Estructuración de los principios SRE en una organización

Juan Esteban Ospina Escobar

Informe de práctica para optar al título de Ingeniero de Sistemas

Asesor

Jaime Humberto Fonseca Espinal, Ingeniero de Sistemas especialista en ciencias electrónicas e Informáticas

Universidad de Antioquia

Facultad de Ingeniería

Ingeniería de Sistemas

Medellín, Antioquia, Colombia

2022

Agradecimientos

A cada profesor de la Facultad de Ingeniería de la Universidad de Antioquia, quienes acompañaron mi proceso durante todos estos semestres.

A la empresa guane Enterprises por la confianza depositada en mí y por todos estos meses de aprendizaje y desarrollo profesional.

TABLA DE CONTENIDO

RESUMEN.....7

ABSTRACT.....8

I. INTRODUCCIÓN.....9

II. PLANTEAMIENTO DEL PROBLEMA 10

III. OBJETIVOS.....11

IV. MARCO TEÓRICO.....12

V. METODOLOGÍA 14

VI. RESULTADOS.....16

VII. CONCLUSIONES 23

REFERENCIAS.....24

LISTA DE FIGURAS

Fig. 1 Estructura de un proyecto. 17
Fig. 2 Configuración básica de un servicio de auto escalamiento de KEDA ……19
Fig. 3. Creación de la imagen ……20
Fig 4. Comportamiento de la CPU ……21
Fig 5. Alerta CPU ……..……22

SIGLAS, ACRÓNIMOS Y ABREVIATURAS

GCP	Google cloud platform
RBAC	Role Based Access Control
SLI	Service Level Agreement
SLO	Service Level Objectives
SLA	Service Level Indicators
IDE	Integrated development environment
SRE	Site Reliability Engineering

RESUMEN

La industria de la tecnología, más específicamente en las empresas que trabajan en las T.I (tecnologías de la información) desean que sus procesos sean óptimos en cualquier etapa del desarrollo, logrando así el mayor beneficio posible. Por esto implementan metodologías de Devops (metodologías que buscan tener un software de mayor calidad) en sus proyectos, ya que esto les facilita tener productos de mayor calidad haciendo uso de sus estándares, además de utilizar herramientas que ayuden en estos procesos y provean información útil, tanto para una buena toma de decisiones, como para mejoras en el mantenimiento de las aplicaciones.

En el desarrollo de la práctica se obtuvieron excelentes resultados, haciendo uso de nuevas metodologías, herramientas y procesos. Demostrando que al implementar principios de SRE (por sus siglas en Ingles Site Reliability Engineering o en español Ingeniería de Confiabilidad del Sitio) en una organización o empresa, se mejoran diferentes aspectos de calidad en las aplicaciones, proporcionando mejoras en el proceso de desarrollo y en la satisfacción de los clientes.

Adicionalmente, se logró entregar una estructura básica para el uso de diferentes herramientas y metodologías que mejoraron procesos de creación, mantenimiento, rendimiento, eficiencia, gestión de cambios y monitoreo. Además de obtener una mejor respuesta de emergencia para servicios contenerizados y orquestados por kubernetes.

Palabras clave — **Kubernetes, Devops, principios, eficiencia, optimización.**

ABSTRACT

The technology industry, more specifically in companies that work in IT (information technology), want their processes to be optimal at any stage of development, thus achieving the greatest possible benefit. For this reason, they implement Devops methodologies (methodologies that seek to have higher quality software) in their projects, since this makes it easier for them to have higher quality products using their standards, in addition to using tools that help in these processes and provide useful information, both for good decision-making and for improvements in application maintenance.

In the development of the practice, excellent results were obtained, making use of new methodologies, tools and processes. Demonstrating that by implementing SRE principles (for its acronym in English Site Reliability Engineering or in Spanish Site Reliability Engineering) in an organization or company, different aspects of quality in applications are improved, providing improvements in the development process and in customer satisfaction.

Additionally, it was possible to deliver a basic structure for the use of different tools and methodologies that improved processes of creation, maintenance, performance, efficiency, change management and monitoring. In addition to obtaining better emergency response for containerized services and orchestrated by kubernetes

Keywords — **Kubernetes, Devops, fundamentals, performance, optimize.**

I. INTRODUCCIÓN

La era de la tecnología se encuentra en su mayor auge en los últimos años, la mayoría de nuestras actividades las podemos realizar a través de un sitio web, aplicación móvil o aplicativo integrado. Lo cual hace que debamos darles un seguimiento a estos servicios, garantizando: buena disponibilidad del sistema, latencia, rendimiento, eficiencia, gestión de cambios, monitoreo, respuesta de emergencia y planificación de la capacidad.

