Universidad Nacional de Loja
Facultad de la Energía, las Industrias y los de Recursos Naturales No Renovables

Maestría en Ingeniería en Software

Implementación de pruebas de carga y estrés automatizadas basadas en Chaos Engineering para medir la tasa de disponibilidad de un Clúster Kubernetes
Implementation of automated load and stress tests based on Chaos Engineering to measure the availability rate of a Kubernetes cluster

Trabajo de Titulación, previo a la obtención del título de Magíster en Ingeniería en Software

AUTOR:
Maria Anabel Encalada Córdova

DIRECTOR:
Ing. Wilman Patricio Chamba Zaragocín Mg. Sc.

Loja - Ecuador
2025

Certificación

Loja, 18 de marzo de 2025

Ing. Wilman Patricio Chamba Zaragocín, Mg. Sc.

DIRECTOR DEL TRABAJO DE TITULACIÓN

Certifico:

Que he revisado y orientado todo proceso de la elaboración del Trabajo de Titulación denominado: **Implementación de pruebas de carga y estrés automatizadas basadas en Chaos Engineering para medir la tasa de disponibilidad de un Clúster Kubernetes** previo a la obtención del título de Magíster en Ingeniería en Software, de autoría del estudiante **Maria Anabel Encalada Córdova**, con cédula de identidad Nro. **0707031662**, una vez que el trabajo cumple con todos los requisitos exigidos por la Universidad Nacional de Loja para el efecto, autorizo la presentación para la respectiva sustentación y defensa.

Firmado electrónicamente por:
WILMAN PATRICIO
CHAMBA ZARAGOCIN

Ing. Wilman Chamba Zaragocín, Mg.Sc.

DIRECTOR DEL TRABAJO DE TITULACIÓN

Autoría

Yo, **Maria Anabel Encalada Córdova**, declaro ser autor del Trabajo de Titulación y eximo expresamente a la Universidad Nacional de Loja y a sus representantes jurídicos de posibles reclamos y acciones legales, por el contenido del mismo. Adicionalmente acepto y autorizo a la Universidad Nacional de Loja la publicación del Trabajo de Titulación en el Repositorio Digital Institucional – Biblioteca Virtual.

Firma: 

Cédula de Identidad: 0707031662

Fecha: 18/03/2025

Correo electrónico: maria.a.encalada@unl.edu.ec

Teléfono: +593997940592

Carta de autorización por parte del autor, para consulta, reproducción parcial o total y/o publicación electrónica de texto completo, del Trabajo de Titulación

Yo, **Maria Anabel Encalada Córdova**, declaro ser autor del Trabajo de Titulación denominado: **Implementación de pruebas de carga y estrés automatizadas basadas en Chaos Engineering para medir la tasa de disponibilidad de un Clúster Kubernetes** como requisito para optar el título de **Magíster en Ingeniería en Software**, autorizo al sistema Bibliotecario de la Universidad Nacional de Loja para que con fines académicos muestre la producción intelectual de la Universidad, a través de la visibilidad de su contenido de la siguiente manera en el Repositorio Institucional.

Los usuarios pueden consultar el contenido de este trabajo en el Repositorio Institucional, en las redes de información del país y del exterior con las cuales tenga convenio la Universidad.

La Universidad Nacional de Loja, no se responsabiliza por el plagio o copia del Trabajo de Titulación que realice un tercero.

Para constancia de esta autorización, en la ciudad de Loja, a los dieciocho días del mes de marzo de dos mil veinticinco.

Firma: 

Autor: Maria Anabel Encalada Córdova

Cédula de identidad: 0707031662

Dirección: Machala 8805

Correo electrónico: maría.a.encalada@unl.edu.ec

Teléfono: +5939979405

DATOS COMPLEMENTARIOS:

Director del Trabajo de Titulación: Ing. Wilman Chamba Zaragocín, Mg.

Dedicatoria

A mi padre, quien siempre ha estado conmigo, apoyándome en cada proyecto y creyendo en mí incluso cuando las cosas parecían difíciles.

Y a mi compañero de aventuras, con quien escuché el tema ganador que inspiró este Trabajo de Titulación y quien fue testigo de cada paso que dio hasta convertirse en realidad.

Su apoyo y compañía han sido mi mayor fortaleza.

Gracias por ser mi constante.

Maria Anabel Encalada Córdova

Agradecimiento

A Dios, por ser mi guía y fortaleza en cada paso de este camino.

A mi padre, mi madre y mis hermanos, quienes sacrificaron momentos juntos para que yo pudiera dedicarme plenamente a este proyecto. Su amor incondicional y su apoyo constante han sido fundamentales para alcanzar esta meta.

A Víctor, gracias por tu paciencia y comprensión, por acompañarme en esta aventura y creer en mis decisiones, incluso cuando implicaban desafíos y sacrificios.

Al Ing. Wilman Chamba Zaragocín, director de este Trabajo de Titulación, quien no solo ha sido un guía invaluable durante este proceso, sino también un amigo constante a lo largo de mi vida académica. Su dedicación, consejos y apoyo han dejado una huella imborrable en mi formación profesional.

A todos ustedes, quienes han sido mi pilar durante esta Maestría en Ingeniería de Software, les debo más de lo que las palabras pueden expresar. Este logro es tanto mío como suyo.

Índice de contenidos

Índice de tablas

Índice de figuras

Índice de anexos

1. Título

Implementación de pruebas de carga y estrés automatizadas basadas en Chaos Engineering para la medir la tasa de disponibilidad de un Clúster Kubernetes

2. Resumen

Este trabajo explora la implementación de Chaos Engineering en un clúster Kubernetes para evaluar su resiliencia frente a fallos intencionales y no previstos. El objetivo principal fue determinar cómo las pruebas automatizadas de fallos aleatorios pueden mejorar la disponibilidad y tolerancia a errores de los servicios desplegados en Kubernetes. Se utilizó una metodología experimental basada en la integración de herramientas de Chaos Engineering como Chaos Mesh para inyectar fallos simulados en nodos, pods y redes del clúster. Los experimentos se ejecutaron en un entorno controlado, recopilando métricas de tiempo de servicio. Los resultados demostraron que, tras la aplicación de pruebas iterativas, el clúster logró su objetivo de nivel de servicio del 99% de disponibilidad en comparación a un promedio inicial del 58%. Además, se identificaron configuraciones críticas en Kubernetes que pueden optimizarse para garantizar una recuperación más rápida, como políticas de rescheduling y ajustes en los límites de recursos. Se concluye que Chaos Engineering es una herramienta efectiva para fortalecer la resiliencia de clústeres Kubernetes, ofreciendo beneficios tangibles en la estabilidad del sistema y proporcionando un enfoque práctico para la mejora continua en entornos de producción.

Palabras clave: Chaos engineering, Kubernetes, Resiliencia, Tolerancia a fallos, Pruebas automatizadas, Disponibilidad.

Abstract

This paper explores the implementation of Chaos Engineering on a Kubernetes cluster to evaluate its resilience to intentional and unintended failures. The main objective was to determine how automated random fault testing can improve the availability and fault tolerance of services deployed on Kubernetes. An experimental methodology based on the integration of Chaos Engineering tools such as Chaos Mesh was used to inject simulated failures into cluster nodes, pods and networks. The experiments were run in a controlled environment, collecting service time metrics. The results showed that, after iterative testing, the cluster achieved its service time goal, the cluster achieved its service level target of 99% availability compared to an initial average of 58%. In addition, critical configurations were identified in Kubernetes that can be optimized to ensure faster recovery, such as rescheduling policies and rescheduling policies and resource limit adjustments. It is concluded that Chaos Engineering is an effective tool to strengthen the resiliency of Kubernetes clusters, offering tangible benefits Kubernetes clusters, offering tangible benefits in system stability and providing a practical approach for continuous a practical approach for continuous improvement in production environments.

Key Word: Chaos engineering, Kubernetes, Resilience, Fault tolerance, Automated testing, Availability.

3. Introducción

El 100% de las empresas tienen al día de hoy al menos un servicio en la nube y en otros casos, todo su modelo de negocios[1]. Aplicaciones potentes y robustas requieren ser sometidas a pruebas de infraestructura en un entorno de producción. Kubernetes, un entorno nativo de cloud ha sido protagonista de muchas soluciones informáticas por sus funciones de orquestar contenedores complementadas con gestión de recursos y varios objetos para facilitar su configuración y uso[2].

¿Cómo probar este orquestador que en un entorno de desarrollo y pruebas si puede llegar a parecer una solución infalible y resiliente? Además, los escenarios como pérdida de red, latencia o incomunicación en PODs probablemente no se consideren en el paquete de pruebas ya que éstas se suelen centrar en la aplicación y sus diferentes funciones[3][4]. Algo que desde 2013, cuando Netflix estuvo fuera de servicio durante 72 horas, quedó demostrado como una necesidad en aplicaciones que tengan grandes volúmenes de usuarios es la urgencia de pruebas que sometan a condiciones extremas[5] en el entorno de destino, es decir, probar la infraestructura en producción ya que solo este entorno funciona como producción y es ahí donde se deben medir tasas de disponibilidad y otras métricas[6].

La herramienta para ejecutar pruebas de esta naturaleza fue desarrollada en su inicio por el mismo Netflix, lanzando la primera tecnología de la familia de Chaos Engineering, Chaos Monkey[5]. Esta teoría propone que se desarrollen pruebas controladas sobre el entorno de producción para identificar fallos antes de que estos pasen con usuarios reales[7]. Ofrece un modelo de tests que, una vez ejecutados, regresan la aplicación a su estado previo a menos que el daño sea muy crítico y la reparación no se pueda hacer sin la intervención de especialistas[8].

Las herramientas necesarias para la aplicación de Ingeniería del Caos forman parte de la filosofía cultural de DevOps[9]. La automatización de tareas aplicando la práctica de desarrollo y operaciones por medio de ingeniería de softwares es una de las funciones que vuelven atractivo al Site Reliability Engineering[10] para la construcción y despliegue de aplicaciones de forma más rápida y eficiente.

Ante esto, cuestionarse la resiliencia de un clúster Kubernetes expuesto a pruebas automatizadas basadas en Chaos Engineering aplicando prácticas de Site Reliability Engineering da origen al presente Trabajo de Titulación que busca responder la siguiente pregunta de investigación: "¿Cuál es la tasa de disponibilidad que poseerá un clúster Kubernetes en condiciones estables después de aplicar pruebas de carga y estrés basadas de Chaos Engineering?". Estableciendo el objetivo general "Determinar la tasa de disponibilidad de un clúster Kubernetes en condiciones estables antes y después de aplicar pruebas de carga y estrés basadas de Chaos Engineering" y para lograrlo 2 objetivos específicos "Establecer escenarios de pruebas de carga y estrés en un sistema informático para la toma de pedidos en cafeterías físicas de la ciudad de Loja desplegada en un clúster Kubernetes, basados en la teoría de Chaos Engineering, para introducir fallos controlados" y "Estimar la tasa de disponibilidad de un sistema informático basado en los principios de Site Reliability Engineering (SRE), comparando resultados antes y después de implementar mejoras derivadas de pruebas de carga y estrés".

El desarrollo de los objetivos específicos fue precedido por una revisión de obra gris, tomando como referencia estudios científicos con propósitos similares y/o aplicación de principios correspondientes a las teorías que forman parte del proyecto. Además de la consulta en sitios oficiales respecto a documentación de tecnologías y herramientas para su implementación durante el segundo objetivo específico.

