles para evaluar la calidad del código y proporcionan información valiosa para solucionar problemas.

TestNG también es compatible con la ejecución de pruebas en paralelo, lo que puede acelerar significativamente el proceso de prueba, especialmente en aplicaciones grandes y complejas. Esto es particularmente beneficioso para proyectos que buscan aumentar la eficiencia de las pruebas y reducir el tiempo de desarrollo. Una de las principales ventajas de TestNG es su flexibilidad, esto debido a que permite la ejecución de pruebas en paralelo, lo que acelera significativamente el proceso de prueba, especialmente en proyectos grandes y complejos. [28] Esto es crucial para garantizar que las pruebas sean eficientes y se completen en un tiempo razonable.

Otra característica importante de TestNG es la parametrización de pruebas. Esto permite ejecutar la misma prueba con diferentes conjuntos de datos, lo que es útil para probar diferentes condiciones o escenarios utilizando la misma lógica de prueba. De este modo, TestNG también admite Data Providers, lo que facilita la importación de datos de fuentes externas, como hojas de cálculo o archivos XML, para su uso en pruebas. [28] Esto mejora la reutilización de datos y la capacidad de ejecutar pruebas con múltiples conjuntos de datos.

El marco de pruebas proporciona una gestión eficiente de dependencias entre pruebas, lo que garantiza que se ejecuten en el orden correcto. Si una prueba depende de otra que falla, TestNG puede omitir automáticamente las pruebas subsecuentes, lo que ahorra tiempo en la depuración. [28] Otra característica importante es que TestNG se integra de manera fluida con IDE populares como Eclipse, así como con herramientas de construcción como Maven y Gradle. Esto permite una ejecución de prueba continua y automatizada, lo que es esencial en entornos de desarrollo ágiles.

Lenguaje Yaml

El lenguaje YAML, que significa "YAML Ain't Markup Language" (YAML no es un lenguaje de marcado), es un formato de serialización de datos legible por personas sin

conocimientos técnicos y de fácil escritura. [35] Aunque no se utiliza para propósitos de programación como un lenguaje de alto nivel, desempeña un papel fundamental en la configuración y el intercambio de datos estructurados en una variedad de aplicaciones, incluidas las relacionadas con la gestión de la configuración, la automatización y la definición de infraestructura.

Las características clave de YAML incluyen su legibilidad, ya que utiliza una sintaxis sencilla y fácil de entender basada en sangrías y la estructura de anidamiento. Esto lo hace muy accesible para los usuarios y permite que los archivos YAML sean editados y comprendidos con facilidad. [35]

YAML se utiliza ampliamente para configurar aplicaciones y sistemas, especialmente en el contexto de la automatización y la infraestructura como código (IaC) [36]. Herramientas populares como Ansible, Kubernetes, Docker Compose y AWS CloudFormation permiten a los usuarios definir configuraciones y recursos utilizando YAML. Por ejemplo, un archivo YAML en Kubernetes puede definir cómo se deben implementar y escalar aplicaciones en un clúster de contenedores. También se menciona su uso en la configuración de aplicaciones y la definición de recursos, YAML se utiliza en la creación de archivos de configuración para herramientas de CI/CD (Integración Continua / Entrega Continua) como Jenkins o GitLab CI/CD. [36] Los archivos YAML especifican cómo deben llevarse a cabo los flujos de trabajo de desarrollo y entrega, lo que permite la automatización de pruebas, compilaciones y despliegues.

Otra área en la que YAML ha ganado popularidad en la definición de plantillas para la infraestructura en la nube. Por ejemplo, AWS CloudFormation utiliza archivos YAML (o JSON) para describir recursos como instancias de EC2, grupos de seguridad y redes virtuales, lo que permite a los usuarios definir toda su infraestructura como código. [36]

Docker

Docker es una plataforma para almacenar servicios en contenedores que se ha convertido en una herramienta fundamental en el desarrollo y despliegue de aplicaciones. Los contenedores

son entornos ligeros y portátiles que encapsulan una aplicación y todas sus dependencias, lo que permite que se ejecute de manera consistente en cualquier entorno que admita Docker. [37] Esta tecnología se basa en el concepto de contenedores, que son instancias aisladas de aplicaciones y su entorno de ejecución.

"Una de las ventajas clave de Docker es su capacidad para crear contenedores de aplicaciones que son independientes del sistema operativo subyacente, lo que facilita la portabilidad y la distribución de aplicaciones". [37] Esto significa que los desarrolladores pueden crear un contenedor que incluye su aplicación y todas las bibliotecas y dependencias necesarias, y luego ejecutar ese contenedor en cualquier sistema que ejecute Docker, ya sea en un entorno de desarrollo local, un servidor de pruebas o en la nube.

Docker utiliza un formato estándar llamado Dockerfile para definir la configuración de un contenedor, lo que facilita la creación y gestión de contenedores de manera reproducible. Docker también ofrece herramientas como Docker Compose, que permite definir y gestionar aplicaciones multi-contenedor, y Docker Swarm y Kubernetes para la orquestación de contenedores en clústeres de servidores. [38]

Otra característica fundamental de Docker es su capacidad de aislamiento. Los contenedores utilizan técnicas de aislamiento a nivel de sistema operativo para garantizar que sean independientes entre sí. [38] Esto quiere decir que múltiples contenedores pueden coexistir en un solo servidor sin interferir entre ellos. Este nivel de aislamiento proporciona una mayor seguridad y confiabilidad en comparación con otras formas de empaquetar aplicaciones. La eficiencia de recursos es otro aspecto destacado de Docker. A diferencia de las máquinas virtuales tradicionales, los contenedores Docker comparten el mismo kernel del sistema operativo. [37] Esto los hace ligeros y permite una mayor eficiencia en la utilización de recursos, lo que significa que se pueden ejecutar más contenedores en una sola máquina física.

En el contexto de la automatización y la integración continua, Docker se ha convertido en una herramienta esencial. Los equipos de desarrollo pueden crear imágenes de contenedores

que encapsulan sus aplicaciones y configuraciones, lo que garantiza la consistencia entre los entornos de desarrollo, pruebas y producción. La automatización de la construcción, pruebas y despliegue de contenedores Docker se ha vuelto esencial para acelerar la entrega continua de software de alta calidad.

Blue green deployment

El despliegue de Blue-Green es una estrategia avanzada en la gestión de la implementación de software que se utiliza para lograr una entrega continua confiable y de baja interrupción de aplicaciones. [39] Esta estrategia se basa en la idea de tener dos entornos completamente separados, uno denominado "azul" (blue) que representa la versión en producción actual, y el otro denominado "verde" (green) que representa la nueva versión que se va a implementar. Ambos entornos son idénticos en términos de infraestructura y configuración.

La idea principal del despliegue de Blue-Green es permitir una transición suave y segura de la versión anterior (azul) a la nueva versión (verde) de una aplicación. En un despliegue de Blue-Green, ambas versiones (azul y verde) del software se ejecutan simultáneamente en entornos separados, lo que permite que las dos versiones se ejecuten sin conflictos y sin afectar a los usuarios finales.

En un estudio realizado por (Yng B, Sailer A, Mohindra A) [40] se identificó que antes de realizar el cambio de tráfico de la versión azul a la verde, se deben realizar pruebas exhaustivas en el entorno verde, lo que incluye pruebas funcionales, pruebas de rendimiento y cualquier otra verificación necesaria para garantizar que la nueva versión funcione correctamente. Una vez confirmado que la nueva versión en el entorno verde es estable y cumple con los criterios de calidad, se redirige el tráfico de manera controlada desde la versión azul a la verde mediante ajustes en la configuración de enrutamiento o equilibradores de carga o balanceadores.

Una de las ventajas clave del despliegue de Blue-Green es la capacidad de reversión rápida en caso de que surjan problemas inesperados en la nueva versión (verde). Si se detectan problemas críticos, se puede revertir el tráfico al entorno azul sin tiempo de inactividad

perceptible para los usuarios ya que todo esto se hace en el backend y sin afectar el entorno productivo. [39] Durante y después del despliegue, se monitorea de cerca el rendimiento y la estabilidad de la nueva versión en el entorno verde para identificar problemas de manera temprana y tomar medidas correctivas rápidamente en caso de que sea necesario.

La automatización juega un papel fundamental en el despliegue de Blue-Green, ya que se utiliza en todas las etapas, desde la construcción y prueba de la nueva versión hasta la redirección del tráfico y la reversión en caso de problemas. [39] Esta estrategia es especialmente valiosa en entornos donde la disponibilidad y la confiabilidad son críticas, como aplicaciones en producción.

4. DESARROLLO DEL PROYECTO DE GRADO

4.1 Requerimientos

Requerimientos Funcionales

1. El sistema debe ser capaz de implementar y configurar entornos virtuales en la nube de AWS de la Universidad El Bosque.

2. La solución debe permitir realizar pruebas unitarias de los contenedores de la infraestructura virtualizada, utilizando herramientas de automatización de pruebas unitarias.

3. El sistema debe permitir realizar un conjunto de pruebas de usuario para validar el correcto funcionamiento de la infraestructura virtualizada, utilizando herramientas de automatización de pruebas de usuario.

4. La solución debe permitir la integración con otras herramientas y servicios de AWS para realizar el despliegue automatizado en diferentes entornos (desarrollo, prueba y producción).

5. El sistema debe permitir la personalización de los procesos de despliegue y configuración para adaptarse a las necesidades específicas de la Dirección de Tecnología de la Universidad El Bosque.

6. El sistema debe estar disponible para realizar despliegues, evitando interrupciones en el proceso de despliegue y configuración, realizando la configuración automatizada de los contenedores de la infraestructura tecnológica.

7. La plataforma debe permitir la gestión de versiones de los componentes de la infraestructura tecnológica y de los scripts de automatización.

Requerimientos de Calidad

1. El sistema debe garantizar la calidad del código y la funcionalidad de la infraestructura a través de pruebas unitarias y de usuario automatizadas.

2. La documentación proporcionada debe asegurar la correcta implementación y configuración de la solución, garantizando su funcionamiento a largo plazo. La documentación debe incluir diseño técnico, especificación funcional y manual de operación.

3. El sistema debe generar informes de resultados de pruebas unitarias y pruebas de usuario automatizadas para evaluar la calidad y el rendimiento del sistema.

Requerimientos Restrictivos

1. La implementación se debe realizar en la cuenta CODE de manera transversal con las cuentas de DEV, QA y PROD.
2. La implementación se realizará de manera escalonada, iniciando por la cuenta DEV, QA y finalmente PROD.
3. La implementación se llevará a cabo, hasta donde se permita, únicamente con herramientas de la suite de AWS ya que es el proveedor contratado por la Universidad el Bosque por lo cual los lineamientos de la dirección de tecnología restringen el uso de herramientas de otras suites.

4.2 Metodología del Diseño

El trabajo realizado en este proyecto de grado se dividió en cuatro fases, cada una con objetivos específicos y una metodología de trabajo definida. Se siguió un enfoque metodológico que abarcó desde la planificación y diseño hasta la implementación, pruebas y entrega. Cada fase tuvo sus objetivos específicos y actividades relacionadas, lo que permitió una gestión eficiente del proyecto y el logro de los resultados esperados.

Fase 1: Planificación y diseño

En esta etapa inicial, el equipo se centró en definir los objetivos y alcances del proyecto, identificar los requerimientos y seleccionar las herramientas y tecnologías que se utilizarían. Se creó un equipo de trabajo y se diseñó la arquitectura de la solución, incluyendo diagramas de flujo de los procesos de automatización. La metodología se basó en la recopilación y análisis de información, así como en la toma de decisiones estratégicas para el proyecto. A continuación se relacionan las actividades llevadas a cabo durante esta fase:

- **Semana 1 (10/04/2023 - 14/04/2023):** Definición de objetivos, alcance y requerimientos del proyecto. Selección de herramientas y tecnologías a utilizar. Creación del equipo de trabajo.
- **Semana 2 (17/04/2023 - 21/04/2023):** Diseño de la arquitectura de la solución y creación de los diagramas de flujo de los procesos de automatización.
- **Semana 3 (24/04/2023 - 28/04/2023):** Identificación de los recursos necesarios para la implementación del proyecto. Definición de los criterios de aceptación y de las pruebas de validación.
- **Semana 4 (01/05/2023 - 05/05/2023):** Revisión y aprobación del plan de proyecto. Presentación del plan a los interesados y patrocinadores del proyecto.

Fase 2: Implementación

Esta fase representó la parte más extensa del proyecto y se centró en la implementación práctica de las herramientas y tecnologías seleccionadas. Se configuró la infraestructura de la solución virtualizada y se integraron las herramientas de automatización en los procesos de despliegue de infraestructura tecnológica. La metodología involucró la programación y configuración técnica, además de pruebas intermedias para verificar la funcionalidad de los componentes. A continuación se relacionan las actividades llevadas a cabo durante esta fase:

- **Semana 5 (08/05/2023 - 12/05/2023):** Entrega y gestión de accesos a las cuentas de AWS otorgados por la dirección de tecnología de la Universidad El Bosque para empezar el desarrollo del proyecto.

- **Semana 6-7-8-9-10 (15/05/2023 - 16/06/2023):** Implementación de las herramientas y tecnologías seleccionadas. Configuración de la infraestructura de la solución virtualizada, desarrollo del Pipeline y configuración del despliegue de infraestructura. Integración de las herramientas de automatización en los procesos de despliegue de infraestructura tecnológica.
- **Semana 11-12-13-14 (19/06/2023 - 07/07/2023):** Configuración de las pruebas unitarias y de usuario. Integración de las herramientas de pruebas con la solución de automatización. Desarrollo de scripts de pruebas usando Java y el framework de TestNG.

Fase 3: Pruebas y validación

En esta etapa, se ejecutaron pruebas unitarias y pruebas de usuario en la solución automatizada. La metodología se basó en la ejecución de pruebas planificadas y la identificación de errores. Se realizó una corrección continua de problemas detectados para garantizar la calidad de la solución. A continuación se relacionan las actividades llevadas a cabo durante esta fase:

- **Semana 15 (10/07/2023 - 14/07/2023):** Ejecución de pruebas unitarias y pruebas de usuario en la solución automatizada. Identificación y corrección de errores.
- **Semana 16 (17/07/2023 - 21/07/2023):** Ejecución de pruebas de aceptación con usuarios reales. Identificación y corrección de errores. Aprobación por parte de la dirección de tecnología de la Universidad El Bosque de la solución automatizada y de las pruebas de validación.

Fase 4: Implementación y entrega

Finalmente, se llevaron a cabo actividades de capacitación para los usuarios finales de la solución automatizada, asegurando su correcto uso. Se procedió a la implementación de la solución en producción y su entrega a los usuarios finales. La metodología se enfocó en la transición de la solución al entorno de producción de manera controlada y en la formación de los usuarios. A continuación se relacionan las actividades llevadas a cabo durante esta fase:

- **Semana 17 (24/07/2023 - 28/07/2023):** Capacitación para los usuarios finales de la solución automatizada.
- **Semana 18 (31/07/2023 - 04/08/2023):** Implementación de la solución en producción y entrega a los usuarios finales.

Selección de componentes

Esta sección del proyecto se enfoca en uno de los aspectos fundamentales de este proyecto: la selección de las herramientas y componentes clave que impulsarán la automatización de procesos en el ciclo de vida del software. En un entorno DevOps, la elección adecuada de estas herramientas es esencial para garantizar la eficiencia, calidad y confiabilidad de la infraestructura y el software desplegado. En particular, se abordará la selección de cuatro tipos de componentes esenciales:

1. Version Control System (Sistema de control de versiones)

Esta herramienta desempeña un papel fundamental en la gestión de versiones y el control de cambios en el código fuente y otros activos de software. La elección de un sistema de control de versiones adecuado impactará directamente en la colaboración del equipo de desarrollo y la trazabilidad de las modificaciones. A continuación, se presenta una tabla comparativa que destaca los aspectos más significativos de estas herramientas (véase Figura 4), abarcando los cuatro sistemas de control de versiones líderes en el mercado.

Plataforma	Fabricante	Precio	Open Source	Escalabilidad	Seguridad
AWS CodeCommit	Amazon Web Services	Gratis para 5 usuarios, $1 por usuario/mes después	No	Escalabilidad automática gracias a AWS Infrastructure	AWS cuenta con políticas de seguridad y auditorías regulares
Bitbucket	Atlassian	Gratis para 5 usuarios, $2 por usuario/mes después	No	Escalabilidad horizontal mediante clústeres y otros servicios de Atlassian	Almacenamiento seguro y protección mediante SSH y SSL
GitHub	Microsoft	Gratis para usuarios públicos y proyectos Open Source, planes desde $4 por usuario/mes	Parcialmente (solo la versión del servidor)	Escalabilidad horizontal mediante servidores distribuidos	Cuenta con funciones avanzadas de autenticación y control de acceso
GitLab	GitLab Inc.	Gratis para usuarios ilimitados y repositorios privados, planes desde $4 por usuario/mes	Sí	Escalabilidad horizontal mediante clústeres y balanceadores de carga	Cuenta con opciones avanzadas de autenticación y autorización, incluyendo autenticación de dos factores y permisos granulares

Figura 4. Sistemas de control de versiones líderes en el mercado.