Google una de las empresas más grandes de tecnologías de la información en el mundo, y es quien propuso el término SRE (Site Reliability Engineering) el cual se enfoca en los aspectos anteriormente mencionados, siendo un área específica de Devops, vigilando tanto la infraestructura como la rentabilidad de algún servicio que se proporcione.

Para apaciguar estas problemáticas que se presentan en el desarrollo de software, Guane Enterprises una empresa dedicada a proporcionar soluciones de TI, comenzó a realizar prácticas relacionadas al SRE, sin embargo, se encuentra en una etapa muy temprana y se desea crear una estructuración mucho más robusta que pueda mantenerse en el tiempo y lograr un cambio en todos los proyectos que se tengan.

Finalmente, se espera tener definidos unos principios que ayuden a temas de monitoreo, automatización de procesos y manejo de permisos en la organización, logrando así que los proyectos actuales y venideros sean mucho mejores.

II. PLANTEAMIENTO DEL PROBLEMA

Cada vez que inicia un proyecto en una empresa de TI, siempre existe la incógnita:
¿Cuáles son los principios que se deberán seguir para entregar un producto de calidad?

La mayoría de las empresas pequeñas y medianas se enfocan en sacar más rápido un producto y dejan a un lado estándares de calidad, como lo son la eficiencia o disponibilidad de un aplicativo, trayendo problemas a futuro que deberán resolverse seguramente con mucho más trabajo y dinero, lo que conllevará a un ahorro si se logra hacer de buena manera. Muchos de estos inconvenientes no sucederían, si desde una etapa temprana del desarrollo hubiéramos pensado en ellas y no simplemente en hacerlo funcionar; teniendo pautas ya estimuladas se pueden solucionar estas falencias de forma muy sencilla y no despreciar tiempo o dinero restructurando algo que nos tome por sorpresa.

Muchos productos que se encuentran en el mercado ya sean aplicativos web, móviles, plataformas bancarias, entre otras, tienen caídas o desconexiones del servicio por algún tiempo. Afectando su funcionamiento con normalidad y son causadas por diferentes razones, una de las más comunes son las actualizaciones de versiones que llevan a la no disponibilidad del aplicativo por unas horas, en lo general en horas de madrugada, este es buen ejemplo para nuestro problema, cómo logramos que estos lapsos de tiempo no sean largos o sean prácticamente imprescindibles para el usuario, obteniendo un proceso tan eficiente que proporcione una mejor experiencia para el usuario y un producto de más calidad para la empresa.

III. OBJETIVOS

A. Objetivo general

Liderar la estructuración de principios SRE en los proyectos para los clientes de Guane Enterprises.

B. Objetivos específicos

- Realizar una exploración sobre cómo las grandes empresas realizan tareas de SRE dentro de sus organizaciones.
- Aprendizaje de herramientas que ayuden a monitorear servicios de Desarrollo de Software, su rendimiento, disponibilidad, recursos destinados, manejo y configuración.
- Desarrollo de diferentes herramientas de automatización, como lo son los pipelines para despliegues continuos ayudando en la gestión de cambios de los diferentes ambientes.
- Creación de un sistema RBAC (Role Based Access Control) para el manejo de roles dentro de cada proyecto.
- Creación de sistemas de notificación para el monitoreo, fallo y éxito de procesos en el desarrollo de un proyecto.
- Implementación de herramientas que ayuden a reducir los tiempos de despliegue a producción y entrega de productos.

IV. MARCO TEÓRICO

El enfoque esencial en este proyecto de prácticas se da sobre el SRE. En primera instancia se dio una contextualización sobre los principales términos que se deben comprender para tener claridad sobre el proceso que se realizará en la empresa.

SRE es una disciplina que incorpora diversos aspectos al momento de desarrollar software, se aplica a problemas y tareas de TI específicamente. Adicionalmente, es un área específica de Devops siendo está "un conjunto de prácticas destinadas a reducir el tiempo entre realizar un cambio en un sistema y el tiempo que tarda en entrar a producción, al mismo momento garantiza una alta calidad" [1] lo que se desea es tener tanto un software de alta calidad, como buenos procesos para llegar al software final.