La estructura del presente informe, se compone de la siguiente manera: el **Marco Teórico** expone antecedentes y trabajos relacionados referentes al tema, incluye también los conceptos necesarios para la comprensión del tema principal; la **Metodología** presenta el área de estudio, los procedimientos y recursos utilizados para el desarrollo del presente TT; el apartado de **Resultados** presenta detalladamente las evidencias obtenidas durante el desarrollo de cada objetivo planteado; dentro de la **Discusión** se analizan los resultados obtenidos desde la perspectiva del autor; y para finalizar, se presentan la **Conclusiones y Recomendaciones** donde se abarcan los hechos más relevantes y sugerencias sobre el TT.

4. Marco teórico

5.1 Antecedentes

La creciente adopción de plataformas de contenedores, como Kubernetes, ha transformado el panorama de la computación en la nube y la arquitectura de aplicaciones modernas[4]. Kubernetes se ha posicionado como el orquestador de contenedores líder, proporcionando funcionalidades avanzadas de escalabilidad, alta disponibilidad y tolerable a fallos[11]. Sin embargo, a medida que las organizaciones migran sus cargas de trabajo a Kubernetes, han surgido desafíos relacionados con la resiliencia y la estabilidad de estos entornos distribuidos.

La complejidad inherente de los clústeres de Kubernetes, con múltiples componentes interconectados y dependencias entre ellos, hace que la identificación y mitigación de puntos de fallo sea una tarea desafiante[12]. Los fallos en los nodos, problemas de red, errores de configuración y agotamiento de recursos son algunos de los problemas comunes que pueden afectar la disponibilidad y el rendimiento de los sistemas basados en Kubernetes[13]. Estos eventos disruptivos pueden tener un impacto significativo en la experiencia del usuario final y generar pérdidas económicas para las organizaciones que dependen de la fiabilidad de sus aplicaciones. Para abordar estos desafíos, las empresas han adoptado enfoques como Site Reliability Engineering (SRE) y Chaos Engineering, que se centran en mejorar la resiliencia de los sistemas distribuidos[14]. Mientras que SRE proporciona prácticas y principios para la gestión de la confiabilidad[15], Chaos Engineering se enfoca en introducir de manera controlada fallos y disrupciones en el sistema, con el objetivo de observar y mejorar su comportamiento bajo condiciones de estrés[14]. La combinación de estos enfoques ha demostrado ser efectiva en la identificación y mitigación de vulnerabilidades en entornos de Kubernetes, lo que a su vez aumenta la disponibilidad y la capacidad de recuperación de estos sistemas críticos[16].

5.2 Tecnologías

5.2.1 Pruebas de carga y estrés

Las pruebas de carga y estrés se utilizan para evaluar el desempeño y la resiliencia de los sistemas informáticos bajo condiciones de uso intensivo o extremo[17]. Estas pruebas implican la generación de una carga de trabajo artificial que simula patrones de uso realistas, con el objetivo de identificar cuellos de botella, puntos de fallo y límites de capacidad[18]. Al someter al sistema a niveles de estrés controlados, los equipos de desarrollo y operaciones pueden probar la estabilidad y escalabilidad de sus aplicaciones y infraestructura, lo que les permite tomar medidas correctivas antes de que se produzcan incidentes en producción[19].

5.2.2 Disponibilidad de un sistema TI

La disponibilidad de un sistema de Tecnologías de la Información (TI) se define como el porcentaje de tiempo durante el cual el sistema está operativo y accesible para los usuarios[20]. Esta métrica es fundamental para evaluar la confiabilidad y el desempeño de los sistemas críticos de una organización. Para calcular la disponibilidad, se suelen utilizar métricas como el tiempo medio entre fallas (MTBF) y el tiempo medio de reparación (MTTR), las cuales permiten cuantificar la capacidad del sistema para mantener un funcionamiento ininterrumpido[21].

5.2.3 Kubernetes

Kubernetes es una plataforma de orquestación de contenedores de código abierto que automatiza la implementación, el escalado y la gestión de aplicaciones en contenedores[22]. Desarrollado originalmente por Google, Kubernetes se ha consolidado como el estándar de facto para la gestión de cargas de trabajo en entornos de nube y ha sido ampliamente adoptado por organizaciones de todo el mundo[23]. Sus capacidades de alto rendimiento, escalabilidad y tolerancia a fallos lo convierten en una solución ideal para desplegar y administrar aplicaciones distribuidas y de microservicios[24].

5.2.4 Chaos Engineering

Chaos Engineering es una disciplina que se enfoca en construir sistemas resilientes mediante la introducción controlada de fallos y disrupciones[25]. Esta metodología se basa en el principio de que, al probar el comportamiento del sistema bajo condiciones de estrés, se pueden identificar y mitigar vulnerabilidades antes de que se produzcan incidentes reales[26]. Las técnicas de Chaos Engineering implican la inyección de fallos, el uso de herramientas de análisis de caos y la observación del sistema para evaluar su capacidad de recuperación[27].

5.2.5 Site Reliability Engineering (SRE)

Site Reliability Engineering (SRE) es un enfoque desarrollado por Google para la gestión de la confiabilidad de los sistemas[10]. SRE combina principios de ingeniería de software y operaciones de TI para garantizar que los sistemas a gran escala sean escalables, eficientes y resilientes[28]. Los equipos de SRE son responsables de definir objetivos de nivel de servicio (SLOs), implementar prácticas de monitorización y observabilidad, y aplicar técnicas de Chaos Engineering para mejorar la disponibilidad y la capacidad de recuperación de los sistemas[29].

5.2.6 Prometheus

Prometheus es una herramienta de monitorización y alerta de código abierto, diseñada específicamente para entornos de nube y microservicios[30]. Proporciona una forma flexible y escalable de recopilar, almacenar y consultar métricas de rendimiento y estado de los sistemas[31]. Prometheus se ha convertido en un componente fundamental en la implementación de soluciones de observabilidad en arquitecturas basadas en Kubernetes, ya que permite a los equipos de SRE y DevOps monitorizar y analizar el estado del clúster y las aplicaciones[32].

5.2.7 Grafana

Grafana es una plataforma de visualización de datos de código abierto que se utiliza ampliamente para crear cuadros de mando y paneles de control personalizados[33]. Integrada con diversas fuentes de datos, incluyendo Prometheus, Grafana permite a los equipos de TI visualizar, analizar y correlacionar métricas clave de rendimiento y disponibilidad de sus sistemas[34]. Esta herramienta facilita la supervisión

y la toma de decisiones informadas basadas en datos, lo que es fundamental para la implementación efectiva de prácticas de SRE y Chaos Engineering[35].

5.2.8 DigitalOcean

DigitalOcean es un proveedor de servicios de nube pública que ofrece soluciones de infraestructura como servicio (IaaS)[36]. Con una inter- faz sencilla y una amplia gama de recursos, DigitalOcean se ha con- vertido en una opción popular para que los desarrolladores y equipos de operaciones implementen y gestionen entornos de Kubernetes[37]. Gracias a su integración con herramientas de automatización y monitorización, DigitalOcean facilita la creación de entornos de prueba y la ejecución de experimentos de Chaos Engineering en clústeres Kubernetes[38].

5.3 Trabajos relacionados

5.3.1 Evaluación de la preparación operativa mediante simulaciones de ingeniería del caos en arquitecturas Kubernetes para Big Data

En el contexto de la complejidad creciente de los sistemas interconectados en entornos de producción, surge la necesidad de evaluar la preparación operativa de las plataformas. Este estudio aborda dicha necesidad mediante la implementación de simulaciones de ingeniería del caos en arquitecturas Kubernetes para Big Data. Se busca fortalecer aspectos críticos como la copia de seguridad, restauración, transferencia de archivos de red, conmutación por fallo y seguridad general. Para ello, se induce caos en el entorno Kubernetes, eliminando pods aleatorios que procesan datos de dispositivos de borde en centros de datos. A partir de este escenario, se analiza el tiempo de recuperación de los pods y se estima el tiempo de respuesta. Este enfoque per- mite evaluar la preparación operativa de la plataforma y mejorar su resiliencia frente a fallos y errores[39].

5.3.2 Ingeniería de Caos para Microservicios

La ingeniería del caos es un concepto relativamente nuevo que está ganando popularidad, ya que ayuda a las empresas a ser más resilientes frente a fallos inesperados en la red o en el software. La idea detrás de la ingeniería del caos es que, si se pueden

crear fallos controlados, se pueden descubrir los puntos débiles del sistema y corregirlos antes de que ocurra un problema en el entorno de producción. Esta investigación se centró en los microservicios, que son pequeños fragmentos de código que realizan tareas específicas en nombre de una aplicación más grande. Los microservicios suelen estar alojados en servidores diferentes y ser administrados por equipos distintos, lo que los hace más frágiles que las aplicaciones monolíticas. Además, los microservicios suelen estar escritos en diferentes lenguajes, lo que los hace más difíciles de entender y probar en busca de errores. El objetivo de este estudio fue determinar si los microservicios pueden ser más resilientes mediante la ingeniería del caos, específicamente si es posible identificar los tipos de fallos que ocurren con más frecuencia y cuánto tiempo toma resolverlos[40].

5.3.3 Creación de un Framework para la práctica de la Ingeniería del Caos

Verificar la capacidad de un sistema de software para soportar condiciones turbulentas en su entorno de producción es una tarea compleja, pero es de vital importancia para las empresas modernas, que pueden sufrir grandes pérdidas si sus sistemas no son lo suficientemente resilientes para resistir la turbulencia inevitable. Esta tesis propone un marco para trabajar con la ingeniería del caos, una disciplina que busca evaluar y mejorar la resiliencia de un sistema de software mediante un enfoque experimental que expone intencionalmente al sistema a diversas condiciones problemáticas para investigar cómo las maneja. El marco propuesto consiste en cuatro actividades que se basan en ocho documentos de apoyo, que, en conjunto, con la ayuda de doce herramientas de ingeniería del caos de código abierto, buscan definir e implementar una práctica continua y extensible de ingeniería del caos para un sistema de software. El marco se diseñó para adaptarse a las aplicaciones desarrolladas en ICA Gruppen AB y se evaluó aplicando partes de él al sistema de software que constituye el sitio web, incluida la tienda en línea accesible desde el sitio web. Las partes aplicadas se consideraron factibles y descubrieron con éxito un conjunto de oportunidades iniciales de mejora para la resiliencia del sistema, así como una práctica adecuada de ingeniería del caos para futuras pruebas de resiliencia del sistema[40].

5. Metodología

5.1. Área de estudio

El presente Trabajo de Titulación se desarrolló sobre un clúster kubernetes compuesto de 4 microservicios desarrollado por los autores de este TT para la gestión de pedidos en cafeterías físicas de cafeterías en la ciudad de Loja aplicando una arquitectura distribuida. El entorno de pruebas se configuró en un ambiente cloud alojado en DigitalOcean con las siguientes características de infraestructura:

Infraestructura dónde se ejecuta la aplicación.

Nodos: 2

1CPU, 2GB RAM, 50GB Store

Infraestructura dónde se ejecuta el stack de monitoreo compuesto por Prometheus y Grafana.

Nodos: 1

2CPU, 2GB RAM, 60GB Store

5.2. Procedimiento

Para la ejecución del presente proyecto se establecieron actividades específicas por cada objetivo específico, éstas se encuentran listadas a lo largo de la sección.

5.2.1. Establecer escenarios de pruebas de carga y estrés en un clúster Kubernetes, basados en la teoría de Chaos Engineering, para introducir fallos controlados.

- Se realizó una revisión de obra gris y revisión de documentación técnica para la definición de métricas que apunten a recolectar datos para la medición de la tasa de disponibilidad.

- Para la selección de herramientas de pruebas se realizó una revisión de tecnologías con criterios de libertad y soberanía tecnológica y en concordancia con la ingeniería del caos y SRE.