En conjunto con el cliente (Dirección de Tecnología de la Universidad El Bosque) se decidió que el sistema para manejar el control de versiones sería AWS CodeCommit, esto debido a que el servicio de AWS CodeCommit se ajusta a las necesidades del proyecto y trabaja en sintonía con los ambientes de AWS de la universidad.

Lo anterior, sumado a las ventajas que ofrece la herramienta. Entre ellas se destacan la integración con otros servicios de AWS, la escalabilidad y disponibilidad, la seguridad de los datos del repositorio, la facilidad de colaboración entre los miembros del equipo, la flexibilidad de protocolos de acceso y la capacidad de automatizar el proceso de entrega

continua (CI/CD). Con CodeCommit, los equipos de desarrollo pueden trabajar de forma más eficiente y segura en un mismo repositorio, lo que facilita la gestión y el control del código fuente.

2. Infraestructura como código (IaC)

La automatización de la infraestructura es esencial para la agilidad y consistencia en la implementación de entornos de desarrollo, prueba y producción. La selección de una herramienta de IaC permitirá definir y gestionar la infraestructura de manera programática. A continuación, se presenta una tabla comparativa que destaca los aspectos más significativos de estas herramientas (véase Figura 5), abarcando las dos herramientas de infraestructura como código líderes en el mercado.

Plataforma	Fabricante	Precio	Open Source	Escalabilidad	Seguridad
AWS CloudFormation	Amazon Web Services	Gratis	No	Escalable	Seguridad en la nube de AWS
Terraform	HashiCorp	Gratis	Si	Escalable	Gestión de identidad y acceso

Figura 5. Herramientas de infraestructura como código líderes en el mercado.

En conjunto con el cliente (Dirección de Tecnología de la Universidad El Bosque) se decidió que el sistema para manejar la infraestructura cómo código sería AWS CloudFormation, esto debido a que AWS CloudFormation se ajusta a las necesidades del proyecto y trabaja en sintonía con los ambientes de AWS de la universidad.

Las ventajas principales de AWS CloudFormation son su facilidad de uso y la capacidad de gestionar recursos en AWS de manera consistente y predecible. Al crear una plantilla de CloudFormation, es posible definir todo el entorno necesario para una aplicación, incluyendo redes, instancias EC2, grupos de seguridad y otros recursos, todo ello de forma automatizada y con una gran capacidad de personalización. Además, CloudFormation permite la actualización de los recursos ya creados, garantizando la continuidad del servicio y evitando errores humanos.

3. Herramienta de flujos de trabajo

La orquestación y automatización de flujos de trabajo es crucial en DevOps para coordinar tareas y procesos en todo el ciclo de vida del software. La elección de una herramienta de flujos de trabajo adecuada facilitará la integración de las diferentes etapas del proyecto. A continuación, se presenta una tabla comparativa que destaca los aspectos más significativos de estas herramientas (véase Figura 6), abarcando las dos herramientas de flujos de trabajo líderes en el mercado.

Plataforma	Fabricante	Precio	Open Source	Escalabilidad	Seguridad
AWS CodePipeline	Amazon Web Services	Pago por uso	No	Altamente escalable	Alta seguridad en el entorno de AWS
Jenkins	Jenkins (Community)	Gratis	Si	Escalable	Depende de la configuración de seguridad del usuario

Figura 6. Herramientas de Flujos de trabajo líderes en el mercado.

En conjunto con el cliente (Dirección de Tecnología de la Universidad El Bosque) se decidió que el sistema para manejar la herramienta de flujos de trabajo sería AWS CodePipeline. Las ventajas principales de AWS CodePipeline son su facilidad de uso y la integración con otros servicios de AWS, como AWS CodeCommit y AWS CodeBuild. CodePipeline permite una entrega continua rápida y segura, lo que se traduce en una mayor eficiencia y una menor posibilidad de errores humanos. Además, al utilizar CodePipeline se pueden definir y personalizar flujos de trabajo complejos de manera sencilla, lo que permite una mayor flexibilidad y adaptación a las necesidades específicas de cada proyecto.

4. Herramienta de automatización de pruebas unitarias

La automatización de pruebas unitarias es esencial para garantizar la calidad del código y la funcionalidad de la infraestructura desplegada. La selección de una herramienta de pruebas unitarias eficaz permitirá identificar errores de manera temprana y mejorar la confiabilidad del software. A continuación, se presenta una tabla comparativa que destaca los aspectos más

significativos de estas herramientas (véase Figura 7), abarcando las cinco herramientas automatización de pruebas unitarias líderes en el mercado.

Plataforma	Fabricante	Precio	Open Source	Escalabilidad	Seguridad
Nunit	Charlie Poole, NUnit.org	Gratis	Sí	Escalable	Seguro
Mockito	Mockito.org	Gratis	Sí	Escalable	Seguro
Junit	JUnit.org	Gratis	Sí	Escalable	Seguro
PHPUnit	Sebastian Bergmann, PHPUnit.de	Gratis	Sí	Escalable	Seguro
TestNG	Cedric Beust, TestNG.org	Gratis	Sí	Escalable	Seguro

Figura 7. Herramientas automatización de pruebas unitarias líderes en el mercado.

Dado que el cliente, en este caso, la Dirección de Tecnología de la Universidad El Bosque, carecía de una herramienta de automatización de pruebas previamente establecida, se procedió a realizar el siguiente análisis. Este análisis se llevó a cabo en colaboración con el cliente con el objetivo de evaluar la opción más adecuada para la ejecución del proyecto, asegurando su compatibilidad con las diversas tecnologías empleadas en el entorno de AWS.

- Nunit es una herramienta popular para pruebas unitarias en .NET y es compatible con una variedad de lenguajes de programación. Es compatible con la mayoría de los marcos de prueba de xUnit y se integra con herramientas como Visual Studio y ReSharper.

- Mockito es una herramienta de prueba unitaria para Java que se utiliza para crear objetos simulados en las pruebas. Es fácil de aprender y usar, y proporciona una sintaxis clara y concisa para crear objetos simulados.

- Junit es una herramienta de prueba unitaria popular para Java y es compatible con una variedad de lenguajes de programación. Es fácil de usar y proporciona una amplia gama de funciones, como aserciones, pruebas paramétricas y pruebas de excepciones.

- PHPUnit es una herramienta de prueba unitaria para PHP que se utiliza para probar código escrito en el lenguaje de programación PHP. Es fácil de usar y proporciona una amplia gama de funciones, como aserciones, pruebas paramétricas y pruebas de excepciones.

- TestNG es una herramienta de prueba de software para Java que se utiliza para pruebas unitarias y de integración. Es compatible con una variedad de marcos de prueba de xUnit y proporciona una amplia gama de funciones, como pruebas paramétricas, pruebas de dependencia y pruebas de grupos.

En conjunto con el cliente (Dirección de Tecnología de la Universidad El Bosque) se decidió que la herramienta para automatización de pruebas unitarias sería TestNG. Esta decisión se justifica por su sólida compatibilidad con Java y AWS, su flexibilidad con una amplia gama de funcionalidades que incluyen pruebas paramétricas y de dependencia, su capacidad para abordar tanto pruebas unitarias como de integración, su integración con entornos de desarrollo y servidores de Integración Continua, y su respaldo de una activa comunidad de usuarios y abundantes recursos en línea. Estas cualidades hacen de TestNG la elección ideal para garantizar la calidad y confiabilidad de la infraestructura tecnológica implementada en la Universidad El Bosque en el contexto del proyecto.

Diseño conceptual del proyecto

El diagrama de diseño conceptual del proyecto presenta una representación visual clara y concisa del flujo de trabajo central. El código de la aplicación sirve como punto de partida para todo el proceso. A medida que el proceso se desarrolla, el código de la aplicación es sometido a un riguroso conjunto de pasos de automatización, que incluyen la automatización del proceso de despliegue de infraestructura tecnológica, pruebas unitarias y pruebas de usuario en arquitecturas de red virtualizadas.

Estos procesos se ejecutan de manera eficiente y efectiva, gracias a la integración de las herramientas y tecnologías seleccionadas. Como resultado de este proceso, se generan salidas

cruciales para el proyecto, incluyendo la aplicación desplegada en los ambientes de seleccionados por el usuario, informes detallados de pruebas que garantizan la calidad del software, y alertas del sistema que ayudan a identificar y abordar rápidamente cualquier problema que pueda surgir durante el proceso de despliegue y configuración. Este diagrama refleja la estructura lógica y las interacciones clave que sustentan el éxito del proyecto de automatización en la Universidad El Bosque.

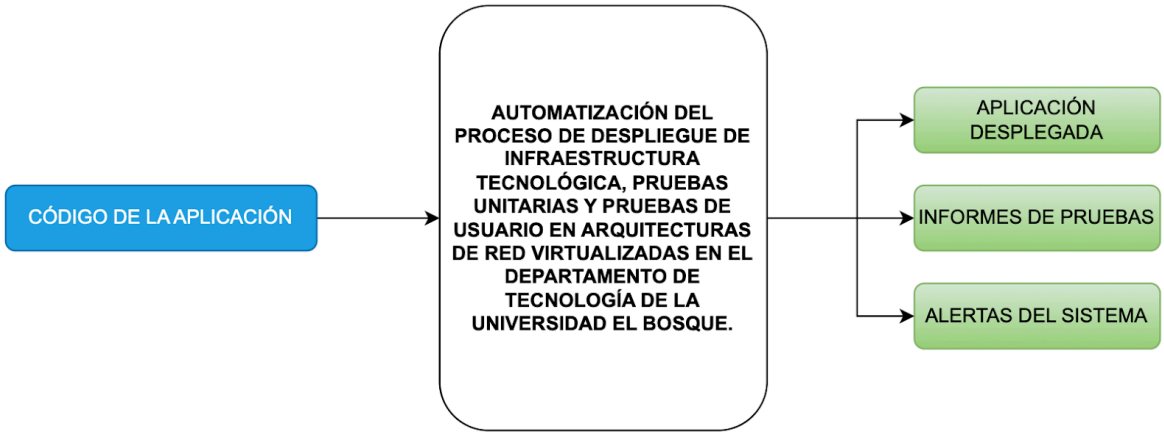


Figura 8. Diseño conceptual del proyecto.

Diseño conceptual del sistema

El diseño conceptual del sistema se organiza en tres bloques principales, cada uno cumpliendo un papel fundamental en el proceso de automatización. En el primer bloque, las pruebas unitarias, se introduce el código de la aplicación desde el repositorio AWS CodeCommit y el script de pruebas unitarias desde AWS CodeBuild. Este bloque se encarga de ejecutar las pruebas unitarias de manera automatizada. La salida deseada es una alerta de pruebas unitarias que indica si las pruebas se han completado con éxito. En caso de que ocurra alguna excepción durante las pruebas unitarias, se genera una notificación para su atención.

El segundo bloque se centra en el despliegue de infraestructura, donde el script de despliegue interactúa con la plantilla de AWS CloudFormation y la plataforma de virtualización. Este proceso permite la implementación eficiente de la infraestructura tecnológica necesaria para la aplicación. Las salidas esperadas son la aplicación desplegada y una alerta de despliegue, que informa sobre el estado de la implementación. En caso de que se presente alguna

excepción durante el despliegue de infraestructura, se activa una notificación para su gestión inmediata.

El tercer bloque aborda las pruebas de usuario, donde se involucran el explorador, el sistema operativo y el script de pruebas de usuario en la herramienta de automatización de pruebas TestNG. Este conjunto de acciones automatizadas se enfoca en evaluar la aplicación desde la perspectiva del usuario final. Las salidas comprenden informes detallados de pruebas que documentan el funcionamiento de la aplicación y alertas de pruebas de usuario. En caso de que se detecten problemas o excepciones durante las pruebas de usuario, se generan notificaciones para su pronta resolución. En conjunto, estos tres bloques forman un sistema integral de automatización que garantiza la calidad y la eficiencia en la entrega de la aplicación en el entorno de producción.

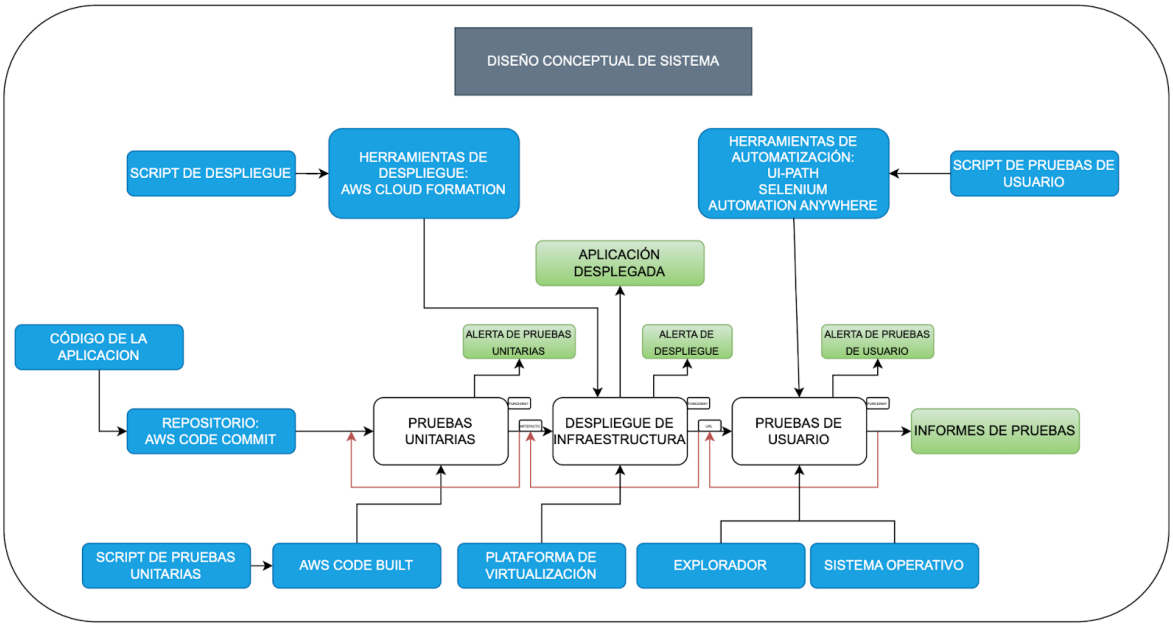


Figura 9. Diseño conceptual del sistema.

Diseño conceptual del manual de usuario

El diseño conceptual del manual de usuario tiene como objetivo proporcionar una guía completa y accesible para los usuarios del sistema, detallando paso a paso cómo utilizar sus principales funciones. En primer lugar, se explica cómo ejecutar scripts de pruebas unitarias, brindando instrucciones claras sobre cómo cargar y ejecutar los scripts desde el repositorio

AWS CodeCommit, y cómo interpretar las alertas y resultados generados durante este proceso. Además, se proporcionan detalles sobre cómo ejecutar scripts de despliegues, destacando la importancia de la interacción con la plantilla de AWS CloudFormation y cómo gestionar las alertas de despliegue.

La siguiente sección del manual se enfoca en cómo realizar un seguimiento de los informes de pruebas y despliegues generados por el sistema. Se explica cómo acceder a estos informes y cómo interpretar los datos proporcionados en ellos. Asimismo, se instruye a los usuarios sobre cómo modificar los informes de pruebas y despliegues según sus necesidades específicas, permitiendo una mayor flexibilidad en la documentación de los resultados.

El manual también incluye una sección dedicada a la activación del trigger de ejecución de los pipelines del sistema, donde se detalla el proceso para activar y gestionar estos procesos de automatización. Esto permite a los usuarios aprovechar al máximo las capacidades del sistema y garantizar un flujo de trabajo más eficiente.

Finalmente, se proporcionan pautas sobre cómo replicar el sistema para otro código, lo que facilita la expansión y la aplicación de la automatización a otros proyectos o aplicaciones similares. En conjunto, el manual de usuario se convierte en una herramienta esencial para maximizar la eficiencia y la efectividad de la automatización en el proceso de desarrollo y despliegue de software.