Se realizaron una serie de servicios independientes en prácticamente cualquier proyecto, para ello, se deben estandarizar los ambientes como parte de la configuración del entorno de desarrollo en los diferentes proyectos, se utilizarán Docker Containers, los cuales son una tecnología basada en contenedores que encapsula la aplicación volviéndola independiente de las demás aplicaciones [2]. Por otro lado, cuando el número de servicios crece en una aplicación, también crece el número de contenedores, volviendo más compleja la tarea de administrar todos los servicios y la comunicación entre sí, por esta razón, se necesitan ejecutar tareas de organización que logren mantener todos los servicios de una manera automática. Kubernetes es una herramienta que permite orquestar todos estos contenedores sin interferir con ninguna implementación a nivel de aplicación, es decir, facilita la gestión y administración del despliegue de los contenedores a nivel de infraestructura y hardware, además, "Kubernetes proporciona una recuperación al reiniciar automáticamente los contenedores fallidos y reprogramarlos cuando se deshabilitan sus hosts. Esta capacidad mejora la disponibilidad de la aplicación." [3]

La seguridad es un aspecto muy importante, por lo tanto, en kubernetes "la práctica de aplicar reglas de autenticación y autorización para evitar usuarios malintencionados obtengan acceso y realicen actividades no autorizadas dentro del clúster de Kubernetes"[4] se implementará RBAC (control de acceso basado en roles) lo que se busca es tener un control de las acciones que se

realizan sobre el clúster y solo cierto tipo de usuarios puedan realizar acciones a las cuales se encuentren autorizados.

Para el manejo de métricas, control, visualización de componentes internos en kubernetes se tienen las siguientes herramientas, Lens, el IDE de Kubernetes, es fruto de un proyecto de código abierto patrocinado por Mirantis. Disponible para Linux, Mac y Windows, Lens le brinda una interfaz y un conjunto de herramientas potentes para administrar, visualizar e interactuar con múltiples clústeres de Kubernetes, además se explorarán más opciones para este mismo uso como lo son kubernetes dashboard y octant.

Es imposible administrar un servicio correctamente, mucho menos sin comprender qué comportamientos son realmente importantes para ese servicio y cómo medir y evaluar esos comportamientos. Con este fin, nos gustaría definir y brindar un determinado nivel de servicio [5] para ello se harán uso de los indicadores de nivel de servicio (SLI - Service Level Agreement), objetivos (SLO - Service Level Objectives) y acuerdos (SLA - Service Level Indicators).

Un SLI es un indicador de nivel de servicio, una medida cuantitativa cuidadosamente definida de algún aspecto del nivel de servicio que se proporciona. Un SLO es un objetivo de nivel de servicio (valor objetivo o rango de valores para un nivel de servicio que es medido por un SLI). Los SLA son acuerdos de nivel de servicio (contrato explícito o implícito con sus usuarios que incluye las consecuencias de cumplir o incumplir los SLO) [6]. Todos estos indicadores serán la base para evaluar nuestro software a través de los procesos y se definirán a través de la práctica.

V. METODOLOGÍA

Se definieron diferentes fases para comenzar a implementar principios de SRE, los cuales son disponibilidad, latencia, persistencia, durabilidad, rendimiento, eficiencia, gestión de cambios, monitoreo, respuesta de emergencia y planificación de la capacidad. La primera fase fue la contextualización con la herramienta de orquestación kubernetes, detallando su alcance, uso, manejo, implementaciones y demás actividades que lleven a un buen uso de la herramienta de manera básica. Dado que fue una nueva tecnología para el aprendizaje se destinará un tiempo prudente para realizar esta fase.

Continuando con su uso en diferentes proveedores de nube, siendo estas empresas que proveen servicios a través de internet dándonos como beneficio no tener que proveer de manera física nuestra infraestructura. Un clúster (Sistema donde se agrupan todos nuestros servicios) de kubernetes, se enfoca en ser distribuido en cualquier infraestructura, es decir, no importa el proveedor de nube que se utilice, un clúster de kubernetes siempre tendrá las mismas bases; dándonos una gran ventaja de aprendizaje y de reutilización.

La segunda fase se enfocó en la exploración de herramientas que nos ayuden con uno de los pilares fundamentales del SRE, que es el monitoreo de nuestra infraestructura, para ello se encontraron herramientas, que nos proporcionen el estado actual de todas las partes de nuestra aplicación. Este punto es de vital importancia, ya que será la principal herramienta que se utilizará en todo el proceso de prácticas para estar vigilando el estado de nuestros productos.
En esta misma fase se explorarán herramientas de notificación, creación de métricas de los servicios y reducción de tiempos, es decir, se buscará y probarán las herramientas que ayuden a los principios de eficiencia, gestión de cambios, monitoreo, para así lograr una mejora significativa al proceso de desarrollo de un proyecto.

