UNIVERSIDAD MAYOR, REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA

VICERRECTORADO

CENTRO DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

FACULTAD DE CIENCIAS Y TECNOLOGÍA

"HERRAMIENTAS DEVOPS PARA PRUEBAS DE RENDIMIENTO EN EL DESARROLLO DEL SISTEMA DE ADMINISTRACIÓN NOTARIAL DE LA EMPRESA EGI"

TRABAJO EN OPCIÓN A DIPLOMADO EN DEVELOPMENT OPERATIONS "DEVOPS" V.1.

AUTOR: CHOQUE CRUZ LUIS

SUCRE-BOLIVIA

2024

CESIÓN DE DERECHOS

Al presentar este trabajo como requisito previo para la obtención del Diploma en Developement Operations "DEVOPS" V.1. de la Universidad Mayor, Real y Pontificia de San Francisco Xavier de Chuquisaca, autorizo al Centro de Estudios de Posgrado e Investigación o a la Biblioteca de la Universidad, para que se haga de este trabajo un documento disponible para su lectura según normas de la Universidad

También cedo a la Universidad Mayor, Real y Pontifica de San Francisco Xavier de Chuquisaca, los derechos de publicación de este trabajo o parte de él, manteniendo mis derechos de autor hasta un periodo de 30 meses posterior a su aprobación.

Luis Choque Cruz

DEDICATORIA Y AGRADECIMIENTOS

A Dios por concederme esta oportunidad; “por que nadie puede tener nada, si Dios no se lo da”.

A mis padres, a mi esposa e hijo por su apoyo incondicional, llenándome siempre de amor.

RESUMEN

La presente investigación surge ante la necesidad de poder desarrollar software de calidad, sobre todo completando el ciclo de vida del desarrollo del software en la etapa de pruebas específicamente pruebas de rendimiento y monitorización, debido a que en el desarrollo del sistema notarial de la empresa EGI, se presentaron constantemente notificaciones de errores por parte del cliente, para lo cual se propone la elección de herramientas DevOps para realizar pruebas de rendimiento que ayuden en esta etapa y poder mitigar lo más posible fallas, identificar los cuellos de botella y otros, además de proporcionar buenas prácticas para que en lo posterior se pueda realizar un producto de buena calidad y se pueda anticipar a posibles errores en la etapa de desarrollo y producción.

INDICE

INDICE DE TABLAS

INDICE DE GRÁFICOS

INTRODUCCIÓN

1. Antecedentes y Justificación

Durante los últimos años se han desarrollado nuevas herramientas que han cambiado la forma en la que los administradores de sistemas y desarrolladores colaboran para producir software de mejor calidad.

(Olmedo Rodríguez, 2018) en su trabajo de investigación en el campo de los test de performance y análisis de rendimiento de sistemas productivos, sobre todo en el despliegue. en el marco con enfoque DevOps, tratando en reducir la distancia entre los grupos de desarrollo y operaciones, profundizando en la automatización y despliegues continuos para lo cual es necesario ciertos test de performance o rendimiento puedan ser realizados, como también la simulación práctica en la que se despliegan algunas herramientas más utilizadas como webapp y poder obtener métricas del uso de la aplicación.

(Villamarín, 2019), en su investigación "Introducción a DevOps para la mejora de los procesos de desarrollo con herramientas Open Source" describe en qué consiste cómo funciona DevOps en una organización, identifica los beneficios de implementar DevOps en las empresas y los desafíos durante el proceso de adopción y cómo utilizar en el ciclo de vida del software, analizando herramientas que se utilizan para trabajar con DevOps sobre todo Open Source.

(Gallego Durango, Wildey Alejandro, 2022 "Investigación y desarrollo de pruebas de Rendimiento automatizadas"), en su investigación de desarrollo de pruebas y posterior comparativa de tres herramientas dedicadas a la automatización de pruebas de carga vía Scripting. El departamento de Aseguramiento de la Calidad (QA) de Wolox Part of Accenture evaluó las herramientas Gatling, K6 y Artillery.io para pruebas de rendimiento, tomando una decisión basada en análisis clave para pruebas automatizadas.

La empresa E.G.I. (Eficiencia en Gestión Informática), se dedica a prestar servicios informáticos en los últimos diez años, también desarrollan productos de software siendo un área nueva con personal limitado, cuyo objetivo es: satisfacer la necesidades del cliente y entregar productos de calidad, procurando mantener la disponibilidad, integridad y confidencialidad de

la información, tanto interna como externa, generando las mejores prácticas para que el personal administre y resguarde la información de forma segura, garantizando la reputación de la empresa y la continuidad del negocio.

El equipo de programación de la empresa está conformado por un equipo pequeño, desarrollando software de forma tradicional. Entre uno de los sistemas que van desarrollando: el sistema de administración notarial; trabajan utilizando un entorno de programación con el lenguaje java, cuyo servicio cliente-servidor está implementado sobre un servidor linux, con administración de base de datos en postgres, dando como uno de sus productos una aplicación de escritorio que trabaja en forma local sobre una red interna dado los requisitos del cliente para mantener toda la información generada de forma confidencial.

Analizando el papel crítico y la importancia que tienen las pruebas de rendimiento en el desarrollo de software variedad herramientas DevOps permiten tanto la automatización y el monitoreo continuo en los diferentes casos, permitiendo así mejorar la calidad y la fiabilidad como también la pronta entrega del software evitando así la menor cantidad de errores posibles (Olmedo Rodrígues, 2018).

Las pruebas de rendimiento ayudan a localizar puntos clave que permiten solucionar errores de manera oportuna antes de que estos ocurran, de manera tal que el equipo de desarrollo sea notificado para dar una retroalimentación. Además de que utilizando herramientas DevOps para este aspecto puedan reducir el tiempo que se demora en hacer estas pruebas, de manera tal que el equipo pueda realizar otras tareas (Sánchez Salinas, 2023).

Justificación. –

Práctica: Al emplear enfoques DevOps como buenas prácticas y el uso de estas herramientas dando como resultado operativizar de mejor manera y resolver los problemas planteados de forma directa al desarrollar software en la fase de pruebas. Con el uso de herramientas DevOps, permite agilizar y automatizar el proceso de pruebas de rendimiento, permitiendo identificar y solucionar rápidamente cualquier problema que pueda surgir, estas herramientas permitirán agilizar el proceso de desarrollo, adaptarse a los cambios del negocio y garantizar una experiencia óptima para los usuarios. En un entorno empresarial cada vez más competitivo y

cambiante, contar con herramientas DevOps se convierte en un factor clave para asegurar el éxito y la eficiencia en el desarrollo de aplicaciones.

Metodológica: Al proponer estrategias y buenas prácticas en la fase de pruebas del ciclo de vida del software, permitirá mejorar en la eficiencia y eficacia en el desarrollo de software.

2. Situación Problémica

El software cuando es sometido a una sobrecarga de operaciones, ocasiona lentitud en la respuesta e interacción con el usuario. Sucede al momento de que el sistema está siendo utilizado por muchos usuarios al mismo tiempo entre 20 usuarios o más, con consultas o registros hace que el sistema se comporte de manera muy lenta, demorando así la atención a sus clientes.

La aplicación en varias ocasiones muestra errores y se cuelga debido a que estos errores al ser analizados no se tuvieron en cuenta al momento de desarrollar, por la falta de varios casos de prueba en situaciones distintas; una de ellas la cantidad de memoria utilizada por el sistema que no se libera una vez que estas ya no están en uso hacen que el equipo computacional se vea limitado en sus recursos y presente lentitud en sus procesos generales.

Existen situaciones imprevistas que no se tomaron en cuenta al momento de desarrollar ni tampoco situaciones dadas por el cliente, por lo que esto genera malestar en el usuario final por que no se tomaron medidas anticipadas, mayormente los casos de prueba son tediosos que toman mucho tiempo, al equipo le toma entre dos semanas o más por lo general el equipo de desarrollo pasa por alto para poder entregar el software lo más antes posible.

Entrega del software tardío debido a que se demoran en promedio uno o dos meses en hacer las correcciones ante un eventual mal funcionamiento debido a que no se tuvo en cuenta pruebas de rendimiento. El equipo de desarrollo demora en hacer las correcciones ante fallas que no se tenían previstas sobre todo cuando se aumenta mayor funcionalidad y mayor cantidad en el procesamiento de información ya que en muchos casos son complejas y que se tiene que tomar más tiempo en resolverlas para mejorar el rendimiento de la ampliación.

