DESARROLLO DE UN SISTEMA DE MONITOREO DE MÉTRICAS DE RENDIMIENTO EN APLICACIONES EN LA NUBE CON PLATAFORMAS TIPO SERVERLESS CONTAINERS Y KUBERNETES

Julian Emilio Osorio Larroche

Nota de Aceptación

Certificamos que el presente Trabajo de Grado Satisface, en alcances y calidad, todos los requisitos que demanda un Trabajo de Grado de Maestría.

Juan Pablo García Cifuentes
Director

Jurado

Jurado

Aprobado en cumplimiento de los requisitos exigidos por la Pontificia Universidad Javeriana Cali, para optar el título de Magister en ingeniería de software .

HERNÁN CAMILO ROCHA NIÑO Ph. D.
Decano Facultad de Ingeniería y Ciencias

JUAN CARLOS MARTÍNEZ ARIAS
Director Posgrados de Ingeniería y Ciencias

Ciudad y fecha 18/05/2023

Acta de Correcciones al Documento de Trabajo de Grado

Santiago de Cali, 18/05/2023

Autor: Julian Emilio Osorio Larroche

Título del Trabajo de Grado: "Desarrollo De Un Sistema De Monitoreo De Métricas De Rendimiento En Aplicaciones En La Nube Con Plataformas Tipo Serverless Containers Y Kubernetes"

Director:

Como indica el artículo 2.13 de las Directrices para Trabajo de Grado de Maestría, he verificado que el estudiante indicado arriba ha implementado todas las correcciones que los Jurados del Proyecto de Trabajo de Grado definieron que se efectuaran, como consta en el Acta de Evaluación correspondiente.

Juan Pablo García Cifuentes
Director del Trabajo de Grado

Santiago de Cali, 12 de 12 del 2022

Ingeniera:
Luisa Rincón
Director Posgrados de Ingeniería
Facultad de Ingeniería
Pontificia Universidad Javeriana Cali

Cumplido los requisitos establecidos en los artículos 5.6 y 5.7 de las Directrices para Trabajo de Grado de Maestría, solicitamos se autorice la sustentación del Trabajo de Grado denominado "Sistema de monitoreo de métricas de rendimiento en aplicaciones en la nube con plataformas tipo Serverless Containers y Kubernetes" realizado por el (la) estudiante Julian Emilio Osorio Larroche con código 0057917 perteneciente al énfasis en Ingeniería Software, bajo la dirección del profesor Juan Pablo García Cifuentes.

El suscrito director del Trabajo de Grado autoriza para que se proceda a hacer su sustentación ante el Tribunal que para el efecto se designe, toda vez que ha revisado meticulosamente el documento y avala que el Trabajo de Grado ya se encuentra listo para ser evaluado oficialmente.

Atentamente,

Julian Emilio Osorio Larroche	Juan Pablo García Cifuentes .
C.C. 1144037886 de Cali	C.C. 1.112.759.052 de Cartago

Nombres: Julian Emilio Osorio Larroche.

Dirección: Carrera 24c #2-56 oeste edificio Bahía apt 502, Cali, Colombia.

Celular: 3154687982.

Correo académico: jeosorio@javerianacali.edu.co.

Correo electrónico: julianosoriolarroche@gmail.com.

Título: Ingeniero de sistemas.

Universidad: Pontificia Javeriana Cali.

Empresa: GreenSQA.

Cargo: Ingeniero de investigación aplicada.

FICHA RESUMEN
ANTEPROYECTO DE TRABAJO DE GRADO DE MAESTRÍA

TITULO: "Sistema de monitoreo de métricas de rendimiento en aplicaciones en la nube con plataformas tipo Serverless Containers y Kubernetes".

1. ÉNFASIS: Ingeniería de software
2. ÁREA DE INVESTIGACIÓN: Innovación
3. ESTUDIANTE: Julian Emilio Osorio Larroche
4. CORREO ELECTRÓNICO: jeosorio@javerianacali.edu.co
5. DIRECTOR: Juan Pablo García Cifuentes
6. PALABRAS CLAVE: Rendimiento, Computación Nube, Serverless, Contenedores, Kubernetes
7. CÓDIGOS UNESCO CIENCIA Y TECNOLOGÍA: 5306.02
8. FECHA DE INICIO: 14 de 12 de 2020 DURACIÓN ESTIMADA: 6 Meses

RESUMEN

El hecho de exponer en internet un software que no tenga buenas pruebas de desempeño hace que las organizaciones aumenten el riesgo de fallar en producción y en consecuencia, pierdan valor a causa de la baja satisfacción del usuario. Para valorar la calidad de un software, no basta con saber que un requerimiento de negocio o caso de uso funcional completó lógica y coherentemente una secuencia de pruebas definida, también debemos ser capaces de juzgar la calidad de manera integral, incluyendo el esfuerzo físico de los recursos disponibles (servidores en la nube) para soportar la demanda de las solicitudes.
GreenSQA construye GreenHeart/AIMaps, plataforma capaz de ejecutar pruebas de cargas, capturar y enviar muestras de rendimiento tras hacia un modelo centralizado de Big Data. En infraestructura tradicional (RAM, CPU, disco, red, etc.). Sin embargo, actualmente GreenSQA, no cuenta con tecnologías que permitan capturar información de métricas de aplicaciones Web que operan en plataformas tipo Serverless Containers y Kubernetes.

En este proyecto se propone la creación de sistema que permita la automatización y facilite monitoreo de métricas de rendimiento en aplicaciones Web que operan en plataformas tipo Serverless Containers y Kubernetes para diferentes clientes de GreenSQA.

RESUMEN

Durante los últimos años, ha habido un crecimiento exponencial del acceso a la tecnología web en Colombia. Esto ha implicado a su vez, una migración masiva de los negocios y las empresas al uso de este tipo de tecnologías. No obstante lo anterior, existen, como es natural, muchos fallos, errores y riesgos en las mismas; sobre todo en lo referente a las arquitecturas de software, que pueden derivar en pérdidas económicas, toda vez que la satisfacción de los usuarios con las plataformas disminuye siempre que estas no cumplen con los requisitos de calidad para su despliegue adecuado. Este contexto explica la necesidad de ejecutar pruebas de rendimiento que ayuden a detectar y corregir los errores y fallos, de manera temprana.

En el presente trabajo se toma como punto de inicio GreenSQA, una plataforma que permite ejecutar este tipo de pruebas. Sin embargo, actualmente la plataforma no cuenta con la tecnología para ejecutarlas en los servicios en la nube tipo Kubernetes y *Serverless Containers*, de forma automatizada y en tiempo de ejecución.

Pensando en lo anterior, se presenta aquí el desarrollo de la creación de un sistema que permita la automatización y facilite el monitoreo y la visualización de métricas de rendimiento en aplicaciones web que operan en plataformas tipo *Serverless* Containers y Kubernetes, para diferentes clientes de GreenSQA.

En la investigación se realizó un exhaustivo análisis del funcionamiento de la tecnología Kubernetes (los servicios que ofrece y su forma de operar). Se identificaron los proveedores de servicios de Kubernetes y se les realizaron pruebas de conceptos. Así mismo, se identificaron los indicadores de salud del clúster con el fin de seleccionar las métricas más relevantes. Se desarrolló un paquete Helm Chart y se desarrollaron cuatro niveles de tableros para monitorear y visualizar las métricas. Se realizó con ello una plataforma sobre la que se realizaron nuevas pruebas de performance; logrando así desplegar el software propuesto.

ABSTRACT

During the last few years, there has been an exponential growth in access to web technology in Colombia. This has implied, in turn, a massive migration of businesses and companies to the use of this type of technology. Notwithstanding the foregoing, there are, of course, many failures, errors and risks in them; especially in relation to software architectures, which can lead to economic losses, since user satisfaction with the platforms decreases whenever they do not meet the quality requirements for proper deployment. This context explains the need to run performance tests that help detect and correct errors and failures early.

In the present work, GreenSQA is taken as a starting point, a platform that allows to execute this type of tests. However, currently the platform does not have the technology to execute them in cloud services such as Kubernetes and Serverless Containers, in an automated manner and at runtime.

Thinking about the above, the development of the creation of a system that allows automation and facilitates the monitoring and visualization of performance metrics in web applications that operate on Serverless Containers and Kubernetes-type platforms, for different GreenSQA clients, is presented here.

In the investigation, an exhaustive analysis of the operation of Kubernetes technology (the services it offers and its way of operating) was carried out. Kubernetes service providers were identified and proof of concepts performed. Likewise, the health indicators of the cluster were identified in order to select the most relevant metrics. A Helm Chart package was developed and four levels of dashboards were developed to monitor and visualize the metrics. With this, a platform was created on which new performance tests were carried out; thus managing to deploy the proposed software.

DESARROLLO DE UN SISTEMA DE MONITOREO DE MÉTRICAS DE RENDIMIENTO EN APLICACIONES EN LA NUBE CON PLATAFORMAS TIPO SERVERLESS CONTAINERS Y KUBERNETES

Julián Emilio Osorio Larroche
Código 0057917

Anteproyecto de trabajo de grado para optar al título de Magister en Ingeniería de Software

Director
Juan Pablo García Cifuentes

FACULTAD DE INGENIERÍA Y CIENCIAS
MAESTRÍA EN INGENIERÍA DE SOFTWARE
SANTIAGO DE CALI, DICIEMBRE DE 2022

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Todo software es susceptible a contener fallos y errores. De no ser identificados y corregidos a tiempo, dichos fallos pueden desencadenar en escenarios catastróficos para el entorno empresarial, que se expresan, en última medida, en la perdida confianza del público, en grandes pérdidas de dinero o incluso, en casos extremos, en pérdidas humanas. Entre más tarde se detecten los defectos o errores, mayores pueden ser las consecuencias. De ahí la necesidad de elaborar mecanismos para la detección y corrección temprana. En este sentido, las pruebas de calidad de software son las encargadas de analizar e identificar las fallas en sistemas informáticos al momento de entregar productos con la calidad esperada, y, por tanto, son las que permiten prevenir los riesgos a futuro.

Esto quiere decir que el hecho de exponer en internet un software que no tenga buenas pruebas de desempeño hace que las organizaciones aumenten el riesgo de fallar en producción y, en consecuencia, pierdan productividad y ventas, a causa de la baja satisfacción del usuario. Ahora bien, para valorar la calidad de un software, no basta con saber que un requerimiento de negocio (o un caso de uso funcional) completó lógica y coherentemente una secuencia de pruebas definida; también debemos ser capaces de juzgar la calidad de manera integral, incluyendo el esfuerzo físico de los recursos disponibles (servidores en la nube) para soportar la demanda de las solicitudes. Las normas ISO 25000, que establecen el sistema para la evaluación de la calidad del producto, definen ocho atributos de calidad, entre los que se resaltan, para el presente trabajo: correctitud funcional, seguridad, compatibilidad y eficiencia de desempeño. Siendo este último la característica que representa el desempeño relativo y cantidad de recursos utilizados bajo determinadas condiciones. Los tiempos de respuesta y procesamiento de un sistema cuando lleva a cabo sus funciones bajo condiciones determinadas y límites máximos en que cumplen con los requisitos.

GreenSQA construye GreenHeart/AIMaps, una plataforma capaz de ejecutar pruebas de cargas, capturar y enviar muestras de rendimiento hacia un modelo centralizado de Big Data, en infraestructura tradicional (RAM, CPU, disco, red, etc.). Sin embargo, GreenSQA, no contaba con tecnologías que permitieran capturar información de métricas de aplicaciones Web que operan en plataformas tipo *Serverless Containers* y Kubernetes. En este proyecto se propuso la creación de un sistema que permita la automatización y facilite monitoreo de métricas de rendimiento en aplicaciones Web que operan en este tipo de plataformas. Para el desarrollo, el trabajo se divide en dos partes: en la primera se exploran las diferentes tecnologías que puedan servir para este fin, sus características y funcionamiento; sobre las que se aplican pruebas de rendimiento, de acuerdo con las métricas que denotan la salud de los clústeres. Y en la segunda, se desarrolla un paquete Helm Chart para captar y centralizar la información de las métricas y se investiga y desarrolla la plataforma para visualizar y monitorear las mismas.

1 DEFINICIÓN DEL PROBLEMA

1.1 Planteamiento del problema o necesidad

La velocidad vertiginosa de aparición de teléfonos móviles en Colombia ha impulsado un creciente acceso a la web por parte de la población del país. Las cifras del boletín trimestral de las TIC para el 2019 [1], indican que el total de accesos fijos a internet alcanzó los 6.8 millones y que, al término del primer trimestre de 2019, los accesos móviles llegaron a 28.3 millones. Este comportamiento ha forzado a las empresas a migrar parte de su energía a operar en la nube.

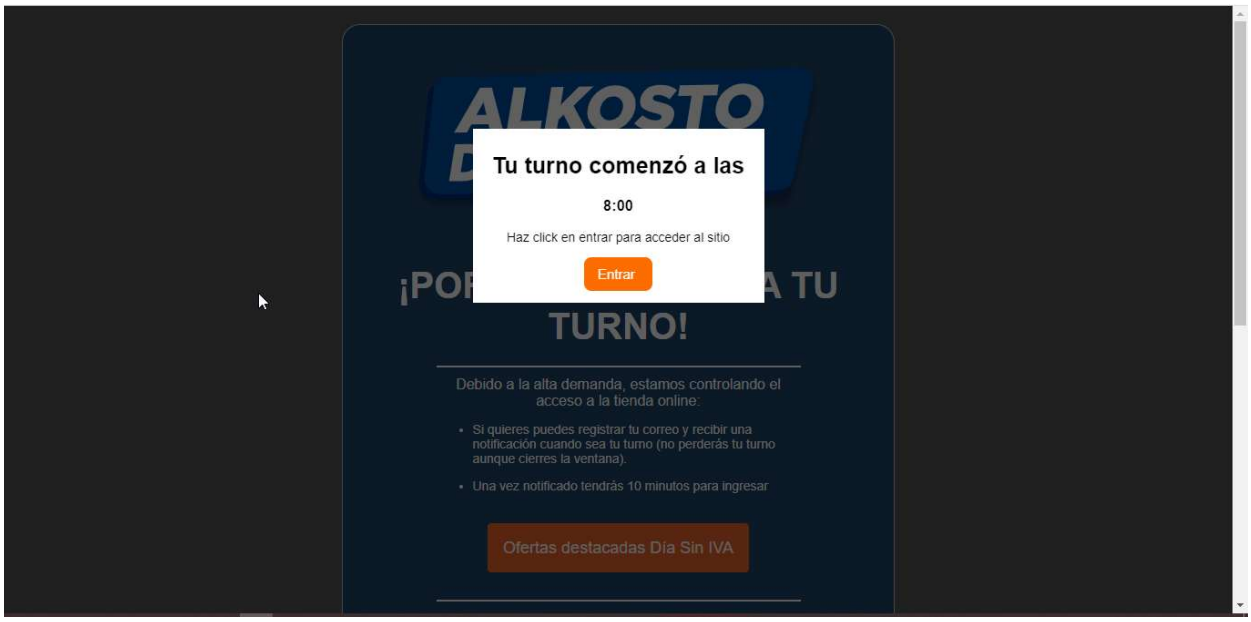

Imagen 1. Fallas en el servicio de Alkosto durante el día sin IVA

El mercado colombiano es uno de los más avanzados de Suramérica [2]; no obstante, se ha evidenciado que las plataformas de *e-commerce* presentan falencias de desempeño cuando hay eventos promocionales que movilizan muchos usuarios a sus portales (como el día sin IVA del 3 de julio del 2020). Tales falencias son causadas, en gran medida, por fallidas arquitecturas de software o por la no utilización de infraestructuras elásticas. Esto se traduce en la pérdida de posibles ventas cuando sus sistemas quedan fuera de línea; además, muchos *e-commerce* implementan estrategias poco amigables, como *filas virtuales*, en las que los usuarios no están dispuestos a esperar. Y demuestra, por tanto, la necesidad cada vez mayor de realizar pruebas de rendimiento a los sistemas de las empresas.

Para este efecto, en GreenSQA se realizó el Primer Reporte de Calidad 2018-19 [3], un significativo estudio dirigido a conocer el estado de la calidad de software en Colombia, basado en una encuesta aplicada a más de 350 profesionales de TI en seis sectores

económicos a nivel nacional: servicios financieros, salud y solidario, servicios públicos, tecnologías de la información, educación, manufactura e industrial. Los resultados del estudio evidenciaron que, en la actualidad, el principal objetivo de las organizaciones nacionales (el 77% de las encuestadas) para realizar pruebas a sus sistemas es "asegurar la satisfacción del usuario final" y el 69% indicó que en sus organizaciones se realizan pruebas a sitios web.

A grandes rasgos, como describe la norma ISO/IEC 9126, la calidad del software se puede evaluar de acuerdo con seis características: funcionalidad, confiabilidad, usabilidad, capacidad de mantenimiento, portabilidad y eficiencia. La última característica trata de la relación entre el nivel de rendimiento del software y el volumen de recursos utilizado, sumado al comportamiento de los tiempos requeridos para ejecutar las funcionalidades probadas bajo ciertas condiciones. Para las aplicaciones web —incluidos los servicios web— las pruebas de volumen, carga, concurrencia y estrés definen escenarios de simulación de usuarios virtuales, datos y metas de rendimiento, que permiten valorar el comportamiento de un sistema de información sometido a escenarios de arribo masivo de usuarios; de tal forma que en la ejecución de este tipo de pruebas se simulan, por ejemplo, entre dos mil y 300 mil usuarios simultáneos o en concurrencia por hora. Según la experiencia de GreenSQA, durante las pruebas mencionadas se evidencian problemas como: lentitud, caídas del servicio, pérdida de transacciones, despliegues erróneos en la infraestructura etc. Incurriendo en una mala experiencia de usuario y afectando negativamente la operación y percepción del negocio.

Parte de las causas de lo anterior radican en la complejidad que acarrea ejecutar y evaluar los resultados de las pruebas de desempeño de las aplicaciones web, incluidos los servicios web durante el desarrollo; ante la llegada de miles usuarios al mismo sitio se pueden experimentar demoras y comportamientos anormales. Por otra parte, los proyectos de desarrollo pocas veces hacen pruebas de desempeño y si las hacen, éstas se incluyen en las etapas finales, justo antes de desplegar las aplicaciones web al ambiente de producción. Adicionalmente, dada la superficie de desempeño a evaluar para realizar las pruebas se requiere la colaboración de ingenieros expertos en *front-end, back-end* e infraestructura; sin embargo, el acceso a especialistas en pruebas de rendimiento resulta ser escaso y costoso (el salario de un experto en Kubernetes puede ascender a 200.000 dólares anuales).

Por lo anterior, y considerando que hoy en día los sistemas software son cada vez más complejos, se hace imperativo emplear técnicas y herramientas de monitoreo que permitan observar y revisar las propiedades y atributos de calidad de los sistemas en tiempo de ejecución; es decir, cuando están interactuando con otros sistemas, hardware y con el entorno [4]. El monitoreo es una actividad fundamental en la ingeniería de software y también un prerrequisito para controlar y mejorar sistemáticamente la calidad del software desarrollado [5]. Implementar un monitoreo efectivo podría permitir que las organizaciones conozcan los problemas de sus sistemas antes de que lo hagan sus clientes; además, proporcionaría defensa adicional contra fallos catastróficos y brindaría

soporte a las pruebas, al exponer recurrentemente información sobre el estado del sistema de software [6] [7].

Pensando en lo anterior es pertinente resaltar que GreenSQA construyó AiMaps (una herramienta de automatización de pruebas que con un diseñador que corre sobre Windows) permite modelar y ejecutar robots de pruebas de software con capacidad de simular usuarios finales sobre un sistema de información y que se ha utilizado ya en importantes proyectos para clientes nacionales e internacionales. Teniendo en cuenta lo expuesto en párrafos antes, se considera que AiMaps es una pieza fundamental para abordar los problemas expuestos. Adicionalmente, el equipo de Investigación y Desarrollo de GreenSQA construyó GreenHeart, una plataforma compatible con AiMaps, capaz de ejecutar cargas, capturar y enviar muestras hacia un modelo centralizado de Big Data. Las muestras de rendimiento son de *Front-end* RUM (*Real User Monitoring*), backend (procesos funcionales), infraestructura tradicional (RAM, CPU, disco, red, etc.).

Las aplicaciones Web modernas, operan en plataformas tipo *Serverless Containers* y Kubernetes. Sin embargo, GreenSQA, no contaba con tecnologías que permitieran capturar información de métricas de aplicaciones Web que operan en plataformas tipo *Serverless Containers* y Kubernetes, por ejemplo:

- El número de nodos de trabajo con errores en el clúster.
- El número total de nodos de trabajo en el clúster.
- El porcentaje de unidades de CPU que están reservadas para los componentes de nodos, como Kubelet, Kube-proxy y Docker.
- El número de unidades de CPU que se utilizan en los nodos del clúster.
- El porcentaje de memoria que se utiliza actualmente en los nodos del clúster.
- El número de contenedores en ejecución por nodo en un clúster.
- El número de bytes por segundo que se están recibiendo a través de la red por el pod.

Ahora bien, según Gartner, para el año 2022 más del 75% de las organizaciones tendrán aplicaciones en contenedores en producción, en comparación con el escenario del año 2019 donde el uso de estos desarrollos estaba por debajo del 30%. Por lo tanto, este trabajo es pertinente porque permite extender el módulo de colectores de rendimiento y crear un sistema para la automatización de pruebas de rendimiento y facilita el monitoreo de métricas de rendimiento en aplicaciones Web que operan en plataformas tipo *Serverless Containers* y Kubernetes en los principales proveedores de nube pública para diferentes clientes de GreenSQA.

1.2 Formulación

¿Cómo desarrollar un sistema de monitoreo de métricas de rendimiento de aplicaciones Web que operan en plataformas tipo *Serverless Containers* y Kubernetes?

1.3 Sistematización

Interrogantes derivados de la formulación: ¿Cuáles son las principales plataformas tecnologías para aplicaciones virtualizadas que operan en la nube de tipo *Serverless Containers* y Kubernetes?, ¿Cómo colectar, capturar y almacenar métricas de rendimiento sobre las principales plataformas tecnologías para aplicaciones virtualizadas que operan en la nube de tipo *Serverless Containers* y Kubernetes?, ¿Cómo presentar y organizar las métricas captaras para el análisis y extracción de información?

2 OBJETIVOS DEL PROYECTO

2.1 Objetivo General

Desarrollar un sistema de monitoreo de métricas de rendimiento en aplicaciones Web que operan en plataformas tipo *Serverless Containers* y Kubernetes.

2.2 Objetivos específicos

- Seleccionar las principales plataformas tecnológicas para aplicaciones virtualizadas que operan en la nube de tipo *Serverless Containers* y Kubernetes.
- Desarrollar colectores de métricas de rendimiento sobre las principales plataformas tecnológicas para aplicaciones virtualizadas que operan en la nube de tipo *Serverless Containers* y Kubernetes.
- Diseñar tablero especializado para visualizar gráficos que expresen fácilmente el comportamiento de las métricas entregadas por los colectores tomadas del despliegue de la aplicación bajo pruebas.

3 ALCANCE

En el proyecto que se propone aquí, se construirá un sistema colector de métricas de rendimiento sobre las dos principales plataformas tecnologías para aplicaciones virtualizadas que operan en la nube, de tipo *Serverless Containers* y Kubernetes. Se reportará en una plataforma web que permita visualizar gráficos que expresen fácilmente el comportamiento de los colectores tomados del despliegue de la aplicación bajo

pruebas en tiempo real. Como mínimo los colectores desarrollados tomarán las siguientes métricas de rendimiento:

- El número de nodos de trabajo con errores en el clúster.
- El número total de nodos de trabajo en el clúster.
- El porcentaje de unidades de CPU que están reservadas para los componentes de nodos, como Kubelet, Kube-proxy y Docker.
- El número de unidades de CPU que se utilizan en los nodos del clúster.
- El porcentaje de memoria que se utiliza actualmente en los nodos del clúster.
- El número de contenedores en ejecución por nodo en un clúster.
- El número de bytes por segundo que se están recibiendo a través de la red por el pod.

4 MARCO TEORICO

4.1 Bases teóricas

4.1.1 Pruebas de rendimiento

La prueba de rendimiento es el procedimiento para encontrar errores. Dos claves para conseguir una aplicación exitosa son: el rendimiento y la funcionalidad. La funcionalidad se refiere al tipo de función (o tarea) que puede realizar una aplicación, mientras que el rendimiento se refiere a la capacidad de la aplicación para completar la tarea en condiciones de alta interacción de múltiples usuarios o recursos de hardware restringidos[8].

La monitorización y el registro del nivel de rendimiento durante el estrés bajo, alto o regular se realiza mediante pruebas de rendimiento. El objetivo de las pruebas de rendimiento es garantizar el rendimiento (capacidad de servicio y tiempo de respuesta) del sistema[8].

La prueba de carga es uno de los criterios de prueba importantes en esta prueba donde podemos observar la respuesta del sistema, de acuerdo con diferentes escenarios de tráfico (bajo, medio y alto)[8].

Para asegurarse de que las aplicaciones cumplan con los requisitos de rendimiento en la nube, los ingenieros de software deben determinar su rendimiento y su fluctuación dentro de la nube. Tradicionalmente, determinar el rendimiento de una aplicación requiere pruebas de rendimiento, en las que se ejecutan casos de prueba para cubrir los factores que pueden afectar el rendimiento. Sin embargo, en un entorno de nube, se deben considerar factores adicionales, como la programación de la máquina virtual (VM) y la tenencia múltiple (en la que el hardware se comparte entre diferentes aplicaciones y VM. Estos factores afectan el rendimiento, provocando incertidumbre[9] y retos a en la ejecución de las pruebas.

4.1.2 Monitoreo del rendimiento de aplicaciones

El monitoreo de rendimiento de aplicaciones (*Application Performance Monitoring - APM*) es una tecnología muy utilizada actualmente para verificar continuamente el rendimiento y la disponibilidad de las aplicaciones[10].

El rendimiento de las aplicaciones de software modernas se ha convertido en un requerimiento no-funcional crítico. Un incidente de rendimiento en la aplicación puede ser muy costoso para las empresas, ya que muchos usuarios y clientes confían en la

disponibilidad de los servicios. A menudo, un servicio afectado disminuye la productividad de procesos relevantes para el negocio o los servicios ofrecidos y puede causar una pérdida significativa de dinero, tiempo o daños a la imagen de la empresa.[10].

Desafortunadamente, debido al incremento de la complejidad de las aplicaciones y al incremento del número de usuarios que interactúan con estas, mantener un nivel apropiado de rendimiento se ha convertido en un desafío [11].

Las aplicaciones en la nube han introducido tanto oportunidades como desafíos. No solo es necesario probar la precisión de la aplicación en la nube, sino también su rendimiento. Satisfacer estos requisitos de rendimiento en las nubes públicas es un gran desafío porque el rendimiento de las aplicaciones que se ejecutan en la nube adolece de una incertidumbre considerable.

4.1.3 Plataformas Serverless

En los últimos años, la abstracción *Serverless* ha ganado un terreno significativo dentro de la industria de TI. Google, Microsoft y AWS ofrecen ahora implementaciones *Serverless* con características equivalentes, como parte de sus ofertas basadas en la nube; y los arquitectos de soluciones de toda la industria están utilizando la tecnología sin servidor como parte de los sistemas empresariales de misión crítica[12].