La tercera fase es la implementación de RBAC en el clúster, para ello se definieron los roles que tendrán tanto para el equipo de desarrollo, producción, pruebas y cliente. Esta fue fase la más larga de todas, ya que se deben estandarizar los roles con sus respectivas acciones para los usuarios y distribuirlos a través de todos los integrantes de un proyecto. Además, debemos tener en cuenta

que el transporte de estas credenciales con las que se darán autorización se debe entregar a través de sistemas seguros y no se pierdan en el transporte de estas.

En la cuarta y última fase se tuvo como objetivo realizar las pipelines para los despliegues continuos, lo que se logro es tener un manejo automático de despliegue en los diferentes servicios que tenga una aplicación, a medida que se vaya desarrollando, acá se hará uso de herramientas como el cloud build de GCP o pipelines de Azure Devops; siendo estas herramientas que ayuden de manera sincronizada con sus demás servicios de nube, es decir, dado que ya se encuentran expuestos nuestros servicios para el ingresos de los clientes, es mucho más sencillo y rápido atacar esta fase con los mismos servicios que destinan los proveedores de nube y lograr así una actualización sin afectar la experiencia de usuario.

VI. RESULTADOS

En el desarrollo de la práctica se definieron varias herramientas a utilizar, muchas de ellas se buscaron y descartaron, debido a que no cumplían los requisitos mínimos para su implementación. Los cuales eran: versatilidad, fácil funcionamiento, precio, opciones de customización y soporte. Cada una de ellas se fue investigando e implementando según se decidía como la mejor herramienta para el problema.

Una problemática donde se aplicó los nuevos principios es la siguiente: se necesita crear una aplicación que procese cotizaciones para contratos logísticos, donde se hará una petición por correo con un texto parecido al siguiente: “Please quote 5 pallets of Tires 42x42x84 2600lbs From 33178 TO 89121” este texto se procesara y se extraerá la información relevante, que en este caso son el número de pallets necesarios, las dimensiones del paquete, el peso, desde donde sale y hacia donde se dirige. Estos correos pueden tener mucha más información acerca del paquete, como lo son su contenido y si se tiene alguna observación acerca de él. Además, hay que tener en cuenta que se deben leer todos los correos que lleguen a la bandeja de entrada de la dirección electrónica destinada, por esto también hay que descartar los correos que no sean una cotización y solo almacenar las potenciales cotizaciones.

Posteriormente se debe verificar que la información del correo se encuentre correcta después de la extracción de datos y se envía para el procedimiento de posibles contratos para la carga, donde se debe seguir un proceso de solicitud de contrato logístico, juntarlos todos y responder el mensaje electrónico con la cotización realizada, en un tiempo menor a 5 minutos.

Para solucionar esta problemática se harán uso de las siguientes herramientas a un nivel de infraestructura: GKE (Google kubernetes engine), LENS, cloud build y KEDA.