A medida que se utiliza el sistema y se agregan nuevas funcionalidades, el sistema se comporta de manera inestable, de cierta medida pasaron por alto las pruebas de escalabilidad que permitan dar certeza de que el software desarrollado crezca ya sea cuando maneje más volumen de información como también se agregue nuevas funciones al mismo.

3. Formulación del Problema de Investigación o Pregunta Científica

¿Qué herramientas DevOps seleccionar para realizar pruebas de rendimiento que ayuden a mejorar el desarrollo de software del sistema de administración notarial de la empresa EGI?

4. Objetivo General

Seleccionar las herramientas DevOps adecuadas para automatizar pruebas de rendimiento en el desarrollo del sistema de administración notarial de la empresa EGI.

5. Objetivos Específicos

- Comparar diferentes herramientas DevOps que se adecuen al tipo de software en desarrollo, para realizar pruebas de rendimiento.
- Proponer un conjunto de buenas prácticas para realizar pruebas de rendimiento.
- Evaluar los beneficios y consideraciones al utilizar herramientas DevOps en las pruebas de rendimiento.

6. Diseño Metodológico (Métodos, técnicas, procedimientos e instrumentos de investigación)

6.1. Tipo de Investigación

La presente investigación tiene un enfoque cualitativo, de carácter descriptivo, donde se explorarán y seleccionarán las diferentes herramientas DevOps que ayuden en el proceso de pruebas automatizadas, bajo criterios que se adecuen al marco de trabajo y el tipo de proyecto de desarrollo de software de la empresa.

6.2. Métodos

6.2.1. Métodos Teóricos

Análisis Documental - Método Bibliográfico: base que se empleó para analizar la temática DevOps, como también en la redacción de las diferentes teorías en el marco teórico.

Análisis-Síntesis: Este método fue el que más utilizado a lo largo del desarrollo de la monografía, puesto que tiene gran utilidad para la búsqueda y el procesamiento de la información (clasificación, descomposición, división), fue aplicado en las siguientes etapas: en la delimitación del tema, redacción de la situación problemática, la formulación del problema, redacción de los objetivos, así como en el marco teórico.

Simulación: Este método se utilizó para determinar los diferentes escenarios con los que se comporta el sistema desarrollado con diferentes herramientas DevOps para las pruebas de rendimiento, este método se utilizó al momento de realizar comparaciones en la hipótesis y los resultados.

6.2.2. Métodos Empíricos

Estudio de casos: Analizar casos de empresas que han implementado herramientas DevOps para pruebas de rendimiento. Esto permitirá identificar los beneficios y desafíos que enfrentaron durante el proceso de implementación.

Experimentación práctica: Realizar pruebas de rendimiento en el sistema de administración notarial de la empresa utilizando diferentes herramientas DevOps. Analizar los resultados obtenidos para evaluar la eficacia de cada herramienta en términos de mejoras en el rendimiento del sistema.

6.3. Técnicas

Análisis documental, para la interpretación del log de archivos generados por el sistema en desarrollo, la bitácora que contiene el registro de eventos en los diferentes procesos.

Observación, en el comportamiento de las diferentes herramientas DevOps frente a situaciones de pruebas de rendimiento.

6.4. Procedimientos e instrumentos de investigación

Reportes: Análisis del reporte de pruebas, como también el log de archivos que genera el sistema ante cualquier eventualidad.

Matriz de comparación para evaluar y comparar diferentes herramientas DevOps en función de criterios importantes como funcionalidades, escalabilidad, costo y soporte, esto te ayudará a tomar decisiones informadas sobre qué herramientas podrían ser más adecuadas para el proyecto.

Guía de observación, como instrumento de registro en el que tomaran nota de cada prueba de rendimiento.

CAPÍTULO I

MARCO TEORICO Y CONTEXTUAL

1.1. Marco Teórico

1.1.1. Principales teorías que abordan la temática

1.1.1.1. ¿Qué es DevOps?

El término DevOps, que es una combinación de los términos ingleses development (desarrollo) y operations (operaciones), designa la unión de personas, procesos y tecnología para ofrecer valor a los clientes de forma constante (Azure Microsoft, 2024)

“Bajo un modelo de DevOps, los equipos de desarrollo y operaciones ya no están aislados. A veces, los dos equipos se fusionan en uno solo, donde los ingenieros trabajan en todo el ciclo de vida de la aplicación, desde el desarrollo y las pruebas hasta la implementación y las operaciones, y desarrollan una variedad de habilidades no limitadas a una única función” (AWS, 2024).

Aunque no existe una definición específica, todos coinciden en que DevOps permite que los roles que antes estaban aislados (desarrollo, operaciones de TI, ingeniería de la calidad y seguridad) se coordinen y colaboren para producir productos mejores y más confiables, por lo que al adoptar una cultura de DevOps junto con prácticas y herramientas de DevOps, los equipos adquieren la capacidad de responder mejor a las necesidades de los clientes, aumentar la confianza en las aplicaciones que crean y alcanzar los objetivos empresariales en menos tiempo.

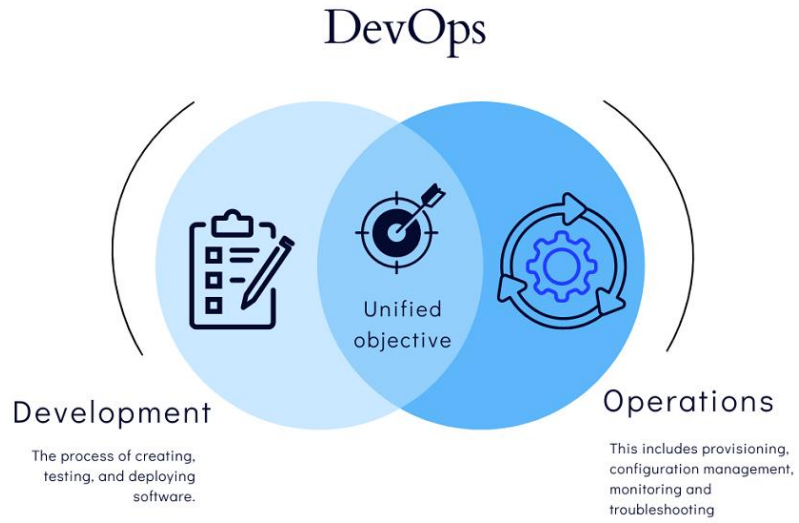

Gráfico 1: DevOps Concepto Fuente: https://www.manageengine.com/latam/applications_manager/images/devops-meaning.png

1.1.1.2. Cultura DevOps

Con la llegada de las metodologías ágiles de desarrollo y las necesidades de realizar integración y entrega continua (CI, continuous integration y CD,continuous delivery) aparece una nueva corriente organizativa llamada DevOps, que, en resumidas cuentas pretende aunar en un único equipo a perfiles muy separados en organizaciones más tradicionales como puedan ser los desarrolladores y los equipos de operaciones, todo ello con el objetivo final de realizar despliegues en entornos productivos de forma más regular. Haciendo nuevas entregas del software de una forma regular (semanalmente, diariamente o, incluso, varias veces al día) se consigue dotar al proceso del paso a producción de más seguridad o estabilidad y más eficiencia. Cuanto más regularmente se haga una tarea, en este caso un despliegue en producción, menos «doloroso» será. Para ello se requiere un nivel muy alto en la automatización de los procesos de compilación, empaquetado, pruebas, despliegues, pruebas y monitorización.

Algunas de las más importantes empresas que dan servicio con diferentes herramientas y grandes infraestructuras para implementar DevOps se tienen los siguientes criterios en cuanto a la *cultura DevOps*: “DevOps implica una cultura de colaboración que sigue los principios del open source y adopta enfoques transparentes y ágiles para el trabajo” (RedHat, "Cultura DevOps", 2022).

” Aunque la adopción de prácticas de DevOps automatiza y optimiza los procesos con tecnología, todo comienza con la cultura interna de la organización y con las personas que participan en ella. El desafío de cultivar una cultura de DevOps requiere cambios profundos en la forma en la que las personas trabajan y colaboran. Pero cuando las organizaciones se comprometen a implementar una cultura de DevOps, pueden crear un entorno que facilite el desarrollo de equipos de alto rendimiento” (Azure Microsoft, 2024).

