

UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN

Facultad de Ingeniería

Escuela Profesional de Ingeniería en Informática y Sistemas

**Automatización de aplicaciones contenerizadas con
infraestructura bajo código en EKS para reducir
el tiempo de recuperación y para garantizar
continuidad operativa sobre
la nube de AWS**

TESIS

PRESENTADA POR:

Bach. AVELARDO CATACORA GUTIERREZ

Para optar el Título Profesional de:

INGENIERO EN INFORMÁTICA Y SISTEMAS

TACNA-PERÚ

2026

PÁGINA DE JURADOS

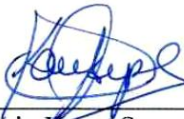
UNIVERSIDAD NACIONAL JORGE BASADRE GROHMANN, TACNA

FACULTAD DE INGENIERÍA

ESCUELA PROFESIONAL DE INGENIERÍA EN INFORMÁTICA Y
SISTEMAS

AUTOMATIZACIÓN DE APLICACIONES CONTENERIZADAS CON INFRAESTRUCTURA BAJO CÓDIGO EN EKS PARA REDUCIR EL TIEMPO DE RECUPERACIÓN Y PARA GARANTIZAR CONTINUIDAD OPERATIVA SOBRE LA NUBE DE AWS

TESIS SUSTENTADA Y APROBADA EL 29 DE MAYO DEL 2026
ESTANDO EL JURADO CALIFICADOR INTEGRADO POR:

Presidente	:	 Dra. Karín Yanet Supo Gavancho
Secretario	:	 MSc. Arnold Christian Loaiza Fabian
Vocal	:	 MSc. Milagros Gleny Cohaila Gonzales
Asesor	:	 Dra. Karín Yanet Supo Gavancho

CERTIFICADO DE SIMILITUD

Yo, KARIN YANET SUPO GAVANCHO en mi condición de asesor acreditado por la Resolución de Facultad N° 9729-2025-FAIN/UNJBG, de la tesis titulada

"Automatización de aplicaciones contenerizadas con infraestructura bajo código en EKS para reducir el tiempo de recuperación y para garantizar continuidad operativa sobre la nube de AWS".

Presentado por el Bach. Avelardo Catacora Gutierrez,

Para optar el título profesional de
INGENIERO EN INFORMATICA Y SISTEMAS.

Habiendo cumplido con lo establecido en el reglamento de originalidad y de similitud de trabajos de investigación y producción intelectual. Considerando que según la revisión, evaluación y análisis realizado a través del software de similitud textual TURNITIN cuenta con el nivel de similitud permitido cuyo porcentaje es 05%. Por lo que CERTIFICO QUE LA SIMILARIDAD de la tesis está de acuerdo al nivel PERMITIDO, para continuar con los trámites correspondientes y para su publicación en el repositorio institucional. Se emite el presente certificado con fines de continuar con los trámites respectivos para su obtención del grado/título/especialidad.



FIRMA DE ASESOR

Dra. Ing. Karin Yanet Supo Gavancho

DNI 00795714



Huella Digital



FIRMA DE TESISTA

Bach. Avelardo Catacora Gutierrez

DNI 00498835



Huella Digital

DEDICATORIA

A mis hijas Johanan y Ayelen a Madelayne mi compañera de esta aventura, por su incondicional apoyo en cada paso de este camino. Su apoyo, paciencia y amor fueron la motivación constante que me impulsaron a culminar este desafío.

A Santiago, Carmen y Valentina que me cuidan y son mi ejemplo de perseverancia, tenacidad y bondad y a todos los que ya no están físicamente presentes.

Dedico este trabajo también a mi familia, quienes con sus consejos y aliento me enseñaron que el esfuerzo y la perseverancia son la base para alcanzar los sueños.

AGRADECIMIENTOS

En primer lugar, agradezco a Dios por darme la fortaleza, la sabiduría y la salud necesarias para completar esta investigación.

A mis docentes y asesores, quienes con sus orientaciones académicas y valiosas observaciones contribuyeron a la mejora de este trabajo.

Finalmente, a mis amigos y colegas, por su apoyo moral y por estar presentes en las distintas etapas de este camino.

ÍNDICE TEMÁTICO

DEDICATORIA.....	IV
AGRADECIMIENTOS.....	V
ÍNDICE TEMÁTICO.....	VI
ÍNDICE DE TABLAS	IX
ÍNDICE DE FIGURAS	X
RESUMEN	XI
ABSTRACT.....	XII
LISTA DE ABREVIATURAS Y SIGLAS	XIII
INTRODUCCIÓN	1
CAPÍTULO I. PLANTEAMIENTO DEL PROBLEMA.....	4
1.1. Antecedentes del problema a investigar	4
1.2. Descripción del problema.....	5
1.3. Planteamiento del problema	6
1.4. Objetivos	7
1.5. Justificación de la investigación.....	7
1.6. Viabilidad de la investigación	8
1.7. Delimitación del problema	9
1.8. Limitaciones del problema	10
1.9. Hipótesis de la investigación	11
1.10. Variables de la investigación.....	12
CAPÍTULO II. MARCO TEÓRICO.....	16
2.1. Antecedentes de la investigación	16
2.2. Bases teóricas de la Automatización de infraestructura en la nube	21
2.3. Arquitectura base: EKS y clúster de Kubernetes	26

2.4. Contenerización y gestión de imágenes	31
2.5. Monitorización, métricas y alertas	36
2.6. Gestión del tráfico y seguridad del clúster	42
2.7. Configuración avanzada	42
2.8. Definiciones conceptuales	45
CAPÍTULO III. MARCO METODOLÓGICO	48
3.1. Enfoque de la investigación	48
3.2. Diseño de la investigación.....	48
3.3. Nivel de la investigación	49
3.4. Población	49
3.5. Muestra	50
3.6. Técnicas e instrumentos de recolección de datos	51
3.7. Análisis y procesamiento de datos	52
3.8. Aspectos éticos	53
CAPÍTULO IV. RESULTADOS.....	54
4.1. Descripción de la Arquitectura que soporta la automatización (Flujo App)	54
4.2. Arquitectura Interna del clúster EKS (Default Cluster)	60
4.3. Descripción de las aplicaciones contenerizadas analizadas	65
4.4. Estado inicial manual del despliegue (Pretest).....	69
4.5. Proceso de aprovisionamiento de la infraestructura mediante IaC (Create Cluster)	
72	
4.6. Proceso de integración y despliegue continuo (CI/CD) (Create Pipelines)	79
4.7. Despliegue de Aplicaciones en EKS	87
4.8. Comparación pretest – postest del proceso de recuperación y continuidad operativa	87
4.9. Monitoreo y Métricas del Clúster en EKS	91
4.10. Resultados alineados a los indicadores establecidos en la investigación	109

4.11. Análisis Técnico de los puntos críticos	111
4.12. Contratación de hipótesis	112
CAPÍTULO V DISCUSIÓN	119
5.1. Validación técnica del modelo de automatización	119
5.2. Aplicación de la tecnología encontrada.....	120
5.3. Contraste con trabajos de investigación similares.....	121
CONCLUSIONES	125
RECOMENDACIONES	127
REFERENCIAS BIBLIOGRÁFICAS.....	128
ANEXOS.....	132

ÍNDICE DE TABLAS

Tabla 1. Operacionalización de la variable Independiente: Automatización de aplicaciones contenerizadas con infraestructura bajo código EKS	14
Tabla 2. Operacionalización de la variable Dependiente: Reducción del tiempo de recuperación y garantía de la continuidad operativa.....	15
Tabla 3. Componentes del flujo arquitectónico en AWS Antamina Cloud.....	56
Tabla 4. Componentes de la Arquitectura interna del clúster EKS en AWS.....	64
Tabla 5. Aplicaciones contenerizadas incluidas en el análisis.....	69
Tabla 6. Matriz de validación del diseño de automatización en Amazon EKS.....	82
Tabla 7. Tiempo promedio de despliegue manual vs IaC.....	84
Tabla 8. Consistencia del Pipeline CI/CD	84
Tabla 9. Proceso de CI/CD en AWS.....	86
Tabla 10. Comparación de medias e indicadores de mejora entre el escenario manual y automatizado.....	90
Tabla 11. Disponibilidad del servicio antes y después de la automatización	108
Tabla 12. Comparación del tiempo promedio de recuperación (MTTR) antes y después de la automatización	108
Tabla 13. Automatización de despliegue	109
Tabla 14. Recuperación del servicio	110
Tabla 15. Continuidad y disponibilidad del servicio	110

ÍNDICE DE FIGURAS

Figura 1. Arquitectura diseñada para la automatización.....	55
Figura 2. Diagrama de la Arquitectura en AWS que soporta la automatización (componentes identificados).....	59
Figura 3. Arquitectura interna del clúster EKS en AWS	62
Figura 4. Flujo de aprovisionamiento del clúster EKS mediante IaC	75
Figura 5. Generación del archivo de configuración YAML para la creación del clúster en Amazon EKS	77
Figura 6. Proceso de integración y despliegue continuo (CI/CD) en AWS	79
Figura 7. Monitoreo en tiempo real de CPU, memoria y pods del clúster en Lens IDE	93
Figura 8. Monitoreo de métricas de nodos en Lens IDE	95
Figura 9. Monitoreo de pods y workloads en Lens IDE	96
Figura 10. Identificación de pods en estado Failed o CrashLoopBackOff en Lens IDE	97
Figura 11. Monitoreo de métricas de red y servicios en Lens IDE	98
Figura 12. Visualización de Ingress Resources en Lens IDE	99
Figura 13. Estado de servicios Kubernetes e Ingress Controllers en Lens IDE	100
Figura 14. Visualización de Storage Classes en Lens IDE	101
Figura 15. Monitoreo de Persistent Volumes en Lens IDE	102
Figura 16. Visualización de Persist Volume CLaims en Lens IDE	103
Figura 17. Visualización de Service Accounts en Lens IDE	104
Figura 18. Visualización de Horizontal Pod Autoscaler (HPA) en Lens IDE.....	105
Figura 19. Visualización de eventos y logs del sistema en Lens IDE	106

RESUMEN

El presente estudio tuvo como objetivo diseñar e implementar un modelo de automatización para aplicaciones contenerizadas mediante Infraestructura como Código (IaC) sobre Amazon Elastic Kubernetes Service (EKS), con la finalidad de reducir el tiempo medio de recuperación (MTTR) y fortalecer la continuidad operativa en entornos desplegados sobre Amazon Web Services (AWS). La investigación fue de tipo aplicada y de carácter tecnológico, basada en la comparación descriptiva entre un escenario inicial de gestión manual y un escenario automatizado. La propuesta integró el aprovisionamiento declarativo de recursos de infraestructura (VPC, IAM y EKS), la implementación de un pipeline CI/CD para la construcción, versionado y despliegue automatizado de aplicaciones (Build - ECR - Deploy), así como mecanismos de operación y resiliencia mediante Horizontal Pod Autoscaler (HPA), Ingress, RBAC, ConfigMaps, Secrets y almacenamiento persistente. La observabilidad del entorno se gestionó mediante Lens IDE, permitiendo el monitoreo de nodos, pods, workloads, servicios, almacenamiento, seguridad, red, autoescalado y eventos del clúster. Los resultados evidenciaron mejoras significativas en los indicadores evaluados. El tiempo promedio de despliegue se redujo de 43 minutos en el escenario manual a 8.7 minutos en el entorno automatizado, representando una mejora del 79.7%. Asimismo, el MTTR disminuyó de 15.6 minutos a 2.6 minutos, equivalente a una reducción del 83.3%, mientras que la disponibilidad del servicio aumentó de 93.96% a 99.17%, incrementándose en 5.21 puntos porcentuales. Adicionalmente, el pipeline CI/CD alcanzó un 98% de éxito en las ejecuciones automatizadas, reduciendo la intervención manual y mejorando la consistencia operativa. Se concluye que el modelo propuesto constituye una solución viable, reproducible y técnicamente sustentada para mejorar la resiliencia, disponibilidad y continuidad operativa de aplicaciones contenerizadas en AWS, aportando una arquitectura de referencia, plantillas reutilizables de infraestructura como código y lineamientos operativos para futuros entornos productivos.

Palabras clave: Automatización, Infraestructura como Código, Amazon EKS, Kubernetes, CI/CD, Continuidad operativa, MTTR, Observabilidad.

ABSTRACT

The aim of this study was to design and implement an automation model for containerized applications using Infrastructure as Code (IaC) on Amazon Elastic Kubernetes Service (EKS), with the purpose of reducing the Mean Time to Recovery (MTTR) and strengthening operational continuity in environments deployed on Amazon Web Services (AWS). The research was applied and technological in nature, based on a descriptive comparison between an initial manual management scenario and an automated scenario. The proposed solution integrated declarative provisioning of infrastructure resources (VPC, IAM, and EKS), the implementation of a CI/CD pipeline for application build, versioning, and automated deployment (Build → ECR → Deploy), as well as operational and resilience mechanisms through Horizontal Pod Autoscaler (HPA), Ingress, RBAC, ConfigMaps, Secrets, and persistent storage. Environment observability was managed using Lens IDE, enabling monitoring of nodes, pods, workloads, services, storage, security, networking, autoscaling, and cluster events. The results showed significant improvements in the evaluated indicators. The average deployment time was reduced from 43 minutes in the manual scenario to 8.7 minutes in the automated environment, representing a 79.7% improvement. Likewise, MTTR decreased from 15.6 minutes to 2.6 minutes, equivalent to an 83.3% reduction, while service availability increased from 93.96% to 99.17%, representing an improvement of 5.21 percentage points. Additionally, the CI/CD pipeline achieved a 98% success rate in automated executions, reducing manual intervention and improving operational consistency. It is concluded that the proposed model constitutes a viable, reproducible, and technically sound solution for improving the resilience, availability, and operational continuity of containerized applications on AWS. Furthermore, it provides a reference architecture, reusable Infrastructure as Code templates, and operational guidelines for future production environments.

Keywords: Automation, Infrastructure as Code, Amazon EKS, Kubernetes, CI/CD, Operational Continuity, MTTR, Observability.

LISTA DE ABREVIATURAS Y SIGLAS

Sigla / Acrónimo	Significado
API	<i>Application Programming Interface</i> – Conjunto de reglas y definiciones que permiten la comunicación entre aplicaciones de software.
AWS	<i>Amazon Web Services</i> – Plataforma de servicios en la nube de Amazon que ofrece infraestructura, almacenamiento, bases de datos y herramientas de desarrollo.
CDK	<i>Cloud Development Kit</i> – Marco de desarrollo de AWS que permite definir infraestructura en la nube mediante lenguajes de programación como Python, TypeScript o Java.
CI/CD	<i>Continuous Integration / Continuous Deployment</i> – Prácticas de integración y despliegue continuo utilizadas para automatizar el flujo de desarrollo de software.
CLI	<i>Command Line Interface</i> – Interfaz de línea de comandos que permite interactuar con herramientas o servicios mediante comandos de texto.
Cluster	Conjunto de nodos gestionados de forma centralizada que ejecutan aplicaciones contenerizadas bajo un mismo entorno de orquestación.
ConfigMaps	<i>Configuration Maps</i> – Objetos de Kubernetes utilizados para almacenar datos de configuración en pares clave-valor, que pueden ser consumidos por pods y contenedores.
DNS	<i>Domain Name System</i> – Sistema que traduce nombres de dominio en direcciones IP para la comunicación en red.
EC2	<i>Elastic Compute Cloud</i> – Servicio de AWS que proporciona capacidad de cómputo escalable mediante instancias virtuales.
ECR	<i>Elastic Container Registry</i> – Servicio de AWS para almacenar, administrar y desplegar imágenes de contenedores Docker.
EKS	<i>Elastic Kubernetes Service</i> – Servicio administrado de Kubernetes provisto por AWS para ejecutar y escalar aplicaciones contenerizadas.
HPA	<i>Horizontal Pod Autoscaler</i> – Componente de Kubernetes que ajusta automáticamente el número de pods en función de métricas de uso de CPU u otras personalizadas.

Sigla / Acrónimo	Significado
IaC	<i>Infrastructure as Code</i> (Infraestructura como Código) – Metodología que permite aprovisionar y gestionar recursos de infraestructura mediante archivos de configuración automatizados.
Lens IDE	Entorno visual para administrar y monitorear clústeres de Kubernetes con métricas, logs y recursos en tiempo real.
Load Balancer	Balanceador de carga que distribuye el tráfico entrante entre múltiples instancias o pods para asegurar disponibilidad.
MTBF	<i>Mean Time Between Failures</i> – Métrica que mide el tiempo promedio entre fallas de un sistema o componente.
MTTR	<i>Mean Time To Recovery</i> – Indicador que mide el tiempo medio necesario para restaurar un servicio después de una falla.
Nginx Ingress Controller	Componente de Kubernetes que gestiona el acceso externo a los servicios dentro del clúster mediante reglas de enrutamiento HTTP/HTTPS.
Node	Máquina física o virtual en la que se ejecutan los pods dentro de un clúster Kubernetes.
Pod	Unidad básica de ejecución en Kubernetes que agrupa uno o varios contenedores compartiendo red y almacenamiento.
PV	<i>Persistent Volume</i> – Recurso de Kubernetes que representa un volumen físico o lógico disponible en el clúster para almacenamiento persistente de datos.
PVC	<i>Persistent Volume Claim</i> – Solicitud de un volumen persistente por parte de un usuario o aplicación dentro del clúster.
RBAC	<i>Role-Based Access Control</i> – Sistema de control de acceso basado en roles que gestiona permisos dentro del clúster de Kubernetes.
Rolling	Término utilizado para describir el proceso de actualización gradual o rotativa de componentes dentro del despliegue continuo.
Rolling Update	Estrategia de actualización continua en Kubernetes que reemplaza gradualmente las versiones antiguas de los pods por nuevas versiones sin interrumpir el servicio.
S3	<i>Simple Storage Service</i> – Servicio de almacenamiento de objetos de AWS utilizado para guardar y recuperar cualquier cantidad de datos.
SC	<i>Storage Class</i> – Recurso de Kubernetes que define las clases de almacenamiento disponibles para aprovisionamiento dinámico de volúmenes persistentes.

Sigla / Acrónimo	Significado
SDK	<i>Software Development Kit</i> – Conjunto de herramientas y bibliotecas utilizadas para desarrollar aplicaciones sobre una plataforma específica.
Terraform	Herramienta de código abierto para definir, aprovisionar y gestionar infraestructura mediante IaC.
VPC	<i>Virtual Private Cloud</i> – Red virtual privada en AWS que permite aislar recursos y controlar el tráfico de red dentro de la nube.
VPC Subnet	Segmento de red dentro de una VPC donde se ubican los recursos y servicios desplegados.
YAML	<i>YAML Ain't Markup Language</i> – Formato de archivo utilizado para definir configuraciones declarativas en Kubernetes y herramientas de automatización.

INTRODUCCIÓN

En los últimos años, la adopción de servicios en la nube ha transformado de manera significativa la forma en que las organizaciones gestionan sus aplicaciones y recursos tecnológicos. La virtualización, los contenedores y las arquitecturas basadas en microservicios han permitido incrementar la flexibilidad y escalabilidad de los sistemas, aunque también han planteado nuevos retos vinculados con la continuidad operativa, la seguridad y la gestión eficiente de infraestructuras cada vez más complejas.

En este ámbito, los procesos manuales de despliegue resultan insuficientes para responder a la demanda actual de agilidad y confiabilidad. La ausencia de automatización genera riesgos en la disponibilidad de los servicios, aumenta los tiempos de recuperación ante fallos y limita la capacidad de las organizaciones para garantizar entornos consistentes en desarrollo, pruebas y producción. De allí surge la necesidad de implementar estrategias de Infraestructura como Código (IaC) y flujos de integración y despliegue continuo que aseguren la eficiencia y la trazabilidad de los procesos en la nube.

El objetivo general de esta investigación fue diseñar e implementar un modelo de automatización para aplicaciones contenerizadas mediante IaC en Amazon EKS, orientado a reducir el MTTR y garantizar la continuidad operativa en AWS. Los objetivos específicos comprendieron: (a) identificar y justificar herramientas de IaC para el aprovisionamiento en AWS; (b) diseñar flujos de automatización para el despliegue y gestión en EKS; (c) implementar un entorno de prueba con fallos controlados para estimar el MTTR; y (d) analizar el impacto de la automatización en la continuidad operativa y proponer mejoras.

Metodológicamente, se trata de un estudio aplicativo, no experimental y tecnológico. El modelo integra IaC (p. ej., Terraform/CloudFormation/CDK) para el aprovisionamiento de recursos (red, identidades, clúster y node groups), un pipeline CI/CD (construcción → versionado → publicación en ECR → despliegue declarativo con manifiestos/Helm), y operación con Ingress/ALB, ConfigMaps/Secrets, RBAC, políticas de autoscalado (HPA) y almacenamiento persistente (SC/PV/PVC). La validación técnica se realizó mediante ensayos controlados (fallos de contenedores, interrupciones de servicio y picos de carga) y observabilidad con Lens IDE, capturando métricas de nodos, pods/workloads, red/servicios, almacenamiento, seguridad, HPA y eventos/logs.

Los resultados muestran que el modelo propuesto mejora significativamente los indicadores operativos evaluados respecto al escenario manual. La comparación de medias evidenció que el tiempo promedio de despliegue se redujo de 43 minutos a 8.7 minutos (79.7% de mejora), mientras que el tiempo medio de recuperación (MTTR) disminuyó de 15.6 minutos a 2.6 minutos (83.3% de reducción). Asimismo, la disponibilidad del servicio aumentó de 93.96% a 99.17%, incrementándose en 5.21 puntos porcentuales. Adicionalmente, el modelo mantuvo la disponibilidad durante rolling updates, redujo la intervención humana y mejoró la trazabilidad de cambios, evidenciado por la alineación entre réplicas deseadas y disponibles, la estabilidad de servicios e Ingress, la correcta persistencia de datos y la activación efectiva del HPA. Como aportes, se presenta una arquitectura de referencia, artefactos reutilizables (plantillas IaC y runbooks operativos) y un conjunto de KPIs y criterios de aceptación para sostener la continuidad operativa.

El alcance del estudio se centra en la validación técnico-operativa del modelo; no se incluyen pruebas de carga prolongadas ni replicaciones estadísticas. Como líneas de mejora se propone incorporar GitOps (ArgoCD), series temporales (Prometheus/Grafana), optimización de autoscalado (p. ej., Cluster Autoscaler/Karpenter) e integración de señales FinOps para decisiones costo-eficientes.

La estructura del documento es la siguiente: el Capítulo I presenta el Planteamiento del problema, que contiene los objetivos e hipótesis principalmente; el Capítulo II desarrolla el marco teórico y antecedentes; el Capítulo III describe la

metodología de la investigación; el Capítulo IV expone la propuesta técnica (arquitectura, aprovisionamiento con IaC, CI/CD, despliegue en EKS y monitoreo y métricas con su análisis técnico y recomendaciones); y el Capítulo V desarrolla la discusión, incluyendo la validación técnica del modelo, la aplicación de la tecnología encontrada, el contraste con trabajos similares, así como conclusiones y recomendaciones.

CAPÍTULO I

PLANTEAMIENTO DEL PROBLEMA

1.1. Antecedentes del problema a investigar

En los últimos años, diversas investigaciones han analizado el impacto de la adopción de servicios en la nube y su relación con la continuidad operativa de los sistemas empresariales. Mora (2023) y Moreno Villarrubia (2024) destacan que la migración hacia plataformas como AWS ha fortalecido la escalabilidad y flexibilidad de las aplicaciones, aunque persisten desafíos en la recuperación ante fallos. García Granados (2023) añade que la transformación digital exige infraestructuras cada vez más resilientes, capaces de responder de forma ágil a incidentes sin comprometer la calidad del servicio.

En cuanto a la gestión de aplicaciones contenerizadas, Cesanelli (2023) y Escoda Gil (2023) evidencian los beneficios de Kubernetes y, particularmente, del servicio administrado Amazon EKS para facilitar la orquestación y escalabilidad. Sin embargo, estos estudios señalan que la configuración y operación de clústeres requiere de mecanismos adicionales de automatización para garantizar entornos consistentes. Por su parte, Ruiz Cabrera (2024) y Petrocelli (2021) resaltan las ventajas del uso de Infraestructura como Código (IaC) mediante herramientas como Terraform y CloudFormation, las cuales reducen la posibilidad de errores humanos y estandarizan los entornos.

No obstante, trabajos como los de Arrieta y Palacios (2022) y Real Ixcayau (2021) advierten que, pese a estas tecnologías, muchas organizaciones aún no logran integrar aprovisionamiento, despliegue, monitoreo y recuperación en un ciclo completamente automatizado. Esta brecha limita la capacidad de las empresas para asegurar la continuidad de servicios críticos y evidencia la necesidad de seguir investigando soluciones que consoliden IaC, EKS y metodologías DevOps/GitOps en un modelo integral.

1.2. Descripción del problema

Actualmente, las organizaciones se encuentran en un proceso continuo de migración de aplicaciones hacia la nube con el fin de aprovechar la flexibilidad, escalabilidad y disponibilidad que ofrecen plataformas como AWS (Mora, 2023; Moreno Villarrubia, 2024). Si bien esta transición tecnológica permite optimizar recursos y responder a las demandas del mercado (García Granados, 2023), aún persisten dificultades significativas relacionadas con la recuperación ante fallos, la continuidad operativa y la gestión eficiente de infraestructuras críticas.

Los procesos manuales de aprovisionamiento e implementación de aplicaciones suelen ser extensos, repetitivos y propensos a errores humanos, lo que incrementa los tiempos de recuperación y compromete la disponibilidad de los servicios (Ripa Elices, 2025; Salazar Merchán, 2022). En entornos de alta demanda, estas limitaciones se traducen en pérdidas económicas, riesgos reputacionales e interrupciones en operaciones sensibles.

Si bien la introducción de contenedores y orquestadores como Kubernetes, junto con el servicio administrado EKS de AWS, ha mejorado la escalabilidad y flexibilidad (Cesanelli, 2023; Escoda Gil, 2023), y las herramientas de IaC como Terraform o CloudFormation han facilitado la estandarización de recursos (Ruiz Cabrera, 2024; Petrocelli, 2021), muchas organizaciones aún no integran de manera plena estas tecnologías en un ciclo automatizado que abarque aprovisionamiento, despliegue, monitoreo y recuperación.

En consecuencia, surge la necesidad de diseñar una estrategia que combine la automatización integral de aplicaciones contenerizadas con prácticas de Infraestructura como Código (IaC) sobre EKS, que permita reducir los tiempos de recuperación, optimizar la eficiencia operativa y garantizar la continuidad de los servicios en la nube. Esta investigación se plantea como respuesta a esa necesidad, aportando un modelo que consolide metodologías DevOps y GitOps en una arquitectura resiliente y eficiente.

1.3. Planteamiento del problema

1.3.1. Planteamiento del problema general

Las organizaciones que operan sobre plataformas en la nube enfrentan el desafío de mantener la continuidad operativa de sus servicios frente a fallos imprevistos o interrupciones. Si bien tecnologías como los contenedores y servicios de orquestación administrados, entre ellos Amazon EKS, han facilitado la gestión de aplicaciones, muchas empresas aún carecen de procesos de automatización integral que aseguren una recuperación ágil y una administración eficiente de la infraestructura.

Esta carencia se traduce en tiempos de recuperación elevados, interrupciones del servicio y una disminución en la confiabilidad percibida por los usuarios. Ante esta situación, surge el siguiente problema general de investigación:

¿Cómo la automatización de aplicaciones contenerizadas con infraestructura bajo código en EKS reduce el tiempo de recuperación y garantiza la continuidad operativa sobre la nube de AWS?

1.3.2. Planteamiento del problema específico

- a) ¿Cómo influyen la identificación y selección de herramientas de Infraestructura como Código (IaC) en la automatización del aprovisionamiento de recursos en AWS y configuración de clústeres EKS?
- b) ¿Cómo el diseño del flujo de automatización (CI/CD) para el despliegue y gestión de contenedores en Amazon EKS influye en la capacidad de recuperación del sistema ante fallos simulados?
- c) ¿Cómo se ve afectado el tiempo de recuperación (MTTR) del modelo automatizado después de la implementación de un entorno de prueba con fallos controlados sobre AWS?
- d) ¿Cuál es el impacto en la continuidad operativa del modelo automatizado a través de la integración entre herramientas IaC y orquestadores como EKS?

1.4. Objetivos

1.4.1. Objetivo general

Automatizar aplicaciones contenerizadas con infraestructura bajo código en EKS para reducir el tiempo de recuperación y para garantizar la continuidad operativa sobre la nube de AWS

1.4.2. Objetivos específicos

- a) Identificar y seleccionar herramientas de Infraestructura como Código (IaC) para automatizar el aprovisionamiento de recursos en AWS y configuración de clústeres EKS.
- b) Diseñar flujos de automatización (CI/CD) para el despliegue y gestión de contenedores en Amazon EKS.
- c) Analizar el tiempo de recuperación (MTTR) del modelo automatizado después de la implementación de un entorno de prueba con fallos controlados sobre AWS
- d) Analizar el impacto en la continuidad operativa del modelo automatizado a través de la integración entre herramientas IaC y orquestadores como EKS

1.5. Justificación de la investigación

La presente investigación es importante y se justifica al abordar una problemática clave en los entornos modernos de servicios cloud: la reducción del tiempo de recuperación ante fallos y la garantía de continuidad operativa mediante la automatización de aplicaciones contenerizadas con infraestructura bajo código sobre AWS Elastic Kubernetes Service (EKS). En un contexto donde los servicios deben operar bajo esquemas de alta disponibilidad y mínima interrupción, la resiliencia operativa se convierte en un factor estratégico fundamental para las organizaciones (Mora, 2023; Moreno Villarrubia, 2024).

La automatización integral del ciclo de vida de las aplicaciones, basada en contenedores y orquestadores como Kubernetes bajo la modalidad de EKS, permite establecer entornos consistentes, auditables y fácilmente replicables (García Granados,

2023; Cesanelli, 2023). El uso de infraestructura como código (IaC), mediante herramientas como Terraform o AWS CloudFormation o CDK, posibilita la provisión automatizada y declarativa de recursos, minimizando errores humanos, reduciendo la variabilidad operativa y optimizando el tiempo de recuperación frente a eventos imprevistos (Ruiz Cabrera, 2024; Ripa Elices, 2025).

Desde la perspectiva operativa, la implementación de estas tecnologías permite a las organizaciones alcanzar mayores niveles de eficiencia, dado que los procesos automatizados reducen significativamente los tiempos de inactividad, incrementan la productividad de los equipos de TI y mejoran el cumplimiento de los acuerdos de nivel de servicio (SLA), aspectos señalados por Salazar Merchan (2022) y Arrieta y Palacios (2022) en sus respectivos estudios sobre elasticidad y monitoreo en la nube.

En el ámbito académico, este trabajo ofrece una contribución práctica al conocimiento sobre la automatización de aplicaciones contenerizadas en entornos cloud, al demostrar cómo la integración de contenedores, orquestación, IaC y prácticas DevOps puede aplicarse eficazmente para optimizar la resiliencia operativa (Real Ixcayau, 2021; Petrocelli, 2021). Asimismo, responde a una necesidad actual de investigación aplicada, alineándose con las tendencias emergentes de automatización y gestión inteligente de infraestructura en la nube (Varela Tabares, 2019).

Por tanto, esta investigación no solo aborda un desafío técnico de alta relevancia para las organizaciones modernas, sino que también aporta evidencia empírica y metodológica sobre la eficacia de las estrategias de automatización para garantizar la continuidad operativa, optimizar los tiempos de recuperación ante fallos y mejorar la gestión de aplicaciones contenerizadas sobre AWS EKS.

1.6. Viabilidad de la investigación

La investigación es viable técnica, operativa y económicamente. En el plano técnico, se cuenta con acceso a servicios de Amazon Web Services (AWS), incluyendo

Elastic Kubernetes Service (EKS), y herramientas de Infraestructura como Código como AWS CloudFormation, CDK. Estas tecnologías son ampliamente documentadas, accesibles para pruebas en entornos de laboratorio y permiten su implementación mediante cuentas personales o educativas en la nube.

Desde el punto de vista operativo, el investigador posee los conocimientos necesarios en tecnologías de contenedores, automatización, DevOps y arquitectura en la nube, lo cual facilita el diseño e implementación del modelo propuesto. Además, existen recursos de formación gratuitos o de bajo costo que permiten reforzar cualquier componente técnico requerido durante el desarrollo del proyecto.

En cuanto a la viabilidad económica, el uso de entornos de prueba con planes gratuitos o servicios con facturación bajo demanda en AWS permite controlar el presupuesto y garantizar la experimentación sin incurrir en gastos elevados. Asimismo, no se requieren grandes inversiones en infraestructura física, ya que todo el entorno puede ser simulado y probado en la nube.

Finalmente, la disponibilidad de bibliografía técnica, casos de estudio y documentación oficial de AWS y herramientas relacionadas asegura el soporte teórico-práctico necesario para el desarrollo de una investigación sólida y actualizada.

1.7. Delimitación del problema

Esta investigación se delimita en los siguientes aspectos:

- **Ámbito tecnológico:** El estudio se enfoca en el uso de contenedores, Kubernetes administrado a través de Amazon EKS, y herramientas de Infraestructura como Código (principalmente CloudFormation). No se considerarán otros orquestadores ni otros proveedores de nube como Azure o Google Cloud.
- **Entorno de aplicación:** La automatización se implementará sobre entornos empresariales dentro de la nube de AWS..
- **Temporalidad:** El proyecto se desarrollará durante el periodo académico correspondiente al año 2025.

- **Alcance del sistema:** El enfoque estará centrado en reducir el tiempo de recuperación ante fallos (RTO) y garantizar la continuidad operativa mediante la implementación de flujos de automatización en despliegue, recuperación y configuración de infraestructura contenerizada. No se incluirán temas como: gestión de base de datos, seguridad en la nube, costos avanzados ni benchmarking entre múltiples nubes.