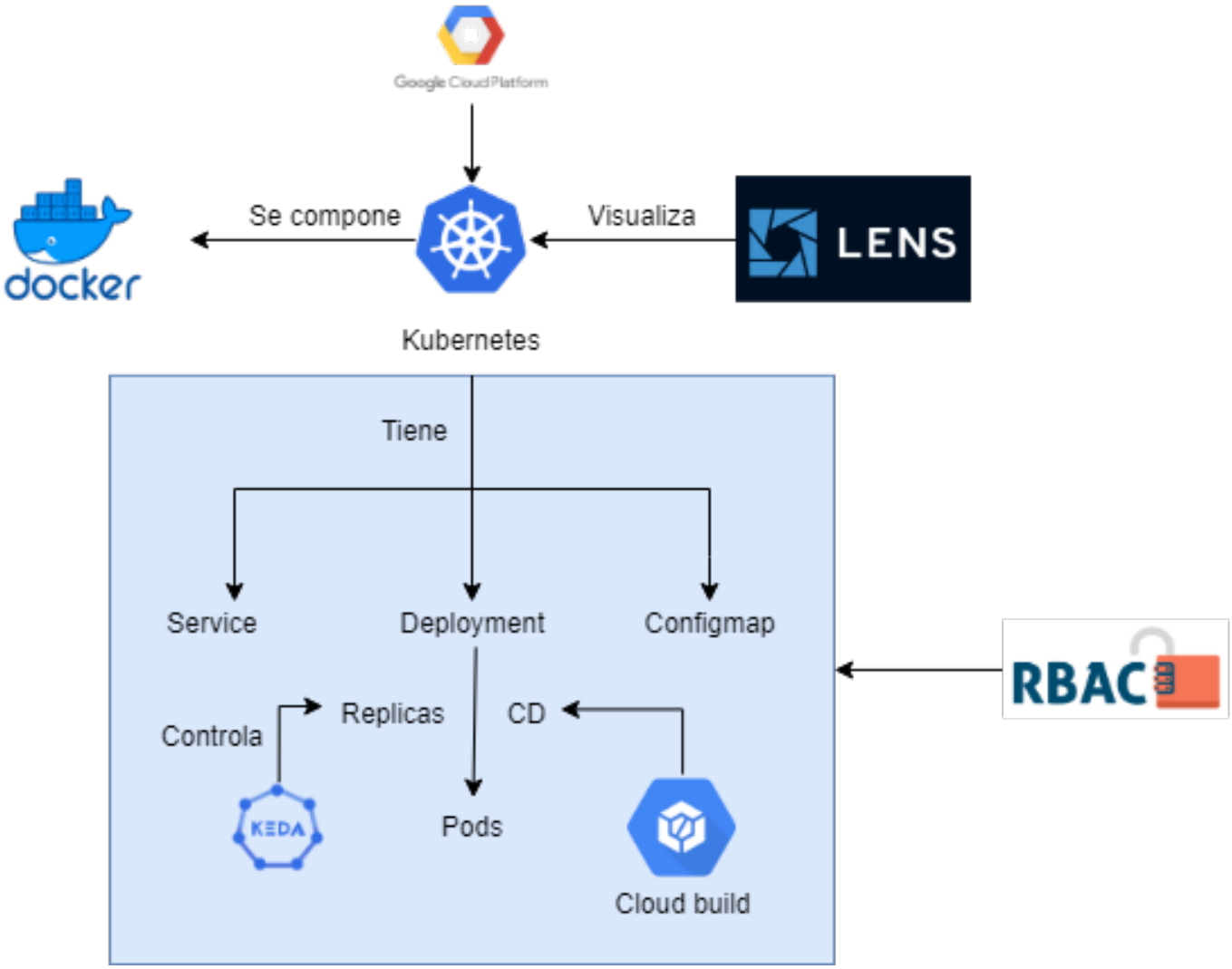

Figura. 1 Estructura de un proyecto.

En la figura 1 podemos apreciar el flujo básico para una aplicación. En primera instancia debemos crear un clúster de kubernetes en la nube de gcp con su servicio de GKE (Google kubernetes engine), luego conectarnos a él y administrarlo a través de LENS, hacer nuestro desarrollo de aplicaciones, generar las imágenes de Docker y manifiestos para su despliegue en el clúster, específicamente serian el service, configmap y deployment. Posteriormente hacer uso de Keda para el auto escalamiento y cloud build para un despliegue continuo; además de hacer uso del RBAC para manejar la autenticación y autorización en el clúster.

Definición de los componentes principales para el acondicionamiento de la aplicación:

Google cloud platform (GCP): Es el proveedor de nube en la mayoría de los proyectos donde se implementaron estás prácticas, es el proveedor de nube de Google, denominado Google cloud platform.

Google kubernetes engine (GKE): Es el servicio que provee Google para tener un clúster en la nube, en el cual se encuentra desplegado una arquitectura de microservicios para los proyectos.

Docker: Es el servicio de contenerización que se usa para montar los servicios en el clúster.

LENS: Es la herramienta que se usa para visualizar y administrar los diferentes clústers, tanto de grafica como de monitoria de creación y eliminación de diferentes servicios.

Deployments: Archivo de configuración para el despliegue del servicio donde se le administra la imagen de Docker a desplegar, el comando de arranque, los recursos a utilizar y las réplicas que tendrá dicho servicio.

Service: Archivo de configuración que expone nuestro servicio al clúster y lo vinculamos con el archivo de despliegue.

Configmap: Archivo de configuración para las variables de ambientes de un servicio en particular.

Cloud build: Es una herramienta de la plataforma de GCP para hacer despliegues continuos de diferentes servicios, en nuestro caso estaremos escuchando un push de una rama en git, cuando se ejecute se realizarán una serie de pasos en donde se descargarán los cambios de la rama, se comenzará a crear una imagen nueva de Docker, se subiera al container registry y se mandara la orden al workload para que la actualice.

KEDA: Es una herramienta de auto escalamiento, la cual nos provee grandes ventajas al poder aumentar las réplicas de un servicio dependiendo de diferentes variables. las dos más importantes son el tamaño de una cola para una tarea a realizar y el uso de crontab para crear replicas en el horario especifico.

RBCA: La creación de roles para obtener acceso con archivos de configuración de Rol, ClusterRole, ClusteRoleBinding y RoleBinding proporcionados por kubernetes, básicamente manejan los permisos que tenga una persona sobre los servicios en el clúster, estos se pueden crear de forma general para todo un namespace (un mecanismo para aislar grupos de recursos dentro de un único clúster) o servicios en particular.