Historia de la evolución de DevOps:

Para entender los inicios y la evolución de DevOps a lo largo de la Historia aquí se tiene algunos hitos importantes:

2009: El término "DevOps" es acuñado por Patrick Debois y Andrew Clay Shafer durante una conferencia en Bélgica.

2010: John Willis y Damon Edwards organizan la primera conferencia DevOps Days en Estados Unidos.

2011: Se publica el libro "The Phoenix Project" de Gene Kim, Kevin Behr y George Spafford, que populariza los principios de DevOps a través de una novela empresarial.

2013: Se publica "The DevOps Handbook" de Gene Kim, Jez Humble, Patrick Debois y John Willis, ofreciendo una guía práctica para implementar DevOps en organizaciones.

2014: La conferencia AWS re:Invent presenta el concepto de "DevOps en la nube", destacando la importancia de la automatización y la escalabilidad en entornos cloud.

2015: Se establece el "State of DevOps Report" por DORA (DevOps Research and Assessment), proporcionando datos empíricos sobre la efectividad de las prácticas DevOps.

2017: Microsoft adquiere GitHub, una plataforma líder en el desarrollo colaborativo de software, lo que refuerza su enfoque en DevOps.

2018: Kubernetes, una herramienta clave para la orquestación de contenedores, gana popularidad y se convierte en un elemento central en las implementaciones de DevOps.

2019: La adopción de prácticas DevOps continúa en aumento, con más empresas reconociendo los beneficios de la automatización, la colaboración y la entrega continua.

2020-2021: La pandemia de COVID-19 impulsa la transformación digital en muchas organizaciones, acelerando la adopción de DevOps y herramientas relacionadas.

2022: La adopción de DevOps sigue siendo una prioridad para muchas empresas, con un enfoque creciente en la seguridad, la observabilidad y la integración de IA/ML en los flujos de trabajo de DevOps.

Herramientas DevOps en el ciclo de vida del desarrollo de software

DevOps fomenta la integración entre los equipos de desarrolladores y los profesionales de operaciones hacia TI, inclinándose un ciclo iterativo, buscando siempre la mejora continua y entregando valor al negocio.

El autor (AllSpaw, 2017) menciona que: DevOps tiene su propio ciclo de vida, relacionado con la automatización del proceso de desarrollo de software, desde el concepto (idea) hasta la operación (producto en producción) y se divide en 8 fases, que varían tanto por la adición de componentes por etapas o simplificación de las mismas.

Los seguidores de las prácticas de DevOps a menudo agregan a su "cadena de herramientas" específicas de DevOps algunas herramientas que se adaptan bien a estos enfoques. El objetivo de estas herramientas es tratar de simplificar, cortar y automatizar varias etapas del flujo de trabajo de creación de software (o "canalización"). Muchas de estas herramientas promueven los principios básicos para el desarrollo de operaciones.

Ejemplos de herramientas utilizadas en varias etapas del ciclo DevOps según (Vecillas, 2020):

Planificación. En esta etapa se definen las necesidades y valores del negocio. Algunas herramientas de ejemplo son Jira o Git, que se pueden usar para rastrear problemas conocidos y administrar proyectos.

Codificación. Esta fase consiste en diseñar el software y crear el código. Algunas herramientas de ejemplo son GitHub, GitLab, Bitbucket o Stash.

Compilación. Esta fase administra el lanzamiento y la construcción del software, utilizando herramientas automatizadas para ayudar a compilar y empaquetar el código para su posterior lanzamiento a producción. Utilizando repositorios de código fuente o repositorios de paquetes, también "empaquetan" la infraestructura necesaria para el lanzamiento del producto. Algunas

herramientas de ejemplo son Docker, Ansible, Puppet, Chef, Gradle, Maven o JFrog Artifactory.

Prueba. Esta fase incluye pruebas continuas (manuales o automatizadas) para garantizar la calidad de la programación. Algunas herramientas de ejemplo son JUnit, Codeception, Selenium, Vagrant, TestNG o BlazeMeter.

Puesta en marcha. En esta fase, las herramientas se utilizan para ayudar a administrar, coordinar, programar y automatizar las tareas de producción para los lanzamientos de productos. Algunas herramientas de ejemplo son Puppet, Chef, Ansible, Jenkins, Kubernetes, OpenShift, OpenStack, Docker o Jira.

Supervisión. Esta fase identifica y recopila información sobre problemas que ocurren en una versión de software específica en producción. Algunas herramientas de ejemplo son New Relic, Datadog, Grafana, Wireshark, Splunk, Nagios o Slack.

Funcionamiento. En esta etapa, el software se gestiona durante su producción. Algunas herramientas de ejemplo son Ansible, Puppet, PowerShell, Chef, Salt o Otter.

Estas son algunas de las herramientas disponibles ya que existen una variedad de soluciones en todo el ciclo de vida de desarrollo de software.

Gráfico 2; *Herramientas DevOps en el ciclo de vida del software*

1.1.1.3. Pruebas

Las pruebas son una parte fundamental en el proceso de desarrollo de software, ya que permiten asegurar la calidad del producto final y detectar posibles errores o fallos en el funcionamiento del software.

El comportamiento de un sistema de software depende de factores: internos y externos, por lo que resulta fundamental que los desarrolladores cuenten con estrategias capaces de minimizar los errores de un producto informático desarrollando pruebas.

Es gracias a este objetivo que las pruebas de software o software testing, han tenido un impulso y resultan más relevantes que nunca, al permitir verificar que los productos estén libres de defectos o bugs, y contribuyen a validar el funcionamiento integral de un sistema.

Los tipos de pruebas más generales son:

Pruebas de unidad: se centran en verificar el correcto funcionamiento de cada componente individual del software, de forma aislada.

Pruebas de integración: se realizan para comprobar que todos los componentes del software interactúan correctamente entre sí.

Pruebas de sistema: se llevan a cabo para validar que el sistema en su conjunto cumple con los requisitos especificados.

Pruebas de aceptación: son realizadas por los clientes o usuarios finales para comprobar que el software cumple con sus necesidades y expectativas.

En el proceso de pruebas, se utilizan diferentes técnicas y herramientas, como la automatización de pruebas, que facilitan la detección de errores de forma más eficiente y rápida. Además, es importante llevar a cabo un seguimiento de los resultados de las pruebas y mantener un registro de los errores encontrados, para poder corregirlos y mejorar la calidad del software.

Para el presente caso de estudio se tomarán en cuenta las pruebas de rendimiento:

1.1.1.4. Pruebas de rendimiento

Las pruebas de rendimiento son un tipo de pruebas de software que ayudan a determinar el rendimiento de un software en términos de velocidad, tiempo de respuesta, escalabilidad, uso de recursos y estabilidad bajo una carga de trabajo determinada, cuyo objetivo es identificar todos los cuellos de botella de rendimiento de una aplicación.

1.1.1.5. Tipos de prueba de rendimiento:

A continuación, se describen los tipos de prueba que se tienen en el desarrollo de software:

Pruebas de carga

Las pruebas de carga determinan cómo se comporta el software con el aumento de la carga de trabajo en un tiempo determinado. Esta carga de trabajo puede ser usuarios concurrentes, el número de transacciones, el comportamiento del software, etc.

Su objetivo es observar el tiempo de respuesta, las tasas de rendimiento, la utilización de los recursos, etc. Al identificar cualquier cuello de botella en el rendimiento de estos atributos, puede solucionarlos antes de lanzar la aplicación para garantizar una mejor experiencia del usuario final.

Gracias a las pruebas de carga, los desarrolladores pueden saber cuántos usuarios concurrentes pueden manejar una aplicación de software en un momento dado.

Pruebas de resistencia

El también llamado soak testing, endurance testing evalúa el rendimiento del software durante un período prolongado bajo una carga de trabajo regular y fija. En otras palabras, determina cuánto tiempo puede soportar el software una carga de trabajo constante para proporcionar sostenibilidad a largo plazo.

Durante estas pruebas, los equipos de pruebas supervisan los KPI como las fugas de memoria, el uso de la memoria, la escasez de memoria, etc. Las pruebas de resistencia también analizan los tiempos de respuesta y el rendimiento tras un uso prolongado para mostrar si estas métricas son consistentes o no.