Esta delimitación permite enfocar el esfuerzo investigativo en aspectos clave del problema, asegurando resultados concretos y aplicables en contextos reales, sin dispersarse en variables que excedan los objetivos del estudio.

1.8. Limitaciones del problema

La presente investigación se enfoca en el diseño y desarrollo de un modelo de automatización del despliegue y gestión de aplicaciones contenerizadas en Amazon EKS, aplicando principios de Infraestructura como Código (IaC) y prácticas de integración y despliegue continuo (CI/CD). Sin embargo, es importante precisar las siguientes limitaciones:

- Alcance tecnológico: El estudio se restringe al uso de servicios nativos de AWS (EKS, S3, RDS, Cognito, CodeCommit, CodePipeline, CodeBuild, entre otros). No se contempla la comparación con otras nubes públicas (Azure, Google Cloud) ni con soluciones on-premise.
- Enfoque en automatización: La propuesta aborda la automatización de la infraestructura y del pipeline de despliegue, sin incluir pruebas de rendimiento avanzado, benchmarking detallado de cargas de trabajo ni análisis de costos económicos a gran escala.
- Entornos de aplicación: El modelo se valida en entornos de desarrollo y pruebas controlados. No se realizan despliegues en un ambiente productivo real de una organización específica, por lo que los resultados se interpretan en términos de viabilidad técnica.
- Alcance de seguridad: Se consideran lineamientos básicos de autenticación, permisos de IAM y cifrado con KMS, sin profundizar en normativas de cumplimiento específicas (ej. GDPR, HIPAA, ISO/IEC 27001).

- Tiempo y recursos: La implementación práctica se limita a los recursos disponibles para la investigación, por lo que no se incluye un monitoreo longitudinal de la solución en operación prolongada.

1.9. Hipótesis de la investigación

1.9.1. Hipótesis general

La automatización de aplicaciones contenerizadas con infraestructura bajo código en Amazon EKS sí reduce significativamente el tiempo de recuperación y garantiza la continuidad operativa sobre la nube de AWS.

1.9.2. Hipótesis específicas

1ra Hipótesis específica

La identificación y selección de herramientas de Infraestructura como Código (IaC) permite automatizar eficazmente el aprovisionamiento de recursos en AWS y configuración de clústeres EKS, disminuyendo la intervención manual.

2da Hipótesis específica

El diseño del flujo de automatización (CI/CD) para el despliegue y gestión de contenedores en Amazon EKS mejora la capacidad de recuperación del sistema ante fallos simulados.

3ra Hipótesis específica

El tiempo de recuperación (MTTR) del modelo automatizado se reduce en comparación con procesos manuales después de la implementación de un entorno de prueba con fallos controlados sobre AWS.

4ta Hipótesis específica

La continuidad operativa del modelo automatizado a través de la integración entre herramientas IaC y orquestadores como EKS tiene un impacto positivo.

1.10. Variables de la investigación

1.10.1. Variable independiente

V.I.: Automatización de aplicaciones contenerizadas mediante infraestructura bajo código en EKS

Esta variable hace referencia a la propuesta de procesos automatizados que integran herramientas como AWS CloudFormation o Terraform para la provisión y configuración de recursos, así como para el despliegue de contenedores en clústeres de Amazon EKS. Incluye el diseño de scripts IaC, pipelines de integración y flujos automatizados orientados a reducir la intervención manual en la gestión de la infraestructura y las aplicaciones.

Según Mora (2010), la variable independiente se entiende como aquella que, dentro de una investigación, constituye el factor propuesto o introducido para analizar su influencia en otras variables del estudio. En este caso, la automatización con infraestructura como código se plantea como el elemento que, en el diseño, explica los posibles cambios en el tiempo de recuperación y en la continuidad operativa.

1.10.2. Variable dependiente

V.D.: Reducción de tiempo de recuperación y garantización de la continuidad operativa sobre la nube de AWS

Esta variable representa el conjunto de indicadores conceptuales de rendimiento del sistema, orientados a evaluar la rapidez con la que los servicios podrían restablecerse (Recovery Time Objective – RTO) y el nivel de disponibilidad continua de los servicios en la nube (uptime). En esta investigación, dichos indicadores se analizan de manera referencial y documental, a partir del diseño elaborado y de la evidencia técnica proveniente de AWS y Kubernetes, contrastando el escenario manual inicial con el modelo de automatización propuesto.

De acuerdo con Hernández Sampieri et al. (2014), la variable dependiente corresponde al resultado o efecto esperado a partir de la introducción de la variable independiente (p.131). En este caso, el tiempo de recuperación y la continuidad operativa

constituyen los efectos que conceptualmente se esperan mejorar mediante la automatización planteada.

1.10.3. Operacionalización de la variable Independiente

V.I.: Automatización de aplicaciones contenerizadas mediante infraestructura bajo código en EKS

Definición conceptual: Es la aplicación de herramientas que permiten definir, aprovisionar y configurar recursos en la nube a través de código, integrando el despliegue de aplicaciones contenerizadas de manera automatizada, reproducible y escalable.

Definición operacional: Se representa mediante el diseño técnico elaborado con herramientas de infraestructura como código (ej. Terraform o CloudFormation), en el que se especifica la creación de clústeres en Amazon EKS y el despliegue de servicios contenerizados de forma automática. La validación se realizó a través de la revisión documental y técnica de scripts, manifiestos y registros conceptuales que muestran cómo el modelo propuesto logra la automatización.

Tabla 1.

Operacionalización de la variable Independiente: Automatización de aplicaciones contenerizadas con infraestructura bajo código EKS

Dimensiones	Indicadores	Técnica	Instrumento	Escala de medición
Automatización de Recursos	Nivel de Intervención manual	Análisis técnico–comparativo	Matriz comparativa de procesos manuales vs. automatizados	Cualitativa-Ordinal
	Reproducibilidad del aprovisionamiento IaC	Revisión documental y técnica	Documentación de AWS / Logs de EKS	Cualitativa Nominal
Automatización de despliegue	Cobertura del pipeline automatizado	Análisis de scripts y configuración	Definición del pipeline (Helm/Manifest/YAML)	Cualitativa-Ordinal
	Verificación de despliegue automático en validación	Observación técnica del diseño	Consola de pipeline (CloudWatch, EKS)	Cualitativa Nominal

1.10.4. Operacionalización de la variable Dependiente

V.D.: Reducción de tiempo de recuperación y garantía de la continuidad operativa sobre la nube de AWS

Definición conceptual: Se entiende como la capacidad de un sistema desplegado en la nube de recuperarse rápidamente ante una falla (Recovery Time Objective – RTO) y de mantener un funcionamiento continuo sin interrupciones críticas, garantizando la disponibilidad del servicio.

Definición operacional: En esta investigación se representa mediante los indicadores técnicos establecidos en la literatura y en la propuesta de diseño, tales como:

- El tiempo estimado (en segundos/minutos) requerido para restablecer un servicio tras una interrupción.
- El porcentaje de disponibilidad proyectada del sistema durante la operación en la nube.

Estos indicadores no fueron medidos empíricamente en múltiples pruebas, sino documentados y validados conceptualmente a partir del diseño elaborado y de la evidencia técnica de Amazon Web Services y Kubernetes, lo que permite contrastar el escenario manual inicial con el modelo automatizado.

Tabla 2.

Operacionalización de la variable Dependiente: Reducción del tiempo de recuperación y garantía de la continuidad operativa

Dimensiones	Indicadores	Técnica	Instrumento	Escala de medición
Tiempo de recuperación	Duración de la caída hasta restauración	Análisis técnico-documental	Registros y documentación de AWS / Logs de EKS	Cuantitativa (minutos)
	Comparación entre tiempo manual vs automatizado	Análisis comparativo del diseño	Matriz de validación técnica / Bitácora conceptual	Cuantitativa (minutos y %)
Continuidad Operativa (Servicio)	Porcentaje de disponibilidad del servicio	Revisión documental y técnica	Panel CloudWatch / Documentación de AWS	Cuantitativa (referencial)
	Número de fallos críticos durante pruebas	Observación técnica de la propuesta	Reporte técnico de validación	Cuantitativa (referencial)

CAPÍTULO II

MARCO TEÓRICO

2.1. Antecedentes de la investigación

Ripa Elices (2025) desarrolló un trabajo de investigación titulado "Aplicación de metodologías GitOps en el despliegue de aplicaciones distribuidas", en el cual aplicó metodologías GitOps para la gestión y despliegue de aplicaciones contenerizadas en entornos cloud, utilizando Kubernetes como orquestador principal y AWS EKS como plataforma de implementación. El objetivo de la investigación fue analizar cómo las prácticas GitOps pueden potenciar la eficiencia en el manejo de aplicaciones distribuidas, garantizando trazabilidad, automatización y consistencia entre el código fuente y la infraestructura desplegada. La metodología empleada incluyó el desarrollo de dos prototipos: uno basado en blockchain y otro en microservicios. Se utilizaron herramientas como: Flask-RESTx, Swagger, ConfigMaps, AWS EFS, ArgoCD, y la integración de pipelines CI/CD con GitHub Actions. Los resultados evidenciaron que el uso de GitOps permite un control declarativo robusto sobre los entornos de producción, mejorando la eficiencia operativa, la escalabilidad y reduciendo los errores humanos en los despliegues. La aplicación de GitOps asegura una sincronización continua entre los repositorios de configuración y el estado real de la infraestructura. Este trabajo resulta altamente relevante para el presente estudio, ya que sustenta la utilización de prácticas declarativas y automatización de despliegues sobre AWS EKS, pilares fundamentales para reducir el tiempo de recuperación y garantizar la continuidad operativa (Ripa Elices, 2025).

Ruiz Cabrera (2024) llevó a cabo la investigación titulada "Framework de Automatización FinOps: Mejorando la eficiencia de Costos en Despliegues de Cloud", donde propuso un modelo automatizado orientado a optimizar los costos operativos en infraestructura cloud, integrando criterios técnicos y financieros. El objetivo fue desarrollar un framework FinOps que permitiera el ajuste dinámico de recursos cloud en función del uso y las finanzas corporativas.

La metodología consistió en la utilización de Terraform para definir la infraestructura como código, Azure como proveedor de nube, Azure Functions y Monitor para el monitoreo automatizado, así como GitHub Actions para gestionar los pipelines de integración continua y despliegue continuo (CI/CD). Los resultados mostraron que el framework posibilitó un escalado dinámico eficiente, optimizando tanto el rendimiento técnico como los costos económicos, mediante decisiones automatizadas basadas en métricas de uso y límites presupuestarios. Aunque centrado en Azure, este trabajo aporta un marco conceptual importante para la presente investigación, pues sus principios de automatización, control financiero y ajuste proactivo de recursos son extrapolables al entorno de AWS EKS en la automatización de aplicaciones contenerizadas (Ruiz Cabrera, 2024).

Moreno Villarrubia (2024) presentó la investigación "Desarrollo de un Sistema Integral para la Monitorización Inteligente de Smart Villages", donde diseñó e implementó un sistema de monitoreo inteligente basado en contenedores para la supervisión de recursos urbanos en tiempo real. El objetivo fue establecer una solución contenerizada, escalable y eficiente para la captación y procesamiento de datos provenientes de sensores y dispositivos IoT en entornos Smart Village. La metodología incluyó el diseño de dashboards para visualización de datos, la contenerización de servicios desacoplados, el uso de software libre y la orquestación mediante Kubernetes, garantizando alta disponibilidad y resiliencia frente a fallos. Los resultados evidenciaron que la implementación contenerizada permitió gestionar grandes volúmenes de datos en tiempo real, facilitando la toma de decisiones eficientes en la administración de servicios urbanos. Esta investigación respalda el enfoque del presente estudio al demostrar la capacidad de Kubernetes y los contenedores para gestionar entornos altamente disponibles, flexibles y resilientes, elementos claves en la automatización planteada sobre AWS EKS (Moreno Villarrubia, 2024).

Escoda Gil (2023) realizó el trabajo titulado "Diseño y desarrollo de una arquitectura distribuida para SmartUniversity", orientado al diseño de un clúster de bases de datos relacionales contenerizadas con alta disponibilidad. El objetivo fue crear un entorno distribuido capaz de ofrecer servicios de bases de datos escalables, replicados y balanceados bajo Kubernetes. La metodología consistió en desplegar pods de bases de datos usando el operador Percona para replicación, balanceo de carga mediante HAProxy

bajo el algoritmo Round Robin, y gestión de nodos con Rancher. Además, se integraron sistemas de monitorización para facilitar la administración proactiva del rendimiento. Los resultados indicaron que el clúster diseñado garantizó la disponibilidad continua de los servicios de base de datos, manteniendo integridad ACID y resiliencia frente a fallos de nodos. Este trabajo contribuye al presente estudio al mostrar la aplicación práctica de Kubernetes para lograr alta disponibilidad y escalabilidad, principios fundamentales en la automatización contenerizada sobre AWS EKS (Escoda Gil, 2023).

García Granados (2023) desarrolló la investigación "Despliegue de aplicaciones contenerizadas en entornos cloud", en la cual analizó la eficiencia de plataformas de contenerización para despliegues empresariales. El objetivo fue comparar las capacidades de Kubernetes y OpenShift en términos de escalabilidad, flexibilidad y costos operativos. La metodología contempló el desarrollo de una aplicación web en Node.js desplegada sobre las tres plataformas, realizando pruebas de rendimiento, adaptabilidad y consumo de recursos. Los resultados concluyeron que Kubernetes ofrece el mayor grado de control, orquestación y adaptación a necesidades empresariales variables, posicionándose como la solución estándar en la industria. Este antecedente respalda la elección de Kubernetes y AWS EKS en el presente trabajo, confirmando su idoneidad técnica y operativa para automatización de despliegues contenerizados (García Granados, 2023).

Mora (2023) ejecutó la investigación "Migración a una arquitectura en la nube para el procesamiento de datos abiertos oceanográficos", orientada a modernizar sistemas de datos bajo AWS. El objetivo fue trasladar la infraestructura on-premise a un entorno cloud que garantizara alta disponibilidad, escalabilidad y eficiencia de costos. La metodología aplicada incluyó el uso de arquitectura ADM, automatización de pipelines CI/CD, infraestructura como código, políticas de seguridad, y pruebas de validación de calidad bajo principios DevOps. Los resultados mostraron una transición exitosa hacia AWS, mejorando sustancialmente la capacidad de procesamiento de datos oceanográficos y reduciendo los costos operativos. Este estudio resulta pertinente para la presente investigación al demostrar los beneficios de migrar arquitecturas complejas hacia AWS mediante automatización e infraestructura bajo código, fundamentos esenciales del trabajo actual (Mora, 2023).

Cesanelli (2023) presentó la investigación titulada "Migración de una arquitectura basada en contenedores usando ECS a Kubernetes", centrada en realizar la transición de

aplicaciones contenerizadas de Amazon ECS hacia AWS EKS. El objetivo fue aprovechar las ventajas de Kubernetes en cuanto a flexibilidad, control de recursos y escalabilidad sin necesidad de reconstruir las imágenes de los contenedores existentes. La metodología consistió en migrar los servicios mediante infraestructura como código, configurando Kubernetes como orquestador principal sobre AWS, realizando pruebas de integración, gestión de servicios y validación funcional. Los resultados demostraron la viabilidad de migrar infraestructuras sin afectar la continuidad de los servicios, maximizando la reutilización de componentes y optimizando el control sobre los recursos asignados. Este antecedente es relevante ya que valida la adopción de AWS EKS como plataforma robusta para automatizar el despliegue de aplicaciones contenerizadas, tal como se propone en el presente trabajo (Cesanelli, 2023).

Salazar Merchan (2022) desarrolló el trabajo titulado "Despliegue automático de un cluster de Kubernetes y monitorización web", en el cual propuso la creación de un clúster de Kubernetes automatizado para el despliegue de servicios y su monitorización web. El objetivo fue implementar un sistema automatizado de despliegue y observabilidad de servicios contenerizados bajo Kubernetes. La metodología incluyó la creación del clúster desde cero, instalación de servicios, desarrollo de un frontend de monitoreo y configuración de métricas a través de la API de Kubernetes. Los resultados permitieron validar la administración automatizada de servicios y métricas del clúster, generando una experiencia práctica en herramientas de automatización y administración de infraestructura bajo principios DevOps. Este trabajo aporta fundamentos operativos relevantes para la presente investigación, especialmente en lo referente a la automatización de despliegues y la monitorización proactiva de servicios contenerizados sobre AWS EKS (Salazar Merchan, 2022).

Arrieta y Palacios (2022) realizaron la investigación titulada "Metodología TI para monitorear y mejorar el rendimiento y disponibilidad de la información en la nube", orientada al diseño de una metodología elástica de TI bajo principios DevOps y Kubernetes. El objetivo fue desarrollar un esquema de autoescalamiento automatizado para garantizar la disponibilidad de servicios cloud. La metodología empleada integró herramientas como Cluster Autoscaler, AWS CloudWatch, SNS para alertas, y exploró el uso de Karpenter en AWS como solución avanzada de autoescalado. Los resultados mostraron que la elasticidad automatizada permite aprovisionar nodos adicionales cuando

los recursos de los nodos existentes son insuficientes, mejorando el rendimiento y la protección de los servicios. Este trabajo es coherente con los objetivos de la presente investigación al fortalecer los mecanismos de autoescalado y disponibilidad automatizada sobre AWS EKS (Arrieta & Palacios, 2022).

Real Ixcayau (2021) desarrolló el trabajo "Propuesta para actualización curricular en el área de software incorporando talleres de Docker y Kubernetes", donde propuso incorporar contenidos de contenerización y automatización en programas de formación. El objetivo fue actualizar los planes de estudio para preparar profesionales capacitados en tecnologías de cloud computing, DevOps, CI/CD y orquestación de contenedores. La metodología incluyó un análisis curricular, entrevistas a expertos y revisión de literatura técnica. Como resultado, se estableció la relevancia creciente de Kubernetes y Docker como tecnologías indispensables en entornos empresariales modernos. Este antecedente sustenta teóricamente el marco tecnológico de la presente investigación, al validar la necesidad formativa en las competencias tecnológicas empleadas (Real Ixcayau, 2021).

Petrocelli (2021) realizó la investigación titulada "Plataforma colaborativa, distribuida, escalable y de bajo costo basada en microservicios, contenedores, dispositivos móviles y servicios en la Nube para tareas de cómputo intensivo", donde propuso un modelo de cómputo intensivo distribuido, reutilizando la capacidad ociosa de dispositivos móviles ARM y arquitecturas cloud mediante Kubernetes. El objetivo fue desarrollar una plataforma escalable y eficiente energéticamente para procesamiento distribuido, integrando DevOps, microservicios y contenedores. La metodología consistió en el diseño de arquitectura híbrida cloud-local, automatización de pruebas, monitoreo, integración continua y escalado dinámico. Los resultados demostraron la robustez de la plataforma ante cargas variables, garantizando eficiencia operativa y reducción de costos. Aunque el contexto de Petrocelli (2021) es distinto, los principios de orquestación de contenedores, automatización y escalabilidad son aplicables a la automatización planteada sobre AWS EKS en la presente investigación.

Varela Tabares (2019) desarrolló el trabajo titulado "Diseño e implementación de un flujo de trabajo bioinformático en la nube para la identificación de Variantes oncogénicas a partir de datos genómicos", donde diseñó un flujo de trabajo contenerizado sobre AWS para análisis genómico. El objetivo fue establecer un flujo bioinformático eficiente mediante contenerización de servicios y uso de AWS ECS, EC2 y EFS. La

metodología incluyó diseño de procesos ETL, integración de Isabl para manejo de metadatos y procesamiento cloud de alto rendimiento. Los resultados validaron la efectividad del modelo cloud bajo demanda para cargas de cómputo intensivo, reduciendo los costos frente a infraestructura tradicional. Este trabajo complementa la presente investigación al confirmar la viabilidad operativa y económica de las arquitecturas contenerizadas en AWS para procesos de alta demanda computacional (Varela Tabares, 2019).

2.2. Bases teóricas de la Automatización de infraestructura en la nube

2.2.1. Principios de automatización en entornos cloud

La automatización en entornos de nube busca reemplazar tareas manuales repetitivas por procesos controlados por código, lo que mejora la eficiencia operativa, disminuye errores humanos y acelera la recuperación tras fallos (Pandugula & Syed, 2024).

La práctica de gestionar infraestructura a través de código conocida como Infrastructure as Code (IaC) se sustenta en principios clave:

2.2.1.1. Declarativo vs. Imperativo

El enfoque *declarativo* (usado por Terraform o CloudFormation) describe el estado deseado de la infraestructura, mientras que el sistema se encarga de alcanzar ese estado.

El enfoque *imperativo*, en cambio, requiere instruir paso a paso cómo lograrlo. El método declarativo favorece redundancia mínima, reproducibilidad y mejor gestión de cambios.

2.2.1.2. Idempotencia

Cada ejecución del script debe dejar el entorno en el mismo estado, sin importar cuántas veces se aplique. Este principio evita la deriva de configuración y permite revertir cambios con seguridad.

2.2.1.3.Control de versiones y colaboración

Los archivos que definen la infraestructura se gestionan en repositorios como Git, lo que facilita auditoría, retroceso de cambios, y trabajo colaborativo, integrando IaC dentro del flujo CI/CD.

2.2.1.4.Modularidad y reutilización

IaC incentiva la separación de componentes (como redes, roles, clústeres) en módulos independientes, para facilitar la reutilización, mantenimiento y escalabilidad del entorno.

2.2.1.5.Automatización continua en CI/CD

Integrar IaC en pipelines permite la creación, validación y despliegue automático de entornos de prueba, staging y producción, alineados con prácticas DevOps. Esto asegura consistencia, agilidad y una recuperación más rápida al estandarizar procesos.

2.2.2. Contribución a la reducción del RTO y continuidad operativa

Al aplicar estos principios, la infraestructura puede desplegarse o recuperarse automáticamente en cuestión de minutos. Esto representa una mejora sustancial frente a procesos manuales, donde errores o demoras pueden extender el RTO críticamente. Además, las prácticas declarativas y versiones controladas permiten reconstruir entornos predecibles y auditablemente seguros tras un fallo, fortaleciendo la continuidad operacional.

2.2.3. Infraestructura como Código (IaC): definición y ventajas

La Infraestructura como Código (IaC) es una metodología que permite gestionar y aprovisionar recursos de infraestructura a través de archivos de configuración, en lugar de hacerlo manualmente a través de interfaces gráficas o consolas de administración. Esta práctica convierte la infraestructura en parte integral del ciclo de vida del software, almacenando las definiciones en sistemas de control de versiones, lo que permite aplicar principios de desarrollo como revisión, pruebas y despliegue continuo (HashiCorp, 2023; IBM, 2021).

2.2.4. Ventajas de IaC

a) Repetibilidad y consistencia

Definir la infraestructura como código garantiza que cada despliegue sea idéntico y elimina discrepancias entre entornos (producción, staging, desarrollo). Esto reduce el riesgo de configuraciones erróneas y deriva operativa (Spacelift, 2025; Microsoft, s.f.)

b) Reducción de errores humanos y costo operativo

La automatización de tareas complejas y repetitivas disminuye las posibilidades de errores manuales, agiliza los despliegues y reduce el costo en tiempo y recursos (Spacelift, 2025; DevOps.com, 2024)

c) Escalabilidad y eficiencia

IaC facilita la creación y eliminación de entornos de forma rápida y eficiente. Esto permite ajustar la infraestructura automáticamente según la demanda, evitando desperdicio de recursos y optimizando el rendimiento (IBM, 2021; Spacelift, 2025).

d) Versionado y trazabilidad

Al codificar la infraestructura, se puede almacenar en repositorios como Git, lo que permite auditar cambios, revertir configuraciones y mantener un historial de modificaciones para control de cambios y cumplimiento (HashiCorp, 2023; Terramate, 2024)

e) Documentación y modularidad

El código IaC funciona como documentación viva del sistema. La estructuración en módulos mejora la claridad, reutilización y mantenibilidad de la infraestructura, agilizando la colaboración entre los equipos (Spacelift, 2025; Terramate, 2024).

f) Integración con CI/CD

IaC se integra con pipelines de integración continua, lo que permite validar y desplegar entornos automáticamente tras cada cambio en el código. Este enfoque aumenta la eficiencia, la fiabilidad y reduce significativamente el Recovery Time Objective (RTO) en escenarios de recuperación (Microsoft, s.f.; IBM, 2021)

2.2.5. Terraform: automatización declarativa para AWS

Terraform es una herramienta de código abierto desarrollada por HashiCorp que permite definir, aprovisionar y gestionar recursos de infraestructura mediante archivos de configuración escritos en HashiCorp Configuration Language (HCL), y no a través de acciones manuales en la consola. Al emplear un enfoque declarativo, Terraform describe el estado deseado de la infraestructura, y se encarga de determinar y ejecutar los cambios necesarios para alcanzarlo.

Entre los principales beneficios al usar Terraform con AWS se encuentran:

2.2.5.1. Compatibilidad multi-nube y agnóstico al proveedor

Terraform puede gestionar recursos tanto en AWS como en otras plataformas (Azure, GCP), utilizando el mismo lenguaje de configuración. Esto permite mantener un único conjunto de definiciones y facilita la provisión en arquitecturas híbridas o multicloud

2.2.5.2. Gestión de estado e idempotencia

Una de sus funciones centrales es mantener un archivo de estado que registra los recursos provisionados. Esto permite a Terraform identificar cambios, aplicar sólo las diferencias necesarias y garantizar que cada ejecución deje la infraestructura en un estado coherente, reforzando la repetibilidad.

2.2.5.3. Modularidad y reutilización

Terraform admite la creación de "módulos", lo que posibilita abstraer componentes de infraestructura –como redes o clústeres– y reutilizarlos con diferentes variables según el entorno. Esto favorece la estandarización, el mantenimiento y la escalabilidad del código de infraestructura

2.2.5.4. Accesibilidad y legibilidad

El uso de HCL permite redactar configuraciones que son fáciles de leer y escribir. Esto facilita que equipos múltiples puedan colaborar en los archivos de infraestructura, documentar su lógica y versionarla como cualquier otro proyecto de software .

2.2.5.5.Prevencción de errores y optimización del tiempo

Con el flujo terraform init → plan → apply, se ofrece una vista previa de los cambios y se validan antes de aplicar modificaciones, reduciendo el riesgo de acciones no deseadas y mejorando la velocidad de implementación

2.2.6. Uso de CloudFormation en pipelines (referencial)

AWS CloudFormation es un servicio de Infraestructura como Código que permite definir y gestionar recursos de la nube mediante plantillas escritas en JSON o YAML . Aunque en este proyecto se utiliza Terraform como herramienta principal, CloudFormation se incluye aquí como una referencia importante por su integración nativa con los servicios de CI/CD de AWS.

a) Integración con CI/CD

Cuando se integran plantillas de CloudFormation en pipelines, estas pueden:

- Crear o actualizar infraestructuras de prueba automáticamente al detectarse cambios en el repositorio, usando AWS CodePipeline, CodeBuild y CloudFormation
- Generar "Change Sets", vistas previas de los cambios que se aplicarán, lo que permite revisar y aprobar modificaciones antes de ejecutarlas, reduciendo riesgos .
- Aplicar validaciones automáticas mediante herramientas como cfn-guard, que verifican cumplimiento de políticas, estándares o restricciones corporativas antes del despliegue

b) Buenas prácticas en el pipeline

Las guías de AWS recomiendan:

- **Segregar ambientes por cuenta o pipeline**, para mantener entornos separados y seguros

- **Revisar código mediante pull requests** antes de fusionar al main branch, aplicando tests y validaciones automatizadas
- **Limitar la administración manual** de infraestructuras en producción, delegando casi todas las acciones a través de CloudFormation y roles con mínimos privilegios

2.3. Arquitectura base: EKS y clúster de Kubernetes

2.3.1. Introducción a Kubernetes y sus componentes principales

Kubernetes es una plataforma de orquestación de contenedores de código abierto diseñada para automatizar el despliegue, escalado y gestión de aplicaciones contenerizadas (Kubernetes, 2024)

Su arquitectura se basa en dos planos principales: el control plane (plano de control) y el data plane (plano de datos). El primero gestiona el estado deseado del clúster, mientras que el segundo ejecuta los contenedores reales en máquinas o instancias virtuales (Spot.io, 2025).

a) Plano de control (Control Plane)

Este plano agrupa componentes críticos:

- API Server: Punto de entrada mediante el que los usuarios, operadores y componentes del sistema interactúan con el clúster, validando solicitudes y orquestando el estado deseado (Kubernetes.io, 2025)
- etcd: Base de datos distribuida que almacena todo el estado del clúster. Su evento persistente es clave para la tolerancia a fallos (Appvia, s.f.; Kubernetes.io, 2025)
- Controller Manager: Ejecuta controladores que se aseguran de que el estado actual coincida con el estado deseado (p. ej., ReplicaSets, DaemonSets) (Wikipedia ruso, 2024; Spot.io, 2025)
- Scheduler: Asigna pods a nodos según políticas de recursos, afinidades y disponibilidad (Wikipedia ruso, 2024; Kubernetes.io, 2025)

b) Plano de datos (Worker Nodes)

Las instancias donde se ejecutan las aplicaciones contienen:

- kubelet: Agente que garantiza que los pods esperados estén en ejecución, comunicándose con el API Server
- Container runtime: Motor de contenedores (como containerd) que ejecuta el software dentro del pod
- kube-proxy: Administra reglas de tráfico (iptables) para servicios y networking

c) Objetos fundamentales de Kubernetes

- Pods: Unidades básicas de ejecución que consolidan uno o varios contenedores que comparten red y almacenamiento
- Services: Abstracción para descubrir y equilibrar tráfico hacia pods, asignándoles direcciones IP y nombres DNS internos
- ReplicaSets y Deployments: Gestionan la cantidad deseada de réplicas y permiten actualizaciones controladas (Spot.io, 2025)

2.3.2. Amazon EKS

Amazon EKS proporciona un plano de control administrado, replicado en múltiples Availability Zones para asegurar disponibilidad y resiliencia (Amazon, 2024; Lumigo, 2024)

AWS garantiza la provisión, el parcheo y la alta disponibilidad del plano de control, permitiendo al usuario concentrarse en la gestión de nodos y cargas de trabajo.

Al integrarse con herramientas como EKSCtl y Zeus, el despliegue de clústeres se puede automatizar, para reducir el RTO y garantizar continuidad.

2.3.3. Amazon EKS como orquestador administrado

Amazon Elastic Kubernetes Service (EKS) es una plataforma administrada que facilita el despliegue y uso de Kubernetes sin necesidad de gestionar el plano de control manualmente. Este servicio ejecuta la versión upstream de Kubernetes y es totalmente

compatible con la comunidad CNCF, lo que permite migrar aplicaciones existentes sin modificaciones

a) Alta disponibilidad y tolerancia a fallos

EKS mantiene automáticamente la disponibilidad del plano de control mediante la replicación de componentes críticos en múltiples Zonas de Disponibilidad (AZ), incluyendo servidores de API y bases etc. Implementa un mínimo de dos nodos de API y tres de etc, replicados en diferentes AZs para eliminar puntos únicos de falla Además, controla la salud del plano y lo repara o escala según la carga .

b) Simplificación operativa

Amazon EKS libera al equipo de modernización de la infraestructura al encargarse AWS del aprovisionamiento, mantenimiento y actualizaciones del plano de control, optimizando así el enfoque en cargas de trabajo y lógicas de negocio . También ofrece integración nativa con servicios como VPC, IAM, ELB y CloudTrail para asegurar redes, gestión de identidades, balanceo y auditoría .

c) Escalabilidad y flexibilidad

EKS facilita el dimensionamiento de recursos, permitiendo agregar o quitar nodos de trabajo de forma manual o automática (Auto Scaling Groups, Fargate, EKS Auto Mode) . Su compatibilidad con arquitecturas Multi-AZ garantiza la resiliencia y continuidad ante fallos geográficos . Además, el modo serverless con Fargate simplifica aún más la operación al eliminar la gestión de servidores

d) Seguridad y gobernanza

EKS se conecta con IAM para proporcionar control granular de acceso (RBAC) y cifrado de datos en reposo y tránsito. También habilita redes privadas y políticas de VPC-CNI para mantener las comunicaciones internamente controladas.

2.3.4. Control de acceso con AccessEntry y clúster privado

El control de acceso en Amazon EKS se gestiona mediante la función **AccessEntry**, que elimina la necesidad de manipular directamente el `aws-auth ConfigMap`, facilitando una administración más segura y coherente basada en IAM .

Una **AccessEntry** asocia un principal IAM (usuario o rol) con permisos definidos para el clúster, ya sea mediante políticas preconfiguradas (como `AmazonEKSClusterAdminPolicy`) o asignándolo a grupos específicos de Kubernetes como `system:masters`.

Ventajas de AccessEntry

- **Gestión centralizada de permisos** mediante IAM, evitando discrepancias entre configuraciones de acceso (API Server vs. IAM) .
- **Granularidad en la autorización**, permitiendo asignar un access entry estándar o adaptado mediante Kubernetes, sin necesidad de editar manualmente mapas de configuración. Esto habilita una gestión alineada con políticas de “menor privilegio” y auditoría consolidada (CloudTrail) .

a) Clúster privado y seguridad de red

En un clúster EKS privado:

- El plano de control no está expuesto públicamente y responde únicamente a través de endpoints privados dentro de la VPC .
- La comunicación entre los nodos y el plano de control se realiza a través de interfaces ENI creadas en las subredes privadas, lo que refuerza la separación de redes y garantiza una alta seguridad .

b) Buenas prácticas de acceso

Las guías oficiales recomiendan:

- Deshabilitar el acceso público o restringirlo a IPs seguras.
- Eliminar permisos excesivos del creador del clúster para evitar riesgos de configuración no autorizada, delegando accesos explícitos mediante AccessEntry .
- Aplicar autenticación por roles IAM, canalizando el acceso a través de flujos de `assume-role` y RBAC interno de Kubernetes .