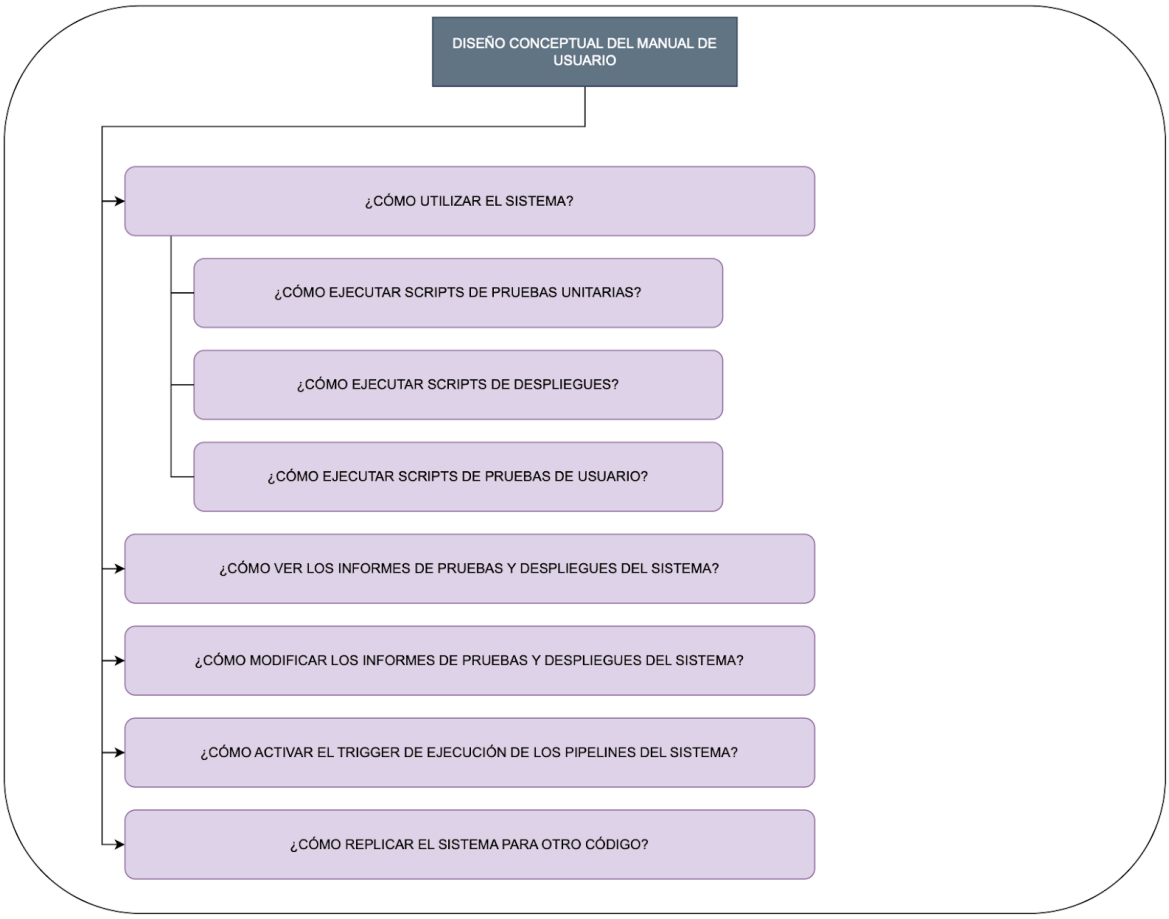


Figura 10. Diseño conceptual del manual de usuario.

Método de pruebas

Se utilizó el enfoque de pruebas de integración continua. Este enfoque implica la realización de pruebas de manera continua a lo largo del proceso de desarrollo y automatización, lo que permite detectar y corregir errores de manera temprana y garantiza que el producto final cumpla con los requisitos y expectativas de los usuarios. También se utilizará el método de pruebas de caja negra, que se centra en la evaluación de la funcionalidad de la solución sin tener en cuenta su estructura interna o código fuente. Este método es ideal para probar soluciones automatizadas, ya que se enfoca en la entrada y salida del sistema, permitiendo probar su comportamiento como un todo.

Plan de pruebas

Pruebas unitarias: son pruebas que se enfocan en comprobar el correcto funcionamiento de cada componente del sistema de manera individual. En este caso, se podrían crear casos de

prueba para los diferentes scripts de automatización de despliegue de infraestructura y pruebas unitarias.

Pruebas de integración: estas pruebas se enfocan en comprobar la interacción entre los diferentes componentes del sistema. Se podrían crear casos de prueba para la integración de las herramientas de automatización en los procesos de despliegue de infraestructura tecnológica, así como también para la integración de las herramientas de pruebas con la solución de automatización.

Pruebas de aceptación de usuario: son pruebas que se realizan con usuarios finales del sistema para evaluar la usabilidad y la funcionalidad del mismo. En este caso, se podrían crear casos de prueba para las pruebas de usuario en la solución automatizada y para las pruebas de aceptación con usuarios reales.

Para llevar a cabo el plan de pruebas se siguieron los siguientes pasos:

1. Identificar los requisitos de la solución: Basado en los requerimientos del proyecto, se establecieron los criterios de aceptación y validación de la solución.

2. Identificar los casos de prueba: Una vez identificados los requisitos de la solución, se deben identificar los casos de prueba necesarios para validar cada uno de ellos.

3. Crear los casos de prueba: En este paso, se deben crear los casos de prueba y documentarlos adecuadamente.

4. Ejecutar los casos de prueba: Después de crear los casos de prueba, es necesario ejecutarlos y registrar los resultados.

5. Identificar y reportar errores: Si se detectan errores durante la ejecución de los casos de prueba, se deben documentar y reportar adecuadamente para que el equipo de desarrollo pueda corregirlos.

6. Verificar la corrección de errores: Una vez que se han corregido los errores, se deben volver a ejecutar los casos de prueba para verificar que se hayan corregido satisfactoriamente.

7. Verificar la funcionalidad de la solución: Finalmente, se debe verificar que la solución cumpla con los requisitos y criterios de aceptación y validación establecidos en la etapa de planificación.

Se deben realizar pruebas de integración para comprobar que todas las herramientas y tecnologías seleccionadas trabajen correctamente en conjunto. También es recomendable realizar pruebas de rendimiento para asegurarse de que la solución sea escalable y pueda manejar una carga de trabajo mayor a medida que se agreguen más usuarios.

4.3 Descripción Técnica del Producto

Arquitectura de solución proyecto: ingeniería del proceso

Se trabajó sobre el mismo modelo de infraestructura con el cual cuenta la Dirección de Tecnología de la Universidad El Bosque, teniendo en AWS 3 cuentas, DEV, QA y PRD. Se realizó la solicitud y creación de una cuenta adicional llamada CODE en la cual estarán almacenados los siguientes recursos de infraestructura:

- Repositorio de código fuente de infraestructura en AWS CodeCommit.
- Repositorio de código fuente de microservicio en AWS CodeCommit.
- Pipeline en AWS CodePipeline que orquesta los procesos desde la modificación del código del microservicio hasta la compilación del artefacto.
- Compilador del artefacto incluyendo realización de pruebas unitarias en AWS CodeBuild.

El repositorio de código fuente del microservicio va a tener 3 ramas (DEV, QA y PROD) cómo se muestra en la Figura 11:

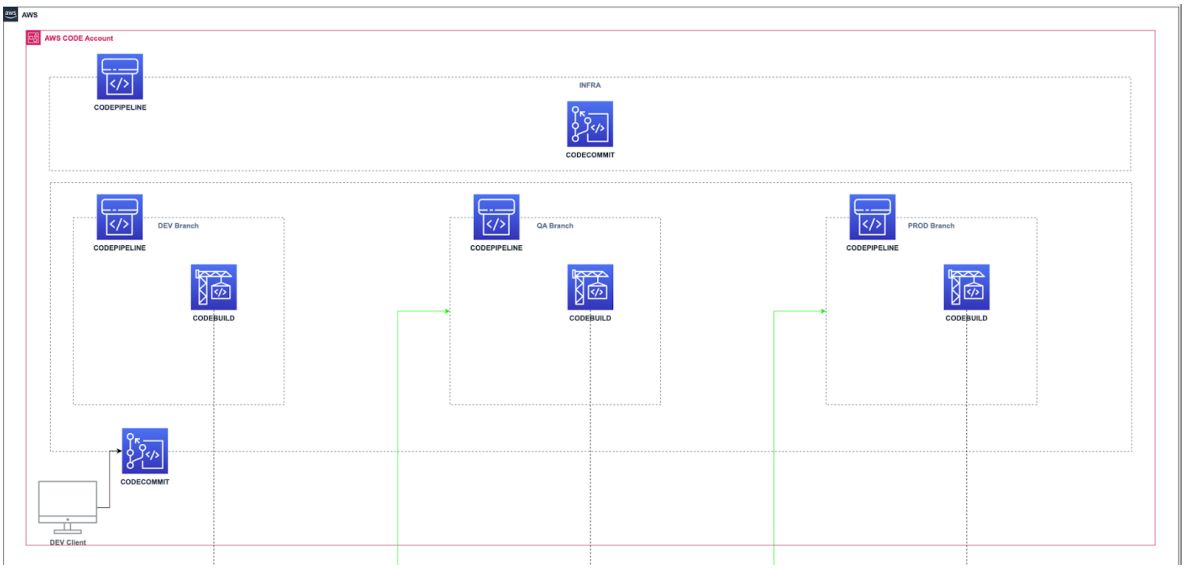


Figura 11.Arquitectura de la nueva cuenta CODE con el repositorio de CodeCommit y las ramas de desarrollo, pruebas y producción.

Dentro de la cuenta CODE como se observa en la Figura 12, cada rama está relacionada con un Pipeline, este Pipeline va a tener la tarea de generar el artefacto en AWS CodeBuild, esta estructura va a ser la misma para cada una de las tres ramas antes mencionadas. Cabe destacar que esta disposición se replica de manera consistente en las tres ramas previamente mencionadas, lo que garantiza un enfoque estandarizado y coherente en todo el proceso de desarrollo y despliegue.

Los Pipelines desempeñan un papel crucial en la generación de artefactos e imágenes en AWS CodeBuild. Este enfoque modular y altamente automatizado garantiza que cada rama, correspondiente a un entorno específico (como desarrollo, pruebas y producción), cuente con su propio Pipeline dedicado. Estos Pipelines están diseñados para realizar una serie de tareas esenciales, incluyendo la construcción y prueba de la infraestructura tecnológica de acuerdo con las especificaciones y requisitos de cada rama.

La separación en ramas y la correspondiente asignación de Pipelines garantizan una gestión precisa y aislada de cada entorno. Además, esta estructura brinda flexibilidad para realizar modificaciones y mejoras en un entorno sin afectar a los otros. La sincronización entre las diferentes ramas y Pipelines se lleva a cabo mediante una cuidadosa coordinación y control, lo que permite mantener la integridad del sistema en todo momento.

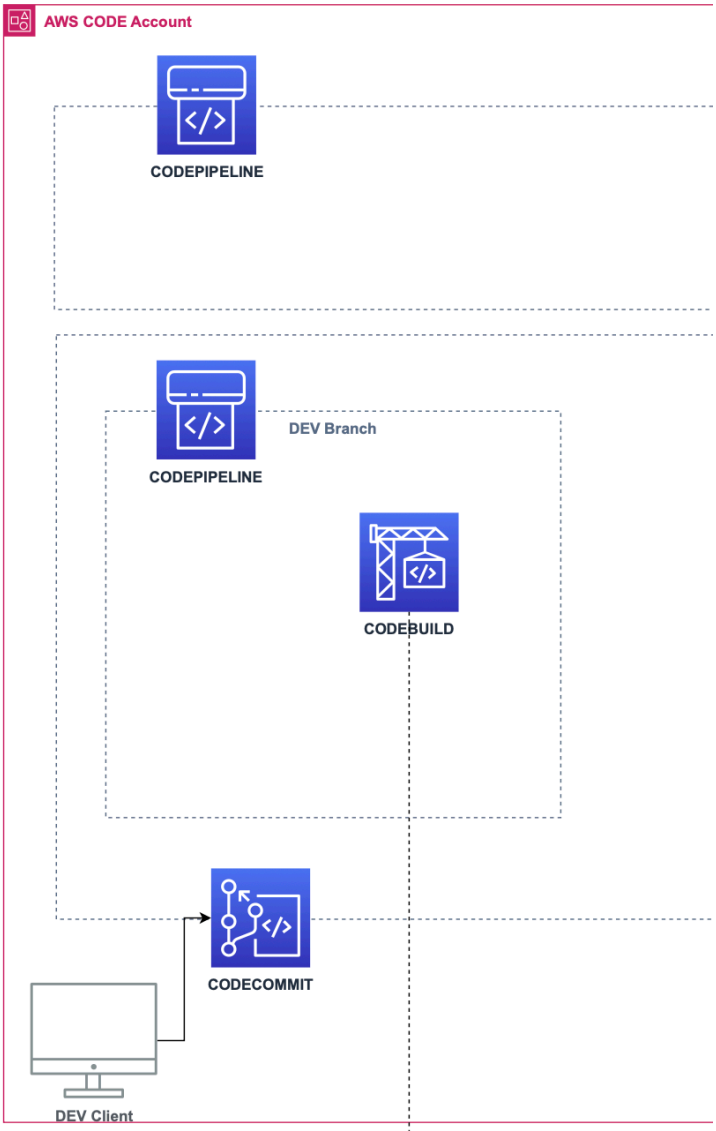


Figura 12. AWS Cuenta CODE en la rama de desarrollo.

De esta manera, el flujo de trabajo sería el siguiente:

1. El desarrollador hace PUSH al CodeCommit de la cuenta de AWS CODE Account en la rama de DEV.
2. Utilizando CodePipeline se hace el Build de la imagen o artefacto.
3. Teniendo el artefacto se hace un Cross Account Depoyment para cambiar de la cuenta de AWS CODE Account a la cuenta de AWS DEV Account
4. La cuenta de AWS DEV Account va a almacenar el artefacto en ECR.

5. La cuenta de AWS DEV Account va a tener una pipeline que va a estar encargado de hacer un despliegue del microservicio en CodeDeploy.
6. AWS CodeDeploy va a ir a Fargate y publicará el artefacto en ECS.
7. Una vez el artefacto esté publicado en la cuenta de AWS DEV Account, se va a utilizar CloudFormation y una máquina EC2 que va a ejecutar las pruebas utilizando TestNG.
8. TestNG va a estar encargado de hacer las pruebas del Front-end y del Back-end.
9. TestNG va a entregar un informe de las pruebas y utilizando otro CodePipline va a quedar en estado Ready para la cuenta de QA y se repetiría el proceso en QA y Producción.
10. Por último, se va a tener una cuenta de CodeCommit transversal a todos los servicios, este va a funcionar cómo un repositorio de infraestructura donde estaría el versionamiento de los Pipelines para hacer los despliegues de infraestructura hacía Fargate y hacía ECS, este repositorio también va a almacenar la imagen de EC2 con su respectivo código para lanzar las pruebas

En conjunto, esta arquitectura permite un flujo de trabajo continuo, desde la construcción y el almacenamiento del artefacto hasta las pruebas y el despliegue en diferentes entornos, todo ello de manera altamente automatizada y eficiente en la nube de AWS. Cómo se muestra en la Figura 13 para los ambientes de desarrollo, pruebas y producción en sus respectivas cuentas de AWS..

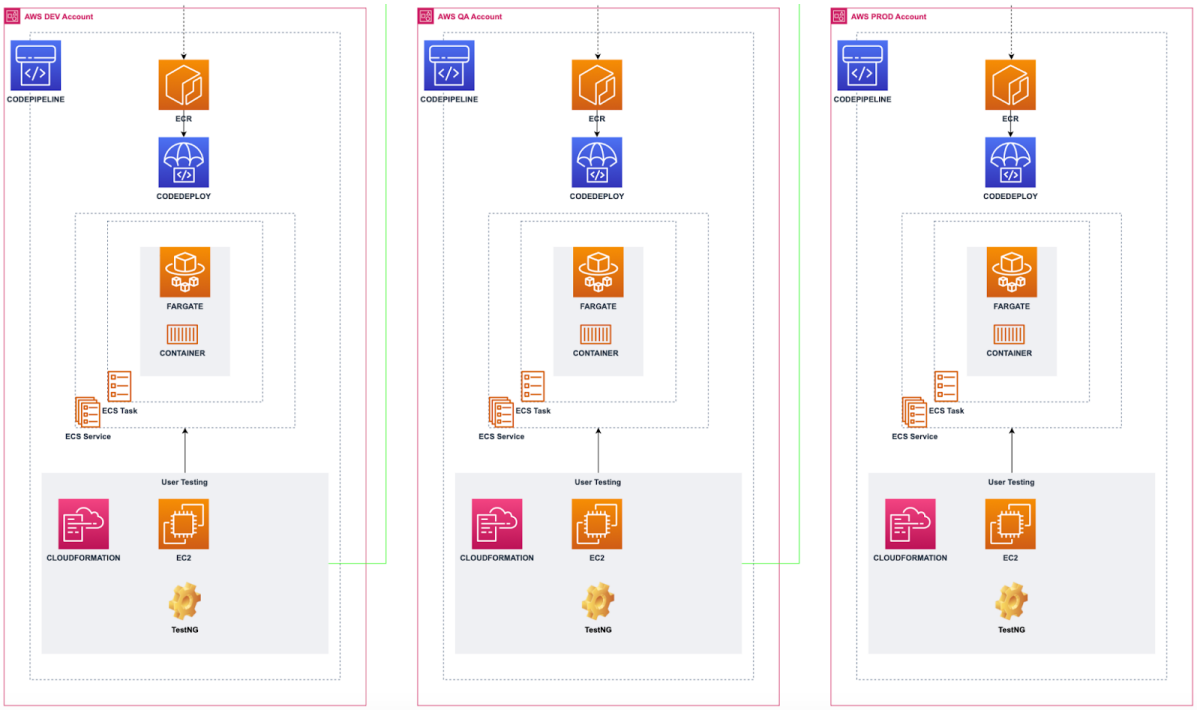


Figura 13. Arquitectura diseñada para la creación de las cuentas de AWS, esta estructura será la que se va a desarrollar en el Pipeline.

De acuerdo a los diseños anteriores, se obtiene el diagrama de arquitectura observado en la Figura 14, obteniendo de esta manera un despliegue homogéneo entre ambientes, así como recursos y rutas de trabajo estandarizadas en cada uno de los ambientes.