- Con la información contenida y el conocimiento en Chaos Engineering, se planifica diseñar escenarios de prueba para medir resiliencia y disponibilidad.

5.2.2. Medir la tasa de disponibilidad de un clúster Kubernetes basado en los principios de Site Reliability Engineering (SRE), comparando resultados antes y después de implementar mejoras derivadas de pruebas de carga y estrés.

- En base a principios y prácticas SRE se estableció un plan de medición de tasa de disponibilidad del clúster kubernetes para la recolección de datos de rendimientos del clúster Kubernetes.

- Se ejecutaron pruebas de carga y estrés sobre el entorno configurado con aplicaciones de monitoreo y captura de datos vinculadas.

- Se ejecutó una segunda evaluación posterior a realizar mejoras necesarias identificadas producto de la primera ejecución de las pruebas.

5.3. Recursos

Para dar cumplimiento a los objetivos propuestos, se utilizaron los siguientes recursos:

5.3.1. Recursos científicos

5.3.1.1. Investigación bibliográfica

Mediante el uso de esta técnica se logró recabar información fundamentada científicamente que aportó a la estructuración del presente trabajo de titulación. Se tomaron como fuentes de información: artículos científicos, revistas científicas, tesis y libros.

5.3.2. Recursos técnicos

5.3.2.1. Documentación técnica

Por medio del uso de sitios oficiales se logró configurar herramientas y tecnologías utilizadas, así como una ejecución exitosa de las pruebas. Se tomó como referencia los sitios detallados en las hojas de instrucciones de la plataforma de computación en la nube usada, así como las páginas web correspondientes a los recursos usados.

5.3.3. Recursos de hardware y software

5.3.3.1. Hardware

- Computador con acceso a Internet: Utilizado para la realización del presente TT.

5.3.3.2. Software

- Mendeley Reference Manager: Utilizado para la gestión de bibliografía.
- VPS correspondiente a DigitalOcean.
- Stack de monitoreo (Prometheus y Grafana).
- Chaos mesh

5.3.4. Recursos humanos

Para el desarrollo del presente TT, se contó con los siguientes participantes:

- Ing. Maria Anabel Encalada Córdova, autora del trabajo de investigación
- Ing. Wilman Patricio Chamba Zaragocín, director y revisor del proyecto.

6. Resultados

Con la finalidad de exponer la evidencia de los resultados obtenidos para el cumplimiento de los objetivos específicos establecidos para el presente TT, se obtuvieron los siguientes resultados.

6.1. Establecer escenarios de pruebas de carga y estrés en un clúster Kubernetes, basados en la teoría de Chaos Engineering, para introducir fallos controlados.

El cumplimiento del primer objetivo estuvo guiado por los principios de la Ingeniería del Caos (Ver marco teórico) y las siguientes fases: (i) Definición de métricas, (ii) Selección de herramientas y (iii) Diseño de escenarios de pruebas. Estas actividades se detallan desde la sección 6.1.1. hasta 6.1.3.

6.1.1. Definir métricas.

La definición de métricas respondió a necesidades de disponibilidad, rendimiento y recursos. Dando como resultado las métricas expuestas en la tabla 1.

Tabla 1: Métricas a capturar sobre el clúster

Disponibilidad	**Rendimiento**	**Recursos**
Tiempo de actividad del clúster	Latencia de respuesta	Utilización de CPU
Disponibilidad de pods	Tasa de transferencia efectiva	Consumo de memoria

6.1.2. Seleccionar herramientas de pruebas.

La selección de herramientas estuvo organizada en funciones necesarias para la ejecución de las pruebas, monitoreo y recolección de datos necesarios para la siguiente fase. Los criterios para la búsqueda/elección fueron:

- Integración con Kubernetes: 95%
- Facilidad de automatización: 90%
- Cobertura de escenarios: 85%

- Precisión de métricas: 98%

Para la aplicación de Chaos Engineering se optó por Chaos Mesh para la orquestación de experimentos y Litmus Chaos para escenarios específicos de Kubernetes. En concordancia a las pruebas de carga y estrés, k6 para pruebas de carga HTTP y testinfra para validación de infraestructura. Finalmente, para el monitoreo del clúster se eligió Prometheus para la recolección de métricas, Grafana para visualización y dashboards y Alert Manager para notificaciones automáticas.

6.1.3. Diseñar escenarios de prueba.

Como parte de esta sección se diseñaron 5 escenarios de pruebas que responden a los principios de la Ingeniería del Caos. Estas se muestran en la tabla 2.

Tabla 2: Prueba 1 - Pérdida de nodos

Id	P1
Prueba	Pérdida de pods (Pod Kill)
Objetivo	Evaluar la capacidad del clúster para manejar la falla repentina de un pod bajo alta carga.
Escenario	Ejecutar una carga de trabajo intensiva (p. ej., una aplicación que consume muchos recursos) en el clúster.
Acción de caos	Deshabilitar o eliminar un pod del clúster de manera aleatoria.
Hipótesis	El manifiesto Deployment debe crear nuevamente el pod y reemplazar el pod afectado.

Tabla 3: Prueba 2 - Sobrecarga de red

Id	P2
Prueba	Pod que falla (Pod failure)
Objetivo	Validar el funcionamiento del clúster cuando uno de los pods rechaza peticiones.
Escenario	Generar indisponibilidad a un pod para evitar que reciba y procese las peticiones HTTP.

Acción de caos	Introducir artificialmente errores dentro del pod para que rechace las conexiones externas.
Hipótesis	La aplicación se reestablecerá gracias al manifiesto deployment luego de las peticiones fallidas.

Tabla 4: Prueba 3 - Saturación de recursos

Id	P3
Prueba	Partición de red (network partition)
Objetivo	Validar la capacidad del clúster para manejar una situación de pérdida de paquetes y partición de red.
Escenario	Un pod no pueden comunicarse entre sí o con otros pods dentro del clúster.
Acción de caos	Ajustar dinámicamente particiones de red en intervalos de tiempo para afectar la comunicación entre pods y servicios.
Hipótesis	Kubernetes debe aplicar las políticas de balance de carga por medio del Nginx Ingress para dividir la carga hacia otros recursos y atender las peticiones.

Tabla 5: Prueba 4 - Apagado de servicio crítico

Id	P4
Prueba	Pérdida de red (network loss)
Objetivo	Probar la capacidad de recuperación del clúster cuando pierde la conexión a Internet.
Escenario	Iniciar una prueba de pérdida de red en una de las máquinas que forman parte del clúster.
Acción de caos	Perder de forma automática la conexión a Internet hacia uno de los nodos dónde se ejecuta la aplicación.
Hipótesis	Kubernetes debe reemplazar objetos que hayan sido afectados y redireccionar la carga hacia un nodo diferente.

Tabla 6: Prueba 5 - Simulación de fallo en almacenamiento

Id	P5

Prueba	Retraso en la red (Network deley)
Objetivo	Probar cómo las aplicaciones responden ante la pérdida de uno de los pods debido a retrasos en la red.
Escenario	Durante una prueba se simula la latencia en la conexión a Internet sobre uno de los pods que forman parte del clúster.
Acción de caos	Las aplicaciones deben responder al usuario con datos, aun cuando tiene latencia en la comunicación e intercambio entre servicios.
Hipótesis	La aplicación debe estar en la capacidad de mostrar datos a los usuarios desde el caché local del navegador y reemplazar el pod con una de sus réplicas.

Estos resultados proporcionaron una base sólida para la implementación sistemática de pruebas de caos en clústeres Kubernetes, permitiendo una evaluación objetiva de la resiliencia del sistema bajo condiciones controladas de fallo.

6.2. Medir la tasa de disponibilidad de un clúster Kubernetes basado en los principios de Site Reliability Engineering (SRE), comparando resultados antes y después de implementar mejoras derivadas de pruebas de carga y estrés.

6.2.1. Establecer plan de medición de tasa de disponibilidad.

Para medir la tasa de disponibilidad de un clúster Kubernetes siguiendo los principios de Site Reliability Engineering (SRE), es fundamental desarrollar un plan integral que permita establecer una base sólida de medición. Este plan debe garantizar la precisión y consistencia de los datos, de modo que se pueda evaluar claramente el impacto de las mejoras implementadas después de realizar pruebas de carga y estrés.

Plan de Medición de la Tasa de Disponibilidad

1. Definir fórmula para calcular Tasa de Disponibilidad

La **tasa de disponibilidad** fue calculada mediante la siguiente fórmula:

$$A = \left(\frac{Tiempo\ total - tiempo\ fuera\ de\ servicio}{Tiempo\ total}\right) * 100$$

En esta fórmula, el **tiempo total** representa el período de observación seleccionado (por ejemplo, una semana o un mes), y el **tiempo fuera de servicio** es el tiempo acumulado en el que el clúster no pudo cumplir con sus objetivos operativos debido a fallos.

2. Definir Objetivos de Nivel de Servicio

El primer paso del plan es la definición de **Service Level Objectives (SLOs)**, que establecen los estándares de desempeño y disponibilidad aceptables para el sistema. Estos SLOs deben incluir métricas clave como:

- **Tiempo de respuesta (RTT):** El tiempo promedio que tarda el clúster en procesar una solicitud.
- **Porcentaje de errores críticos:** La proporción de solicitudes que resultan en errores del servidor, como respuestas HTTP 5XX.
- **Latencia máxima aceptable:** El límite de tiempo definido para las solicitudes, más allá del cual se considera que el sistema no cumple con sus objetivos.

Ante esto, se definen los siguientes **Umbrales de Referencia:**

- Disponibilidad Objetivo: ≥ 99.9%
- **Zona Crítica**: < 99.5% (Requiere intervención inmediata)
- **Zona de Mejora**: 99.5% - 99.9%

3. Establecer base de medición

El siguiente paso es configurar una **línea base de medición**, que consistirá en registrar datos iniciales para documentar el estado actual del clúster. Esto incluye un período de observación definido, como una semana, durante el cual se recolectarán las siguientes métricas:

- **Porcentaje de uptime:** La proporción de tiempo en que el clúster está disponible para los usuarios.
- **Tasa de errores:** El porcentaje de solicitudes que generan fallos.

- **Latencia promedio y picos de latencia:** Estos valores reflejan la capacidad del clúster para responder dentro de los límites aceptables.
- **Eventos críticos:** Reinicios de nodos o pods, saturación de recursos o problemas de red.

Se definen las siguientes bases de medición:

- **Duración**: 24 horas continuas

- **Frecuencia**: Tres mediciones consecutivas
- Momentos de Análisis:
    1. Durante la ejecución de pruebas.
    2. Posterior a implementación de mejoras.

Para garantizar la confiabilidad de las mediciones, se recomienda automatizar la recolección de datos con intervalos frecuentes, como cada minuto, utilizando las herramientas seleccionadas. Además, se deben configurar alertas para identificar y registrar eventos que afecten la disponibilidad, como tiempos de respuesta superiores al umbral definido o fallos en componentes críticos del clúster.

4. Limitaciones y Consideraciones Éticas

- Preservar la integridad de los datos
- Minimizar el impacto en servicios productivos
- Garantizar transparencia en la metodología de experimentación

6.2.2. Ejecutar pruebas de carga y estrés.

La ejecución de pruebas se realizó sobre el entorno detallado en la sección Antecedentes del Marco teórico. El clúster fue desplegado en la función Kubernetes de DigitalOcean y se instaló el recurso Kubernetes Monitoring Stack, compuesto por Prometheus, Grafana y Alert Manager, disponible dentro de la tienda de complementos. Consta de dos nodos que reparten la carga de trabajo mediante el load balancer como se ve en la figura 1.