Pruebas de estrés

Las pruebas de estrés miden el rendimiento del software más allá de los parámetros normales de funcionamiento. Significa que el software se somete a cargas de tráfico más elevadas (más usuarios, transacciones, etc.) para ayudar a los desarrolladores a conocer el rendimiento del software por encima de sus límites de capacidad previstos.

El objetivo de estas pruebas es determinar la estabilidad del software. Ayuda a entender el punto en el que el software falla y cómo se recupera del fallo. Se esfuerzan los recursos de hardware como la CPU, la memoria, el espacio en disco, etc., para medir el punto de ruptura de la aplicación en función de la utilización de los recursos.

Ejemplo: Las empresas realizan pruebas de estrés antes de algunos eventos importantes como el Black Friday para comprobar el rendimiento de sus aplicaciones de comercio electrónico.

Pruebas de picos

La prueba de picos es un tipo de prueba de estrés que mide el rendimiento del software bajo un «pico» significativo y repentino o una carga de trabajo creciente como la de los usuarios simulados. Indica si el software puede manejar ese aumento abrupto de la carga de trabajo de forma repetida y rápida.

Pruebas de volumen

Durante la fase de desarrollo, sólo se utiliza una pequeña cantidad de datos para probar el código. Las pruebas de volumen (también llamadas «pruebas de inundación») comprueban la eficacia del software cuando se somete a grandes volúmenes de datos. Comprueba la pérdida de datos, el tiempo de respuesta del sistema, la fiabilidad del almacenamiento de datos, etc.

Pruebas de escalabilidad

Las pruebas de escalabilidad miden la eficacia del software a la hora de manejar una cantidad creciente de carga de trabajo. La prueba te informará sobre el comportamiento cuando aumenten o disminuyan los atributos de rendimiento del software.

Se pueden realizar pruebas de escalabilidad añadiendo volumen de datos o usuarios de forma gradual mientras supervisa el rendimiento del software.

1.1.1.6. Beneficios de las pruebas de rendimiento

Según un informe publicado en marzo de 2019, Facebook sufrió un apagón catastrófico y perdió unos 90 millones de dólares en ingresos. Del mismo modo, el tiempo de inactividad de App Store provocó pérdidas de 25 millones de dólares, según otro informe de marzo de 2015.

A pesar de contar con una sólida infraestructura informática y de seguridad, estos gigantes tecnológicos sufrieron enormes pérdidas. Esto demuestra la importancia de las pruebas de rendimiento para tus aplicaciones. Garantiza que todas las características, funcionalidades y sistemas funcionen de forma óptima para ofrecer una mejor experiencia al usuario.

Entre algunos beneficios de las pruebas de rendimiento se tienen:

Mide la velocidad, la estabilidad y la precisión del software: La velocidad, la estabilidad y la precisión son algunos de los atributos significativos del rendimiento del software. Por lo tanto, probar tu aplicación en estos aspectos te permite controlar cómo se comporta el software bajo presión y te proporciona detalles cruciales sobre cómo puedes manejar la escalabilidad.

Al ver los resultados de las pruebas, los desarrolladores pueden saber qué cambios deben incorporar al software para superar los puntos negativos y hacerlo más eficiente.

Validar las características básicas del software: El éxito de tu aplicación depende de la solidez de sus cimientos. Medir el rendimiento de las características y funcionalidades fundamentales del software le ayuda a tomar decisiones informadas y a planificar su estrategia empresarial sobre la configuración del software.

Identificar los problemas y resolverlos: Las pruebas de rendimiento te indicarán los principales problemas que puedes corregir antes de lanzar tu sitio o aplicación. Esto significa que puedes resolver rápidamente todos los problemas identificados y concentrarte en mejorar la tecnología en lugar de luchar contra los problemas después del lanzamiento.

Mejorar la capacidad de carga y optimización del software: Con las pruebas de rendimiento, puedes manejar mejor la capacidad de volumen y carga y optimizar tu software de tal manera que te ayude a soportar un alto número de usuarios concurrentes. También detectará los

problemas de escalabilidad que puedes abordar lo antes posible y permitirá a los probadores adaptar la capacidad para manejar las altas demandas.

Mejorar la calidad del código y la funcionalidad del software: Cuando puedes detectar los problemas de tu software, puedes eliminarlos rápidamente. Como resultado, los desarrolladores pueden mantener un código de calidad y mejorar la funcionalidad de la aplicación, ofreciendo la velocidad, fiabilidad, escalabilidad y estabilidad esperadas.

Satisfacer las expectativas de los usuarios: A tus clientes no les gustará un software lento y con retrasos que les haga perder el tiempo y les moleste. Esperan que se cargue rápidamente -en 2 o 3 segundos- y que funcione con fluidez, sin ninguna confusión. La primera impresión de tu software es crucial para decidir si los usuarios querrán seguir utilizándolo o pulsar el botón de desinstalación.

Las pruebas de rendimiento le ayudan a cumplir con los atributos deseados del software y a mantener a tus usuarios contentos para que sigan utilizando tu software, aumentando tus ingresos.

1.1.1.7. Herramientas para pruebas de rendimiento

Existen varias herramientas disponibles para realizar pruebas de rendimiento, cada una con sus propias características y capacidades. Algunas de las herramientas más populares para llevar a cabo pruebas de rendimiento:

Apache JMeter:

Es una herramienta de código abierto diseñada para realizar pruebas de carga y rendimiento en aplicaciones web. Permite simular una carga pesada en un servidor, analizar el rendimiento general y medir la capacidad de respuesta del sistema bajo diferentes condiciones de carga.

Gatling:

Gatling es otra herramienta de código abierto que se utiliza para realizar pruebas de rendimiento y estrés en aplicaciones web. Está diseñada para ser altamente escalable y permite escribir escenarios de prueba en lenguaje DSL (Domain Specific Language) basado en Scala.

LoadRunner:

LoadRunner es una herramienta de pruebas de rendimiento desarrollada por Micro Focus. Ofrece una amplia gama de características para realizar pruebas de carga, estrés y rendimiento en una variedad de aplicaciones, incluyendo aplicaciones web, móviles, y basadas en protocolos.

BlazeMeter:

BlazeMeter es una plataforma en la nube para realizar pruebas de rendimiento y carga. Permite a los equipos ejecutar pruebas de forma simultánea desde múltiples ubicaciones geográficas, proporcionando análisis detallados y reportes en tiempo real.

Locust:

Locust es una herramienta de código abierto para realizar pruebas de carga distribuida. Permite escribir escenarios de prueba en Python y simular el comportamiento de miles de usuarios concurrentes en una aplicación.

NeoLoad:

NeoLoad es una herramienta comercial para realizar pruebas de carga y rendimiento. Ofrece capacidades avanzadas para simular el comportamiento de usuarios reales, monitorizar el rendimiento de la aplicación y analizar los resultados de las pruebas.

Apache Bench (ab):

Apache Bench es una herramienta de línea de comandos incluida en el paquete de Apache HTTP Server. Permite realizar pruebas simples de rendimiento y carga en servidores web, generando una carga de solicitudes HTTP y midiendo la velocidad de respuesta del servidor.

Mejores prácticas para realizar pruebas de rendimiento

Crear escenarios realistas. - Piensa como lo haría un usuario. ¿Qué es importante para tu base de usuarios? ¿Qué funciones de tu aplicación son críticas para ellos? ¿Utilizan diferentes dispositivos? Al crear pruebas de carga realistas, puedes entender más de cerca cómo se

comporta tu aplicación o se comportaría en producción con usuarios reales. Los usuarios reales, hasta cierto punto, son impredecibles, así que ten en cuenta la aleatoriedad y la variabilidad al evaluar los pasos a seguir en tus pruebas. Varía el tipo de dispositivo y navegador para sentirte seguro de que tu aplicación está lista para su implementación.

Prueba temprana, prueba con frecuencia. - Ya sea que tu equipo adopte una metodología ágil o DevOps esencial probar temprano y con frecuencia. Con frecuencia, las pruebas de rendimiento están aisladas y comienzan cuando un proyecto de desarrollo ha finalizado. Sin embargo, en los últimos años, aumentar la cantidad de retroalimentación a lo largo del ciclo de vida del desarrollo de software ha demostrado ser inmensamente valioso para encontrar y corregir problemas rápidamente. Prioriza hacer de las pruebas de rendimiento, y en particular, las pruebas de carga, parte de tus prácticas ágiles, de integración continua y automatización.