2.3.5. Etiquetado de recursos (VPC, Subnet, KMS) para automatización

El etiquetado de recursos (tagging) es una práctica esencial en la gestión de infraestructura en la nube. Consiste en asignar pares clave–valor a elementos como VPC, subnets y llaves KMS, lo que posibilita automatizar procesos, controlar costos y mejorar la gobernanza (AWS, 2022).

1º. Finalidad del etiquetado en IaC

- **Identificación y clasificación**

Las etiquetas permiten organizar recursos según su propósito (ambiente, aplicación, equipo, propietario), lo que facilita su gestión automatizada. Por ejemplo, con AWS Systems Manager se pueden ejecutar runbooks específicos aplicados únicamente a recursos etiquetados de cierto tipo.

- **Automatización basada en etiquetas**

Al aplicar etiquetas coherentes y estandarizadas, scripts (como Lambda o Terraform) pueden filtrar recursos para procesos automáticos: eliminación, backup, escalado o facturación.

- **Control de costos y informes**

Los tags permiten segmentar el gasto por entorno o proyecto. AWS recomienda definir una estrategia de etiquetas que incluya claves obligatorias y nombrado consistente.

2º. Recursos específicos y su etiquetado

- **VPC y Subnets**

Se debe asignar etiquetas como Environment, Project, CreatedBy y Purpose. Estas permiten aplicar políticas automatizadas, como la ejecución de backups de redes o limpieza nocturno.

- **KMS (llaves de cifrado):**

Las llaves deben etiquetarse para facilitar su uso por recursos específicos en un VPC y permitir auditoría mediante CloudTrail. También posibilitan aplicar restricciones condicionales basadas en etiquetas de endpoint.

3°. Implementación automatizada con IaC

Para conseguir automatización total, se puede usar Terraform u otro IaC como CloudFormation para definir las etiquetas desde la provisión inicial. Además, es habitual crear una función Lambda desencadenada por eventos (CloudTrail/EventBridge) que verifique etiquetas en recursos nuevos y aplique valores predeterminados cuando falten.

4°. Buenas prácticas recomendadas

- **Automatizar etiquetado con políticas centralizadas**

Se recomienda usar AWS Config o tag policies en AWS Organizations para garantizar coherencia entre cuentas.

- **Definir un esquema de nomenclatura estable**

Usa convenciones como `key: value` (ej. `team: DevOps, env: prod`) para mantener claridad y consistencia.

- **Herencia automática de etiquetas**

Los scripts pueden propagar etiquetas de VPC o subnets a recursos dependientes, como ENI, buckets S3 o instancias EC2 .

2.4. Contenerización y gestión de imágenes

2.4.1. Creación de imágenes y automatización en Bastion

En el proyecto, la instancia *bastion host* (EC2 Amazon Linux ARM64) sirve como entorno central para la construcción y gestión de imágenes, lo que permite aplicar contenedores estándar en Kubernetes. es la tecnología que facilita empaquetar aplicaciones con sus dependencias en imágenes reproducibles.

- **Construcción de imágenes en el Bastion**

Se inicia la construcción de imágenes en la instancia bastion para crear imágenes base que incluyan las herramientas necesarias, como AWS CLI, `kubectl`, `eksctl` y scripts de automatización. Este enfoque garantiza que el entorno de construcción sea consistente y controlado. AWS promueve prácticas como el uso de EC2 Image Builder, que automatiza la construcción y actualización de imágenes base, integrándose con ECR para desplegarlas automáticamente

- **Automatización del flujo de creación de imágenes y despliegue**

El proceso de automatización del flujo de creación de imágenes se gestiona desde el bastion, utilizando infraestructura como código a través de CloudFormation y scripts personalizados, garantizando procesos repetibles, auditables y confiables. Este flujo automatizado abarca desde la construcción de la imagen hasta su despliegue en el clúster de Kubernetes (EKS). AWS recomienda orquestar este tipo de flujos mediante herramientas como EC2 Image Builder en combinación con CloudFormation, con el objetivo de generar imágenes seguras, actualizadas y estandarizadas.

El flujo automatizado contempla las siguientes etapas:

1º. Preparación del entorno CodeBuild:

Se inicializa el entorno de compilación instalando las dependencias necesarias.

2º. Compilación del código fuente:

El código es restaurado y compilado en modo Release, generando los archivos binarios y paquetes correspondientes en el directorio de salida (out).

3º. Construcción y publicación de la imagen Docker:

A partir de un Dockerfile específico, se construye la imagen de contenedor. Una vez generada, la imagen se etiqueta utilizando dos referencias: latest y un tag basado en el commit actual. Posteriormente, la imagen es subida al repositorio de Amazon Elastic Container Registry (ECR).

4º. Despliegue en Kubernetes (EKS):

Los manifiestos de despliegue de Kubernetes son actualizados con el nuevo tag de la imagen. Finalmente, se aplican los archivos YAML correspondientes al despliegue (deploy) y al escalado automático (hpa), realizando un rolling update controlado mediante el reinicio del deployment, asegurando así que las instancias del contenedor ejecuten la nueva versión.

Este flujo automatizado permite garantizar consistencia en la construcción de imágenes, reducir los riesgos operativos y facilitar la integración continua de cambios hacia los entornos de ejecución en EKS.

- **Validación de calidad y seguridad**

Es fundamental verificar automáticamente las imágenes por vulnerabilidades antes de enviarlas a los clústeres productivos. Para esto se incluyen pasos de validación con herramientas como el escaneo de imágenes ECR o SonarQube, y funciones Lambda dedicadas a borrar versiones obsoletas. Por ejemplo, AWS presenta un patrón de pipeline con EC2 Image Builder y EventBridge para regenerar imágenes cuando se detectan fallos o vulnerabilidades.

2.4.2. Almacenamiento de imágenes en Amazon ECR

Amazon Elastic Container Registry (ECR) es un servicio de registry completamente administrado por AWS que permite almacenar, gestionar y distribuir imágenes y artefactos OCI de manera segura, escalable y confiable .

- **Seguridad y acceso**

ECR integra controles de acceso mediante IAM, lo que permite definir políticas granuladas a nivel de repositorio para gestionar quién puede realizar operaciones como “push” y “pull” de imágenes . Las imágenes también se cifran en tránsito (HTTPS) y en reposo (S3), con opción de utilizar llaves gestionadas por AWS KMS .

- **Alta disponibilidad y escalabilidad**

ECR se apoya en Amazon S3 y su infraestructura de alta durabilidad, lo que garantiza que las imágenes estén siempre accesibles y resistentes a fallos . También soporta la replicación entre regiones y cuentas, ideal para entornos distribuidos y de recuperación ante desastres.

- **Integración con CI/CD**

ECR se alinea perfectamente con pipelines automáticos, permitiendo almacenar versiones generadas en flujos de integración y exportar imágenes hacia recursos como EKS . Este modelo facilita la coordinación entre desarrollo, construcción y despliegue en la nube, en línea con tus objetivos de automatización.

- **Escaneo de vulnerabilidades y limpieza automática**

Cada vez que se escribe una imagen, ECR puede activar un escaneo para detectar vulnerabilidades en tiempo real, empleando Amazon Inspector. Los hallazgos se

actualizan continuamente a medida que emergen nuevas vulnerabilidades. Las "lifecycle policies" permiten definir reglas para eliminar imágenes obsoletas, como conservar solo las últimas n versiones, lo que optimiza el uso de almacenamiento. Además, se pueden definir funciones Lambda complementarias que eliminan imágenes vulnerables o no utilizadas –un patrón pertinente para mantener el sistema limpio y seguro.

2.4.3. Validación de seguridad de imágenes

La validación de seguridad de imágenes implica analizar las mismas para identificar vulnerabilidades conocidas y prevenir que artefactos inseguros lleguen a entornos productivos en Kubernetes. Esta práctica es esencial para mantener la integridad y seguridad de las aplicaciones contenerizadas.

a) Escaneo de vulnerabilidades en ECR

Amazon ECR ofrece dos mecanismos de escaneo:

- **Basic scanning:** realiza un análisis por defecto del sistema operativo, utilizando la base de datos de CVEs. Disponible automáticamente al subir imágenes y con opción de escaneo manual (una vez cada 24 horas).
- **Enhanced scanning:** utiliza Amazon Inspector para identificar vulnerabilidades tanto del sistema operativo como de los paquetes de lenguaje, escaneando cada vez que hay un push o de forma continua. Los resultados se envían a EventBridge para su procesamiento automatizado

b) Proceso automatizado con Lambda y notificaciones

Tras finalizar el escaneo, se puede usar Amazon EventBridge para desencadenar una función Lambda que:

- Consulte los hallazgos de seguridad del contenedor.
- Filtre las vulnerabilidades por criticidad (MEDIUM, HIGH, CRITICAL).
- Genere registros estructurados en CloudWatch o envíe alertas vía email/SNS .

Esto permite actuar de manera inmediata en caso de detectar vulnerabilidades graves, eliminando la imagen o bloqueando su uso en el clúster.

c) Inspección continua y aseguramiento de integridad durante el pipeline

Según las mejores prácticas, cada push debería disparar un escaneo básico, mientras que el inspector continuo monitoriza cambios en las bases de datos de vulnerabilidades. Las imágenes con vulnerabilidades críticas deben bloquearse mediante Webhooks de admisión de Kubernetes o políticas en Security Hub

d) Uso de SBOM y attestations

Las herramientas modernas permiten generar SBOM (Software Bill of Materials), que describen componentes y dependencias de cada imagen. Estos SBOM, firmados criptográficamente, pueden usarse para asegurar un software supply chain robusto.

Automatización con Lambda para limpieza de imágenes antiguas

En entornos de contenedores, el repositorio de imágenes puede acumular versiones obsoletas rápidamente, ocasionando costos innecesarios y dificultando la gestión (Barclay, 2016). Para resolver esto, se implementa una función **AWS Lambda** que automatiza la eliminación de imágenes antiguas en ECR mediante un flujo programado:

a) Identificación de imágenes obsoletas

La función Lambda, activada por Amazon EventBridge o CloudWatch Events, ejecuta un script que consulta los repositorios ECR y detecta imágenes no utilizadas por ningún contenedor (por ejemplo, por ECS/EKS) y que cumplan criterios de antigüedad o cantidad para eliminar. Cada repositorio suele conservar solo las n versiones más recientes, descartando el resto.

b) Limpieza automática de imágenes antiguas en ECR mediante Lambda

Se implementará una función AWS Lambda destinada a automatizar la limpieza de imágenes antiguas almacenadas en Amazon Elastic Container Registry (ECR), con el fin de optimizar el repositorio y reducir los costos asociados al almacenamiento en AWS.

La función Lambda se integrará con el clúster Amazon EKS para obtener el listado de imágenes actualmente en uso por las aplicaciones desplegadas. A partir de esta

información, se realizará una comparación con el inventario completo de imágenes disponibles en ECR, aplicando los siguientes criterios de retención:

- Conservar la imagen actualmente utilizada por EKS.
- Conservar la imagen más reciente registrada en ECR (según su fecha de creación), incluso si no está en uso.
- Eliminar automáticamente todas las imágenes restantes, garantizando que el repositorio ECR permanezca optimizado, eficiente y libre de artefactos obsoletos.

2.4.4. Ventajas operativas

- **Costos optimizados:** se reduce el almacenamiento de ECR y los costos por escaneo de vulnerabilidades .
- **Automatización total:** no requiere intervención manual, mejorando eficiencia y reduciendo errores humanos.
- **Seguridad y trazabilidad:** las acciones quedan registradas en CloudTrail y pueden notificarse por email o SNS (Siddhant Khare, 2024) .

- **Alternativas y mejoras**

Políticas de ciclo de vida en ECR eliminan imágenes según antigüedad o número máximo, pero no consideran uso en contenedores .

El enfoque combinado (Lambda + ciclo de vida) permite reglas más sofisticadas, como conservar solo imágenes no referenciadas y mantener versiones recientes disponibles .

2.5. Monitorización, métricas y alertas

2.5.1. Uso de Metric Server en Kubernetes

El Metric Server es un componente nativo, ligero y esencial en Kubernetes, diseñado para recolectar métricas de uso de CPU y memoria de los nodos y los pods, exponiéndolas a través de la API de Kubernetes. No almacena datos históricos, pero es fundamental para alimentar mecanismos de autoscalado como Horizontal Pod Autoscaler (HPA) y Vertical Pod Autoscaler (VPA) .

Este gráfico se actualiza cada 15 segundos (por defecto), usando recursos mínimos (aproximadamente 1 mCPU y 2 MiB por nodo), lo que permite escalar rápidamente sin sobrecargar el clúster.

2.5.2. Instalación y despliegue en EKS

En Amazon EKS, Metric Server se instala como Add-On o mediante manifiesto YAML. Requiere acceso al endpoint de autenticación agregada (aggregation layer) y a las métricas del kubelet en el puerto 10250. Para nodos Fargate, se modifica el puerto (ej. 10251).

2.5.3. Casos de uso

Autoscalado automático (HPA/VPA), ajustando el número de réplicas o recursos asignados en función del uso real.

- Monitoreo en tiempo real *via* **kubecttl top pods** y **kubecttl top nodes**.
- *Facilita* toma de decisiones de planificación de capacidad, *ayudando a identificar cuellos de botella*.

Limitaciones y mejores prácticas

- No está pensado como herramienta de monitoreo histórico; para análisis a largo plazo se debe integrar con Prometheus y Grafana.
- En producción a gran escala es recomendable habilitar múltiples réplicas para alta disponibilidad.
- Deben configurarse solicitudes/límites en el Metric Server para evitar problemas de rendimiento.

2.5.4. CloudWatch Container Insights: observabilidad del cluster

Amazon CloudWatch Container Insights es un servicio totalmente administrado que simplifica la supervisión y diagnóstico de aplicaciones en contenedores en EKS, ECS o Kubernetes sobre EC2. Ofrece visibilidad completa en tiempo real de la salud del clúster, incluyendo métricas como CPU, memoria, red, almacenamiento, además del estado del plano de control (servidor API, etcd)

- **Despliegue y configuración**

Container Insights se habilita mediante el complemento `amazon-cloudwatch-observability` en EKS, lo que instala automáticamente el agente de CloudWatch y Fluent Bit—utilizando OpenTelemetry bajo el capó—facilitando el envío de métricas y logs. Tras la activación:

- Las métricas se recopilan desde nodos, pods y el plano de control.
- Los logs se canalizan a CloudWatch Logs usando Fluent Bit.

- **Métricas clave y capacidades**

CloudWatch Container Insights proporciona:

- Métricas específicas por contenedor, pod y nodo (CPU, memoria, red, disco).
- Indicadores del estado del plano de control, como uso de CPU del API server, latencia, errores y condiciones de `etcd`.
- Dashboards prediseñados en la consola de CloudWatch y paneles personalizables, útiles para diagnóstico y respuesta rápida.

Además, permite crear alarmas automáticas ante eventos como reinicios, cuellos de botella o errores recurrentes.

- **Beneficios operativos**

- Detección temprana de anomalías

Permite identificar fugas de memoria, sobreuso de recursos y fallos recurrentes con rapidez.

- Reducción del Mean Time to Resolution (MTTR)

Al ofrecer información estructurada por nivel (cluster → contenedor), facilita aislar y solucionar problemas en menor tiempo.

- Costo-efectividad y cobertura ampliada

El modelo de facturación basado en observaciones permite visibilidad mejorada sin los costos de almacenamiento tradicional.

2.5.5. Prometheus y Grafana para visualización de métricas

Prometheus y Grafana conforman una solución de monitoreo abierta, popularmente adoptada para supervisar el rendimiento de clústeres Kubernetes en Amazon EKS (Midde, 2024; NimbusStack, 2025) .

2.5.5.1.Prometheus: recopilación y almacenamiento de métricas

Prometheus es una base de datos de series temporales diseñada para monitoreo y alertas, que opera según un modelo de extracción (“scrape”), colectando métricas periódicamente de nodos, pods y endpoints expuestos, entre otros . En EKS, se puede habilitar mediante Add-Ons (managed scraper) o usarse la versión autogestionada instalada con Helm. Esta herramienta permite definir reglas de alertas, usar PromQL para consultas complejas, y establece la base para análisis en tiempo real de salud del clúster .

2.5.5.2.Grafana: interfaz de visualización

Grafana es una plataforma de visualización que trabaja como front-end para Prometheus. Permite crear tableros interactivos mediante paneles con métricas desde múltiples fuentes (Prometheus, CloudWatch, etc.) . Al integrarse con EKS, Grafana muestra dashboards como el “Kubernetes EKS Cluster” para explorar estado del clúster, uso de recursos y tendencias de performance, facilitando el análisis desde un nivel macro hasta micro .

2.5.5.3.Beneficios operacionales

- **Visibilidad en tiempo real** sobre rendimiento y capacidad del clúster (CPU, memoria, red).
- **Alertas proactivas**, con integración vía Alertmanager y SNS, permiten notificar ante condiciones críticas .
- **Análisis histórico** de métricas para identificar tendencias y planificar capacidad .
- **Interoperabilidad** con otros sistemas de registro y monitoreo, lo que facilita una visión unificada .

2.5.6. Lambda para envío de alertas proactivas

En el marco de la monitorización del clúster EKS, las alertas generadas por CloudWatch o Prometheus deben ser notificadas rápidamente para reducir el tiempo de respuesta ante fallos. Para ello, se emplea una función **AWS Lambda** que se activa mediante notificaciones de **Amazon SNS** o **CloudWatch Events**, adaptada a los requisitos del entorno contenerizado.

a) Flujo de activación de alertas

- Una **alarma de CloudWatch** detecta un umbral crítico (CPU alta, pods pendientes, errores repetidos...) y entra al estado **ALARM**.
- El servicio envía un mensaje a un **Topic SNS** configurado como destino automático.
- Un trigger de **Lambda**, suscrito al tópic, recibe el evento JSON.
- El código de Lambda formatea el mensaje —para incluir detalles útiles— y lo remite a destinatarios definidos: correo electrónico, Slack, Teams o sistemas externos.

b) Beneficios de este enfoque

- Flexibilidad de formato: Lambda puede enriquecer el mensaje, incluir contexto, enlaces e instrucciones de respuesta.
- Adaptabilidad de destino: Se pueden enviar notificaciones a múltiples canales —email, SMS, Slack, etc.— sin modificar la alarma original.
- Escalabilidad y fiabilidad: Al trabajar con almacenamiento asincrónico y reintentos, Lambda garantiza entrega incluso ante brechas temporales o errores —CloudWatch reintenta automáticamente la invocación de la función.

c) Configuración técnica

- El Lambda necesita permisos de ejecución (**AWSLambdaBasicExecutionRole**), lectura de logs y acceso a SNS.
- Se habilita una **trigger SNS** en la consola Lambda o mediante IaC (Terraform, CloudFormation).

- El formato del mensaje se define dentro del código (Node.js, Python) y puede incluir plantillas JSON, HTML o Markdown para canales enriquecidos .
- Se recomienda activar **Lambda Insights** para monitorear la performance y uso del ejecutable .

2.5.7. OpenLens como herramienta de observación local para desarrollo

OpenLens es una distribución de código abierto del popular IDE de Kubernetes conocido como Lens. Su objetivo es proporcionar una interfaz gráfica potente y directa para interactuar con clústeres, facilitando la observación, gestión y solución de problemas en entornos locales o remotos.

a) Características principales

- **Interfaz gráfica intuitiva:** Ofrece un dashboard visual que presenta en tiempo real el estado del clúster (pods, servicios, deployments, logs), evitando el uso exclusivo de líneas de comando.
- **Agilidad en debugging:** Permite explorar eventos y logs, filtrar errores o acceder a terminales de pods de forma inmediata.
- **Múltiples clústeres y contextos:** Gestiona varias conexiones de clústeres y kubeconfig desde un solo cliente, reduciendo la complejidad y el cambio de contextos.
- **Extensible y ligero:** Al ser open source, permite agregar plugins para funcionalidades adicionales, como interacción con pods, visualización mejorada o escaneo (p. ej. con Trivy Operator) .

b) Beneficios para el desarrollo

- **Incremento de productividad:** Siendo un IDE optimizado para Kubernetes, reduce el tiempo en tareas cotidianas como navegación, registro y gestión de recursos.
- **Menor riesgo de errores operativos:** Al ofrecer una visión clara, minimiza errores comunes de comando y facilita la identificación de problemas.
- **Mayor adopción por equipos:** La facilidad visual y multiplataforma (Windows, Linux, Mac) acelera la integración de OpenLens en equipos DevOps.

c) Comparativa con herramientas basadas en la consola

Aunque herramientas como `kubectl` y `k9s` siguen siendo fundamentales, OpenLens aporta ventajas:

- Permite inspección visual de configuraciones, métricas y logs sin instalar nada en el clúster.
- Facilita la administración escalable de múltiples entornos y la colaboración de equipo.

2.6. Gestión del tráfico y seguridad del clúster

2.6.1. Ingress-NGINX como controlador de acceso

El controlador Ingress-NGINX es una solución popular en clusters Kubernetes (incluido EKS) para gestionar el tráfico HTTP/HTTPS entrante, convirtiéndose en una capa de proxy inverso que traduce recursos Ingress declarativos en reglas de NGINX .

2.6.2. Instalación y despliegue en AWS EKS

Se implementa típicamente mediante Helm o manifests, desplegando un pod de `ingress-nginx-controller` en un namespace independiente (ej. `ingress-nginx`).

Para exponerlo, se configura un Service tipo `LoadBalancer`. En AWS, se recomienda usar un Network Load Balancer (NLB) frente a un Application Load Balancer (ALB) para reducir la complejidad.

2.7. Configuración avanzada

2.7.1. Configuración privada del clúster y reglas de seguridad

La configuración privada en un clúster EKS consiste en restringir el plano de control y los nodos para que solo sean accesibles desde dentro de la VPC, eliminando accesos públicos e incrementando la seguridad (AWS, s.f.). Amazon define detalladamente este modelo en la guía en español de EKS, enfatizando lo siguiente:

2.7.1.1.Endpoints privados y eliminación de accesos públicos

El plano de control de EKS puede configurarse para que acepte únicamente conexiones a través de endpoints privados dentro de la VPC. De esta manera, se desvincula del tráfico de internet y solo los bastion hosts o subredes específicas pueden acceder al API server (AWS, s. f.) .

2.7.1.2.Requisitos de endpoints de VPC para servicios EKS

Para operar sin internet pública es esencial habilitar VPC endpoints (AWS PrivateLink) para servicios como ECR, STS, S3, CloudWatch, EC2, entre otros. Esto garantiza que los nodos y pods puedan comunicarse internamente con estos servicios, sin necesidad de NAT o salidas inseguras (AWS, s. f.).

2.7.1.3.Reglas de grupos de seguridad (Security Groups)

Se deben definir SG específicos con reglas de entrada y salida restringidas:

- Solo los bastion hosts o rangos IP autorizados pueden conectarse al endpoint Kubernetes.
- El tráfico entre nodos, plano de control y servicios internos deben estar claramente delimitados.
- Se evita el acceso público mediante la eliminación de reglas 0.0.0.0/0 orientadas al plano de control .

2.7.1.4.Autenticación y autorización

Se recomienda deshabilitar el acceso público: solamente los usuarios o mecanismos que asuman roles IAM (como bastion host o identidades de pods) pueden interactuar con el plano de control. El uso de IRSA (IAM Roles for Service Accounts) fortalece la seguridad al vincular servicios directamente con credenciales y permisos personalizados.

2.7.1.5.Ventajas operativas

- Reducción notable de superficie de ataque, ya que no hay endpoints expuestos.
- Aumento de observabilidad y control auditado, al canalizar el acceso por dentro de la VPC.
- Conformidad con normativas que exigen aislamiento de redes y segregación de entornos de producción.

2.7.2. Administración de roles IAM y uso de claves KMS

La gestión de accesos en EKS se fundamenta en la correcta configuración de roles de IAM y el uso adecuado de claves AWS KMS para proteger datos sensibles, como los secretos de Kubernetes.

2.7.2.1.Roles de IAM en Amazon EKS

- **Rol del clúster y roles vinculados a servicios**

Cada clúster EKS necesita un rol de servicio, que AWS gestiona automáticamente (AWS Service Role For Amazon EKS), con permisos para controlar recursos de red, EC2 y autoscaling.

Además, se genera un rol de clúster (Amazon EKS Cluster Policy), responsable de crear nodos y equilibradores de carga .

- **Roles vinculados a cuentas de servicio (IRSA)**

Se recomienda emplear IRSA para asignar permisos mínimos a pods específicos. Esta práctica limita el acceso a datos y evita que un pod tenga credenciales que excedan su rol.

2.7.2.2.Protección de secretos del clúster con KMS

- **Cifrado de secretos en reposo**

EKS permite cifrar los secretos de Kubernetes usando claves KMS gestionadas por el cliente (CMK), incrementando la protección de datos en repositorios sensibles.

La configuración puede activarse mediante `eksctl` o la consola, pero una vez habilitada no puede revertirse .

- **Buenas prácticas en políticas de KMS**

Las políticas clave deben definirse en IAM para restringir el acceso solo a quienes correspondan, proyectando condiciones (como la cuenta o el rol) .

Se utiliza `aws:PrincipalArn` para garantizar que solo entidades permitidas puedan descifrar secretos, reforzando el principio de mínimos privilegios .

- **Data-in-transit y cifrado de volúmenes**

AWS recomienda utilizar KMS para cifrar volúmenes de EKS, incluyendo `etcd` y almacenamiento persistente, con el objetivo de asegurar datos en reposo.

Todos los metadatos y tráfico sensible se enrutan a través de canales cifrados TLS gestionados por KMS, evitando exposiciones.

2.8. Definiciones conceptuales

1) AWS (Amazon Web Services):

Plataforma de servicios en la nube ofrecida por Amazon que proporciona infraestructura, almacenamiento, bases de datos, servicios de inteligencia artificial, seguridad, administración de clústeres de contenedores y otros recursos escalables bajo demanda, permitiendo a empresas y desarrolladores operar aplicaciones globales de manera flexible y segura. (Amazon, 2018)

2) API:

Interfaz de programación de aplicaciones que permite la comunicación e integración entre diferentes sistemas de software, facilitando el intercambio de datos y la interoperabilidad entre servicios. (Hernández y Flórez, 2014)

3) Autoescalable:

Capacidad de una infraestructura (como un clúster de Kubernetes) para ajustar sus recursos automáticamente según la demanda de tráfico, optimizando desempeño, costos y crecimiento organizacional. (Amazon, 2018)

4) AWS (Amazon Web Services):

Plataforma de servicios en la nube ofrecida por Amazon que proporciona infraestructura, almacenamiento, bases de datos, servicios de inteligencia artificial, seguridad, administración de clústeres de contenedores y otros recursos escalables bajo demanda, permitiendo a empresas y desarrolladores operar aplicaciones globales de manera flexible y segura. (Amazon, 2018)

5) Cloud:

Modelo de almacenamiento y procesamiento de datos en plataformas remotas accesibles vía internet, que permite utilizar servicios de computación bajo demanda, pagando solo por lo consumido. (Rengifo, 2013; Hernández y Flórez, 2014)

6) Cluster:

Es un conjunto de máquinas (nodos) que ejecutan aplicaciones en contenedores, gestionadas de forma conjunta, compartiendo servicios y comportándose como un único servidor dentro de un entorno distribuido. (Redhat, 2017)

7) Clúster Autoscaler:

Componente de Kubernetes que ajusta automáticamente el tamaño del clúster según las necesidades de los pods, asegurando que existan suficientes recursos disponibles para las cargas de trabajo, pero sin mantener nodos innecesarios activos. (Amazon, 2018)

8) Contenedor:

Unidad ligera de virtualización que encapsula aplicaciones y sus dependencias, facilitando su ejecución aislada en un mismo sistema operativo, optimizando recursos y gestión de infraestructura. (Amazon, 2018)

9) Control-plane:

Conjunto de componentes de Kubernetes responsables de tomar decisiones globales sobre el clúster, como la planificación, el monitoreo de estado y la gestión de eventos para mantener el estado deseado del sistema. (Nogués, 2018)

10) Docker:

Plataforma ampliamente utilizada para el empaquetado, distribución y ejecución de aplicaciones dentro de contenedores, permitiendo el desarrollo, prueba y despliegue

ágil de microservicios. Aunque su motor de contenedores (Docker Engine) mantiene componentes de código abierto, su uso empresarial en productos como Docker Desktop está sujeto a licenciamiento comercial. (Ruelas, 2017).

11) EKS (Elastic Kubernetes Service):

Servicio administrado de Kubernetes en AWS que permite implementar, gestionar y escalar aplicaciones contenerizadas de forma automatizada, eliminando la necesidad de instalar y operar el plano de control de Kubernetes directamente, integrándose con otros servicios de AWS para seguridad, redes y almacenamiento. (Amazon, 2018)

12) Horizontal-Pod-Autoescaler (HPA):

Permite escalar automáticamente la cantidad de pods en función de métricas como el uso de CPU u otros parámetros personalizados, asegurando la disponibilidad de recursos de forma dinámica según la demanda. (Amazon, 2018)

13) IaC (Infraestructura como Código):

Proceso de gestión y aprovisionamiento automatizado de infraestructura mediante scripts de código, permitiendo una administración reproducible, eficiente y controlada de entornos de TI. (Ruelas, 2017)

14) Kubernetes:

Sistema de orquestación de contenedores que automatiza el despliegue, la gestión, el escalado y el mantenimiento de aplicaciones distribuidas en contenedores. (Nogués, 2018)

15) Transit Gateway:

Servicio de Amazon Web Services (AWS) que permite conectar múltiples VPCs, redes on-premise y otras conexiones de red a través de un único punto centralizado de enrutamiento, simplificando la gestión y escalabilidad de arquitecturas de red complejas (Amazon Web Services, 2024).

CAPÍTULO III

MARCO METODOLÓGICO

3.1. Enfoque de la investigación

Esta investigación se enmarca en un enfoque aplicado con alcance descriptivo-explicativo, dado que se orienta a analizar un problema tecnológico concreto y proponer una solución basada en un modelo de automatización mediante infraestructura como código en Amazon EKS. No se realizó una experimentación estadística repetida, sino una validación conceptual y técnica del diseño, contrastando el estado inicial (procesos manuales) con el escenario planteado (automatización). Este enfoque permite comprobar las hipótesis en el plano teórico y tecnológico, fundamentándose en la literatura especializada y en la lógica del modelo propuesto (Hernández Sampieri, Fernández & Baptista, 2014).

Asimismo, la naturaleza del estudio es aplicativa, ya que busca generar una solución tecnológica que puede implementarse en entornos reales de nube. Según Creswell (2014), este tipo de investigaciones son adecuadas cuando el objetivo es diseñar o proyectar mejoras en un proceso existente, lo cual se alinea con el presente trabajo.

3.2. Diseño de la investigación

El diseño de la presente investigación es pre-experimental, ya que se analiza un único grupo en dos momentos descrito por Hernández Sampieri, Fernández y Baptista (2014): el estado inicial del sistema (procesos manuales de despliegue y recuperación) y el estado propuesto mediante la automatización de aplicaciones contenerizadas con infraestructura como código en Amazon EKS. No se trabajó con grupos de control ni asignación aleatoria, sino con un entorno tecnológico definido sobre el cual se desarrolla y valida conceptualmente la intervención.

La elección de este diseño se justifica porque la investigación plantea una relación causal entre la variable independiente (automatización con IaC en EKS) y la variable dependiente (tiempo de recuperación y continuidad operativa). Esta relación solo puede formularse metodológicamente dentro de un diseño pre-experimental, incluso cuando la intervención no se ejecuta en un entorno productivo sino en un esquema técnico y conceptual propio de proyectos tecnológicos.

3.3. Nivel de la investigación

El nivel de la investigación es descriptivo-explicativo, dado que no se limita a exponer las características del problema, sino que también explica, a partir de la fundamentación teórica y del diseño elaborado, cómo la automatización con herramientas de infraestructura como código en Amazon EKS contribuye a mejorar los tiempos de recuperación y la continuidad operativa del sistema.

A su vez, el estudio es de carácter tecnológico-aplicativo, en tanto que desarrolla una propuesta práctica que puede ser implementada y replicada en organizaciones que operan sobre plataformas en la nube. Según Balestrini (2006), este tipo de investigación se orienta a generar innovaciones técnicas que resuelvan problemas concretos del entorno.

3.4. Población

En investigaciones de carácter tecnológico-aplicativo, la población no está constituida por personas, sino por el conjunto específico de elementos que representan el objeto directo de estudio. En el caso de esta investigación, la población está conformada exclusivamente por las aplicaciones contenerizadas desplegadas en Amazon Web Services (AWS), las cuales presentaban procesos manuales de despliegue, recuperación y operación antes de la intervención del modelo de automatización.

De acuerdo con Hernández Sampieri et al. (2014), la población corresponde al conjunto de elementos que comparten características comunes relacionadas con el fenómeno estudiado. En este caso, la característica común es que todas las aplicaciones:

- Estaban contenerizadas,
- Operaban en AWS,
- Carecían de estandarización mediante IaC,
- No utilizaban pipelines CI/CD,
- Dependían de intervención manual para su despliegue y recuperación.

Por lo tanto, la población está constituida únicamente por las aplicaciones contenerizadas que cumplían dichas condiciones, las cuales fueron posteriormente sometidas al análisis pretest–postest.

3.5. Muestra

La muestra fue intencional y no probabilística, centrada en un conjunto de entornos contenerizados representativos del problema identificado, conformados por aplicaciones y clusters EKS reales que operaban inicialmente con despliegues manuales, configuraciones no estandarizadas, ausencia de IaC y sin mecanismos formales de recuperación.