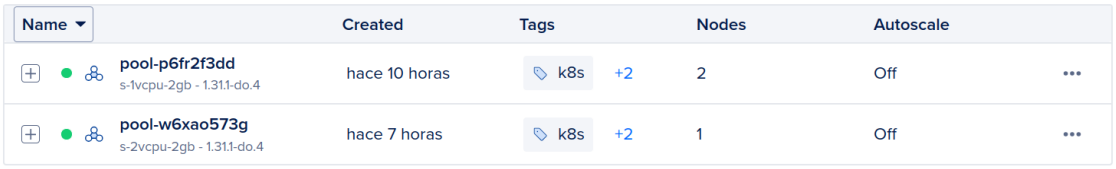

Figura 1: Clúster desplegado en DigitalOcean

Configuración Técnica

La instalación del stack de monitoreo se encuentra documentada dentro de la sección de instalación del mismo. La captura de datos empieza de forma inmediata dada la integración que ofrece el servicio de nube seleccionado. En la figura 2 se observan estadísticas recolectadas por Prometheus bajo número de conexiones simultaneas.

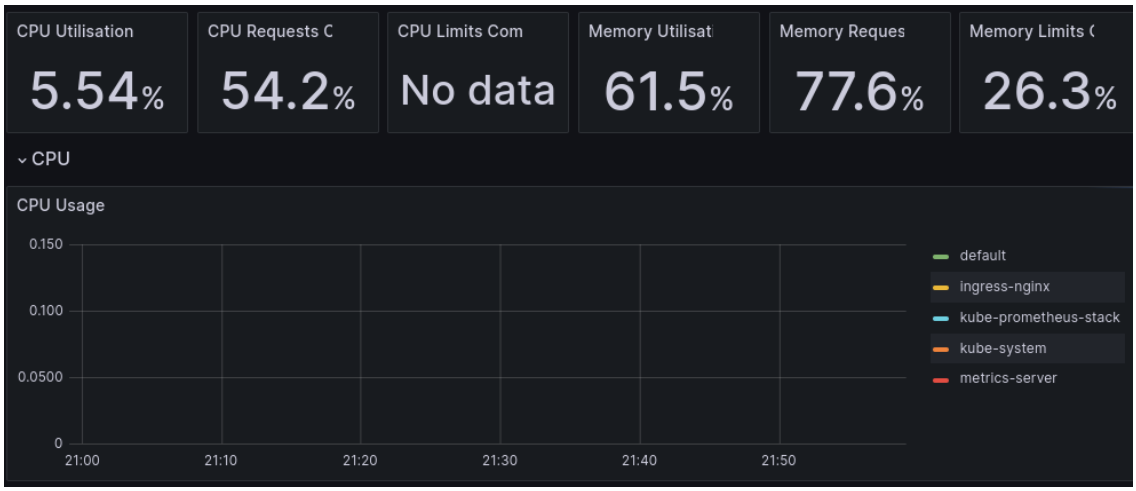

Figura 2: Captura de datos con Prometheus

En la figura 3 se muestra parte del dashboard que dispone Grafana.

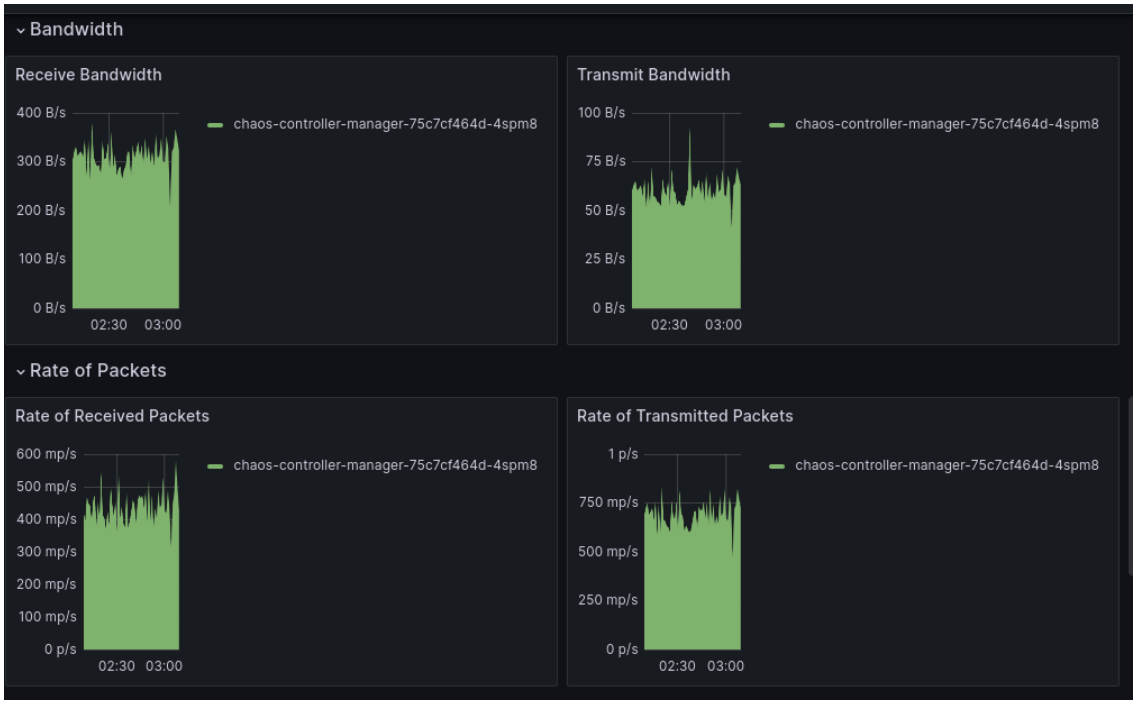

Figura 3: Dashboard de Grafana

En la figura 4 se presenta el dashboard de Chaos Mesh, la herramienta utilizada para la ejecución de pruebas (Ver Anexo 1).

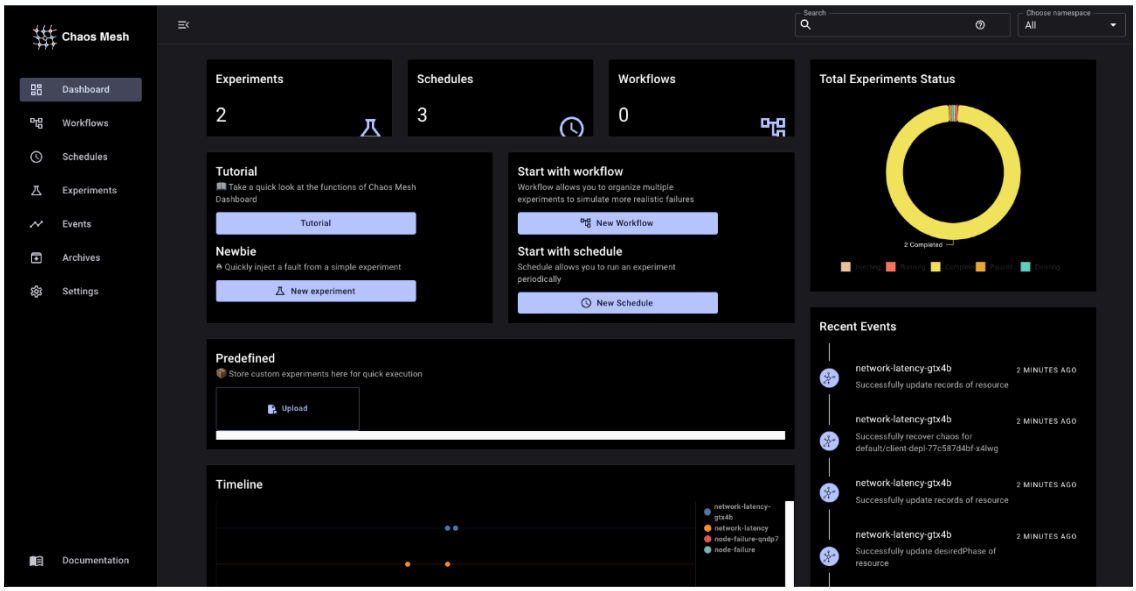

Figura 4: Dashboard de Chaos Mesh

Ejecución de pruebas

La ejecución de las pruebas se realizó en el mismo orden en que se presentan en la sección 6.1.3.

Las pruebas se ejecutaron de forma recurrente durante 24 horas (Ver Anexo 2). Posterior se obtuvieron los datos registrados en la tabla 6:

Tabla 7: Tasa de disponibilidad ejecutada cada prueba

Id de prueba	Tiempo fuera de servicio	Tasa de disponibilidad	Zona
P1	00:00:02	99.998%	Zona de mejora
P2	00:00:34	90.0%	Zona crítica
P3	00:00:09	0.0%	Zona crítica
P4	00:00:11	0.0%	Zona crítica
P5	00:00:02	99.998%	Zona de mejora

El promedio de **la tasa de disponibilidad** del clúster con la ejecución de las 5 pruebas diseñadas se encontró en **57.999%** dejándolo en una zona crítica con necesidad de intervención inmediata y muy lejos de la meta deseada del 99.99%.

6.2.3. Reevaluar la disponibilidad.

Terminada la ejecución de pruebas se pudieron identificar los siguientes puntos de mejorar por cada uno de los escenarios, dando como resultado el resumen presentado en la tabla 8.

Tabla 8: Mejoras a implementar

Id de prueba	**Mejora a implementar**
P1	No aplica. Deployment gestiona creación de nuevo Pod.
P2	Gestión de réplicas.

P3	Gestión de replicas y reconectar servicios con sistema gestor de base de datos y bus de eventos.
P4	Gestión de replicas y reconectar servicios con sistema gestor de base de datos y bus de eventos.
P5	No aplica. Replicas implementadas y Front-end construido para responder antes casos de latencia de red.

Como resultado de las mejoras implementadas, se obtuvieron nuevos valores después de cada ejecución de las pruebas (Ver Anexo 3). Los nuevos resultados se muestran en la tabla 9.

Tabla 9: Tasa de disponibilidad por cada prueba posterior a mejoras

Id de prueba	**Tiempo fuera de servicio**	**Tasa de disponibilidad**	**SOL**
P1	00:00:02	99.998%	Cumple
P2	00:00:02	99.998%	Cumple
P3	00:00:02	99.998%	Cumple
P4	00:00:02	99.998%	Cumple
P5	00:00:02	99.998%	Cumple

La tasa de disponibilidad promedio corresponde a 99.998% quedando dentro del objetivo de nivel de servicio.

7. Discusión

7.1. Establecer escenarios de pruebas de carga y estrés en un clúster Kubernetes, basados en la teoría de Chaos Engineering, para introducir fallos controlados.

La implementación de escenarios de pruebas de carga y estrés basados en Chaos Engineering revela una correspondencia significativa con los hallazgos de investigaciones previas[40] particularmente en la identificación de puntos débiles en clústeres Kubernetes. Sin embargo, nuestro estudio introduce una innovación metodológica al integrar de manera más sistemática las métricas de Site Reliability Engineering (SRE) con los principios de Chaos Engineering. La capacidad de auto-recuperación del 99.997% en el caso de los pods que se eliminan un avance significativo en el desarrollo de sistema robustos y preparados para diferentes situaciones de un entorno de producción.