Establecer metas realistas. - Optimizar el rendimiento requiere una comprensión de tu aplicación y sus usuarios. Identifica pruebas prácticas y realistas que puedan reflejar la realidad, ya sea seleccionando dispositivos, navegadores, cantidad de usuarios, etc. Además, las pruebas de carga no pueden comenzar desde cero. En el mundo real, es poco probable que los sistemas que estás buscando actualizar no estén en funcionamiento bajo carga ya. En lugar de comenzar desde cero e incrementar gradualmente los usuarios virtuales hasta alcanzar la carga deseada, intenta realizar pruebas después de que tus sistemas ya estén bajo carga. De esta manera, evitas los ‘falsos positivos’ que pueden surgir al iniciar tus pruebas de carga desde cero.

Aprovechar datos de la vida real. -Para lograr escenarios y benchmarks realistas, aprovecha los datos que ya tienes. Reutilizar datos de tus herramientas de monitoreo puede ayudar a iluminar lo que ‘realista’ significa en tu caso específico. En la mayoría de los casos, las herramientas de monitoreo se ejecutan desde un punto de vista proactivo y reactivo, lo que significa que puedes utilizar datos sintéticos y de usuarios reales para crear escenarios que fallaron en producción con un monitor sintético y/o agregar interacciones que tus usuarios ya están teniendo con tu aplicación en tus escenarios de prueba. Esto puede incluir datos impulsados por el usuario, como navegadores, dispositivos, rutas de usuario, puntos de abandono y datos basados en el sistema, como carga del DOM, tiempo hasta el primer byte y más.

Analizar datos de prueba para descubrir problemas subyacentes. -Después de realizar tus pruebas de carga, el primer paso obvio es identificar cualquier área problemática y tomar los siguientes pasos para mejorar el rendimiento de ese componente. Esto implica correlacionar cuellos de botella de rendimiento con el código para aislar la causa raíz del problema. A menudo, esto puede ser difícil si estás utilizando una herramienta de prueba tradicional, ya que requiere 'traducir' los resultados de la prueba en métricas que puedes utilizar para compartir con tu equipo de desarrollo.

1.1.1.8. Casos de pruebas

Los casos de prueba son los escenarios que se utilizan para medir la funcionalidad de la aplicación a través de un conjunto de ciertas acciones o condiciones para verificar los resultados esperados. En otras palabras, un caso de prueba es un conjunto de acciones ejecutadas para autenticar la funcionalidad de su aplicación de software.

Un caso de prueba consta de varias cosas, como pasos de prueba, datos de prueba y condiciones previas y posteriores desarrolladas para un escenario de prueba particular. Los casos de prueba se pueden aplicar a cualquier aplicación de software. Se puede hacer a través de pruebas manuales y automatizadas o cualquier herramienta de gestión de pruebas.

1.1.1.9. Tipos de casos de prueba

Hay varios tipos diferentes de casos de prueba:

Caso de prueba de funcionalidad – Los casos de prueba de funcionalidad, como su nombre indica, se utilizan para analizar si el sistema está funcionando como se esperaba o no. El equipo de control de calidad es responsable de escribir los casos de prueba funcionales. Este tipo de prueba se puede realizar tan pronto como el equipo de desarrollo haga que la primera función de la aplicación esté disponible para la prueba. Por ejemplo, verificar si el usuario puede subir una foto de perfil.

Caso de prueba de integración – Los casos de prueba de integración se utilizan para analizar si los diferentes módulos de la aplicación interactúan entre sí correctamente o no. El equipo de pruebas es responsable de separar las áreas que deben someterse a pruebas de integración. Por otro lado, el equipo de desarrollo ayuda a escribir los casos de prueba de integración. Por

ejemplo, verificar si la página de inicio de sesión aparece cuando hacemos clic en el botón 'iniciar sesión' en la página de inicio.

Caso de prueba de usabilidad – Los casos de prueba de usabilidad, también conocidos como tareas o escenarios, son casos en los que los probadores presentan escenarios de alto nivel o tareas para completar en lugar de instrucciones paso a paso para realizar la prueba. Estos casos de prueba se utilizan para analizar cómo los usuarios suelen abordar y utilizar una aplicación. Por ejemplo, verificar si el usuario puede agregar más de un artículo a su carrito de compras en un sitio web de compras en línea y cómo es esa experiencia.

1.1.1.10. Pasos para escribir un caso de prueba

Hay varios pasos que se deben seguir para escribir casos de prueba. Incluyen:

ID de caso de prueba – Todos y cada uno de los casos de prueba deben estar representados con una identificación única. Esto hace que sea más fácil y conveniente entenderlos y distinguirlos.

Descripción del caso de prueba – Cada caso de prueba debe tener una descripción adecuada que contenga detalles importantes como qué característica, unidad o función se está probando y qué debe verificarse.

Condiciones previas – También acumulamos las condiciones que se deben tener en cuenta al ejecutar la prueba.

Pasos de prueba – Para poder ejecutar una prueba correctamente, debe comprender correctamente cómo ejecutar ese caso de prueba en particular. Entonces, escriba los pasos para ejecutar el caso de prueba en un lenguaje fácil y comprensible.

Información de prueba – Recopilar los datos necesarios para ejecutar el caso de prueba de forma precisa y concisa. Acumule esos datos en un documento claro y completo.

Resultado deseado – Explicar los resultados deseados del caso de prueba. Por ejemplo, al ingresar al botón de inicio de sesión, el usuario iniciará sesión en el sitio web con éxito.

Condiciones de publicación – Estas son las condiciones que se deben cumplir para la ejecución exitosa del caso de prueba.

Resultado real – Estos son los resultados que obtenemos después de ejecutar el caso de prueba. Con base en estos resultados, averiguamos si el caso de prueba pasó o falló.

Estado – Finalmente, encontramos el estado del caso de prueba como aprobado o reprobado sobre la base de los resultados reales que obtenemos al ejecutar el caso de prueba. Si obtenemos los resultados deseados, la prueba se marca como aprobada, si no, se marca como fallida. Si la prueba falla, pasa por el ciclo de vida del error para que los desarrolladores lo

1.1.1.11. Ciclo de vida DevOps

Debido al carácter continuo de DevOps, se usa un bucle infinito para representar que las fases del ciclo de vida de DevOps se relacionan entre sí. A pesar de que parece fluir de forma secuencial, este bucle simboliza la necesidad de colaboración constante y mejora iterativa a lo largo de todo el ciclo de vida

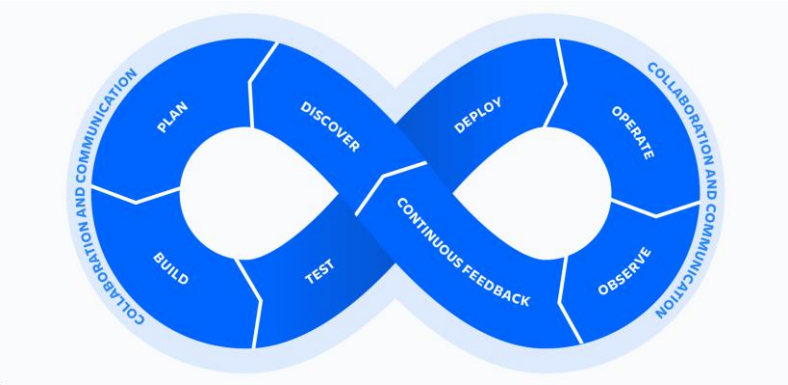

Gráfico 3 Ciclo de vida DevOps

El ciclo de vida de DevOps se divide en ocho fases, que representan los procesos, capacidades y herramientas necesarios para el desarrollo (en la parte izquierda del bucle) y para las operaciones (en la parte derecha). A lo largo de cada una de estas fases, los equipos trabajan juntos y mantienen la comunicación para ir coordinados y ajustarse a la misma velocidad y calidad. (Atlasian, 2024)

El ciclo de vida DevOps generalmente se divide en las siguientes etapas:

Planificación: En esta etapa se definen los requisitos del software y se planifica la estrategia de desarrollo.

Desarrollo: Los equipos de desarrollo crean el software siguiendo las especificaciones y requisitos establecidos en la etapa de planificación.

Pruebas: Una vez que se completa el desarrollo, el software se somete a pruebas exhaustivas para garantizar su calidad y funcionamiento correcto.