La computación *Serverless* es un nombre poco apropiado. Si bien el servidor no está expuesto de manera significativa al desarrollador, el código se está ejecutando en hardware físico de la misma manera que lo ha hecho durante décadas. Donde los servicios *Serverless* realmente difieren de los enfoques más tradicionales, y de hecho de las primeras ofertas basadas en la nube, es en que la potencia de cómputo solo se aprovisiona, y el código solo se ejecuta, cuando se activa por algún evento externo. Esto tiene varias ventajas, la facturación solo ocurre mientras se ejecuta el código, los procesos se pueden escalar horizontalmente casi indefinidamente y el concepto de funcionalidad discreta autónoma se presta de forma adecuada a la tendencia actual hacia diseños de sistemas basados en micro y nano servicios[12].

4.1.4 Contenedores

Es una de esas tecnologías centrales que está ganando cada vez más terreno en los últimos años. La virtualización basada en contenedores (*containers*), o *contenerización*, se refiere a una metodología para encapsular una aplicación junto con sus dependencias; de tal manera que pueda ejecutarse de manera eficiente en diferentes entornos informáticos.

En comparación con las técnicas tradicionales de virtualización (es decir, soluciones basadas en hipervisor) donde los controladores de dispositivos y hardware se virtualizan mediante una capa de virtualización, la virtualización basada en contenedores ofrece una alternativa más ligera: en lugar de ejecutar una máquina virtual (VM) completa con su propio sistema operativo (SO) sobre el hardware virtualizado, los contenedores proporcionan un entorno aislado para que los recursos del sistema (como por ejemplo los procesos, sistemas de archivos y redes) se ejecuten en el nivel del sistema operativo del host. Esto significa que varios contenedores pueden compartir el mismo *kernel* del sistema operativo y, por lo tanto, comenzar mucho más rápido y usar solo una fracción de la memoria del sistema; pequeña en comparación con el arranque de un sistema operativo virtualizado completo. Varios estudios han demostrado que este enfoque da como resultado gastos generales más bajos, y se optimiza el rendimiento general de la aplicación en contenedores. Además, debido a que las aplicaciones se dividen en unidades pequeñas e independientes, generalmente denominadas microservicios, aumenta la escalabilidad y portabilidad de los contenedores[13].

Ventajas

- Una forma de empaquetar aplicaciones con todas las dependencias y configuraciones necesarias.
- Artefacto portátil que se puede compartir y mover fácilmente.
- Hace que el desarrollo y la implementación sean más eficientes.
- Un proceso aislado.
- Aprovecha el sistema operativo.
- Desconocer el entorno externo.
- Accede solo a sus propios recursos.

Máquinas virtuales vs contenedores

- Un contenedor es más rápido que una máquina virtual porque utiliza el mismo sistema operativo del equipo anfitrión (host).
- Los contenedores de Linux se pueden usar en Windows, pero solo con Windows 10 Pro con Hypervisor.
- Un contenedor tiene su propio sistema de archivos y desconoce el sistema de archivos del host.

4.1.5 Docker

Docker es una herramienta de software que posibilita la creación, evaluación y despliegue ágil de aplicaciones. Docker empaqueta el software en unidades estandarizadas

(contenedores) que incluyen todo lo necesario para que el software se ejecute; incluidas bibliotecas, herramientas de sistema, código y tiempo de ejecución[14].

Arquitectura de Docker

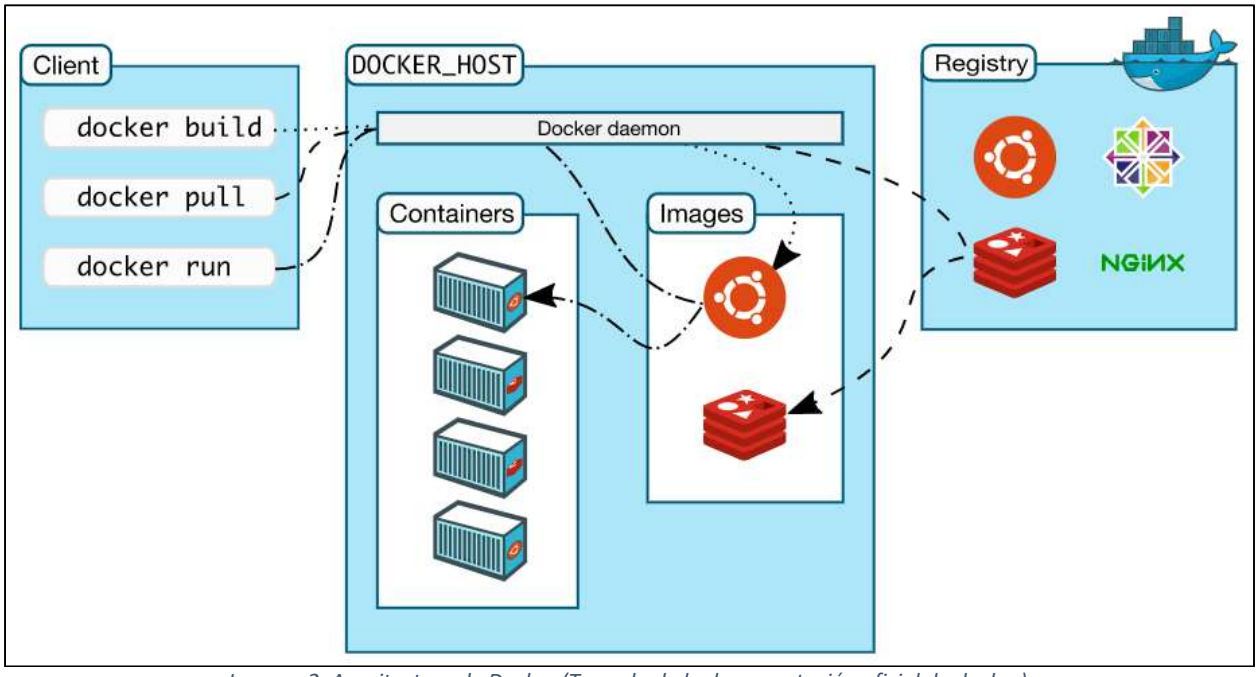

Imagen 2. Arquitectura de Docker (Tomado de la documentación oficial de docker)

Docker utiliza una arquitectura cliente-servidor. El cliente de Docker se comunica con el demonio de Docker (*Docker daemon*), que hace el trabajo de construir, ejecutar y distribuir los contenedores. El cliente y el demonio de Docker se comunican mediante una API REST, a través de sockets UNIX o una interfaz de red. Otro cliente de Docker es Docker Compose, que permite trabajar con aplicaciones que constan de un conjunto de contenedores[14]. A continuación, se explica cada parte.

El demonio de Docker

El demonio de Docker (*dockerd*) escucha las solicitudes del API de Docker y administra objetos de Docker como imágenes, contenedores, redes y volúmenes. Además puede comunicarse con otros demonios para gestionar los servicios de Docker [14].

El cliente de Docker

El cliente de Docker es la forma principal en la cual se interactúa con Docker. Cuando se utilizan comandos, el cliente envía estos comandos a *dockerd*, que los ejecuta mediante el API de Docker [14].

Registros de Docker

Un registro de Docker almacena imágenes de contenedores Docker. Docker Hub es un registro público de imágenes de contendedores Docker. Docker está configurado para buscar imágenes en Docker Hub de forma predeterminada. Sin embargo, es también posible usar otros registros de docker. Cuando se usan los comandos *docker pull* o *docker run*, las imágenes requeridas se extraen del registro configurado. Y cuando usa el comando *docker push*, la imagen se envía al registro configurado[14].

Imágenes

Una imagen es una plantilla de solo lectura con las instrucciones necesarias para crear un contenedor de Docker. A menudo, una imagen se basa en otra imagen, con alguna personalización adicional. Se pueden crear imágenes personalizadas o se pueden usar imágenes que se encuentren publicadas en un registro de Docker.

Para construir una imagen propia, se crea un *Dockerfile*, que cuenta con una sintaxis simple donde se definen los pasos necesarios para la creación y ejecución de la imagen. Cada instrucción en un *Dockerfile* crea una capa en la imagen. Cuando se cambia el *Dockerfile* solo se reconstruyen las capas que han cambiado. Esto es parte de lo que hace que las imágenes sean ligeras, pequeñas y rápidas en comparación con otras tecnologías de virtualización[14].

4.1.6 Serverless containers

A medida que aumenta el uso de contenedores a una escala significativa, aumenta también la necesidad de herramientas de administración; en orden de controlar, en toda la infraestructura, las tareas de automatización de implementación, escalado y operación general de aplicaciones en contenedores. Un ejemplo ilustrativo de cómo se están haciendo indispensables las herramientas de orquestación proviene de Google. Los creadores de la herramienta de orquestación de contenedores de código abierto Kubernetes ejecutan todos sus servicios, en todas sus unidades de negocio en contenedores, y se ha informado que ponen en marcha más de dos mil millones de contenedores cada semana[13]. Adicionalmente, Kubernetes se ha convertido en el estándar de facto debido a su solidez, madurez y características ricas que liberan de la carga de tener que configurar y mantener infraestructuras complejas [13].

Kubernetes

Kubernetes es una plataforma de código abierto para automatizar la implementación, el escalado y la administración de aplicaciones, en contenedores.

En Kubernetes se utilizan los objetos de la API de Kubernetes para describir el estado deseado del clúster, por ejemplo: qué aplicaciones u otras cargas de trabajo se quieren ejecutar, qué imágenes en los contenedores se deben usar, el número de replicas, qué red y qué recursos de almacenamiento se tienen disponibles, etc.

Se especifica el estado deseado del clúster típicamente mediante la interfaz de línea de comandos, *kubectl*. También se puede usar la API de Kubernetes directamente para interactuar con el clúster y especificar o modificar el estado deseado. Una vez que se especifica el estado deseado, el Control Plane de Kubernetes se encarga de realizar las acciones necesarias para que el estado actual del clúster coincida con el estado deseado. Para esto, Kubernetes pone en marcha diferentes tareas de forma automática, como pueden ser: parar o arrancar contenedores, escalar el número de réplicas de una aplicación dada, etc. [15].

¿Por qué la necesidad de una herramienta de orquestación de contenedores?

- Tendencia de migración de aplicaciones monolíticas a microservicios.
- Aumento en el uso de contenedores.
- Exigencia de una forma adecuada de gestionar la cantidad de contenedores desplegados.

Componentes de Kubernetes

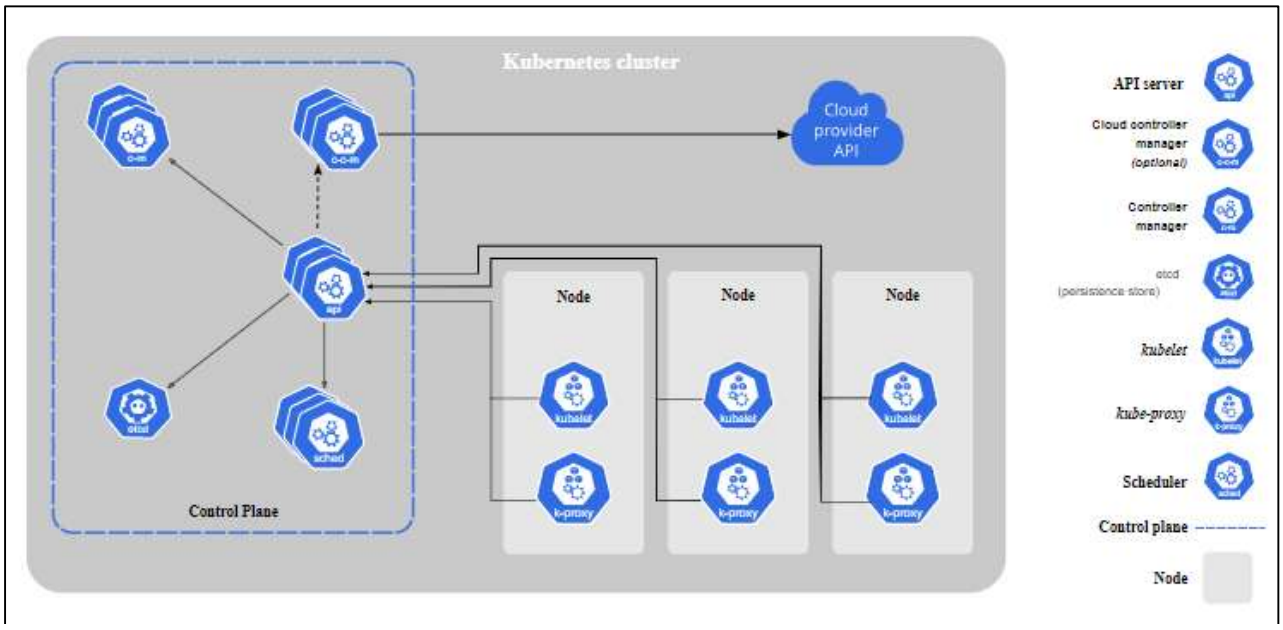

Imagen 3. Componentes de Kubernetes (Tomado de la documentación oficial de Kubernetes)

Componentes del Control Plane

El "Control Plane" (Plano de Control) de Kubernetes es el conjunto de componentes que se encargan de gestionar y orquestar los recursos del clúster. Estos componentes incluyen el "kube-apiserver", que proporciona una API para interactuar con el clúster; "etcd", que almacena la configuración y el estado del clúster; el "kube-scheduler", que programa los contenedores en los nodos disponibles; y el "kube-controller-manager", que gestiona los controladores que automatizan los procesos en el clúster [15].

Kube-apiserver

El "kube-apiserver" es un componente clave del "Control Plane" de Kubernetes que proporciona una API RESTful para interactuar con el clúster Kubernetes. Es el punto de entrada principal para todas las solicitudes al clúster y actúa como un intermediario entre los usuarios o las herramientas de la línea de comandos y el resto de componentes de Kubernetes, como "etcd", el "kube-scheduler" y el "kube-controller-manager" [15].

Etcd

Almacén de datos persistente, consistente y distribuido de clave-valor utilizado para almacenar toda a la información del clúster de Kubernetes.

Kube-scheduler

Es el componente del Control Plane que monitorea el estado actual de los Pods en contraste con el estado deseado y selecciona en qué nodos ejecutar los Pods. Para decidir en qué nodo se ejecutará un Pod, se tienen en cuenta diversos factores: requisitos de recursos, restricciones de hardware/software/políticas, entre otros[15].

4.2 Antecedentes de la investigación (Trabajos previos)

En la literatura hallada, se encuentran algunos estudios relacionados con el monitoreo de desempeño de aplicaciones. En 2016 Kim, Rhee, Lee, Zhang, & Xu [16], propusieron un enfoque automatizado para analizar los binarios de la aplicación e instrumentar el código binario de forma transparente, en orden de inyectar y aplicar aserciones de rendimiento de las transacciones de la aplicación. Para esto, definieron la herramienta PerGuard, que identifica las transacciones de aplicación y monitorea su rendimiento sin conocimiento del código y usando únicamente binarios. Para validar la propuesta, analizaron los binarios de seis aplicaciones populares y detectaron 10 problemas de desempeño del mundo real conocidos públicamente sin el código fuente o conocimiento de código. El impacto de rendimiento de PerGuard en estas aplicaciones es de menos del 3%. De igual forma el

presente proyecto se centrará en realizar de manera transparente la medición del software, contando únicamente con los binarios (caja negra). Es decir, que no busca de alguna forma realizar un monitoreo activo, si no mediante monitoreo sintético (no invasivo) con mediciones hechas por robots de pruebas que manipulan la interfaz de usuario.

En 2012 Hoorn, Waller, & Hasselbring [17], presentaron Kieker, un framework para monitoreo y análisis del comportamiento de un sistema de software distribuido en tiempo de ejecución. Al enfocarse en monitoreo a nivel de aplicación, Kieker incluye sondas de monitoreo para recolectar información de tiempo y traza desde ejecuciones distribuidas de operaciones de software, y, adicionalmente, sondas para muestreo de medidas a nivel de sistema (como por ejemplo uso de CPU y memoria). Realizar un análisis dinámico con Kieker requiere la instrumentación del sistema software, así como la especificación de la configuración del monitoreo y análisis. En el proyecto presente se realiza actividades de monitoreo con herramientas opensource ya existentes con las que capturan trazas para enviar a un repositorio Big Data Centralizado.

En 2018 Holst & Schupp [10], presentaron un nuevo enfoque de monitoreo que incluye la configuración de la medición, el diseño de simulaciones estables y realistas del usuario, así como métodos basados en estadísticas para la detección de anomalías. Para monitorear las aplicaciones, los robots simulan usuarios en diferentes ubicaciones y miden el tiempo de respuesta de la aplicación. La característica distintiva de este enfoque de medición es que los robots tienen las mismas capacidades que un usuario real; lo que significa que el robot interactúa con la aplicación mediante la Interfaz Gráfica de Usuario (GUI). Este enfoque garantiza que la medición detecte las mismas anomalías que un usuario real notaría.

En 2017 Otarán, Perera, & Luengo [18], propusieron una solución de monitoreo de rendimiento para la dirección de desarrollo del centro superior para el procesamiento de la información (CeSPI) de la Universidad Nacional de La Plata. En el estudio presentan la comparación y propuesta de una estrategia de recolección, análisis y utilización de información de la infraestructura y aplicaciones del CeSPI, que permita simplificar y enriquecer su análisis posterior y resulte en un monitoreo eficiente. La solución propuesta utiliza InfluxDB para almacenar los datos, Kibana para la visualización y Kapacitor para implementar las alertas en la solución. La pila de tecnologías seleccionadas servirá de guía en el presente proyecto.

5 SELECCIÓN DE SERVICIOS DE KUBERNETES EN LA NUBE

A continuación, se describen las actividades realizadas en la selección de los servicios de computación en la nube sobre los cuales se desarrollaron los colectores de rendimiento del clúster para las pruebas de rendimiento:

1. Identificación de los servicios líderes del mercado en computación de la nube.
2. Identificación de la pila tecnológica de los principales clientes de GreenSQA.
3. Investigación de los principales servicios *Serverless Containers* y Kubernetes y realización de pruebas de concepto sobre estos.
4. Selección de dos de los servicios de computación en la nube para colectar la información de las pruebas de rendimiento.

5.1 Identificación de los servicios de computación de la nube líderes del mercado

Con el fin de determinar las opciones con las cuales se realizó la selección de los servicios de computación en la nube, se hizo una identificación de los servicios lideres del mercado.

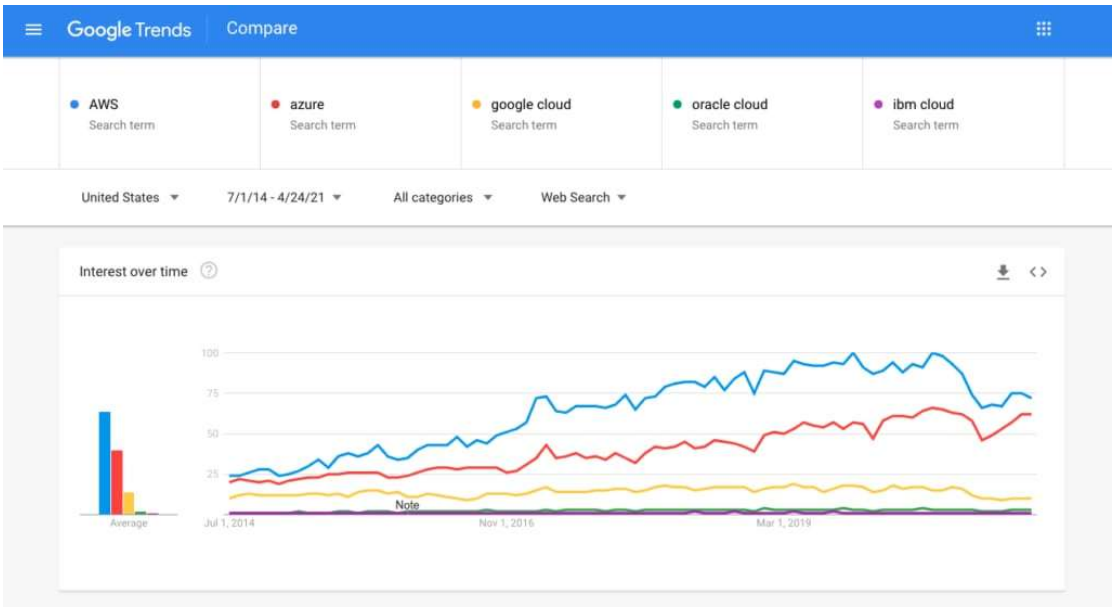

Imagen 4. Proveedores de servicio de nube más populares, Google Trends, 2022.

Según lo evidenciado en las tendencias de búsqueda de Google reportadas en la herramienta *Google Trends* los siete proveedores de servicio de nube más populares

corresponden a Amazon Web Services, Microsoft Azure, Google Cloud, Alibaba Cloud, Oracle, IBM y Tencent Cloud.

Por otra parte, de acuerdo con Gartner Inc. el mercado de servicios de plataforma en la nube se encuentra distribuido, con más del 90% del mercado mundial concentrado en solo cuatro proveedores[19]: Amazon Web Services y Microsoft Azure lideran el mercado seguido por Google y Alibaba, como los próximos competidores más cercanos.

Imagen 5. Cuadrante de Gartner 2019 para proveedores de servicios de la nube.

En base a esta información podemos destacar a Amazon Web Services, Microsoft Azure y Google como las tres principales opciones de servicios de computación en la nube para la realización de los colectores.

5.2 Identificación de la pila tecnológica de los clientes principales de Green SQA

Mediante varias interacciones de diseño con el equipo de investigación de GreenSQA, se desarrolló un formulario que con el objetivo de encuestar a los principales clientes de GreenSQA, y conocer el *stack* tecnológico con el que cuentan. Este formulario se centró

en conocer si los clientes de GreenSQA contaban con tecnologías de computación en la nube, contenerización, *serverless containers* en la nube, y, además, para determinar qué proveedores de servicio usaban para ellos (ver Anexo 1).
.

Imagen 6. Formulario Web desarrollado para la encuesta de stack tecnológico

La encuesta se realizó mediante entrevistas guiadas a través del diligenciamiento de un formulario web.

Como primer paso se identificó la información que se necesitaba recopilar de los clientes y que sería de utilidad en la selección de las plataformas sobre la cual realizar la investigación:

- Dentro de los principales clientes de GreenSQA, ¿cuáles son las arquitecturas de software más comunes?
- Dentro de los principales clientes de GreenSQA, ¿se cuenta con sistemas de monitoreo y con cuáles?

- Dentro de los principales clientes de GreenSQA que realizan procesos de CD, ¿estos se hacen hacia servidores o sistemas serverless?, ¿en caso de ser realizados con servidores estos son: *onpremise*, en la nube, *hosting*, ¿máquinas virtuales en la nube?, ¿en caso de ser *serverless* con que proveedores se realiza?
- Dentro de los principales clientes de GreenSQA, ¿cuáles usan tecnologías de contenerización del software?
- Dentro de los principales clientes de GreenSQA, ¿qué tipo de tecnologías de contenerización de software usan?
- Dentro de los principales clientes de GreenSQA, ¿qué cantidad cuenta con componentes de cómputo en la nube dentro de su *stack* tecnológico?
- Dentro de los principales clientes de GreenSQA, ¿qué tipo nube manejan: nube pública, nube privada o nube híbrida?
- Dentro de los principales clientes de GreenSQA que hacen uso de nubes públicas o hibridas, ¿cuáles son los principales proveedores que de nubes públicas que usan?
- Dentro de los principales clientes de GreenSQA, ¿qué cantidad usa tecnologías *serverless containers* en la nube y cuáles de estas tecnologías?

5.2.1 Resultados

Se logró encuestar 9 clientes de Green SQA, de mercados variados como lo son la industria petrolera, servicios de apoyo de negocios, servicios de cadena de suministro y logística, servicios médicos, telecomunicaciones y servicios financieros.

Una vez realizada la encuesta se pudo conocer el *stack* tecnológico con el que actualmente cuentan los principales clientes encuestados de GreenSQA. A continuación, se puede visualizar los resultados más relevantes:

Tipos de tecnologías de contenerización de software usados dentro de la organización:

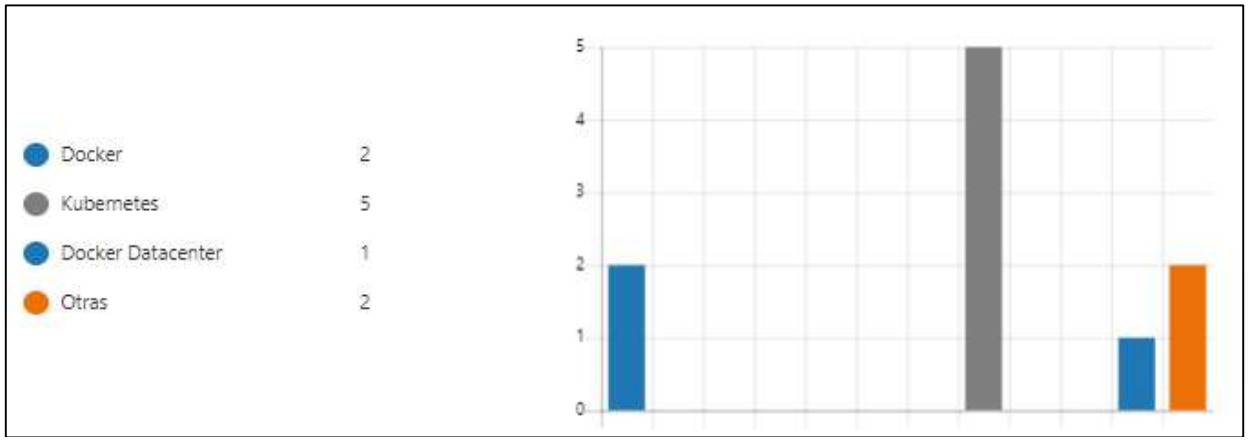

Imagen 7. Resultados tipos de tecnologías de contenerización

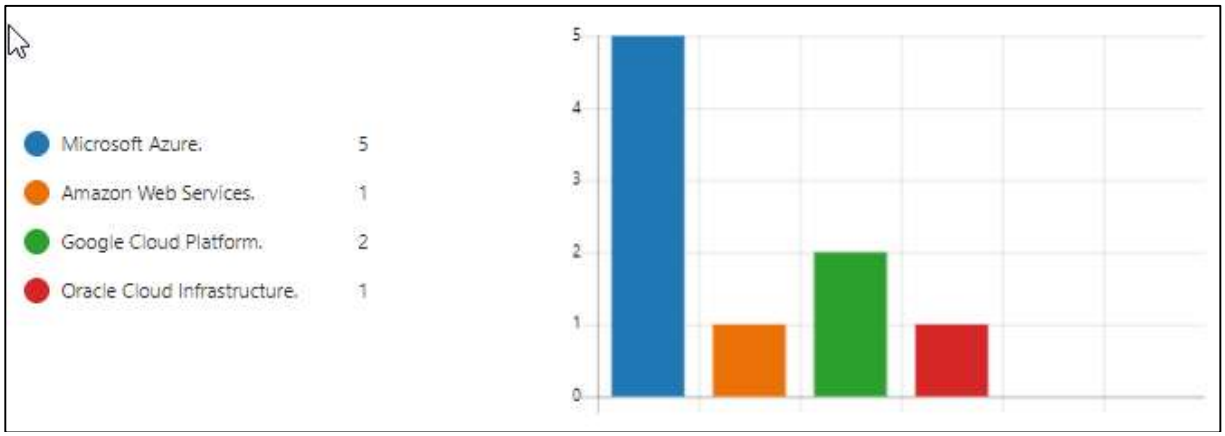

Imagen 8. Resultados de proveedores de nubes públicas usados

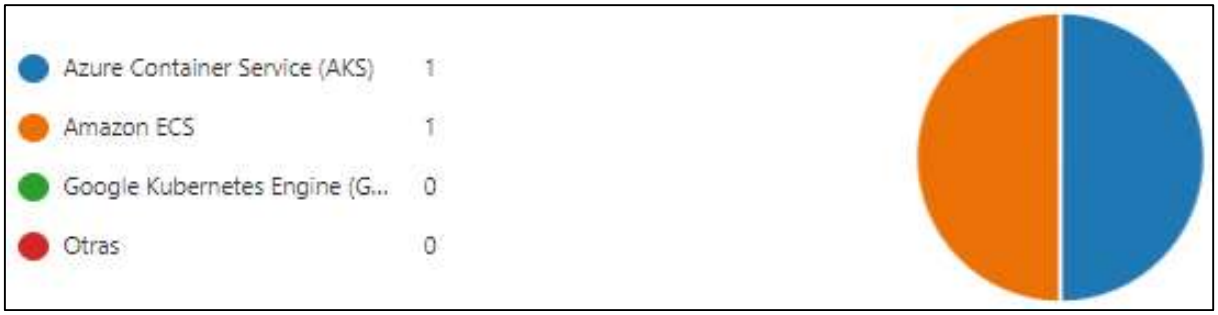

Imagen 9. Resultados servicios Kubernetes en la nube más usados

5.3 Investigación de servicios Serverless Containers y Kubernetes

A continuación, se recopila la información más relevante tomada en la investigación realizada sobre las características principales de los servicios principales *Serverless Container* y Kubernetes en la nube. Esta investigación se hizo con el fin de contar con un entendimiento más profundo sobre los servicios opcionados y que facilitara la selección del servicio sobre el cual realizar el proyecto.