POD: Unidad de encapsulamiento de una o más aplicaciones. Los pods son efímeros por naturaleza, si un pod (o el nodo en el que se ejecuta) falla, Kubernetes puede crear automáticamente una nueva réplica de ese pod para continuar con las operaciones.

Está estructura se encuentra de forma muy general, sin embargo, el uso de las herramientas se necesita para cada servicio del clúster, es decir, por cada implementación de KEDA que es el servicio de auto escalamiento, se debe realizar uno por cada servicio que tuviera una tarea a encolar en el rabbitmq (sistema de mensajería entre colas), este es un ejemplo general:

```yaml
apiVersion: keda.sh/v1alpha1
kind: ScaledObject
metadata:
  name: worker-load-creation-deploy
  namespace: prod-assistant
spec:
  scaleTargetRef:
    name: worker-load-creation-deploy
  pollingInterval: 10
  cooldownPeriod: 100
  minReplicaCount: 8
  maxReplicaCount: 10
  advanced:
    restoreToOriginalReplicaCount: true
    horizontalPodAutoscalerConfig:
      behavior:
        scaleDown:
          stabilizationWindowSeconds: 60
          policies:
            - type: Percent
              value: 20
              periodSeconds: 1800
  triggers:
    - type: rabbitmq
      metadata:
        host: "string de conexion al rabitmq"
        queueName: load-creation
        mode: QueueLength
        value: "10"
    - type: cron
      metadata:
        timezone: America/Bogota
        start: 0 8 * * 1-5
        end: 0 16 * * 1-5
        desiredReplicas: "8"
```

Figura 2. Configuración básica de un servicio de auto escalamiento de KEDA

A continuación expandiremos los puntos más importantes en esta configuración de auto escalamiento de servicios, como se puede apreciar en la figura 2, tenemos un archivo de configuración que tiene múltiples parámetros, entre ellos los más importantes son estos: scaledTargetRef es el servicio que queremos auto escalar, el mínimo de replicas que queremos y el máximo de replicas que deseamos, en este caso son 8 y 10 respectivamente, por ultimo los triggers son las reglas por las cuales se disparan los auto escaladores, en este caso hacemos uso de

rabbitmq, teniendo en cuenta que si en la cola hay más de 10 tareas estaqueadas en load-creation comenzaremos a crear nuevas replicas para dar abasto al flujo. Del mismo modo el trigger de cron, será un disparador en formato crontab, básicamente es un formato por el cual podemos definir lapsos de tiempo, en este caso el disparador estará activo entre las 8 de la mañana y las cuatro de la tarde de lunes a viernes, que es el horario con más flujo en la aplicación. Con esto ya tenemos una configuración básica para escalar nuestros servicios, posteriormente se implementó la configuración en los servicios que se consideraban pertinentes

Otra herramienta utilizada, fue el cloud build que es utilizado para el despliegue continuo de los diferentes servicios del clúster, esta herramienta también se encuentra escrita en archivos yaml, los cuales se encuentran estructurados por pasos, lo primero es la creación de la imagen de Docker del servicio en específico:

```yaml
steps:
  - name: 'gcr.io/kaniko-project/executor:latest'
    args:
      - '--destination=$_IMAGE_NAME:$COMMIT_SHA'
      - '--cache=true'
      - '--cache-ttl=10h'
      - '--dockerfile=$_DOCKERFILE_NAME'
      - '--build-arg=VUE_APP_BACK_ROUTE= "Ruta de nuestro backend"'
    id: Image create with kaniko
  - name: gcr.io/cloud-builders/gke-deploy
    args:
      - prepare
      - '--filename=$_K8S_YAML_PATH'
      - '--image=$_IMAGE_NAME:$COMMIT_SHA'
      - '--app=$_K8S_APP_NAME'
      - '--version=$COMMIT_SHA'
      - '--namespace=$_K8S_NAMESPACE'
      - '--label=$_K8S_LABELS'
      - '--annotation=$_K8S_ANNOTATIONS,gcb-build-id=$BUILD_ID'
      - '--create-application-cr'
      - >-
        --links="Build
        details=https://console.cloud.google.com/cloud-build/builds/$BUILD_ID?project=$PROJECT_ID"
      - '--output=output'
```

Figura 3. Creación de la imagen

En este caso hacemos uso de kaniko, es una herramienta para crear imágenes de contenedores a partir de un Dockerfile, dentro de un contenedor o clúster de Kubernetes.

Kaniko no depende de un demonio Docker y ejecuta cada comando dentro de un Dockerfile completamente en el espacio del usuario. Esto permite crear imágenes de contenedores en entornos que no se pueden ejecutar de manera fácil o segura un demonio de Docker, como un clúster estándar de Kubernetes.