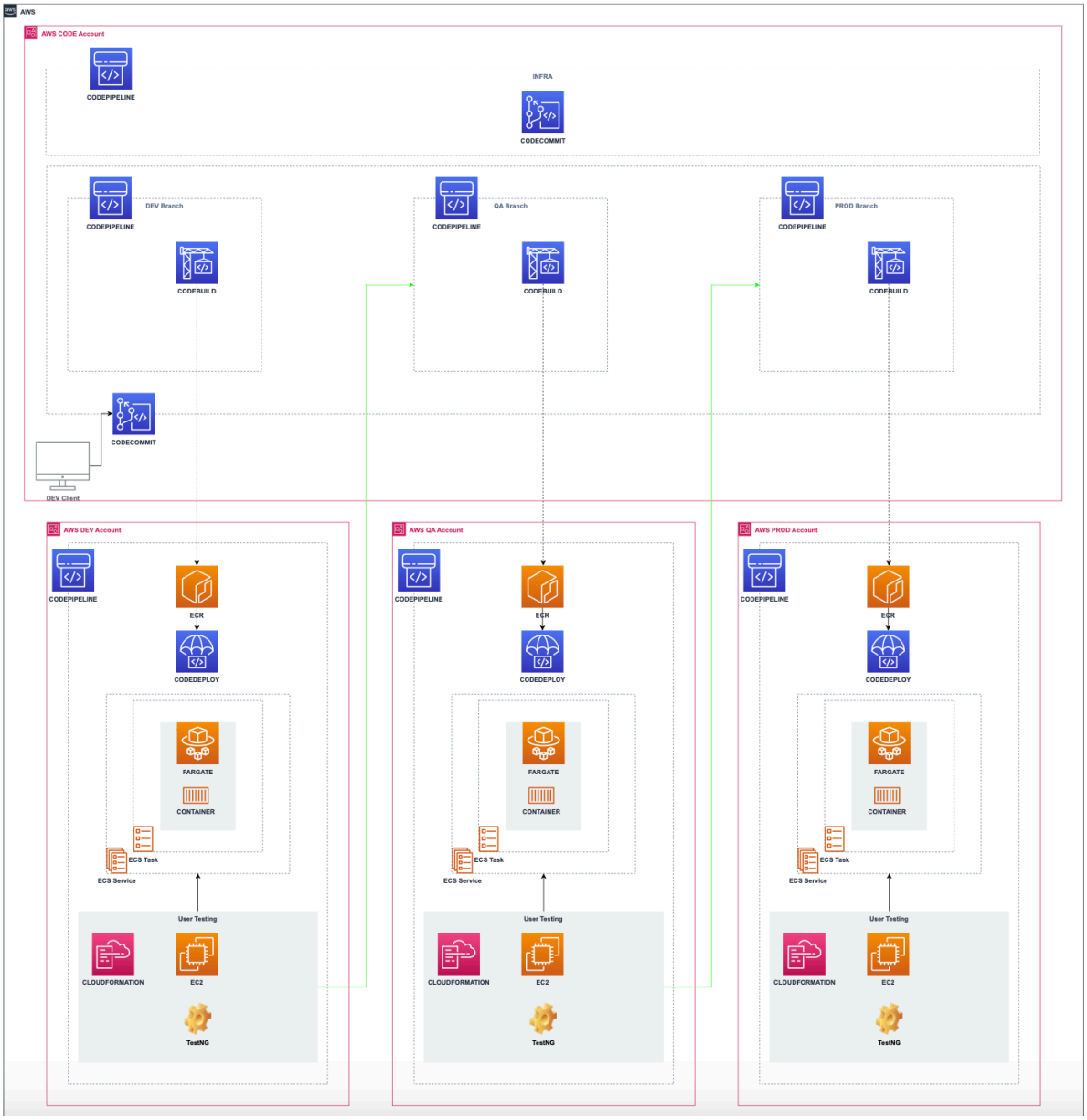


Figura 14. Diseño final de la arquitectura implementada al cliente.

5. RESULTADOS Y ANÁLISIS DE RESULTADOS

5.1 Resultados

La dirección de tecnología de la Universidad El Bosque realizó la asignación un microservicio “inscriptions” con el fin de realizar la implementación del proyecto de grado en los servicios de Backend.

5.1.1 Plantillas creadas para AWS CloudFormation

Se realizó la creación de 3 archivos que ejecutan y orquestan la creación de los recursos de infraestructura en cada uno de las cuentas de AWS (CODE, DEV, QA y PROD).

Tabla 1. Plantillas creadas para AWS CloudFormation

Plantilla	Descripción	Recursos	Descripción del recurso
main.yaml (información confidencial no anexo)	Esta plantilla de AWS CloudFormation orquesta la creación de todos los recursos de infraestructura necesarios para la ejecución de los despliegues y pruebas en cada uno de los ambientes y cuentas. La plantilla main.yaml recibe como parámetro el nombre del microservicio que se desea crear. La plantilla realiza la creación de los siguientes StackSets por medio de CloudFormation	StackSetDev	Crea los recursos de infraestructura en la cuenta DEV de acuerdo con el microservicio, mediante la utilización de la plantilla de CloudFormation Account.yaml.
		CODEStackSetDev	Crea los recursos de infraestructura en la cuenta CODE para el ambiente DEV de acuerdo con el microservicio, mediante la utilización de la plantilla de CloudFormation CODE.yaml.
		StackSetQa	Crea los recursos de infraestructura en la cuenta QA de acuerdo con el microservicio, mediante la utilización de la plantilla de CloudFormation Account.yaml.
		CODEStackSetQa	Crea los recursos de infraestructura en la cuenta CODE para el ambiente QA de acuerdo al microservicio, mediante la utilización de la plantilla de CloudFormation CODE.yaml.
		StackSetProd	Crea los recursos de infraestructura en la cuenta PROD de acuerdo con el microservicio, mediante la utilización de la plantilla de CloudFormation Account.yaml.
		CODEStackSetProd	Crea los recursos de infraestructura en la cuenta CODE para el ambiente PROD de acuerdo con el microservicio, mediante la utilización de la plantilla de CloudFormation CODE.yaml.
		Repo CodeCommit	Crea el repositorio de CodeCommit del microservicio, en la cuenta CODE.
CODE.yaml (información confidencial no anexo)	Esta plantilla realiza la creación de los recursos de infraestructura de la cuenta CODE para cada ambiente.	CodeBuildProject	Es el proyecto para compilación y pruebas unitarias del código fuente, contiene las definiciones para realizar la compilación y creación de la imagen del artefacto, así como la subida de estos al repositorio de ECR.
		CodeBuildServiceRole	Contiene los permisos y acciones necesarias en IAM para permitir el uso de los recursos de AWS desde el servicio de CodeBuild en la cuenta CODE.
		CodePipelineServiceRole	Contiene los permisos y acciones necesarias en IAM para permitir el uso de los recursos de AWS desde el servicio de CodePipeline en la cuenta CODE.

		Pipeline	Contiene la secuencia de acciones a llevar a cabo para cumplir las tareas de acuerdo con la arquitectura propuesta.
Account.yaml (información confidencial no anexo)	Esta plantilla realiza la creación de los recursos de infraestructura en las cuentas de destino (DEV, QA, PROD) y contiene los parámetros necesarios para la implementación en cada uno de los ambientes (ELB Listener, Security Group, Subnets, VPC)	CodeBuildProjectTest	Contiene las definiciones y pasos para la utilización de CodeBuild como herramienta para la realización de las pruebas de usuario mediante la utilización del archivo buildspec.yaml.
		CodeBuildServiceRole	Contiene los permisos y acciones necesarias en IAM para permitir el uso de los recursos de AWS desde el servicio de CodeBuild en las cuentas del ambiente de destino.
		CodePipelineServiceRole	Contiene los permisos y acciones necesarias en IAM para permitir el uso de los recursos de AWS desde el servicio de CodePipeline en las cuentas del ambiente de destino.
		ArtifactBucket	Bucket de S3 donde se almacenarán los artefactos y archivos generados necesarios para las pruebas de usuario y el despliegue mediante CodeDeploy.
		S3BucketPolicy	Política de IAM que contiene las definiciones de quién o qué recurso tiene acceso al bucket de S3.
		CodeDeployApp	Se especifica el nombre de la aplicación como la plataforma de ejecución para su posterior utilización en CodeDeploy.
		AppDeploymentGroup	Es el conjunto de tareas, definiciones y permisos necesarios para realizar el despliegue del microservicio mediante CodeDeploy, incluye definiciones tales como el nombre de la aplicación de CodeDeploy, el nombre del DeploymentGroup, el nombre del servicio de ECS, el cluster de ECS, el tipo de despliegue de ECS, la información relacionada al balanceador de carga ELB, el tipo de despliegue (Blue/Green deployment).
		Pipeline	Contiene las acciones y secuencias a realizar para completar el despliegue y las pruebas de usuario. Contiene las etapas de Source, Aprobación, Deploy y Test, la etapa de Source realiza la descarga del artefacto y los archivos generados desde el pipeline de la cuenta CODE almacenados en el bucket de S3, la etapa de Aprobación detiene el proceso en las cuentas QA y PROD a la espera de una aprobación manual de acuerdo a solicitud de la dirección de tecnología, la etapa de Deploy contiene las especificaciones de los archivos necesarios para realizar el despliegue a ECS mediante CodeDeploy, la etapa Test mediante CodeBuild realiza la creación de un ambiente temporal para realizar las pruebas de usuario al microservicio de acuerdo a las definiciones del archivo buildspec.yaml en la carpeta usertest.
		ServiceRepository	Repositorio de ECR donde se almacena la imagen del contenedor utilizando Docker generada en CodeBuild mediante el pipeline de la cuenta CODE.
		ECSServiceDef	Contiene las definicones para la creación del servicio de ECS, incluyendo la defindión de la tarea, el tipo de lanzamiento (FARGATE), las configuraciones de red (subredes y grupos de seguridad), definiciones del balanceador de carga.
		TargetGroup	La sección TargetGroup contiene los objetivos del ELB los cuales serán alternados de acuerdo con el despliegue Blue/Green.
		TargetGroup1	La sección TargetGroup1 contiene los objetivos del ELB los cuales serán alternados de acuerdo con el despliegue Blue/Green.
		ListenerRule	Regla del balanceador de carga ELB que realiza el apuntamiento hacia uno de los target groups definidos anteriormente de acuerdo al despliegue realizado.

5.1.2 Archivos para el despliegue del microservicio

Se realizó la creación desde cero de los siguientes archivos para el despliegue de la arquitectura necesaria donde corren los microservicios.

Tabla 2. Archivos para el despliegue del microservicio

Archivo	Descripción
buildspec.yaml (información confidencial no anexo)	El archivo proveído por la dirección de tecnología contenía valores que no se podían usar de forma genérica, se modificó para permitir la utilización del mismo archivo para cualquier microservicio. El archivo de buidspec contiene las especificaciones para que CodeBuild realice la creación del artefacto y la subida de la imagen de Docker a ECR. Los parámetros del microservicio anteriormente se ingresaban manualmente en dos ubicaciones, en el archivo de buildspec y al momento de crear el servicio de ECS. Se modificó la estructura del archivo para iterar en el archivo taskdef.json y extraer los parámetros del microservicio y evitar un reproceso. Mediante buildspec se realiza la compilación del artefacto y se sube la imagen de Docker a ECR en las cuentas de destino.
Dockerfile (ANEXO B)	El archivo Dockerfile se modificó con el fin de poder ser utilizado por cualquier microservicio, a diferencia de cómo se realizaba antes, en donde se insertaba en el archivo una URL y de esta manera no se podía reutilizar con los demás microservicios. El archivo de Dockerfile contiene las definiciones para la creacion de la imagen de Docker que será posteriormente subida a ECR y utilizada en ECS.
appspec.yaml (ANEXO C)	Contiene las definiciones del target a utilizar desde CodeDeploy al momento de realizar el despliegue a ECS.
taskdef.json (información confidencial no anexo)	Contiene las definiciones para la creación de la tarea de ECS, incluyendo los parámetros del microservicio, parámetros del contenedor, y parámetros de cómputo del microservicio.

5.1.3 Archivos de infraestructura para pruebas de usuario

Se realiza la creación de los siguientes archivos que dictan los lineamientos para la realización de pruebas de usuario en la infraestructura creada anteriormente.

Tabla 3. Archivos de infraestructura de pruebas de usuario

Archivo	Descripción
buildspec.yaml (ANEXO D)	El archivo de buildspec almacena las tareas para que CodeBuild realice la creación del ambiente temporal que realiza las pruebas de usuario, descomprime la carpeta de pruebas de usuario almacenada en el repositorio del microservicio, crea el ambiente de Docker usando Docker compose, realiza las pruebas de usuario mediante el comando mvn clean test y copia el archivo de resultados de las pruebas al bucket de S3 del artefacto.
docker-compose.yml (ANEXO E)	El archivo realiza la creación de un contenedor que cumple la función de grid de Selenium para orquestar las pruebas realizadas, genera también el contenedor con las definiciones del explorador a utilizar, en este caso Google Chrome.

pom.xml (ANEXO F)	El archivo pom contiene información del proyecto de Maven, como dependencias y plugins que se utilizan para la ejecución de las pruebas de usuario.
testng.xml (ANEXO G)	Describe la suite de pruebas de TestNG, indica las funciones del archivo FirsTest.java a las que se les realizará pruebas, como también los parámetros del tipo de explorador a utilizar.

5.1.4 Archivos para ejecución de pruebas de usuario

Se realiza la creación de los siguientes archivos que dictan los lineamientos de las pruebas de usuario.

Tabla 4. Archivos para ejecución de pruebas de usuario

Archivo	Descripción
BaseTest.java (ANEXO H)	La clase BaseTest es una clase que proporciona funcionalidad común para todas las clases de prueba. La clase BaseTest incluye métodos para iniciar y finalizar el navegador, navegar a sitios web y ubicación del remote driver.
CapabilityFactory.java (ANEXO I)	La clase CapabilityFactory es una clase que proporciona una forma de crear capacidades para el navegador. Las capacidades son propiedades que configuran el navegador, como el tipo de navegador, la versión del navegador y el sistema operativo.
FirstTest.java (información confidencial no anexo)	La clase FirstTest inicia el navegador, navega a Google y realiza las pruebas de usuario del microservicio.
OptionsManager.java (ANEXO J)	La clase OptionsManager es una clase que proporciona una forma de gestionar las opciones del navegador. Las opciones del navegador son configuraciones que afectan al comportamiento del navegador, incluye configuraciones para maximizar el explorador, ignorar errores de certificado y deshabilitar avisos.

5.1.5 Despliegue en CloudFormation

- Mediante el siguiente comando se realiza la creación de toda la infraestructura necesaria para la ejecución y puesta en marcha de un microservicio:

```
aws cloudformation create-stack --stack-name ub-stack-inscriptions-backend --template-url \
https://ub-stackbucket-codeaccount.s3.amazonaws.com/main.yaml --capabilities CAPABILITY_NAMED_IAM \
--parameters ParameterKey=ServiceName,ParameterValue=ub-serv-inscription-backend \
ParameterKey=DEVPriority,ParameterValue=49 \
ParameterKey=QAPriority,ParameterValue=30 \
ParameterKey=PRODPriority,ParameterValue=30
```