5.3.1 Servicios Serverless

Azure Container Instances

Azure Container Instances (ACI) es una herramienta para ejecutar un contenedor en Azure, sin tener que gestionar máquinas virtuales y sin tener que adoptar un mayor nivel de servicio.

Es una solución para escenarios que operan en contenedores aislados, incluidas aplicaciones simples, automatización de tareas y trabajos de compilación. Sin embargo, para escenarios en los que se necesita una orquestación completa de contenedores, incluido el descubrimiento de servicios en varios contenedores, escalado automático y coordinación de actualización de aplicación, es más recomendable el uso de servicios como Azure Kubernetes Servicio (AKS)[20].

Tiempos de puesta en marcha

Azure Container Instances puede iniciar contenedores en Azure en segundos sin la necesidad de aprovisionar y administrar máquinas virtuales. Es posible traer imágenes de contenedor de Linux o Windows desde Docker Hub, un registro de contenedores de Azure privado u otro registro de Docker basado en la nube[20].

Acceso al contenedor

Azure Container Instances permite exponer grupos de contenedores directamente a Internet con una dirección IP y un nombre de dominio completo (FQDN). Cuando se crea una instancia de contenedor se puede especificar una etiqueta de nombre DNS personalizada para que la aplicación sea accesible.

Azure Container Instances también admite la ejecución de comandos en el contenedor en ejecución, al proporcionar un *shell* interactivo para ayudar con el desarrollo de aplicaciones y la solución de problemas. El acceso se realiza a través de HTTPS, utilizando TLS[20].

Tamaños personalizados

Los contenedores suelen estar optimizados para ejecutar una sola aplicación, pero las necesidades exactas de esas aplicaciones pueden diferir enormemente. Azure Container Instances permite especificaciones exactas de núcleos de CPU y memoria.

Por otra parte, el pago se realiza en función de lo que se selecciona y se factura por segundo, por lo que se pueden ajustar los gastos en función de la necesidad real[20].

Grupos de contenedores en Azure Container Instances

El recurso de nivel superior en Azure Container Instances es el grupo de contenedores. Un grupo de contenedores es una colección de contenedores que se programan en la misma máquina host. Los contenedores de un grupo de contenedores comparten un ciclo de vida, recursos, red local y volúmenes de almacenamiento. Es similar en concepto a un *pod* en Kubernetes.

Azure Container Instances y orquestadores de contenedores

Debido a su pequeño tamaño y orientación a las aplicaciones, los contenedores son adecuados para entornos de entrega ágiles y arquitecturas basadas en microservicios.

La tarea de automatizar y administrar una gran cantidad de contenedores (y de cómo interactúan) se conoce como orquestación. Los orquestadores de contenedores populares incluyen Kubernetes, DC/OS y Docker Swarm.

Azure Container Instances proporciona algunas de las capacidades de programación básicas de las plataformas de orquestación. Y aunque no cubre los servicios de mayor valor que brindan esas plataformas, sí que puede complementarlas[20].

Orquestación con Azure Container Instances: un enfoque por capas

Azure Container Instances permite un enfoque en capas para la orquestación; proporcionando todas las capacidades de programación y administración necesarias para ejecutar un solo contenedor, al tiempo que permite que las plataformas de orquestadores administren tareas de varios contenedores[20].

Nodos virtuales para Azure Kubernetes Service (AKS)

Para escalar rápidamente las cargas de trabajo de las aplicaciones en un clúster de Azure Kubernetes Service (AKS), se pueden usar nodos virtuales creados de forma dinámica en Azure Container Instances. Los nodos virtuales permiten la comunicación de red entre los pods que se ejecutan en ACI y el clúster de AKS[20].

Amazon Fargate

AWS Fargate es el servicio serverless de Amazon para la ejecución de contenedores, una tecnología que proporciona capacidad de computación del tamaño especifico bajo demanda para los contenedores. Con AWS Fargate, no es necesario realizar la configuración, escalado o aprovisionamiento de grupos de máquinas virtuales para la ejecución de contenedores. Fargate proporciona un entorno aislado para cada *pod*, lo que significa que no hay compartición de recursos, tales como el *kernel* subyacente, memoria, CPU o la interfaz de red elástica con otro *pod* [21].

Pod CPU y memoria

Cuando se programan pods en Fargate, la CPU virtual y las reservas de memoria dentro de la especificación del *pod* determinan la cantidad de CPU y memoria que se debe aprovisionar para el *pod*.

- La solicitud de recursos máxima del contenedor se utiliza para definir los requerimientos de inicio de CPU y memoria.
- Las solicitudes de todos los contenedores de ejecución prolongada se suman para determinar los requisitos de memoria y CPU de solicitud de ejecución prolongada.

La siguiente tabla muestra las combinaciones de CPU virtual y memoria que están disponibles para los pods que se ejecutan en Fargate[21].

valor de vCPU	Valor de memoria
.25 vCPU	0,5 GB, 1 GB, 2 GB
.5 vCPU	1 GB, 2 GB, 3 GB, 4 GB
1 vCPU	2 GB, 3 GB, 4 GB, 5 GB, 6 GB, 7 GB, 8 GB
2 vCPU	Entre 4 GB y 16 GB en incrementos de 1 GB
4 CPU virtuales	Entre 8 GB y 30 GB en incrementos de 1 GB

Tabla 1. Combinaciones de CPU virtual y memoria disponibles para pods que se ejecutan en Fargate

Almacenamiento Fargate

Cuando se aprovisiona, cada *pod* que se ejecuta en Fargate recibe 20 GB de almacenamiento de capa de imagen de contenedor. El almacenamiento de los contenedores es efímero, una vez que se detiene un pod, se elimina el almacenamiento[21].

Métricas de uso de AWS Fargate

Se puede utilizar *CloudWatch* para proporcionar visibilidad en del consumo de los recursos. Las métricas de uso de AWS Fargate corresponden a las cuotas de servicio de AWS. Se pueden configurar las alarmas que alertan para cuando el uso se aproxima a las cuotas de servicio.

5.3.2 Servicios Kubernetes

Azure Kubernetes Service (AKS)

AKS simplifica la implementación de un clúster de Kubernetes, al relegar la sobrecarga operativa a Azure, responsable de labores críticas tales como supervisión y mantenimiento del estado en su servicio alojado de Kubernetes.

AKS es gratuito; y se paga por los nodos del clúster, no por los maestros. Se pueden crear un clúster de AKS usando:

- La CLI de Azure.
- El Portal Azure.

- Azure PowerShell.

AKS cuenta con opciones de implementación basadas en plantillas. Cuando se implementa un clúster de AKS, el maestro de Kubernetes y todos los nodos se implementan y configuran automáticamente. Las redes, la integración de Azure Active Directory (Azure AD), la supervisión y otras características se pueden configurar durante el proceso de implementación [22].

Gestión de identidad y seguridad

- Kubernetes RBAC: Para limitar el acceso a los recursos del clúster, AKS admite Kubernetes RBAC. Kubernetes RBAC controla el acceso y los permisos a los recursos y espacios de nombres de Kubernetes.
- Azure AD: Es posible configurar un clúster de AKS para que se integre con Azure AD. Con la integración de Azure AD, se puede configurar el acceso a Kubernetes según la identidad existente y la pertenencia a grupos.

Logging y monitoreo integrado

Azure Monitor for Container Health recopila métricas de rendimiento de procesador y memoria de contenedores, nodos y controladores dentro del clúster de AKS y las aplicaciones implementadas.

Es posible revisar tanto los logs de contenedores como los logs de los maestros de Kubernetes, que son disponible a través de Azure Portal, Azure CLI o un punto de conexión REST.

Clústeres y nodos

Los nodos de AKS se ejecutan en máquinas virtuales (VM) de Azure.

- Con los nodos de AKS, puede conectar el almacenamiento a los nodos y pods, actualizar los componentes del clúster y usar GPU.
- AKS admite clústeres de Kubernetes que ejecutan varios grupos de nodos para admitir sistemas operativos mixtos y contenedores de Windows Server.

Escalado de pods y nodos del clúster

A medida que cambia la demanda de recursos, la cantidad de nodos de clúster o pods que ejecutan los servicios aumentan o disminuyen automáticamente.

Es posible ajustar tanto el escalamiento horizontal automático de pods como el escalador automático del clúster para ajustarse a las demandas y ejecutar solo los recursos necesarios[22].

Soporte de volumen de almacenamiento

Se pueden montar volúmenes de almacenamiento estáticos o dinámicos para datos persistentes. Dependiendo de la cantidad de pods conectados que se espera que compartan los volúmenes de almacenamiento, se puede usar el respaldado por[22]:

- Azure Disks para acceso a un solo pod.
- Azure Files para acceso simultáneo a varios pods.

Redes virtuales e Ingress

Un clúster de AKS se puede implementar en una red virtual preexistente. En esta configuración, a cada *pod* del clúster se le asigna una dirección IP en la red virtual y puede comunicarse directamente con otros pods del clúster y con otros nodos de la red virtual.

Los pods también pueden conectarse a otros servicios en una red virtual emparejada y a redes locales a través de ExpressRoute o conexiones VPN de sitio a sitio (S2S)[22].

Integración de herramientas de desarrollo

Azure proporciona varias herramientas que ayudan a optimizar Kubernetes, como Azure Dev Spaces y DevOps Starter.

Soporte de imágenes de Docker y registro de contenedores privados

AKS admite el formato de imagen de Docker, para el almacenamiento privado de imágenes de Docker, puede integrar AKS con Azure Container Registry (ACR).

Servicio Amazon Elastic Kubernetes

Amazon Elastic Kubernetes Service (Amazon EKS) es un servicio administrado que se utiliza para ejecutar Kubernetes en AWS sin necesidad de instalar, operar y mantener un propio clúster o nodos de Kubernetes.

- Para garantizar una alta disponibilidad, se puede ejecutar y dimensionar el plano de control de Kubernetes en varias zonas de disponibilidad de AWS.

- Escala automáticamente las instancias del control plane en función de la carga, detectada, reemplaza las instancias del control plane en mal estado y proporciona actualizaciones de versión automatizadas y parches para ellas.
- Está integrado con muchos servicios de AWS para proporcionar escalabilidad y seguridad para sus aplicaciones, incluidas las siguientes capacidades:
    - Amazon ECR (Elastic Container Registry) para imágenes de contenedores.
    - Balanceo de carga elástico para distribución de carga.
    - IAM para autenticación.
    - Amazon VPC para aislamiento.

En Amazon EKS, las aplicaciones son compatibles con cualquier entorno estándar de Kubernetes, ya sea en un centro de datos local o en una nube pública. Por lo tanto, es posible migrar sin realizar modificaciones de código cualquier aplicación estándar de Kubernetes a Amazon EKS.[23].

Arquitectura de un control plane de Amazon EKS

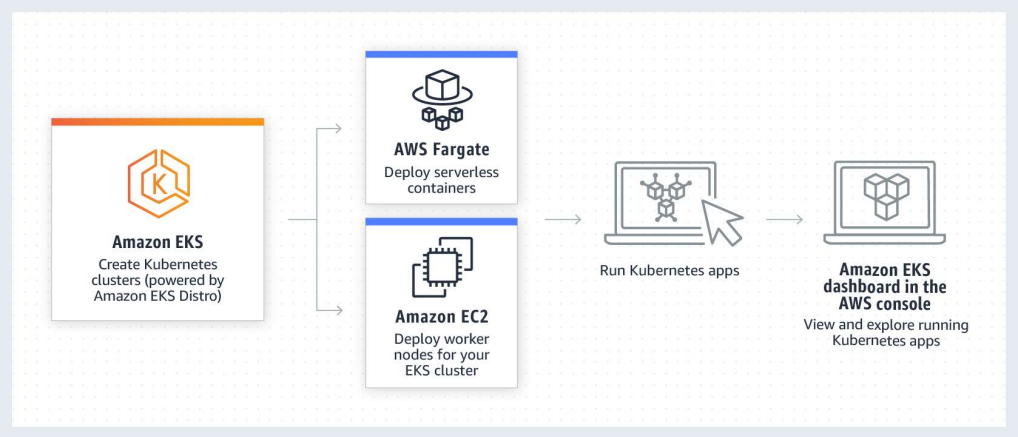

Imagen 10. Amazon EKS (Tomado de la documentación Amazon EKS)

Amazon EKS ejecuta un único control plane de Kubernetes inquilino para cada clúster. La infraestructura del control plane no se comparte entre clústeres o cuentas de AWS. El control plane consta de al menos dos instancias de servidor API y tres instancias *etcd* que se ejecutan en tres zonas de disponibilidad dentro de una región.

- Supervisa activamente la carga en las instancias del control plane y las escala automáticamente para garantizar un alto rendimiento.
- Detecta y reemplaza automáticamente las instancias del plano de control plane en mal estado y las reinicia en las Zonas de disponibilidad dentro de la Región según sea necesario.

- Aprovecha la arquitectura de las regiones de AWS para mantener una alta disponibilidad. Debido a esto, Amazon EKS puede ofrecer un SLA para la disponibilidad del punto final del servidor API.

Amazon EKS usa las políticas de red de Amazon VPC para restringir el tráfico entre los componentes del plano de control plane dentro de un solo clúster. Los componentes del plano de control plane de un clúster no pueden ver ni recibir comunicaciones de otros clústeres u otras cuentas de AWS, excepto lo autorizado por las políticas de Kubernetes RBAC.

Gestión de identidad y acceso para Amazon EKS

EKS usa AWS Identity and Access Management (IAM) para la administración de inicio de sesión y permisos para utilizar los recursos de Amazon EKS.

Clústeres de Amazon EKS

Un clúster de Amazon EKS consta de dos componentes principales:

- El control plane de Amazon EKS (*Masternode*).
- Nodos de Amazon EKS que están registrados en el control plane.

Nodos de Amazon EKS

En Amazon EKS, el clúster tiene la capacidad de programar pods en una combinación de nodos autoadministrados, grupos de nodos administrados de Amazon EKS y AWS Fargate.

Escalador automático vertical de pod

El escalador automático vertical de pods de Kubernetes ajusta automáticamente la CPU y las reservas de memoria de sus pods para ayudar a "dimensionar correctamente" las aplicaciones.

5.4 Pruebas de concepto en las plataformas identificadas

Con el fin de tener un entendimiento más profundo sobre los servicios Kubernetes opcionados, se realizaron pruebas de concepto que permitieran tomar decisiones sobre la estructuración de la plataforma desarrollada y facilitara la selección del servicio sobre el cual se realizó el proyecto. A continuación, se detallan los pasos realizados en las pruebas de concepto.

El objetivo de las pruebas de concepto consistió en realizar el despliegue de una aplicación Web .NET sencilla, contenerizada, con base de datos SQL Server, en los servicios de AKS, GKE y EKS, para identificar las siguientes necesidades claves:

- ¿Cómo realizar la creación de un clúster de los servicios de AKS, GKE y EKS?
- ¿Como conectar kubectl con el clúster de los servicios de AKS, GKE y EKS?
- Conocer cómo se crean y despliegan en AKS, GKE y EKS, los siguientes componentes:

    - StorageClass para crear un disco.
    - PersistentVolumeClaim.
    - Secrets.
    - LoadBalancer.
    - Deployments para contenedor de SQLServer y contenedor de .NET Web App.
    - Ingress Controller.
    - Habilitar HTTPS con CertManager.

5.4.1 Elementos necesarios y configuración inicial

Durante la realización de la prueba de concepto se usaron las siguientes herramientas:

- **kubectl**: Herramienta de línea de comandos para trabajar con clústeres de Kubernetes que automatiza muchas tareas de administración del clúster.
- **Helm**: Herramienta administradora de paquetes para Kubernetes.

Para la realización la prueba de concepto en AKS, se usó adicionalmente:

- Azure command-line interface (Azure CLI).

Para la realización de las pruebas de concepto en EKS, se usó:

- AWS Command Line Interface (AWS CLI).
- EKSCTL: Herramienta de línea de comandos para trabajar con clústeres de EKS que automatiza muchas tareas.

Para la realización de la prueba de concepto en GKE, se usó:

- Cloud SDK: Command Line Interface

5.4.2 Desarrollo de la prueba de concepto

A continuación, se detallan los pasos principales en la ejecución de las pruebas de concepto:

Creación de un clúster de Kubernetes en AKS

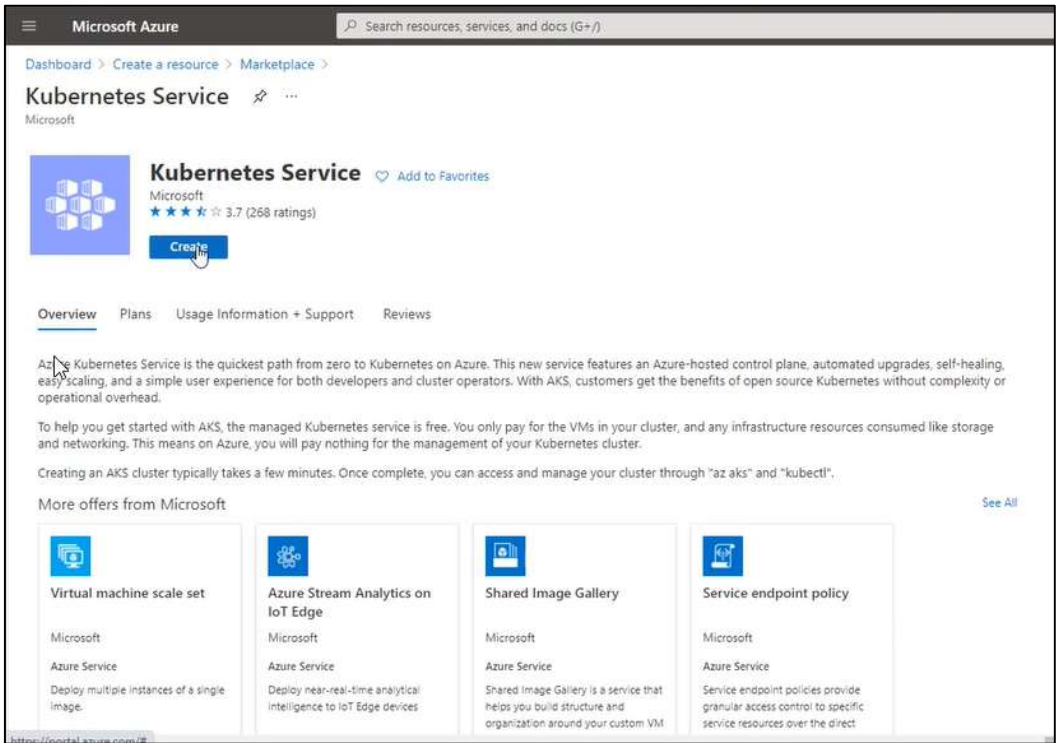

Imagen 11. Interfaz de creación del clúster paso 1

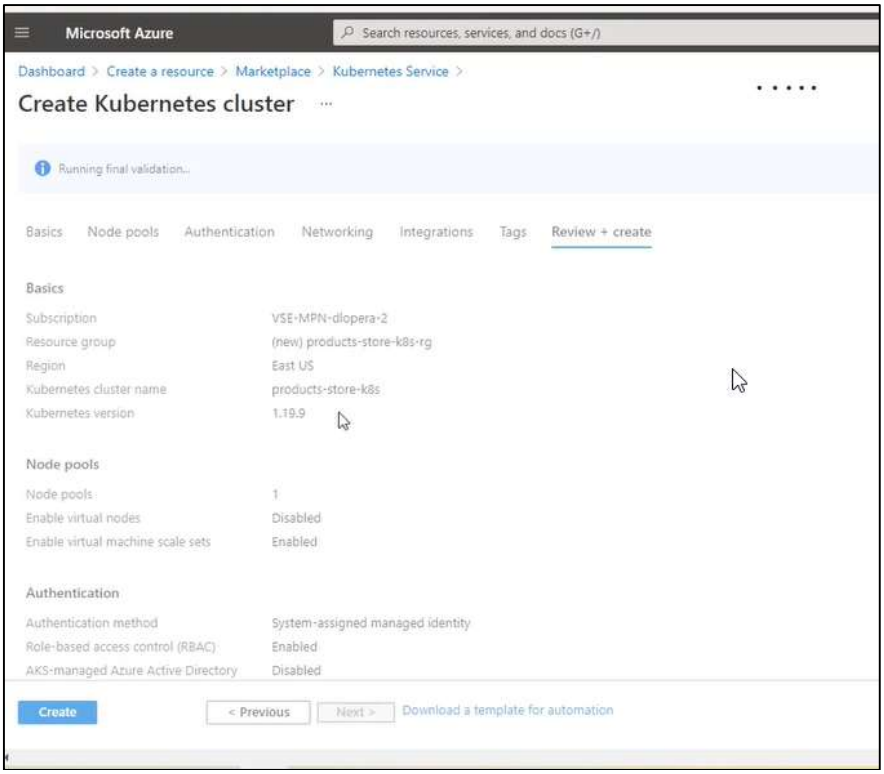

Imagen 12. Interfaz de creación del clúster paso 2

La creación del clúster de Kubernetes en AKS se realizó mediante la interfaz de creación de clúster AKS en portal de Azure, con dos nodos de máquinas virtuales de tipo B2s

Creación de un clúster de Kubernetes en EKS

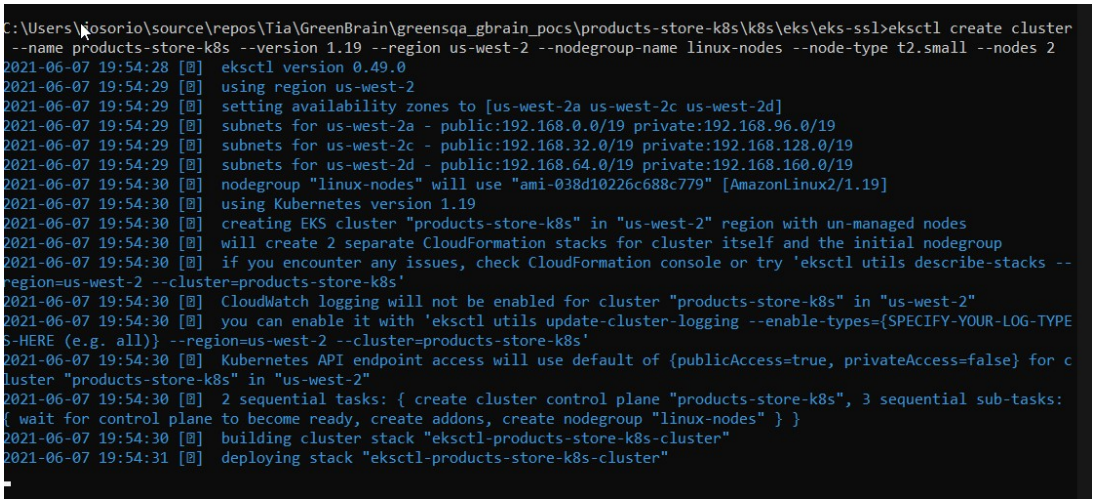

Imagen 13. Línea de comandos EKSCTL

La creación del clúster de Kubernetes en EKS se realizó mediante la herramienta EKSCTL, con dos nodos de máquinas virtuales de tipo t2.small

Creación de un clúster de Kubernetes en GEK

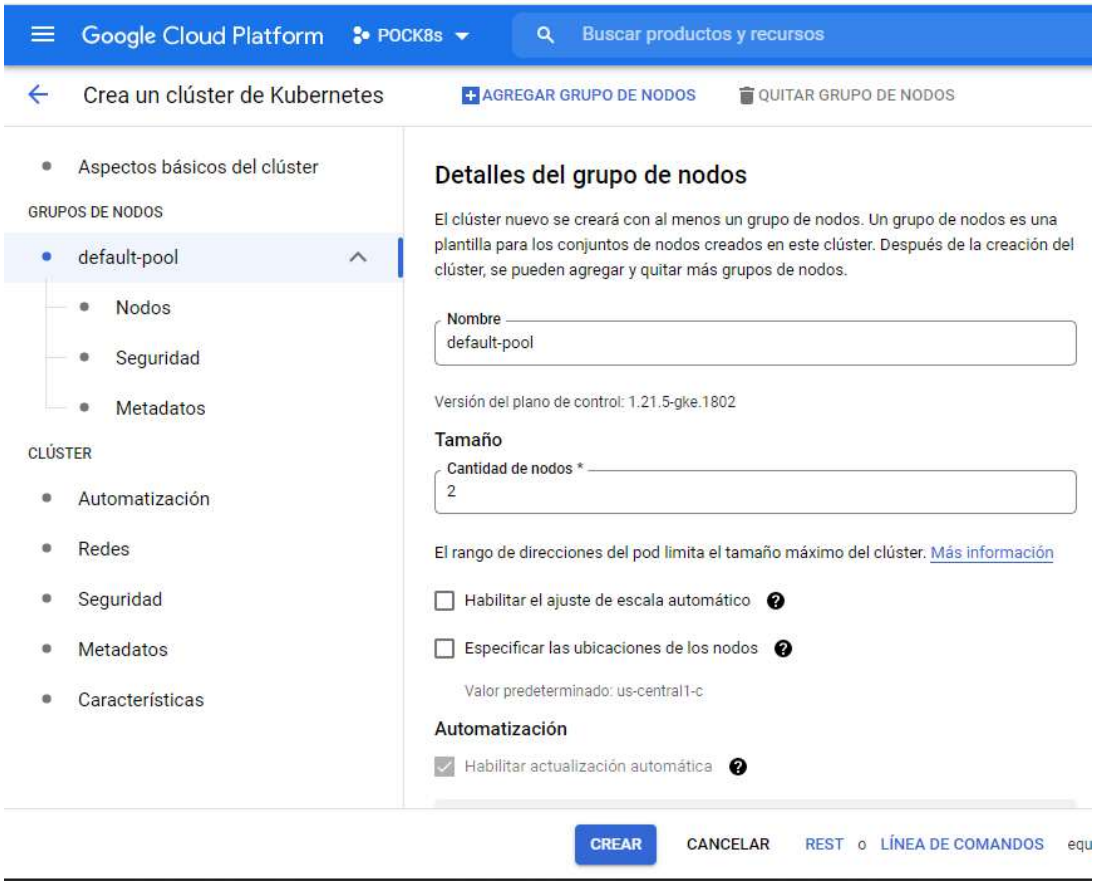

Imagen 14. Interfaz para crear clúster en Google Cloud

La creación del clúster de Kubernetes en EKS se realizó mediante la interfaz de creación de clúster en portal de Google Cloud Console, con dos nodos de máquinas virtuales de tipo *e2-medium.*

Conectar kubectl con el clúster de AKS

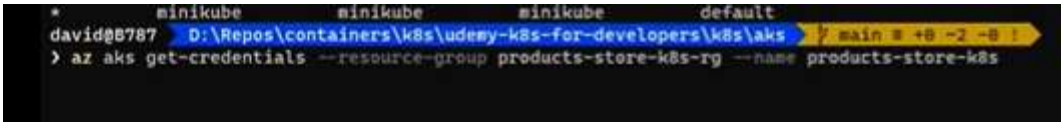

Imagen 15. Ejecución del comando para la conexión de kubectl a AKS

Para conectar kubectl al clúster de AKS, se hizo uso de la herramienta Azure CLI y el comando *az aks get-credentials*, indicando el nombre del grupo de recursos y el nombre del clúster.