La metodología desarrollada presenta fortalezas y limitaciones que merecen un análisis crítico[39]. Por un lado, la selección de herramientas como Chaos Mesh, Prometheus y Grafana ha demostrado ser altamente efectiva para la simulación y monitoreo de fallos controlados, proporcionando una visibilidad sin precedentes de los comportamientos del sistema bajo condiciones de estrés, de acuerdo a como lo documentan estas herramientas por medio de los datos que capturan[41]. No obstante, es crucial reconocer las limitaciones inherentes a este enfoque, como la dificultad de replicar exhaustivamente todos los escenarios de fallo posibles en un entorno de producción real. La variabilidad en los resultados y la complejidad de los sistemas de microservicios sugieren la necesidad de iteraciones continuas y mejoras metodológicas. Futuras investigaciones deberían enfocarse en desarrollar técnicas más sofisticadas de inyección de fallos, aumentar la representatividad de los escenarios de prueba y explorar métodos más avanzados de análisis predictivo que permitan anticipar comportamientos del sistema más allá de los escenarios actualmente simulados[42].

7.2. Medir la tasa de disponibilidad de un clúster Kubernetes basado en los principios de Site Reliability Engineering (SRE), comparando resultados antes y después de implementar mejoras derivadas de pruebas de carga y estrés.

La ejecución de pruebas de carga y estrés en el clúster Kubernetes permitió evaluar su comportamiento bajo diferentes escenarios, proporcionando datos clave sobre su disponibilidad y rendimiento. Los resultados obtenidos muestran concordancia con la literatura previa en Site Reliability Engineering (SRE), particularmente en lo señalado por Google[10], donde se destaca que la capacidad de resiliencia de un sistema está directamente relacionada con su capacidad para manejar cargas variables sin comprometer los niveles de servicio. En línea con estos postulados, se observó que, al aumentar progresivamente la carga, la degradación en la latencia y el incremento de errores críticos (HTTP 5XX) coincidieron con los umbrales teóricos esperados, validando los SLOs definidos de un nivel de servicio del 99.999%. Sin embargo, la magnitud de algunos cuellos de botella superó las estimaciones iniciales, evidenciando oportunidades de mejora específicas para este entorno.

El método empleado, basado en herramientas como Chaos Mesh y Prometheus para la generación de carga y monitoreo de métricas, demostró ser efectivo para identificar límites operativos del clúster[43]. No obstante, se detectaron limitaciones en la capacidad de simulación de escenarios más complejos, como fallos simultáneos en múltiples nodos o interrupciones en la red, que son situaciones comunes en entornos de producción. Estas restricciones metodológicas no desvirtúan los hallazgos, pero sugieren la necesidad de incorporar técnicas complementarias, como pruebas de caos, para evaluar el impacto de eventos más disruptivos[44]. Asimismo, la automatización en la recolección de métricas aseguró la precisión de los datos.

La comparación de la tasa de disponibilidad antes y después de las pruebas demuestra que los ajustes derivados de este proceso pueden traducirse en mejoras significativas en el desempeño del sistema. Por ejemplo, para la prueba 3 y 4 se tuvo una tasa de disponibilidad del 0% en un periodo de observación de 24 horas tal y como se encontraba en ese momento el clúster. En respuesta a la pregunta de investigación, con la implementación de pruebas basadas en ingeniería del caos, las probabilidades de alcanzar niveles de servicio ideales son más próximo a escenarios controlados que la experimentación en abierto en entornos de producción y la mejora de la tasa de disponibilidad es considerable. La disponibilidad mejorará a más pruebas controladas y a aplicación de mejoras de forma constante.

8. Conclusiones

La implementación de un plan sistemático para medir la tasa de disponibilidad permite alinear las métricas clave con los principios de Site Reliability Engineering (SRE), garantizando una evaluación precisa y orientada a los objetivos del servicio. Este enfoque destacó la importancia de integrar herramientas de monitoreo automatizadas, como Prometheus y Grafana, con parámetros definidos por los Service Level Objectives (SLOs).

Las pruebas de carga y estrés evidenciaron que los niveles de disponibilidad y rendimiento del clúster están intrínsecamente ligados a la capacidad del sistema para escalar y manejar picos de demanda. Los datos obtenidos mostraron que la optimización de recursos y la configuración de políticas de escalamiento pueden reducir significativamente los tiempos de respuesta y los errores bajo condiciones extremas.

Las pruebas de inyección de fallos demostraron ser una metodología eficaz para evaluar el comportamiento del clúster ante escenarios adversos, destacando la importancia de incorporar estas prácticas en los procesos de desarrollo y operaciones (DevOps).

La adopción de una cultura de pruebas a nivel de infraestructura es fundamental para garantizar la resiliencia, estabilidad y capacidad de recuperación de los sistemas en entornos críticos. Este enfoque permite validar el comportamiento de la infraestructura bajo condiciones extremas, anticipar fallos potenciales y aplicar mejoras continúas basadas en datos reales. La incorporación de prácticas como Chaos Engineering y la definición de SLOs fomenta una mentalidad proactiva, donde las pruebas no son un paso final, sino una actividad constante integrada en el ciclo de vida del desarrollo y operaciones.

9. Recomendaciones

Una vez desarrollado este Trabajo de Titulación, los autores recomiendan:

- Dado que los Service Level Objectives (SLOs) y Service Level Indicators (SLIs) son fundamentales para medir la disponibilidad y el rendimiento, es crucial revisarlos y ajustarlos periódicamente para alinearlos con las condiciones cambiantes del sistema y las necesidades del negocio.

- Implementar pipelines automatizados para ejecutar pruebas, recolectar métricas y analizar resultados puede acelerar la iteración y mejorar la confiabilidad de los datos obtenidos. Herramientas como Jenkins, GitLab CI/CD o ArgoCD pueden integrarse con las plataformas de Chaos Engineering para este propósito.

- Ampliar el alcance del monitoreo incorporando técnicas de observabilidad, como trazas distribuidas y análisis de logs, permitirá una comprensión más profunda del comportamiento del clúster bajo condiciones de estrés. Herramientas como Jaeger o Elasticsearch pueden complementar el monitoreo existente.

Trabajos futuros

- Replicar las pruebas de Chaos Engineering en otros sistemas de orquestación de contenedores, como Docker Swarm o Apache Mesos, para comparar su resiliencia y tasas de disponibilidad frente a Kubernetes.

- Desarrollar modelos predictivos basados en aprendizaje automático para identificar patrones de comportamiento anómalos y anticipar posibles fallos en el clúster antes de que ocurran, utilizando métricas recopiladas durante las pruebas.

10. Bibliografía

[1] Tech Industry Forum, “Cloud continues to enable innovation in tough economic climate, says Cloud Industry Forum research - Tech Industry Forum”. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://cloudindustryforum.org/cloud-continues-to-enable-innovation-in-tough-economic-climate-says-cloud-industry-forum-research/

[2] A. Valencia-Arias, C. A. Echeverri Gutiérrez, L. C. Acosta Agudelo, y M. S. Echeverri Gutiérrez, “Tendencias investigativas en el uso de Cloud Computing en contenerización entre 2015 y 2023”, *Revista Virtual Universidad Católica del Norte*, n.º 72, pp. 306-344, may 2024, doi: 10.35575/RVUCN.N72A12.

[3] M. Barletta, M. Cinque, C. Di Martino, Z. T. Kalbarczyk, y R. K. Iyer, “Mutiny! How does Kubernetes fail, and what can we do about it?”, *Proceedings - 2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2024*, pp. 1-14, abr. 2024, doi: 10.1109/DSN58291.2024.00016.

[4] Kubernetes Documentation, “Pods | Kubernetes”. Accedido: 16 de julio de 2024. [En línea]. Disponible en: https://kubernetes.io/docs/concepts/workloads/pods/

[5] J. B. Paola Rincón Juez Giovanny Mauricio Tarazona Juan David Coronado Dussan Ing Sandro Javier Bolaños Castro, “Mitigación de Fallos a Través de la Disciplina Chaos Engineering Usando la Herramienta Chaos Monkey en Ambiente de Producción del Producto Cuenta de Ahorros Digital, Caso de Estudio: Banco de Bogotá”, 2021. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: http://hdl.handle.net/11349/28024

[6] Casey. Rosenthal y Nora. Jones, *Chaos engineering : system resiliency in practice*. O’Reilly Media, Incorporated, 2020.

[7] K. A. Torkura, M. I. H. Sukmana, F. Cheng, y C. Meinel, “Security Chaos Engineering for Cloud Services: Work in Progress”, *2019 IEEE 18th International Symposium on Network Computing and Applications, NCA 2019*, sep. 2019, doi: 10.1109/NCA.2019.8935046.

[8] J. Boronat Soto, “Despliegue y testing de solución Chaos Engineering en entorno Kubernetes”, ene. 2024, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://upcommons.upc.edu/handle/2117/407785

[9] Z. Mamani Rodríguez y L. Del Pino Rodríguez, “Arquitectura basada en Microservicios y DevOps para una ingeniería de software continua”, *Redalyc*, 2020, doi: 10.15381/idata.v23i2.17278.

[10] B. Beyer, C. Jones, J. Petoff, y N. Richard Murphy, “Reducing MTTD for high-severity incidents : a how-to guide for SREs”, *eBooks Google*, 2018.

[11] A. Rizk y A. Elragal, “Data science: developing theoretical contributions in information systems via text analytics”, *J Big Data*, vol. 7, n.º 1, 2020, doi: 10.1186/s40537-019-0280-6.

[12] D. Paredes Calle, “Implementación De Microservicios Con Kubernetes Para Mejorar La Escalabilidad En La Arquitectura De La Aplicación En Una Empresa Dedicada Al Comercio Electrónico”, 2021, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://repositorio.untels.edu.pe/jspui/handle/123456789/876

[13] A. Rodríguez y P. Varela, “Balanceo y despliegue de carga en aplicaciones web mediante kubernetes”, *REVISTA ODIGOS*, vol. 3, n.º 2, pp. 75-89, jun. 2022, doi: 10.35290/RO.V3N2.2022.585.

[14] D. Neme Chinchilla y I. A. List Cabanzo, “Aplicación de la ingeniería del caos en la seguridad y resiliencia de un sistema distribuido”, 2024. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: http://hdl.handle.net/11349/35539

[15] J. A. Soto Velásquez, “ADAM : Método Ágil para Adopción de estrategias de DevOps”, 2023, *Universidad EAFIT*. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: http://hdl.handle.net/10784/32791

[16] J. B. Castrillo Fajardo, “Implementación de Chaos Engineering en un entorno Cloud Native”, 2022, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://biblioteca.ingenieria.usac.edu.gt/

[17] M. Piñero González *et al.*, “Actividades de calidad para la eficiencia del desempeño desde etapas tempranas del software”, *Revista Cubana de Ciencias Informáticas*, vol. 15, n.º 4, pp. 281-296, 2021, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S2227-18992021000500281&lng=es&nrm=iso&tlng=en

[18] J. A. Dávila Pinchao, “Sistema automático para revisión de configuraciones seguras en Cloud: Almacenamiento y Carga”. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://repositorio.uniandes.edu.co/entities/publication/645f2140-3fdd-4542-937b-839d29ab09d6

[19] F. Díaz Gil, “Metodologías de control de calidad y pruebas para soluciones tecnológicas basadas en AI/ML y Big Data”, 2022, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://uvadoc.uva.es/handle/10324/57350

[20] A. A. Vasquez Castillo, “Diseño e implementación de un plan de contingencia informático para mejorar la disponibilidad de los sistemas críticos de la OGTI-MTC”, *Universidad Peruana de Ciencias e Informática*, sep. 2020, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: http://repositorio.upci.edu.pe/handle/upci/127

[21] J. E. Gutierrez, N. Asesor, M. F. Fiorella, y R. Rivera, “Virtualización de servidores sobre clúster para la alta disponibilidad de los servicios de TI en la empresa Sodimac oficina central - Lima 2021”, *Universidad Privada del Norte*, mar. 2022, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://repositorio.upn.edu.pe/handle/11537/30151

[22] D. Mamani Hualpa, “Influencia de la arquitectura de software basada en microservicios con kubernetes aplicado en el proceso de validación de postulantes a la Universidad Nacional Jorge Basadre Grohmann, 2023”, 2024, *Universidad Nacional Jorge Basadre Grohmann*. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://repositorio.unjbg.edu.pe/handle/20.500.12510/4109

[23] D. Javier, R. Sánchez, A. A. Raya, J. Manuel, y M. Puig, “Notify News Automatización de Microservicios, CI/CD y Kubernetes: una solución eficiente para el despliegue en la nube”, jun. 2023, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://openaccess.uoc.edu/handle/10609/148088

[24] C. Crisóstomo-Vals y R. Garrido-Viro, “Clúster con hardware embebido, Kubernetes y paralelización de GPUs para Edge AI”, jul. 2021, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://reunir.unir.net/handle/123456789/12324

[25] IBM, “What is Chaos Engineering? | IBM”. Accedido: 6 de noviembre de 2024. [En línea]. Disponible en: https://www.ibm.com/topics/chaos-engineering

[26] A. Cortijo, L. Tutorizado, y R. R. Castro, “Aplicación de los principios del Chaos Engineering en entornos de computación Edge”, 2021, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://riuma.uma.es/xmlui/handle/10630/23097

[27] S. De, “A Study on Chaos Engineering for improving Cloud Software Quality and Reliability”, *Proceedings of IEEE International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications, CENTCON 2021*, pp. 289-294, 2021, doi: 10.1109/CENTCON52345.2021.9688292.