Figura 15. Comando CloudFormation para la creación de la infraestructura

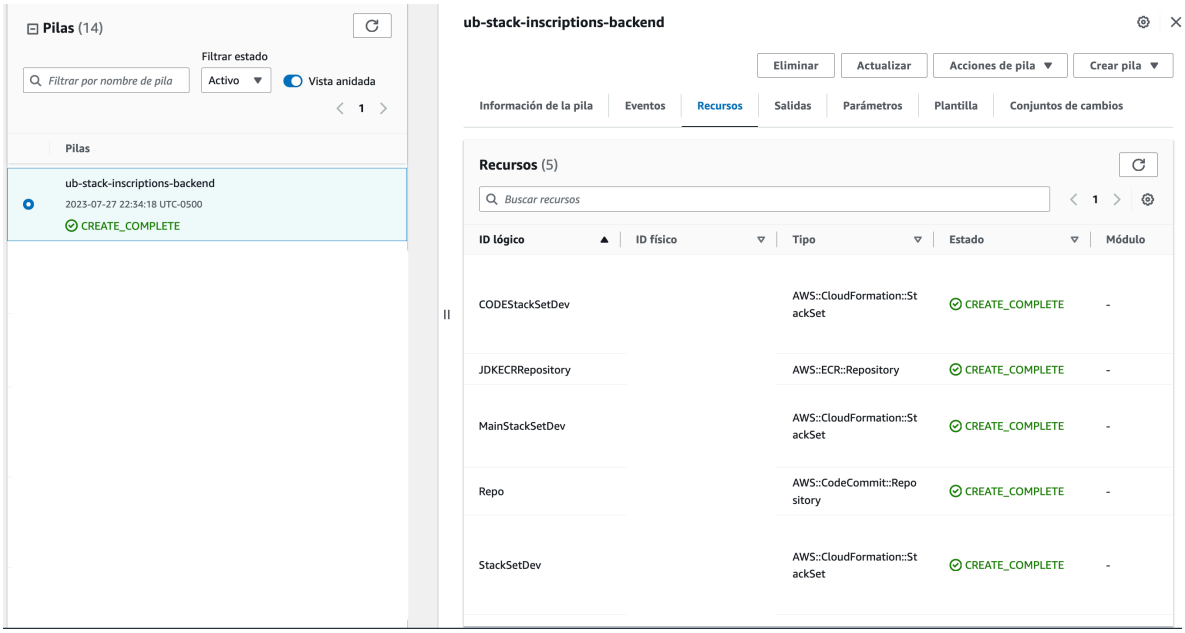


Figura 16. Creación de recursos de infraestructura en la cuenta CODE

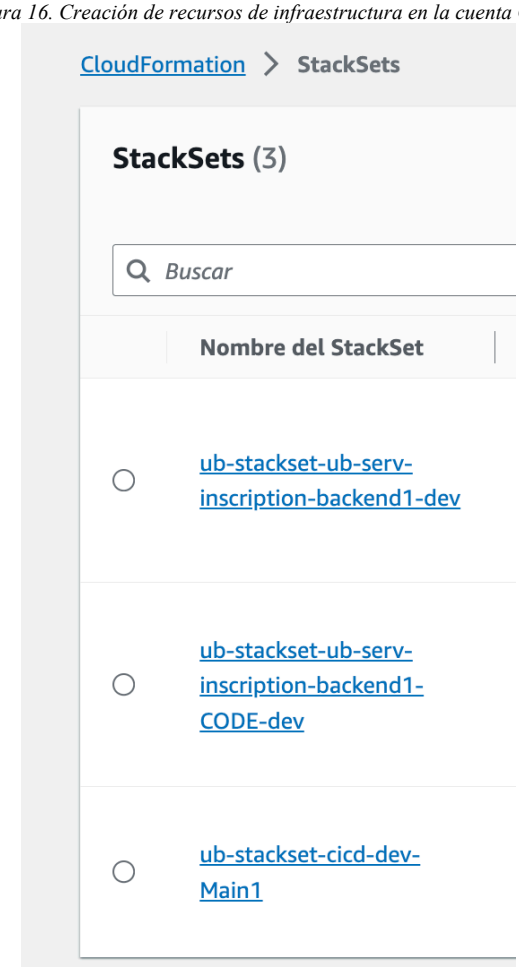


Figura 17. Creación de StackSets desde cuenta CODE hacia cuenta DEV

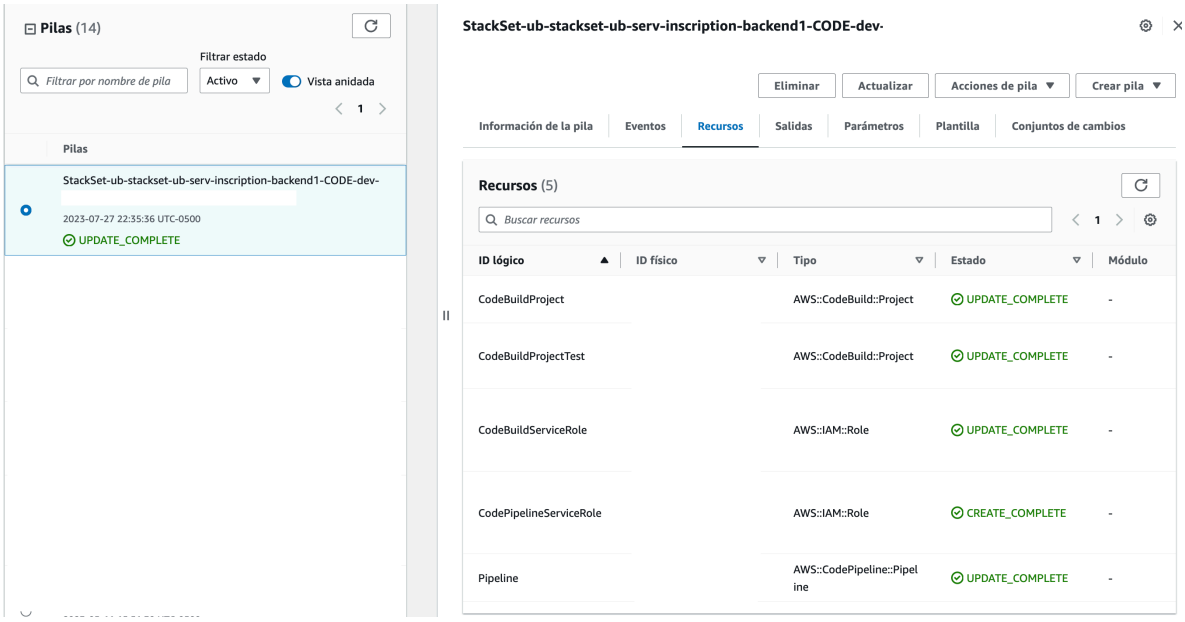


Figura 18. Creación de recursos de infraestructura en StackSet cuenta CODE

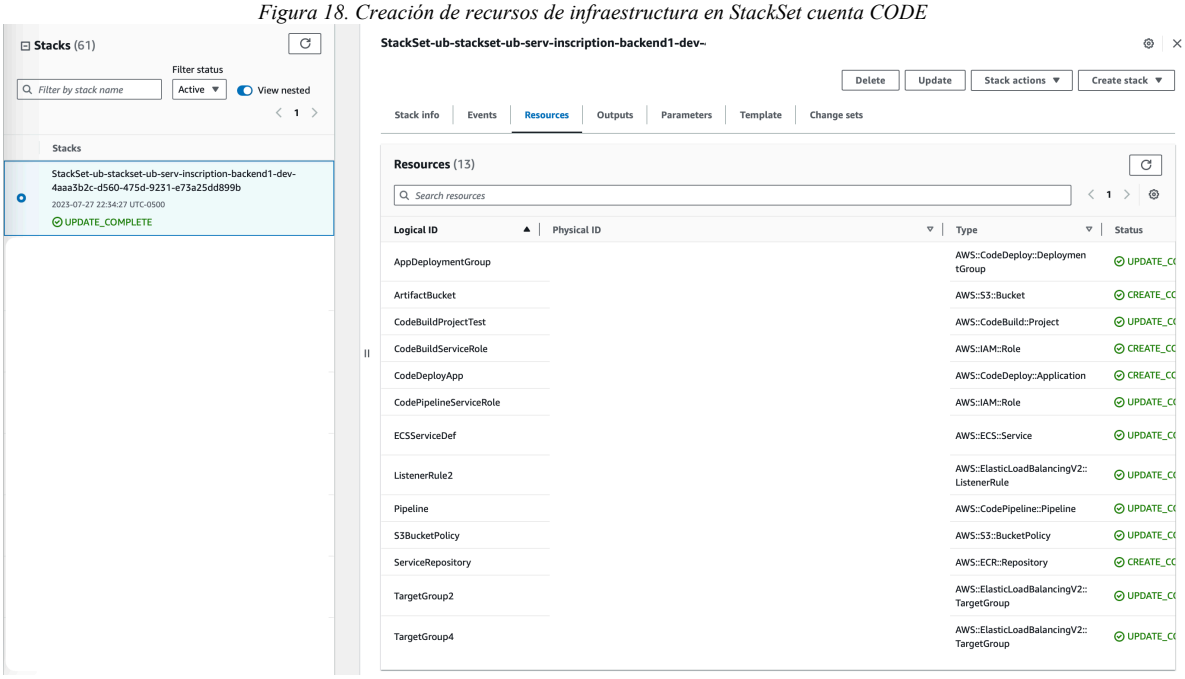


Figura 19. Creación de recursos de infraestructura en StackSet cuenta DEV

5.1.6 Activación del flujo de trabajo de despliegue

- Dentro de la fase de pruebas y validación se realizan modificaciones al código. Estas modificaciones se deben realizar en el repositorio del microservicio en la rama DEV, al ejecutar el comando git push, CodePipeline reconoce el cambio y desencadena la

ejecución de los procesos desde la cuenta CODE hasta el proceso de pruebas de usuario en CodeBuild en la cuenta DEV.

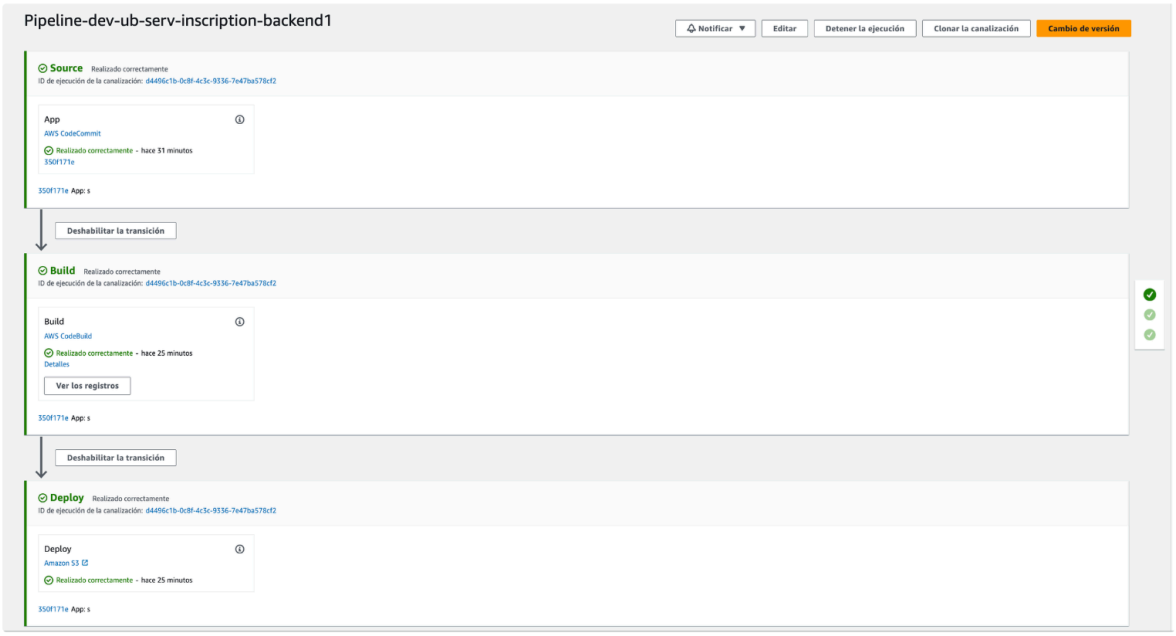


Figura 20. Ejecución de las etapas del pipeline en la cuenta CODE

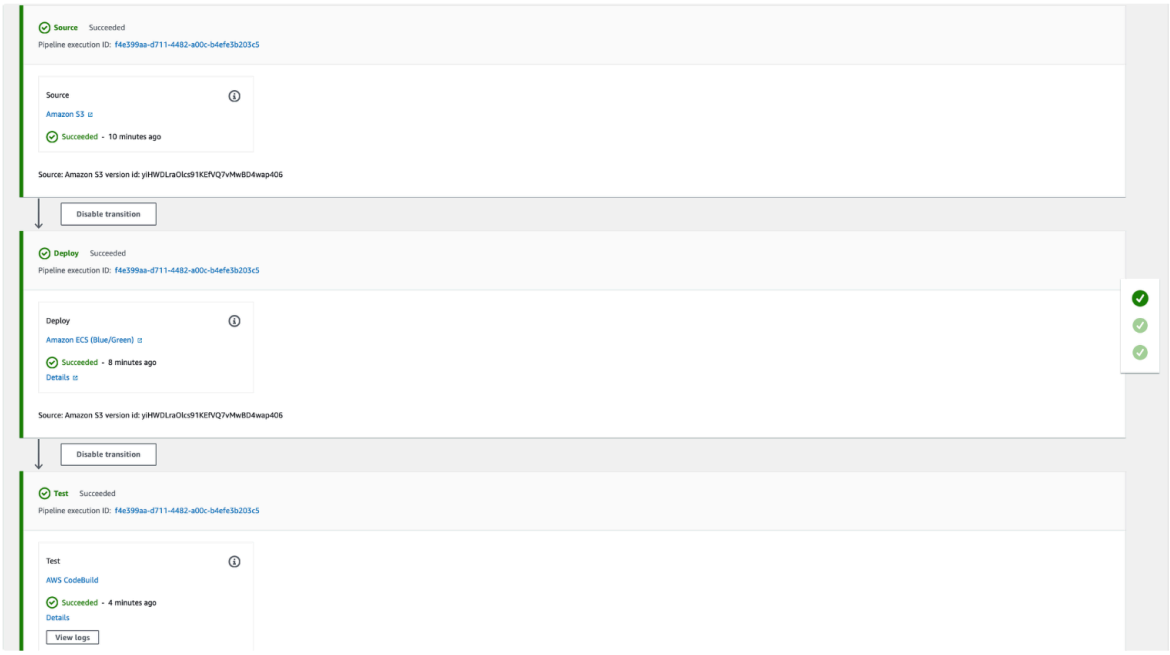


Figura 21. Ejecución de las etapas del pipeline en la cuenta DEV

Los cambios en las ramas de QA y PROD se desencadenan cuando se hace Merge desde alguna de las ramas anteriores en la cadena, desde la construcción del artefacto hasta la realización de las pruebas de usuario.

La cuenta PROD y la etapa QA tienen una etapa adicional de aprobación manual y la cuenta PROD no realiza etapa de pruebas de usuario.

5.1.7 Visualización de informes

Cada microservicio tiene asociado en cada una de las cuentas, un bucket S3 llamado ArtifacBucket.

Ej. Artifact bucket del microservicio de inscriptions en la cuenta DEV.

Existe un objeto en la carpeta BuildOutpu que contiene el HTML generado después de las pruebas.

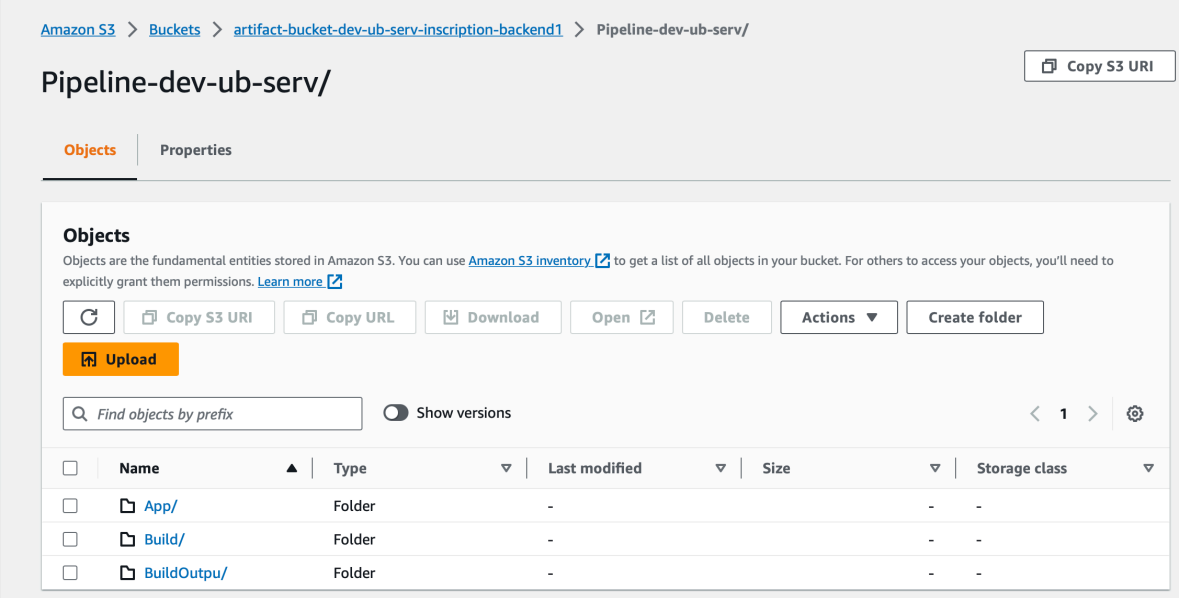


Figura 22. Bucket de S3 donde se almacenan los artefactos del microservicio

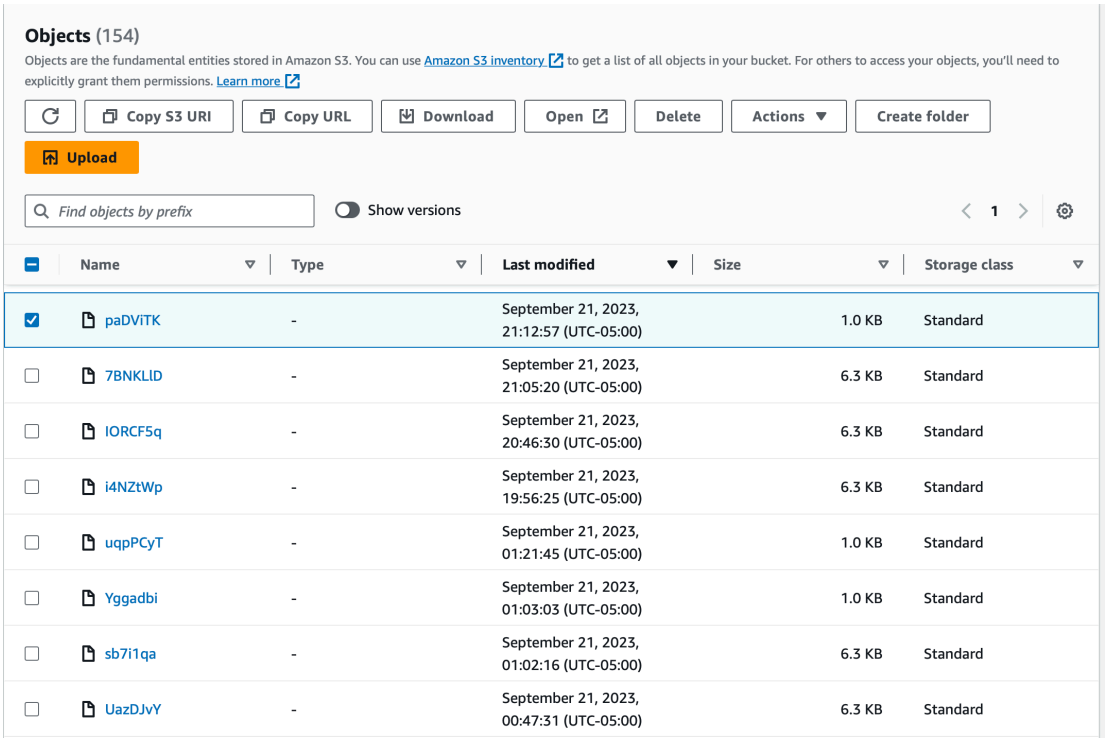
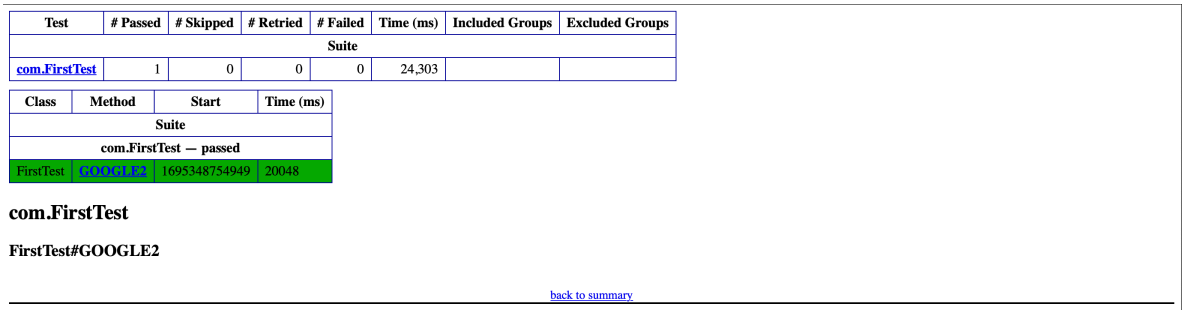