Además de crear la imagen, se hace uso de la cache, para que las imágenes posteriores a la creación se creen de manera más rápida y reducir el tiempo de ejecución del pipeline, logrando así reducir costos.

La figura 3 la podemos asimilar como un simple archivo de configuración escrito en yaml, haciendo la función de un pipeline; siendo está un proceso que se ejecuta con los pasos definidos en el archivo, cuando hay un cambio de una rama en específico del git, se toma el sha del commit para colocárselo como tag en el nombre de la imagen y así darle un nombre completo a la imagen creada. Por otro lado, activamos el almacenamiento de cache por 10h, esto nos proporciona que las imágenes se creen de manera más rápida, haciendo uso de las capas cache guardadas, para finalizar agregamos la ruta del Dockerfile y le añadimos como argumento la ruta del backend que desea utilizar, en este paso pueden ser rutas tanto de un entorno de desarrollo como uno de producción según se deseé.

Ya creada la imagen y asignadas todas las variables necesarias, subimos la imagen en el registry donde se almacenarán y se podrán descargar posteriormente tanto por un desarrollador o el clúster para implementar el nuevo servicio con los nuevos cambios que se hayan realizado.

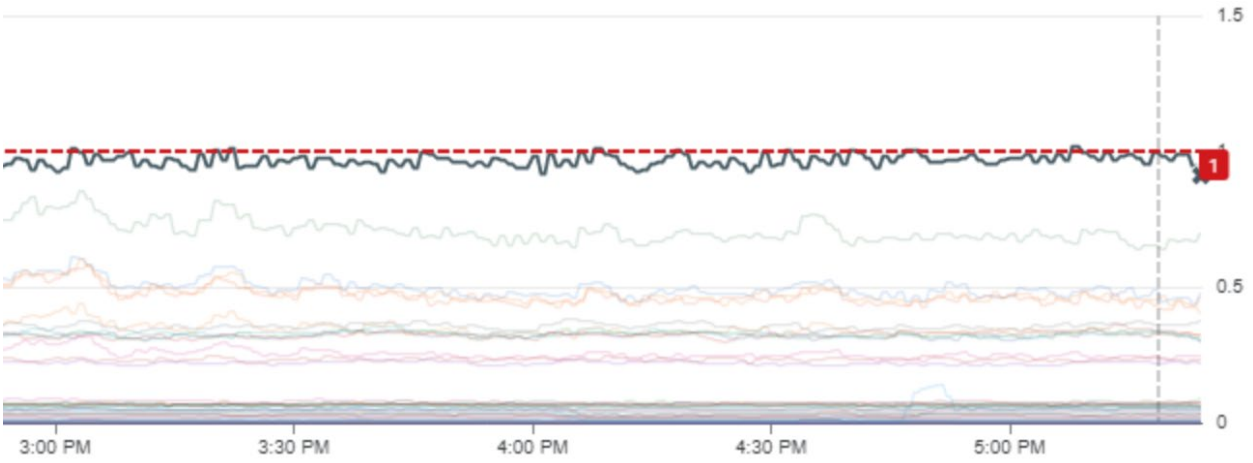

Figura 4. Comportamiento de la CPU.

Fuera de la estructura base que se utilizó, se hace uso el cloud monitoring, una herramienta para ver métricas de los diferentes servicios en el clúster, como lo son el procesador y la memoria RAM que utiliza un servicio en particular.

En está grafica podemos apreciar diferentes servicios en su uso de CPU, más específicamente el servicio que se mantiene entre 900 milicpu y 1 core siendo está la línea gris y una línea recta de color rojo no continua; delimitando el máximo que puede llegar un servicio, está línea la utilizaremos posteriormente. Al momento que esta línea es sobrepasada, se lanzará una notificación en nuestro slack (aplicativo de comunicación) donde nos dirá el nombre del servicio, el tiempo de

la alerta, el porcentaje que se usó, el nivel de la alerta y un enlace para acceder directamente al servicio.