Despliegue: El software se despliega en un entorno de producción para que los usuarios finales puedan acceder a él.

Monitoreo y mantenimiento: Una vez que el software está en producción, es importante monitorear su rendimiento y realizar actualizaciones y mantenimiento regularmente para asegurar su funcionamiento óptimo.

Estas etapas se repiten de forma continua para garantizar que el software se mantenga actualizado, seguro y cumpla con las necesidades de los usuarios. El ciclo de vida DevOps se caracteriza por la colaboración entre los equipos de desarrollo y operaciones, la automatización de tareas repetitivas y la implementación de prácticas ágiles y de entrega continua.

1.1.1.12. Infraestructura DevOps

La infraestructura DevOps se refiere a la combinación de prácticas, herramientas y tecnologías utilizadas para automatizar, gestionar y monitorizar de forma eficiente el ciclo de vida de desarrollo de software.

Esta infraestructura tiene como objetivo principal agilizar el proceso de entrega de software, garantizando la colaboración entre equipos de desarrollo y operaciones. Algunos de los elementos clave de la infraestructura DevOps incluyen la integración continua, la entrega continua, la monitorización en tiempo real y la automatización de tareas repetitivas.

Con una infraestructura DevOps sólida, las organizaciones pueden mejorar la calidad del software, reducir los tiempos de entrega y aumentar la colaboración entre equipos.

1.1.1.13. Implementación de la infraestructura DevOps cumpliendo sus principios

Automatización de procesos: El uso de herramientas y plataformas para automatizar tareas repetitivas en el ciclo de desarrollo y despliegue de software.

Orquestación de contenedores: La gestión y coordinación de contenedores virtuales, como Docker, para facilitar la implementación y escalabilidad de aplicaciones.

Gestión de configuración: La implementación de herramientas que permiten gestionar de forma eficiente la configuración de los sistemas y aplicaciones en entornos DevOps.

Monitoreo y análisis de rendimiento: La implementación de herramientas de monitoreo y análisis que permiten identificar y solucionar problemas de rendimiento en tiempo real.

Infraestructura como código: El uso de herramientas y prácticas que permiten gestionar la infraestructura de forma automatizada a través de código, como Terraform o Ansible.

Gestión de recursos en la nube: La utilización de servicios en la nube para gestionar recursos de infraestructura de forma eficiente y escalable.

Seguridad y cumplimiento normativo: La implementación de herramientas y prácticas para garantizar la seguridad de la infraestructura y cumplir con los requisitos normativos establecidos.

Integración continua y entrega continua: La implementación de prácticas y herramientas que permiten la integración y entrega continua de software de forma automatizada y eficiente.

1.2. Descripción del contexto socioeconómico y cultural en el que se realiza la investigación

La implementación de herramientas DevOps para el desarrollo de software en una empresa se ha vuelto cada vez más importante. Con la creciente competencia en el mercado, las empresas buscan formas de aumentar la eficiencia en la entrega de sus productos y servicios.

Las herramientas DevOps permiten a los equipos de desarrollo y operaciones colaborar de manera más estrecha y eficiente, lo que resulta en un ciclo de desarrollo más rápido y en la entrega de software de mayor calidad. Esto a su vez, puede ayudar a las empresas a reducir costos, mejorar la satisfacción del cliente y mantenerse competitivas en un mercado en constante evolución.

Además, la implementación de herramientas DevOps puede tener un impacto positivo en la fuerza laboral de la empresa, al fomentar una cultura de colaboración, aprendizaje continuo y mejora constante. Esto puede contribuir a mejorar la moral y el compromiso de los empleados, lo que a su vez puede tener un impacto positivo en la productividad y en la retención del talento en la empresa.

Es así que la empresa E.G.I. (Eficiencia en Gestión Informática) que se dedica a la venta de productos y servicios informáticos implementa un nuevo servicio de desarrollo de software con todas las limitantes posibles ante un campo muy competitivo en nuestro mercado local, por lo que a necesidad de implementar un sistema de administración notarial desarrollando este sistema, en la primera etapa como implementación en una red local, un software de escritorio cliente-servidor. La empresa no tiene ninguna infraestructura en la nube, por lo que solo se almacena de manera local todo el software generado, entre los cuales manejan lenguajes de programación php, groovy, java script, java, framework como Larabel, grails y otros, IDEs de desarrollo, Visual Studio Code, IntelliJ, Aptana.

Actualmente la empresa de apoco está complementando utilizando buenas prácticas DevOps como enfoque para mejorar sus servicios, se comenzarán con algunas capacitaciones en el manejo de herramientas que ayuden en diferentes etapas del ciclo de vida de desarrollo de software.

CAPITULO II

DIAGNOSTICO

2.1. Introducción

En el presente capitulo se analizó las diferentes herramientas para realizar pruebas de rendimiento haciendo el uso de en una guía de observación que categoriza los tipos de pruebas y las métricas, como también se hizo una tabla comparativa de las diferentes alternativas que se tienen para realizar dichas pruebas, para así poder elegir una o varias herramientas, proponiendo además el empleo de buenas prácticas.

Buenas prácticas:

- *Identificar escenarios realistas:* Crea escenarios de prueba que reflejen las actividades y comportamientos de los usuarios reales, simulando así situaciones que puedan surgir en producción.
- *Utilizar herramientas adecuadas:* Selecciona herramientas de prueba de rendimiento que se ajusten a las necesidades y requisitos específicos.
- *Automatizar las pruebas:* Permitiendo ejecutar pruebas de manera repetida y consistente, lo que facilita la detección de problemas y la comparación de resultados a lo largo del tiempo.
- *Incluir diferentes configuraciones de hardware y red*: Para poder simular las condiciones del mundo real. Esto incluye diferentes dispositivos, navegadores, conexiones de red y ubicaciones geográficas.
- *Monitorear y analizar métricas relevantes*: Durante las pruebas, monitorea y analiza métricas clave de rendimiento, como el tiempo de respuesta del servidor, la utilización de recursos, el tiempo de carga de la página, etc. permite identificar áreas problemáticas y oportunidades de mejora.
- *Escalar gradualmente la carga:* Incrementa gradualmente la carga en el sistema durante las pruebas para evaluar cómo responde a diferentes niveles de demanda. Esto ayuda a identificar los límites de capacidad y a planificar la escalabilidad.

2.2. Procesamiento y Análisis de Datos

Para la presente investigación se hicieron pruebas de rendimiento con diferentes herramientas conforme a ciertos parámetros los cuales se centralizó obteniendo los resultados, basados en la guía de observación cuyas variables y métricas guiaran a obtener una tabla comparativa.

Guía de observación:

Se propone una guía de observación, donde se registran varios aspectos que coadyuven al análisis y puedan captar información relevante al momento de realizar las pruebas frente al empleo de diferentes herramientas DevOps para pruebas de rendimiento.

Categorías y métricas:

Tabla Nº 1

Categorías y métricas

N°	TIPOS DE PRUEBAS DE RENDIMIENTO	METRICAS
1	Pruebas de Carga (Load Testing):	Tiempo de respuesta
2	Pruebas de Estrés (Stress Testing):	Capacidad máxima Recursos utilizados
3	Pruebas de Resistencia (Endurance Testing):	Recursos utilizados
4	Pruebas de Escalabilidad (Scalability Testing):	Capacidad máxima
5	Pruebas de Concurrencia (Concurrency Testing):	Capacidad máxima de usuarios
6	Pruebas de Velocidad (Speed Testing):	Tiempo de espera
	HERRAMIENTAS DEVOPS PARA PRUEBAS DE RENDIMIENTO	
1	JMeter	
2	Gatling	
3	Apache Bench (ab):	
4	LoadRunner	

5	VisualVM	
6	JProfiler	

Tabla 1: Categorías, tipos de pruebas, métricas y herramientas DevOps para pruebas de rendimiento