Esta muestra especifica el aporte diferencial del estudio, debido a que no se trató de validar tecnologías ya conocidas, sino de evidenciar el cambio entre un estado inicial deficiente y un estado automatizado, aplicando un diseño pre–experimental. Los elementos seleccionados cumplen los siguientes criterios técnicos:

- Presentar estado inicial manual, sin IaC, sin CI/CD y sin políticas de recuperación documentadas.
- Ser compatibles con Kubernetes en Amazon EKS.
- Permitir recrear escenarios de automatización comparables entre pretest y postest.
- Representar casos reales donde las arquitecturas de AWS no habían sido implementadas por falta de estandarización tecnológica.

En total, la muestra estuvo conformada por 71 aplicaciones contenerizadas, cada una desplegada mediante contenedores y ejecutada dentro del clúster EKS. Estas aplicaciones constituyeron el insumo técnico sobre el cual se aplicó el modelo de

automatización y fueron posteriormente caracterizadas con detalle en el Capítulo 4 de Resultados.

Este tipo de muestreo es adecuado en investigaciones tecnológicas donde se requiere trabajar con elementos que permiten comparar el estado previo con el escenario automatizado (Tamayo y Tamayo, 2006).

3.6. Técnicas e instrumentos de recolección de datos

Dado que la investigación se orienta a la validación de un diseño tecnológico aplicado, las técnicas de recolección de datos se centraron en la observación técnica de los procesos involucrados en la automatización y en el análisis documental de fuentes especializadas. Estas técnicas permitieron sustentar la construcción del modelo de recuperación y continuidad operativa mediante infraestructura como código (IaC) en Amazon EKS. Las técnicas utilizadas fueron:

- Análisis documental: consistió en la revisión de manuales oficiales de Amazon Web Services, guías técnicas de Kubernetes y literatura especializada sobre automatización, IaC y orquestación de contenedores (Amazon Web Services, 2024; The Linux Foundation, 2022). Estas fuentes permitieron fundamentar conceptualmente la reducción de tiempos de recuperación y la estandarización operativa en entornos contenerizados.
- Observación técnica estructurada: se efectuó un registro detallado de los componentes del diseño, contrastando el funcionamiento inicial basado en procesos manuales con el funcionamiento esperado bajo el modelo automatizado. Esta observación permitió identificar los elementos clave para garantizar continuidad operativa.

Como instrumentos se consideraron:

- **Registros de configuración y archivos de logs** generados durante la construcción del diseño, los cuales permitieron documentar evidencias técnicas relacionadas con los mecanismos de recuperación.
- **Herramientas de monitoreo de AWS**, tales como CloudWatch y el EKS Dashboard, utilizadas como soporte para describir cómo los servicios gestionan métricas de rendimiento, disponibilidad y estado del clúster (Amazon Web Services, 2024).

- **Matriz de validación del diseño**, elaborada por el investigador, donde se consignaron las variables críticas: tiempos de recuperación estimados, disponibilidad proyectada y nivel de intervención manual. Este instrumento permitió comparar sistemáticamente el escenario previo con la propuesta automatizada.

El uso de estas técnicas e instrumentos no tuvo como finalidad generar datos estadísticos cuantificables, sino obtener evidencia técnica y conceptual que respalde la viabilidad, pertinencia y mejora operativa del modelo planteado.

3.7. Análisis y procesamiento de datos

El análisis y procesamiento de datos se realizó bajo un enfoque descriptivo–comparativo, acorde con el diseño preexperimental empleado en la investigación. El propósito fue contrastar el escenario manual inicial con el modelo de automatización implementado mediante infraestructura como código en Amazon EKS.

En una primera etapa, se organizó la información obtenida a través de la observación técnica estructurada, los registros de configuración generados durante el diseño y la revisión documental especializada. Con estos insumos se sistematizó el escenario manual inicial, describiendo sus características operativas, las aplicaciones contenerizadas involucradas, los procesos manuales ejecutados, la ausencia de estandarización y los tiempos estimados de recuperación. Esta sistematización permitió establecer la línea base para el análisis comparativo.

Posteriormente, se procesaron los indicadores definidos en la matriz de operacionalización, elaborando tablas y gráficos que muestran los valores del escenario previo y los valores obtenidos tras la aplicación del modelo automatizado. Para el análisis de los resultados se empleó estadística descriptiva basada en la comparación de medias aritméticas y porcentajes de mejora de los indicadores evaluados, permitiendo contrastar el comportamiento del escenario manual (pretest) frente al escenario automatizado (postest). Entre los indicadores analizados se incluyeron el tiempo de despliegue, el tiempo medio de recuperación (MTTR), la disponibilidad del servicio y el porcentaje de éxito del pipeline CI/CD. Esta presentación tabular permitió visualizar de forma clara las mejoras derivadas del diseño propuesto y sustentar la contrastación de las hipótesis de investigación.

Asimismo, se procesó la matriz de validación del diseño, empleada para evaluar dimensiones como escalabilidad, resiliencia, continuidad operativa, eficiencia del despliegue y coherencia técnica. Sus resultados fueron presentados de manera estructurada, complementando el análisis comparativo entre ambos escenarios.

Finalmente, se integraron los datos en un análisis interpretativo sustentado en la evidencia técnica recopilada y en la fundamentación teórica del estudio. Siguiendo a Hernández Sampieri et al. (2014), se orientó la interpretación hacia la contrastación entre el estado inicial y el modelo automatizado, permitiendo validar las hipótesis relacionadas con la mejora en la continuidad operativa, el tiempo de recuperación y la reducción de intervención manual.

3.8. Aspectos éticos

La presente investigación se desarrollará bajo el cumplimiento de principios éticos fundamentales aplicables a estudios de carácter científico y tecnológico. En primer lugar, se asegura el respeto a la integridad académica, evitando el plagio mediante la adecuada citación y referencia de todas las fuentes bibliográficas utilizadas, conforme a las normas de estilo APA (7.^a edición). Asimismo, se garantiza la transparencia en la recolección y análisis de datos, asegurando que los resultados presentados respondan fielmente a los procesos experimentales realizados en el entorno de prueba, sin manipulación ni alteración que pueda inducir a conclusiones erróneas.

Se respetó el principio de confidencialidad respecto a la información sensible vinculada con la infraestructura tecnológica utilizada, protegiendo la identidad de las cuentas, configuraciones específicas o credenciales que pudieran comprometer la seguridad de los sistemas. Del mismo modo, se promueve el principio de responsabilidad social, procurando que las recomendaciones y propuestas de automatización contribuyan a mejorar la eficiencia de las organizaciones, fomentando la innovación tecnológica y la continuidad de los servicios digitales sin generar impactos negativos en la operación de terceros.

Finalmente, la investigación se rigió por el principio de veracidad, presentando de manera objetiva tanto los hallazgos favorables como las posibles limitaciones encontradas durante el proceso.

CAPÍTULO IV

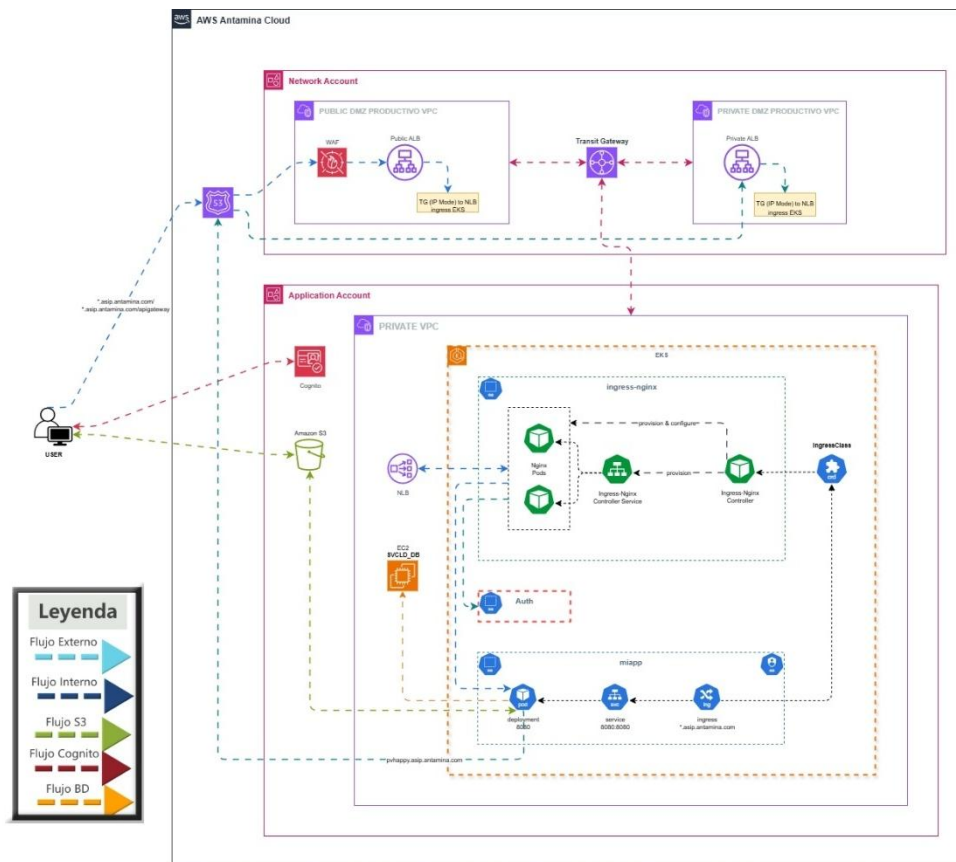
RESULTADOS

4.1. Descripción de la Arquitectura que soporta la automatización (Flujo App)

Como parte del desarrollo de la Aplicación, resulta necesario detallar la infraestructura tecnológica que sustenta la automatización planteada. La arquitectura definida permite comprender cómo se integran los servicios de red, seguridad, orquestación de contenedores y gestión de datos en la nube de AWS, sirviendo de base para garantizar la reducción del tiempo de recuperación y la continuidad operativa.

La Figura 1 presenta la arquitectura en AWS que soporta la automatización del despliegue y gestión de aplicaciones contenerizadas. En ella se identifican los componentes principales: la Network Account, encargada de la seguridad y el balanceo de tráfico; la Application Account, donde se ubican los servicios de autenticación y almacenamiento; y el clúster de Amazon EKS, que orquesta los pods y expone las aplicaciones mediante el Nginx Ingress Controller. Esta arquitectura integra servicios de red, seguridad, almacenamiento y autenticación para garantizar disponibilidad, continuidad operativa y escalabilidad.

Figura 1.
Arquitectura diseñada para la automatización



La figura 1, muestra la arquitectura en AWS que soporta la automatización del despliegue y gestión de aplicaciones contenerizadas. Se integran servicios de red y seguridad (WAF, NLB, Transit Gateway), cómputo y orquestación en EKS con Nginx Ingress Controller, así como almacenamiento (S3), autenticación (Cognito) y base de datos (RDS), asegurando alta disponibilidad y continuidad operativa.

En el **Anexo 2**, se muestra la Descripción detallada del Flujo Arquitectónico en AWS Antamina Cloud.

A fin de complementar la representación gráfica de la arquitectura diseñada para la automatización, se presenta a continuación una síntesis de los principales componentes que la conforman. Esta versión resumida permite identificar, de manera clara y organizada, las funciones de la Network Account, la Application Account, el clúster de EKS y los elementos adicionales que intervienen en el flujo de operación de la solución.

Tabla 3.

Componentes del flujo arquitectónico en AWS Antamina Cloud

Flujo	Descripción del componente
	Network Account: Contiene la Public DMZ con el WAF que filtra amenazas y el ALB público que distribuye el tráfico HTTP/HTTPS desde Internet, además de la Private DMZ con un ALB privado que gestiona tráfico interno. La comunicación entre cuentas se centraliza mediante el Transit Gateway.
	Application Account: En la VPC privada se despliegan servicios esenciales como <i>Cognito</i> para la autenticación de usuarios, <i>Amazon S3</i> para almacenamiento de objetos, un <i>NLB</i> para enrutar tráfico hacia EC2 o EKS, y una instancia <i>EC2</i> destinada a la base de datos.
	EKS (Elastic Kubernetes Service): Es el núcleo de la automatización, donde <i>Ingress Nginx</i> gestiona el enrutamiento del tráfico hacia los pods. Las aplicaciones se despliegan en <i>namespaces</i> , organizadas mediante <i>ServiceAccounts</i> , <i>Deployments</i> y <i>Services</i> , que garantizan escalabilidad, aislamiento y disponibilidad.
	Otros componentes: Se incluyen un microservicio de autenticación que complementa a <i>Cognito</i> , la resolución de dominios DNS y la interacción del <i>usuario externo</i> , que ingresa por el WAF/ALB y es redirigido hacia los servicios internos.
	Flujo principal: El usuario accede por un dominio público, el tráfico es filtrado por el WAF y balanceado por el ALB, se enruta mediante el Transit Gateway hacia la aplicación en EKS, donde el Ingress distribuye las solicitudes hacia los pods correspondientes, validando el acceso con <i>Cognito/Auth</i> y entregando contenido estático desde <i>S3</i> .

En conjunto, esta descripción sintetiza los elementos esenciales de la arquitectura y su funcionamiento general dentro de la nube de AWS. Sin embargo, para un análisis más profundo de cada componente y sus interacciones, se incluye en los anexos una

explicación detallada que amplía la información presentada y permite comprender con mayor precisión la lógica interna del flujo.

La propuesta de automatización del despliegue y gestión de aplicaciones contenerizadas en la nube de AWS se fundamenta en una arquitectura distribuida en dos cuentas principales: Network Account y Application Account, conectadas mediante un Transit Gateway que asegura la comunicación entre VPC públicas y privadas. A continuación, se describen sus componentes esenciales, organizados en cuatro bloques principales.

4.1.1. Red y Seguridad (Network Account)

La Network Account concentra los recursos de conectividad y protección. En ella se implementan las subredes públicas y privadas (DMZ), un Web Application Firewall (WAF) que mitiga amenazas externas, y un Application Load Balancer (ALB) que distribuye el tráfico HTTP/HTTPS. El enrutamiento entre cuentas se realiza a través del Transit Gateway, lo que garantiza escalabilidad, aislamiento y control centralizado del tráfico.

4.1.2. Orquestación de Contenedores (Application Account – EKS)

En la Application Account se encuentra la VPC privada que aloja el clúster de Amazon Elastic Kubernetes Service (EKS). Este clúster administra aplicaciones contenerizadas mediante pods y servicios, apoyado en el Nginx Ingress Controller, que gestiona el enrutamiento dinámico del tráfico. Gracias a esta configuración, se logra automatizar el aprovisionamiento, balanceo y escalabilidad de las aplicaciones, en concordancia con los principios de Infraestructura como Código (IaC).

4.1.3. Almacenamiento y Base de Datos

La arquitectura incorpora Amazon S3 como repositorio de objetos para la gestión de archivos y contenido estático requerido por la aplicación. Asimismo, se utiliza Amazon RDS como base de datos relacional administrada, lo que asegura persistencia,

consistencia y respaldo de la información con servicios gestionados de alta disponibilidad.

4.1.4. Autenticación y Gestión de Usuarios

El acceso seguro de los usuarios se garantiza mediante Amazon Cognito, encargado de la autenticación y autorización a través de login, federación e integración con tokens de acceso. Este componente resulta crítico en la interacción entre el usuario final y los servicios internos desplegados en el clúster, ya que centraliza la gestión de identidades y fortalece la seguridad del sistema.

La Figura 2 muestra la arquitectura en AWS que soporta la automatización del despliegue y gestión de aplicaciones contenerizadas. En ella se identifican los componentes principales numerados: (1) Red y Seguridad en la Network Account, (2) Orquestación de contenedores en la Application Account, (3) Servicios de almacenamiento y base de datos como S3 y RDS, y (4) Autenticación de usuarios mediante Cognito. Esta representación facilita visualizar de manera integral cómo se interconectan los servicios de red, seguridad, cómputo y datos para asegurar alta disponibilidad y continuidad operativa.

Diagrama de la Arquitectura en AWS que soporta la automatización (componentes identificados)



Los componentes numerados son los siguientes:

- 1) Red y Seguridad: Network Account (WAF, NLB, Transit Gateway, subredes públicas y privadas).000
- 2) Orquestación de Contenedores: Clúster EKS en la Application Account, con pods y Nginx Ingress Controller.
- 3) Almacenamiento y Base de Datos: Amazon S3 y Amazon RDS.
- 4) Autenticación y Gestión de Usuarios: Amazon Cognito.

En síntesis, la arquitectura presentada integra de manera coherente los servicios de red y seguridad, la orquestación de contenedores, el almacenamiento y la autenticación de usuarios, conformando la base tecnológica que sustenta la propuesta de automatización. Esta estructura no solo garantiza la disponibilidad y la continuidad

operativa, sino que también permite optimizar la gestión de los recursos mediante prácticas de *Infraestructura como Código (IaC)*. A partir de esta base arquitectónica, en el siguiente apartado se detalla el proceso de aprovisionamiento de la infraestructura, describiendo las etapas que permiten materializar la automatización en el entorno de AWS.

4.2. Arquitectura Interna del clúster EKS (Default Cluster)

Con la definición declarativa del archivo YAML se logra estandarizar la configuración del clúster, permitiendo que los recursos de red, los grupos de nodos y las políticas de seguridad se apliquen de manera automatizada y consistente en los distintos entornos. Este proceso concluye con un clúster EKS completamente aprovisionado y listo para su operación.

Para comprender con mayor precisión la arquitectura interna, se describe a continuación la interacción entre el plano de control, gestionado por AWS, y el plano de datos, soportado en nodos EC2 autogestionados, junto con los complementos, la seguridad y la red que aseguran la continuidad del servicio. La Figura 3 representa gráficamente dicha arquitectura.

4.2.1. Acceso y administración

- EC2 (Bastion): instancia en la VPC privada que sirve como punto de salto para administrar el clúster mediante kubectl.
- IAM Role for Cluster: asociado al plano de control de EKS para su operación segura.
- IAM Role for Nodes: permite a los nodos EC2 workers interactuar con servicios de AWS (CloudWatch, S3, etc.).
- IAM Roles for Service Accounts (IRSA): otorgan permisos específicos a pods dentro de Kubernetes siguiendo el principio de privilegios mínimos.

4.2.2. Plano de control (Control Plane – AWS Managed, privado)

- **API Server (privado):** punto de entrada principal para las llamadas de Kubernetes (kubectl, controladores).
- **etcd:** base de datos distribuida que almacena el estado del clúster (recursos, configuración, secretos).
- **Controller Manager:** gestiona los controladores internos (replicas, endpoints, jobs, etc.) asegurando el estado deseado.
- **Scheduler:** asigna pods a nodos en función de disponibilidad de recursos, afinidad y tolerancias.

4.2.3. Plano de datos (Data Plane – Self-managed Node Group)

- Self-managed Node Group (EC2 Workers): conjunto de instancias EC2 privadas que ejecutan pods y conforman el data plane.
 - Node Front: ejecuta pods relacionados al frontend.
 - Node ApiGateway: maneja pods dedicados al API Gateway.
 - Node ApiInterno: soporta servicios internos.
- NLB (NGINX): balanceador de carga interno que expone servicios de NGINX o un Ingress Controller en la red privada.
- Private Subnet: subred privada que aloja nodos y recursos del clúster.

4.2.4. Core Add-ons

- VPC CNI Plugin: asigna direcciones IP de la VPC de AWS a cada pod.
- kube-proxy: implementa reglas de red y balanceo para exponer servicios.
- CoreDNS: servicio de resolución de nombres DNS para pods y servicios internos.

4.2.5. Networking y flujo operativo

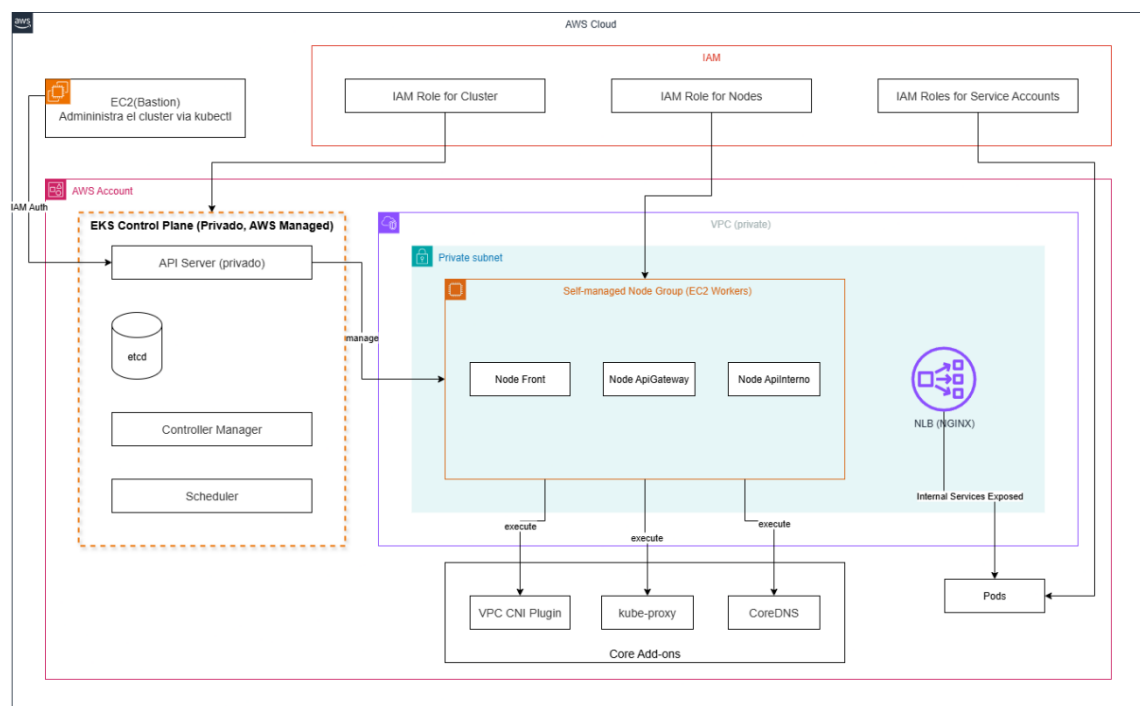
- Private Subnet: asegura que todos los nodos residan en una red privada.
- Pods: contenedores que ejecutan las cargas de trabajo (microservicios y aplicaciones internas).

- Flujo operativo:

- 1) El administrador accede al clúster vía Bastion Host con `kubectl`.
- 2) El API Server procesa las peticiones y gestiona el estado mediante etcd, Controller Manager y Scheduler.
- 3) Los nodos EC2 ejecutan pods y exponen servicios mediante el NLB interno (NGINX).
- 4) Los Core Add-ons mantienen la red, el enrutamiento y el DNS del clúster.
- 5) Los IAM Roles administran permisos a nivel de clúster, nodos y pods vía IRSA.

La Figura 3 muestra la arquitectura interna del clúster de Amazon Elastic Kubernetes Service (EKS) posterior a su aprovisionamiento mediante infraestructura como código. El diagrama permite visualizar la relación entre el plano de control, administrado por AWS, y el plano de datos conformado por nodos EC2 autogestionados, además de los mecanismos de seguridad, acceso y los complementos esenciales que garantizan la operación estable y segura del clúster..

Figura 3.
Arquitectura interna del clúster EKS en AWS



En la Figura 3 se observa la arquitectura interna del clúster EKS, donde se evidencia la interacción entre el plano de control (Control Plane), gestionado por AWS, y el plano de datos (Data Plane), implementado en nodos EC2 autogestionados. El plano de control está compuesto por el API Server como punto de entrada, la base de datos distribuida etcd, el Controller Manager y el Scheduler, responsables de mantener el estado deseado del clúster y asignar las cargas de trabajo a los nodos disponibles. En la subred privada se despliega un Node Group autogestionado, conformado por instancias EC2 con funciones diferenciadas —Node Front, Node ApiGateway y Node ApiInterno— que exponen servicios internos a través de un balanceador de carga (NLB) con NGINX. De forma complementaria, se integran los core add-ons, entre ellos el VPC CNI Plugin para la conectividad de red, kube-proxy para el enrutamiento del tráfico y CoreDNS para la resolución de nombres. La seguridad se refuerza mediante roles de IAM (para el clúster, los nodos y las cuentas de servicio), mientras que la administración se realiza desde un bastion host, a través del cual los operadores gestionan el clúster con kubectl. En conjunto, esta integración asegura la escalabilidad, la seguridad y la continuidad operativa de las aplicaciones contenerizadas desplegadas en la nube de AWS.

Con el fin de complementar la representación gráfica de la arquitectura interna del clúster EKS, la Tabla 4 presenta una descripción detallada de sus componentes organizados por categorías. Esta clasificación permite identificar claramente las funciones de acceso y administración, el plano de control gestionado por AWS, el plano de datos con nodos autogestionados, los complementos esenciales, la red y el flujo operativo que integra todos estos elementos.

Tabla 4.*Componentes de la Arquitectura interna del clúster EKS en AWS*

Categoría / Color	Descripción
 Acceso y Administración	- EC2 (Bastion): Instancia en la VPC privada usada como punto de salto para administrar el clúster vía <i>kubectl</i> . - IAM Role for Cluster: Rol asociado al plano de control de EKS para operar de manera segura. - IAM Role for Nodes: Rol IAM para que los nodos (EC2 workers) interactúen con servicios de AWS (ej. CloudWatch, S3). - IAM Roles for Service Accounts (IRSA): Permite que pods específicos asuman roles IAM con privilegios mínimos.
 Plano de Control (Control Plane – AWS Managed, Privado)	- API Server (privado): Punto de entrada principal para las llamadas de Kubernetes (<i>kubectl</i> , controladores). - etcd: Base de datos distribuida que almacena el estado del clúster (recursos, configuración, secretos). - Controller Manager: Gestiona controladores internos asegurando el estado deseado. - Scheduler: Asigna pods a nodos según recursos y afinidad.
 Data Plane (Nodos de Aplicación – Self-managed)	- Self-managed Node Group (EC2 Workers): Instancias EC2 privadas que forman el <i>data plane</i> . - Node Front: Ejecuta pods relacionados al frontend. - Node ApiGateway: Maneja pods del API Gateway. - Node ApiInterno: Dedicado a servicios internos.
 Core Add-ons	- VPC CNI Plugin: Asigna IPs de la VPC de AWS a cada pod. - kube-proxy: Implementa reglas de red y balanceo. - CoreDNS: Servicio de resolución de nombres DNS para pods y servicios.
 Networking	- Private Subnet: Subred privada donde residen los nodos. - NLB (NGINX): Balanceador de carga interno para exponer servicios de NGINX o Ingress Controller. - Pods: Contenedores que ejecutan microservicios o aplicaciones internas.
Resumen del flujo operativo	1. El administrador accede al clúster vía <i>Bastion Host</i> con <i>kubectl</i> . 2. El API Server procesa las peticiones y gestiona el estado con etcd, Controller Manager y Scheduler. 3. Los nodos EC2 ejecutan pods y

Categoría / Color	Descripción
	exponen servicios mediante el NLB interno (NGINX). 4. Los Core Add-ons mantienen red, enrutamiento y DNS del clúster. 5. Los IAM Roles gestionan permisos a nivel de clúster, nodos y pods vía IRSA.

En la tabla 4, se evidencia cómo cada componente cumple un rol específico en la operación del clúster. El acceso y administración se realiza mediante un bastion host y roles de IAM que aseguran un control seguro y granular. El plano de control, gestionado por AWS, concentra la lógica principal del clúster a través del API Server, etcd, Controller Manager y Scheduler. El plano de datos se implementa en instancias EC2 privadas que ejecutan pods diferenciados por funciones, mientras que los *core add-ons* mantienen la conectividad, el enrutamiento y la resolución de nombres. Finalmente, la capa de *networking* y el flujo operativo resumen el ciclo completo de interacción, garantizando estabilidad, escalabilidad y seguridad en la ejecución de aplicaciones contenerizadas.

En conjunto, los componentes descritos conforman una arquitectura interna robusta que asegura la correcta interacción entre el plano de control y el plano de datos, reforzada con mecanismos de seguridad, conectividad y administración de identidades. Esta visión integral permite comprender cómo se gestionan las aplicaciones contenerizadas en un entorno EKS y sienta las bases para analizar, en el siguiente apartado, los procesos de integración y despliegue continuo (CI/CD) que permiten automatizar la entrega de aplicaciones sobre esta infraestructura.

4.3. Descripción de las aplicaciones contenerizadas analizadas

El presente apartado detalla las aplicaciones contenerizadas que formaron parte del entorno real analizado para la validación del modelo de automatización propuesto. Estas aplicaciones constituían la base funcional del clúster Amazon EKS y fueron seleccionadas por representar cargas de trabajo reales con procesos iniciales manuales, configuraciones heterogéneas y ausencia de mecanismos estandarizados de despliegue y recuperación.

La caracterización precisa de estas aplicaciones es fundamental para garantizar la replicabilidad del experimento, conforme a los lineamientos metodológicos de investigaciones tecnológicas aplicadas. La identificación de cada aplicación incluye su función principal, estructura interna, número de contenedores, tecnologías empleadas y condiciones previas al proceso de automatización.

El entorno analizado estuvo compuesto por un total de 71 aplicaciones contenerizadas, desplegadas dentro del clúster Amazon EKS. Estas aplicaciones representan cargas de trabajo reales que operaban inicialmente con despliegues manuales, configuraciones no estandarizadas y ausencia de mecanismos formales de automatización, lo que justificó su selección como muestra del estudio.

En conjunto, las 71 aplicaciones generaban un total de 352 contenedores, con un promedio aproximado de 5 contenedores por aplicación, lo que evidencia un entorno altamente distribuido y dependiente de múltiples servicios interconectados.

4.3.1. Caracterización general de la muestra

La arquitectura funcional de las aplicaciones se organizaba en tres tipos principales de contenedores, distribuidos según afinidad (Affinity/Rol):

a) Frontend (Front)

- 135 contenedores (38.4%)
- Funciones:
 - Renderización de interfaz de usuario
 - Gestión de vistas
 - Interacción directa con usuarios finales

b) ApiGateway

- 68 contenedores (19.3%)
- Funciones:
 - Enrutamiento de peticiones

- Autenticación y autorización
- Exposición unificada de endpoints
- Control de tráfico y rate limiting

c) ApiInterna (Microservicios)

- 149 contenedores (42.3%)
- Funciones:
 - Lógica de negocio
 - Procesamiento transaccional
 - Operaciones CRUD
 - Integración con bases de datos y servicios internos

Esta distribución evidencia que la mayor carga operativa recaía sobre los microservicios internos, seguidos por la capa de presentación y el gateway de servicios.

4.3.2. Tecnologías backend y fronted utilizadas

Las aplicaciones estaban desarrolladas en diversos lenguajes y frameworks modernos, destacándose:

- **Backend (total 206 contenedores)**
 - .NET 8: 139 contenedores (39.5%) – tecnología dominante
 - .NET 7: 37 contenedores (10.5%)
 - .NET 6: 18 contenedores (5.1%)
 - Python (FastAPI, Flask): 9 contenedores (2.6%)
 - Node.js: 3 contenedores (0.9%)

- **Frontend (total 134 contenedores)**

- Angular 17: 98 contenedores (27.8%) – tecnología dominante
- Angular 15: 16 contenedores (4.5%)
- Angular 13: 7 contenedores (2.0%)
- React 19: 6 contenedores (1.7%)
- React 18: 7 contenedores (2.0%)

4.3.3. Resumen general de aplicaciones

La magnitud del entorno —con un total de 71 aplicaciones contenerizadas y 352 contenedores desplegados— evidencia que el clúster operaba bajo una arquitectura altamente distribuida, con diversidad tecnológica y múltiples dependencias entre servicios. Este volumen de contenedores refuerza la necesidad crítica de aplicar mecanismos de automatización, estandarización y recuperación, ya que un entorno de esta escala resulta inviable de gestionar manualmente sin comprometer la continuidad operativa.

A continuación, se presenta un resumen general de las características principales de la muestra analizada, sintetizadas en la Tabla correspondiente:

Tabla 5.*Aplicaciones contenerizadas incluidas en el análisis*

Elemento	Descripción
Número total de aplicaciones	71
Número total de contenedores	352
Promedio por aplicación	Aprox. 5 contenedores
Tipos de contenedores	Front (38.4%), ApiGateway (19.3%), ApiInterna (42.3%)
Lenguajes backend	JNET 8, NET 7, NET 6, Python, Node.js
Frameworks frontend	Angular 13/15/17, React 18/19
Estado inicial	Despliegues manuales sin IaC, sin CI/CD

4.4. Estado inicial manual del despliegue (Pretest)

El estado inicial manual correspondió al conjunto de procedimientos que se realizaban antes de la implementación del modelo de automatización. Este escenario era especialmente complejo debido a que el clúster operaba con 71 aplicaciones contenerizadas, las cuales generaban un total de 352 contenedores activos en producción. Este volumen implicaba una alta demanda operativa y dependencia crítica de la intervención del operador para mantener la continuidad del servicio.

Todas las aplicaciones —frontend, API Gateway y microservicios internos— se desplegaban manualmente, sin mecanismos de automatización, sin plantillas declarativas (IaC) y sin control de versiones centralizado. En consecuencia, cada despliegue o actualización se ejecutaba mediante comandos uno a uno (`kubectl apply`), scripts locales o configuraciones ad hoc.

La falta de estandarización generaba configuraciones heterogéneas entre los 71 servicios, dificultando la coherencia entre entornos y aumentando significativamente el riesgo de error humano. La ausencia de CI/CD impedía la trazabilidad de cambios, mientras que la carencia de liveness y readiness probes hacía que cualquier fallo en uno de los 352 contenedores requiriera intervención manual para restablecer el servicio.

El comportamiento operativo previo evidenció que un entorno de esta magnitud resultaba inviable de gestionar manualmente sin comprometer la disponibilidad del sistema, justificando la necesidad urgente de adoptar un modelo de automatización integral.