Conectar kubectl con el clúster de EKS

Para conectar kubectl con el clúster en EKS, no fue necesario ningún paso adicional debido que la herramienta EKSCTL automáticamente realiza la conexión en la creación del clúster.

Conectar kubectl con el clúster de GEK

```
PS C:\Users\GreenSQA\OneDrive - PUJ Cali\tesis\Desarrollo de tesis\Nueva carpeta\gke\gke> gcloud container clusters get-
credentials products-store-k8s --zone us-central1-c --project pock8s-332602
Fetching cluster endpoint and auth data.
kubeconfig entry generated for products-store-k8s.
```

Imagen 16. Ejecución del comando para la conexión de kubectl a GEK

Para conectar kubectl al clúster de GEK, se realizó uso de la herramienta Cloud SDK Command Line y el comando *gcloud container clusters get-credentials products-store-k8s --zone us-central1-c --project pock8s-332602*.

Creación de un namespace

Para la creación de un namespace en AKS, EKS y GKE, se creó el archivo *app-namespace.yml* con las siguientes especificaciones:

```yaml
1  apiVersion: v1
2  kind: Namespace
3  metadata:
4    name: products-store-ns
```

Imagen 17. Creación del archivo app-namespace.yml

Una vez creado, se ejecutó el comando *kubectl apply -f app-namespace.yml* en las pruebas de concepto*:*

```
> kubectl apply -f app-namespace.yaml
namespace/products-store-ns created
```

Imagen 18. Ejecución del comando kubectl apply -f app namespace.yml

Creación de una StorageClass y un PersistentVolumeClaim

Para la creación de un StorageClass y un PersistentVolumeClaim en AKS, se creó el archivo *products-store-mssql-pv.yaml,* con las siguientes especificaciones:

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  namespace: products-store-ns
  name: products-store-mssql-sc
provisioner: kubernetes.io/azure-disk
parameters:
  storageaccounttype: Standard_LRS
  kind: Managed
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  namespace: products-store-ns
  name: products-store-mssql-pv-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: products-store-mssql-sc
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: default
  resources:
    requests:
      storage: 1Gi
```

Imagen 19. Archivo de configuración de StorageClass y un PersistentVolumeClaim en AKS

Para la creación de un StorageClass y un PersistentVolumeClaim en EKS, se creó el archivo *products-store-mssql-pv.yaml,* con las siguientes especificaciones:

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: products-store-mssql-sc
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: products-store-mssql-pv-claim
  annotations:
    volume.beta.kubernetes.io/storage-class: products-store-mssql-sc
spec:
  accessModes:
  - ReadWriteOnce
  storageClassName: defualt
  resources:
    requests:
      storage: 1Gi
```

Imagen 20. Archivo de configuración de StorageClass y un PersistentVolumeClaim en EKS

Para la creación de un StorageClass y un PersistentVolumeClaim en GKE, se creó el archivo *products-store-mssql-pv.yaml,* con las siguientes especificaciones:

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  namespace: products-store-ns
  name: products-store-mssql-sc
provisioner: kubernetes.io/gce-pd
parameters:
  type: pd-standard
  fstype: ext4
---
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: products-store-mssql-pv-claim
  namespace: products-store-ns
  annotations:
    volume.beta.kubernetes.io/storage-class: products-store-mssql-sc
spec:
  accessModes:
    - ReadWriteOnce
  storageClassName: standard
  resources:
    requests:
      storage: 1Gi

```

Imagen 21. Archivo de Configuración de StorageClass y un PersistentVolumeClaim en EKS

Luego de la creación de los archivos, se ejecutó el comando *kubectl apply -f products-store-mssql-pv.yaml,* en las pruebas de concepto*:*

```
> kubectl apply -f products-store-mssql-pv.yaml
storageclass.storage.k8s.io/products-store-mssql-sc created
persistentvolumeclaim/products-store-mssql-pv-claim created
```

Imagen 22. Aplicación de Configuración de StorageClass y un PersistentVolumeClaim en el clúster

Creación de un Secrets

Para la creación de un Secrets en las tres pruebas, se creó el archivo *products-store-mssql-secret,* con las siguientes especificaciones:

```yaml
apiVersion: v1
kind: Secret
metadata:
  namespace: products-store-ns
  name: products-store-mssql-secret
data:
  db-connection-string-secret: U2VydmVyPXByb2R1Y3RzLXN0b3JlLW1zc3FsLXNlcnZpY2U7RGF0YWJhc2U9UH
  db-password-secret: QEFhMTIzNDU2
type: Opaque

```

Imagen 23. Archivo para configuración de secrets

Luego de la creación de los archivos, se ejecutó el comando *kubectl apply -f products-store-mssql-secret,* para ambas pruebas de concepto*:*

Imagen 24. Aplicación de Configuración de secretes en el kluster

Deployments para contenedor de SQLServer

Para el componente *deployments* del contenedor de SQLServer, se creó el archivo *products-store-mssql-deployment,* con las siguientes especificaciones:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  namespace: products-store-ns
  name: products-store-mssql-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: products-store-mssql
  template:
    metadata:
      labels:
        app: products-store-mssql
    spec:
      terminationGracePeriodSeconds: 30
      hostname: mssqlinst
      securityContext:
        fsGroup: 10001
      containers:
      - name: mssql
        image: mcr.microsoft.com/mssql/server:2019-latest
        ports:
        - containerPort: 1433
        env:
        - name: MSSQL_PID
          value: "Express"
        - name: ACCEPT_EULA
          value: "Y"
        - name: SA_PASSWORD
          valueFrom:
            secretKeyRef:
              name: products-store-mssql-secret
```

Imagen 25. Archivo de configuración para deployments de contenedor de SQLServer.

```yaml
                key: db-password-secret
          volumeMounts:
          - name: mssqldb
            mountPath: /var/opt/mssql
        volumes:
        - name: mssqldb
          persistentVolumeClaim:
            claimName: products-store-mssql-pv-claim
---
apiVersion: v1
kind: Service
metadata:
  namespace: products-store-ns
  name: products-store-mssql-service
spec:
  selector:
    app: products-store-mssql
  ports:
    - protocol: TCP
      port: 1433
      targetPort: 1433
      nodePort: 30200
  type: NodePort
```

Imagen26. Archivo de configuración para deployments de contenedor de SQLServer.

Luego de la creación del archivo, se ejecutó el comando *kubectl apply -f products-store-mssql-deployment.yaml* para ambas pruebas de concepto*:*

```
kubectl apply -f products-store-mssql-deployment.yaml
deployment.apps/products-store-mssql-deployment created
service/products-store-mssql-service created
david@B787 D:\Repos\containers\k8s\udemy-k8s-for-developers\k8s\aks-ssl
```

Imagen27. Aplicación de configuración para deployments de contenedor de SQLServer

Deployments para App Web

Para el deployments de la aplicación Web en ambas pruebas de concepto, se creó el archivo *products-store-app-deployment.yml,* con las siguientes especificaciones:

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  namespace: products-store-ns
  name: products-store-deployment
spec:
  selector:
    matchLabels:
      app: products-store-app
  replicas: 2
  template:
    metadata:
      labels:
        app: products-store-app
    spec:
      containers:
      - name: products-store-app
        image: dloperab/products-store-mvc:v1.0
        ports:
        - containerPort: 80
        env:
        - name: ConnectionStrings__MvcAppContext
          valueFrom:
            secretKeyRef:
              name: products-store-mssql-secret
              key: db-connection-string-secret
---
kind: Service
apiVersion: v1
metadata:
  name: products-store-service
spec:
  selector:
    app: products-store-app
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
  type: LoadBalancer
```

Imagen 28. Archivo de configuración para deployments de contenedor de App Web

Luego de la creación del archivo, se ejecutó el comando *kubectl apply -f products-store-app-deployment.yaml*, en ambas pruebas de concepto*:*

```
> kubectl apply -f products-store-app-deployment-v1.yaml
deployment.apps/products-store-deployment created
service/products-store-service created
```

Imagen 29. Aplicación de configuración para deployments de contenedor de App Web

Habilitar Ingress Controller y HTTPS con CertManager

Para instalar el Ingress Controller en el clúster se usó la herramienta Helm, ejecutando el siguiente comando:

```
helm install app-ingress ingress-nginx/ingress-nginx '
    --namespace ingress '
    --create-namespace '
    --set controller.replicaCount=2 '
    --set controller.nodeSelector."beta\.kubernetes\.io/os"=linux '
    --set defaultBackend.nodeSelector."beta\.kubernetes\.io/os"=linux
```

Imagen 30. Comando para instalación de Ingress Controller

Una vez instalado el Ingress Controller, se creó el archivo app-ingress.yml con la siguiente configuración para el Ingress Controller:

app-ingress.yaml

C: > Users > josorio > source > repos > Tia > GreenBrain > greensqa_gbrain_pocs > products-store-k8s > k8s > aks > aks-ssl > app-ingress.yaml

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: webapp-ingress
  namespace: products-store-ns
  annotations:
    kubernetes.io/ingress.class: nginx
    nginx.ingress.kubernetes.io/ssl-redirect: "false"
    nginx.ingress.kubernetes.io/use-regex: "true"
    nginx.ingress.kubernetes.io/rewrite-target: /$1
spec:
  rules:
  - http:
      paths:
      - backend:
          serviceName: products-store-service
          servicePort: 80
        path: /(.*)
  - host: productsstore.52.146.84.66.nip.io # change the IP address here
    http:
      paths:
      - backend:
          serviceName: products-store-service
          servicePort: 80
        path: /(.*)
```

Imagen 31. Archivo de configuración de para Ingress Controller

Luego de la creación de los archivos se ejecutó el comando *kubectl apply -f app-ingress.yaml* para ambas pruebas de concepto*:*

```
C:\Users\josorio\source\repos\Tia\GreenBrain\greensqa_gbrain_pocs\products-store-k8s\k8s\eks\eks-ssl>kubectl apply -f app-
ingress.yaml
Warning: networking.k8s.io/v1beta1 Ingress is deprecated in v1.19+, unavailable in v1.22+; use networking.k8s.io/v1 Ingres
s
ingress.networking.k8s.io/webapp-ingress created
```

Imagen 32. Aplicación de configuración para Ingress Controller

Habilitar HTTPS con CertManager

Para habilitar el CertManager se usó herramienta Helm con la cual se ejecutó el siguiente comando:

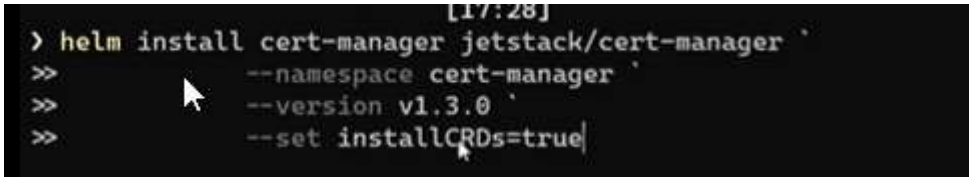

Imagen 33. Comando para instalación de CertManger

Una vez instalado el CertManager, se crearon los archivos ssl-tls-cluster-issuer.yml y ssl-tls-ingress, con la siguiente configuración:

```yaml
apiVersion: cert-manager.io/v1alpha2
kind: ClusterIssuer
metadata:
  name: letsencrypt
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    email: josorio@greensqa.com
    privateKeySecretRef:
      name: letsencrypt
    solvers:
    - http01:
        ingress:
          class: nginx
```