2.3. Tabulación y Codificación de datos

Gráfico Nº 1

Ejecución de 4 tipos de pruebas de rendimiento con la herramienta DevOps JMeter

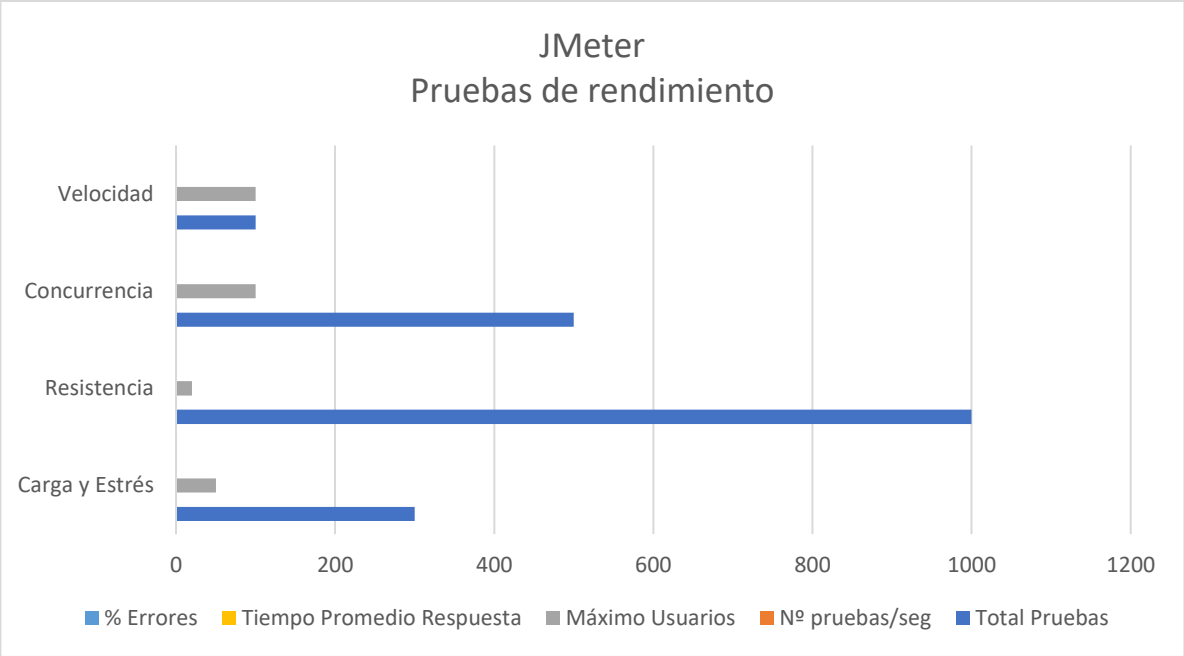

Gráfico 4 Resultado de pruebas de rendimiento con JMeter,

Pruebas de rendimiento con la herramienta DevOps JMeter, al módulo de acceso al sistema, para determinar puntos de quiebre, velocidad, máximo de usuarios, cargas y estrés. Se hizo diferentes casos de prueba hasta encontrar el punto de quiebre del software al proceso, donde se evidenció que a la concurrencia de 300 usuarios ocasiona colapso del software.

Gráfico Nº 2

Identificación de errores de memoria, cuellos de botella empleando la herramienta VisualVM

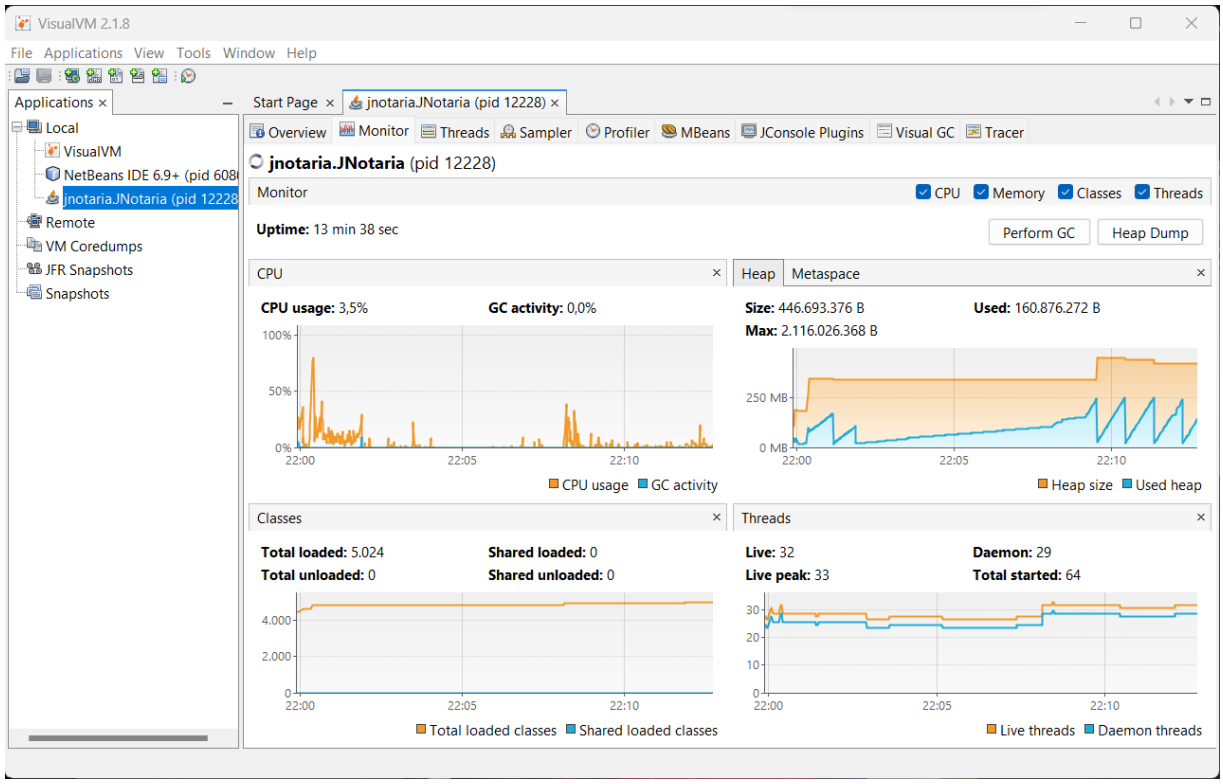

Gráfico 5 Resultado de pruebas de rendimiento con la herramienta VisualVM para cuellos de botella

Con esta herramienta se pudo determinar cuellos de botella en el flujo de datos hacia el servidor sobre todo cuando se tienen muchas peticiones que superen los 100 usuarios al mismo tiempo, como también se puedo ver cuánto de recursos consume la ampliación y en caso de equipos de bajos recursos se tiene que puede ser mayor a los 800 Mb llegando a ocupar hasta 1 Gb, no obstante, no se tiene mucho recurso de uso de procesamiento.

Matriz de comparación:

El presente cuadro comparativo corresponde a la selección de herramientas en función a diferentes características cualitativas que serán tomadas en cuenta al momento de elegir una herramienta que se adecue al proyecto que se requiere hacer las pruebas de rendimiento, no obstante aclarar también que se podría emplear más de una.

Tabla Nº 2

Matriz comparativa de herramientas DevOps para pruebas de rendimiento

Característica	JMeter	VisualVM	Apache Benchmark
Tipos de Prueba	Amplia variedad	Monitorización	Rendimiento
Lenguaje de Escritura de Escenarios	XML, GUI (interface gráfica)	Limitada	Línea de comandos
Escalabilidad y Concurrencia	Buena	Limitada	Buena
Soporte de Protocolos	HTTP, HTTPS, FTP, JDBC, JMS, etc.	JVM	HTTP, HTTPS
Flexibilidad de Configuración	Muy flexible, gran cantidad de opciones	Limitada	Limitada, enfoque simple
Comunidad y Soporte	Grande	Limitada	Menor
Integración con CI/CD	Buen soporte	No especifica	No específico
Curva de Aprendizaje	Moderada	bajo	Baja
Reportes y Análisis	Variedad de plugins y opciones de informes	HTML	Simple, orientado a línea de comandos
Costo	Código abierto	Código abierto	Código abierto

Tabla 2 Matriz comparativa de herramientas DevOps, fuente elaboración propia.

2.4. Análisis y discusión de resultados

Para la presente investigación se analizaron diferentes herramientas para realizar pruebas de rendimiento, pero muchas de ellas no se adecuan al tipo de proyecto, no obstante se exploró otras alternativas que ayuden no solo en dichas pruebas, también para la elección de la(s) herramienta(s) analizando sus características, ventajas y desventajas, soporte, flexibilidad, costo y otros, considerando también métricas como tiempos de respuesta promedios, máximo de usuarios concurrentes, capacidad de memoria, etc., al momento de realizar las pruebas.