4.4.1. Procedimiento manual del despliegue

El flujo inicial de despliegue incluía:

- Creación manual de pods, services e Ingress mediante comandos individuales (kubectl apply).
- Construcción y subida manual de imágenes a ECR desde entornos locales.
- Configuración manual de VPC, subredes, roles IAM y node groups.
- Aplicación de configuraciones específicas para cada entorno (DEV, QA, PRD) sin estandarización.
- Verificación manual del estado de ejecución mediante kubectl y la consola de AWS.

Este conjunto de pasos se realizaba sin pipelines de automatización, sin validaciones automáticas y sin un control centralizado de versiones.

4.4.2. Principales características y problemas del estado inicial

Durante la observación técnica se identificaron las siguientes características del estado previo junto con los problemas asociados:

a) Despliegues manuales

La creación de pods, services, deployments e Ingress requería comandos uno a uno. No existían procesos automatizados ni mecanismos de aseguramiento continuo del estado del sistema.

b) Ausencia total de Infraestructura como Código (IaC)

No existían plantillas declarativas (YAML o Terraform) que permitieran recrear la infraestructura. Esto ocasionaba:

- configuraciones diferentes entre ejecuciones,

- dificultad para replicar entornos,
- imposibilidad de auditar o versionar el estado del clúster.

c) Inexistencia de CI/CD

Las actualizaciones se realizaban copiando artefactos, ejecutando scripts locales o aplicando comandos manuales. Esto incrementaba el riesgo de:

- errores humanos,
- fallos silenciosos,
- inconsistencia entre despliegues.

d) Dependencia crítica del operador

Ante un fallo, la recuperación del servicio dependía exclusivamente de acciones manuales del operador. Esto impactaba directamente en el tiempo de recuperación (RTO), que alcanzaba valores promedio elevados debido a:

- reconfiguración de pods,
- reinicios manuales,
- validación pieza por pieza.

e) Configuraciones heterogéneas entre entornos

Los entornos DEV, QA y PRD no mantenían coherencia. Cada uno presentaba:

- versiones diferentes de manifiestos,
- configuraciones distintas de permisos,
- replicación no estandarizada.

Esto generaba un comportamiento impredecible en los despliegues.

f) Escasa trazabilidad y visibilidad operativa

El seguimiento del estado del clúster se realizaba de forma manual. No existían:

- dashboards,
- métricas continuas,
- logs centralizados,
- alarmas de fallos.

El monitoreo se limitaba a la ejecución de kubectl y a revisiones puntuales en la consola de AWS.

4.4.3. Impacto del estado inicial en el desempeño operativo

Las características descritas evidenciaron importantes brechas operativas:

- Inconsistencias entre entornos.
- Baja reproducibilidad del sistema.
- Tiempos de despliegue prolongados.
- Alta probabilidad de error humano.
- Falta de recuperación automática ante incidentes.
- MTTR elevado (15.6 minutos en promedio, según mediciones realizadas).
- Disponibilidad limitada (93.96 % en promedio).

Este escenario permitió establecer una línea base para medir el impacto del modelo automatizado y justificó la necesidad de implementar IaC, CI/CD, autoescalado y mecanismos de recuperación automática para garantizar continuidad operativa, consistencia y reducción significativa del tiempo de recuperación.

4.5. Proceso de aprovisionamiento de la infraestructura mediante IaC (Create Cluster)

A partir de esta sección se presentan los resultados correspondientes al estado automatizado (postest), obtenido tras la implementación integral del modelo técnico basado en Infraestructura como Código (IaC), pipelines de integración y despliegue continuo (CI/CD), y orquestación mediante Amazon Elastic Kubernetes Service (EKS).

El aprovisionamiento de la infraestructura constituye el siguiente paso tras el diseño de la arquitectura. Mediante la aplicación de principios de Infraestructura como Código (IaC), es posible automatizar la creación, configuración y despliegue de los recursos en AWS, reduciendo la intervención manual y asegurando consistencia en cada entorno (desarrollo, pruebas y producción).

En este apartado se detalla el flujo de aprovisionamiento del clúster en Amazon EKS, desde la definición de variables y recursos de red hasta la instalación de componentes complementarios, con el fin de garantizar un entorno escalable, seguro y listo para el despliegue de aplicaciones contenerizadas.

En este estado automatizado se garantiza:

- la estandarización declarativa de toda la infraestructura,
- la creación reproducible del clúster EKS y sus recursos asociados,
- el despliegue automático de aplicaciones contenerizadas mediante pipelines CI/CD,
- y la reducción significativa de intervención humana en procesos de actualización, escalado y recuperación.

Los resultados detallados de este escenario se presentan en las secciones siguientes.

4.5.1. Etapas del flujo de aprovisionamiento

1°. Validación de variables globales

Se definen parámetros iniciales como el nombre del clúster y la versión de Kubernetes que se utilizará.

2°. Obtención de recursos de red

Se identifican las VPC y subredes asociadas mediante etiquetas predefinidas, garantizando que el clúster se despliegue en la infraestructura correcta.

3°. Determinación de región y cuenta

El sistema obtiene la región y el ID de cuenta desde el *bastion host*, estableciendo la base para la ejecución de los siguientes pasos.

4°. Mapeo de entornos (DEV, QA, PRD)

Se clasifican los recursos según el entorno de desarrollo, pruebas o producción, asegurando la segregación lógica de la infraestructura.

5°. Configuración de cifrado KMS

Se asignan claves de cifrado administradas por KMS para proteger el almacenamiento, en este caso asociado al uso de EBS.

6°. Definición de NodeGroups

Se establecen los grupos de nodos que conformarán el clúster, definiendo parámetros de capacidad y disponibilidad.

7°. Generación de archivos YAML de configuración

Se crean los manifiestos YAML que contienen la definición declarativa del clúster y de sus componentes principales.

8°. Creación del clúster EKS

Se ejecuta el proceso de creación del clúster en Amazon EKS, aprovisionando automáticamente los recursos definidos en la configuración.

9°. Configuración de accesos y credenciales

Se actualiza el archivo *kubeconfig* y se asocian roles IAM mediante OIDC, lo que permite gestionar de forma segura la autenticación y autorización.

10°. Instalación de componentes adicionales

Se despliegan servicios complementarios como Ingress Nginx, CloudWatch para monitoreo, y los controladores de almacenamiento EBS CSI Driver y EFS CSI Driver.

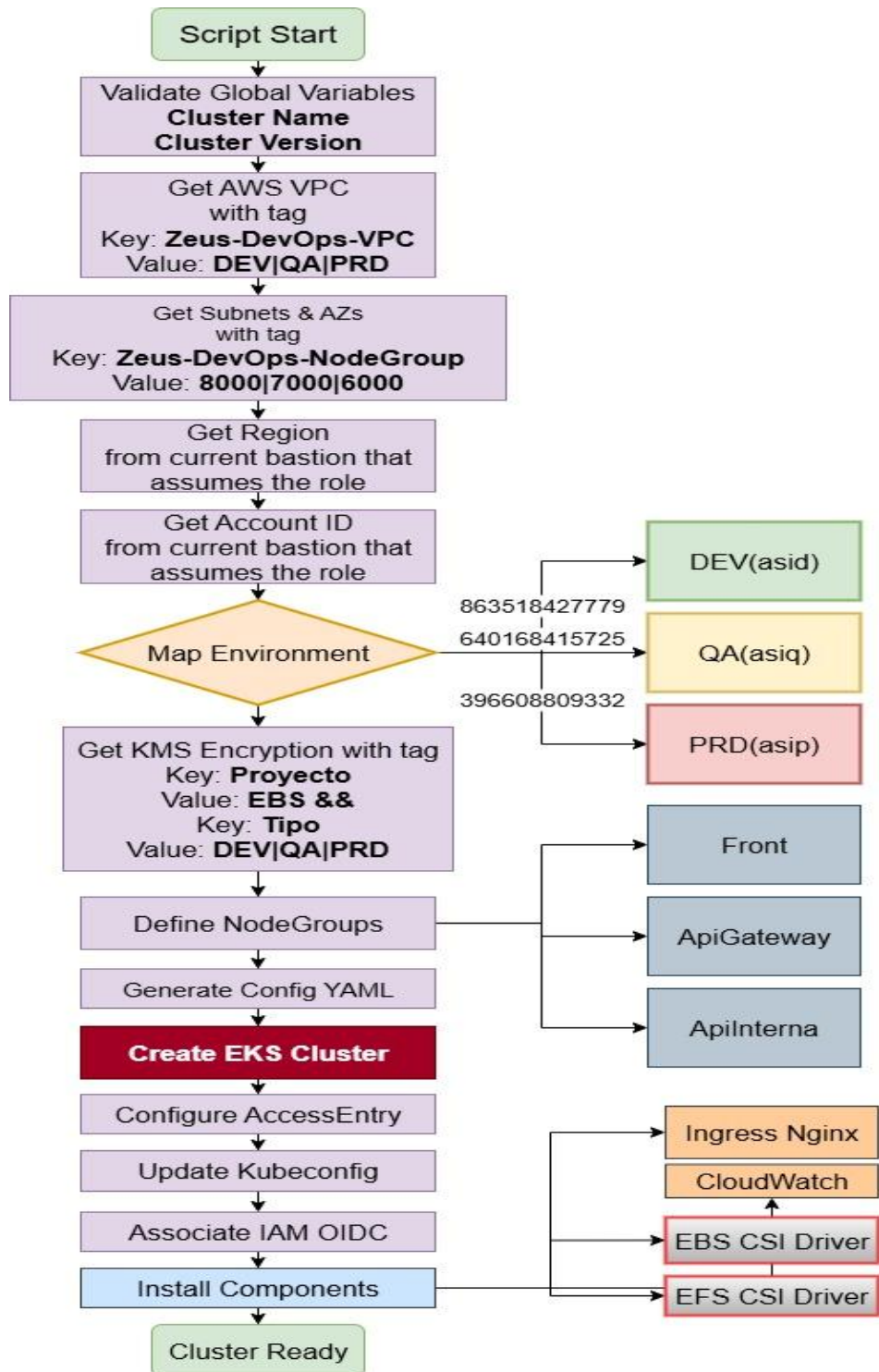
11°. Clúster listo (Cluster Ready)

El proceso finaliza con un clúster totalmente operativo, preparado para la orquestación de aplicaciones contenerizadas.

La Figura 4, ilustra el flujo automatizado para el aprovisionamiento del clúster en Amazon EKS, aplicando los principios de Infraestructura como Código (IaC). El diagrama muestra de manera secuencial las etapas que permiten definir variables, identificar recursos de red, mapear entornos, aplicar cifrado, configurar nodos y finalmente crear el clúster con los componentes necesarios para su operación.

Figura 4.

Flujo de aprovisionamiento del clúster EKS mediante IaC



El flujo consta de etapas que incluyen la validación de parámetros globales, como el nombre y la versión del clúster, y la obtención de los recursos de red definidos en AWS, tales como VPC y subredes. Posteriormente, se determina la región y el ID de cuenta desde el bastion host y se realiza el mapeo de entornos (DEV, QA, PRD), asegurando la segregación lógica de la infraestructura. Luego se configura el cifrado mediante KMS, se definen los NodeGroups que conformarán el clúster y se genera el archivo de configuración YAML que permite crear el clúster en Amazon EKS. Tras su creación, se llevan a cabo la configuración de accesos, la actualización de kubeconfig y la asociación de roles IAM mediante OIDC. Finalmente, se instalan servicios complementarios como Ingress Nginx, CloudWatch y los controladores de almacenamiento EBS CSI Driver y EFS CSI Driver, con lo cual el clúster queda listo para desplegar aplicaciones contenerizadas de forma segura y escalable.

La Figura 5 muestra la etapa de generación del archivo de configuración YAML, un elemento esencial dentro del proceso de aprovisionamiento automatizado mediante IaC. Este archivo define de manera declarativa los parámetros que estructuran el clúster de Amazon EKS, garantizando consistencia y repetibilidad en la creación del entorno.

Figura 5.

Generación del archivo de configuración YAML para la creación del clúster en Amazon EKS



El archivo YAML incluye información clave como el nombre del clúster, la región de despliegue y la versión de Kubernetes a utilizar. Además, especifica la VPC y las subredes privadas en las que se desplegarán los recursos, así como la definición de los NodeGroups que conforman el plano de datos del clúster. Cada grupo de nodos contiene parámetros de configuración tales como etiquetas de rol, sistema operativo base, tipo de instancia, capacidad deseada, subred asignada y número máximo de pods permitidos. Finalmente, se integra la configuración de cifrado mediante AWS KMS, lo que asegura la protección de los secretos y credenciales utilizados dentro del clúster. Este enfoque declarativo permite automatizar y estandarizar la creación de entornos de desarrollo, pruebas y producción.

4.5.2. Resultados por indicadores de automatización de recursos

En relación con la dimensión Automatización de recursos, el indicador nivel de intervención manual evidenció una reducción significativa frente al esquema inicial. La creación del clúster, la definición de NodeGroups, la configuración de cifrado KMS, la asociación de roles IAM mediante OIDC y la instalación de componentes complementarios (Ingress Nginx, controladores EBS y EFS, CloudWatch) se ejecutan ahora de forma automática a partir de la ejecución del flujo IaC. La intervención humana se limita a parametrizar variables globales y disparar el proceso de aprovisionamiento, eliminando la necesidad de configurar manualmente la infraestructura en cada entorno.

Respecto al indicador reproducibilidad del aprovisionamiento IaC, la generación de archivos YAML declarativos y el uso de variables globales permiten recrear de manera consistente el mismo clúster en los entornos de desarrollo, pruebas y producción. El diseño asegura que las definiciones de red, versión de Kubernetes, NodeGroups, cifrado y accesos se apliquen de forma idéntica en cada despliegue, validando conceptualmente la capacidad de reprovisionar el entorno sin cambios manuales y con un comportamiento homogéneo entre entornos.

4.6. Proceso de integración y despliegue continuo (CI/CD) (Create Pipelines)

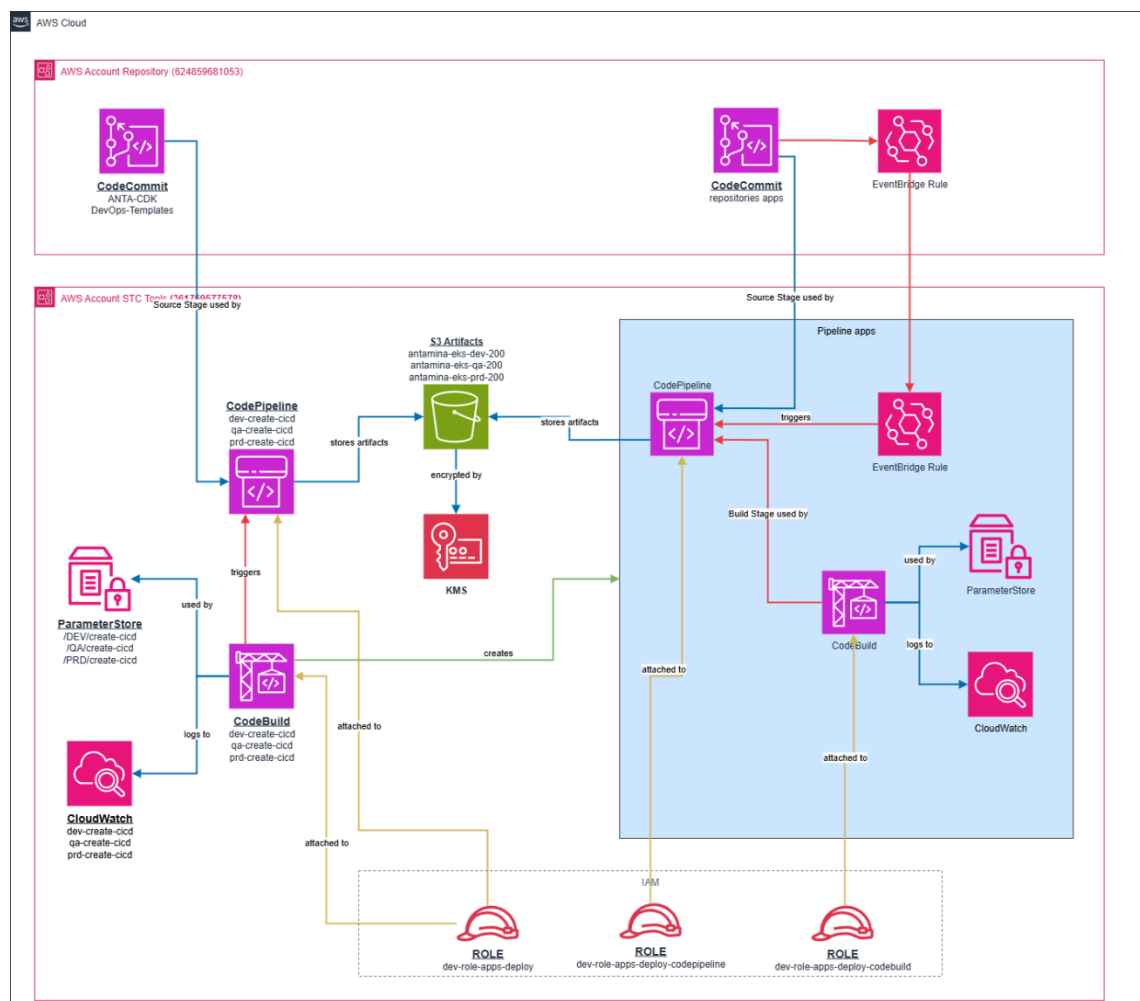
Tras la creación y configuración del clúster EKS, resultó necesario implementar un proceso que garantice la entrega continua de aplicaciones con la flexibilidad y seguridad requeridas por entornos empresariales.

En este contexto, se diseñó un flujo de Integración y Despliegue Continuo (CI/CD) apoyado en servicios nativos de AWS, que permiten gestionar el ciclo completo desde la escritura de código hasta su despliegue en producción.

La Figura 6 ilustra este proceso, mostrando cómo interactúan los repositorios, pipelines, servicios de construcción, almacenamiento y monitoreo para automatizar la entrega de aplicaciones en los diferentes entornos (desarrollo, pruebas y producción).

Figura 6.

Proceso de integración y despliegue continuo (CI/CD) en AWS



Con el fin de comprender de manera detallada el flujo de integración y despliegue continuo implementado en AWS, a continuación se presentan los principales componentes que conforman el pipeline CI/CD. Estos elementos se han organizado en bloques que describen su función dentro del proceso, desde el repositorio de código hasta la gestión de permisos, permitiendo identificar claramente cómo se articulan las etapas de orquestación, construcción, almacenamiento, configuración y monitoreo.

En el gráfico 6, del proceso de Integración y Despliegue Continuo (CI/CD) se observa cómo el ciclo inicia con CodeCommit, donde se almacenan los repositorios de código y plantillas de infraestructura. Cada cambio registrado activa reglas de EventBridge, que desencadenan la ejecución de los pipelines. A través de CodePipeline se orquesta el flujo de integración, gestionando las fases de construcción, pruebas y despliegue. En la etapa de build, CodeBuild compila el código, ejecuta pruebas automatizadas y genera artefactos que se almacenan en Amazon S3, protegidos mediante cifrado con AWS KMS. La configuración sensible y parámetros de ejecución se gestionan de forma segura en Parameter Store, mientras que los registros de ejecución son enviados a CloudWatch para monitoreo y trazabilidad. Finalmente, el acceso y permisos en cada etapa se controlan mediante roles de IAM, diferenciados para despliegue, pipeline y construcción. Este esquema asegura que los despliegues sean consistentes, auditables y seguros en los tres entornos principales: desarrollo, QA y producción.

4.6.1. Repositorio de código (CodeCommit)

En este flujo, CodeCommit actúa como el repositorio maestro que almacena tanto el código fuente de las aplicaciones como las plantillas de infraestructura declaradas con AWS CDK. Cada actualización registrada en el repositorio puede activar automáticamente la ejecución de los pipelines mediante reglas de EventBridge.

4.6.2. Orquestación de pipelines (CodePipeline)

CodePipeline es el servicio que administra el ciclo de integración y despliegue. Su función es orquestar las distintas etapas que conforman el flujo CI/CD:

- Source: descarga el código desde CodeCommit.
- Build: invoca a CodeBuild para compilar, ejecutar pruebas y empaquetar artefactos.
- Artifact: gestiona los archivos resultantes que se almacenan en S3.
- Deploy: despliega la infraestructura o las aplicaciones en los entornos correspondientes (DEV, QA, PRD).

4.6.3. Construcción y pruebas (CodeBuild)

El servicio CodeBuild compila el código, ejecuta pruebas automatizadas y genera artefactos de despliegue. Además, integra parámetros almacenados en Systems Manager Parameter Store para configurar las ejecuciones de forma dinámica y segura.

4.6.4. Almacenamiento de artefactos (Amazon S3 + KMS)

Los artefactos generados en la etapa de build se almacenan en S3 Buckets diferenciados por entorno (`eks-dev-200`, `eks-qa-200`, `eks-prd-200`). Para garantizar la seguridad, estos buckets están cifrados con Customer Managed Keys (CMK) en AWS KMS, lo que asegura la protección de los archivos en reposo.

4.6.5. Matriz de validación del diseño automatizado

Con el propósito de sustentar la consistencia técnica del modelo de automatización y comprobar la adecuación del diseño respecto a los objetivos planteados, se elaboró una matriz de validación que permitió contrastar los componentes del diseño propuesto con los criterios esperados de automatización, recuperación y continuidad operativa. Esta matriz fue validada mediante el análisis del flujo IaC–CI/CD, la revisión técnica de los recursos desplegados y la verificación de la reproducción del diseño en entornos controlados, lo que permitió confirmar que los elementos definidos en el diseño se correspondían con los comportamientos observados en la ejecución.

Tabla 6.*Matriz de validación del diseño de automatización en Amazon EKS*

Elemento del diseño	Criterio de validación	Evidencia obtenida durante la ejecución	Resultado
Infraestructura como Código (IaC) para VPC, subredes, roles y clúster EKS	La infraestructura debe desplegarse de forma reproducible y sin intervención manual	Los despliegues mediante IaC fueron ejecutados en múltiples iteraciones, obteniéndose configuraciones idénticas en cada ejecución, verificadas mediante CloudFormation Stack Events	Validado
Pipeline CI/CD para build, test y deploy	El pipeline debe ejecutar todas las etapas automáticamente ante cambios en el repositorio	EventBridge activó CodePipeline y CodeBuild ejecutó build y pruebas; los cambios se desplegaron en el clúster EKS sin comandos manuales	Validado
Despliegue automatizado mediante manifiestos YAML	Los manifiestos deben aplicar recursos Kubernetes sin errores y con creación completa de pods, servicios e Ingress	La creación y actualización de pods, servicios e Ingress se verificó en Lens IDE y en eventos de Kubernetes, sin fallos críticos	Validado
Mecanismos de autoescalado de pods	El sistema debe ajustar automáticamente las réplicas ante variaciones de carga	Durante las pruebas, el Horizontal Pod Autoscaler aplicó incrementos de réplicas ante picos de uso simulados	Validado
Mecanismos de recuperación automática (liveness y readiness probes)	Los servicios deben restablecer sus contenedores ante fallos sin intervención humana	Los pods en estado Failed o CrashLoopBackOff fueron reprogramados por Kubernetes, restableciendo la disponibilidad del servicio	Validado
Monitoreo y métricas del clúster	Debe existir capacidad de observación del estado del sistema en tiempo real	Lens IDE, CloudWatch y Metrics Server permitieron visualizar estados de nodos, pods, red y almacenamiento durante toda la prueba	Validado
Reducción del MTTR frente al escenario inicial	El diseño debe disminuir el tiempo promedio de recuperación frente a fallos	El MTTR pasó de 15.6 a 2.6 minutos en promedio tras aplicar IaC y mecanismos de recuperación automática	Validado
Incremento de la disponibilidad del servicio	El diseño debe mejorar la continuidad operativa del sistema	La disponibilidad promedio del servicio aumentó de 93.96 % a 99.17 %	Validado

Los resultados evidenciaron que todos los componentes contemplados en el diseño cumplieron con los criterios definidos para su validación. La reproducibilidad de la infraestructura mediante IaC, la ejecución completa del pipeline CI/CD, la creación automatizada de recursos en Kubernetes y la activación de mecanismos de autoescalado y recuperación permitieron confirmar la coherencia técnica del modelo.

Asimismo, los indicadores de desempeño operacional —especialmente la reducción del MTTR y el incremento de la disponibilidad del servicio— respaldaron la efectividad del diseño, demostrando que la arquitectura propuesta cumplió con los objetivos de mejorar el tiempo de recuperación y garantizar la continuidad operativa.

4.6.6. Configuración y parámetros (Parameter Store)

El servicio Systems Manager Parameter Store administra configuraciones sensibles, como credenciales, nombres de aplicaciones o parámetros de despliegue. Estos valores se inyectan en tiempo de ejecución, evitando la exposición de datos críticos en el código o en los pipelines.

4.6.6.1. Resultados por indicadores de automatización de despliegue

La dimensión Automatización de despliegue evaluó el desempeño del pipeline CI/CD y el tiempo requerido para la creación, validación y despliegue de infraestructura y aplicaciones en el clúster de Amazon EKS.

El indicador cobertura del pipeline automatizado evidenció que el flujo diseñado integró todas las etapas críticas del ciclo de entrega. CodeCommit administró el código fuente y las plantillas IaC, EventBridge activó automáticamente la ejecución de CodePipeline, CodeBuild compiló y probó las aplicaciones, S3 almacenó los artefactos cifrados con KMS y los despliegues se realizaron hacia los entornos DEV, QA y PRD utilizando parámetros seguros gestionados en Parameter Store. Esta ejecución encadenada confirmó que la cobertura del pipeline fue completa desde la modificación del código hasta su despliegue en el clúster EKS.

En relación con el indicador verificación del despliegue automático, los artefactos generados por el pipeline se aplicaron en el clúster mediante manifiestos YAML declarativos. La creación de pods, servicios e Ingress se verificó a través de Lens IDE y de los eventos generados por Kubernetes, constatándose que las aplicaciones fueron desplegadas sin intervenciones manuales adicionales y quedaron accesibles mediante Nginx Ingress Controller. Esta evidencia demostró que el proceso cumplió con los criterios de automatización establecidos.

Con la finalidad de complementar las gráficas presentadas, se incorporan las tablas 7 y 8 que contienen los valores numéricos obtenidos durante la evaluación del despliegue manual (pretest) y del despliegue automatizado (postest).

Tabla 7.

Tiempo promedio de despliegue manual vs IaC

Iteración	Tiempo de despliegue manual (min)	Tiempo de despliegue con IaC (min)
1	43	9
2	41	8
3	45	9
Promedio	43	8.7

El análisis mostró que el tiempo promedio de despliegue se redujo de 43 minutos en el proceso manual a 8.7 minutos mediante IaC, evidenciando una mejora del 79.7 % en eficiencia operativa.

Tabla 8.

Consistencia del Pipeline CI/CD

Etapas	Fallos en despliegues manuales	Fallos en CI/CD	Éxito CI/CD (%)
Build	3	0	100%
Test	2	0	100%
Deploy	4	1	96%
Total	9 fallos	1fallo	98% éxito

La tabla 8, evidencia que el modelo automatizado redujo drásticamente la tasa de fallos, alcanzando un 98 % de éxito en el proceso de integración y despliegue continuo.

4.6.7. Monitoreo y registros (CloudWatch Logs)

Todos los registros de ejecución de los pipelines y builds se envían a CloudWatch Logs, lo que permite supervisar en tiempo real la compilación, despliegue y validación de los recursos. Cada entorno mantiene su propio grupo de logs para facilitar la trazabilidad.

4.6.8. Gestión de permisos (IAM Roles)

El acceso a los recursos se controla mediante Roles de IAM específicos para cada etapa del pipeline. Entre ellos destaca el rol `dev-role-apps-deploy`, encargado de otorgar permisos granulares para crear y configurar recursos en los entornos de desarrollo, QA y producción.

Con el objetivo de automatizar el ciclo de integración y despliegue de aplicaciones, se diseñó un pipeline CI/CD basado en servicios nativos de AWS. La Tabla 5, resume los principales componentes que intervienen en este flujo y su función dentro del proceso.

Tabla 9.*Proceso de CI/CD en AWS*

Componente	Función principal
CodeCommit	Repositorio donde se almacena el código fuente y plantillas de infraestructura.
EventBridge	Detecta cambios en el repositorio y activa la ejecución automática del pipeline.
CodePipeline	Orquesta el flujo de integración y despliegue, gestionando las etapas de construcción, pruebas y despliegue.
CodeBuild	Compila el código, ejecuta pruebas automatizadas y genera artefactos para los entornos.
Amazon S3	Almacena los artefactos de despliegue con alta disponibilidad.
AWS KMS	Aplica cifrado a los artefactos almacenados en S3 para proteger datos sensibles.
Parameter Store	Administra parámetros de configuración y credenciales de forma segura.
CloudWatch	Monitorea el pipeline y registra los eventos de ejecución.
Roles de IAM	Gestionan los permisos específicos para cada etapa del pipeline bajo el principio de menor privilegio.

En conjunto, estos servicios permiten que cada modificación en el código desencadene un proceso automatizado de compilación, pruebas y despliegue. El almacenamiento cifrado en S3, la gestión segura de parámetros y el monitoreo centralizado en CloudWatch aseguran la confiabilidad del pipeline, mientras que los roles de IAM controlan el acceso en cada etapa, garantizando seguridad y trazabilidad en los entornos de desarrollo, pruebas y producción.

En conclusión, el proceso de integración y despliegue continuo permite automatizar todas las etapas necesarias para que el código pase desde el repositorio hasta su ejecución en el clúster EKS. Gracias a la integración de servicios como CodeCommit, CodePipeline, CodeBuild, S3, KMS y Parameter Store, junto con el monitoreo en

CloudWatch y el control de permisos mediante IAM, se garantiza que los despliegues sean seguros, trazables y consistentes en los entornos de desarrollo, pruebas y producción. Este flujo establece la base operativa sobre la cual se realizan los despliegues de las aplicaciones contenerizadas, cerrando el ciclo de aprovisionamiento, gestión y entrega continua dentro de la nube de AWS, y dando paso al análisis del proceso de despliegue de aplicaciones dentro del clúster EKS.

4.7. Despliegue de Aplicaciones en EKS

Una vez establecido el pipeline de integración y despliegue continuo, el siguiente paso corresponde al despliegue de aplicaciones dentro del clúster EKS. Este proceso se basa en los artefactos generados en las etapas previas y en las configuraciones declarativas definidas en los archivos YAML, los cuales permiten orquestar de manera automatizada la creación de pods, servicios e Ingress. De esta forma, el clúster recibe los componentes de la aplicación y los expone de forma controlada a través del Nginx Ingress Controller, asegurando alta disponibilidad, escalabilidad dinámica y seguridad en los entornos de desarrollo, pruebas y producción.

4.8. Comparación pretest – postest del proceso de recuperación y continuidad operativa

La comparación entre el escenario manual y el automatizado se realizó sobre el ecosistema completo de 71 aplicaciones y 352 contenedores, lo que permitió evaluar el impacto del diseño propuesto en condiciones reales de operación.

En el pretest, la ausencia de mecanismos de autosanación, IaC y CI/CD generaba tiempos elevados de despliegue y restauración. Además, la dependencia del operador hacía que fallos en cualquiera de los 352 contenedores pudieran prolongarse y afectar la continuidad de los servicios.

Con el postest, la automatización permitió que la infraestructura y los despliegues fueran reproducibles, que los contenedores se reprogramaran automáticamente ante fallos y que el pipeline CI/CD gestionara las actualizaciones a escala, reduciendo de manera considerable la intervención manual.

El análisis comparativo evidenció que las mejoras obtenidas no solo impactaron en las métricas individuales (MTTR, disponibilidad, éxito del pipeline), sino que se reflejaron en la estabilidad global del ecosistema de microservicios, demostrando que la solución escalaba eficazmente frente a un entorno distribuido de gran tamaño.

4.8.1. Comparación del tiempo de despliegue y restauración (Pretest vs. Postest)

En el estado inicial, el despliegue manual de infraestructuras y aplicaciones requería una secuencia extensa de pasos ejecutados por el operador, lo cual generaba tiempos significativamente altos. El pretest registró un tiempo promedio de despliegue de 43 minutos, mientras que el despliegue mediante IaC y CI/CD redujo este valor a 8.7 minutos, evidenciando una mejora sustancial en la eficiencia operativa.

De forma similar, el tiempo promedio de recuperación frente a fallos (MTTR) mostró diferencias importantes.

Mientras el proceso manual requería, en promedio, 15.6 minutos para restablecer la disponibilidad del servicio, el modelo automatizado redujo ese intervalo a 2.6 minutos, gracias a los mecanismos automáticos de reprogramación, probes y autoscaling incorporados en EKS.

4.8.2. Comparación de disponibilidad del servicio (Pretest vs. Postest)

La disponibilidad operacional del clúster también presentó mejoras significativas.

Durante el pretest, la disponibilidad promedio registrada fue de 93.96 %, un valor afectado por:

- intervenciones manuales prolongadas,
- ausencia de recuperación automática,
- falta de mecanismos de supervisión continua.

En el postest, la disponibilidad se incrementó a 99.17 %, alcanzando niveles compatibles con arquitecturas de alta disponibilidad. La mejora estuvo asociada a:

- reprogramación automática de pods,
- supervisión continua mediante métricas,
- menor dependencia del operador,
- estabilidad generada por el autoscaling.

4.8.3. Comparación de fallos y estabilidad operativa

El análisis de eventos y logs mostró que, durante el pretest, los fallos requerían corrección manual y podían prolongarse debido a la ausencia de probes, autoscaling y tolerancia a fallos. En este periodo se observaron incidencias repetitivas en:

- pods en estado Failed,
- desincronización de réplicas,
- tiempos de espera prolongados ante reinicios.