Figura 23. Carpeta que contiene los archivos de ejecución de las pruebas

Al descargar el archivo del objeto del bucket de S3, se obtiene el siguiente reporte:



Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite							
com.FirstTest	1	0	0	0	24,303		

Class	Method	Start	Time (ms)
Suite			
com.FirstTest — passed			
FirstTest	GOOGLE2	1695348754949	20048

com.FirstTest

FirstTest#GOOGLE2

back to summary

Figura 24. Reporte de pruebas en formato HTML, descargado del artifact bucket

5.2 Cumplimiento de requerimientos

Se verifican con los requerimientos establecidos anteriormente en el numeral 4.1 el cumplimiento de los mismos.

Requerimientos funcionales

1. El sistema debe ser capaz de implementar y configurar entornos virtuales en la nube de AWS de la Universidad El Bosque.

Se confirma que el sistema cumple con este requerimiento. Figuras: Figura #. Creación de StackSets desde cuenta CODE hacia cuenta DEV. Figura #Creación de recursos de infraestructura en StackSet cuenta CODE. Figura #Creación de recursos de infraestructura en StackSet cuenta DEV. (Figura #) dispara la creación automática de los recursos de infraestructura en todas las cuentas.

Por lo cual, esto, mediante cloud formation busca las plantillas y de acuerdo a esto crea toda la infraestructura donde se van a ejecutar las pruebas, las compilaciones, los despliegues y el microservicio. A comparación de antes donde el proceso de creación de infraestructura en el servicio de ECS se realizaba manual.

2. La solución debe permitir realizar pruebas unitarias de los contenedores de la infraestructura virtualizada, utilizando herramientas de automatización de pruebas unitarias.

   Se evidencia el cumplimiento de este requerimiento ya que en la etapa de Build del pipeline de la cuenta CODE de acuerdo al archivo de especificaciones "buildspec.yaml" se realiza la ejecución de pruebas unitarias mediante el comando "mvn test". Como se evidencia en la figura "Figura 20 Ejecución de las etapas del pipeline en la cuenta code."

3. El sistema debe permitir realizar un conjunto de pruebas de usuario para validar el correcto funcionamiento de la infraestructura virtualizada, utilizando herramientas de automatización de pruebas de usuario.

   Se verifica el cumplimiento de este requerimiento en la etapa de test del pipeline de las cuentas de destino. La automatización de estas pruebas se da con el uso de la herramienta TestNG , de acuerdo a los Archivos de infraestructura de pruebas de usuario (Tabla 3) y los Archivos para ejecución de pruebas de usuario (Tabla 4)
   Como se evidencia en la figura Figura 21 Ejecución de las etapas del pipeline en la cuenta DEV.

La infraestructura de ejecución de pruebas se cambió de EC2 a CodeBuild.

4. La solución debe permitir la integración con otras herramientas y servicios de AWS para realizar el despliegue automatizado en diferentes entornos (desarrollo, prueba y producción).

5. Se da cumplimiento al requerimiento ya que la plantilla despliega en todos los ambientes (desarrollo, pruebas y producción) y al utilizar herramientas únicamente de AWS, se asegura su interoperabilidad. Adicionalmente, al usar Docker, JAVA y TestNG, la solución se vuelve portable ya que Docker permite su ejecución en diferentes herramientas.

6. El sistema debe permitir la personalización de los procesos de despliegue y configuración para adaptarse a las necesidades específicas de la Dirección de Tecnología de la Universidad El Bosque.

   Las plantillas empleadas de CloudFormation (Tabla 1 Plantillas creadas para AWS CloudFormation), fueron creadas de acuerdo a las especificaciones dadas por la Dirección de tecnología, por lo cual, son modificables en caso tal de requerir añadir funcionalidades adicionales. Por lo cual, se da cumplimiento al requerimiento.

7. El sistema debe estar disponible para realizar despliegues, evitando interrupciones en el proceso de despliegue y configuración, realizando la configuración automatizada de los contenedores de la infraestructura tecnológica.

   Al momento de creación de la infraestructura no hay restricciones de horarios, y código y pruebas pueden ser ejecutados en cualquier momento y los flujos de trabajo son automáticos. (Figura 14. Diseño final de la arquitectura implementada al cliente)

8. La plataforma debe permitir la gestión de versiones de los componentes de la infraestructura tecnológica y de los scripts de automatización.

9. En la cuenta CODE, se realizó la creación de un repositorio en la herramienta CodeCommit donde se almacenan las plantillas de creación de infraestructura y su versionamiento.

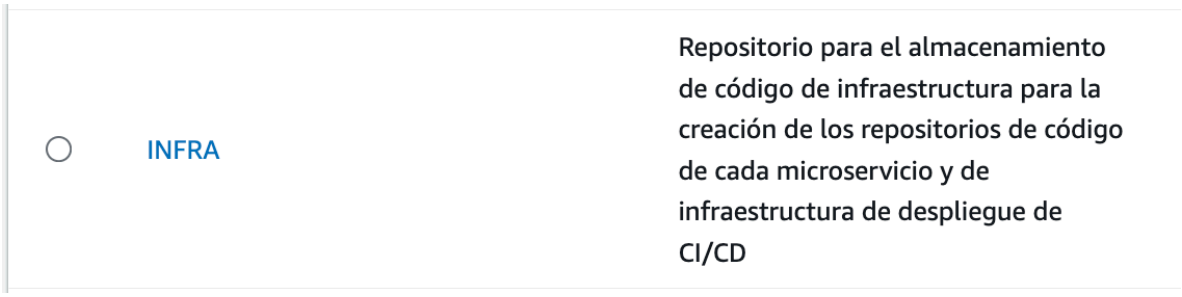


Figura 25. Repositorio de infraestructura en cuenta CODE

Requerimientos de calidad

1. El sistema debe garantizar la calidad del código y la funcionalidad de la infraestructura a través de pruebas unitarias y de usuario automatizadas.

   Mediante la etapa “Build” del pipeline de la cuenta CODE y la etapa “Test” del pipeline de la cuenta DEV, se realizan las pruebas unitarias y pruebas de usuario al microservicio generando reportes y pudiendo evidenciar fallas directamente en el pipeline. Se evidencia el cumplimiento de este requerimiento en la sección 5.1.5 Activación del flujo de trabajo de despliegue.

2. La documentación proporcionada debe asegurar la correcta implementación y configuración de la solución, garantizando su funcionamiento a largo plazo. La documentación debe incluir diseño técnico, especificación funcional y manual de operación.

3. Se da el cumplimiento de este requerimiento ya que se realizó la creación de un manual de operación (ANEXO A) , este incluye el diseño técnico y las especificaciones funcionales.

4. El sistema debe generar informes de resultados de pruebas unitarias y pruebas de usuario automatizadas para evaluar la calidad y el rendimiento del sistema.

   Se cumple este requerimiento ya se que genera el informe de pruebas y se almacena en el ArtifactBucket como se evidencia en la sección 5.1.7 Visualización de informes.

Requerimientos restrictivos

1. La implementación se debe realizar en la cuenta CODE de manera transversal con las cuentas de DEV, QA y PROD.

2. Se realiza la implementación de estas cuentas de manera transversal, ya que la cuenta CODE orquesta la creación de los recursos en las demás cuentas. Como se puede evidenciar en la Tabla 1. Plantillas creadas para AWS CloudFormation.

3. La implementación se realizará de manera escalonada, iniciando por la cuenta DEV, QA y finalmente PROD.

4. Se dio la implementación de manera escalonada ya que se ejecutó mediante fases como se evidencia la sección 4.2 de metodología del diseño.

5. La implementación se llevará a cabo, hasta donde se permita, únicamente con herramientas de la suite de AWS ya que es el proveedor contratado por la Universidad el Bosque por lo cual los lineamientos de la dirección de tecnología restringen el uso de herramientas de otras suites.

   Se da cumplimiento a este requerimiento de acuerdo a la sección 4.2 metodología del diseño, fase 4, selección de componentes.

5.3 Análisis

Los resultados obtenidos muestran una correcta implementación en la automatización de procesos relacionados con la infraestructura y el despliegue de microservicios en la arquitectura de red virtualizada. Con la ejecución de un solo comando, se logra la creación de toda la infraestructura necesaria, lo que representa una mejora considerable en términos de eficiencia y rapidez en comparación con los procesos manuales previos. En primer lugar, se asignó el microservicio "inscriptions" como el objeto de estudio para llevar a cabo la implementación de este proyecto en el ámbito de los servicios de Backend y Frontend. Esta elección de microservicio proporcionó un escenario real y aplicable para demostrar la eficacia de las soluciones propuestas.

La reducción a la ejecución de un comando y la modificación de solo dos archivos para la puesta en marcha de un microservicio demuestra una simplificación considerable en la implementación de nuevos servicios. Esto puede acelerar significativamente el desarrollo y despliegue de nuevas funcionalidades, lo que es un avance notable en términos de eficiencia operativa.

La adición de una etapa de autorización en los entornos de calidad (QA) y producción (PROD) es un paso importante en términos de seguridad y control de acceso. Esto garantiza que los cambios en estos entornos críticos sean supervisados y aprobados adecuadamente, lo que contribuye a la integridad de los sistemas.

No obstante, es importante destacar que no se implementó la entrega continua de infraestructura. Ya que la entrega continua de infraestructura no aportaba valor a la arquitectura actual de la Dirección de tecnología, por lo tanto la sección de CodePipeline de la infraestructura en la arquitectura no se implementó.

El cambio de EC2 a CodeBuild para las pruebas de usuario es otro avance positivo, ya que mejora la eficiencia y escalabilidad de las pruebas. Esto puede llevar a una identificación más rápida de problemas y errores en las aplicaciones.

En lo que respecta a la automatización de los pasos de ECS que anteriormente se realizaban manualmente, se ha logrado una simplificación significativa en la gestión de contenedores y la implementación de microservicios en la infraestructura tecnológica. Esto reduce la posibilidad de errores humanos y garantiza una mayor coherencia en los despliegues.

Una de las contribuciones más notables de este proyecto fue la creación de plantillas para AWS CloudFormation. Estas plantillas, diseñadas para orquestar la creación de recursos de infraestructura en las cuentas de AWS (CODE, DEV, QA y PROD), representan un avance importante en la automatización y estandarización de los procesos. Estas plantillas permiten la creación rápida y confiable de entornos de desarrollo, pruebas y producción con un solo comando, reduciendo así la complejidad y el tiempo necesario para configurar la infraestructura.

Además de las plantillas de CloudFormation, se desarrollaron archivos específicos para el despliegue del microservicio. Estos archivos, como el "buildspec.yaml," se crearon desde cero y se diseñaron para ser genéricos, lo que facilita su reutilización en otros microservicios. Esta reutilización es especialmente valiosa ya que reduce la necesidad de crear configuraciones personalizadas para cada servicio, lo que a su vez ahorra tiempo y reduce posibles errores.

En la fase de pruebas de usuario, se crearon archivos de infraestructura específicos que establecen los lineamientos para la realización de pruebas. Estos archivos proporcionan la estructura necesaria para llevar a cabo pruebas automatizadas de manera eficiente, garantizando la calidad y el rendimiento del microservicio. Además, se implementaron archivos específicos para la ejecución de las pruebas de usuario, lo que simplifica aún más este proceso.

La activación del flujo de trabajo de despliegue es otro aspecto crucial. La automatización de la ejecución de procesos desde la cuenta CODE hasta el proceso de pruebas de usuario en CodeBuild en la cuenta DEV representa un avance significativo en la entrega continua de software. Los cambios en las ramas de QA y PROD se desencadenan mediante merge desde

ramas anteriores en la cadena, lo que garantiza un control de cambios y la coordinación entre los equipos de desarrollo.

Finalmente, la visualización de informes proporciona una valiosa retroalimentación sobre el rendimiento y la calidad de los microservicios desplegados. Los informes generados en formato HTML y almacenados en los buckets S3 permiten una supervisión detallada de los resultados de las pruebas, lo que facilita la identificación de problemas y la toma de decisiones informadas.

6. CONCLUSIONES

1. A través del levantamiento de información y el diseño de una arquitectura de red virtualizada, se logró implementar con éxito la automatización del proceso de despliegue de la infraestructura tecnológica requerida para respaldar las operaciones del Departamento de Tecnología de la Universidad El Bosque. Esta automatización se llevó a cabo utilizando herramientas de AWS, lo que permitió una rápida y coherente implementación de la infraestructura necesaria, reduciendo significativamente el tiempo y los posibles errores asociados a despliegues manuales.

2. Se desarrolló un proceso de automatización de pruebas unitarias de código utilizando la herramienta TestNG. Este proceso permitió verificar la funcionalidad de componentes de software individuales de manera eficiente y confiable. La automatización de estas pruebas proporcionó una mayor confianza en la calidad del código y facilitó la detección temprana de errores, lo que llevó a una mejora en el proceso de desarrollo y pruebas, contribuyendo al proceso de entrega continua de software evitando interrupciones y dando mayor disponibilidad a los microservicios de la Universidad El Bosque.

3. Se implementó exitosamente un proceso de automatización de pruebas de usuario en aplicaciones utilizando TestNG. Esto posibilitó la evaluación de la funcionalidad de la aplicación desde la perspectiva de un usuario final (frontend) y desde la perspectiva del equipo de desarrollo y QA (backend). Las pruebas automatizadas garantizaron una cobertura exhaustiva de las funcionalidades clave y proporcionaron informes detallados sobre el rendimiento de la aplicación. Esto contribuyó a la mejora de la calidad del software y a la reducción del tiempo necesario para realizar pruebas de usuario manuales.

7. RECOMENDACIONES

1. Continuar la implementación de DevOps: Los beneficios de la implementación de la metodología DevOps han quedado demostrados a lo largo de este proyecto. Se recomienda continuar fomentando la colaboración y comunicación entre los equipos de desarrollo y operaciones de TI. La automatización de procesos, la integración continua y la entrega continua de software son prácticas que deben seguir siendo pilares en la búsqueda de la eficiencia y calidad en el ciclo de desarrollo y despliegue de aplicaciones.

2. Cambiar toda la Infraestructura a IaC: La adopción completa de la Infraestructura como Código (IaC) es una recomendación clave. La automatización del despliegue de infraestructura tecnológica ha demostrado ser una estrategia efectiva en este proyecto, y se sugiere que esta práctica se extienda a todos los aspectos de la infraestructura de TI de la universidad. Esto garantizará una gestión más eficiente, trazable y escalable de los recursos tecnológicos.

3. Implementar lo planteado en este documento en los demás Microservicios: A medida que la universidad continúa su camino hacia la modernización tecnológica, se sugiere que la automatización y las prácticas de DevOps se implementen en todos los microservicios y aplicaciones críticas. Esto permitirá una mayor uniformidad en los procesos de desarrollo y despliegue, así como una mayor agilidad en la entrega de nuevas funcionalidades.

4. Implementar Despliegue Continuo de Infraestructura: El despliegue continuo de infraestructura es una extensión natural de las prácticas de DevOps y la IaC. Se recomienda explorar la implementación de pipelines de despliegue automatizado para la infraestructura, lo que permitirá cambios en tiempo real y una mayor flexibilidad en la administración de recursos.

8. REFERENCIAS BIBLIOGRÁFICAS

[1] Gartner, “Information Technology (IT) Glossary - Essential Information Technology (IT) Terms & Definitions | Gartner,” https://www.gartner.com/en/information-technology/glossary.