Imagen 34. Archivo de configuración de para CertManger 1

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ssl-tls-ingress
  annotations:
    kubernetes.io/ingress.class: nginx
    cert-manager.io/cluster-issuer: letsencrypt
spec:
  tls:
    - hosts:
      - productsstore.44.239.247.158.nip.io/ # update IP address here
      secretName: products-store-cert
  rules:
  - host: productsstore.44.239.247.158.nip.io/ # update IP address here
    http:
      paths:
      - backend:
          serviceName: products-store-service
          servicePort: 80
        path: /
```

Imagen 35. Archivo de configuración de para CertManger 2

Luego de la creación de los archivos se ejecutó los comandos *kubectl apply --namespace products-store-ns -f ssl-tls-cluster-issuer.yaml y* kubectl apply --namespace products-store-ns -f ssl-tls-ingress.yaml, en ambas pruebas de concepto*:*

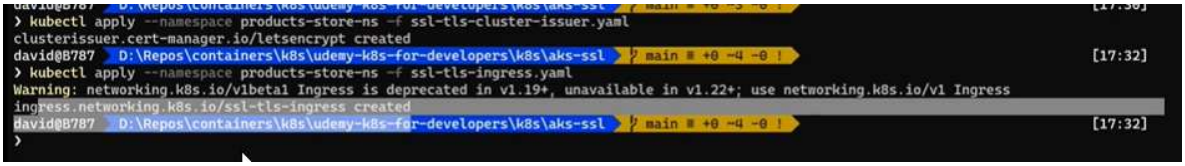

Imagen 36. Aplicación de configuración de para CertManger

5.4.3 Resultados prueba de concepto

Una vez terminadas las pruebas de concepto y habiendo logrado realizar el despliegue de una aplicación Web .NET sencilla, contenerizada, con base de datos SQL Server, en los servicios de AKS, GKE y EKS, se pudo identificar que Kubernetes, al ser un estándar y exponer sus funcionalidades por medio de APIs, permite abstraer en gran media las tareas de despliegue y administración de clúster independiente de la plataforma en la que se implemente, lo anterior evidenciado en los cambios mínimos que se tuvieron que hacer en los archivos de configuración para el despliegue en una plataforma u otra en las pruebas de concepto realizadas.

5.5 Selección de plataforma en la nube a colectar información

De acuerdo con las investigaciones y pruebas de concepto realizadas y descritas en este documento, se seleccionan los servicios de Kubernetes en la nube de Microsoft Azure Kubernetes Service y Amazon Elastic Kubernetes Service, inicialmente basándose en los criterios que se describen a continuación:

Plataformas en la nube líderes del mercado:

Amazon Web Services, Microsoft Azure y Google, en tanto las tres principales opciones de servicios de computación en la nube para la realización de los colectores.

Uso de los servicios de Microsoft Azure y Amazon Web Services por parte de los clientes de GreenSQA:

De acuerdo con las encuestas realizadas a los principales clientes de GreenSQA, un 63% de los clientes manifestaron hacer uso de los servicios de Microsoft Azure, y un 25% que hace uso de Amazon Web Services (la segunda plataforma más usada).

Curva de aprendizaje y experiencia previa:

Siendo parte el equipo GreenSQA (un Microsoft Partner Gold), para el desarrollo del presente proyecto se cuenta con amplia experiencia manejando diferentes herramientas y servicios que provee Microsoft Azure, además de haber ejecutado ya proyectos usando la misma plataforma. Por lo anterior la selección de Microsoft Azure sería una decisión dinamizadora del proyecto al disminuir la curva de aprendizaje. Dicho sea de paso, con la segunda plataforma con la que más se cuenta experiencia es con Amazon Web Services.

Portabilidad y transparencia entre proveedores de nubes:

Kubernetes, al ser un estándar y exponer sus funcionalidades por medio de APIs, garantiza que las aplicaciones funcionen en gran medida de forma independiente respecto al entorno. De esta manera, las aplicaciones se pueden mover a diferentes plataformas en la nube sin que su funcionalidad se vea afectada. Lo anterior evidenciado en los cambios mínimos que se tuvieron que hacer en los archivos de configuración para el despliegue en una plataforma u otra en las pruebas de concepto realizadas.

6 DESARROLLO DE LA PLATAFORMA DE MONITOREO

Para el desarrollo de la plataforma se usaron las dos primeras etapas de la Metodología de Analíticas de GreenSQA. Esta está diseñada para agilizar el proceso de construcción de soluciones para el monitoreo de software en ambientes de producción o similares, y permitir interpretar las analíticas generadas. Dicha metodología propone cinco etapas de desarrollo de las cuales se describen las dos primeras a continuación[24] (para la metodología completa revisar el Anexo 2):

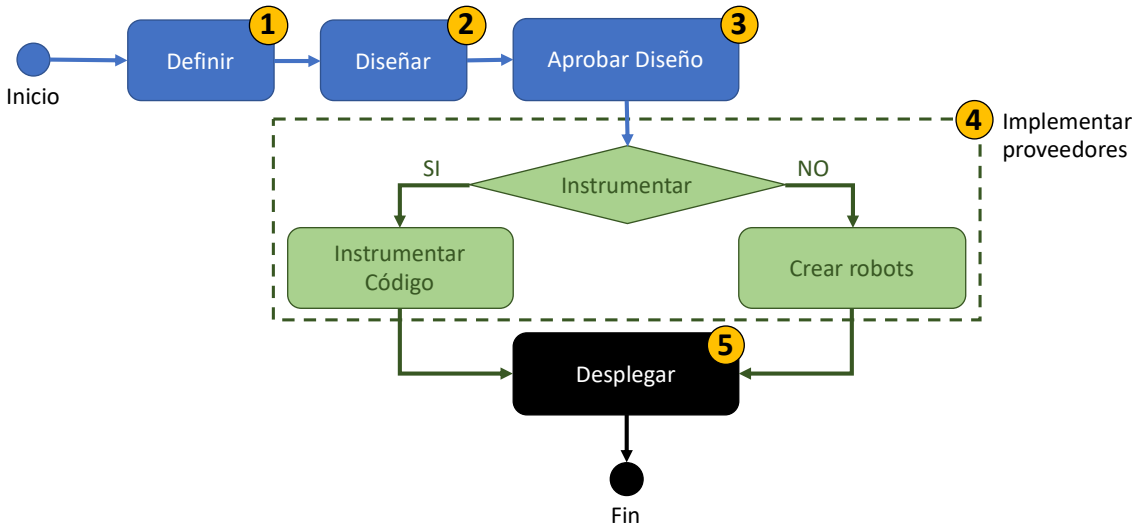

Imagen 37. Etapas de Desarrollo de la metodología de analíticas de GreenSQA

- En la etapa *I Definir*, las actividades están orientadas a identificar las necesidades de información de la organización, estructurar esta información y definir los proveedores de datos, es decir, que se decide si se instrumentará el código o si se crearán robots.
- La etapa *II Diseñar* busca modelar los tableros en los que se mostrará la información a través de gráficas sencillas y comprensibles, además, en esta etapa se diseñan los proveedores de información (instrumentación de código o robots).

Es importante resaltar que durante el desarrollo de la plataforma se contó la participación y retroalimentación del equipo experto pruebas con el fin de validar que la plataforma sea una herramienta útil en el proceso de pruebas.

6.1 Etapa de definición

6.1.1 Definición de las necesidades de información

GreenSQA diseña la estrategia RUPET (*Real User Performance Test*) que consiste en ejecutar pruebas de r

endimiento mediante las múltiples ejecuciones robots que simulan usuarios que interactúan con el sistema bajo prueba, emulando una carga masiva de los mismos. Durante la realización de estas pruebas se hace necesario capturar las métricas de rendimiento desde el *frontend* y el *backend*, y se precisa que la información obtenida sea fácilmente correlacionable para comprender cómo se comportaron los servicios y procesos durante todas las ejecuciones de la prueba bajo diferentes cantidades de volumen de tráfico.

Para la captura de datos y métricas de rendimiento desde la perspectiva del *FronEnd*, en GreenSQA, son creados y ejecutados robots mediante AIMaps. Al ejecutarse envían métricas de rendimiento a la plataforma GreenHeart. Estas métricas son almacenadas en una base de datos InfluxDB y muestran información sobre tiempos de ejecución de las pruebas, estado exitoso o fallido y calificación del rendimiento en relación los tiempos de respuesta en comparación con los tiempos esperados (desde la perspectiva del usuario).

Para el *backend* se identificó la necesidad de monitorear el estado de todo el clúster de Kubernetes. En particular, interesaba saber si todos los nodos del clúster funcionan correctamente y a qué capacidad lo hacen, cuántas aplicaciones se están ejecutando en cada nodo y la utilización de recursos de todo el clúster.

Revisando la documentación de Kubernetes podemos identificar un conjunto inicial de 132 métricas (Detalles en Anexo 3) pertenecientes a las siguientes categorías:

Tipo de métrica	Cantidad de métricas
1. configmap	3
2. Daemon set	11
3. deployment	13
4. endpoint	5
5. ingress	6
6. mutating web hook configuration	3
7. namespace	3
8. network policy	4
9. node	8
10. persistent volume	10

11. pod	36
12. replica set	9
13. secret	5
14. service	5
15. stateful set	11

Tabla 2. Métricas por categoria Categorías de Kubertenes

6.1.2 Priorizar las variables apoyados con un panel de expertos

Tras una revisión de la literatura y mediante el juicio de expertos del equipo de GreenSQA (Priorización de las variables relevantes por parte del equipo con las expectativas y su experiencia) se seleccionaron las variables o métricas que son *críticas* (ver Anexo 4) para la captura en la realización de las pruebas.

La definición de métricas críticas obedece al objetivo de selección de variables de salud y de correcto desempeño. Con base en lo anterior, se definieron 16 categorías denominadas “condiciones críticas para la salud de Kubernetes”, que agrupan 36 métricas asociadas. Como observación general, la definición de las métricas se basa en las siguientes consideraciones respecto a las respectivas categorías:

- ***Crash Loops***: Un pod se inicia, se bloquea (crash) y luego sigue intentando reiniciarse, pero no lo logra (sigue bloqueándose y reiniciando en un bucle).
- ***Cluster State Metrics****:* Los pods deseados, los pods actuales, los nodos disponibles y los no disponibles.
- **CPU utilization**: Indica cuántos ciclos de CPU están usando los nodos.
- ***Memory utilization****:* Indica cuanta memoria están usando los nodos.
- ***POD utilization****:* Indica cuántos pods están usando los nodos.
- ***Disk Pressure****:* La presión del disco es una condición que indica que un nodo está usando demasiado espacio en disco o está agotando el espacio en disco demasiado rápido.
- ***Memory Pressure****:* Indica que un nodo se está quedando sin memoria.
- ***PID Pressure****:* Es una condición poco común en la que un pod o contenedor genera demasiados procesos y priva al nodo de los ID de proceso disponibles. Cada nodo tiene un número limitado de ID de proceso para distribuir entre los procesos en ejecución.
- ***Network Unavailable****:* Todos los nodos necesitan conexiones de red y este estado indica que hay algún problema con la conexión de red de un nodo.
- ***Resources Requested and Limits by Containers:*** Si los recursos requeridos por los contenedores superan los límites definidos por estos, puede generar que los pods fallen.

- ***Job Failures**:* Están diseñados para ejecutar pods durante un período de tiempo limitado y eliminarlos cuando hayan completado las funciones previstas. Si un trabajo no se completa correctamente debido a que un nodo se bloquea o se reinicia, o debido al agotamiento de los recursos, debe saber que el trabajo falló.
- ***Persistent Volume Failures**:* Los volúmenes persistentes son recursos de almacenamiento que se especifican en el clúster y están disponibles como almacenamiento persistente para cualquier pod que lo solicite. Durante su ciclo de vida están vinculados a un módulo y luego se recuperan cuando ese módulo ya no los necesita. Si esa recuperación falla por cualquier motivo, debe saber que hay algún problema con su almacenamiento persistente.
- ***Pod Pending Delays**:* Durante el ciclo de vida de un pod, el estado es "pendiente" indica que un pod está esperando para ser programado en un nodo. Si está quedando en el estado "pendiente", generalmente significa que no hay suficientes recursos para programar e implementar el pod dentro del nodo.
- ***Deployment Glitches**:* Se deben vigilar las implementaciones (Deployments) para asegurarse de que finalicen correctamente. La mejor manera es asegurarse de que la cantidad de implementaciones observadas coincida con la cantidad de implementaciones deseadas.
- ***StatefulSets Not Ready**:* Se debe asegurar de que la cantidad de StatefulSets observados coincida con la cantidad de StatefulSets deseada.
- ***DaemonSets Not Ready**:* Se debe asegurar de que el número de DaemonSets observados coincida con el número de DaemonSets deseados.

6.1.3 Clasificar necesidades por niveles y subniveles

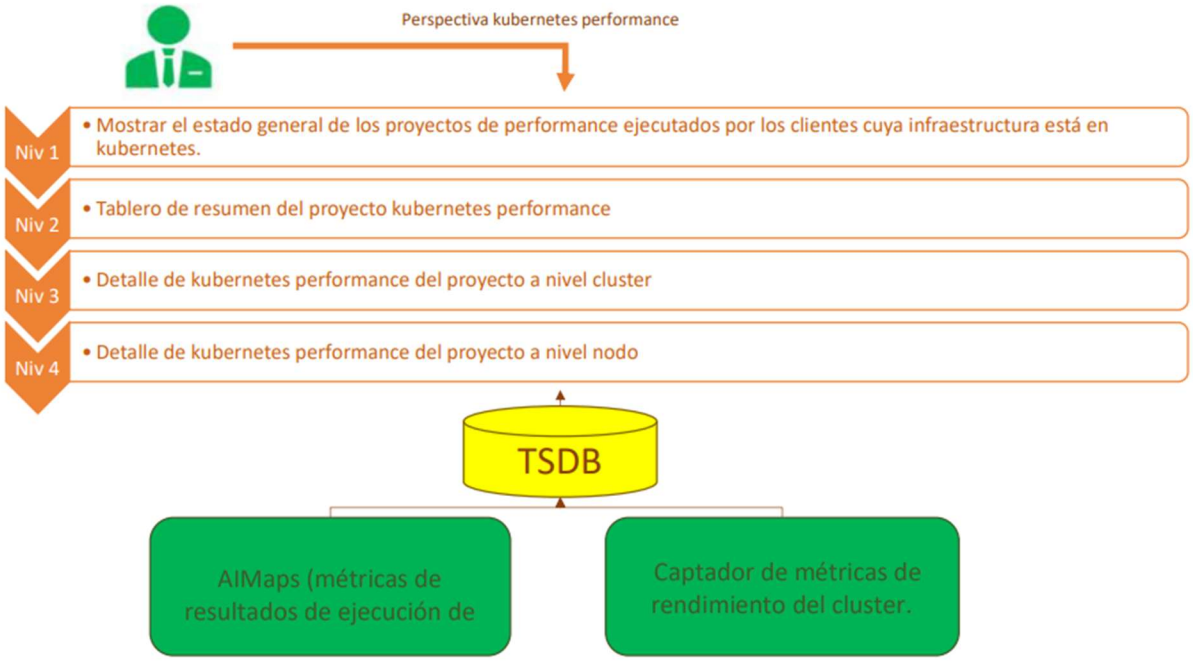

Imagen 38. Diagrama de niveles de necesidades de información diligenciado para el proyecto.

Siguiendo la metodología y mediante la validación iterativa con el equipo de GreenSQA, se identifican cuatro niveles de necesidades de información como se ve en el diagrama de la imagen 39

Nivel 1 de información

Para el nivel 1 se identifica la necesidad de mostrar el estado general de los proyectos de performance ejecutados por los clientes cuya infraestructura está en Kubernetes (ver Anexo 5).

Nivel 2 de información

Para el nivel 2 se identifica la necesidad de mostrar el estado general de un proyecto de performance ejecutado *sumarizando* el comportamiento RUM (de transacciones y subprocesos + page navigation timing) y una visión general del comportamiento de los recursos del *backend* en el entorno de Kubernetes (ver Anexo 5).

Nivel 3 de información

Para el nivel 3 se identifica la necesidad de mostrar una visión detallada del comportamiento de los recursos del backend en un entorno de Kubernetes a nivel de clúster (ver Anexo 5).

Nivel 4 de información

Para el nivel 4 se identifica la necesidad de mostrar una visión detallada del comportamiento de los recursos del backend en un entorno de Kubernetes a nivel de nodo (ver Anexo 5).

6.1.4 Definición de proveedores de información

Se realizó una investigación de las posibles arquitecturas con las cuales se podría implementar el colector de métricas de rendimiento y las herramientas usadas para su desarrollo e implementación.

En la investigación realizada se identificaron cuatro posibles arquitecturas, revisando las respectivas ventajas y desventajas que cada arquitectura ofrece. A continuación, se puede ver el detalle de la investigación sobre cada opción de arquitectura:

Arquitectura - Opción 1

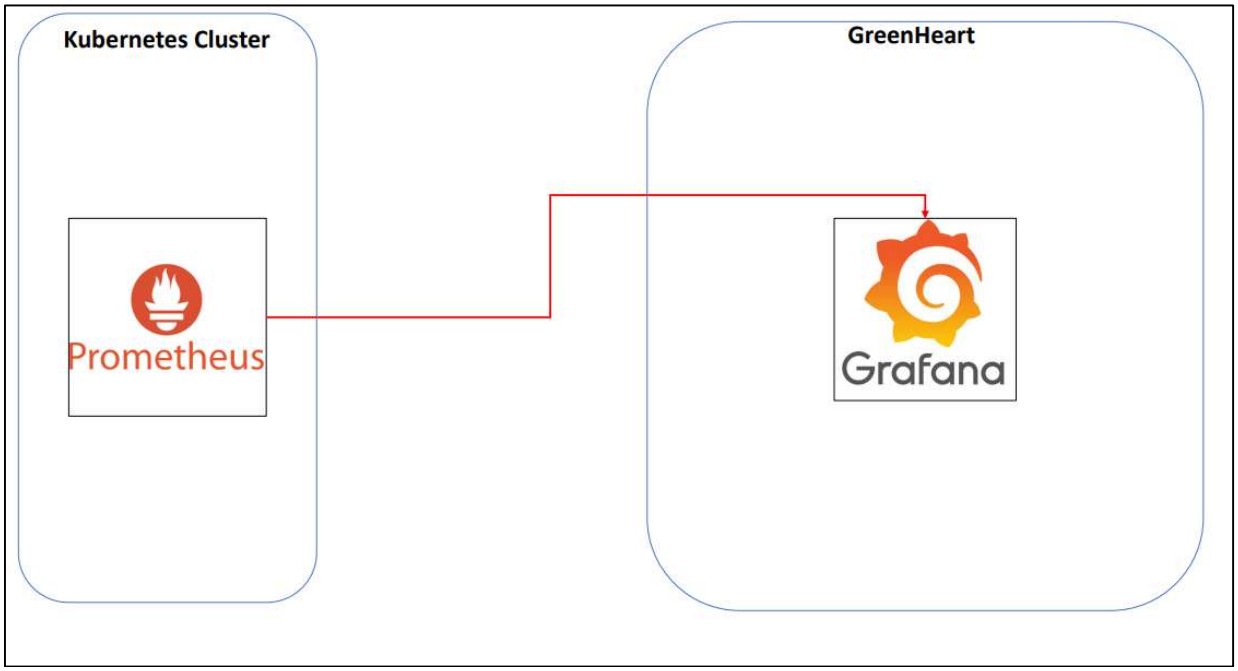

Imagen 39. Diagrama Arquitectura.

Esta opción consiste en instalar el *stack* de Prometheus en el clúster de Kubernetes y luego acceder a las métricas de Prometheus desde el componente de Grafana instalado en la plataforma GreenHeart.

- **Ventajas**

La opción brinda un monitoreo más completo porque se obtienen la mayor cantidad de métricas posibles del clúster de Kubernetes.

- **Desventajas**

Es una opción invasiva en el clúster de Kubernetes, porque se instalan diversos componentes, tales como: Prometheus, Grafana, Alert Manager, Node Exporter, Kube State Metrics y Controller Manager. Adicionalmente, la base de datos de Prometheus almacena en el clúster información de métricas, por lo que ocasiona costos adicionales de almacenamiento.

Arquitectura - Opción 2

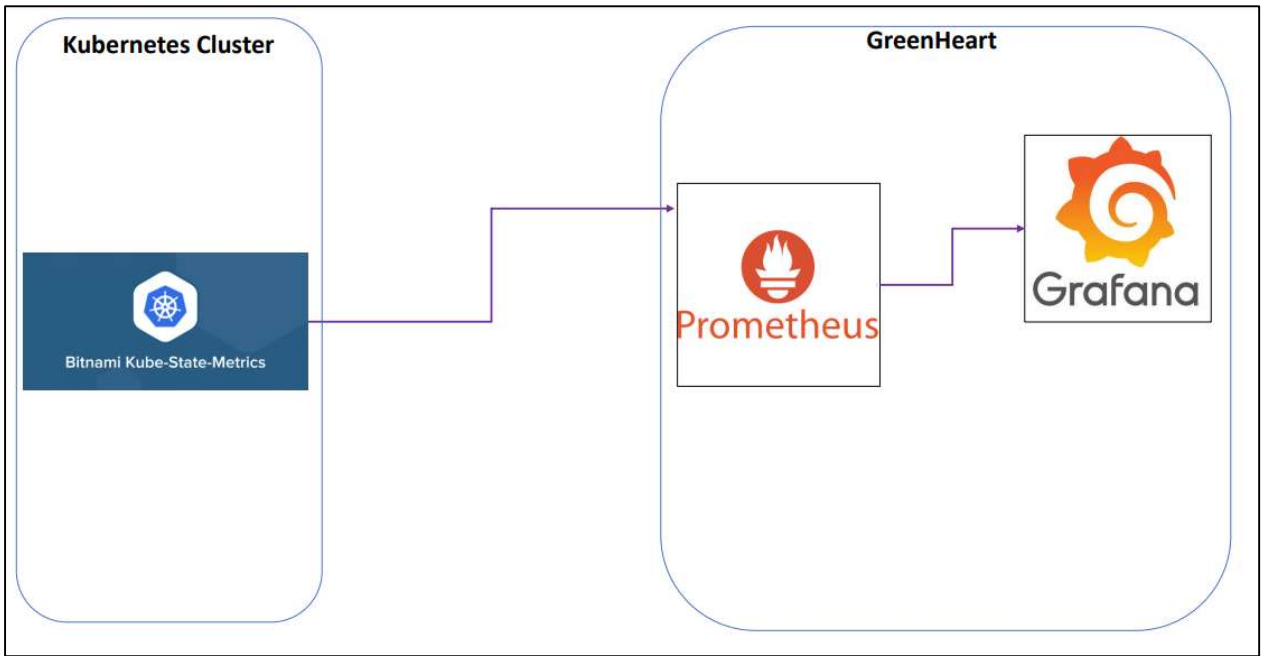

Imagen 40. Diagrama Arquitectura Opción 2

Esta opción consiste en instalar el componente de Kube-State-Metrics en el clúster de Kubernetes, el cual se encarga de exponer las métricas de rendimiento en un formato compatible con la herramienta Prometheus. En GreenHeart se instala el componente de Prometheus que extrae las métricas expuestas por el componente Kube-State-Metrics. Finalmente, Grafana instalado en la plataforma GreenHeart accede a las métricas de Prometheus.

- **Ventajas**

  Es una opción poco invasiva para el clúster de Kubernetes porque solo se instala el componente Kube-State-Metrics. Adicionalmente, la base de datos de Prometheus se almacena en la plataforma GreenHeart.

- **Desventajas**

  No se puede obtener la totalidad de las métricas brindadas por el *stack* de Prometheus de la opción 1. No obstante, las métricas obtenidas son las más importantes para identificar el comportamiento del estado del clúster.

Arquitectura - Opción 3

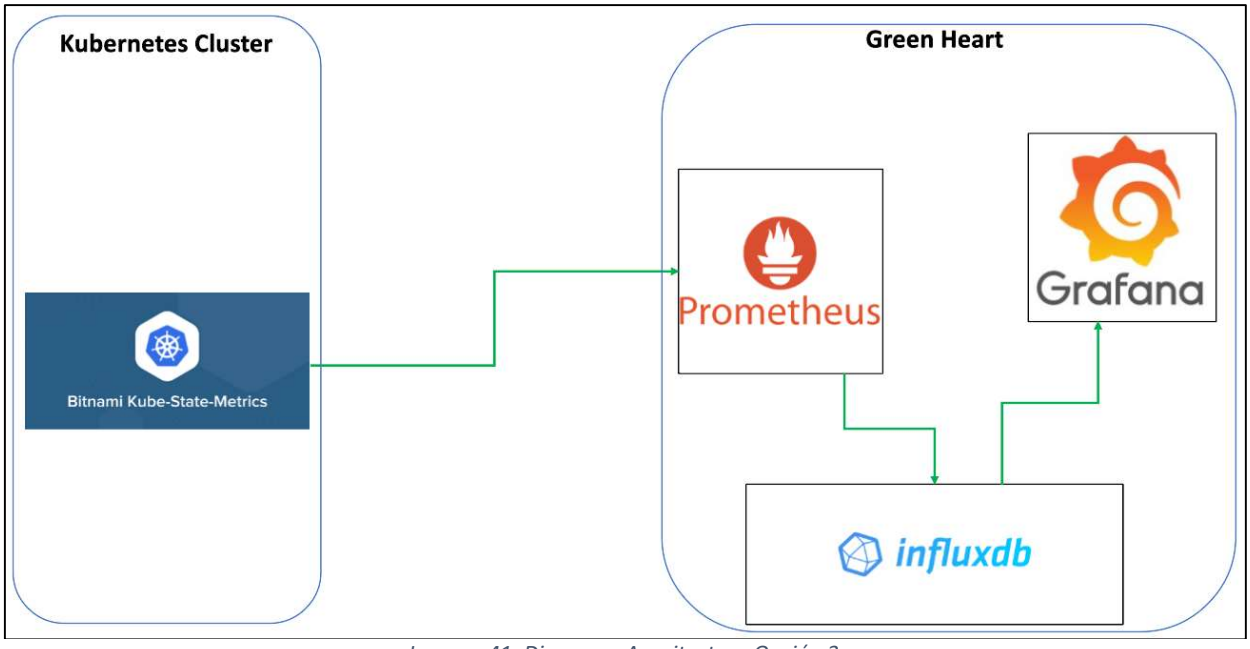

Imagen 41. Diagrama Arquitectura Opción 3

Esta opción consiste en instalar el componente de Kube-State-Metrics en el clúster de Kubernetes. Este componente expone las métricas del estado del clúster en un formato compatible con Prometheus. En GreenHeart se instala el componente de Prometheus que extrae las métricas expuestas por el componente Kube-State-Metrics y Prometheus envía las métricas a la base de datos de InfluxDB instalada en GreenHeart. Finalmente, Grafana instalado en GreenHeart accede a las métricas almacenadas en InfluxDB.

- **Ventajas**

  Es una opción poco invasiva para el clúster de Kubernetes porque solo se instala el componente Kube-State-Metrics. Al usar InfluxDB como mecanismo de almacenamiento de métricas entre Prometheus y Grafana logra una similitud con las bases de datos utilizadas en GreenHeart.

- **Desventajas**

  La información de las métricas es almacenada en dos bases de datos (Prometheus e InfluxDB), lo que genera duplicidad de información y con el tiempo implica mayor necesidad de capacidad de almacenamiento y costos, además de la duplicación de información.

Arquitectura - Opción 4

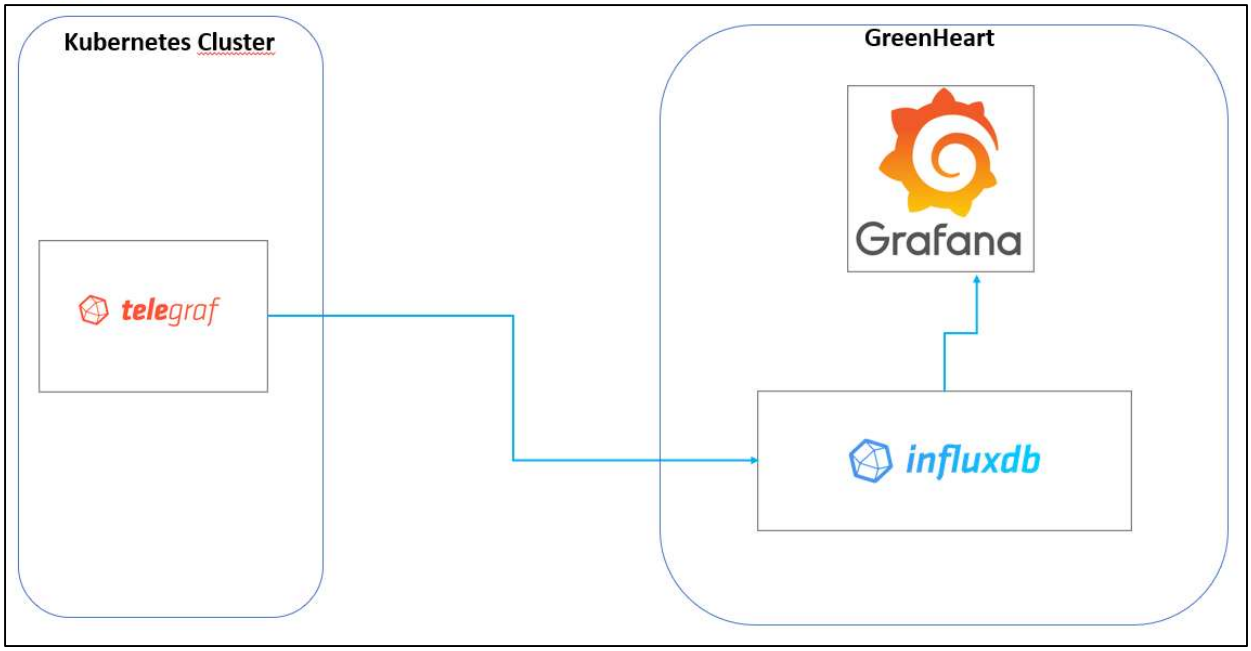

Imagen 42. Diagrama Arquitectura Opción 4

Esta opción consiste en instalar el componente Telegraf en el clúster de Kubernetes. En este se componente se configura un plugin que kube_inventory, que recopila las métricas del estado del clúster y las envía a la base de datos de InfluxDB instalada en GreenHeart. Finalmente, Grafana instalado en la plataforma GreenHeart accede a las métricas de InfluxDB.

- **Ventajas**

  Es una opción no tan invasiva para el clúster de Kubernetes porque solo se instala el componente de Telegraf.
  Al utilizar InfluxDB como mecanismo de almacenamiento de métricas se logra similitud con las bases de datos utilizadas en GreenHeart.

- **Desventajas**

  La cantidad de métricas que recopila kube_inventory es menor en comparación con Kube-State-Metrics o el *stack* de Prometheus, además que las métricas recopiladas tienen que ver más con el hardware que compone el clúster de Kubernetes y no del estado del clúster como tal.

6.1.5 Selección de la arquitectura a implementar para el colector de rendimiento

El área de Innovación de Green SQA tiene definidas las siguientes características para la selección de la arquitectura adecuada:

- Tener una alta portabilidad entre diferentes proveedores en la nube que tengan servicios Kubernetes.
- Que requiera la menor intervención posible en los clústeres de los clientes donde se requiera instalar el colector.

Con base a las características anteriormente definidas y al estudio de diferentes frentes de arquitectura ejecutados en la *Fase 1*, se selecciona la *Opción dos* para la implementación de la arquitectura del colector de rendimiento.

Las razones principales de la selección de esta arquitectura son las siguientes:

- Presenta una menor invasión en el clúster del cliente al momento de instalar y configurar.
- El componente Kube-State-Metrics se comunica nativamente con Kubernetes para obtener las métricas necesarias para el desarrollo del proyecto.
- Sobre el componente Kube-State-Metrics se pueden realizar configuraciones adicionales para mejorar las métricas que se recolectan y el modo de enviar estos datos a la plataforma.

Sintetizando lo anterior, se seleccionó el componente **Kube-State-Metrics**, un servicio que se conecta al API del servidor de Kubernetes y genera métricas sobre el estado de sus componentes, tales como el estado de los despliegues, nodos, pods, entre otras, y posteriormente estas métricas son extraídas y almacenadas por Prometheus. Se puede instalar y configurar tanto en ambientes administrados de Kubernetes en la nube como Google Kubernetes Engine (GKE), Elastic Cloud Kubernetes (EKS) y Azure Kubernetes Service (AKS), como en despliegues de Kubernetes administrados independientemente.

Adaptación y extensión del componente Kube-State-Metrics:

Se realizaron investigaciones y pruebas que permitieron identificar la viabilidad de modificar el componente Kube-State-Metrics de acuerdo con las necesidades del proyecto, para permitir asegurar la exposición de información de las métricas.

Creación de paquete Helm Chart para el componente Kube-State-Metrics con las adaptaciones y extensiones

En esta fase se realizó la creación de un paquete Helm Chart para un componente que se nombró como GSQA-Kube-Collector.

Helm es el gestor de paquetes para Kubernetes, que permitirá la administración e instalación del componente GSQA-Kube-Colector. Con Helm Charts es posible crear, versionar y publicar una aplicación Kubernetes; permitiendo la administración e instalación de las aplicaciones Kubernetes y el proceso de empaquetamiento de estas.

Así, el componente GSQA-Kube-Collector, corresponde el componente Kube-State-Metrics con las adaptaciones y extensiones correspondientes que permitan la captura y exposición de información de rendimiento.

Entre las modificaciones realizadas se encuentra la preconfiguración de los archivos de configuración del componente Kube-State-Metrics. Esto se realizó con el objetivo de que el componente solo exponga las métricas de rendimiento identificadas de alta importancia y no exponga las que no se requieren para el proyecto. Además, se realizó la modificación para exponer las métricas mediante el protocolo seguro HTTPS; y la instalación automática de las dependencias para la comunicación segura.

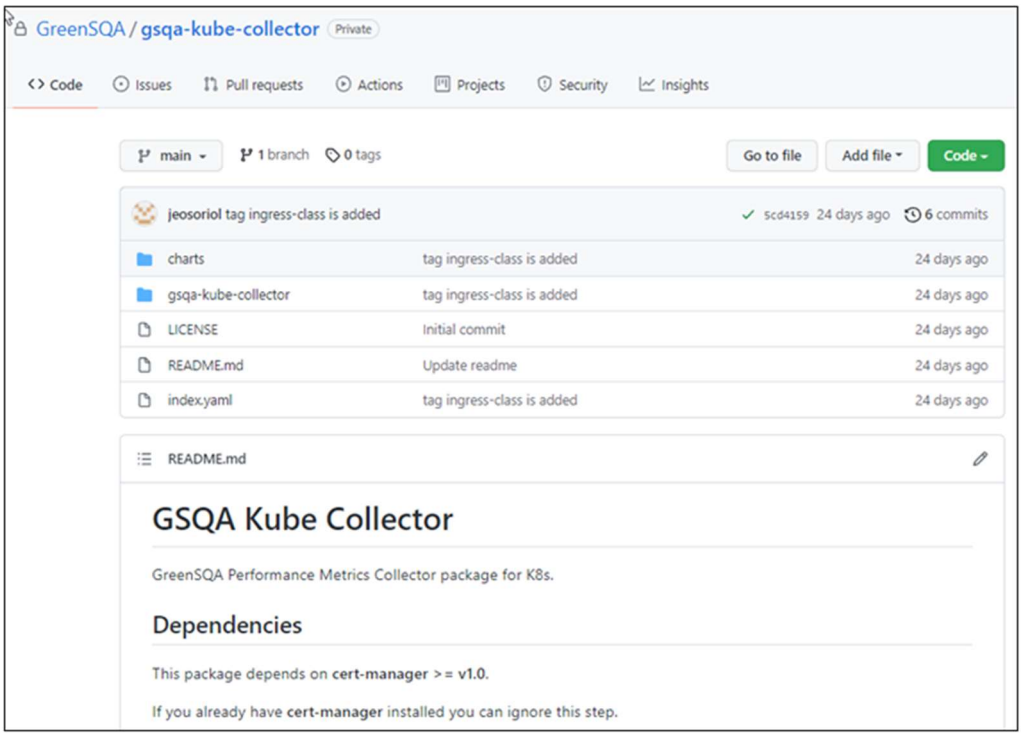

Imagen 43. Repositorio GitHub paquete GSQA-Kube-Collector

Las anteriores adaptaciones y extensiones fueron empaquetadas para su fácil instalaciones y distribución en un Helm Chart, que fue publicado en el repositorio GitHub oficial de la organización GreenSQA

7 DESARROLLO DE DASHBOARDS DE DESEMPEÑO

7.1 Plataforma de visualización de métricas

Componentes de la plataforma

Para la plataforma se seleccionaron los siguientes componentes:

- InfluxDB: Base de datos de series temporales para almacenar las métricas de ejecución de robots de prueba.
- API: servicio que recibe las métricas de los robots de prueba y almacena en la base de datos influxDB.
- Prometheus: raspador de métricas de rendimiento para Kubernetes y base de datos de series temporales.
- Grafana: interfaz visualización tableros y métricas de las ejecuciones de las pruebas.

Creación de archivo Docker-compose y repositorio para el despliegue automatizado de la plataforma

Docker-Compose es una herramienta para definir, ejecutar y desplegar aplicaciones Docker de varios contenedores. Mediante Docker-Compose, se crea un repositorio con un archivo YAML donde se configurar los servicios de plataforma y los diferentes archivos de configuración para su fácil despliegue y se publica en un repositorio privado de GreenSQA.

Docker-compose.yml

```
version: "3.3"

services:

  metrics_data:
    image: influxdb:1.8.3
    ports:
      - '8086:8086'
    volumes:
      - metrics_data_storage:/var/lib/influxdb
    environment:
      - INFLUXDB_DB=GSQA_DATA; CREATE DATABASE GSQA_DATA_TEST;
      - INFLUXDB_ADMIN_USER=${METRICS_DATA_USERNAME}
```

```
      - INFLUXDB_ADMIN_PASSWORD=/run/secrets/metrics_data_pwd
      - INFLUXDB_DATA_MAX_VALUES_PER_TAG=0
      - INFLUXDB_HTTP_FLUX_ENABLED=true
    restart: always
    secrets:
      - metrics_data_pwd

  prometheus:
    image: prom/prometheus:latest
    restart: unless-stopped
    volumes:
      - prometheus-config:/etc/prometheus/
      - prometheus-data:/prometheus
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - 9090:9090

  metrics_apis:
    image: greensqa.azurecr.io/greenheartapis:1.1.20221006.3
    expose:
      - "5000"
    restart: always

  analytics:
    image: grafana-oss:9.1.5
    ports:
      - '3000:3000'
    volumes:
      - analytics_storage:/var/lib/grafana
      - ${CERT_FILE_ROUTE}:/usr/share/grafana/ssl/cert_file.crt
      - ${CERT_KEY_ROUTE}:/usr/share/grafana/ssl/cert_key.key
    depends_on:
      - metrics_data
    environment:
      - GF_SECURITY_ADMIN_USER=${ANALYTICS_USERNAME}
      - GF_SECURITY_ADMIN_PASSWORD__FILE=/run/secrets/analytics_pwd
      - GF_INSTALL_PLUGINS=grafana-polystat-panel,dalvany-image-panel,aidanmountford-html-
panel,grafana-clock-panel
      - GF_SERVER_CERT_FILE=/usr/share/grafana/ssl/cert_file.crt
      - GF_SERVER_CERT_KEY=/usr/share/grafana/ssl/cert_key.key
      - GF_SERVER_PROTOCOL=https
    restart: always
    secrets:
```