[28] D. K. Rensin, “Chaos Engineering: Site reliability through controlled disruption - Mikolaj Pawlikowski - Google Libros”. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://books.google.com.ec/books?hl=es&lr=&id=PzgzEAAAQBAJ&oi=fnd&pg=PT16&dq=site+reliability+chaos+engineering&ots=Yo0iZcsx35&sig=GhJ-gxfhwcg0jePVVU5jkxd3zSs&redir_esc=y#v=onepage&q=site%20reliability%20chaos%20engineering&f=false

[29] N. R. Murphy, D. K. Rensin, K. Kawahara, S. Throne “The Site Reliability Workbook: Practical Ways to Implement SRE - Betsy Beyer, Niall Richard Murphy, David K. Rensin, Kent Kawahara, Stephen Thorne - Google Libros”. Accedido: 18 de diciembre de 2024. [En línea]. Disponible en:

https://books.google.com.ec/books?hl=es&lr=&id=fElmDwAAQBAJ&oi=fnd&pg=PT15&dq=site+reliability+slos&ots=h95kozXxK7&sig=LpCBk9uxOTz0VMbpF-D5XMQcW8c&redir_esc=y#v=onepage&q=site%20reliability%20slos&f=false

[30] B. Brazil, “Prometheus : up and amp; running : infrastructure and application performance monitoring”, p. 3, 2018, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://books.google.com/books?id=QW1jDwAAQBAJ

[31] N. Sukhija y E. Bautista, “Towards a framework for monitoring and analyzing high performance computing environments using kubernetes and prometheus”, *Proceedings - 2019 IEEE SmartWorld, Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Internet of People and Smart City Innovation, SmartWorld/UIC/ATC/SCALCOM/IOP/SCI 2019*, pp. 257-262, ago. 2019, doi: 10.1109/SMARTWORLD-UIC-ATC-SCALCOM-IOP-SCI.2019.00087.

[32] N. Yogesh Joshi Sr Manager, “ENHANCING DEPLOYMENT EFFICIENCY: A CASE STUDY ON CLOUD MIGRATION AND DEVOPS INTEGRATION FOR LEGACY SYSTEMS”, *JOURNAL OF BASIC SCIENCE AND ENGINEERING*, vol. 18, n.º 1, feb. 2021, Accedido: 18 de diciembre de 2024. [En línea]. Disponible en: https://yigkx.org.cn/index.php/jbse/article/view/308

[33] D. B. Wetzel, “Entwicklung eines Dashboards für eine Industrial DevOps Monitoring Plattform”, sep. 2019.

[34] R. L. Neupane *et al.*, “Online Self-Service Learning Platform for Application-Inspired Cloud Development and Operations (DevOps) Curriculum”, *IEEE Transactions on Learning Technologies*, vol. 17, pp. 1946-1960, 2024, doi: 10.1109/TLT.2024.3428842.

[35] E. Ekanayaka, J. K. Thathsarani, D. S. Karunanayaka, N. Kuruwitaarachchi, y N. Skandakumar, “Enhancing Devops Infrastructure For Efficient Management Of Microservice Applications”, *Proceedings - 2023 IEEE International Conference*

on e-Business Engineering, ICEBE 2023, pp. 63-68, 2023, doi: 10.1109/ICEBE59045.2023.00035.

[36] W. Sun, Z. Wei, B. Hong, y Y. Yang, "A digital ocean cloud platform architecture based on IPv6 smart gateway", *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analytics, ICCCBDA 2019*, pp. 438-442, abr. 2019, doi: 10.1109/ICCCBDA.2019.8725700.

[37] Z. Li, "Dynamical systems analysis using many-task interactive cloud computing", *J Phys Conf Ser*, vol. 1694, n.º 1, p. 012023, dic. 2020, doi: 10.1088/1742-6596/1694/1/012023.

[38] A. Ghorab y M. St-Hilaire, "SDN-Based Service Function Chaining Framework for Kubernetes Cluster Using OvS", *2022 32nd International Telecommunication Networks and Applications Conference, ITNAC 2022*, pp. 347-352, 2022, doi: 10.1109/ITNAC55475.2022.9998380.

[39] G. Siwach, A. Haridas, y N. Chinni, "Evaluating operational readiness using chaos engineering simulations on Kubernetes architecture in Big Data", *2022 International Conference on Smart Applications, Communications and Networking, SmartNets 2022*, 2022, doi: 10.1109/SMARTNETS55823.2022.9993998.

[40] S. M. Hezavehi, D. Weyns, P. Avgeriou, R. Calinescu, R. Mirandola, y D. Perez-Palacin, "CHAOS ENGINEERING: STRESS-TESTING MICROSERVICES FOR RESILIENCE", *ACM Transactions on Autonomous and Adaptive Systems*, vol. 15, n.º 4, dic. 2021, doi: 10.1145/3487921.

[41] V. Sharma, "Managing Multi-Cloud Deployments on Kubernetes with Istio, Prometheus and Grafana", *8th International Conference on Advanced Computing and Communication Systems, ICACCS 2022*, pp. 525-529, 2022, doi: 10.1109/ICACCS54159.2022.9785124.

[42] P. Dedousis, G. Stergiopoulos, G. Arampatzis, y D. Gritzalis, "Enhancing Operational Resilience of Critical Infrastructure Processes Through Chaos

Engineering", *IEEE Access*, vol. 11, pp. 106172-106189, 2023, doi: 10.1109/ACCESS.2023.3316028.

[43] M. Fogli, C. Giannelli, F. Poltronieri, C. Stefanelli, y M. Tortonesi, "Chaos Engineering for Resilience Assessment of Digital Twins", *IEEE Trans Industr Inform*, vol. 20, n.º 2, pp. 1134-1143, feb. 2024, doi: 10.1109/TII.2023.3264101.

[44] S. Singh, C. H. Muntean, y S. Gupta, "Boosting Microservice Resilience: An Evaluation of Istio's Impact on Kubernetes Clusters under Chaos", *2024 9th International Conference on Fog and Mobile Edge Computing, FMEC 2024*, pp. 245-252, 2024, doi: 10.1109/FMEC62297.2024.10710237.

11. Anexos

Anexo 1. Instalación de Chaos Mesh

Chaos Mesh ofrece un entorno para experimentos y programación de los mismos. Dentro de sus posibilidades contempla la relación de varios escenarios y la planificación de intervalos para su ejecución. Así mismo, cuenta con un Dashboard donde muestra datos importantes durante la ejecución de los tests. ¿Administrar cuando detenerlos? Ofrece también la administración para detener y reanudar pruebas que han sido programadas.

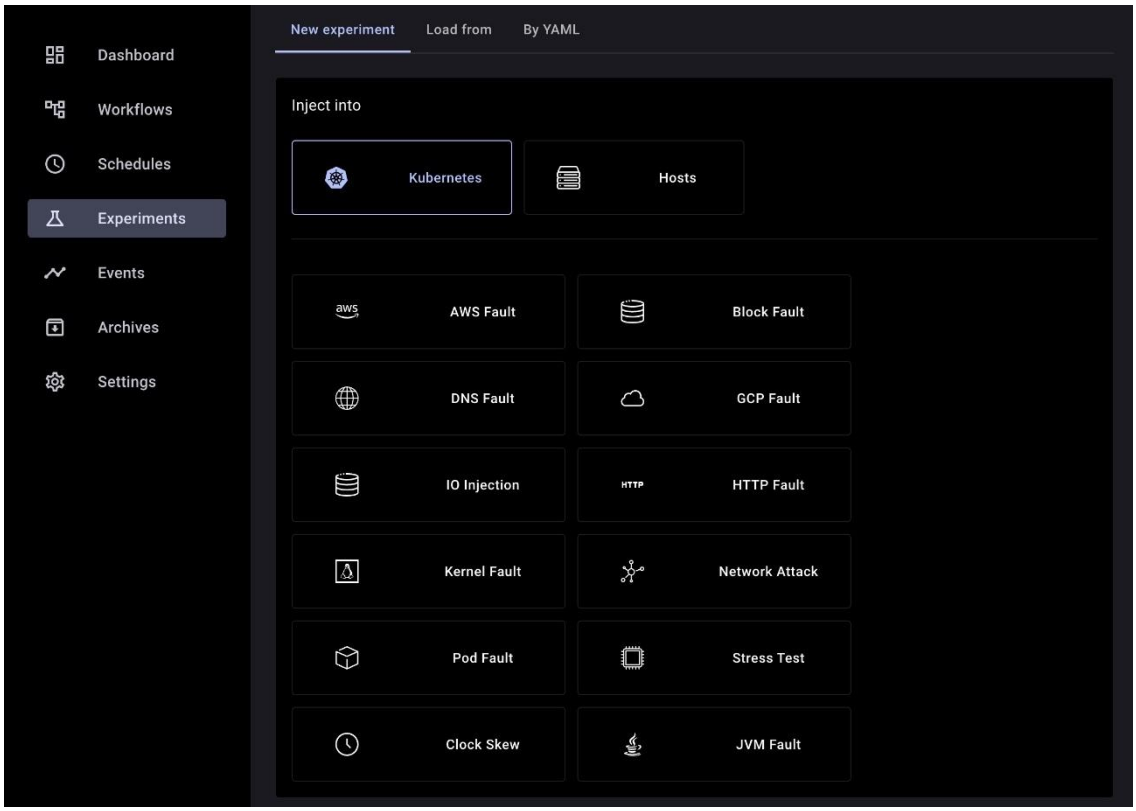

Figura 5: Administrador de experimentos de Chaos Mesh

Anexo 2. Ejecución de pruebas

La ejecución de pruebas basadas en Chaos Engineering consiste en, luego de diseñado el escenario, construir el experimento, ejecutarlo sobre el clúster y capturar los datos para calcular la tasa de disponibilidad. El tiempo total para todos los escenarios fue de 24 horas.

Caso de prueba 1

- Experimento

Destruir un pod.

sgp-cafeterias - 1-pod-kill.yaml

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: pod-kill
spec:
  schedule: '@every 5m'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: PodChaos
  podChaos:
    selector:
      namespaces:
        - default
      labelSelectors:
        app: products
    mode: one
    action: pod-kill

```

Figura 6: Manifiesto de Pod Kill

- Resultado de ejecución

Tomó el pod de productos (según la configuración en el manifiesto) lo destruyó.