[2] Amazon Web Services (AWS), “AWS glossary - AWS Glossary,” https://docs.aws.amazon.com/glossary/latest/reference/glos-chap.html.

[3] M. A. Akbar, S. Rafi, A. A. Alsanad, S. F. Qadri, A. Alsanad, and A. Alothaim, “Toward Successful DevOps: A Decision-Making Framework,” *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2022.3174094.

[4] M. Z. Prieto, L. S. Quinde, E. Insfran, and Y. Cabrera, “Automatización del proceso de despliegue de servicios en la nube,” *Maskana*, vol. 7, no. Supl., 2017.

[5] A. Qadeer, A. Waqar Malik, A. Ur Rahman, H. Mian Muhammad, and A. Ahmad, “Virtual Infrastructure Orchestration for Cloud Service Deployment,” *Computer Journal*, vol. 63, no. 2, pp. 295–307, Feb. 2020, doi: 10.1093/comjnl/bxz125.

[6] J. Caparroso, “Se acelera el viaje de la automatización en Colombia - Forbes Colombia,” *Se acelera el viaje de la automatización en Colombia*, 2020.

[7] A. Rafael, D. Rincón, and B. Profesor, “PRÁCTICAS DEVOPS DE ENTREGA CONTINUA DE SOFTWARE PARA LA TRANSFORMACIÓN DIGITAL DE LOS NEGOCIOS PABLO ANDRES CASTAÑEDA GARCIA Trabajo de Grado para Optar al Título de Magister en Ingeniería,” 2019.

[8] L. F. Ortiz Clavijo, J. D. Fernández Ledesma, S. Cadavid Nieto, and C. J. Gallego Duque, “Computación en la Nube: Estudio de herramientas orientadas a la Industria 4.0,” *Lámpsakos*, no. 20, 2018, doi: 10.21501/21454086.2560.

[9] N. Forsgren, J. Humble, G. Kim, and I. T. R. Press, *Accelerate: The Science of Lean Software and Devops: Building and Scaling High Performing Technology Organizations*. 2018.

[10] R. Duijs, P. Ravesteijn, and M. van Steenbergen, “Adaptation of enterprise architecture efforts to an agile environment,” in *31st Bled eConference: Digital Transformation: Meeting the Challenges, BLED 2018*, 2018. doi: 10.18690/978-961-286-170-4.26.

[11] M. S. Khan, A. W. Khan, F. Khan, M. A. Khan, and T. K. Whangbo, "Critical Challenges to Adopt DevOps Culture in Software Organizations: A Systematic Review," *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2022.3145970.

[12] DASA DEVOPS GLOSSARY, "DEVOPS AGILE SKILLS ASSOCIATION DASA," https://www.devopsagileskills.org/resources/ .

[13] J. Willis, "Convergence of DevOps," https://itrevolution.com/theconvergence-of-devops/ .

[14] N. Forsgren, J. Humble, G. Kim, and I. T. R. Press, *Accelerate: The Science of Lean Software and Devops: Building and Scaling High Performing Technology Organizations*. 2018.

[15] N. F. Velasquez, G. Kim, N. Kersten, and J. Humble, "2014 State of DevOps Report," *Puppetlabs*, 2014.

[16] D. C. Slater, "Agile development, at scale.," *Fierce CIO*. 2013.

[17] Northware, "Desarrollo en Cascada (Waterfall) VS Desarrollo Agile (SCRUM)," https://www.northware.mx/blog/desarrollo-en-cascada-waterfall-vs-desarrollo-agile-scrum/.

[18] M. Fowler, " MARTINFOWLER.COM," https://martinfowler.com/bliki/ContinuousDelivery.html.

[19] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.

[20] J. Humble and D. Farley, *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 2010.

[21] P. Duvall, S. Matyas, and a Glover, *Continuous integration: improving software quality and reducing risk*. 2007.

[22] J. Verona, P. Swartout, and M. Duffy, *Learning DevOps: Continuously Deliver Better Software*. Birmingham, UK: Packt Publishing, 2016.

[23] CMMI Product Team, "CMMI for Development, Version 1.3: Improving Processed for Better Products and Services," *Carnegie Mellon University, Software Engineering Institute*, 2010.

[24] A. Hemon, B. Lyonnet, F. Rowe, and B. Fitzgerald, “From Agile to DevOps: Smart Skills and Collaborations,” *Information Systems Frontiers*, vol. 22, no. 4, pp. 927–945, Aug. 2020, doi: 10.1007/s10796-019-09905-1.

[25] M. Fowler, “ MARTINFOWLER.COM,” https://martinfowler.com/bliki/ContinuousDelivery.html.

[26] Unam, “Lenguaje de programacion JAVA,” Docencia Unam.

[27] LSC. G. Moreno Beltrán, “JAVA como lenguaje universal de programación,” *XIKUA Boletín Científico de la Escuela Superior de Tlahuelilpan*, vol. 4, no. 8, 2016, doi: 10.29057/xikua.v4i8.332.

[28] P. Bindal, “Test Automation Selenium WebDriver using TestNG,” *Journal of Engineering Computers & Applied Sciences(JECAS)*, vol. 3, no. 9, pp. 18–40, 2014, [Online]. Available: http://docs.seleniumhq.org/

[29] A. M. R. Vincenzi, J. C. Maldonado, W. E. Wong, and M. E. Delamaro, “Coverage testing of Java programs and components,” *Sci Comput Program*, vol. 56, no. 1–2, 2005, doi: 10.1016/j.scico.2004.11.013.

[30] X. Devroey, S. Panichella, and A. Gambi, “Java Unit Testing Tool Competition: Eighth Round,” in *Proceedings - 2020 IEEE/ACM 42nd International Conference on Software Engineering Workshops, ICSEW 2020*, 2020. doi: 10.1145/3387940.3392265.

[31] S. Raemaekers, A. Van Deursen, and J. Visser, “The maven repository dataset of metrics, changes, and dependencies,” in *IEEE International Working Conference on Mining Software Repositories*, 2013. doi: 10.1109/MSR.2013.6624031.

[32] The Apache Software Foundation, “Maven – Welcome to Apache Maven,” *Maven.Apache.Org*. 2016.

[33] J. Bell, E. Melski, G. Kaiser, and M. Dattatreya, “Accelerating Maven by delaying test dependencies,” in *Proceedings - 3rd International Workshop on Release Engineering, RELENG 2015*, 2015. doi: 10.1109/RELENG.2015.16.

[34] C. Rishab Jain and R. Kaluri, “Design of automation scripts execution application for selenium webdriver and testng framework,” *ARPN Journal of Engineering and Applied Sciences*, vol. 10, no. 6, 2015.

[35] O. Ben-Kiki, C. Evans, and B. Ingerson, "YAML Ain't Markup Language (YAML™) Version 1.2," *Language (Baltim)*, 2009.

[36] M. Eriksson and V. Hallberg, "Comparison between JSON and YAML for data serialization," *The School of Computer Science and Engineering Royal Institute of Technology*, 2011.

[37] C. Anderson, "Docker," *IEEE Software*, vol. 32, no. 3. 2015. doi: 10.1109/MS.2015.62.

[38] D. Reis, B. Piedade, F. F. Correia, J. P. Dias, and A. Aguiar, "Developing Docker and Docker-Compose Specifications: A Developers' Survey," *IEEE Access*, vol. 10, 2022, doi: 10.1109/ACCESS.2021.3137671.

[39] B. Yang, A. Saile, S. Jain, A. E. Tomala-Reyes, M. Singh, and A. Ramnath, "Service discovery based blue-green deployment technique in cloud native environments," in *Proceedings - 2018 IEEE International Conference on Services Computing, SCC 2018 - Part of the 2018 IEEE World Congress on Services*, 2018. doi: 10.1109/SCC.2018.00031.

[40] B. Yang, A. Sailer, and A. Mohindra, "Survey and Evaluation of Blue-Green Deployment Techniques in Cloud Native Environments," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2020. doi: 10.1007/978-3-030-45989-5_6.

ANEXOS

ANEXO A: MANUAL DE USUARIO

Manual de operación

Índice

Introducción

En el presente documento se define cómo realizar el despliegue de un nuevo microservicio en la infraestructura de AWS de la Universidad El Bosque, en el cual se incluye como utilizar las plantillas de CloudFormation y los archivos necesarios a agregar al repositorio del código fuente para realizar las pruebas de usuario y una correcta utilización de CodeDeploy.

Marcados en verde los puntos a tener en cuenta para cada nuevo despliegue.

Especificaciones de cuentas

CODE

La cuenta CODE almacena los recursos de infraestructura necesarios para cada microservicio, en los cuales se almacena el código fuente y se realiza la creación de los artefactos.

Figura 26. Anexo A: Manual de usuario (Página 1)

DEV - QA - PROD

Las cuentas destino donde se almacenan los recursos de infraestructura que realizan la exposición de los servicios creados, almacenan los artefactos en ECR, realizan la creación de la revisión en ECS mediante CodeDeploy, realizan la creación de la tarea de ECS y realizan las pruebas de usuario mediante CodeBuild

Cómo realizar el despliegue de la infraestructura en CloudFormation (Requiere cambios)

Realiza la creación del stack principal en la cuenta CODE con sus correspondientes StackSets en las cuentas DEV, QA y PROD. Es necesario cambiar el nombre del microservicio y las prioridades de las reglas en el ELB de cada ambiente.

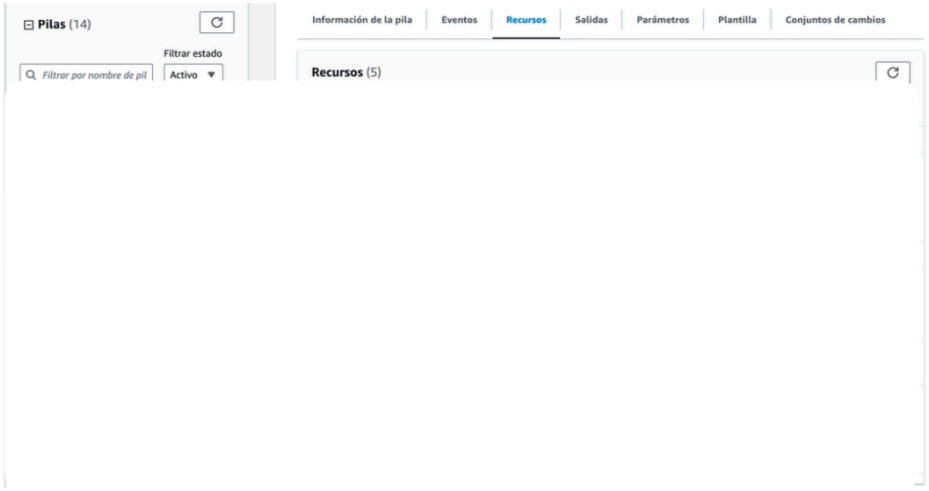
Ej.

```
aws cloudformation create-stack --stack-name ub-stack-inscriptions-backend --template-url https://ub-stackbucket-
```

Verificación de recursos creados

Una vez ejecutado el comando desde el CLI de AWS o desde la consola de AWS se pueden verificar los recursos creados en la sección de Stacks de CloudFormation en la cuenta CODE.

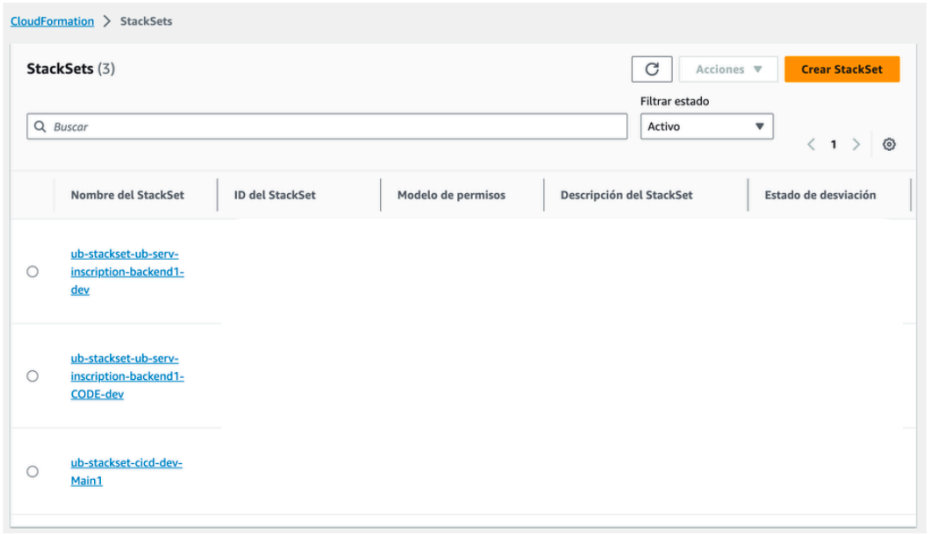
Ej.



En la sección de StackSets de CloudFormation en la cuenta CODE se puede evidenciar la creacion de cada uno de los Stacks de destino en cada una de las cuentas.

Ej.

Figura 27. Anexo A: Manual de usuario (Página 2)



En cada una de las cuentas destino se deben reflejar también los recursos de infraestructura creados en la sección de CloudFormation de AWS.

Ej.

Figura 28. Anexo A: Manual de usuario (Página 3)

Especificaciones de archivos almacenados en el repositorio INFRA y Bucket

main.yaml

Ubicación en S3

Ubicación en repositorio

Esta plantilla orquesta la creación de todos los recursos de infraestructura necesarios para la ejecución de los despliegues y pruebas en cada uno de los ambientes.

Crea los siguientes recursos:

- `StackSetDev`
  Crea los recursos de infraestructura en la cuenta DEV de acuerdo al microservicio.
- `CODEStackSetDev`
  Crea los recursos de infraestructura en la cuenta CODE para el ambiente DEV de acuerdo al microservicio.
- `StackSetQa`
  Crea los recursos de infraestructura en la cuenta QA de acuerdo al microservicio.
- `CODEStackSetQa`
  Crea los recursos de infraestructura en la cuenta CODE para el ambiente QA de acuerdo al microservicio.
- `StackSetProd`
  Crea los recursos de infraestructura en la cuenta PROD de acuerdo al microservicio.
- `CODEStackSetProd`
  Crea los recursos de infraestructura en la cuenta CODE para el ambiente PROD de acuerdo al microservicio.
- `Repo Codecommit`
  Crea el repositorio de CodeCommit del microservicio.

CODE.yaml

Ubicación en S3

Ubicación en repositorio

Esta plantilla realiza la creación de los recursos den la cuenta CODE para cada ambiente.

Crea los siguientes recursos:

- `CodeBuildProject`
- `CodeBuildServiceRole`
- `CodePipelineServiceRole`
- `Pipeline`

Account.yaml

Ubicación en S3

Ubicación en repositorio

Esta plantilla realiza la creacion de recursos en las cuentas de destino (DEV, QA, PROD).

Crea los siguientes recursos:

- `CodeBuildProjectTest`
- `CodeBuildServiceRole`

Figura 29. Anexo A: Manual de usuario (Página 4)

- `CodePipelineServiceRole`
- `ArtifactBucket`
- `S3BucketPolicy`
- `CodeDeployApp`
- `AppDeploymentGroup`
- `Pipeline`
- `ServiceRepository`
- `ECSServiceDef`
- `TargetGroup`
- `TargetGroup1`
- `ListenerRule`

MainAccount.yaml

Ubicación en S3

Ubicación en repositorio

Esta plantilla realiza la creación del cluster de ECS

Especificaciones de archivos requeridos para el despliegue

Microservicio

buildspec.yaml

Contiene las definiciones para la realización del build del artefacto y subida a ECR mediante CodeBuild.

Los parámetros son agregados mediante un script que realiza la lectura desde el archivo `taskdef.json`

Dockerfile

Contiene especificaciones para la creación del contenedor.

appspec.yaml

Contiene especificaciones para la creación del despliegue en CodeDeploy.

taskdef.json (Requiere cambios)

Contiene especificaciones para la creación del task definition del microservicio.

Es necesario agregar desde la línea 17 las variables y parámetros necesarios para cada microservicio.

Ej.

Parametro valor

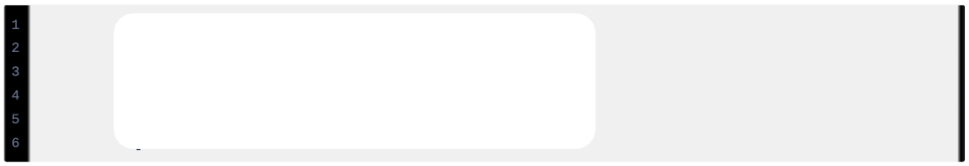


Figura 30. Anexo A: Manual de usuario (Página 5)

Pruebas de usuario (Carpeta usertest)

buildspec.yaml

Contiene las definiciones para la realización de las pruebas de usuario mediante Codebuild y exporta los informes generados en formato HTML.

docker-compose.yml

Contiene especificaciones para la creación del hub de Selenium desde donde se orquesta la generación de las pruebas de acuerdo a los exploradores definidos, crea dos contenedores, el hub y el contenedor del explorador.

pom.xml

Contiene las dependencias y parámetros para la ejecución de las pruebas en TestNG.

testng.xml

Es el archivo orquestador de las pruebas, en caso de requerir se pueden agregar pruebas en paralelo en múltiples exploradores.