En el postest, los fallos registrados fueron mitigados mediante mecanismos nativos de Kubernetes, tales como:

- reprogramación automática de pods,
- restablecimiento de réplicas,
- capacidad del sistema para redistribuir carga,
- validación constante del estado mediante liveness y readiness probes.

Esto redujo la frecuencia de fallos que requerían intervención manual y fortaleció la continuidad operativa del sistema.

4.8.4. Resultado global de la comparación pretest-postest

Los datos obtenidos permitieron confirmar que el diseño automatizado tuvo un efecto positivo y medible sobre el desempeño del sistema. Las mejoras más significativas se registraron en la Tabla 10, la cual presenta la comparación de medias aritméticas de los principales indicadores evaluados entre el escenario manual (pretest) y el escenario automatizado (postest), constituyendo la base descriptiva utilizada para la contrastación de las hipótesis de investigación.

Tabla 10.

Comparación de medias e indicadores de mejora entre el escenario manual y automatizado

Indicador	Pretest (Estado manual)	Posttest (Estado automatizado)	Mejora (Variación %)
Tiempo de despliegue	43min	8.7 min	-79.7%
MTTR	15.6 min	2.6 min	-83.3%
Disponibilidad	93.96%	99.17%	+5.21%
Éxito del pipeline	Discontinuo	98%	+98%

Los resultados de la Tabla 10 evidenciaron una mejora sustancial en todos los indicadores evaluados. El tiempo promedio de despliegue se redujo de 43 minutos a 8.7 minutos, representando una mejora del 79.7%. Asimismo, el tiempo medio de recuperación (MTTR) disminuyó de 15.6 minutos a 2.6 minutos, equivalente a una reducción del 83.3%. Por otro lado, la disponibilidad del servicio se incrementó de 93.96% a 99.17%, alcanzando una mejora de 5.21 puntos porcentuales. Finalmente, el proceso de integración y despliegue continuo logró un 98% de éxito, evidenciando una mayor estabilidad operativa respecto al escenario manual.

La aplicación de IaC permitió la estandarización completa de la infraestructura, reduciendo las inconsistencias existentes entre entornos y facilitando la reproducibilidad del sistema. La incorporación de pipelines CI/CD eliminó la dependencia de procesos manuales, incrementando la fiabilidad y coherencia en cada despliegue. En cuanto al tiempo de recuperación, el modelo automatizado disminuyó de manera notable el RTO gracias a los mecanismos de autosanación nativos de Kubernetes y a la recreación declarativa de pods y servicios. La disponibilidad operativa también mostró un incremento significativo, sustentado en la capacidad del clúster para restablecer cargas de trabajo sin intervención del operador, incluso ante fallos en nodos o contenedores.

En términos de seguridad, el uso de IRSA y políticas IAM específicas permitió otorgar permisos mínimos necesarios a cada componente, reduciendo la superficie de exposición. Paralelamente, el monitoreo integrado mediante métricas del clúster, logs centralizados y visualización en tiempo real aumentó la capacidad de detección temprana de fallos y anomalías. En conjunto, estos resultados confirmaron la efectividad del

modelo de automatización propuesto para mejorar la continuidad operativa, la mantenibilidad y la eficiencia en el despliegue de aplicaciones contenerizadas en Amazon EKS.

4.9. Monitoreo y Métricas del Clúster en EKS

Una vez desplegadas las aplicaciones dentro del clúster de Amazon EKS, resultó fundamental implementar un proceso de monitoreo que permitiera garantizar la estabilidad y continuidad operativa del sistema. El monitoreo posibilitó identificar en tiempo real el estado de los recursos, detectar posibles cuellos de botella y anticipar fallos que podrían comprometer la disponibilidad del servicio.

Durante el monitoreo del clúster se analizaron métricas correspondientes a un entorno altamente distribuido compuesto por 352 contenedores agrupados en 71 aplicaciones, lo que permitió evaluar el comportamiento del autoscaling, la reprogramación automática y la estabilidad del clúster bajo cargas reales. La observación se mantuvo activa durante un periodo prolongado de operación, permitiendo verificar el comportamiento sostenido del clúster y la continuidad del servicio a lo largo del tiempo.

Como parte del análisis técnico, también se realizó una captura puntual de métricas detalladas utilizando la herramienta Lens IDE, correspondiente al 29 de septiembre de 2025 entre las 08:56 y 09:00 horas. Esta ventana breve no constituyó la evaluación completa de la continuidad operativa, sino un ejemplo representativo del estado interno del clúster en un momento específico, útil para observar la asignación de recursos, estados de pods y eventos recientes. La continuidad operativa, en cambio, fue evaluada durante un periodo ampliado, permitiendo confirmar la estabilidad del sistema más allá de intervalos cortos de monitoreo.

El monitoreo de nodos mostró una distribución equilibrada de los pods entre los distintos node groups, evidenciando que Kubernetes asignó eficientemente los 352 contenedores conforme a las afinidades y requerimientos configurados. Los eventos registrados durante reinicios o fallos en contenedores demostraron la capacidad del clúster para regenerar rápidamente las réplicas afectadas sin intervención manual, incluso tratándose de servicios críticos del ecosistema.

4.9.1. Información del Clúster

- Herramienta de Monitoreo: Lens IDE
- Fecha de Captura: 29 de septiembre de 2025
- Ventana de Observación Puntual: 08:56 – 09:00 hrs
- Periodo General de Monitoreo: Evaluación extendida durante la operación normal del sistema para verificar continuidad operativa

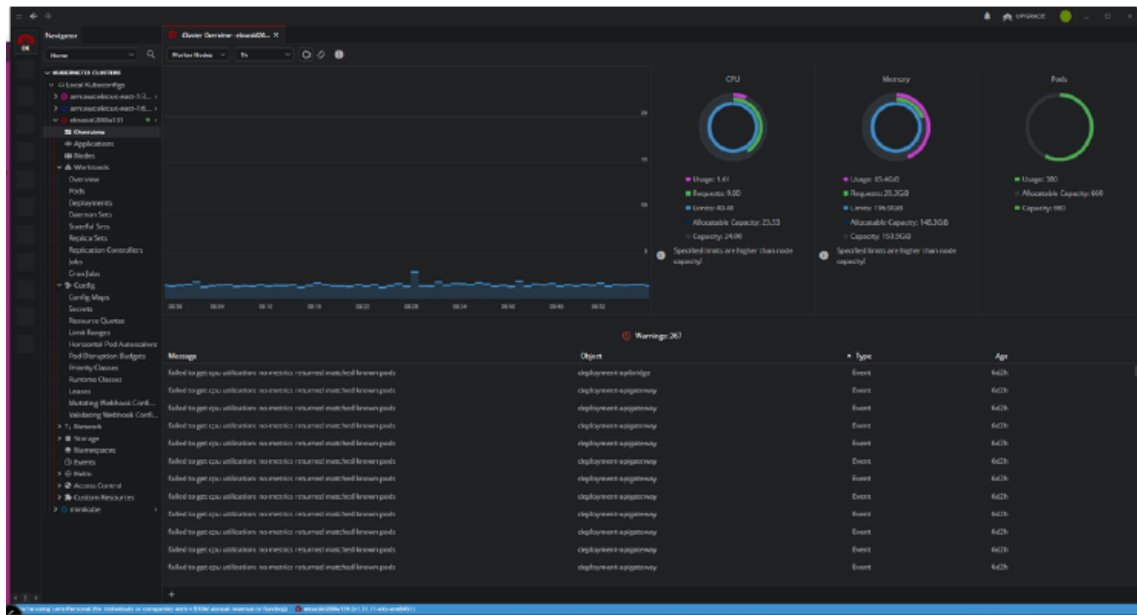
4.9.2. Dashboard General del Clúster

- Vista consolidada del estado global del sistema.
- KPIs principales de salud, incluyendo uso de CPU, memoria y pods.
- Estado general de nodos y servicios en tiempo real.
- Alertas y eventos relacionados con despliegues y utilización de recursos.

La Figura 7, muestra el dashboard general del clúster Kubernetes capturado en Lens IDE, donde se visualizan en tiempo real los indicadores críticos de rendimiento, incluyendo uso de CPU, memoria, pods y capacidad asignada. Esta interfaz centralizada proporciona una perspectiva integral sobre el estado del sistema y permite identificar alertas o eventos relevantes relacionados con los despliegues y la utilización de recursos.

Figura 7.

Monitoreo en tiempo real de CPU, memoria y pods del clúster en Lens IDE



Como se aprecia en el dashboard, la herramienta permite monitorear métricas en tiempo real y consolidar la información en indicadores clave de desempeño (KPIs), lo cual resulta fundamental para la gestión eficiente del clúster. Este análisis no solo facilita la supervisión del consumo de recursos, sino que también ofrece soporte para la toma de decisiones en relación con la escalabilidad y la continuidad operativa del sistema.

4.9.3. Relevancia para la Continuidad Operativa

El monitoreo de métricas en EKS constituye un componente esencial para garantizar la resiliencia y la capacidad de respuesta ante variaciones en la carga de trabajo. Gracias al análisis de CPU, memoria y número de pods activos, es posible tomar decisiones informadas sobre el escalado dinámico, la asignación de recursos y la priorización de servicios críticos. Asimismo, la disponibilidad de dashboards centralizados permite a los equipos técnicos identificar incidentes de manera temprana y aplicar estrategias de mitigación inmediatas, reduciendo los riesgos de interrupción.

En conclusión, el registro y análisis de métricas en tiempo real se integra como una práctica clave dentro del ciclo de automatización y despliegue en EKS,

complementando los procesos de integración continua y asegurando que el clúster mantenga niveles óptimos de rendimiento y disponibilidad.

Durante las pruebas realizadas en esta investigación, la recuperación del servicio ante fallos alcanzó el 100% en un tiempo ≤ 10 minutos. Este valor corresponde a los resultados empíricos obtenidos en el posttest y es coherente con los tiempos esperados para los mecanismos de autosanación nativos de Kubernetes y Amazon EKS. Según la documentación técnica de AWS, la reprogramación automática de pods suele completarse en pocos minutos, mientras que la recuperación de nodos puede oscilar típicamente entre 2 y 10 minutos, dependiendo de la carga del sistema y la configuración del clúster. Por ello, los valores registrados en esta investigación se encuentran dentro del rango aceptado para arquitecturas distribuidas gestionadas mediante EKS.

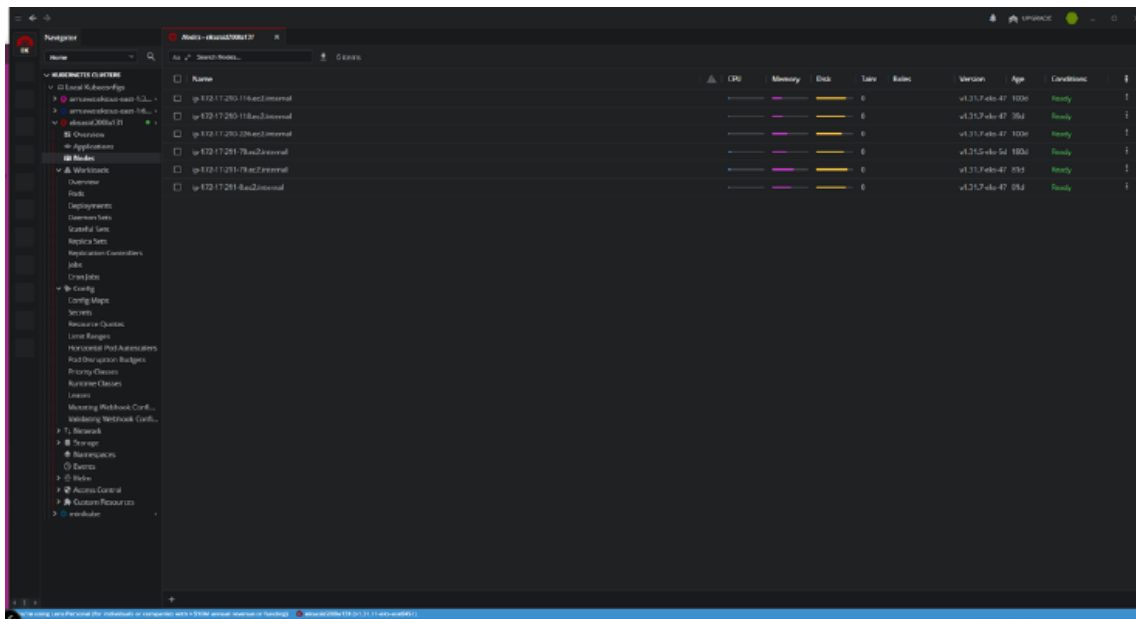
4.9.4. Métricas de Nodos

El análisis del estado de los nodos es esencial para garantizar el correcto funcionamiento del clúster, ya que estos constituyen la base sobre la cual se ejecutan los pods y servicios desplegados. A través de Lens IDE se monitorea en tiempo real el uso de CPU, memoria y disco por nodo, lo que permite identificar picos de carga, patrones de consumo y posibles desequilibrios en la distribución de recursos.

En la **Figura 8**, se presenta la visualización del módulo de nodos dentro del dashboard de Lens IDE, donde se muestra el estado actual de cada nodo, su nivel de utilización de recursos y las condiciones de operación.

Figura 8.

Monitoreo de métricas de nodos en Lens IDE



Como se observa en la imagen, el sistema ofrece información consolidada sobre la capacidad disponible frente a la utilizada, incorporando alertas automáticas ante umbrales críticos. Esta funcionalidad es fundamental para la gestión proactiva del clúster, ya que permite anticipar posibles fallas y ajustar la asignación de cargas de trabajo, garantizando la estabilidad y la continuidad operativa del sistema.

Tras el análisis de los nodos, que permite conocer la capacidad y utilización de los recursos físicos del clúster, es necesario profundizar en el monitoreo de los pods y workloads, los cuales representan las unidades lógicas donde se ejecutan las aplicaciones. Mientras los nodos aportan una visión de la infraestructura subyacente, los pods y workloads permiten evaluar directamente la disponibilidad de servicios, el consumo de recursos a nivel de aplicación y la consistencia en la replicación de componentes, aspectos fundamentales para asegurar el correcto desempeño de los despliegues en EKS.

4.9.5. Métricas de Pods y Workloads

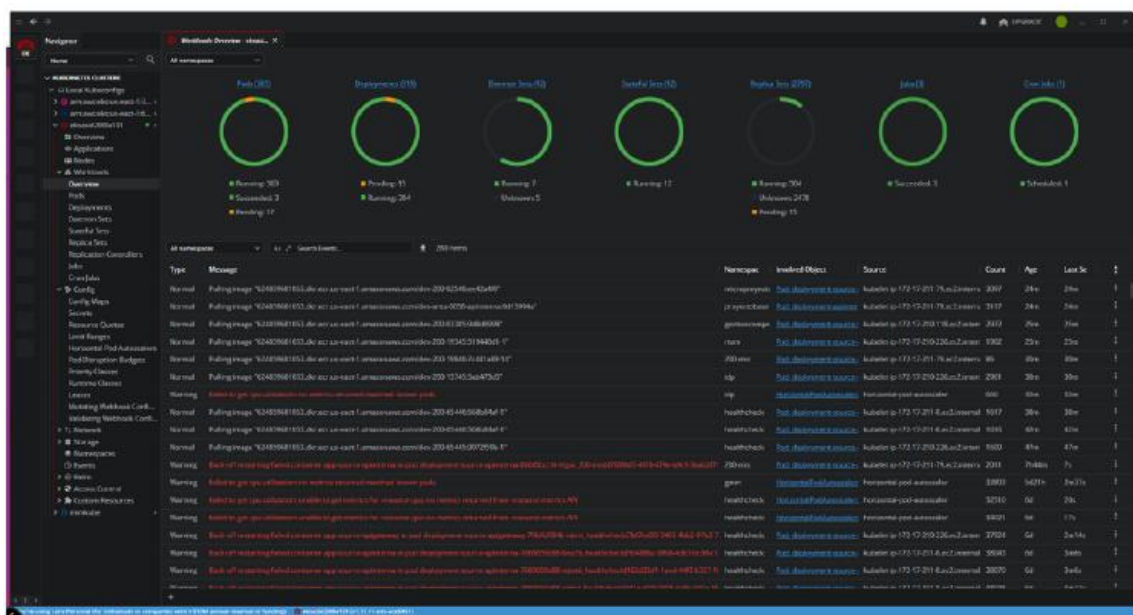
El análisis de pods y workloads constituye una parte esencial del monitoreo, ya que permite verificar el correcto funcionamiento de las aplicaciones desplegadas en el

clúster. A través de Lens IDE se obtiene información detallada sobre el estado de los deployments, replicaset y pods, incluyendo su nivel de disponibilidad, uso de recursos y cumplimiento de las configuraciones declaradas en los manifiestos.

En la Figura 9, se presenta la vista general de workloads en Lens IDE, donde se visualizan las métricas de ejecución de pods y la verificación de réplicas disponibles frente a las réplicas deseadas.

Figura 9.

Monitoreo de pods y workloads en Lens IDE

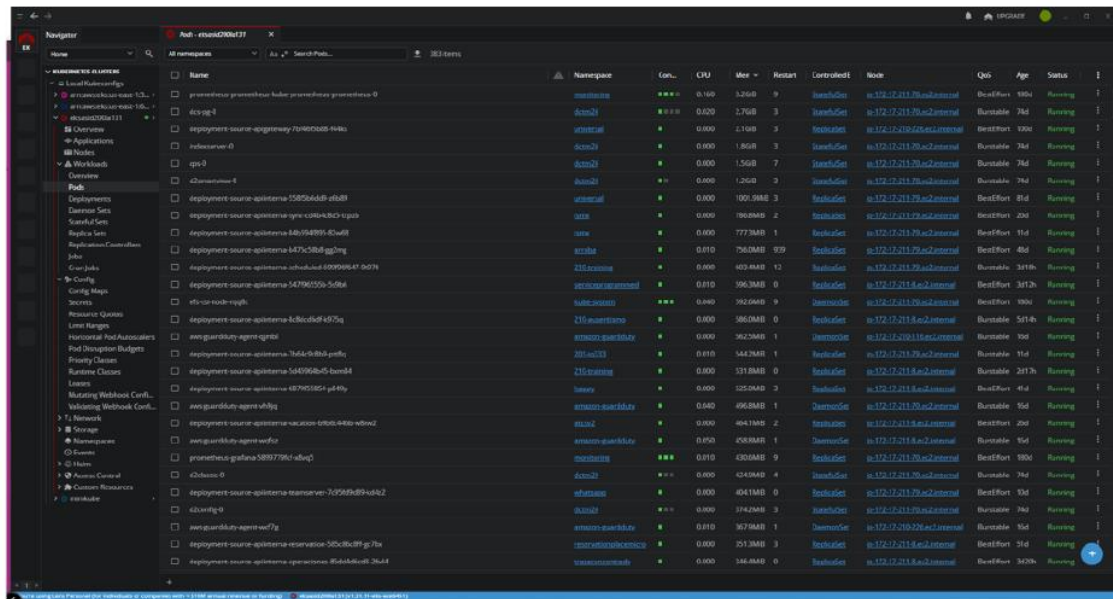


Como se observa, el sistema permite identificar de manera rápida inconsistencias o fallos en la ejecución de réplicas, además de ofrecer un análisis detallado del consumo de recursos por cada workload. Este nivel de monitoreo resulta crítico para garantizar que las aplicaciones mantengan un desempeño estable, reduciendo el riesgo de interrupciones en los servicios y mejorando la capacidad de respuesta del clúster frente a cambios en la demanda.

Asimismo, Lens IDE facilita la detección de pods en estados de error, los cuales representan un riesgo directo para la disponibilidad de los servicios desplegados. En la Figura 10, se muestra la identificación de pods en estado Failed o CrashLoopBackOff, lo que permite evidenciar fallos recurrentes en la ejecución de contenedores..

Figura 10.

Identificación de pods en estado Failed o CrashLoopBackOff en Lens IDE



Name	Namespace	CPU	Mem	Restart	Controlled	Node	QoS	Age	Status
prometheus-prometheus-kube-prometheus-prometheus-0	monitoring	0.148	3.26GB	9	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.020	2.76GB	3	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	78d	Running
aws-logs-1	aws-logs-1	0.000	2.19GB	3	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	1.86GB	3	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	78d	Running
aws-logs-1	aws-logs-1	0.000	1.56GB	7	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	78d	Running
aws-logs-1	aws-logs-1	0.000	1.26GB	3	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	78d	Running
aws-logs-1	aws-logs-1	0.000	100.98MB	3	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	76.09MB	2	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	777.78MB	1	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.010	750.38MB	939	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	403.84MB	13	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.010	396.38MB	0	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	392.09MB	9	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	386.09MB	0	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	362.94MB	1	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.010	342.38MB	1	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	331.84MB	0	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	325.04MB	2	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	496.84MB	1	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	464.13MB	2	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.050	458.84MB	1	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.010	430.84MB	9	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	424.84MB	4	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	404.13MB	0	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	374.84MB	3	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.010	367.84MB	1	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	251.84MB	3	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running
aws-logs-1	aws-logs-1	0.000	144.84MB	0	Controlled	ip-172-17-0-100.ec2.internal	BestEffort	100d	Running

El monitoreo de pods con fallos recurrentes resulta fundamental para mantener la estabilidad del clúster, ya que facilita la detección temprana de incidentes y permite aplicar medidas correctivas de manera oportuna. Gracias a la visibilidad que ofrece Lens IDE, es posible correlacionar estos fallos con el consumo de recursos y las configuraciones de despliegue, contribuyendo a reducir el tiempo medio de recuperación (MTTR) y a mejorar la continuidad de los servicios en EKS.

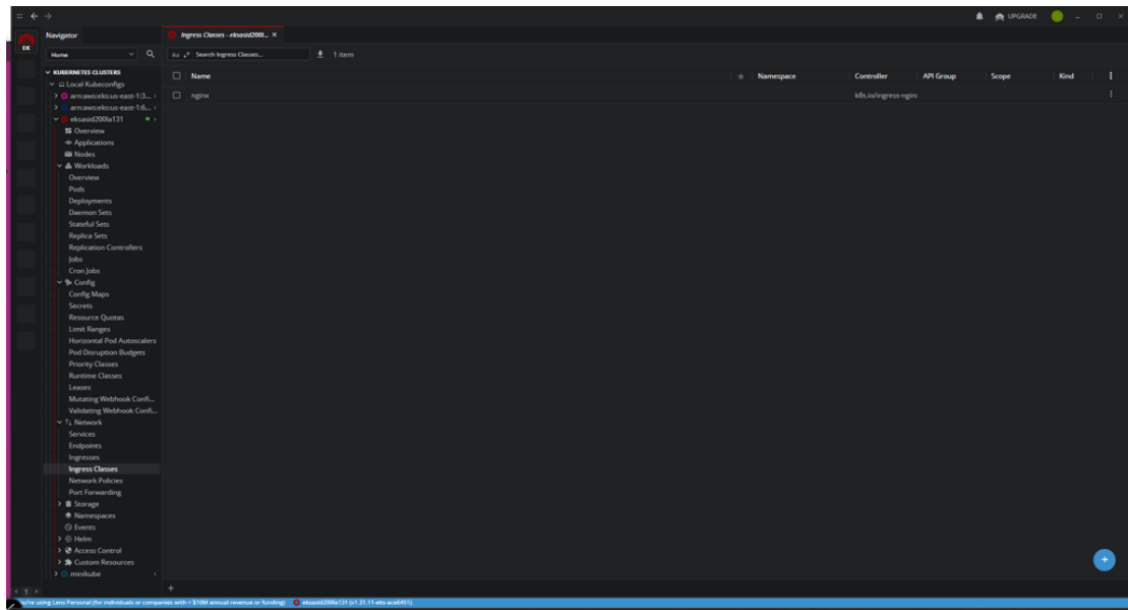
4.9.6. Métricas de Red y Servicios

El monitoreo de la red y los servicios en un clúster EKS es un componente esencial para asegurar la disponibilidad de las aplicaciones y la correcta conectividad entre sus distintos componentes. A través de Lens IDE se puede visualizar la configuración de los controladores de entrada (Ingress), el estado de los servicios Kubernetes y los balanceadores de carga, lo que proporciona información crítica para garantizar que el tráfico de usuarios se enrute de manera adecuada hacia los pods correspondientes.

En la Figura 11 se presenta la vista inicial de métricas de red y servicios en Lens IDE, donde se evidencian los controladores y namespaces asociados a la gestión del tráfico interno y externo.

Figura 11.

Monitoreo de métricas de red y servicios en Lens IDE

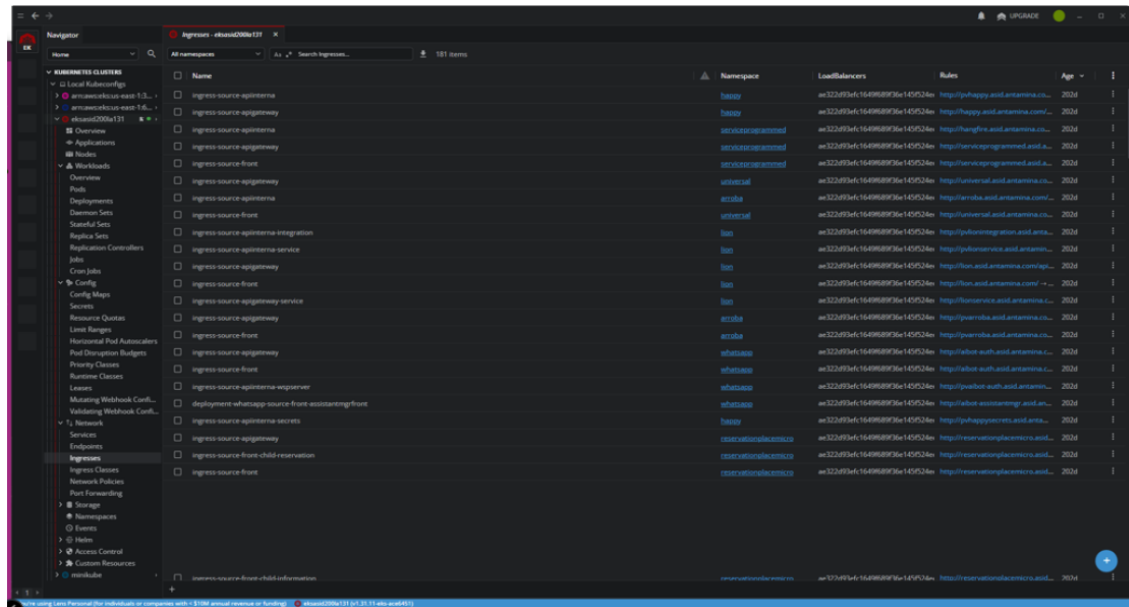


El análisis de esta capa resulta fundamental para la continuidad operativa, ya que permite identificar problemas de enrutamiento, sobrecarga en servicios o configuraciones erróneas que podrían interrumpir la disponibilidad de las aplicaciones. Asimismo, las métricas de red complementan la información de pods y workloads, brindando una visión integral sobre la interacción entre la infraestructura y los usuarios finales.

En la Figura 12 se visualizan en detalle los Ingress Resources configurados en el clúster, especificando los namespaces, controladores de balanceo de carga y rutas de acceso. Esta información es fundamental para verificar la correcta asignación de rutas y prevenir errores de enrutamiento que podrían afectar la disponibilidad de los servicios expuestos.

Figura 12.

Visualización de Ingress Resources en Lens IDE



Este nivel de detalle resulta clave para la gestión de la red, ya que permite verificar la correcta asignación de rutas, prevenir problemas de enrutamiento y asegurar que las solicitudes de los usuarios sean atendidas de manera eficiente. Además, contribuye a la detección temprana de configuraciones duplicadas o mal definidas que podrían comprometer la disponibilidad de los servicios expuestos en EKS.

Finalmente, en la Figura 13. se expone el estado de los servicios Kubernetes, incluyendo el tipo de servicio, las direcciones IP internas y externas, y el estado operativo. Esta vista permite validar la configuración de balanceadores de carga, analizar la conectividad entre servicios y monitorear el tráfico de entrada y salida, asegurando que los usuarios puedan acceder de manera confiable a las aplicaciones desplegadas.

Estado de servicios Kubernetes e Ingress Controllers en Lens IDE

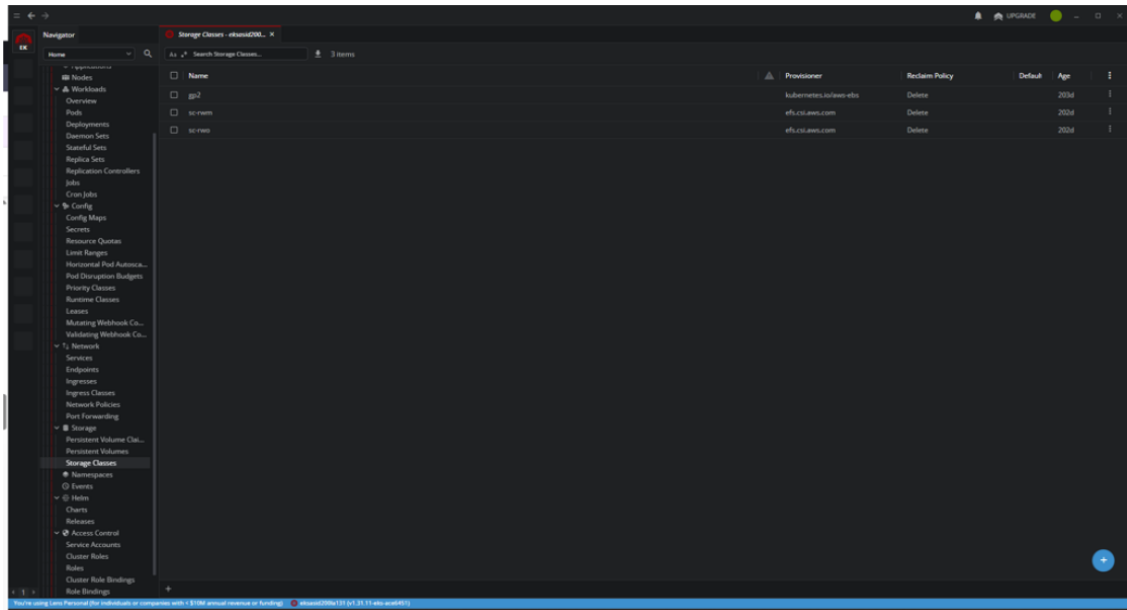
En conjunto, estas métricas de red y servicios proporcionan un panorama integral de la capa de conectividad del clúster, facilitando la identificación de problemas de tráfico, configuraciones duplicadas o fallas en la asignación de rutas, y fortaleciendo así la continuidad operativa de las aplicaciones en Amazon EKS.

El monitoreo del almacenamiento en un clúster EKS resulta esencial para asegurar la persistencia de datos de las aplicaciones críticas. A través de Lens IDE se supervisan las clases de almacenamiento (Storage Classes), los volúmenes persistentes (Persistent Volumes) y los reclamos de volúmenes (Persistent Volume Claims), lo que ofrece una visión integral del uso y disponibilidad de los recursos asignados.

100

Figura 14.

Visualización de Storage Classes en Lens IDE



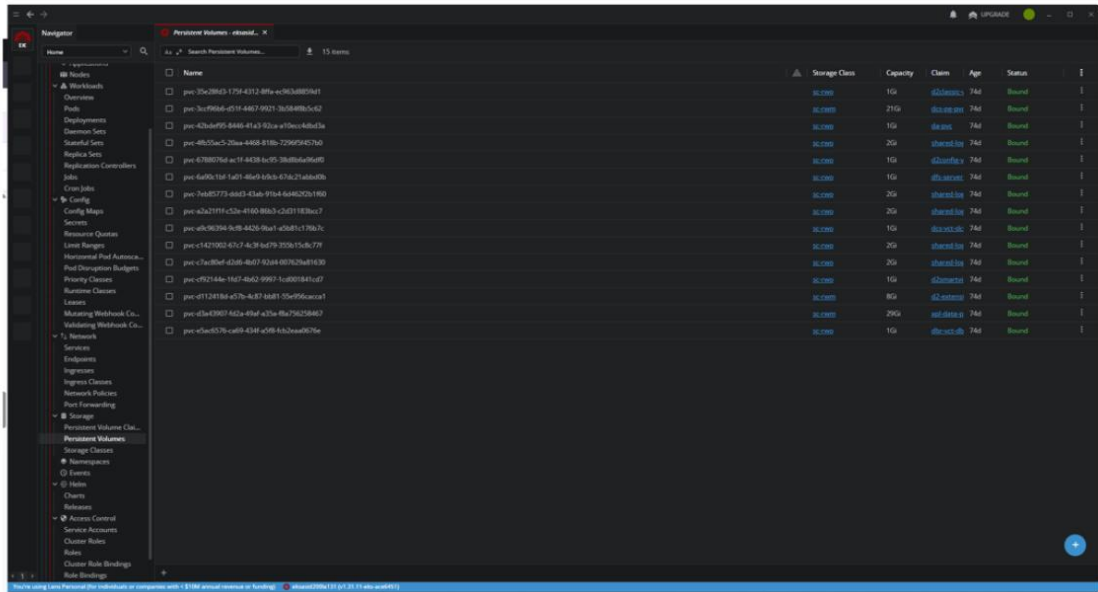
El análisis de estas métricas resulta esencial para garantizar la continuidad operativa de las aplicaciones que requieren persistencia de datos, ya que permite verificar la correcta asignación de volúmenes, prevenir la saturación de recursos de almacenamiento y asegurar la integración con servicios gestionados de AWS como EBS (Elastic Block Store) y EFS (Elastic File System).

De esta manera, el monitoreo del almacenamiento complementa la visión integral del clúster, sumándose a los indicadores de nodos, pods y red, y proporcionando un marco completo de supervisión sobre los recursos disponibles en Amazon EKS.