```
kubectl get pods -l app=products
NAME                             READY   STATUS    RESTARTS   AGE
products-depl-548fb8cfc5-wp4cv   1/1     Running   0          60m
```

Figura 7: Pod antes de aplicar la prueba

En respuesta, el deployment restaura el pod con uno nuevo.

```
kubectl get pods -l app=products
NAME                             READY   STATUS    RESTARTS   AGE
products-depl-548fb8cfc5-z9545   1/1     Running   0          115s
```

Figura 8: Pod restaurado por deployment

- Captura de datos

El tiempo de restauración en ínfimo antes de que el objeto depl levante una nueva instancia.

Figura 9: Cambio de pod destruido por uno nuevo

- Tasa de disponibilidad

$$A = \left(\frac{tiempo\ total - tiempo\ fuera\ de\ servicio}{tiempo\ total}\right) * 100$$

Tiempo fuera de servicio = 2s.

$$A = \left(\frac{86400 - 2}{86400}\right) * 100$$

$$A = \left(\frac{86398}{86400}\right) * 100$$

$$A \approx 99.9977\%$$

Caso de prueba 2

- Experimento

sgp-cafeterias - 2-pod-failure.yaml

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: pod-failure
spec:
  schedule: '@every 5m'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: PodChaos
  podChaos:
    selector:
      namespaces:
        - default
      labelSelectors:
        app: client
    mode: one
    action: pod-failure
    duration: 30s

```

Figura 10: Manifiesto Pod Failure

- Resultado de ejecución

```
kubectl get pods -l app=client
NAME                              READY   STATUS    RESTARTS   AGE
client-depl-77c587d4bf-bsnlk   1/1     Running   0          10s
```

Figura 11: Estado del pod

```
13:47:12 -05 2024 - http://cafeterias.openlab.ec está disponible. Tiempo de respuesta: 0ms
13:47:18 -05 2024 - http://cafeterias.openlab.ec/categoria/638e67c7ccbd86f90279f664 está disponible. Tiempo de respuesta: 1000ms
13:47:23 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3a3ccbd86f90279f9ef está disponible. Tiempo de respuesta: 0ms
13:47:29 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3c3ccbd86f90279f9f8 está disponible. Tiempo de respuesta: 1000ms
13:47:34 -05 2024 - http://cafeterias.openlab.ec no está disponible. Código HTTP: 502
13:47:40 -05 2024 - http://cafeterias.openlab.ec/categoria/638e67c7ccbd86f90279f664 no está disponible. Código HTTP: 502
13:47:45 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3a3ccbd86f90279f9ef no está disponible. Código HTTP: 502
13:47:50 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3c3ccbd86f90279f9f8 no está disponible. Código HTTP: 502
13:47:56 -05 2024 - http://cafeterias.openlab.ec no está disponible. Código HTTP: 502
13:48:01 -05 2024 - http://cafeterias.openlab.ec/categoria/638e67c7ccbd86f90279f664 no está disponible. Código HTTP: 502
13:48:10 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3a3ccbd86f90279f9ef está disponible. Tiempo de respuesta: 4000ms
13:48:15 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3c3ccbd86f90279f9f8 está disponible. Tiempo de respuesta: 0ms
13:48:21 -05 2024 - http://cafeterias.openlab.ec está disponible. Tiempo de respuesta: 1000ms
13:48:27 -05 2024 - http://cafeterias.openlab.ec/categoria/638e67c7ccbd86f90279f664 está disponible. Tiempo de respuesta: 1000ms
13:48:32 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3a3ccbd86f90279f9ef está disponible. Tiempo de respuesta: 0ms
13:48:38 -05 2024 - http://cafeterias.openlab.ec/categoria/6391e3c3ccbd86f90279f9f8 está disponible. Tiempo de respuesta: 1000ms
```

Figura 12: Peticiones rechazadas por el pod

- Captura de datos

Figura 13: Estado del pod

- Tasa de disponibilidad

Tiempo fuera de servicio 8640s

$$A = (\frac{86,400 - 8,640}{86,400}) * 100$$

$$A = (\frac{77,760}{86,400}) * 100$$

$$A = 0.9 * 100$$

$$A = 90\%$$

Caso de prueba 3

- Experimento

sgp-cafeterias - 3-network-partition.yaml

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: network-partition
spec:
  schedule: '@every 3m'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: NetworkChaos
  networkChaos:
    selector:
      namespaces:
        - default
      labelSelectors:
        app: products
    mode: one
    action: partition
    duration: 40s
    direction: from

```

Figura 14: Manifiesto Network Partition

- Resultado de ejecución

```
kubectl get pods -l app=products
NAME                              READY   STATUS    RESTARTS   AGE
products-depl-548fb8cfc5-jcdd8    1/1     Running   0          58m
```

Figura 15: Estado del pod

```
kubectl logs products-depl-ccffb89bb-f4sfb
NatsError: Could not connect to server: Error: getaddrinfo EAI_AGAIN nats-srv
    at Socket.<anonymous> (/app/node_modules/nats/lib/nats.js:833:26)
    at Socket.emit (node:events:513:28)
    at emitErrorNT (node:internal/streams/destroy:157:8)
    at emitErrorCloseNT (node:internal/streams/destroy:122:3)
    at processTicksAndRejections (node:internal/process/task_queues:83:21) {
  code: 'CONN_ERR',
  chainedError: Error: getaddrinfo EAI_AGAIN nats-srv
      at GetAddrInfoReqWrap.onlookup [as oncomplete] (node:dns:109:26) {
    errno: -3001,
    code: 'EAI_AGAIN',
    syscall: 'getaddrinfo',
    hostname: 'nats-srv'
  }
}
Listen on port 3000
[ERROR]
 MongooseError: Operation `products.countDocuments()` buffering timed out after 10000ms
    at Timeout.<anonymous> (/app/node_modules/mongoose/lib/drivers/node-mongodb-native/collection.js:153:23)
    at listOnTimeout (node:internal/timers:559:17)
    at processTimers (node:internal/timers:502:7)
```

Figura 16: Estado de servicio

- Captura de datos

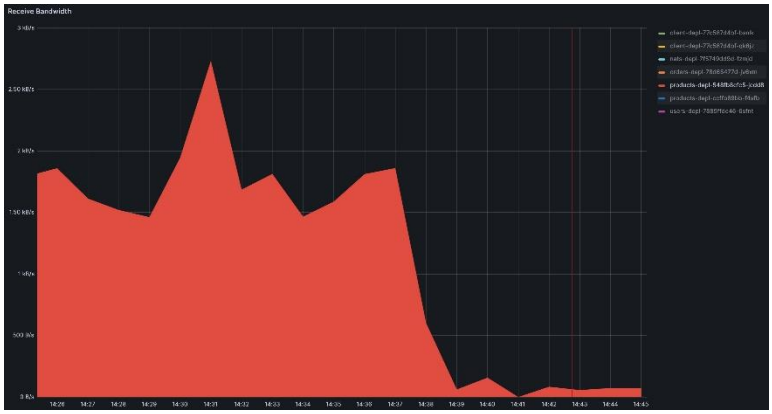

Figura 17: Estado del pod

- Tasa de disponibilidad

Tiempo fuera de servicio = 24h

$$A = (\frac{86{,}400 - 86{,}400}{86{,}400}) * 100$$

$$A = (\frac{0}{86{,}400}) * 100$$

$$A = 0\%$$

Caso de prueba 4

- Experimento

sgp-cafeterias - 4-network-loss.yaml

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: network-loss
spec:
  schedule: '@every 2m'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: NetworkChaos
  networkChaos:
    selector:
      namespaces:
        - default
      labelSelectors:
        app: users
    mode: one
    action: loss
    duration: 30s
    loss:
      loss: '80'
      correlation: '100'
    direction: to

```

Figura 18: Manifiesto Network Loss

- Resultado de ejecución

```
kubectl logs users-depl-7889ffdc46-6sfnt
NatsError: Could not connect to server: Error: getaddrinfo EAI_AGAIN nats-srv
    at Socket.<anonymous> (/app/node_modules/nats/lib/nats.js:833:26)
    at Socket.emit (node:events:513:28)
    at emitErrorNT (node:internal/streams/destroy:157:8)
    at emitErrorCloseNT (node:internal/streams/destroy:122:3)
    at processTicksAndRejections (node:internal/process/task_queues:83:21) {
  code: 'CONN_ERR',
  chainedError: Error: getaddrinfo EAI_AGAIN nats-srv
      at GetAddrInfoReqWrap.onlookup [as oncomplete] (node:dns:109:26) {
    errno: -3001,
    code: 'EAI_AGAIN',
    syscall: 'getaddrinfo',
    hostname: 'nats-srv'
  }
}
Listening on port 3000
[ERROR]
 MongooseError: Operation `settings.findOne()` buffering timed out after 10000ms
    at Timeout.<anonymous> (/app/node_modules/mongoose/lib/drivers/node-mongodb-native/collection.js:153:23)
    at listOnTimeout (node:internal/timers:559:17)
    at processTimers (node:internal/timers:502:7)
[ERROR]
 MongooseError: Operation `settings.findOne()` buffering timed out after 10000ms
    at Timeout.<anonymous> (/app/node_modules/mongoose/lib/drivers/node-mongodb-native/collection.js:153:23)
    at listOnTimeout (node:internal/timers:559:17)
    at processTimers (node:internal/timers:502:7)
```

Figura 19: Estado del servicio

- Captura de datos

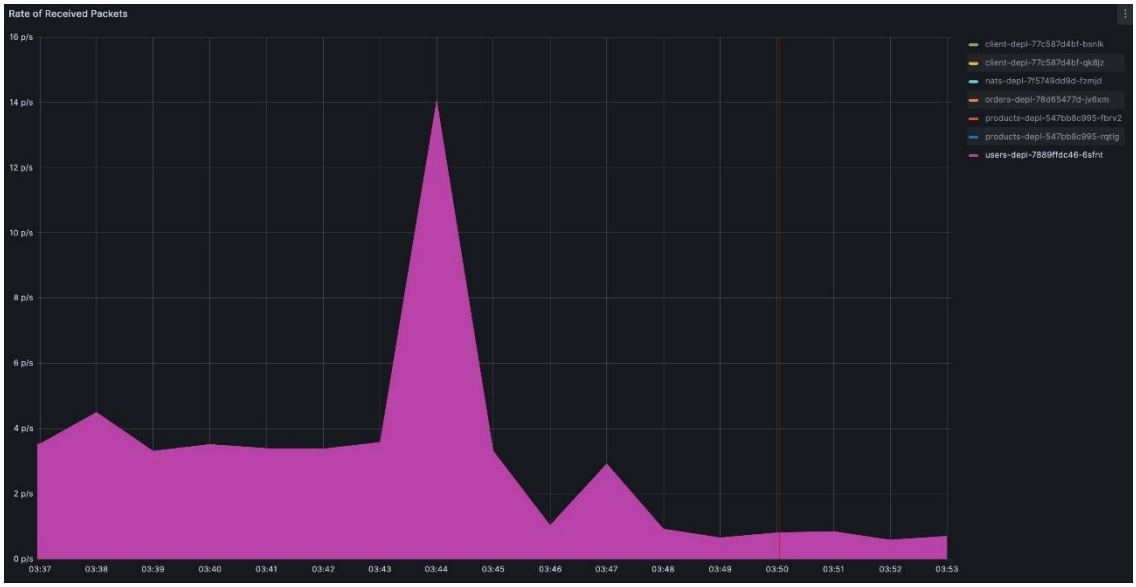

Figura 20: Estado del servicio

- Tasa de disponibilidad

Tiempo fuera de servicio = 24h

$$A = (\frac{86{,}400 - 86{,}400}{86{,}400}) * 100$$

$$A = (\frac{0}{86{,}400}) * 100$$

$$A = 0\%$$

Caso de prueba 5

- Experimento

sgp-cafeterias - 5-network-delay.yaml

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: network-delay
spec:
  schedule: '@every 3m'
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: NetworkChaos
  networkChaos:
    selector:
      namespaces:
        - default
      labelSelectors:
        app: products
    mode: one
    action: delay
    duration: 30s
    delay:
      latency: 5s
      correlation: '100'
      jitter: 10ms
    direction: to
```

Figura 21: Manifiesto Network Delay

- Resultado de ejecución