2.5. Conclusiones

Las herramientas DevOps existentes para realizar pruebas de rendimiento en su mayoría están enfocadas a solo aplicaciones o servicios web y no así a aplicaciones de escritorio, por lo que se tuvo que hacer una amplia búsqueda, siendo JMeter y VisualVM los más adecuados para el tipo de proyecto que lleva a cabo la empresa EGI en el desarrollo del sistema de administración notarial; ambas herramientas son muy buenas en el sentido de que ayudaron en primera instancia a detectar errores en situaciones de estrés, como también el poder llevar el control del rendimiento de la aplicación en situaciones extremas, excelentes al momento de monitorizar la aplicación, presentan además diferentes tipos de reportes. Ambas herramientas son de uso gratuito y de código abierto, enfocados a sistemas basados en el lenguaje java, siendo JMeter el más complejo y mejor adaptable.

El uso de buenas prácticas en el proceso de desarrollo de pruebas de rendimiento, permitirán ayudar en gran medida ya sea en la planificación como el análisis de las herramientas, comparación de los resultados, análisis de métricas, para así poder escalar de forma gradual y realizar mejoras continuas en cada prueba.

El empleo de herramientas DevOps en el proceso de desarrollo de software notarial de la empresa EGI, para las pruebas de rendimiento son de gran utilidad ya que mediante ellas permite anticiparse a posibles errores, mejorando así la eficiencia y eficacia como también de minimizar la cantidad de errores, analizar posibles soluciones ante problemas en la etapa de desarrollo y producción.

.

2.6. Recomendaciones

Tener a mano un conjunto de herramientas DevOps, integrarlos al entorno de desarrollo para poder automatizarlos, esto permitirá ayudar de gran manera al área de desarrollo de la empresa EGI, permitiendo así producir software escalable, reduciendo tanto el tiempo de entrega como también de anticiparse a posibles fallas en la etapa de desarrollo y producción.

Implementar de forma gradual los principios de DevOps con el enfoque de: integración continua, automatización de pruebas, retroalimentación y monitorización constante, mediante el uso adecuado de herramientas que permitan así crea un mejor ambiente de desarrollo y producción.

Analizar la posibilidad de utilizar otras herramientas DevOps como contenedores, los cuales permitirán crear otros ambientes de pruebas con diferentes características.

REFERENCIAS BIBLIOGRAFICAS

(2022). Medium: https://nowitsanurag.medium.com/stress-testing-using-apache-bench-ab-98a3f1312246

AllSpaw, J. (2017). *GUÍA COMPLETA PARA PRINCIPIANTES APRENDE DEVOPS PASO A PASO.* https://books.google.com.ec/books?id=ui8hDgAAQBAJ&printsec=frontcover&dq=Dev

Arriaza, C. (2023). *youtube.* Clase 4 Pruebas de rendimiento: https://www.youtube.com/watch?v=57ePo_9dRHo

AWS. (2024). *Amazón Web Service: .* ¿Qué es DevOps?: https://aws.amazon.com/es/devops/what-is-devops/

Azure Microsoft. (2024). https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-devops#culture

Gallego Durango, W. A. (2022). *Investigación y desarrollo de pruebas de Rendimiento automatizadas.*

Gatling Corp . (2024). *Gatling.* https://gatling.io/

Hamilton, T. (2024). Cómo utilizar JMeter para pruebas de carga y rendimiento: https://www.guru99.com/es/jmeter-performance-testing.html

JMeter. (2024). Apache JMeter: https://jmeter.apache.org/

Loadview. (2024). La guía definitiva de JMeter: https://www.loadview-testing.com/es/la-guia-definitiva-de-jmeter-tutorial-de-pruebas-de-carga-y-rendimiento/

Olmedo Rodrígues, Á. 2. (2018). *DevOps y análisis de performance automáticos.*

Olmedo Rodríguez, Á. (2018). DevOps y análisis de performance automáticos.: https://openaccess.uoc.edu/handle/10609/71505

RedHat, "Cultura DevOps". (10 de Mayo de 2022). https://www.redhat.com/es/topics/devops#cultura

Sánchez Salinas, Á. H. (2023). https://cybertesis.unmsm.edu.pe/handle/20.500.12672/21039

Vecillas, M. (2020). *Herramientas DevOps, definicion de procesos en el desarrollo de Software.* Maracaibo: EDICIONES PAIDÓS.

Villamarín, A. É. (2019). *Introducción a DevOps para la mejora de los procesos de desarrollo con herramientas Open Source.*

ANEXOS

Guía de observación

El diseño del instrumento propuesto:

GUIA DE OBSERVACIÓN	
Tipo de prueba- Objetivo:	[Evaluar el rendimiento de la aplicación bajo diferentes condiciones de carga y estrés.]
Detalles del Entorno de Prueba	
Aplicación bajo Prueba:	[Nombre de la aplicación]
Versión de la Aplicación:	[Versión]
Configuración del Entorno:	[Detalles de hardware, software y red]
Herramientas de Prueba:	[Lista de herramientas]
Condiciones de la Prueba	
Carga de Usuarios:	[Número de usuarios simulados]
Escenario de Prueba:	[Descripción del escenario de prueba, acciones que realizan los usuarios]
Duración de la Prueba:	[Tiempo de ejecución de la prueba]
Observaciones:	

Resultados	
Estado	[Métricas] /[Errores] / [Cuellos de botella]

GUIA DE OBSERVACIÓN N°1	
Tipo de prueba-Objetivo:	Pruebas de carga y estrés: Evaluar cómo se comporta la aplicación cuando se somete a una carga más allá de sus límites normales. El objetivo es identificar el punto de quiebre de la aplicación y cómo se recupera después de un estrés excesivo.
Detalles del Entorno de Prueba	
Aplicación bajo Prueba:	Sistema de Administración Notarial (JNotaría)
Versión de la Aplicación:	2
Configuración del Entorno:	Servidor Linux Ubuntu v22, cliente Windows 10, red local.
Herramienta de Prueba:	JMeter Gatling Apache Bench (ab): LoadRunner VisualVM JProfiler
Condiciones de la Prueba	
Carga de Usuarios:	20 - 50 – 100 - 200 usuarios
Escenario de Prueba:	Varios usuarios acceden al sistema al mismo tiempo
Duración de la Prueba:	10 minutos
Observaciones:	
Resultados	
Estado	

GUIA DE OBSERVACIÓN N°2	
Tipo de prueba-Objetivo:	Pruebas de Resistencia (Endurance Testing): Evaluar cómo se comporta la aplicación durante un período prolongado de tiempo bajo una carga constante. El objetivo es identificar posibles problemas de memoria, fugas de recursos o degradación del rendimiento a largo plazo.
Detalles del Entorno de Prueba	
Aplicación bajo Prueba:	Sistema de Administración Notarial (JNotaría)
Versión de la Aplicación:	2
Configuración del Entorno:	Servidor Linux Ubuntu v22, cliente Windows 10, red local.
Herramienta de Prueba:	JMeter Gatling Apache Bench (ab): LoadRunner VisualVM JProfiler
Condiciones de la Prueba	
Carga de Usuarios:	20
Escenario de Prueba:	Solicitar lista de servicios generados en el día
Duración de la Prueba:	2 horas
Observaciones:	
Resultados	
Estado	

GUIA DE OBSERVACIÓN N°3	
Tipo de prueba-Objetivo:	Pruebas de Concurrencia (Concurrency Testing): Evaluar cómo se comporta la aplicación cuando múltiples usuarios acceden y realizan operaciones simultáneamente. El objetivo es identificar posibles problemas de concurrencia, como condiciones de carrera y bloqueos de recursos compartidos.
Detalles del Entorno de Prueba	
Aplicación bajo Prueba:	Sistema de Administración Notarial (JNotaría)
Versión de la Aplicación:	2
Configuración del Entorno:	Servidor Linux Ubuntu v22, cliente Windows 10, red local.
Herramienta de Prueba:	JMeter Gatling Apache Bench (ab): LoadRunner VisualVM JProfiler
Condiciones de la Prueba	
Carga de Usuarios:	100
Escenario de Prueba:	Se somete a prueba de ingres al sistema en el módulo Login
Duración de la Prueba:	10 minutos
Observaciones:	
Resultados	
Estado	