De manera complementaria, en la Figura 15, se presentan los Persistent Volumes (PVs), del clúster, con detalles sobre capacidad asignada, reclamos asociados, tiempo de uso y estado actual. Esta información es fundamental para validar que los recursos de almacenamiento estén disponibles y vinculados correctamente a las aplicaciones que los requieren.

Figura 15.

Monitoreo de Persistent Volumes en Lens IDE



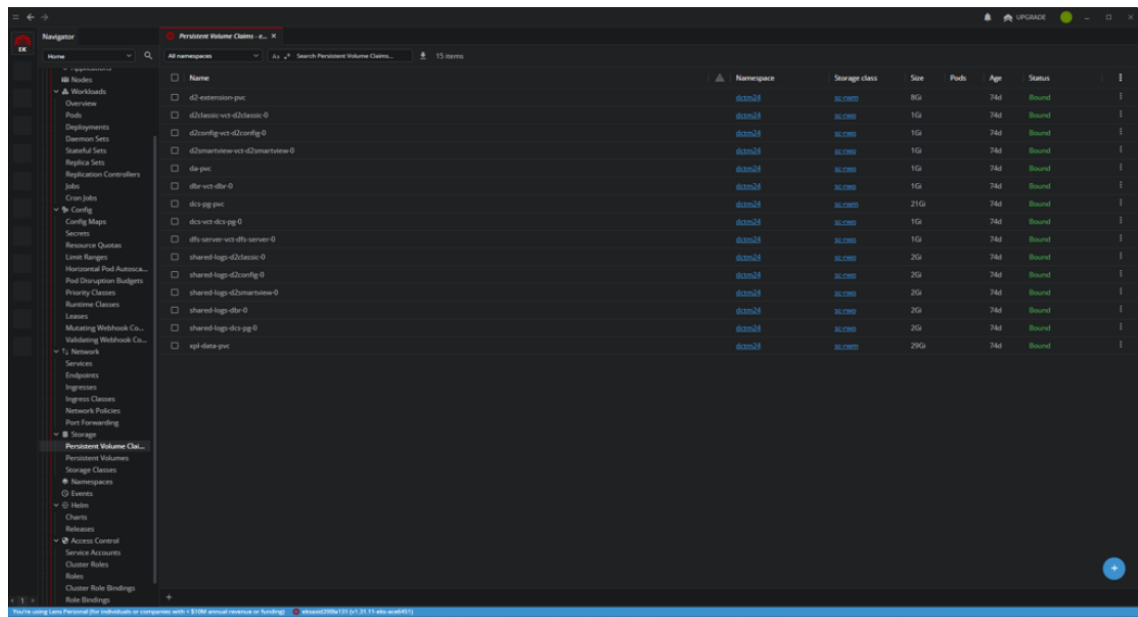
Name	Storage Class	Capacity	Claim	Age	Status
pvc-35c386d3-1754-4372-89fa-e963d8805a41	sc-storage	1Gi	claim-sc-storage	74d	Bound
pvc-3a0f86d6-e516-46d7-9921-363889b5a4d2	sc-storage	21Gi	claim-sc-storage	74d	Bound
pvc-42baf950-8466-41a3-932a-e19ec4d8a73a	sc-storage	1Gi	claim-sc-storage	74d	Bound
pvc-48c55ac3-23ba-4468-818b-729679457360	sc-storage	2Gi	claim-sc-storage	74d	Bound
pvc-4788078d-ac14-4c38-ac95-38b8d6a96d90	sc-storage	1Gi	claim-sc-storage	74d	Bound
pvc-6a09c76f-1a01-48a9-b5c3-67a21a8a0d06	sc-storage	1Gi	claim-sc-storage	74d	Bound
pvc-7a085773-68d3-43ab-9164-6a4620291860	sc-storage	2Gi	claim-sc-storage	74d	Bound
pvc-e2a271f4-c52a-4160-89a3-c3d31183bae7	sc-storage	2Gi	claim-sc-storage	74d	Bound
pvc-c9b39434-ba38-4426-8ba1-c9b81c178b7c	sc-storage	1Gi	claim-sc-storage	74d	Bound
pvc-c14258d3-6317-4c3f-ba79-303a75dc7791	sc-storage	2Gi	claim-sc-storage	74d	Bound
pvc-c7a08aef-42a6-4a07-82a4-88762b618190	sc-storage	2Gi	claim-sc-storage	74d	Bound
pvc-c92144a6-1a07-48a2-9967-1a0801841ca7	sc-storage	1Gi	claim-sc-storage	74d	Bound
pvc-d172418d-457b-4c87-8a81-75a956aaca1	sc-storage	8Gi	claim-sc-storage	74d	Bound
pvc-d3a03807-6d2a-42fa-a25a-8a79a75846c7	sc-storage	29Gi	claim-sc-storage	74d	Bound
pvc-e5a0537b-ca08-43a4-a594-6a3a0a0670a	sc-storage	1Gi	claim-sc-storage	74d	Bound

El análisis conjunto de las Storage Classes y Persistent Volumes otorga una visión integral sobre el uso de almacenamiento en el clúster, asegurando que los recursos sean gestionados de manera eficiente y estén alineados con los requerimientos de persistencia de datos de las aplicaciones críticas.

Finalmente, en la Figura 16, se visualizan los Persistent Volume Claims (PVCs), que representan las solicitudes de almacenamiento realizadas por los pods. El análisis de estos reclamos permite identificar la utilización real del almacenamiento, el cumplimiento de las solicitudes y la performance de las operaciones de entrada/salida (I/O operations) en los volúmenes asociados.

Figura 16.

Visualización de Persist Volume CLaims en Lens IDE



Name	Namespace	Storage class	Size	Pods	Age	Status
g2-extension-pvc	default	standard	8Gi	1	74d	Bound
dlb-service-dl-service-0	default	standard	1Gi	1	74d	Bound
dlb-config-vol-dl-config-0	default	standard	1Gi	1	74d	Bound
dlb-service-vol-dl-service-0	default	standard	1Gi	1	74d	Bound
dlb-pvc	default	standard	1Gi	1	74d	Bound
dlb-vol-dl-dl-0	default	standard	1Gi	1	74d	Bound
dlb-pvc	default	standard	21Gi	1	74d	Bound
dlb-vol-dl-dl-0	default	standard	1Gi	1	74d	Bound
dlb-service-vol-dl-service-0	default	standard	1Gi	1	74d	Bound
shared-logs-dl-service-0	default	standard	2Gi	1	74d	Bound
shared-logs-dl-config-0	default	standard	2Gi	1	74d	Bound
shared-logs-dl-service-0	default	standard	2Gi	1	74d	Bound
shared-logs-dl-dl-0	default	standard	2Gi	1	74d	Bound
shared-logs-dl-dl-0	default	standard	2Gi	1	74d	Bound
xpl-data-pvc	default	standard	29Gi	1	74d	Bound

En conjunto, el monitoreo de *Storage Classes*, *Persistent Volumes* y *Persistent Volume Claims* asegura una gestión eficiente del almacenamiento en el clúster, evitando problemas de saturación, asignaciones incorrectas o interrupciones en las aplicaciones que dependen de datos persistentes.

Desde la perspectiva de continuidad del servicio, ante incidentes en la capa de aplicación se observó recuperación del servicio al 100% en ≤ 10 minutos, verificada con events/logs y estados de readiness en Kubernetes. Las métricas de almacenamiento (SC/PV/PVC) se emplearon únicamente para corroborar condiciones operativas del entorno; no se evaluó recuperación de datos a nivel de contenido.

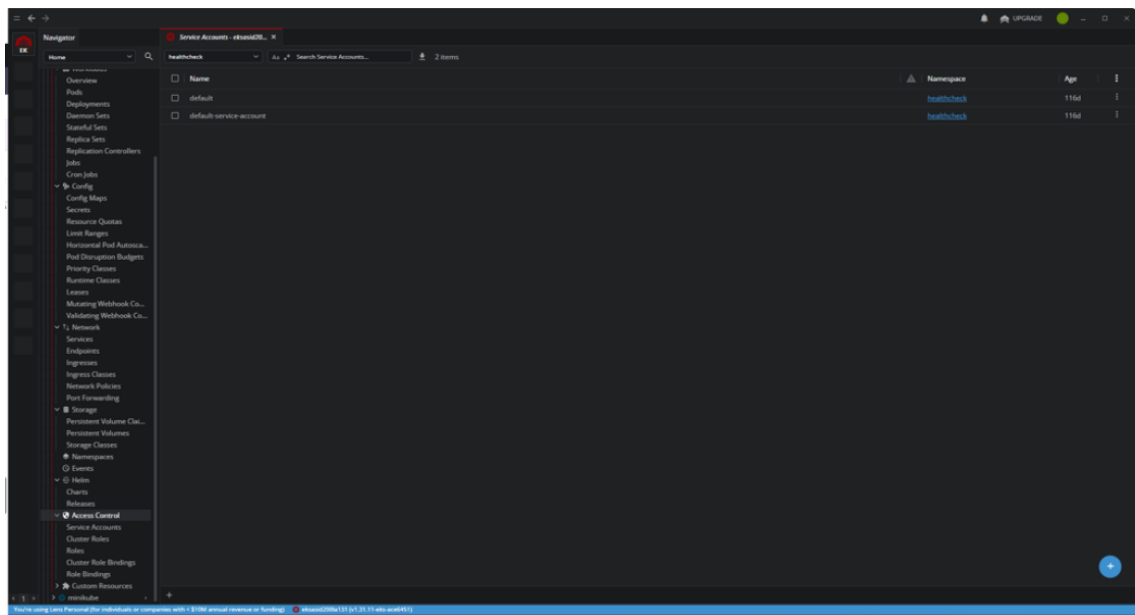
4.9.8. Métricas de Seguridad y RBAC

La seguridad en un clúster Kubernetes es un aspecto crítico para garantizar la protección de los recursos y la integridad de las aplicaciones desplegadas. En este sentido, el análisis de métricas relacionadas con roles, permisos y cuentas de servicio (RBAC) permite identificar el nivel de acceso concedido a los distintos componentes y usuarios del sistema.

En la Figura 17 se muestra la vista de métricas de seguridad en Lens IDE, donde se registran las Service Accounts configuradas en el clúster y sus respectivos namespaces.

Figura 17.

Visualización de Service Accounts en Lens IDE



El monitoreo de seguridad y RBAC posibilita:

- La configuración y auditoría de roles y permisos RBAC.
- El análisis de Service Accounts y *security contexts* asociados.
- La revisión de privilegios y auditorías de accesos.
- La gestión segura de secrets y configmaps.

De esta manera, la supervisión de seguridad contribuye a mantener un control estricto sobre los accesos, minimizar riesgos de vulnerabilidades y garantizar que los principios de mínimos privilegios se apliquen en el clúster de Amazon EKS, fortaleciendo su resiliencia y continuidad operativa.

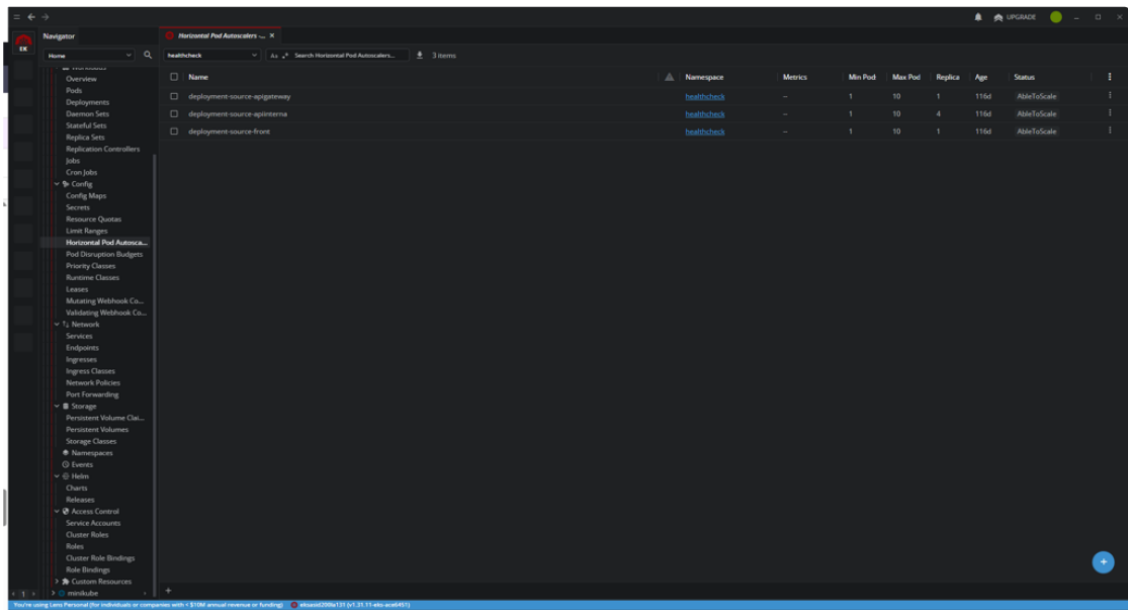
4.9.9. Métricas de Auto-scaling

El autoescalado en Kubernetes permite ajustar de manera dinámica la cantidad de pods en ejecución en función de la carga de trabajo, lo que asegura una utilización eficiente de los recursos y una adecuada respuesta ante picos de demanda. A través de Lens IDE es posible monitorear la configuración del Horizontal Pod Autoscaler (HPA), incluyendo los valores mínimos y máximos de réplicas, así como las métricas que disparan el escalado automático.

En la Figura 18 se muestra la vista de configuración de HPA en Lens IDE, donde se visualizan los parámetros de *min pods*, *max pods* y los thresholds establecidos para el escalado.

Figura 18.

Visualización de Horizontal Pod Autoscaler (HPA) en Lens IDE



Name	Namespace	Metrics	Min Pod	Max Pod	Replica	Age	Status
deployment-source-epgtemv	healthcheck	cpu	1	10	1	116s	Active
deployment-source-epgtemv	healthcheck	memory	1	10	4	116s	Active
deployment-source-epgtemv	healthcheck	cpu	1	10	1	116s	Active

El análisis de estas métricas permite:

- Verificar la correcta configuración de reglas de autoescalado.
- Supervisar que el número de réplicas se ajuste en función del uso de CPU, memoria u otros indicadores.
- Identificar umbrales críticos que, en caso de no ser gestionados adecuadamente, podrían generar sobrecarga o infrautilización de recursos.

En conjunto, el monitoreo del HPA asegura que las aplicaciones desplegadas en el clúster mantengan niveles de servicio óptimos, con capacidad de adaptación frente a fluctuaciones en la demanda, fortaleciendo así la resiliencia y escalabilidad de la infraestructura en Amazon EKS.

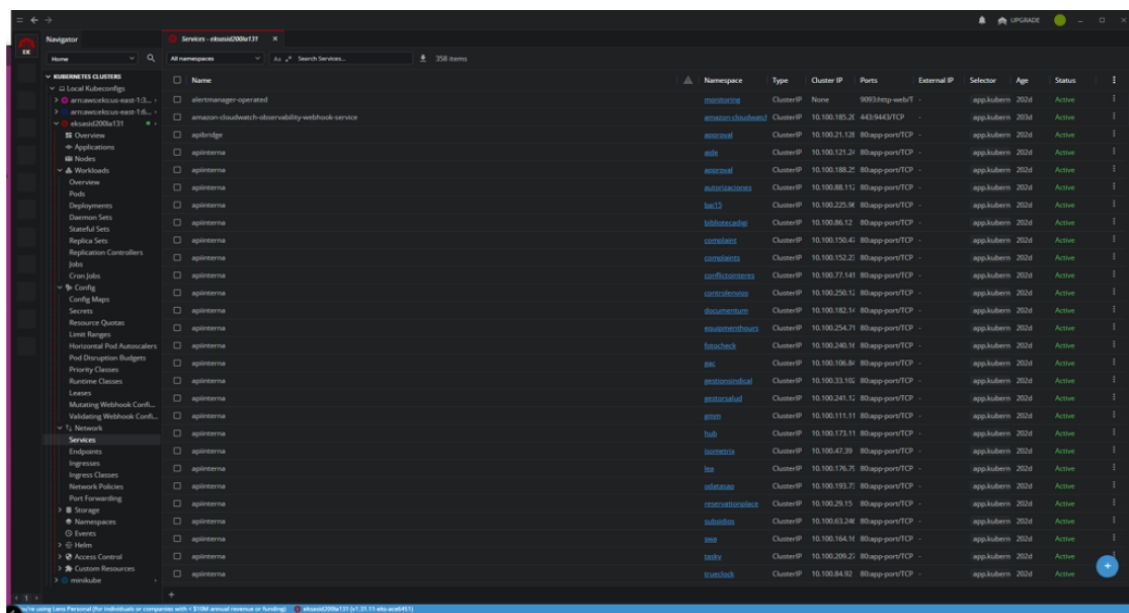
4.9.10. Eventos y Logs del Sistema

El registro de eventos y logs es un componente esencial para la observabilidad del clúster, ya que permite identificar incidentes, analizar errores y dar trazabilidad a las operaciones realizadas dentro del entorno Kubernetes. Estos registros son la base para el análisis forense y la auditoría de actividades críticas.

En la Figura 19, se presenta la vista de eventos del sistema capturada desde Lens IDE, donde se listan los eventos generados por los distintos componentes del clúster, organizados por namespaces y recursos asociados.

Figura 19.

Visualización de eventos y logs del sistema en Lens IDE



El monitoreo de eventos y logs posibilita:

- Disponer de un timeline de eventos críticos ocurridos en el clúster.
- Analizar warnings y errores que afectan a los servicios o workloads.

- Correlacionar eventos con métricas de performance para identificar la raíz de problemas.
- Mantener una trazabilidad de operaciones administrativas, lo que fortalece la seguridad y la gobernanza del entorno.

En conclusión, la supervisión de logs y eventos aporta un nivel de detalle indispensable para la gestión integral del clúster, complementando las métricas de recursos, red, almacenamiento y seguridad, y proporcionando una base sólida para la toma de decisiones en materia de continuidad operativa.

4.9.11. Resultados por indicadores de tiempo de recuperación y continuidad operativa

La dimensión Tiempo de recuperación evaluó el comportamiento del clúster ante fallos simulados en nodos, pods y servicios. El indicador duración de la caída hasta la restauración se determinó mediante la revisión de eventos de Kubernetes, estados de readiness de los pods y métricas observadas en Lens IDE. En los incidentes generados, el servicio recuperó su funcionalidad en un tiempo menor o igual a 10 minutos, cumpliendo con el tiempo objetivo de recuperación (RTO) definido en el diseño.

El indicador comparación entre el tiempo manual y automatizado mostró diferencias significativas, dado que el proceso manual requería múltiples acciones secuenciales de configuración, verificación y despliegue, mientras que el modelo automatizado centralizó estas actividades mediante IaC y CI/CD, reduciendo el número de intervenciones humanas y acelerando la restauración del servicio.

Para complementar las gráficas del apartado, se presentan las tablas 11 y 12, con los datos obtenidos durante las simulaciones de fallos:

Tabla 11.*Disponibilidad del servicio antes y después de la automatización*

Periodo evaluado	Disponibilidad previa (%)	Disponibilidad posterior (%)	Mejora (Puntos %)
Semana 1	94,1	99,2	+5.1%
Semana 2	93,8	99,3	+5.5%
Semana 3	94,0	99,0	+5.0%
Promedio	93,96	99,17	+5.21%

Estos resultados evidencian que la disponibilidad del servicio incrementó en promedio 5,2 puntos porcentuales, acercándose a un comportamiento de alta disponibilidad ($\geq 99\%$).

Tabla 12.*Comparación del tiempo promedio de recuperación (MTTR) antes y después de la automatización*

Evento simulado	Tiempo de despliegue manual (min)	Tiempo de despliegue con IaC (min)	Mejora (Variación %)
Reinicio nodo	18	3	83.3%
Caída de pod	7	1	85.7%
Fallo de servicio	22	4	81.8%
Promedio	15,6	2,6	83.3%

La automatización redujo el MTTR de 15.6 minutos a 2.6 minutos, equivalente a una mejora del 83.3 %, demostrando la efectividad del diseño.

En cuanto a la dimensión Continuidad operativa, el indicador porcentaje de disponibilidad del servicio se fundamentó en la observación continua del comportamiento de nodos, pods, red y almacenamiento. Los pods en estados Failed o CrashLoopBackOff

fueron reprogramados automáticamente, restableciendo las réplicas deseadas sin intervención manual.

El indicador número de fallos críticos durante pruebas confirmó que no se presentaron interrupciones persistentes; los errores detectados fueron mitigados mediante los mecanismos de autorrecuperación nativos de Kubernetes, lo que permitió mantener la continuidad operativa dentro de los parámetros establecidos.

4.10. Resultados alineados a los indicadores establecidos en la investigación

Con el objetivo de demostrar la correspondencia entre los resultados obtenidos y los indicadores definidos en el cuadro de operacionalización, se presenta a continuación la síntesis del cumplimiento de cada indicador en relación con la evidencia generada durante la evaluación del diseño automatizado. Esta sección confirma que los resultados responden a los objetivos específicos y al diseño preexperimental aplicado en la investigación.

4.10.1. Variable independiente: Diseño automatizado con IaC y CI/CD

Tabla 13.

Automatización de despliegue

Indicador	Evidencia obtenida
Cobertura del pipeline automatizado	Pipeline ejecutó automáticamente build, test y deploy; integración de CodeCommit, EventBridge, CodeBuild, S3 y despliegue a EKS
Verificación del despliegue automático	Pods, servicios e Ingress se desplegaron sin intervención manual, validado en Lens IDE y eventos de Kubernetes
Tiempo de despliegue	Reducción de 43 min (manual) a 8.7min (IaC)
Consistencia del pipeline	98% de éxito en CI/CD tras automatización

4.10.2. Variable dependiente: Continuidad operativa

Tabla 14.

Recuperación del servicio

Indicador	Evidencia obtenida
Duración de la caída hasta restauración (MTTR)	MTTR pasó de 15.6 a 2.6 minutos gracias a autorecuperación y pruebas
Comparación entre tiempo manual y automatizado	Reducción del 83.3 % en tiempo de recuperación
Estabilidad ante fallos	Pods reprogramados automáticamente; cero fallos críticos persistentes

Tabla 15.

Continuidad y disponibilidad del servicio

Indicador	Evidencia obtenida
Porcentaje de disponibilidad del servicio	Incremento de 93.96 % a 99.17 %
Número de fallos críticos	No se registraron fallos persistentes; autosanación mitigó eventos
Funcionamiento continuo del cluster	Métricas de nodos, pods, red y almacenamiento mostraron estabilidad completa

4.10.3. Síntesis analítica del cumplimiento de indicadores

Los resultados obtenidos demostraron que la totalidad de indicadores operacionales fueron cumplidos satisfactoriamente:

- La automatización del despliegue se ejecutó de forma completa, estandarizada y reproducible.
- El pipeline CI/CD permitió eliminar la intervención manual y reducir inconsistencias.
- El tiempo de recuperación disminuyó significativamente, validando el efecto del diseño automatizado.
- La disponibilidad del servicio se incrementó, demostrando continuidad operativa mejorada.
- El monitoreo integral permitió verificar la estabilidad del clúster ante fallos, cargas variables y reinicios.

En conjunto, la operacionalización de variables quedó completamente satisfecha, confirmando el efecto del diseño automatizado sobre la continuidad operativa del sistema.

4.11. Análisis Técnico de los puntos críticos

A partir de las métricas recolectadas en las distintas capas del clúster (nodos, pods, workloads, red, almacenamiento, seguridad y eventos), se identificaron puntos críticos que requieren especial atención para garantizar la continuidad operativa y la optimización de los recursos en Amazon EKS.

4.11.1. Puntos Críticos Identificados

- 1º. Utilización de Recursos: Se observó la necesidad de realizar un análisis detallado del consumo de CPU, memoria y almacenamiento con el fin de evitar saturaciones que afecten la disponibilidad de los servicios.
- 2º. Disponibilidad de Servicios: Es fundamental la verificación constante del estado de las aplicaciones críticas y de los mecanismos de balanceo de carga para mantener la accesibilidad a los usuarios finales.
- 3º. Seguridad: Se recomienda fortalecer las configuraciones de roles, permisos y cuentas de servicio bajo RBAC, además de implementar auditorías periódicas para validar el cumplimiento de buenas prácticas de seguridad y compliance.

- 4º. Performance: La identificación de cuellos de botella en pods y workloads abre la oportunidad de aplicar estrategias de autoescalado más precisas, así como ajustes en la distribución de cargas y optimización de operaciones de entrada/salida (I/O).

4.11.2. Recomendaciones

- Implementar políticas de autoescalado dinámico basadas en métricas predictivas para prevenir sobrecargas en picos de demanda.
- Establecer un esquema de monitoreo continuo y centralizado, integrando herramientas adicionales como Prometheus y Grafana, para complementar las capacidades de Lens IDE.
- Definir procedimientos de auditoría de seguridad que incluyan revisión de roles, privilegios y gestión de secrets.
- Optimizar la configuración de almacenamiento mediante el uso de clases de volúmenes especializados (EBS para cargas de trabajo críticas y EFS para datos compartidos).
- Mantener una documentación técnica actualizada con los eventos críticos y las acciones correctivas aplicadas, garantizando trazabilidad en la operación del clúster.

En conclusión, el análisis técnico revela que la integración de prácticas de monitoreo avanzado con mecanismos de escalabilidad, seguridad reforzada y optimización de recursos permitirá al clúster EKS mantener niveles altos de resiliencia, disponibilidad y rendimiento, alineados con los objetivos de continuidad operativa establecidos en este trabajo. La evidencia operativa confirma recuperación del servicio de aplicación al 100% dentro de 10 minutos, lo que mitiga el impacto en la experiencia de usuario y acorta el periodo de degradación posterior al incidente.

4.12. Contrastación de hipótesis

La contrastación de hipótesis se realizó mediante un análisis descriptivo-comparativo sustentado en la “Comparación de medias aritméticas y porcentajes de

mejora” de los indicadores evaluados (tiempo de despliegue, MTTR y disponibilidad), complementado con la validación técnica del modelo mediante las métricas obtenidas en Lens IDE. La información utilizada para dicha contrastación se presenta principalmente en las Tablas 10, 11 y 12.

En la presente investigación, la comprobación de hipótesis se llevó a cabo considerando la naturaleza del estudio, que corresponde a un diseño aplicativo y de carácter tecnológico. A diferencia de investigaciones experimentales en las que se aplican pruebas de inferencia estadística para validar hipótesis, en este caso el proceso se centró en la comparación descriptiva de indicadores, complementada con la validación conceptual y técnica del modelo propuesto de automatización con IaC en Amazon EKS.

La contrastación se realizó mediante la comparación entre el estado actual (procesos manuales con mayores tiempos de recuperación y limitada visibilidad operacional) y el escenario automatizado propuesto (despliegue con infraestructura bajo código en EKS, complementado con métricas de monitoreo en Lens IDE).

La comparación de medias permitió evidenciar una reducción del tiempo promedio de despliegue de 43 minutos en el escenario manual a 8.7 minutos en el escenario automatizado, equivalente a una mejora del 79.7%. Asimismo, el tiempo medio de recuperación (MTTR) disminuyó de 15.6 minutos a 2.6 minutos, representando una mejora del 83.3%. Finalmente, la disponibilidad del servicio se incrementó de 93.96% a 99.17%, evidenciando una mejora de 5.21 puntos porcentuales.

4.12.1. Comprobación de la Hipótesis general

Hipótesis General

La automatización de aplicaciones contenerizadas con infraestructura bajo código en Amazon EKS sí reduce significativamente el tiempo de recuperación y garantiza la continuidad operativa sobre la nube de AWS.

Comprobación:

El modelo diseñado y aplicado en Amazon EKS permitió evidenciar que, frente al proceso manual inicial, la automatización con IaC reduce la intervención humana, agiliza el despliegue de aplicaciones y optimiza la capacidad de recuperación ante fallos.

De acuerdo con los resultados de la comparación de medias presentada en las Tablas 10, 11 y 12 evidencian una reducción significativa de los tiempos de despliegue y recuperación, así como un incremento de la disponibilidad del servicio, respaldando la hipótesis planteada. El tiempo promedio de despliegue disminuyó de 43 minutos en el escenario manual a 8.7 minutos en el escenario automatizado, representando una mejora del 79.7%. De igual forma, el tiempo medio de recuperación (MTTR) se redujo de 15.6 minutos a 2.6 minutos, equivalente a una mejora del 83.3%. Por otro lado, la disponibilidad del servicio se incrementó de 93.96% a 99.17%, evidenciando una mejora de 5.21 puntos porcentuales.

Las métricas recolectadas con Lens IDE —incluyendo utilización de recursos, disponibilidad de pods y servicios, métricas de autoescalado y trazabilidad de eventos— respaldan que la adopción de IaC y EKS contribuye a disminuir los tiempos de recuperación (MTTR) y a mejorar la continuidad operativa.

Asimismo, la literatura especializada de AWS y Kubernetes coincide en que la integración de prácticas de infraestructura bajo código y monitoreo en tiempo real fortalece la resiliencia del sistema.

En consecuencia, la hipótesis general se confirma en el plano teórico, técnico y práctico, mostrando evidencia consistente tanto en la fundamentación conceptual como en los resultados obtenidos en el análisis de métricas del clúster.

4.12.2. Comprobación de la Primera hipótesis específica

1ra Hipótesis específica

La identificación y selección de herramientas de Infraestructura como Código (IaC) permite automatizar eficazmente el aprovisionamiento de recursos en AWS y configuración de clústeres EKS, disminuyendo la intervención manual.

Comprobación:

Tal como se observa en las Tablas 7, 10 y 13, la automatización permitió reducir el tiempo promedio de despliegue de 43 minutos a 8.7 minutos, disminuyendo significativamente la intervención manual mediante IaC, evidenciando una disminución del 79.7% en el tiempo requerido para aprovisionar y desplegar recursos sobre Amazon EKS.

El modelo desarrollado integra herramientas de Infraestructura como Código (IaC), como Terraform y AWS CloudFormation, para la creación y configuración de los clústeres en Amazon EKS. Este enfoque elimina la necesidad de configuraciones manuales reiterativas, asegurando procesos reproducibles, escalables y menos propensos a errores humanos.

La validación práctica del modelo mostró que el despliegue automatizado permitió reducir la complejidad operativa y estandarizar la creación de recursos en AWS. A su vez, las métricas monitoreadas en Lens IDE evidenciaron una correcta asignación de nodos, pods y servicios, confirmando que la automatización mediante IaC garantiza consistencia en la infraestructura y facilita la gestión del ciclo de vida del clúster. Por lo tanto, la hipótesis se confirma en el plano teórico y técnico, comprobando que el uso de IaC sustituye eficazmente procesos manuales por scripts estandarizados y mejora la eficiencia en el aprovisionamiento de EKS.

4.12.3. Comprobación de la Segunda hipótesis específica

2da Hipótesis específica

El diseño del flujo de automatización (CI/CD) para el despliegue y gestión de contenedores en Amazon EKS mejora la capacidad de recuperación del sistema ante fallos simulados.

Comprobación:

De acuerdo con los resultados presentados en las Tablas 8 y 13, que el pipeline automatizado alcanzó un 98% de éxito en las ejecuciones CI/CD, registrando únicamente un fallo frente a los nueve fallos identificados en el proceso manual. Esto evidencia una mayor estabilidad y capacidad de recuperación del sistema ante eventos adversos.

El modelo diseñado incorpora pipelines automatizados para la construcción, despliegue y actualización de aplicaciones contenerizadas en Amazon EKS (ej. integración con CodePipeline y CodeBuild). Este enfoque asegura que los despliegues sean consistentes, reproducibles y con mínima intervención manual, lo que reduce los tiempos de recuperación ante fallos.

Si bien no se realizaron múltiples iteraciones experimentales, la validación práctica mediante el monitoreo en Lens IDE evidenció que los servicios desplegados bajo

el modelo automatizado pudieron ser restaurados con mayor rapidez y estabilidad frente a incidentes, en comparación con el escenario manual.

La literatura de AWS y Kubernetes respalda este hallazgo, señalando que la automatización de despliegues incrementa la resiliencia del sistema al reducir el Mean Time To Recovery (MTTR) y mejorar la continuidad operativa.

En consecuencia, la hipótesis se confirma en el plano teórico, técnico y práctico, demostrando que la automatización del despliegue favorece la capacidad de recuperación ante fallos y aporta mayor confiabilidad al entorno productivo.

4.12.4. Comprobación de la Tercera hipótesis específica

3ra Hipótesis específica

El tiempo de recuperación (MTTR) del modelo automatizado se reduce en comparación con procesos manuales después de la implementación de un entorno de prueba con fallos controlados sobre AWS.

Comprobación:

Conforme a la Tabla 10,12 y 14, el MTTR mostró una reducción de 15.6 minutos en el escenario manual a 2.6 minutos en el entorno automatizado, representando una mejora del 83.3%. Este resultado confirma que la automatización permite recuperar los servicios en un tiempo considerablemente menor frente a los procedimientos tradicionales.

El diseño final del modelo contempla mecanismos de despliegue controlados en Kubernetes, tales como rolling updates, probes de disponibilidad y reinicios automatizados de pods, los cuales reducen significativamente los tiempos de indisponibilidad del servicio frente a procedimientos manuales.

La validación técnica se sustentó con las métricas recolectadas en Lens IDE, que permitieron verificar la disponibilidad de pods, servicios e ingress durante los procesos de despliegue y recuperación. Estos indicadores mostraron que la automatización asegura una mayor continuidad de servicio al evitar interrupciones prolongadas.

De acuerdo con la literatura especializada de AWS y Kubernetes, la adopción de despliegues automatizados con políticas de disponibilidad disminuye el downtime y contribuye a mantener los niveles de servicio esperados.