```
    - analytics_pwd

secrets:
  metrics_data_pwd:
    file: ${METRICS_DATA_PWD_FILE_PATH}
  analytics_pwd:
    file: ${ANALYTICS_PWD_FILE_PATH}

volumes:
  metrics_data_storage:
  analytics_storage:
  prometheus-config:
  prometheus-data:
```

7.2 Dashboards de visualización

A continuación, se describen los pasos en el Diseño y Desarrollo de los cuatro niveles de dashboards de visualización. Para el desarrollo de los tableros los diferentes niveles, se diseñan primero las consultas a la base Prometheus para usar en la visualización de las métricas críticas.

Es importante resaltar que durante el diseño de los tableros se contó la participación y retroalimentación del equipo experto pruebas con el fin de validar que los tableros sean una herramienta útil en el proceso de pruebas y permitan visualizar la información necesaria para la ejecución de esta.

Critical Kubernetes Health Conditions	Metric	Prometheus Query
Crash Loops	Pods Running	sum (kube_pod_status_phase{phase="Running"})
	Pods Failed	sum (kube_pod_status_phase{phase="Failed"})
	Pods Pending	sum (kube_pod_status_phase{phase="Pending"})
	Pods Succeeded	sum (kube_pod_status_phase{phase="Succeeded"})

	Pods Unknown	sum (kube_pod_status_phase{phase="Unknown"})
Cluster State Metrics	**Pod Status Scheduled**	sum(kube_pod_status_scheduled{condition="true"})
	Pod Status Unscheduled	sum(kube_pod_status_scheduled{condition="false, uknown"})
	Number of Nodes	sum (kube_node_info)
	Nodes Unavailable	sum(kube_node_spec_unschedulable)
CPU Utilization	**CPU Usage**	CPU Requested / CPU Allocatable
	CPU Capacity	sum(kube_node_status_capacity{resource="cpu"})
	CPU Allocatable	sum(kube_node_status_allocatable{ resource="cpu"})
	CPU Requested	sum(kube_pod_container_resource_requests{resource="cpu"})
Memory Utilization	**Memory Usage**	Memory Requested / Memory Allocatable
	Memory Capacity	sum(kube_node_status_allocatable{resource="memory"})
	Memory Allocatable	sum(kube_node_status_capacity{resource="memory"})
	Memory Requested	sum(kube_pod_container_resource_requests{resource="memory"})
Pod Utilization	**Pod Usage**	Pod Requested / Pod Allocatable
	Pod Capacity	sum(kube_node_status_capacity{resource="pods"})

	Pod Allocatable	sum(kube_node_status_allocatable{resource="pods"})
	Pod Requested	sum(kube_pod_info)
Resources Requested and Limits by Containers	**CPU Containers Limit**	sum(kube_pod_container_resource_limits{resource="cpu"})
	CPU Containers Request	sum(kube_pod_container_resource_requests{resource="cpu"})
	Memory Containers Limit	sum(kube_pod_container_resource_limits{resource="memory"})
	Memory Containers Request	sum(kube_pod_container_resource_requests{resource="memory",
Disk Pressure	**Disk Pressure**	kube_node_status_condition{condition="DiskPressure", status="true"}
Memory Pressure	**Memory Pressure**	kube_node_status_condition{condition="MemoryPressure", status="true"}
PID Pressure	**PID Pressure**	kube_node_status_condition{condition="PIDPressure", status="true"}
Network Unavailable	**Network Unavailable**	kube_node_status_condition{condition="NetworkUnavailable", status="true"}
Job Failures	**Jobs Failed**	sum(kube_job_status_failed)
Persistent Volume Failures	**PV Failed**	sum(kube_persistentvolume_status_phase{phase="Failed"})
Pod Pending Delays	**Pods Pending**	sum (kube_pod_status_phase{phase="Pending"})
Deployment Glitches	**Deployment Desired**	sum(kube_deployment_spec_replicas)

	Deployment Observed	sum(kube_deployment_status_observed_generation)
StatefulSets Not Ready	**Replicas**	kube_statefulset_status_replicas_ready/kube_statefulset_status_replicas*100
DaemonSets Not Ready	**Daemonsets Desired**	sum(kube_daemonset_status_desired_number_scheduled)
	Daemonsets Observed	sum(kube_daemonset_status_observed_generation)

Tabla 3. Consultas a la base Prometheus para usar en la visualización de las métricas criticas

En el ***nivel 1*** se puede observar:

- Listado de proyectos.
- Promedio de satisfacción de usuario.
- Cantidad de ejecuciones, transacciones correctas y transacciones fallidas.
- Estado actual del clúster en uso de CPU, memoria y pods.

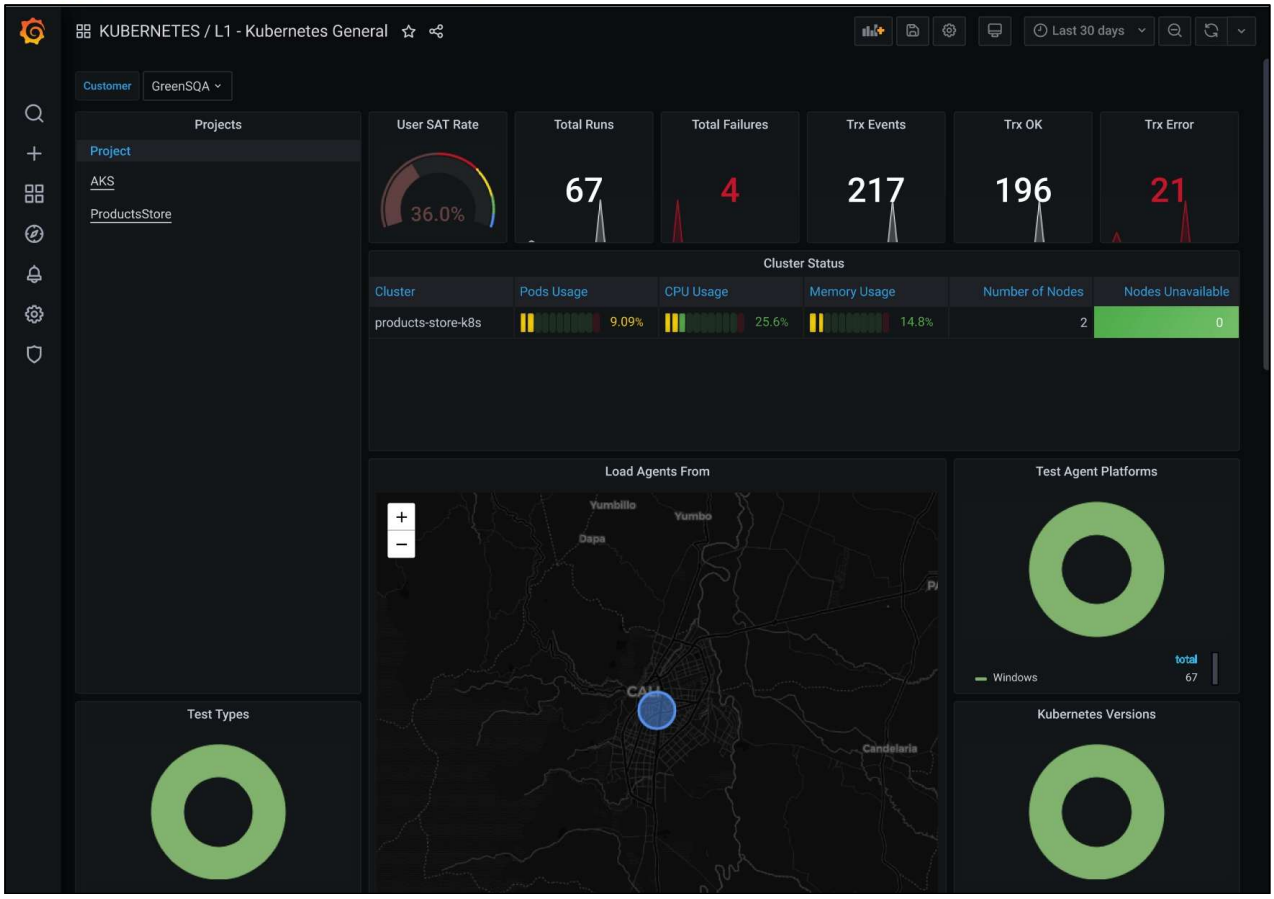

Imagen 44. Dashboard Nivel 1 - Dashboards información rendimiento AKS

En el ***nivel 2*** se puede observar:

- El listado de iteraciones.
- Gráfico del comportamiento de los recursos del clúster (uso de CPU, memoria y pods).
- Tabla resumen de las transacciones.
- Número de nodos en el clúster.
- Comportamiento de los pods, lo cual es muy importante para saber si hay pods fallidos, esto indica caídas de las aplicaciones desplegadas en los pods.
- Gráfico de satisfacción de usuario de los robots.
- Gráfico que muestra la cantidad de usuarios simultáneos, cantidad de transacciones y cantidad de errores.
- Tabla y distribución de los errores.

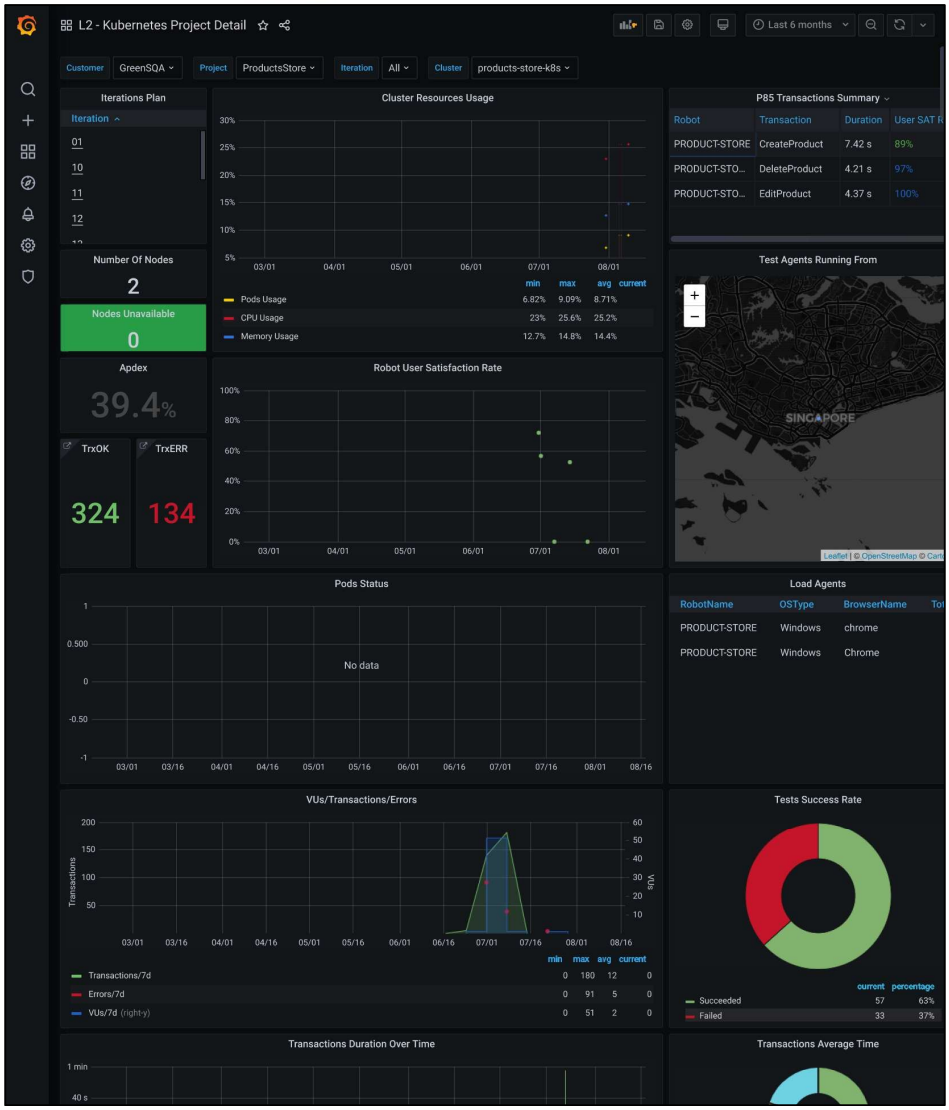

Imagen 45. Dashboard Nivel 2 - Dashboard información rendimiento AKS

En el ***nivel 3*** se puede observar:

- El estado del clúster en cuanto a uso de CPU, memoria y pods.
- Comportamiento de los nodos, como presión de memoria, disco, PID, no disponibilidad de red.
- Comportamiento de los pods, como detección de pods fallidos
- Información detallada de los containers
- Estado y comportamiento de los deployments
- Estado y comportamiento de los daemonset
- Estado y comportamiento de los Statefulsets

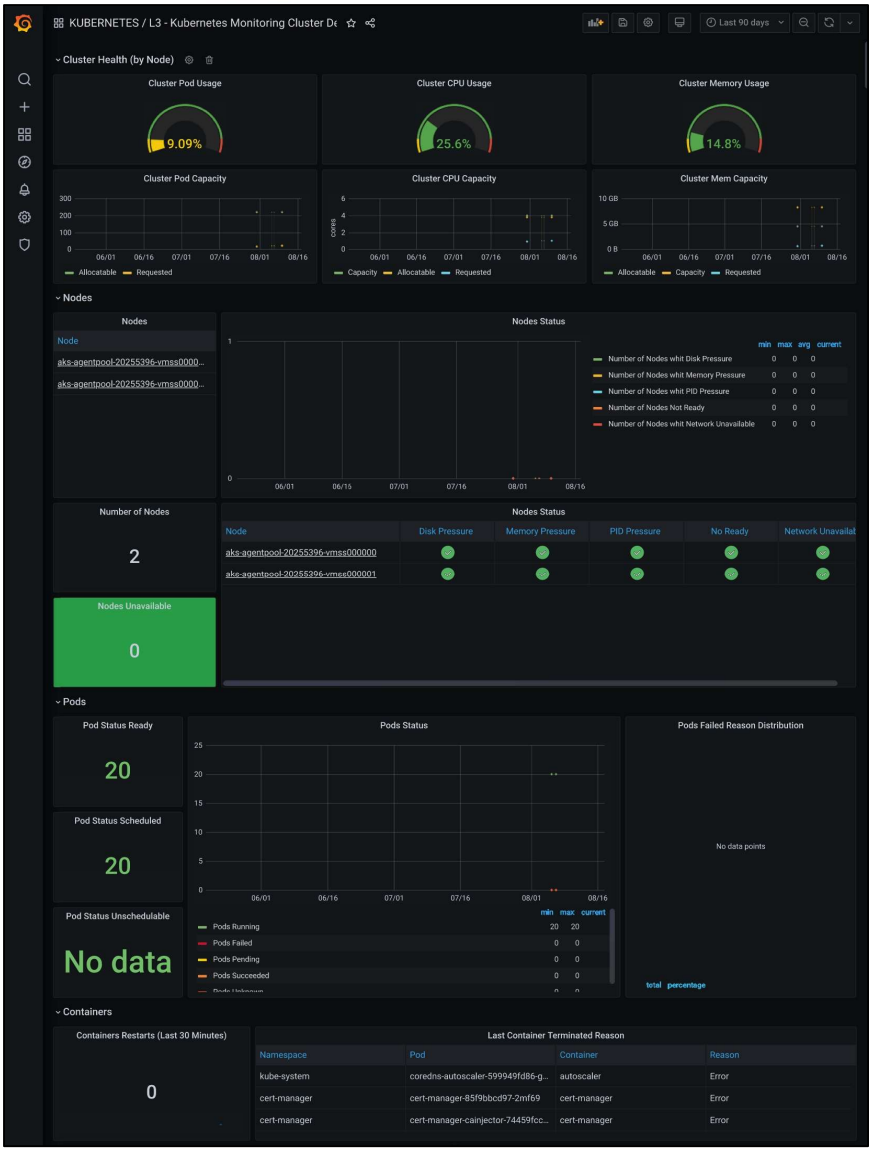

Imagen 46. Dashboard Nivel 3 - Dashboard información rendimiento AKS

En el ***nivel 4*** se puede observar:

El nivel 4 es muy similar al nivel 3, la única diferencia es que en el nivel 4 se puede observar la información para el nodo seleccionado, mientras en el nivel 3 se observa la información resumida de todos los nodos.

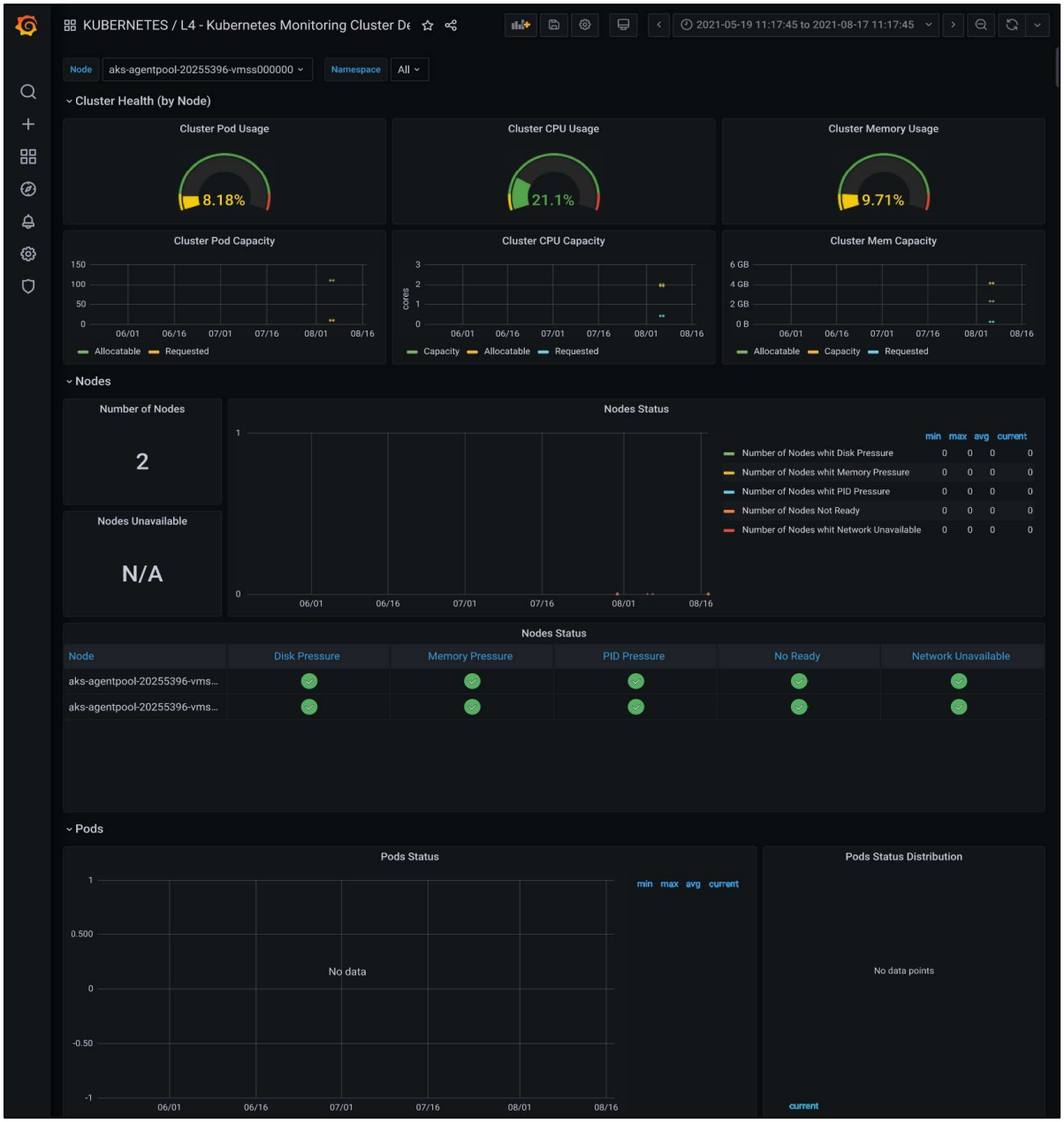

Imagen 47. Dashboard Nivel 4 - Dashboard información rendimiento AKS

8 PRUEBA DE LA PLATAFORMA

Con el objetivo de validar la utilidad de la de la plataforma desarrollada, se seleccionó la plataforma GreenHeart que es de propiedad de GreenSQA y se encuentra alojada en un clúster de Azure Kubernetes Service.

La prueba consistió en ejecutar iteraciones de pruebas de la estrategia RUPET (*Real User Performance Testing)* sobre el clúster en el cual se encuentra alojada GreenHeart. Una vez ejecutadas las iteraciones de la estrategia, junto al equipo de pruebas de GreenSQA validaron que los datos y resultados de las pruebas que se visualizaron los tableros de la plataforma fueran útiles en el proceso de evaluar el rendimiento de la plataforma bajo pruebas.

Para esto, al tiempo que se ejecutaron múltiples robots de prueba sobre GreenHeart, se realizó también monitoreo constante al clúster AKS donde se encuentra desplegada la GreenHeart por medio del componente **GSQA-Kube-Collector**. Posteriormente se comprobaron y analizaron los resultados del comportamiento de la aplicación durante las pruebas, y a su vez, el comportamiento asociado del clúster durante ese mismo periodo de tiempo.

El desarrollo de estas pruebas se realizó en las siguientes fases:

- ***Fase 1***. Pruebas o iteraciones RUPET sobre la plataforma GreenHeart desplegada en un clúster AKS.
- ***Fase 2***. Monitoreo continuo del clúster AKS por medio del componente gsqa-kube-collector durante el periodo de ejecución de las pruebas.
- ***Fase 3***. Análisis de los resultados de las pruebas o iteraciones; teniendo en cuenta el comportamiento, tanto de la aplicación como del clúster.

8.1 FASE 1. Pruebas o iteraciones RUPET sobre la plataforma GreenHeart desplegada en un clúster AKS

El primer paso en esta fase fue la creación de robots que ingresaran a la plataforma GreenHeart desplegada en un clúster AKS y a sus estrategias, de forma automatizada; simulando un usuario real ingresando a la aplicación. Los robots realizan las siguientes acciones automatizadas sobre la plataforma:

- ***Login:*** Ingresa a la plataforma y accede a través de una cuenta de usuario con permisos de lectura.
- ***Estrategia Funcional:*** Ingresa a la estrategia funcional en sus tres niveles: *nivel cliente*, *nivel proyecto* y *nivel evidencias*. Selecciona el cliente, proyecto y fecha, a

través de los controles de menú de los tableros gráficos (estos valores se obtienen desde unas variables preconfiguradas en los robots). Navega por los tableros verificando su correcta funcionalidad.

- ***Estrategia RUPET***: Ingresa a la estrategia RUPET en sus tres niveles: *nivel cliente, nivel proyecto* y *nivel nodo*. Selecciona el cliente, proyecto y fecha, a través de los controles de menú de los tableros gráficos (estos valores se obtienen desde unas variables preconfiguradas en los robots). Navega por los tableros verificando su correcta funcionalidad.
- ***Estrategia JMeter:*** Ingresa a la estrategia JMeter en sus tres niveles: *nivel cliente*, *nivel proyecto* y *nivel nodo*. Selecciona el cliente, proyecto y fecha, a través de los controles de menú de los tableros gráficos (estos valores se obtienen desde unas variables preconfiguradas en los robots). Navega por los tableros verificando su correcta funcionalidad.

Después de creados los robots, se configuraron unos *pipelines* en Azure DevOps. Estos pipelines permiten ejecutar múltiples robots de forma simultánea que ingresan a la aplicación y permiten generar carga y alta transaccionalidad sobre esta; lo cual permite probar el rendimiento (performance) de la aplicación.

Para este piloto se realizaron 12 iteraciones de pruebas, donde cada iteración fue preconfigurada con las variables de *cliente, proyecto*, *fecha* y *cantidad de usuarios paralelos*:

Iteración	**Fecha, cliente y proyecto estrategia Funcional**	**Fecha, cliente y proyecto estrategia RUPET**	**Fecha, cliente y proyecto estrategia JMeter**	**Cantidad máxima usuarios paralelos**	**Cantidad máxima robots ejecutados**
01	Últimos 45 días, GreenSQA, todos.	Últimos 20 días, GreenSQA, GreenBrain.	Últimos siete días, GSQA, GreenHeart.	20	40
02	Últimos 45 días, GreenSQA, todos.	Últimos 20 días, GreenSQA, GreenBrain.	Últimos siete días, GSQA, GreenHeart.	40	120
03	Últimos 45 días, GreenSQA, todos.	Últimos 20 días, GreenSQA, GreenBrain.	Últimos siete días, GSQA, GreenHeart.	60	240
04	Últimos 45 días, GreenSQA, todos.	Últimos 20 días, GreenSQA, GreenBrain.	Últimos siete días, GSQA, GreenHeart.	80	400
05	Últimos 60 días, GreenSQA, todos.	Últimos 30 días,	Últimos 15 días, GSQA, GreenHeart.	20	40

		GreenSQA, GreenBrain.			
06	Últimos 60 días, GreenSQA, todos.	Últimos 30 días, GreenSQA, GreenBrain.	Últimos 15 días, GSQA, GreenHeart.	40	120
07	Últimos 60 días, GreenSQA, todos.	Últimos 30 días, GreenSQA, GreenBrain.	Últimos 15 días, GSQA, GreenHeart.	60	240
08	Últimos 60 días, GreenSQA, todos.	Últimos 30 días, GreenSQA, GreenBrain.	Últimos 15 días, GSQA, GreenHeart.	80	400
09	Últimos 90 días, GreenSQATesting, todos.	Últimos 40 días, GreenSQA, GreenBrain.	Últimos 30 días, GSQA, GreenHeart.	20	40
10	Últimos 90 días, GreenSQATesting, todos.	Últimos 40 días, GreenSQA, GreenBrain.	Últimos 30 días, GSQA, GreenHeart.	40	120
11	Últimos 90 días, GreenSQATesting, todos.	Últimos 40 días, GreenSQA, GreenBrain.	Últimos 30 días, GSQA, GreenHeart.	60	240
12	Últimos 90 días, GreenSQATesting, todos.	Últimos 40 días, GreenSQA, GreenBrain.	Últimos 30 días, GSQA, GreenHeart.	80	400

Tabla 4. Resultado de las 12 iteraciones de las pruebas.

8.2 FASE 2. Monitoreo continuo del clúster AKS por medio del componente kube-collector durante el período de ejecución de las pruebas

Para realizar el monitoreo del clúster durante las pruebas se instaló el componente **GSQA-Kube-Collector** en el clúster; el cual recopila métricas del comportamiento del clúster y las envía a la plataforma de monitoreo para su respectivo análisis. La instalación del componente se ilustra a continuación:

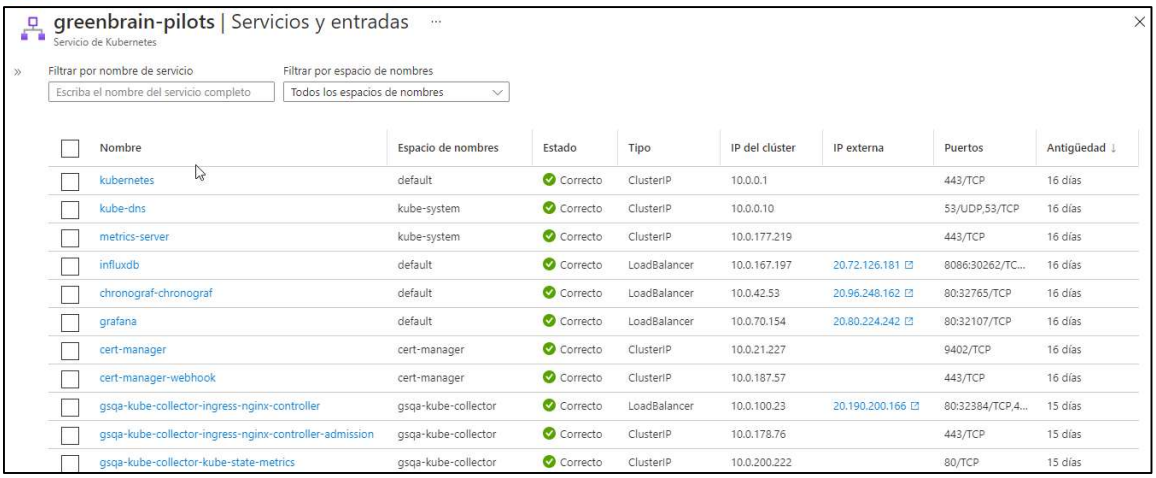

Imagen 48. Componentes instalados en clúster AKS

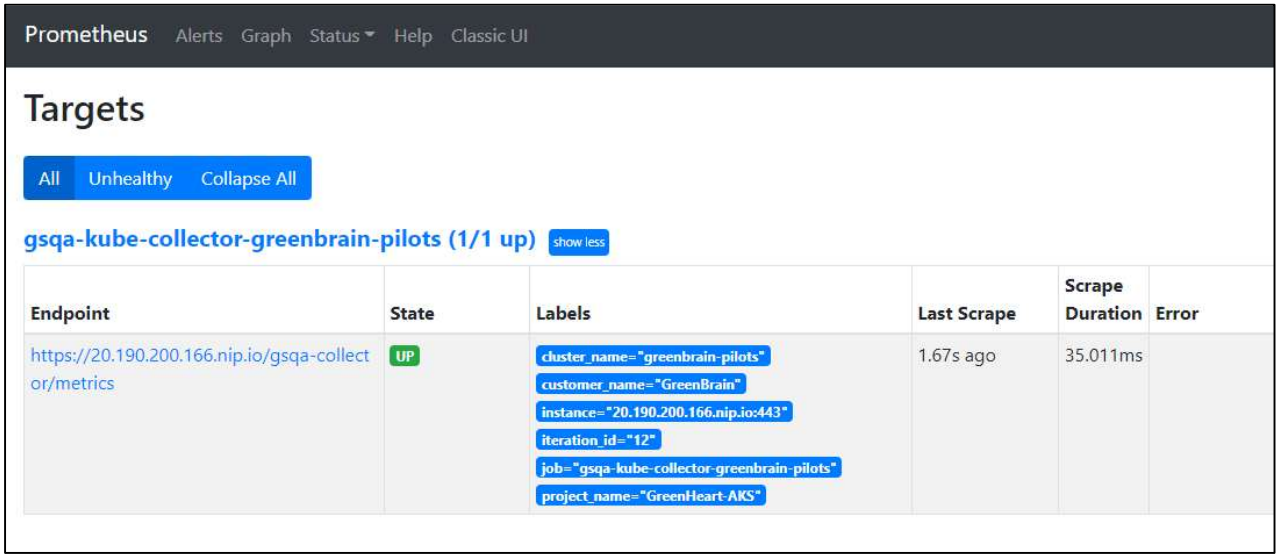

Imagen 49. Conexión de componente kube-collector a Prometheus

8.3 FASE 3. Análisis de los resultados de las pruebas o iteraciones teniendo en cuenta el comportamiento, tanto de la aplicación como del clúster

En el momento en que se realizaron las iteraciones de prueba el clúster AKS contaba con un escalamiento automático con un máximo de dos nodos, cada nodo tenía una capacidad de dos *cores* de CPU y cuatro Gb de memoria RAM. El análisis de los resultados evidenciados en los tableros de la plataforma para cada iteración y que se realizó en con junto al equipo de pruebas de GreenSQA, se describe a continuación:

8.3.1 Análisis Iteración 01

En la iteración 01 se utilizaron 20 robots ejecutándose de forma paralela, simulando 20 usuarios simultáneos ingresando a la aplicación y generando carga sobre esta. Durante todo el período de la prueba se ejecutaron 40 robots (20 robots en paralelo y el resto de forma consecutiva).

Se observa que todos los robots se ejecutaron correctamente, sin errores. El índice de satisfacción de usuario fue muy bueno (80%). Adicionalmente, se observa que el clúster tuvo un comportamiento estable, sin fallos ni variaciones:

Imagen 50. Iteración 01 - Cantidad de transacciones vs Usuarios simultáneos vs Cantidad de errores

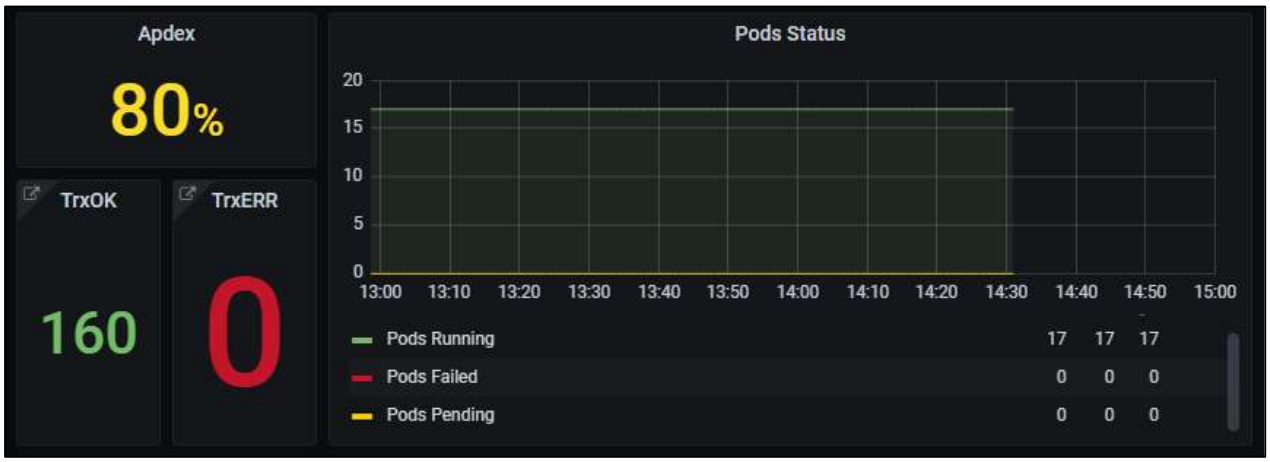

Imagen 51. Iteración 01 - Estado de los pods en el clúster

8.3.2 Análisis Iteración 02

En la iteración 02 se utilizaron 40 robots ejecutándose de forma paralela, simulando 40 usuarios simultáneos ingresando a la aplicación y generando carga sobre esta. Durante todo el período de la prueba se ejecutaron 120 robots (40 robots en paralelo y el resto de forma consecutiva). Se observa que la mayoría de los robots se ejecutaron correctamente; sin embargo, un porcentaje de estos se ejecutó con errores. El índice de satisfacción de usuario fue bueno cuando no hubo demasiada carga; pero cuando la

cantidad de usuarios simultáneos llegó al límite, la satisfacción de usuario decayó y aumentaron los errores.

Adicionalmente, en el momento en que ocurrieron los errores, se observa presión de memoria en el clúster y la caída del recurso *statefulset* de InfluxDB, que es la base de datos de GreenHeart. Por lo tanto, se concluye que la caída de InfluxDB fue provocada por presión de memoria en el clúster; probablemente por falta de recursos al contar solo con dos nodos o insuficiente asignación de recursos en el despliegue. Las evidencias de estas observaciones se ilustran a continuación:

Imagen 52. Iteración 02 - Porcentaje de satisfacción de usuario por robot, Transacciones vs Usuarios vs errores

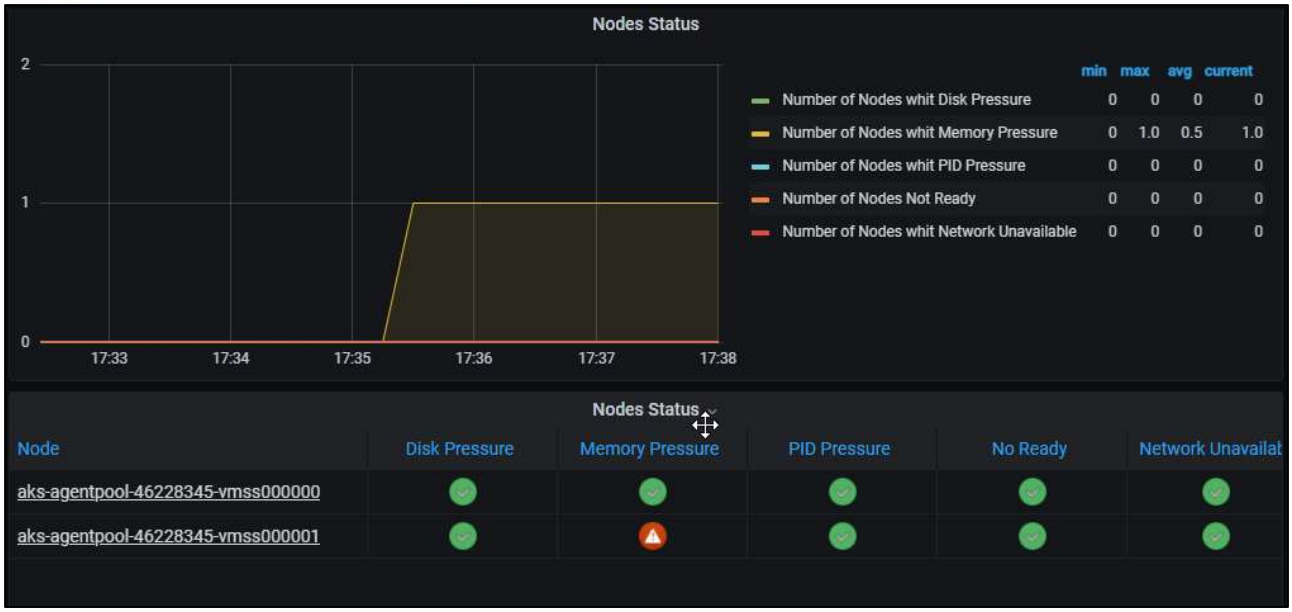

Imagen 53. Iteración 02 - Estado de los nodos del clúster

Imagen 54. Iteración 02 - Estado de los Statefulsets en el clúster

8.3.3 Análisis Iteración 03

En la iteración 03 se utilizaron 60 robots ejecutándose de forma paralela, simulando 60 usuarios simultáneos ingresando a la aplicación y generando carga sobre esta. Durante todo el período de la prueba se ejecutaron 240 robots (60 robots en paralelo y el resto de forma consecutiva). Se observa que una minoría de los robots se ejecutaron correctamente y un alto porcentaje de estos se ejecutó con errores. El índice de satisfacción de usuario fue bueno cuando no hubo demasiada carga; pero cuando la cantidad de usuarios simultáneos llegó aproximadamente a 50 usuarios paralelos la satisfacción de usuario decayó y aumentaron los errores.

Adicionalmente, en el momento en que ocurrieron los errores se observa presión de memoria en el clúster y la caída del recurso *statefulset* de InfluxDB, que es la base de datos de GreenHeart. Por lo tanto, se concluye que la caída de InfluxDB fue provocada por presión de memoria en el clúster; probablemente por falta de recursos al solo contar

con dos nodos o insuficiente asignación de recursos en el despliegue. Las evidencias de estas observaciones se ilustran a continuación:

Imagen 55. Iteración 03 - Porcentaje de satisfacción de usuario por robot, Transacciones vs Usuarios vs Errores

Imagen 56. Iteración 03 - Estado de los nodos del clúster

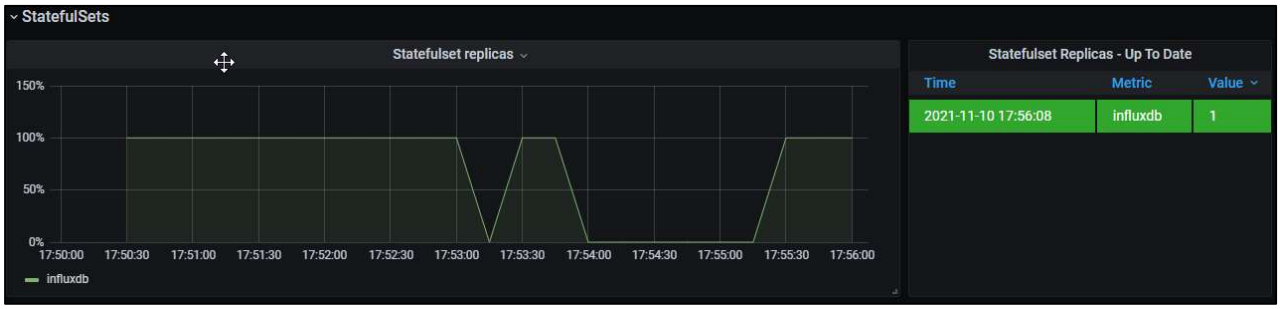

Imagen 57. Iteración 03 - Estado de los Statefulsets en el clúster

8.3.4 Análisis Iteración 04

En la iteración 04 se utilizaron 80 robots ejecutándose de forma paralela, simulando 80 usuarios simultáneos ingresando a la aplicación y generando carga sobre esta. Durante todo el período de la prueba se ejecutaron 400 robots (80 robots en paralelo y el resto de forma consecutiva). Se observa que una minoría de los robots se ejecutaron correctamente y un alto porcentaje de estos se ejecutó con errores. El índice de satisfacción de usuario fue bueno cuando no hubo demasiada carga; pero cuando la cantidad de usuarios simultáneos llegó aproximadamente a 50 usuarios paralelos, la satisfacción de usuario decayó y aumentaron los errores.

Adicionalmente, en el momento en que ocurrieron los errores, se observa presión de memoria en el clúster y la caída del recurso *statefulset* de InfluxDB, que es la base de datos de GreenHeart. Por lo tanto, se concluye que la caída de InfluxDB fue provocada por presión de memoria en el clúster; probablemente por falta de recursos al solo contar con dos nodos o insuficiente asignación de recursos en el despliegue. Las evidencias de estas observaciones se ilustran a continuación:

Imagen 58. Iteración 04 - Porcentaje de satisfacción de usuario por robot, Transacciones vs Usuarios vs Errores

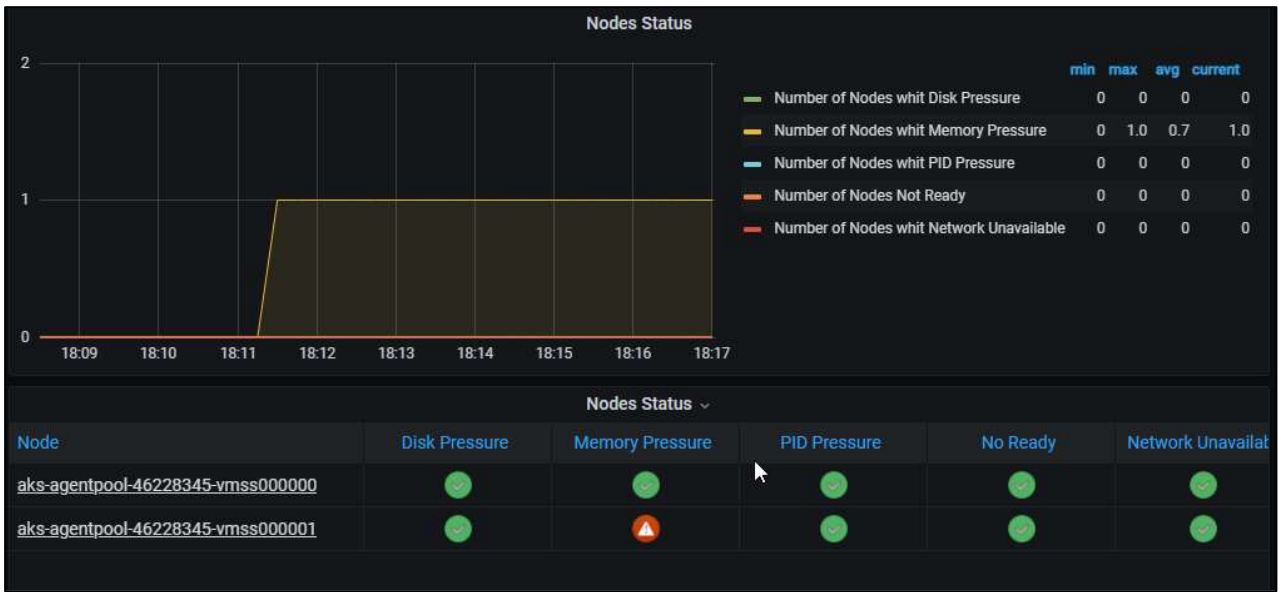

Imagen 59. Iteración 04 - Estado de los nodos del clúster

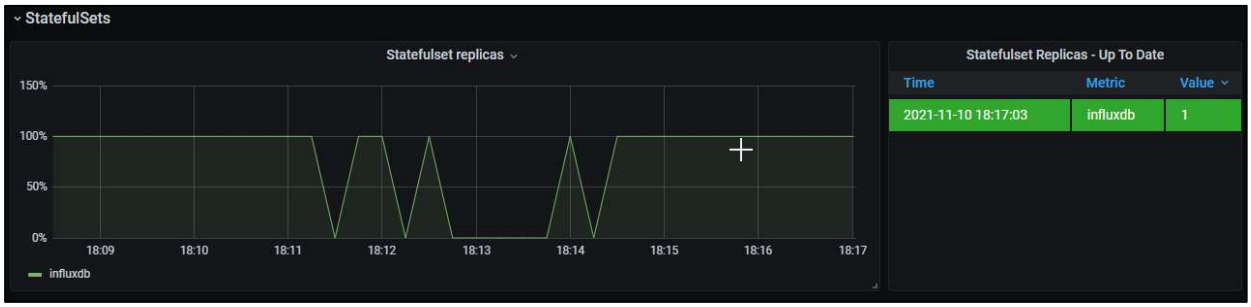

Imagen 60. Iteración 04 - Estado de los Statefulsets en el clúster

8.3.5 Análisis Iteraciones 05 a 08

Para las iteraciones 05 a 08 se observa un comportamiento muy similar al comportamiento mostrado anteriormente en las iteraciones 01 a 04; ya que se realizó el mismo proceso en donde se empezó con 20 usuarios paralelos en la primera iteración, 40 en la segunda, 60 en la tercera y 80 en la cuarta, y solo se modificaron las variables usadas por los robots, como *fecha*, *cliente* y *proyecto*.