En la siguiente figura podemos apreciar la forma de una alerta y la información especifica que nos puede proporcionar:

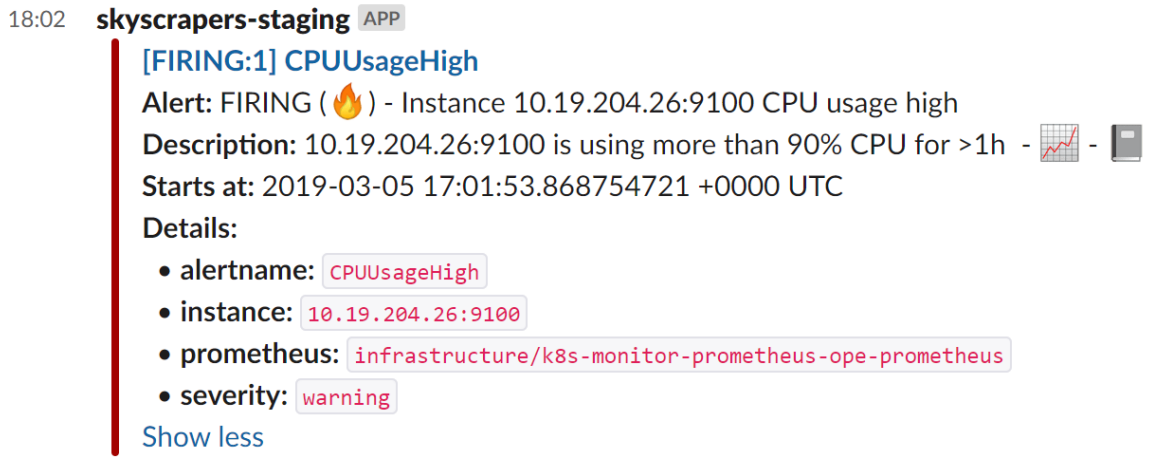

Figura 5. Alerta de CPU

Todo el proceso que se realizó en esta práctica logro disminuir el tiempo de respuesta de diferentes servicios, obtener la meta de enviar una cotización en menos de 5 minutos, además poder procesar un flujo aun mayor de cotizaciones. También debemos de tener en cuenta que varios procesos para los desarrolladores se hicieron más sencillos, ya que las implementaciones de los servicios se hacían de forma automática, igualmente se da una propuesta de alertas que en un futuro será implementada a gran escala para así tener un control más rápido y exacto de los servicios que fallen.

VII. CONCLUSIONES

- La comunicación en un equipo es primordial para un desarrollo adecuado. En ese sentido, marcos de trabajo como tareas y objetivos bien definidos permiten una comunicación rápida y eficaz.
- Hacer uso de herramientas de terceros como lo es KEDA aumenta la automatización en gran medida, ya que no es necesario estar pendiente del auto escalado de los servicios, si no de una buena configuración inicial.
- El uso de pipelines de despliegue continuo como lo es el cloud build, baja la probabilidad de errores humanos haciendo el proceso de manera automática y más rápida.
- El manejo de roles es muy importante, ya que la mayoría de los proyectos se encuentran sin esta capa de seguridad y varios de los servicios se pueden dañar por pequeños cambios en ellos.
- Las métricas con alerta son un gran avance, ya que en la mayoría de los casos no se puede estar pendiente de todos los servicios y ver que ninguno de ellos falle, es una gran estrategia que lleguen alertas de manera automática para avisarnos de un fallo en el sistema por temas de recursos.

REFERENCIAS

[1] J. L. Arango, “Enfermedades respiratorias del recién nacido,” en *Fundamentos de pediatría: generalidades y neonatología*, J. A. Correa, J. F. Gómez, y R. Posada, Eds. Fondo Editorial CIB, 2000, pp. 463–467.

[1] L. Bass, I. Weber, and Z. Liming, DevOps: A Software Architect’s Perspective, Addison-Wesley, 2015

[2] T. Combe, A. Martin, and R. Di Pietro, “To Docker or Not to Docker: A Security Perspective,” IEEE Cloud Comput., vol. 3, no. 5, pp. 54–62, 2016, doi: 10.1109/MCC.2016.100.

[3] L. Abdollahi Vayghan, M. A. Saied, M. Toeroe, and F. Khendek, “Deploying Microservice Based Applications with Kubernetes: Experiments and Lessons Learned,” IEEE Int. Conf. Cloud Comput. CLOUD, vol. 2018-July, pp. 970–973, Sep. 2018, doi: 10.1109/CLOUD.2018.00148.

[4] Kubernetes, “Production-grade container orchestration.” [Online]. Available: https://kubernetes.io/docs/

[5] Jones, C., Wilkes, J., & Murphy, N. (2016). Site Reliability Engineering (p. Chapter 4). O'Reilly Media, Incorporated: Betsy Beyer. O'Reilly Media, Incorporated: Betsy Beyer.

[6] Jones, C., Wilkes, J., & Murphy, N. (2016). Site Reliability Engineering (p. Chapter 4, Service Level Terminology). O'Reilly Media, Incorporated: Betsy Beyer. O'Reilly Media, Incorporated: Betsy Beyer.