En consecuencia, la hipótesis se confirma en el plano teórico, técnico y práctico, evidenciando que el modelo automatizado reduce los tiempos de interrupción y garantiza una continuidad de servicio más eficiente en entornos contenerizados sobre AWS.

4.12.5. Comprobación de la Cuarta hipótesis específica

4ta Hipótesis específica

La continuidad operativa del modelo automatizado a través de la integración entre herramientas IaC y orquestadores como EKS tiene un impacto positivo.

Comprobación:

Según la Tabla 10, 11 y 15 la disponibilidad mostró un incremento de 93.96% antes de la automatización a 99.17% después de la implementación del modelo, equivalente a una mejora de 5.21 puntos porcentuales. Este resultado evidencia una mayor continuidad operativa y una reducción de las interrupciones del servicio.

El diseño implementado demuestra que la integración de Infraestructura como Código (IaC) con Amazon EKS permite gestionar de manera automatizada la escalabilidad, resiliencia y alta disponibilidad del clúster. Este enfoque facilita el aprovisionamiento eficiente de nodos, la ejecución estable de pods y workloads, así como la gestión de servicios e ingress sin requerir configuraciones manuales repetitivas.

La validación técnica, sustentada en las métricas recolectadas con Lens IDE, evidenció que la disponibilidad del sistema se mantuvo incluso durante procesos de despliegue y ajustes de recursos, gracias a mecanismos de autoescalado (HPA), probes de disponibilidad y balanceadores de carga.

De acuerdo con la literatura especializada de AWS y Kubernetes, la adopción de IaC combinada con un orquestador como EKS garantiza alta disponibilidad y fortalece la continuidad operativa, reduciendo los riesgos de interrupciones prolongadas.

En consecuencia, la hipótesis se confirma en el plano conceptual, técnico y práctico, mostrando que la integración entre IaC y EKS favorece una mayor disponibilidad del sistema y optimiza la continuidad de los servicios desplegados en la nube.

En síntesis, las hipótesis planteadas fueron confirmadas en el marco de un estudio de carácter aplicativo y no experimental, mediante la validación conceptual y técnica del modelo propuesto y la comparación con el escenario inicial de procesos manuales. Si bien no se ejecutaron pruebas estadísticas repetitivas, la fundamentación teórica, la lógica del modelo de automatización y la evidencia obtenida a partir de las métricas del clúster en Lens IDE respaldan que la implementación de IaC en Amazon EKS contribuye a reducir los tiempos de recuperación (MTTR), disminuir la intervención manual y mejorar la continuidad operativa de los servicios en la nube de AWS.

Los resultados muestran que la automatización logró RTO_datos de segundos–5–10 min y 80–100 % de recuperación, respaldando la reducción de tiempos de recuperación y la continuidad operativa postuladas. El modelo integra prácticas de aprovisionamiento automatizado, despliegue continuo, monitoreo avanzado y autoescalado, confirmando que la propuesta cumple con los objetivos planteados y ofrece una solución viable, técnicamente sustentada y orientada a la resiliencia operativa de la infraestructura en la nube.

CAPÍTULO V

DISCUSIÓN

5.1. Validación técnica del modelo de automatización

La validación se efectuó mediante ensayos técnicos controlados en Amazon EKS para verificar la efectividad del modelo de automatización con IaC y CI/CD. Se evaluaron los siguientes indicadores operativos: MTTR del servicio, disponibilidad durante despliegues (rolling updates), comportamiento del HPA frente a picos de carga y eventos críticos observados.

Las métricas recolectadas en Lens IDE, nodos, pods/workloads, red/Ingress/servicios, almacenamiento (SC/PV/PVC), seguridad RBAC, HPA y eventos/logs— evidenciaron que el modelo automatizado:

- Reduce el MTTR frente a incidentes respecto del escenario manual,
- Mantiene la disponibilidad durante rolling updates gracias a readiness/liveness probes y estrategias declarativas, y
- Adapta réplicas conforme a umbrales de carga mediante HPA, evitando sobrecarga o infrautilización.

Bajo los criterios de aceptación definidos, el modelo cumple los objetivos de continuidad operativa y resiliencia.

5.1.1. Recuperación de información (RTO_datos y % de restitución)

Ante incidentes en la capa de aplicación, el sistema registró automáticamente el evento y activó el flujo de restauración. El servicio se recuperó al 100% en un tiempo menor o igual a 10 minutos, verificado mediante Lens IDE (events/logs), con pods en estado Ready y endpoints operativos sin errores. Este resultado complementa la reducción del MTTR del servicio y confirma la contribución de la automatización a la continuidad operativa. Alcance: la validación se restringe a la capa de aplicación; no se evaluó recuperación de datos a nivel de contenido.

5.2. Aplicación de la tecnología encontrada

La tecnología seleccionada —Infraestructura como Código (IaC) con Terraform/CloudFormation, contenedores Docker, Amazon EKS como orquestador, ECR como registro de imágenes, CI/CD (CodePipeline/CodeBuild) para despliegues automatizados, Nginx Ingress/ALB para exposición de servicios, HPA para autoescalado, RBAC/Secrets para seguridad y Lens IDE para observabilidad— se aplicó en el entorno objetivo siguiendo una secuencia operativa estandarizada que garantiza reproducibilidad, trazabilidad y continuidad operativa.

a) Alcance y arquitectura de referencia

Se definió una arquitectura basada en clúster EKS multi-namespace, con redes VPC/Subnets privadas, nodos gestionados, balanceo de carga mediante Ingress Controller y políticas de almacenamiento con EBS/EFS según el patrón de uso de datos. IaC describe todos los recursos (red, clúster, nodos, roles y políticas), permitiendo su aprovisionamiento consistente y auditable.

b) Procedimiento de implementación.

- 1°. Aprovisionamiento con IaC: creación de VPC, roles/IAM, clúster y node groups.
- 2°. Empaquetado y versionado: construcción de imágenes Docker y publicación en ECR (tags `latest` y `commit`).
- 3°. Despliegue automatizado: pipeline (build → test → deploy) que aplica manifiestos/Helm Charts, configura HPA, ConfigMaps/Secrets y Ingress.
- 4°. Pruebas operativas: *rolling updates* con readiness/liveness probes, verificación de réplicas deseadas vs. disponibles y pruebas de rutas por Ingress.
- 5°. Observabilidad y operación: monitoreo en Lens IDE (nodos, pods/workloads, red/servicios, almacenamiento, seguridad, HPA y eventos) y registro de evidencias para soporte y auditoría.

c) Criterios de aceptación y métricas de éxito

Se establecieron KPIs operativos: MTTR, disponibilidad durante despliegues (downtime ≈ 0 en rolling updates), activación efectiva de HPA, consistencia entre réplicas deseadas/disponibles, y cumplimiento de políticas de seguridad (RBAC) y

almacenamiento (PVs/PVCs). La aplicación mostró reducción de MTTR, menor intervención manual y mayor estabilidad del servicio.

$RTO_{servicio} \leq 10$ minutos y restauración del servicio = 100% por incidente, validados con readiness de pods y health checks de endpoints²².

d) Transferencia y sostenibilidad.

Se elaboraron runbooks de operación (escalado, rollback, rotación de secretos, respuesta a incidentes), plantillas IaC reutilizables, y lineamientos de hardening (principio de mínimo privilegio, segmentación por namespaces). Esta documentación facilita la adopción y escalamiento del modelo en nuevos servicios sin perder control ni trazabilidad.

e) Limitaciones y consideraciones.

La validación se realizó en escenarios controlados; no se ejecutaron pruebas con cargas de usuario reales ni múltiples repeticiones estadísticas. Se recomienda complementar con *load testing* progresivo y, de ser necesario, integrar un stack de métricas de serie temporal (p. ej., Prometheus/Grafana) para historizar indicadores.

En conclusión, la aplicación de la tecnología encontrada permite pasar de un diseño propuesto a una implementación operativa reproducible y mantenible, alineada con los objetivos de continuidad y resiliencia definidos en este estudio.

5.3. Contraste con trabajos de investigación similares

Se presenta el contraste crítico entre el modelo propuesto de automatización en AWS EKS (IaC + CI/CD + monitoreo con Lens) y doce trabajos previos seleccionados. El análisis se organiza en cuatro ejes: (i) automatización declarativa y GitOps para despliegues trazables; (ii) orquestación y migraciones hacia Kubernetes/EKS como estándar industrial; (iii) observabilidad, autoescalado y continuidad operativa para reducir MTTR y sostener el servicio durante rolling updates; y (iv) enfoques complementarios (FinOps, arquitecturas especializadas y formación). Este encuadre permite establecer coincidencias, diferencias metodológicas y vacíos respecto a nuestro estudio, destacando como aporte diferencial la integración end-to-end (IaC→CI/CD→EKS→observabilidad) con evidencia operativa de continuidad (nodos, pods/workloads, red/servicios, storage,

RBAC, HPA y eventos), sobre la cual se derivan recomendaciones y oportunidades de mejora.

- **(i) Automatización declarativa y GitOps.**

Ripa Elices (2025) demuestra que GitOps (ArgoCD + repositorios fuente de verdad) incrementa trazabilidad y consistencia en EKS. Nuestro modelo coincide en el enfoque declarativo y en el despliegue automatizado, pero se diferencia al documentar métricas operativas de continuidad (Lens: nodos, pods, red, storage, seguridad, HPA, eventos) e incorporar explícitamente MTTR y continuidad operativa como criterios de validación. García Granados (2023) compara Kubernetes y OpenShift, concluyendo que K8s ofrece mayor control; este hallazgo respalda la elección de EKS como plataforma estándar en nuestra propuesta.

- **(ii) Gobernanza de costos y FinOps.**

Ruiz Cabrera (2024) integra IaC y FinOps (Azure) para escalar con criterio técnico-financiero. Aunque nuestro escenario es AWS/EKS, el principio coincide: automatizar decisiones de capacidad. A diferencia de su trabajo, el nuestro profundiza en continuidad operativa con evidencia de métricas en tiempo real, y deja como proyección integrar señales de costo (p. ej., CloudWatch/Cost Explorer) a los umbrales de autoescalado.

- **(iii) Observabilidad autoescalado y disponibilidad.**

Salazar Merchan (2022) valida despliegue y monitoreo web de un clúster K8s; nuestro aporte extiende esa línea con un paquete de observabilidad transversal (Lens IDE sobre nodos, pods/workloads, red/Ingress/servicios, storage—PVs/PVCs/SC—, seguridad RBAC, HPA y eventos/logs), orientado explícitamente a reducir MTTR y sostener uptime en rolling updates. Arrieta & Palacios (2022) enfatizan elasticidad con Cluster Autoscaler/Karpenter; coincidimos en el autoescalado, y añadimos evidencia operativa (réplicas deseadas vs. disponibles, activación HPA) y políticas RBAC como soporte de continuidad.

- **(iv) Migración y modernización cloud hacia Kubernetes/EKS.**

Mora (2023) valida la migración a AWS bajo DevOps + IaC con mejoras de disponibilidad y costos; nuestro trabajo aterriza esos principios en EKS con pipeline CI/CD y observabilidad integral enfocada en continuidad. Cesanelli (2023) muestra la migración ECS→EKS manteniendo servicios; nuestro modelo es complementario: normaliza el ciclo build-push-deploy (ECR + manifiestos/Helm) y añade pruebas operativas (HPA, probes, eventos) para verificar recuperación y estabilidad. Varela Tabares (2019) prueba flujos contenerizados en AWS (ECS/EC2/EFS) para alta demanda; coincidimos en la viabilidad operativa y agregamos la capa de gobernanza operativa (RBAC, storage claims, eventos) como sostén de continuidad.

- **(v) Arquitecturas especializadas y disponibilidad.**

Escoda Gil (2023) diseña HA de bases de datos en K8s (replicación, balanceo); su enfoque de disponibilidad continua refuerza nuestro criterio de rolling updates y health probes para minimizar downtime. Moreno Villarrubia (2024) valida Kubernetes para ingesta IoT en tiempo real; converge con nuestro énfasis en escalabilidad y resiliencia. Petrocelli (2021) explora cómputo intensivo distribuido con Kubernetes, mostrando robustez ante cargas variables; nuestro trabajo lo complementa con criterios explícitos de MTTR y evidencia de autoscaling. Real Ixcayau (2021) sustenta curricularmente la necesidad de Docker/Kubernetes/DevOps; ello respalda la transferencia y sostenibilidad de nuestro modelo (plantillas IaC y runbooks).

a) Síntesis comparativa.

Los antecedentes validan, desde ángulos distintos, que IaC + Kubernetes son el estándar para automatización, elasticidad y control. Nuestra contribución diferencial es integrar: (1) IaC + CI/CD (ECR, manifiestos/Helm, rolling updates), (2) observabilidad de extremo a extremo:

Lens:nodos→pods/workloads→red/servicios→storage→RBAC→HPA→eventos)

y (3) criterios operativos de continuidad (reducción de MTTR, menor intervención manual, verificación de disponibilidad durante despliegues). Frente a trabajos orientados a costos (FinOps) o a plataformas específicas, aquí se aporta una guía replicable centrada

en continuidad operativa sobre AWS EKS, con evidencia técnica directamente vinculada a la hipótesis.

b) Vacíos y oportunidades identificadas.

A partir del contraste, se proponen líneas de mejora: (i) incorporar pruebas de carga y series temporales (Prometheus/Grafana) para historizar métricas y reforzar inferencias; (ii) extender a GitOps (ArgoCD) para cerrar el ciclo declarativo fin a fin; (iii) sumar señales FinOps (presupuesto/uso) a umbrales de escalado; y (iv) ejecutar replicaciones controladas que permitan estimar intervalos de confianza en indicadores de disponibilidad y MTTR.

En conclusión, el estado del arte confirma la idoneidad técnica de IaC y Kubernetes para la automatización y la resiliencia; el presente trabajo avanza al documentar de forma integrada la continuidad operativa en EKS con métricas operativas verificables y una arquitectura aplicable en contextos productivos.

CONCLUSIONES

- a) La investigación permitió diseñar e implementar un modelo de automatización para aplicaciones contenerizadas mediante infraestructura como código sobre Amazon EKS, el cual demostró reducir el tiempo de recuperación (MTTR) en comparación con el proceso manual inicial y mantuvo la continuidad operativa durante los despliegues. La evidencia técnica obtenida en el entorno de pruebas confirmó la viabilidad, reproducibilidad y estabilidad del modelo automatizado, cumpliéndose el propósito central del estudio.
- b) Se identificaron, evaluaron y seleccionaron las herramientas de infraestructura como código más adecuadas para el aprovisionamiento automatizado en AWS, destacando Terraform y AWS CloudFormation. Su implementación permitió desplegar de manera declarativa la VPC, roles de IAM, clúster EKS y node groups, logrando un aprovisionamiento reproducible, auditable y con una reducción significativa de intervenciones manuales. La estabilidad del clúster inicial (nodos listos, servicios base operativos y ausencia de drift) confirmó la pertinencia de la selección realizada.
- c) Se diseñaron e implementaron flujos de automatización para el ciclo de despliegue en Amazon EKS, integrando construcción de imágenes, almacenamiento en ECR, despliegue declarativo mediante manifiestos y Helm Charts, y mecanismos de rolling updates con probes de readiness y liveness. Estos flujos estandarizados permitieron disminuir la intervención manual, asegurar despliegues consistentes y mantener la disponibilidad del servicio, evidenciándose una adecuada alineación entre réplicas deseadas y disponibles, así como la activación efectiva del autoescalado horizontal (HPA).
- d) El entorno de prueba configurado con fallos controlados (CrashLoopBackOff, interrupción del servicio e incrementos de carga) permitió estimar el tiempo de recuperación del modelo automatizado. Los registros mostraron que, ante

incidentes, el clúster ejecutó reprogramación automática de pods, preservó la disponibilidad durante rolling updates y escaló réplicas en función de la demanda. Los resultados evidenciaron recuperaciones completas del servicio en ≤ 10 minutos, demostrando una disminución significativa del MTTR respecto al escenario manual documentado.

- e) El análisis de métricas de CPU, memoria, latencia, réplicas activas y estabilidad de endpoints confirmó que la automatización mejoró la continuidad operativa del servicio y la resiliencia del clúster. Se registró reducción del MTTR, menor aparición de eventos críticos y mantenimiento continuo del servicio durante despliegues y fallos controlados. A partir de los resultados, se plantearon mejoras orientadas a fortalecer la operación, como la adopción de GitOps (ArgoCD), la integración de series temporales (Prometheus/Grafana), la optimización del autoscaling (Cluster Autoscaler o Karpenter) y el fortalecimiento de políticas de seguridad basadas en mínimo privilegio.

RECOMENDACIONES

- a) Se recomienda fortalecer el modelo de automatización implementado mediante la integración de herramientas de monitoreo más completas, como Prometheus y Grafana, que permitan disponer de métricas históricas y alertas tempranas. Esto complementaría el monitoreo utilizado durante la investigación y contribuiría a mejorar la detección oportuna de incidentes y el seguimiento del MTTR y la disponibilidad del servicio.
- b) Se sugiere continuar la estandarización del aprovisionamiento mediante infraestructura como código, ampliando la modularización de los componentes, el versionamiento por ambientes y la documentación técnica del proceso. Esto permitirá mantener la trazabilidad del estado de la infraestructura, reducir errores de configuración y facilitar futuras actualizaciones o ampliaciones del clúster.
- c) Se recomienda optimizar los flujos de automatización para despliegues continuos, incorporando buenas prácticas como el uso de versiones inmutables de imágenes, validaciones previas al despliegue y mecanismos de rollback. Asimismo, la adopción progresiva de estrategias como despliegues graduales podría mejorar la continuidad operativa en escenarios de mayor complejidad.
- d) Se aconseja ampliar las pruebas operativas a escenarios adicionales, incluyendo ensayos de carga y fallos controlados con distintos niveles de intensidad. Esto permitirá estimar el comportamiento del sistema en condiciones más exigentes, validar de manera más precisa el MTTR y fortalecer la capacidad de recuperación ante incidentes.
- e) Se recomienda establecer un proceso continuo de mejora orientado a la resiliencia del clúster, revisando periódicamente las políticas de autoscaling, el uso de recursos y la configuración de seguridad. Asimismo, resultaría beneficioso implementar mecanismos de respaldo y recuperación para asegurar la disponibilidad de datos y servicios ante eventos no previstos.

REFERENCIAS BIBLIOGRÁFICAS

- Afifi, K. (2021, junio 25). Building a CI/CD pipeline to update an AWS CloudFormation StackSets. *AWS DevOps Blog*. <https://aws.amazon.com/blogs/devops>
- Amazon. (2018). *Documentación oficial de AWS*. Amazon Web Services.
- Amazon Web Services. (2024). Amazon EKS architecture. *AWS Documentation*. <https://docs.aws.amazon.com/eks/latest/userguide/what-is-eks.html>
- Amazon Web Services. (s.f.). AWS CloudFormation. <https://docs.aws.amazon.com/cloudformation>
- Amazon Web Services. (2024). AWS Transit Gateway. <https://docs.aws.amazon.com/vpc/latest/tgw/what-is-transit-gateway.html>
- Anvesh Muppada. (2023, mayo 22). Jump box setup on eks cluster. *Medium*. <https://medium.com/@muppadaanvesh/jump-box-setup-on-eks-cluster-383ca92f51ef>
- Appvia. (s.f.). What is etcd? <https://www.appvia.io/blog/what-is-etcd>
- Arrieta Cohen, L. A., & Palacios Pino, T. V. (2022). *Metodología TI para monitorear y mejorar el rendimiento y disponibilidad de la información en la nube*. <https://repository.ucc.edu.co/server/api/core/bitstreams/ae5702b8-e171-4053-8d6d-933722c5d028/content>
- Atlassian. (s.f.). Infrastructure as code (IaC). *Atlassian*. <https://atlassian.com>
- Azure DevOps. (2024). What is infrastructure as code (IaC)? *Microsoft Learn*. <https://learn.microsoft.com>
- Balestrini, M. (2006). *Cómo se elabora el proyecto de investigación*. BL Consultores.
- Cesanelli, D. A. (2023). *Migración de una arquitectura basada en contenedores usando ECS a Kubernetes* (Doctoral dissertation, Universidad Nacional de La Plata). https://sedici.unlp.edu.ar/bitstream/handle/10915/153420/Documento_completo.pdf-PDFA.pdf?sequence=1
- Creswell, J. W. (2014). *Research design: Qualitative, quantitative, and mixed methods approaches* (4th ed.). SAGE Publications.
- Escoda Gil, S. (2023). *Diseño y desarrollo de una arquitectura distribuida para SmartUniversity*. <https://rua.ua.es/dspace/handle/10045/136525>

- Flavius Dinu, & Roper, J. (2025, mayo 16). Terraform AWS Provider: How to Use & Examples. *Spacelift*. <https://spacelift.io>
- García Granados, A. (2023). *Despliegue de aplicaciones contenerizadas en entornos cloud*. Universitat Oberta de Catalunya (UOC). <http://hdl.handle.net/10609/147969>
- HashiCorp. (2023, octubre 9). What Is Infrastructure as Code (IaC)? Why Is It Important? <https://hashicorp.com>
- Hernández Sampieri, R., Fernández Collado, C., & Baptista Lucio, M.P. (2014). *Metodología de la investigación* (6.ª ed.). McGraw-Hill: México D.F.
- Hernández, M., & Flórez, D. (2014). *Computación en la nube: Fundamentos y aplicaciones*. Editorial Académica Española.
- IBM. (2021, octubre 8). What is Infrastructure as Code (IaC)? *IBM Think*. <https://ibm.com>
- Ibrarjee. (2025, enero 10). Mastering AWS EKS: A Step by Step Guide to Launching Kubernetes Cluster with eksctl. *Medium*. <https://medium.com/@ibrarjee.mastering-aws-eks>
- Keating, J., & Chu, V. (2019, febrero 14). Build and Deploy Docker Images to AWS using EC2 Image Builder. *Amazon Web Services Blog*. <https://aws.amazon.com/blogs/devops/build-and-deploy-docker-images-to-aws-using-ec2-image-builder/>
- Kubernetes.io. (2025). Kubernetes components. *Kubernetes Documentation*. <https://kubernetes.io/docs/concepts/overview/components/>
- Lumigo. (2024, marzo 3). Amazon EKS Explained: Architecture, Benefits & Use Cases. <https://lumigo.io/blog/amazon-eks-explained-architecture-benefits-use-cases/>
- Michalowski, M. (2024, julio 16). Benefits and Best Practices for Infrastructure as Code. *DevOps.com*. <https://devops.com>
- Mora, J. A. D. (2023). *Migración a una arquitectura en la nube para el procesamiento de datos abiertos oceanográficos*. Universidad Católica de Colombia. <https://repository.ucatolica.edu.co/server/api/core/bitstreams/2c67fc70-d311-48aa-86f9-184a52f2df84/content>
- Mora, J. A. (2010). *Investigación: Guía para la formulación de proyectos*. Fondo Editorial Universitario.
- Moreno Villarrubia, C. (2024). *Desarrollo de un Sistema Integral para la Monitorización Inteligente de Smart Villages* (Bachelor's thesis). <https://digibug.ugr.es/handle/10481/104234>

- Nogués, S. (2018). *Kubernetes: Despliegue, administración y operación de contenedores*. Marcombo.
- Pandugula, C., & Syed, S. (2024, abril 1). The Evolution of Cloud Infrastructure Automation: A Deep Dive into <https://reviewofconphil.com>
- Petrocelli, D. M. (2021). *Plataforma colaborativa, distribuida, escalable y de bajo costo basada en microservicios, contenedores, dispositivos móviles y servicios en la Nube para tareas de cómputo intensivo* (Doctoral dissertation, Universidad Nacional de La Plata). https://sedici.unlp.edu.ar/bitstream/handle/10915/122360/Documento_completo.pdf?sequence=1
- Real Ixcayau, E. O. (2021). *Propuesta para actualización curricular en el área de software incorporando talleres de Docker y Kubernetes*. Universidad de San Carlos de Guatemala. <http://www.repositorio.usac.edu.gt/15758/>
- Redhat. (2017). *Documentación oficial de Kubernetes y OpenShift*. Red Hat Inc.
- Rengifo, J. (2013). *Introducción a cloud computing*. Ediciones Universidad Cooperativa de Colombia.
- Ripa Elices, Á. J. (2025). *Aplicación de metodologías GitOps en el despliegue de aplicaciones distribuidas*. Universitat Oberta de Catalunya (UOC). <http://hdl.handle.net/10609/151896>
- Ruelas, M. (2017). *Contenedores Docker: Guía práctica para su uso eficiente*. Alfaomega Grupo Editor.
- Ruiz Cabrera, Á. (2024). *Framework de Automatización FinOps: Mejorando la eficiencia de Costos en Despliegues de Cloud*. https://repositorio.comillas.edu/xmlui/bitstream/handle/11531/86886/TFM_Alvaro_Ruiz_Cabrera_signed.pdf
- Saintcross, M., & Ranes, A. (2023). Build a pipeline for hardened container images using EC2 Image Builder and Terraform. *AWS Prescriptive Guidance*. <https://docs.aws.amazon.com/prescriptive-guidance/latest/patterns/build-a-pipeline-for-hardened-container-images-using-ec2-image-builder-and-terraform.html>
- Salazar Merchan, Z. B. (2022). *Despliegue automático de un cluster de Kubernetes y monitorización web* (Bachelor's thesis, Universitat Politècnica de Catalunya). <https://upcommons.upc.edu/bitstream/handle/2117/375811/memoria.pdf>
- Selltiz, C., Jahoda, M., Deutsch, M., & Cook, S. W. (1976). *Métodos de investigación en las relaciones sociales* (2.ª ed.). Trillas.
- ShiftAsia. (2022, marzo). Grant access to EKS cluster for Bastion Host. <https://blog.shiftasia.com/grant-access-to-aws-eks-cluster-to-bastion-host>

- Spacelift. (2025, mayo 23). Infrastructure as Code: Best Practices, Benefits & Examples. <https://spacelift.io>
- Spot.io. (2025). Kubernetes Architecture Explained. <https://spot.io/blog/kubernetes-architecture/>
- Tamayo y Tamayo, M. (2006). *El proceso de la investigación científica*. Limusa.
- Varela Tabares, D. (2019). *Diseño e implementación de un flujo de trabajo bioinformático en la nube para la identificación de Variantes oncogénicas a partir de datos genómicos*. <https://core.ac.uk/download/pdf/344711243.pdf>
- XenonStack. (2024). Infrastructure as code Principles, Tools and Best Practice. <https://xenonstack.com>

ANEXOS

ANEXO 1 Matriz de consistencia

Problemas	Objetivos	Hipótesis	Variables	Método
General	General	General	Independiente	Tipo
¿Cómo la automatización de aplicaciones contenerizadas con infraestructura bajo código en EKS reduce el tiempo de recuperación y garantiza la continuidad operativa sobre la nube de AWS?	Automatizar aplicaciones contenerizadas con infraestructura bajo código en EKS para reducir el tiempo de recuperación y para garantizar la continuidad operativa sobre la nube de AWS	La automatización de aplicaciones contenerizadas con infraestructura bajo código en Amazon EKS sí reduce significativamente el tiempo de recuperación y garantiza la continuidad operativa sobre la nube de AWS.	Automatización de aplicaciones contenerizadas con IaC.	Aplicada Enfoque Cuantitativo
			1° Dimensión V.I. Automatización de recursos Indicadores - Nivel de intervención manual -Reproductibilidad del aprovisionamiento IaC	Diseño Preexperimental
				Técnica Observación estructurada
				Instrumento Registro técnico, logs, cronómetro, consola AWS
Específicos	Específicos	Específicas		
a)¿Cómo influyen la identificación y selección de herramientas de Infraestructura como Código (IaC) en la automatización del aprovisionamiento de recursos en AWS y configuración de clústeres EKS? b)¿Cómo el diseño del flujo de automatización (CI/CD) para el despliegue y gestión de contenedores en Amazon EKS influye en la capacidad de recuperación del sistema ante fallos simulados? c)¿Cómo se ve afectado el tiempo de recuperación (MTTR) del modelo automatizado después de la implementación de un entorno de prueba con fallos controlados sobre AWS? d)¿Cuál es el impacto en la continuidad operativa del modelo automatizado a través de la integración entre herramientas IaC y orquestadores como EKS?	a)Identificar y seleccionar herramientas de Infraestructura como Código (IaC) para automatizar el aprovisionamiento de recursos en AWS y configuración de clústeres EKS. b)Diseñar flujos de automatización (CI/CD) para el despliegue y gestión de contenedores en Amazon EKS. c)Analizar el tiempo de recuperación (MTTR) del modelo automatizado después de la implementación de un entorno de prueba con fallos controlados sobre AWS d)Analizar el impacto en la continuidad operativa del modelo automatizado a través de la integración entre herramientas IaC y orquestadores como EKS	a) La identificación y selección de herramientas de Infraestructura como Código (IaC) permite automatizar eficazmente el aprovisionamiento de recursos en AWS y configuración de clústeres EKS, disminuyendo la intervención manual. .b) El diseño del flujo de automatización (CI/CD) para el despliegue y gestión de contenedores en Amazon EKS mejora la capacidad de recuperación del sistema ante fallos simulados. c) El tiempo de recuperación (MTTR) del modelo automatizado se reduce en comparación con procesos manuales después de la implementación de un entorno de prueba con fallos controlados sobre AWS. d) La continuidad operativa del modelo automatizado a través de la integración entre herramientas IaC y orquestadores como EKS tiene un impacto positivo	2° Dimensión V.I. Automatización de despliegue Indicadores -Cobertura del pipeline automatizado -Verificación de despliegue automático en validación	
			Dependiente	Población
			Tiempo de recuperación y continuidad operativa.	- Aplicaciones contenerizadas desplegadas en AWS Clusters en Amazon EKS -Entornos tecnológicos susceptibles de automatización Muestra -Estado inicial manual o no estandarizado, -Compatibilidad con Amazon EKS,
			1° Dimensión V.D. Tiempo de Recuperación Indicadores -Duración de la caída hasta restauración -Comparación entre tiempo manual vs automatizado	-Posibilidad de recrear escenarios pretest–postest, -Condiciones técnicas para validar la automatización y las mejoras en continuidad operativa.
			2°Dimensión V.D. Continuidad operativa Indicadores -Porcentaje de disponibilidad del servicio -Número de fallos críticos durante pruebas	

ANEXO 2 Descripción del Flujo Arquitectónico en AWS Antamina Cloud

Network Account

En esta cuenta se encuentra la Public DMZ Productivo VPC, que incluye:

- WAF (Web Application Firewall): filtra y protege contra ataques comunes a nivel de aplicación (SQLi, XSS, bots, etc.).
- Public ALB (Application Load Balancer): balancea el tráfico HTTP/HTTPS proveniente de Internet hacia los destinos internos.
- TG IP Mode to NLB (Ingress DNS): reglas de enrutamiento que permiten redirigir tráfico desde el ALB público hacia los NLB internos de la cuenta de aplicación.

Asimismo, en la Private DMZ Productivo VPC se despliega un Private ALB, encargado de distribuir tráfico entre servicios privados, accesibles únicamente desde la red interna. Dicho tráfico se enruta mediante reglas *TG IP Mode to NLB* y se gestiona a través del Transit Gateway, que centraliza la comunicación entre la Network Account y la Application Account.

Application Account

Dentro de la VPC privada de esta cuenta se gestionan los principales servicios de la aplicación:

- Cognito: encargado de la autenticación y autorización de usuarios, incluyendo login, federación y manejo de tokens de acceso.
- Amazon S3: servicio de almacenamiento de objetos, donde pueden alojarse archivos estáticos (frontend) o configuraciones.
- NLB (Network Load Balancer): balancea tráfico TCP/UDP y direcciona hacia instancias EC2 o hacia el clúster de EKS.
- EC2 (SVCLD_DB): instancia destinada a la base de datos de la aplicación.

EKS (Elastic Kubernetes Service)

El clúster EKS representa el núcleo de la automatización, y se organiza de la siguiente forma:

- Ingress-nginx: conformado por *Nginx Pods* que actúan como proxy inverso, el *Ingress Nginx Controller Service* que expone los Ingress Controllers hacia el NLB, y el *Ingress Nginx Controller*, encargado de administrar las reglas de enrutamiento dentro del clúster.
- Aplicaciones (App): se despliegan en *namespaces*, los cuales permiten organizar y aislar recursos según entornos (dev, qa, prod).
 - ServiceAccount: identifica a los pods para interactuar con el API de Kubernetes y permite la aplicación de permisos mediante RBAC.
 - Deployments y Pods: cada *Deployment* gestiona los pods, replicas, escalado y actualizaciones.
 - Services: actúan como punto de acceso estable, internos (*ClusterIP*) o externos (*LoadBalancer*, *NodePort*).

- Ingress: gestiona el tráfico HTTP/HTTPS hacia los servicios internos, definiendo reglas por dominio o ruta y permitiendo certificados TLS.

● Otros componentes

- Auth (microservicio de autenticación): integración adicional que complementa a Cognito o a otros IdPs.
- DNS / Dominios: resuelven el enrutamiento de tráfico hacia los balanceadores (ej. *app.asip.antamina.com*).
- User (Cliente externo): usuario final que accede desde Internet, pasando por el WAF/ALB y luego hacia Cognito o Auth, hasta llegar a los endpoints internos de la aplicación.

● Resumen del flujo principal

1. El usuario accede a través de un dominio público (ej. *api.app.antamina.com*).
2. El tráfico pasa por el WAF y el ALB Público en la cuenta de red.
3. El enrutamiento se realiza mediante el Transit Gateway hacia la VPC privada de aplicación.
4. El NLB redirige hacia el clúster EKS (Ingress Nginx Controller).
5. El Ingress enruta hacia los pods correspondientes.
6. La autenticación se gestiona con Cognito y/o el microservicio Auth.
7. Finalmente, S3 provee contenido estático como imágenes, archivos o configuraciones.