8.3.6 Análisis Iteraciones 09 a 12

Para el rango de iteraciones 09 a 12 se observa que aun siendo pocos usuarios paralelos (20), la gran mayoría de los robots presentaron errores y baja satisfacción de usuario; además de presión de memoria en el clúster. Esto se debe a que, a diferencia de las iteraciones anteriores (en donde las fallas de InfluxDB fueron provocadas por la alta transaccionalidad de muchos usuarios simultáneos accediendo a la aplicación) en estas iteraciones se produjeron principalmente porque las variables, que eran parte de la ejecución de los robots, fueron configuradas con fechas hasta de tres meses atrás; lo cual exige más a la base de datos de InfluxDB provocando errores. Las evidencias de estas observaciones se ilustran a continuación:

Imagen 61. Iteraciones 09 a 12 - Porcentaje de satisfacción de usuario por robot, Transacciones vs Usuarios vs Errores

Imagen 62. Iteraciones 09 a 12 - Estado de los nodos del clúster

Imagen 63. Iteraciones 09 a 12 - Estado de los Statefulsets en el clúster

8.4 Resultado de las pruebas

Durante la realización de las pruebas y en conjunto con el equipo de pruebas de GreenSQA, se obtuvieron las siguientes conclusiones sobre el uso de la plataforma:

- Los tableros gráficos de la plataforma permitieron visualizar la información detallada, tanto de las pruebas automatizadas que se ejecutaron como del comportamiento y rendimiento del clúster donde se despliegan las aplicaciones bajo prueba. Esto permite realizar un análisis detallado de los resultados de las pruebas de rendimiento, ya que los tableros son altamente interpretables y la información se obtiene de forma instantánea y en línea.
- Gracias a los tableros gráficos de la plataforma se pudo detectar cuáles son los límites de usuarios paralelos y transacciones que soporta la aplicación bajo prueba, sin afectar su rendimiento. También se puede observar cuáles fueron las causas del bajo rendimiento de la aplicación cuando se superó el límite de transacciones y usuarios paralelos para rendimiento saludable. Esto es gracias a que se pudieron exponer de manera detallada cuáles fueron los fallos en el clúster donde estaba alojada la aplicación en el período de ejecución de la prueba.

9 MEJORAS Y TRABAJOS FUTUROS

Durante la realización del desarrollo y la ejecución de las pruebas de la plataforma se pudieron identificar las siguientes mejoras o trabajos futuros.

- Al llevar el clúster bajo pruebas a sus límites al punto de no responder y debido a que el componente captador y exportador de las métricas se encontraba instalado dentro del mismo clúster, no fue posible obtener las métricas de rendimiento. Por lo anterior se comienda modificar la arquitectura para que el componente captador de métricas se aloje fuera del clúster.

- Se visualiza como trabajo futuro identificar que métricas o resultados visualizados en los tableros puedan ser usados para genera conclusiones automáticas o reportes.

10 CONCLUSIONES

Con el objetivo de facilitar y automatizar la realización de pruebas de rendimiento en aplicaciones en la nube de tipo Serverless y Kubernetes, se propuso el desarrollo de un sistema que permitiera monitorear y visualizar métricas de rendimiento en clúster de Kubernetes y que esta información fuera fácilmente correlacionable con la información en las ejecuciones de pruebas. Para llevar a cabo el desarrollo del sistema se realizó una investigación exhaustiva del funcionamiento de la tecnológica Kubernetes y los servicios ofrecidos por los diferentes proveedores en la nube.

La investigación mostró tres proveedores principales de servicios de Kubernetes: Azure, AWS y Google Cloud. Después de investigarlos y realizar diferentes pruebas de concepto sobre el uso de estos servicios para despliegues de aplicaciones contenerizadas en ellos, se logró identificar que los servicios de Kubernetes ofrecidos exponen un API estándar que permite que los desarrollos sean independientes de los proveedores y sea usado así en cualquier servicio de nube (hallazgo de utilidad para futuras investigaciones). Siguiendo esta línea, la revisión de la literatura sobre Kubernetes permitió identificar los indicadores de salud de un clúster para seleccionar las métricas más relevantes y que cobran mayor valor en la realización de pruebas rendimiento. Con estas métricas determinadas, se identificó la mejor forma de captarlas en los clústeres; gracias a esto fue posible uno de los principales aportes del presente trabajo: el desarrolló un paquete Helm Chart que facilita la instalación del captador de métricas y sus dependencias para enviar las métricas a una base de datos centralizada.

A partir de lo anterior, se presenta una tecnología capaz de diseñar y desarrollar cuatro niveles de tableros que permiten el monitoreo y la visualización de las métricas de rendimiento en los clústeres para facilitar la realización de pruebas. Así mismo, mediante archivos docker-compose se encapsuló el despliegue la plataforma para su fácil instalación.

Una vez desarrollada la plataforma se realizaron pruebas de performance en una aplicación alojada en un cluster de Kubernetes y, gracias a los tableros gráficos de la plataforma, fue posible (tal como se planteó en la investigación) visualizar la información detallada, tanto de las pruebas automatizadas que se ejecutaron. como del comportamiento y rendimiento del clúster bajo prueba; para poder realizar un análisis detallado de los resultados de las pruebas de rendimiento ya que los tableros son altamente interpretables y la información se obtiene de forma casi instantánea y en línea.

A partir de todo el desarrollo se presenta una plataforma que permite detectar cuáles son los límites de usuarios paralelos y transacciones que soporta la aplicación bajo prueba, sin afectar su rendimiento. También, cuáles fueron las causas del bajo rendimiento de la aplicación cuando se superó el límite de transacciones y usuarios paralelos para rendimiento saludable. Esto es gracias a que se pudieron exponer de

manera detallada los fallos en el clúster donde estaba alojada la aplicación en el período de ejecución de las pruebas.

Es importante resaltar que durante el desarrollo y la ejecución de pruebas en el proyecto se contó la participación y retroalimentación del equipo experto pruebas con el fin de validar que la plataforma sea una herramienta útil en el proceso de pruebas.

Estos son resultados importantes debido a que con ellos se puede pensar en la plataforma como una herramienta que permita ofrecer a GreenSQA el servicio de pruebas de rendimientos sobre aplicaciones en la nube en clusters de Kubernetes. Se cumplió, por tanto, con el objetivo del proyecto de investigación y desarrollo, y con las premisas matrices de la presente investigación.

BIBLIOGRAFÍA

[1] S. Constaín and L. F. Trujillo, “BOLETÍN TRIMESTRAL De las tic,” pp. 1–49, 2019.

[2] “El negocio de la nube crece en Colombia.” https://www.semana.com/empresas/articulo/el-negocio-de-la-nube-crece-en-colombia/225355/ (accessed Mar. 06, 2023).

[3] GreenSQA, “Reporte nacional de calidad 2018-19,” 2019.

[4] R. Rabiser, K. Schmid, H. Eichelberger, M. Vierhauser, S. Guinea, and P. Grünbacher, “A domain analysis of resource and requirements monitoring: Towards a comprehensive model of the software monitoring domain,” *Inf. Softw. Technol.*, vol. 111, no. March, pp. 86–109, 2019, doi: 10.1016/j.infsof.2019.03.013.

[5] H. Eichelberger and K. Schmid, “Flexible resource monitoring of Java programs,” *J. Syst. Softw.*, vol. 93, pp. 163–186, 2014, doi: 10.1016/j.jss.2014.02.022.

[6] N. Delgado, A. Q. Gates, and S. Roach, “A taxonomy and catalog of runtime software-fault monitoring tools,” *IEEE Trans. Softw. Eng.*, vol. 30, no. 12, pp. 859–872, 2004, doi: 10.1109/TSE.2004.91.

[7] C. Northwood, *The Full Stack Developer*. Apress Berkeley, CA, 2018.

[8] R. K. Lenka, M. Rani Dey, P. Bhanse, and R. K. Barik, “Performance and Load Testing: Tools and Challenges,” *2018 Int. Conf. Recent Innov. Electr. Electron. Commun. Eng. ICRIEECE 2018*, pp. 2257–2261, 2018, doi: 10.1109/ICRIEECE44171.2018.9009338.

[9] W. Wang *et al.*, “Testing Cloud Applications under Cloud-Uncertainty Performance Effects,” *Proc. - 2018 IEEE 11th Int. Conf. Softw. Testing, Verif. Validation, ICST 2018*, pp. 81–92, 2018, doi: 10.1109/ICST.2018.00018.

[10] J. Holst and S. Schupp, “A New User Simulation-Based Approach for Application Performance Monitoring,” p. 65, 2018.

[11] T. M. Ahmed, C. P. Bezemer, T. H. Chen, A. E. Hassan, and W. Shang, “Studying the effectiveness of Application Performance Management (APM) tools for detecting performance regressions for web applications: An experience report,” *Proc. - 13th Work. Conf. Min. Softw. Repos. MSR 2016*, pp. 1–12, 2016, doi: 10.1145/2901739.2901774.

[12] D. Bardsley, L. Ryan, and J. Howard, “Serverless performance and optimization strategies,” *Proc. - 3rd IEEE Int. Conf. Smart Cloud, SmartCloud 2018*, pp. 19–26, 2018, doi: 10.1109/SmartCloud.2018.00012.

[13] A. Pereira Ferreira and R. Sinnott, “A performance evaluation of containers running on managed kubernetes services,” *Proc. Int. Conf. Cloud Comput. Technol. Sci. CloudCom*, vol. 2019-Decem, pp. 199–208, 2019, doi: 10.1109/CloudCom.2019.00038.

[14] “Docker Documentation | Docker Documentation.” https://docs.docker.com/ (accessed Nov. 28, 2021).

[15] “Kubernetes Documentation | Kubernetes.” https://kubernetes.io/docs/home/ (accessed Nov. 28, 2021).

[16] C. H. Kim, J. Rhee, K. H. Lee, X. Zhang, and D. Xu, “PerfGuard: Binary-centric application performance monitoring in production environments,” *Proc. ACM*

SIGSOFT Symp. Found. Softw. Eng., vol. 13-18-Nove, pp. 595–606, 2016, doi: 10.1145/2950290.2950347.

[17] A. Van Hoorn, J. Waller, and W. Hasselbring, “Kieker: A framework for application performance monitoring and dynamic software analysis,” *ICPE’12 - Proc. 3rd Jt. WOSP/SIPEW Int. Conf. Perform. Eng.*, pp. 247–248, 2012, doi: 10.1145/2188286.2188326.

[18] L. Miguel, “Propuesta de una solución de Monitoreo para sistemas del CeSPI,” 2017.

[19] “Magic Quadrant for Cloud Infrastructure and Platform Services.” https://www.gartner.com/doc/reprints?id=1-271OE4VR&ct=210802&st=sb (accessed Nov. 28, 2021).

[20] “Documentación de Azure Container Instances: contenedores sin servidor, a petición | Microsoft Docs.” https://docs.microsoft.com/es-es/azure/container-instances/ (accessed Nov. 28, 2021).

[21] “What is AWS Fargate? - Amazon ECS.” https://docs.aws.amazon.com/AmazonECS/latest/userguide/what-is-fargate.html (accessed Nov. 28, 2021).

[22] “Documentación de Azure Kubernetes Service | Microsoft Docs.” https://docs.microsoft.com/es-es/azure/aks/ (accessed Nov. 28, 2021).

[23] “Amazon Elastic Kubernetes Service Documentation.” https://docs.aws.amazon.com/eks/?id=docs_gateway (accessed Nov. 28, 2021).

[24] GreenSQA, “Metodología Analíticas de GreenSQA.” pp. 0–22, 2020.

ANEXO 1. Formato de encuestas para la identificación del stack tecnológico de los principales clientes de GreenSQA

Fecha Creación: 06-abril-2021
Equipo de Tecnología e Investigación Aplicada

Objetivo

Desarrollar un formato que permita encuestar a los principales clientes de GreenSQA, con el cual conocer el stack tecnológico con el que cuenta, principalmente relacionado a tecnologías de computación en la nube, contenerización y *serverless containers* en la nube.

Muestra

La encuesta se realizará a los principales clientes de GreenSQA.

Tipo de encuesta

La encuesta se realiza mediante entrevistas guiando a través del llenado de un formulario web

Principal información que se desea recopilar con la encuesta

- Dentro de los principales clientes de GreenSQA, cuáles son las arquitecturas de software más comunes.
- ¿Dentro de los principales clientes de GreenSQA, Se cuenta y con cuales sistemas de monitoreo?
- Dentro de los principales clientes de GreenSQA, cuales realizan procesos de integración/distribución continua (CI/CD).
- Dentro de los principales clientes de GreenSQA que realizan procesos de CI, que herramientas usan para estos.

- Dentro de los principales clientes de GreenSQA que realizan procesos de Ci, cuanto es el tiempo promedio de duración de los pipelines CI más rápidos y el promedio de los pipelines CI más demorados.
- Dentro de los principales clientes de GreenSQA que realizan procesos de CD, estos se hacen hacia servidores o sistemas sereverless. ¿En caso de ser realizados contra servidores estos son: onpremise, en la nube, hosting, ¿máquinas virtuales en la nube? ¿En caso de ser serverless con que proveedores se realiza?
-
- Dentro de los principales clientes de GreenSQA, cuales usan tecnologías de contenerización del software.
- Dentro de los principales clientes de GreenSQA, que tipo de tecnologías de contenerización de software usan.
- Dentro de los principales clientes de GreenSQA, que cantidad cuentan con componentes de cómputo en la nube dentro de su stack tecnológico.
- Dentro de los principales clientes de GreenSQA, qué tipo nube manejan Nube pública, Nube privada y Nube híbrida.
- Dentro de los principales clientes de GreenSQA que hacen uso de nubes públicas o hibridas, Cuales son los principales proveedores que de nubes públicas que usan.
- Dentro de los principales clientes de GreenSQA, que cantidad usan tecnologías *serverless containers* en la nube y cuáles.

Diseño de encuesta

La encuesta se comprenderá por las siguientes Preguntas:

1. **¿Indique el nombre de la organización?**

2. **¿Proyecto o proyectos de los cuales se describirá el Stack teológico?**

3. **¿Dentro de la organización o proyectos que modelos o tipos de arquitectura son usados?**

    - **Arquitectura de capas**
    - **Arquitectura guiada por eventos**
    - **Microkernel**
    - **Microservicios**

- Tuberías y filtros
- Batch
- Cliente servidor
- Maestro esclavo
- Otros_____

4. ¿Dentro de la organización o proyectos Se cuenta y con cuales sistemas de monitoreo?
   - Nagios
   - AppDynamics
   - Datadog
   - New Relic
   - Microsoft Insights
   - Prometheus
   - Graphite
   - Grafana
   - Kapacitor
   - Telegrapf
   - Kibana Logstash
   - Otros_____

5. ¿Dentro de la organización o proyectos se realizan procesos de integración/distribución continua (CI/CD)?
   - CI
   - CD

6. En caso de realizar procesos de CI ¿qué herramientas se usan para estos?
   - Jenkins
   - AzureDevOps
   - TeamCity
   - CircleCI
   - Otros_____

7. En caso de realizar procesos de CI ¿Cuánto duran en promedio los pipelines más cortos?
   - Entre 1 y 10 minutos
   - Entre 10 y 20 minutos
   - Entre 20 y 30 minutos
   - Más de 30 minutos

8. **En caso de realizar procesos de CI ¿Cuánto duran en promedio los pipelines más largos?**

    - **Entre 1 y 10 minutos**
    - **Entre 10 y 20 minutos**
    - **Entre 20 y 30 minutos**
    - **Más de 30 minutos**

9. **En caso de realizar procesos de CD ¿Estos procesos se realizan hacia servidores o sistemas sereverless?**

    - **Servidores**
    - **Sistemas sereverless**

10. **En caso de ser realizados contra servidores los procesos de CD ¿estos servidores son?**

    - **Onpremise**
    - **En la nube**
    - **Hosting**
    - **Máquinas virtuales en la nube**

11. **En caso de ser realizados contra sistemas sereverless los procesos de CD ¿esto se realiza mediante los siguientes provedores ?**

    - **Microsoft Azure.**
    - **Amazon Web Services.**
    - **Google Cloud Platform.**
    - **Oracle Cloud Infrastructure.**
    - **Alibaba Cloud.**
    - **Otros_____**

12. **¿Dentro de la organización se hace uso de tecnologías de contenerización de software?**

    NO_____
    SI_____(En caso de ser sí ¿Cuáles?)

    Runtimes de contenedores

    - **Docker**
    - **CRI-O**
    - **Rktlet**
    - **Containerd**

- **PouchContainer**
- **Kata Containers**
- **Podman**
- **Otros_____**

Gestión e implementación de clústeres

- **Kubernetes**
- **Apache Mesos**
- **Docker Swarm**
- **Docker Datacenter**
- **Otros_____**

13. **¿Dentro de la organización se cuenta con componentes de cómputo en la nube?**

    NO_____
    SI_____

14. **En caso de responder a la anterior pregunta si ¿qué tipo de nube manejan dentro de la organización?**

    - **Nube pública.**
    - **Nube privada.**
    - **Nube híbrida.**

15. **¿Dentro de la organización, cuáles son los proveedores de nubes públicas que usan?**

    - **Microsoft Azure.**
    - **Amazon Web Services.**
    - **Google Cloud Platform.**
    - **Oracle Cloud Infrastructure.**
    - **Alibaba Cloud.**
    - **Otros_____**

16. **¿Dentro de la organización se hace uso de servicios serverless containers en la nube?**

    NO_____
    SI_____(En caso de ser sí ¿Cuáles?)

    - **Azure: Azure Containers Instance**
    - **AWS: AWS Fargate**
    - **Google: Google Cloud Run**
    - **Otros_____**

17. **¿Dentro de la organización se hace uso de servicios de Kubernetes en la nube?:**

NO_____

SI_____(En caso de ser sí ¿Cuáles?)

- **Azure Container Service (AKS)**
- **Amazon ECS**
- **Google Kubernetes Engine (GKE)**
- **Otros_____**

ANEXO 2. Metodología completa de analíticas de Green SQA

Etapa I – Definir

En esta etapa se definen las necesidades de información estableciendo el dominio de información y quienes serán los proveedores responsables.

Establecer necesidades de información

En esta actividad, se identifican y definen cuáles son las necesidades de información, es decir, se identifican los aspectos relacionados con la calidad del sistema que se deben monitorear.

Clasificar necesidades por niveles y subniveles

Con las necesidades de información definidas, se procede a definir una jerarquía de información; es decir, las necesidades clasificadas por: niveles y subniveles padre, hijo, nieto. Estos niveles se definen en el documento “GSQA_FOR_Definición Analíticas”.