```
kubectl get pods -l app=products
NAME                              READY   STATUS    RESTARTS        AGE
products-depl-d4dc5b7d6-gl9vv     1/1     Running   5 (6m48s ago)   78m
products-depl-d4dc5b7d6-j4bnz     1/1     Running   1 (52s ago)     22m
```

Figura 22: Pod reemplazado

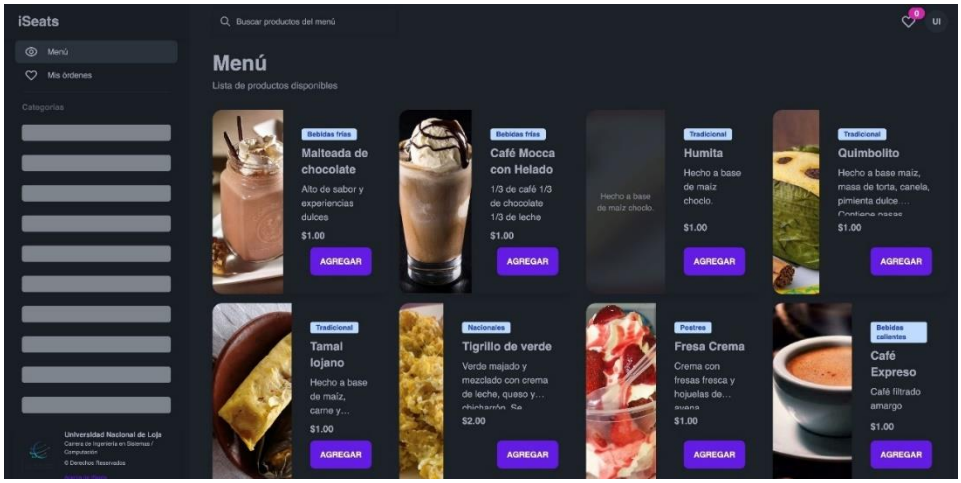

Figura 23: Estado de la aplicación

- Captura de datos

Figura 24: Pod reemplazado

- Tasa de disponibilidad
- Tiempo fuera de servicio = 2s.

$$A = \left(\frac{86400 - 2}{86400}\right) * 100$$

$$A = \left(\frac{86398}{86400}\right) * 100$$

$$A \approx 99.9977\%$$

Anexo 3. Ejecución de mejoras y pruebas posteriores

Caso de prueba 1

No aplica

Caso de prueba 2

- Mejora

sgp-cafeterias - client-depl.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: client-depl
spec:
  replicas: 2 # Incrementando a 2 replicas
  selector:
    matchLabels:
      app: client
  template:
    metadata:
      labels:
        app: client
```

Figura 25: Actualización de manifiesto de deployment Client

- Resultado de ejecución

```
kubectl get pods -l app=client
NAME                          READY   STATUS    RESTARTS        AGE
client-depl-77c587d4bf-bsnlk   1/1     Running   8 (2m34s ago)   25m
```

Figura 26: Estado del pod

```
kubectl get pods -l app=client
NAME                          READY   STATUS    RESTARTS         AGE
client-depl-77c587d4bf-bsnlk   1/1     Running   12 (3m36s ago)   36m
client-depl-77c587d4bf-qk8jz   1/1     Running   0                6m32s
```

Figura 27: Replicas del pod

- Captura de datos

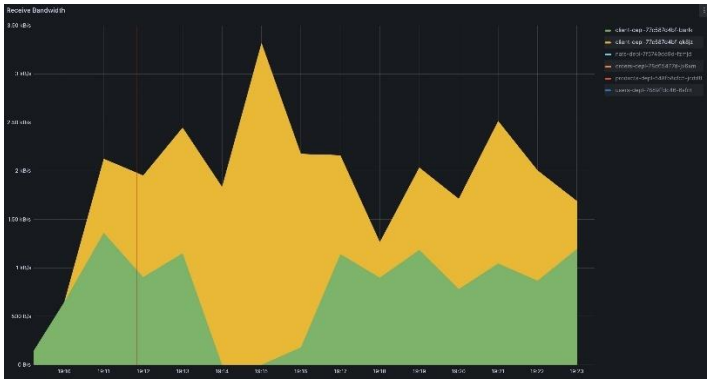

Figura 28: Replicas del pod

- Tasa de disponibilidad

Tiempo fuera de servicio = 2s

$$A = \left(\frac{86400 - 2}{86400}\right) * 100$$

$$A = \left(\frac{86398}{86400}\right) * 100$$

$$A \approx 99.9977\%$$

Caso de prueba 3

- Mejora

sgp-cafeterias - products-delp.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: products-depl
spec:
  replicas: 2 # Incrementando a 2 replicas
  selector:
    matchLabels:
      app: products
  template:
    metadata:
      labels:
        app: products
```

Figura 29: Actualización de manifiesto de deployment Products

- Resultado de ejecución

```
kubectl get pods -l app=products
NAME                            READY   STATUS             RESTARTS      AGE
products-depl-547bb8c995-fbrv2   0/1     CrashLoopBackOff   3 (50s ago)   13m
```

Figura 30: Estado del pod

```
kubectl get pods -l app=products
NAME                            READY   STATUS    RESTARTS      AGE
products-depl-547bb8c995-fbrv2   1/1     Running   4 (53s ago)   13m
```

Figura 31: Estado del pod

```
kubectl logs products-depl-547bb8c995-fbrv2

Connected to MongoDB
Connected to NATS
Connected to NATS
Listen on port 3000
```

Figura 32: Estado del servicio

- Captura de datos

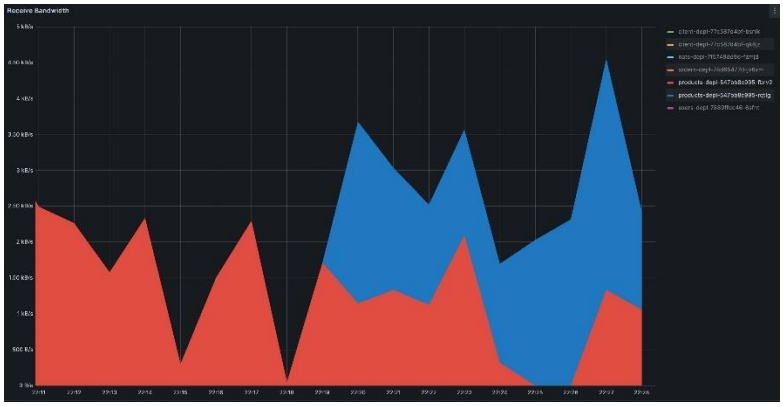

Figura 33: Estado del pod

- Tasa de disponibilidad

Tiempo fuera de servicio = 2s

$$A = \left(\frac{86400 - 2}{86400}\right) * 100$$

$$A = \left(\frac{86398}{86400}\right) * 100$$

$$A \approx 99.9977\%$$

Caso de prueba 4

- Mejora

```yaml
# sgp-cafeterias - users-depl.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: users-depl
spec:
  replicas: 2 # Incrementando a 2 replicas
  selector:
    matchLabels:
      app: users
  template:
    metadata:
      labels:
        app: users
```

Figura 34: Actualización de manifiesto de deployment Users

- Resultado de ejecución

```
kubectl get pods -l app=users
NAME                           READY   STATUS    RESTARTS      AGE
users-depl-79f8d46954-gmdl7    1/1     Running   0             2m25s
users-depl-79f8d46954-lgszn    1/1     Running   1 (54s ago)   2m10s
```

Figura 35: Estado del pod

- Captura de datos

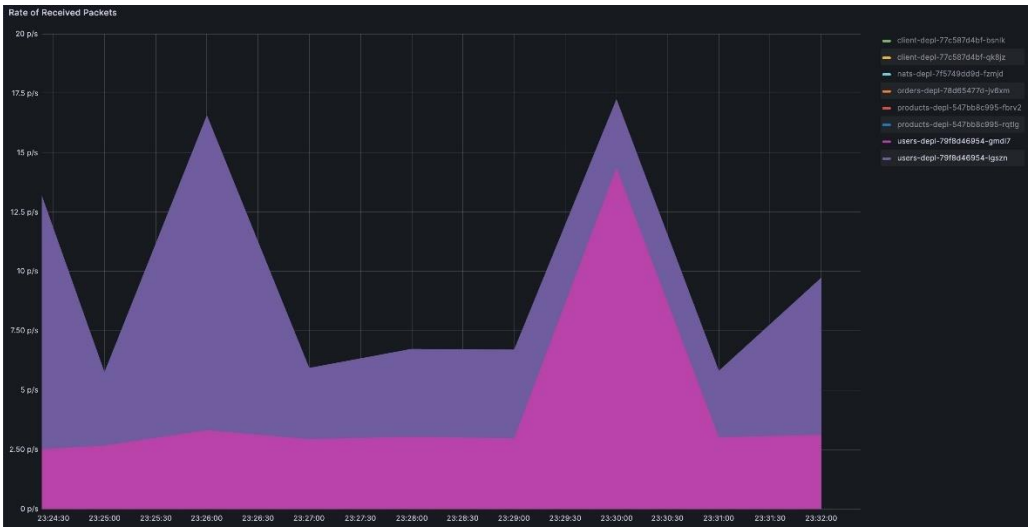

Figura 36: Estado del pod

- Tasa de disponibilidad

Tiempo fuera de servicio = 2s

$$A = \left(\frac{86400 - 2}{86400}\right) * 100$$

$$A = \left(\frac{86398}{86400}\right) * 100$$

$$A \approx 99.9977\%$$

Caso de prueba 5

No aplica

Anexo 4. Certificación que avala la traducción del resumen del TT.

Mg. Yanina Quizhpe Espinoza
Licenciada en Ciencias de Educación mención Inglés
Magister en Traducción y mediación cultural

Celular: 0989805087
Email: yaniqes@icloud.com
Loja, Ecuador 110104

Loja, 19 de diciembre de 2024

Yo, Lic. Yanina Quizhpe Espinoza, con cédula de identidad 1104337553, docente del Instituto de Idiomas de la Universidad Nacional de Loja, y con master en Traducción, con registro 724187576 en la Senescyt, certifico:

Que tengo el conocimiento y dominio de los idiomas español e inglés, y que la traducción del resumen del Trabajo de Titulación **Implementación de pruebas de carga y estrés automatizadas basadas en Chaos Engineering para la medir la tasa de disponibilidad de un Clúster Kubernetes** de autoría de María Anabel Encalada Córdova, con cédula 0707031662, maestrante del programa de Maestría en Ingeniería de Software, perteneciente a la Facultad de la Energía, las Industrias y los Recursos Naturales No Renovables de la Universidad Nacional de Loja, es fiel y correcta conforme a mi mejor saber y entender.

Atentamente

Firmado electrónicamente por:
YANINA BELEN QUIZHPE ESPINOZA

Mg. Yanina Quizhpe Espinoza.
Traductora freelance