Archivos fuente de pruebas de usuario

BaseTest.java

Contiene las generalidades del driver para la ejecución de las pruebas de usuario.

CapabilityFactory.java

Realiza el llamado para obtener las opciones de configuración del explorador.

FirstTest.java (Requiere cambios)

Se deben especificar las pruebas de acuerdo al microservicio.

Ej.

Número de clicks, listas desplegables.

OptionsManager.java

Se definen las opciones de configuración para el explorador. (Ej. maximizar pantalla, ignorar errores de certificado)

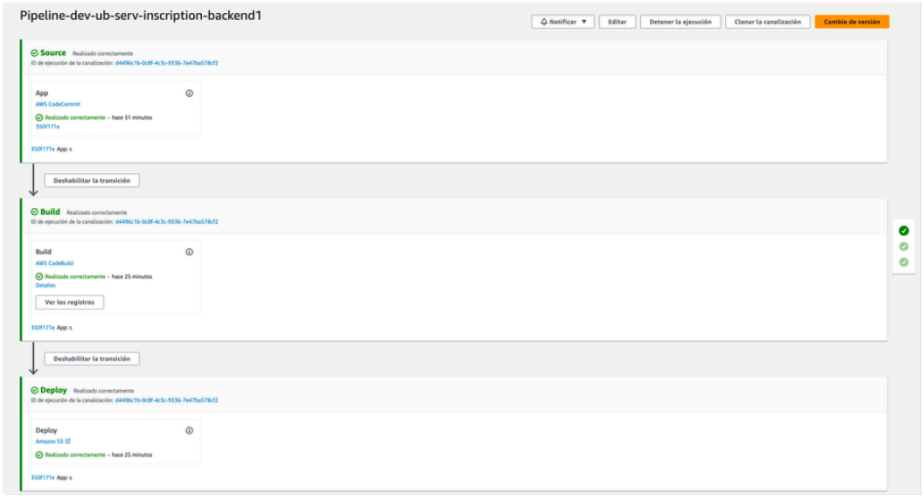
Cómo activar el trigger para la ejecucion del flujo completo de trabajo

Después de haber realizado los cambios anteriores de acuerdo al microservicio, las modificaciones se deben almacenar en el repositorio del microservicio en la rama DEV, al realizar el git push, CodePipeline escucha el cambio y desencadena la ejecución de los procesos desde la cuenta CODE hasta el proceso de pruebas de usuario en CodeBuild en la cuenta DEV.

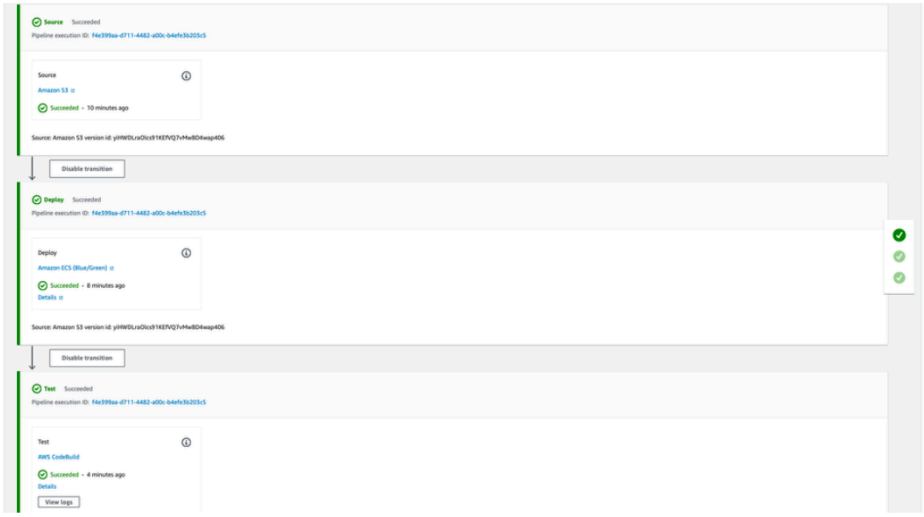
Ej.

Cuenta CODE

Figura 31. Anexo A: Manual de usuario (Página 6)



Cuenta DEV



Los cambios en la rama de QA y PROD se desencadenan cuando se hace Merge desde alguna de las cuentas anteriores en la cadena, desde la construcción del artefacto hasta las pruebas.

La cuenta de PROD tiene una etapa adicional de aprobación manual y no realiza la etapa de testing de usuario.

Cómo obtener el informe de resultados de las pruebas de usuario

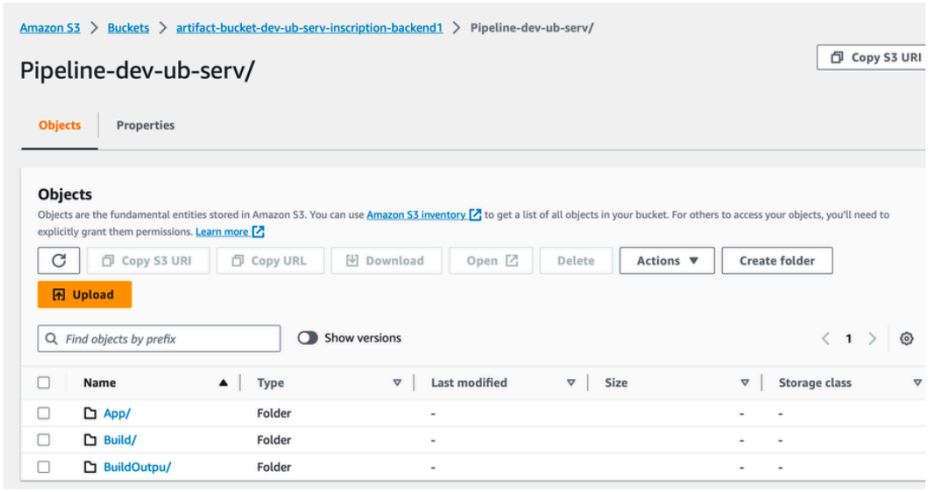
Cada microservicio tiene asociado en cada una de las cuentas un bucket de S3 llamado ArtifacBucket.

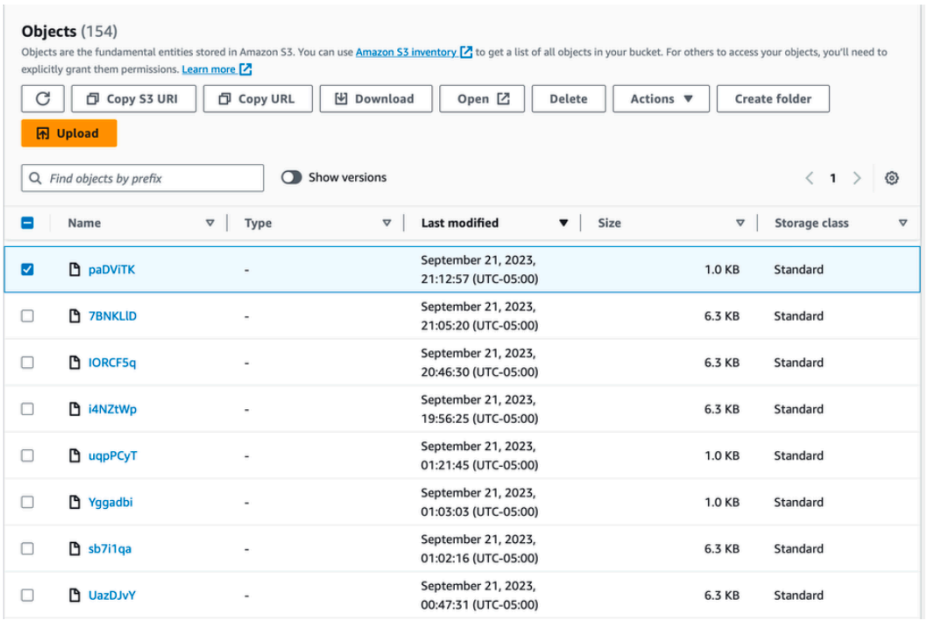
Figura 32. Anexo A: Manual de usuario (Página 7)

Ej.

Artifact bucket del microservicio de inscriptions en la cuenta DEV.

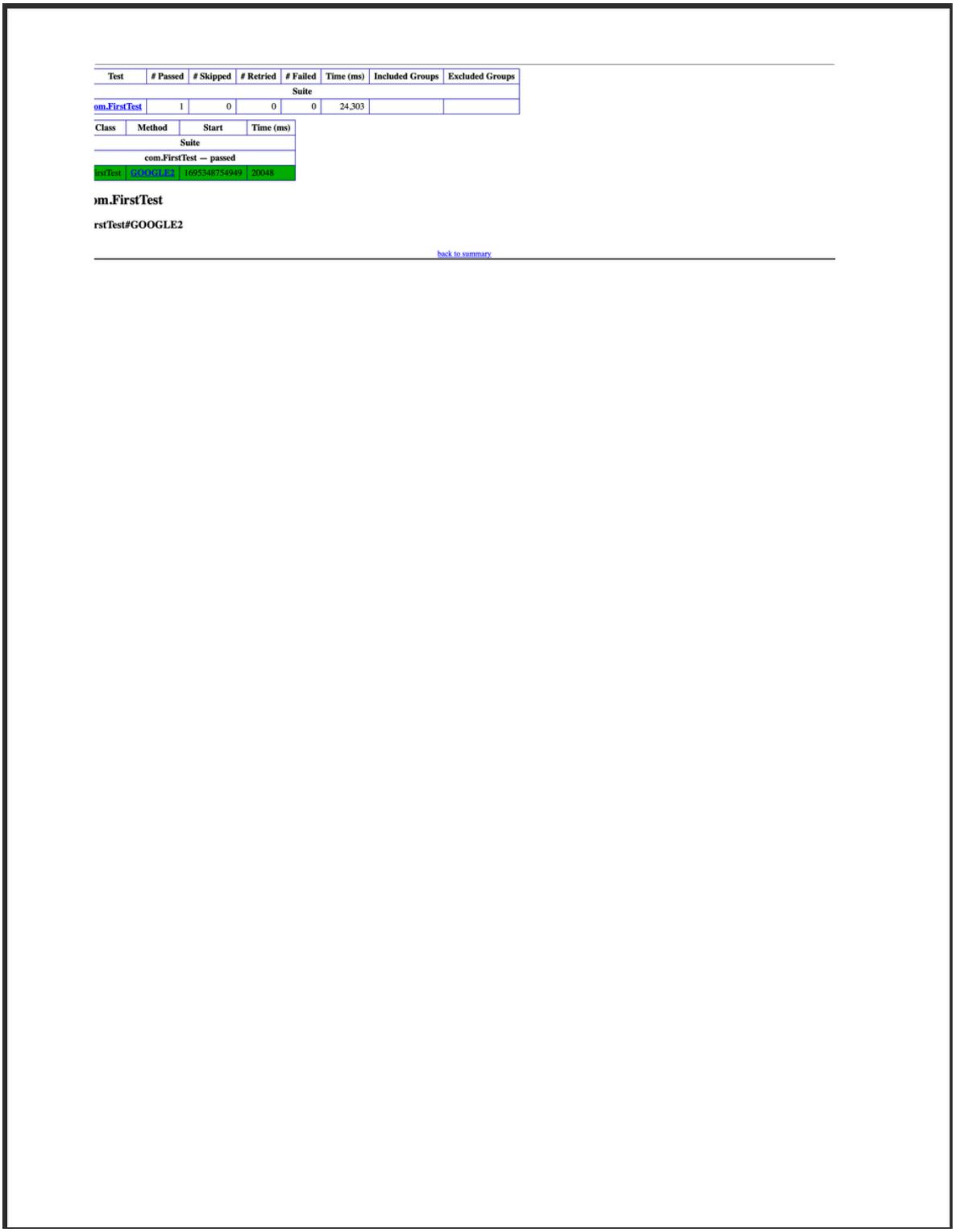
Existe un objeto en la carpeta BuildOutpu que contiene el HTML generado después de las pruebas.





Al descargar el archivo se obtiene el reporte.

Figura 33. Anexo A: Manual de usuario (Página 8)



Test	# Passed	# Skipped	# Retried	# Failed	Time (ms)	Included Groups	Excluded Groups
Suite							
om.FirstTest	1	0	0	0	24,303		

Class	Method	Start	Time (ms)
Suite			
com.FirstTest — passed			
irstTest	GOOGLE2	1695348754949	20048

Figura 34. Anexo A: Manual de usuario (Página 9)

ANEXO B: DOCKERFILE

```
1   ARG REPOSITORY_URI_
2   FROM ${REPOSITORY_URI_}
3
4   WORKDIR /app
5   EXPOSE 8080
6   COPY ./target/*-SNAPSHOT.jar ./app.jar
7   RUN mkdir ~/.aws
8   RUN mkdir ~/.aws/credentials
9   COPY ./.aws ~/.aws
10  CMD java $JAVA_OPTIONS -jar app.jar
```

Figura 35. Anexo B: Dockerfile

ANEXO C: APPSPEC

```
1   version: 0.0
2   Resources:
3     - TargetService:
4         Type: AWS::ECS::Service
5         Properties:
6           TaskDefinition: "<TASK_DEFINITION>"
7           LoadBalancerInfo:
8             ContainerName: "SERVICE_NAME"
9             ContainerPort: 8080
10
```

Figura 36. Anexo C: AppSpec

ANEXO D: BUILDSPEC

```yaml
version: 0.2

phases:
  pre_build:
    commands:
      - tar -xvzf usertest.tar.gz
      - cd usertest
      - docker-compose up -d
  build:
    commands:
      - mvn clean test

  post_build:
    commands:
      - cp target/surefire-reports/emailable-report.html ../.

artifacts:
  files:
    - emailable-report.html
```

Figura 37. Anexo D: Buildspec

ANEXO E: DOCKER-COMPOSE

```yaml
version: '3'
services:
  selenium-hub:
    image: selenium/hub:latest
    container_name: selenium-hub
    ports:
    - '4444:4444'
    - '4443:4443'
    - '4442:4442'
  chrome:
    image: selenium/node-chrome:latest
    shm_size: 2gb
    volumes:
    - '/dev/shm:/dev/shm'
    depends_on:
    - selenium-hub
    environment:
    - SE_EVENT_BUS_HOST=selenium-hub
    - SE_EVENT_BUS_PUBLISH_PORT=4442
    - SE_EVENT_BUS_SUBSCRIBE_PORT=4443
    - SE_EVENT_MAX_INSTANCES=4
    - SE_EVENT_MAX_SESSIONS=4
    - SE_NODE_SESSION_TIMEOUT=10
```

Figura 38. Anexo E: Docker-Compose

ANEXO F: POM

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>TestNGParallel</groupId>
    <artifactId>TestNGParallel</artifactId>
    <version>1.0-SNAPSHOT</version>
    <build>
        <plugins>
            <plugin>…
            </plugin>
            <plugin>…
            </plugin>
        </plugins>
    </build>

    <dependencies>
        <dependency>…
        </dependency>

        <dependency>…
        </dependency>
        <dependency>…
        </dependency>
        <dependency>…
        </dependency>
    </dependencies>
    <properties>
        <encoding>utf-8</encoding>
    </properties>
</project>
```

Figura 39. Anexo F: POM

ANEXO G: TESTNG

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE suite SYSTEM "http://testng.org/testng-1.0.dtd">
<suite thread-count="1" name="Suite" parallel="none" >
    <test name="com.FirstTest">
        <parameter name="browser" value="chrome"/>
        <classes>
            <class name="FirstTest">
                <methods>
                    <include name="GOOGLE1" />
                </methods>
            </class>
        </classes>
    </test>
</suite>
```

Figura 40. Anexo G: TestNG

ANEXO H: BASETEST

```java
import java.net.MalformedURLException;
import java.net.URL;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.remote.RemoteWebDriver;
import org.testng.annotations.AfterClass;
import org.testng.annotations.AfterMethod;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Parameters;

public class BaseTest {

    protected static ThreadLocal<RemoteWebDriver> driver = new ThreadLocal<>();
    public CapabilityFactory capabilityFactory = new CapabilityFactory();

    @BeforeMethod
    @Parameters(value={"browser"})
    public void setup (String browser) throws MalformedURLException {
        driver.set(new RemoteWebDriver(new URL("http://localhost:4444/wd/hub"), capabilityFactory.getCapabilities(browser)));
    }

    public WebDriver getDriver() {
        return driver.get();
    }

    @AfterMethod
    public void tearDown() {
        getDriver().quit();
    }

    @AfterClass void terminate () {
        driver.remove();
    }
}
```

Figura 41. Anexo H: BaseTest

ANEXO I: CAPABILITYFACTORY

```java
import org.openqa.selenium.Capabilities;

public class CapabilityFactory {
    public Capabilities capabilities;

    public Capabilities getCapabilities (String browser) {
        capabilities = OptionsManager.getChromeOptions();
        return capabilities;
    }
}
```

Figura 42. Anexo I: CapabilityFactory

ANEXO J: OPTIONSMANAGER

```java
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.chrome.ChromeOptions;
import org.openqa.selenium.chrome.*;


public class OptionsManager {

    public static ChromeOptions getChromeOptions() {
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--start-maximized");
        options.addArguments("--ignore-certificate-errors");
        options.addArguments("--disable-popup-blocking");
        return options;
    }
}
```

Figura 43. Anexo J: OptionsManager