Estructurar grupos de información de cada nivel

Posteriormente, se estructuran los grupos de información que participan en cada nivel, esto se consigue agrupando las necesidades que guardan relación entre sí. Para esto es necesario:

- Nombrar el grupo.
- Definir las propiedades/etiquetas de los grupos.
- Definir los valores estrictamente numéricos de los grupos.

Definir proveedores de información

Los datos, que serán monitoreados por el sistema, son recolectados desde el mismo a través de dos enfoques diferentes: instrumentación o mediante monitoreo sintético a través de robots. En esta actividad, se decide cuál de los dos enfoques (o inclusive los dos enfoques) será utilizado.

Etapa II – Diseñar

En esta etapa se diseñan las métricas con las que será monitoreado el sistema y también se diseñan los tableros donde se mostrará la información de interés a través de gráficas significativas y fáciles de entender. El artefacto (GSQA-INS-Librería Gráficos.docx) contienen información de interés para facilitar el diseño de las analíticas.

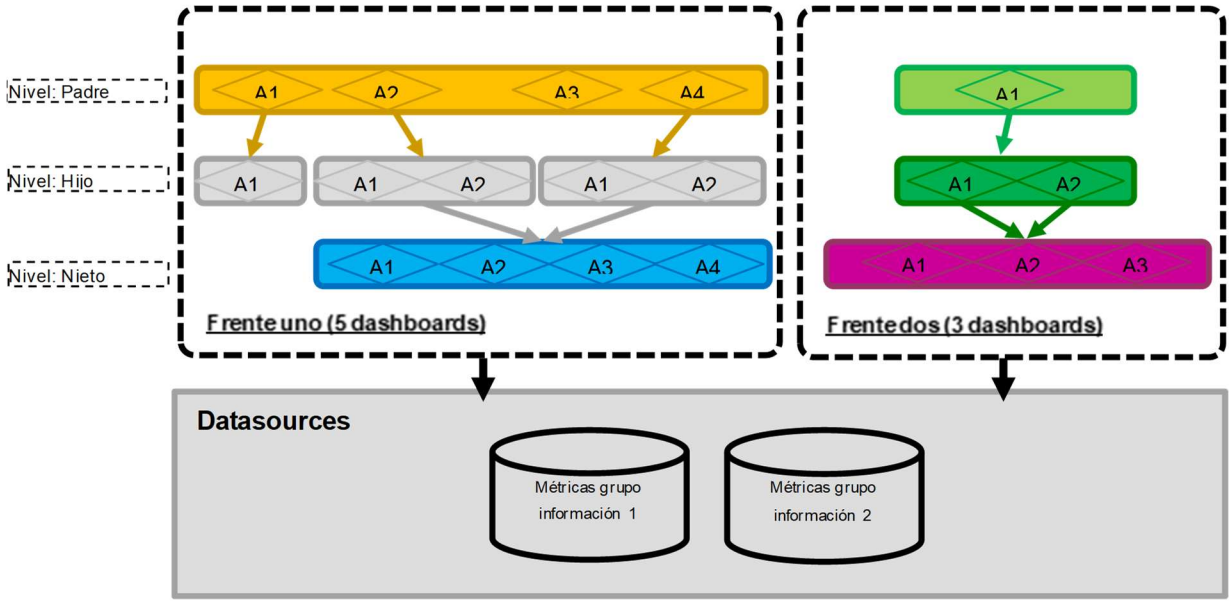

Imagen 64. Librería de gráficos

Modelar los Dashboards

Cada necesidad de información identificada y definida en etapas anteriores es expresada mediante gráficas analíticas, entre las que se incluyen gráficos de barras, líneas, mapa, torta, calor, tacómetro, tablas de datos, etc. Estas gráficas permiten conocer el estado actual del sistema y también realizar comparativos con datos de semanas, meses y/o años anteriores.

Modelar las estructuras de datos

Para monitorear los sistemas es necesario contar con las estructuras que permitan capturar los datos de interés que se desean analizar.

Crear la base de datos de serie de tiempo

Los datos monitoreados son almacenados en una base de datos de series de tiempo, estas son bases de datos optimizadas para series de datos de tiempo. Las series de tiempo son medidas o eventos, que son monitoreados a través del tiempo. Estos eventos

pueden ser métricas del servidor, monitoreo de desempeño de aplicaciones, datos de la red, datos de sensores, eventos, clics y otro tipo de datos analíticos.

Diseño de los proveedores de datos

Los datos de monitoreo son recolectados mediante dos técnicas diferentes: instrumentación y robots sintéticos. En esta actividad se diseña ya sea la instrumentación del código o los robots sintéticos, dependiendo de lo que se haya decidido en la etapa *Definir*.

Infraestructura requerida

Se implementa la infraestructura para la visualización de las métricas.

ANEXO 3. Métricas completas

Grupo	Métricas
1 Métricas de configmap	1.1 kube_configmap_created
	1.2 kube_configmap_info
	1.3 kube_configmap_metadata_resource_version
2 Métricas de daemonset	2.1 kube_daemonset_created
	2.2 kube_daemonset_labels
	2.3 kube_daemonset_metadata_generation
	2.4 kube_daemonset_status_current_number_scheduled
	2.5 kube_daemonset_status_desired_number_scheduled
	2.6 kube_daemonset_status_number_available
	2.7 kube_daemonset_status_number_misscheduled
	2.8 kube_daemonset_status_number_ready
	2.9 kube_daemonset_status_number_unavailable
	2.10 kube_daemonset_status_observed_generation
	2.11 kube_daemonset_status_updated_number_scheduled
3 Métricas de deployment	3.1 kube_deployment_created
	3.2 kube_deployment_labels
	3.3 kube_deployment_metadata_generation
	3.4 kube_deployment_spec_paused
	3.5 kube_deployment_spec_replicas
	3.6 kube_deployment_spec_strategy_rollingupdate_max_surge
	3.7 kube_deployment_spec_strategy_rollingupdate_max_unavailable
	3.8 kube_deployment_status_condition
	3.9 kube_deployment_status_observed_generation
	3.10 kube_deployment_status_replicas
	3.11 kube_deployment_status_replicas_available
	3.12 kube_deployment_status_replicas_unavailable
	3.13 kube_deployment_status_replicas_updated

4 Métricas de endpoint	4.1 kube_endpoint_address_available
	4.2 kube_endpoint_address_not_ready
	4.3 kube_endpoint_created
	4.4 kube_endpoint_info
	4.5 kube_endpoint_labels
5 Métricas de ingress	5.1 kube_ingress_created
	5.2 kube_ingress_info
	5.3 kube_ingress_labels
	5.4 kube_ingress_metadata_resource_version
	5.5 kube_ingress_path
	5.6 kube_ingress_tls
6 Métricas de mutatingwebhookconfiguration	6.1 kube_mutatingwebhookconfiguration_created
	6.2 kube_mutatingwebhookconfiguration_info
	6.3 kube_mutatingwebhookconfiguration_metadata_resource_version
7 Métricas de namespace	7.1 kube_namespace_created
	7.2 kube_namespace_labels
	7.3 kube_namespace_status_phase
8 Métricas de networkpolicy	8.1 kube_networkpolicy_created
	8.2 kube_networkpolicy_labels
	8.3 kube_networkpolicy_spec_egress_rules
	8.4 kube_networkpolicy_spec_ingress_rules
9 Métricas de node	9.1 kube_node_created
	9.2 kube_node_info
	9.3 kube_node_labels
	9.4 kube_node_role
	9.5 kube_node_spec_unschedulable
	9.6 kube_node_status_allocatable
	9.7 kube_node_status_capacity
	9.8 kube_node_status_condition

10 Métricas de Persistent volume	10.1 kube_persistentvolume_capacity_bytes
	10.2 kube_persistentvolume_claim_ref
	10.3 kube_persistentvolume_info
	10.4 kube_persistentvolume_labels
	10.5 kube_persistentvolume_status_phase
	10.6 kube_persistentvolumeclaim_access_mode
	10.7 kube_persistentvolumeclaim_info
	10.8 kube_persistentvolumeclaim_labels
	10.9 kube_persistentvolumeclaim_resource_requests_storage_bytes
	10.10 kube_persistentvolumeclaim_status_phase
11 Métricas de pod	11.1 kube_pod_container_info
	11.2 kube_pod_container_resource_limits
	11.3 kube_pod_container_resource_requests
	11.4 kube_pod_container_state_started
	11.5 kube_pod_container_status_last_terminated_reason
	11.6 kube_pod_container_status_ready
	11.7 kube_pod_container_status_restarts_total
	11.8 kube_pod_container_status_running
	11.9 kube_pod_container_status_terminated
	11.10 kube_pod_container_status_waiting
	11.11 kube_pod_created
	11.12 kube_pod_info
	11.13 kube_pod_init_container_info
	11.14 kube_pod_init_container_status_ready
	11.15 kube_pod_init_container_status_restarts_total
	11.16 kube_pod_init_container_status_running
	11.17 kube_pod_init_container_status_terminated
	11.18 kube_pod_init_container_status_terminated_reason
	11.19 kube_pod_init_container_status_waiting
	11.20 kube_pod_labels

	11.21 kube_pod_owner
	11.22 kube_pod_restart_policy
	11.23 kube_pod_spec_volumes_persistentvolumeclaims_info
	11.24 kube_pod_spec_volumes_persistentvolumeclaims_readonly
	11.25 kube_pod_start_time
	11.26 kube_pod_status_phase
	11.27 kube_pod_status_ready
	11.28 kube_pod_status_reason
	11.29 kube_pod_status_scheduled
	11.30 kube_pod_status_scheduled_time
	11.31 kube_poddisruptionbudget_created
	11.32 kube_poddisruptionbudget_status_current_healthy
	11.33 kube_poddisruptionbudget_status_desired_healthy
	11.34 kube_poddisruptionbudget_status_expected_pods
	11.35 kube_poddisruptionbudget_status_observed_generation
	11.36 kube_poddisruptionbudget_status_pod_disruptions_allowed
12 Métricas de replicaset	12.1 kube_replicaset_created
	12.2 kube_replicaset_labels
	12.3 kube_replicaset_metadata_generation
	12.4 kube_replicaset_owner
	12.5 kube_replicaset_spec_replicas
	12.6 kube_replicaset_status_fully_labeled_replicas
	12.7 kube_replicaset_status_observed_generation
	12.8 kube_replicaset_status_ready_replicas
	12.9 kube_replicaset_status_replicas
13 Métricas de secret	13.1 kube_secret_created
	13.2 kube_secret_info
	13.3 kube_secret_labels
	13.4 kube_secret_metadata_resource_version
	13.5 kube_secret_type

14 Métricas de service	14.1 kube_service_created
	14.2 kube_service_info
	14.3 kube_service_labels
	14.4 kube_service_spec_type
	14.5 kube_service_status_load_balancer_ingress
15 Métricas de statefulset	15.1 kube_statefulset_created
	15.2 kube_statefulset_labels
	15.3 kube_statefulset_metadata_generation
	15.4 kube_statefulset_replicas
	15.5 kube_statefulset_status_current_revision
	15.6 kube_statefulset_status_observed_generation
	15.7 kube_statefulset_status_replicas
	15.8 kube_statefulset_status_replicas_current
	15.9 kube_statefulset_status_replicas_ready
	15.10 kube_statefulset_status_replicas_updated
	15.11 kube_statefulset_status_update_revision

ANEXO 4. Tabla de métricas críticas.

Critical Kubernetes Health Conditions	Métrica
1. Crash Loops	Pods Failed
	Pods Pending
	Pods Succeeded
	Pods Unknown
2. Cluster State Metrics	Pod Status Scheduled
	Pod Status Unschedulable
	Number of Nodes
	Nodes Unavailable
3. CPU Utilization	CPU Usage
	CPU Capacity
	CPU Allocatable
	CPU Requested
4. POD Utilization	POD Usage
	POD Capacity
	POD Allocatable
	POD Requested
6. Memory Utilization	Memory Usage
	Memory Capacity
	Memory Allocatable
	Memory Requested
10. Disk Pressure	Disk Pressure
11. Memory Pressure	Memory Pressure

12. PID Pressure	PID Pressure
13. Network Unavailable	Network Unavailable
14. Resources Requested and Limits by Containers	CPU Containers Limit
	CPU Containers Request
	Memory Containers Limit
	Memory Containers Request
18. Job Failures	Jobs Failed
19. Persistent Volume Failures	PV Failed
20. Pod Pending Delays	Pods Pending
21. Deployment Glitches	Deployment Desired
	Deployment Observed
22. StatefulSets Not Ready	Replicas
23. DaemonSets Not Ready	Daemonsets Desired
	Daemonsets Observed

ANEXO 5. Necesidades de información por nivel

Necesidad de información	Componente Grafico	Origen de los datos
Tabla de listado de proyectos Kubernetes performance.	Tabla	AIMaps
Distribución de tipos de test.	Circle Chart	AIMaps
Distribución de plataformas de agentes de pruebas.	Circle Chart	AIMaps
Distribución de versiones de Kubernetes de los clústeres(backend).	Circle Chart	Proveedores de métricas de Kubernetes
Ubicación de los agentes de pruebas.	World Map	AIMaps
Porcentaje de satisfacción de usuario.	Gauge	AIMaps
Cantidad total de ejecuciones	Panel Text	AIMaps
Cantidad total de ejecuciones fallidas	Panel Text	AIMaps
Cantidad total de transacciones	Panel Text	AIMaps
Cantidad total de transacciones fallidas	Panel Text	AIMaps
Estado General del clúster (porcentaje de uso de CPU, memoria y pods).	Table	Proveedores de métricas de Kubernetes

Tabla 5. Nivel 1 de información.

Necesidad de información	Componente Grafico	Origen de los datos
Cantidad de nodos del clúster.	Panel Text	Captor de métricas de Kubernetes
Cantidad de nodos no disponibles.	Panel Text	Captor de métricas de Kubernetes

APDEX score (Application Performance Index).	Panel Text	AIMaps
Estado de los pods.	Line Chart	Capator de métricas de Kubernetes
Ubicación de agentes de pruebas.	World Map	AIMaps
Porcentaje de satisfacción de usuario de los robots.	Line Chart	AIMaps
Distribución de plataformas de agentes de pruebas.	Circle Chart	AIMaps
Comparación entre la cantidad de usuarios paralelos versus cantidad de transacciones versus cantidad de errores.	Line Chart	AIMaps
Distribución de ejecuciones correctas versus ejecuciones fallidas.	Circle Chart	AIMaps
Tiempos de respuesta de las transacciones.	Line Chart	AIMaps
Distribución de tiempos de respuesta de las transacciones.	Circle Chart	AIMaps
Cantidad de transacciones fallidas.	Line Chart	AIMaps
Distribución de transacciones fallidas.	Circle Chart	AIMaps
Descripción de errores en las transacciones.	Table	AIMaps
Distribución de errores en las transacciones.	Circle Chart	AIMaps
Tiempos de eventos de navegación.	Line Chart	AIMaps
Distribución de tiempos de eventos de navegación	Circle Chart	AIMaps
Porcentaje de uso de CPU de los agentes de prueba. (Frontend)	Points Chart	AIMaps

Porcentaje de uso de memoria de los agentes de prueba. (Frontend)	Points Chart	AIMaps
Cantidad de fallas mayores y menores de memoria de los agentes de prueba. (Frontend)	Points Chart	AIMaps
Cantidad de bytes transferidos por la red en los agentes de prueba. (Frontend)	Points Chart	AIMaps

Tabla 6. Nivel 2 de información

Necesidad de información	Componente Grafico	Origen de los datos
Porcentaje de uso de pods en clúster.	Gauge	Captor de métricas de Kubernetes
Porcentaje de uso de CPU en clúster.	Gauge	Captor de métricas de Kubernetes
Porcentaje de uso de memoria en clúster.	Gauge	Captor de métricas de Kubernetes
Capacidad de pods en clúster.	Line Chart	Captor de métricas de Kubernetes
Capacidad de CPU en clúster.	Line Chart	Captor de métricas de Kubernetes
Capacidad de memoria en clúster.	Line Chart	Captor de métricas de Kubernetes
Listado de nodos del clúster.	Table	Captor de métricas de Kubernetes
Cantidad de nodos en clúster.	Panel Text	Captor de métricas de Kubernetes
Cantidad de nodos no disponibles en clúster.	Panel Text	Captor de métricas de Kubernetes
Estado de los nodos (ok, presión de memoria, disco o PID)	Line Chart	Captor de métricas de Kubernetes
Cantidad de pods en estado listo.	Panel Text	Captor de métricas de Kubernetes

Cantidad de pods en estado programado.	Panel Text	Captor de métricas de Kubernetes
Cantidad de pods en estado no programado.	Panel Text	Captor de métricas de Kubernetes
Estado de los pods.	Line Chart	Captor de métricas de Kubernetes
Distribución de razón de falló de los pods.	Circle Chart	Captor de métricas de Kubernetes
Cantidad de contenedores reiniciados en últimos 30 minutos.	Panel Text	Captor de métricas de Kubernetes
Última razón de terminación de contenedor.	Panel Text	Captor de métricas de Kubernetes
Estado de contenedores.	Line Chart	Captor de métricas de Kubernetes
Cantidad de requerimientos y límites de contenedores.	Line Chart	Captor de métricas de Kubernetes
Cantidad de Jobs desplegados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de Jobs activos.	Panel Text	Captor de métricas de Kubernetes
Cantidad de Jobs fallidos.	Panel Text	Captor de métricas de Kubernetes
Cantidad de persistent volumes enlazados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de persistent volumes desenlazados.	Panel Text	Captor de métricas de Kubernetes
Estado de persistent volumes.	Line Chart	Captor de métricas de Kubernetes
Cantidad de deployments desplegados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de deployments no disponibles.	Panel Text	Captor de métricas de Kubernetes

Cantidad de deployments actualizados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de deployments deseados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de deployments observados.	Panel Text	Captor de métricas de Kubernetes
Estado de deployments.	Line Chart	Captor de métricas de Kubernetes
Cantidad de DaemonSets listos.	Panel Text	Captor de métricas de Kubernetes
Cantidad de DaemonSets actualizados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de DaemonSets no disponibles.	Panel Text	Captor de métricas de Kubernetes
Cantidad de DaemonSets deseados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de DaemonSets observados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de StatefulSets listos.	Panel Text	Captor de métricas de Kubernetes

Tabla 7. Nivel 3 de información

Necesidad de información	Componente Grafico	Origen de los datos
Porcentaje de uso de pods en clúster.	Gauge	Captor de métricas de Kubernetes
Porcentaje de uso de CPU en clúster.	Gauge	Captor de métricas de Kubernetes
Porcentaje de uso de memoria en clúster.	Gauge	Captor de métricas de Kubernetes
Capacidad de pods en clúster.	Line Chart	Captor de métricas de Kubernetes
Capacidad de CPU en clúster.	Line Chart	Captor de métricas de Kubernetes

Capacidad de memoria en clúster.	Line Chart	Captor de métricas de Kubernetes
Listado de nodos en clúster.	Table	Captor de métricas de Kubernetes
Cantidad de nodos en clúster.	Panel Text	Captor de métricas de Kubernetes
Cantidad de nodos no disponibles en clúster.	Panel Text	Captor de métricas de Kubernetes
Estado de los nodos (ok, presión de memoria, disco o PID)	Line Chart	Captor de métricas de Kubernetes
Cantidad de pods en estado listo.	Panel Text	Captor de métricas de Kubernetes
Cantidad de pods en estado programado.	Panel Text	Captor de métricas de Kubernetes
Cantidad de pods en estado no programado.	Panel Text	Captor de métricas de Kubernetes
Estado de los pods.	Line Chart	Captor de métricas de Kubernetes
Distribución de razón de falló de los pods.	Circle Chart	Captor de métricas de Kubernetes
Cantidad de contenedores reiniciados en últimos 30 minutos.	Panel Text	Captor de métricas de Kubernetes
Última razón de terminación de contenedor.	Panel Text	Captor de métricas de Kubernetes
Estado de contenedores.	Line Chart	Captor de métricas de Kubernetes
Cantidad de requerimientos y límites de contenedores.	Line Chart	Captor de métricas de Kubernetes
Cantidad de Jobs desplegados.	Panel Text	Captor de métricas de Kubernetes

Cantidad de Jobs activos.	Panel Text	Captor de métricas de Kubernetes
Cantidad de Jobs fallidos.	Panel Text	Captor de métricas de Kubernetes
Cantidad de persistent volumes enlazados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de persistent volumes desenlazados.	Panel Text	Captor de métricas de Kubernetes
Estado de persistent volumes.	Line Chart	Captor de métricas de Kubernetes
Cantidad de deployments desplegados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de deployments no disponibles.	Panel Text	Captor de métricas de Kubernetes
Cantidad de deployments actualizados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de deployments deseados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de deployments observados.	Panel Text	Captor de métricas de Kubernetes
Estado de deployments.	Line Chart	Captor de métricas de Kubernetes
Cantidad de DaemonSets listos.	Panel Text	Captor de métricas de Kubernetes
Cantidad de DaemonSets actualizados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de DaemonSets no disponibles.	Panel Text	Captor de métricas de Kubernetes
Cantidad de DaemonSets deseados.	Panel Text	Captor de métricas de Kubernetes

Cantidad de DaemonSets observados.	Panel Text	Captor de métricas de Kubernetes
Cantidad de StatefulSets listos.	Panel Text	Captor de métricas de Kubernetes

Tabla 8. Nivel 4 de información