UNIVERSIDAD DE ANTIOQUIA

Aplicación de una arquitectura basada en eventos para una plataforma cognitiva utilizando Kubernetes y Apache Kafka

Autor:
Mateo Llano Avendaño

Universidad de Antioquia
Facultad de ingeniería
Ingeniería de sistemas
Medellín, Colombia
2021

Aplicación de una arquitectura basada en eventos para una plataforma cognitiva utilizando Kubernetes y Apache Kafka

Mateo Llano Avendaño

Informe de práctica presentado como requisito parcial para optar al título de:
Ingeniero de sistemas

Asesores:
Leonardo Augusto Pachón, Ph.D. Physicist.
Martín Elias Quintero, System Engineer

Universidad de Antioquia
Facultad de ingeniería
Ingeniería de sistemas
Medellín, Colombia
2021

Resumen

Las arquitecturas basadas en eventos presentan, hoy en día, un impacto significativo en el desarrollo de software, ya que potencian el manejo de procesos ETL. En este documento se aborda la implementación de un proceso ETL a través de una plataforma cognitiva utilizando Apache Kafka por medio de la cual se aplican los mecanismos más estables y modernos que proporcionan al producto altos estándares respecto a las buenas prácticas del desarrollo tales como el uso de contenedores, guías de estilo de codificación, diseño arquitectónico limpio y metodologías ágiles.
Adicionalmente, se abordan temas relacionados a la nube, las cuales resaltan los servicios ofrecidos por los proveedores, y la implementación de servicios a producción a través de sistemas para la automatización de despliegues como Kubernetes, donde se explora la implementación de manifiestos y la creación de objetos a través de estos.
Como resultado final se desarrolló un servicio web implementado bajo el framework FastAPI el cual inicia un servicio productor encargado de tomar los correos que llegan a un dominio, los procesa y envía a un tópico, y de un consumidor que se encarga de tomar dicha información y enviarla a un algoritmo de machine learning encargado de identificar entidades para luego ser mostradas a un usuario final. Dicho aplicativo fue desplegado en kubernetes mediante los servicios en la nube ofrecidos por GCloud.

Introducción

El análisis e interpretación de datos se hace cada vez más relevante en la industria porque permiten generar estrategias de gestión y además son fundamentales para la construcción de inteligencia empresarial y por tanto, para la toma de decisiones. Es por esta razón que los procesos ETL (Extraction, Transformation and Load) han tomado importancia en este tema, pues son capaces de recopilar, refinar y almacenar los datos[1].

Por otra parte, el uso de contenedores se ha vuelto cada vez más esencial en el desarrollo de software, convirtiéndose en una tecnología muy popular. Los contenedores se definen como una unidad ejecutable de software que contiene el código de la aplicación, junto con sus bibliotecas y dependencias[2]. Esta unidad se logra ejecutar gracias a una forma de virtualización del sistema operativo, donde sus características ofrecen entornos aislados para ejecutar los servicios de software[4]. Gartner Inc., compañía de investigación y asesora líder en el mundo, estima que para el 2024 el 75

Para la realización de la práctica se plantea el desarrollo de una arquitectura basada en eventos que permita soportar los procesos ETL de una plataforma cognitiva. Se estudiarán los mecanismos más estables y modernos que le proporcione al producto altos estándares respecto a las buenas prácticas del desarrollo tales como el uso de contenedores, guías de estilo de codificación, diseño arquitectónico limpio y metodologías ágiles.

Objetivos

Objetivo Principal: Implementar una arquitectura basada en eventos utilizando Apache Kafka para soportar los procesos de ETL de una plataforma cognitiva.

- **OE1:** Construir el diseño de los diferentes procesos de extracción, carga y transformación de datos mediante una estrategia de paso de mensajes.
- **OE2:** Desarrollar servicios de software para soportar los procesos cognitivos de la plataforma en alguna modalidad basada en eventos como Streaming o PUB/SUB.
- **OE3:** Ofuscar el diseño e implementación final con el objetivo de plantear métricas en términos de rendimiento que permitan establecer la viabilidad de los sistemas basados en eventos frente a la versión requests-response de la plataforma.

Marco teórico

Las ciencias computacionales y el desarrollo de software son un campo relativamente nuevo respecto a los descubrimientos y conocimientos recopilados de otras áreas. Sin embargo, es realmente difícil entender los alcances científicos actuales sin el acompañamiento y soporte tecnológico que ha avanzado a pasos gigantescos en las últimas décadas. El hito que más ha cambiado en la historia de esta disciplina es el análisis, ciclo de vida y diseño de software, siendo este último el tema de interés más activo a nivel industrial. El mundo del desarrollo software se caracteriza por los incansables, y a veces drásticos, cambios en su ecosistema a lo largo del tiempo. La aparición de nuevas tecnologías, estrategias, paradigmas y lenguajes permite que cada vez sean mejores los procesos de automatización y experiencia de usuario a la par de la administración de la infraestructura que esto conlleva. Una de estas transformaciones es evidente en la manera como se desarrolla una aplicación.

La Imagen 1 muestra una evolución temporal de las arquitecturas más ampliamente usadas al construir artefactos de software. En primer lugar se encuentra tal vez la más tradicional conocida como monolito, que es, una aplicación que contiene toda su complejidad en un solo artefacto o unidad de software. Es decir, las estructuras lógicas que usa una aplicación como las reglas de negocio, persistencia de datos y capas de presentación se encuentran acopladas en un mismos sitio. Este diseño seguramente es el acercamiento más clásico y aún vigente al momento de plantear diseños arquitectónicos. La razón principal por la que se considera obsoleto es por las distintas dificultades que representan las etapas de implementación como el alto acoplamiento entre componentes y las falencias en la escalabilidad y mantenimiento que depende de lo robusto que pueda a llegar a ser la solución planteada; más adelante, aparece la separación de las capas de presentación y lógica del negocio donde persiste un alto acoplamiento con la base de datos.

Las tres capas antes mencionadas son unidades de software bien definidas que pueden ser desacopladas al momento de desarrollar un producto. Es por ello, que una implementación donde se realiza la separación entre lo que se presenta al cliente y los procesos de cómputo que realiza el servidor es una propuesta atractiva cuando se quiere implementar una solución. En otras palabras, se define muy bien qué componentes pertenecen a la interacción de usuario final con la plataforma y la responsabilidad de estas se delegan a otro artefacto que se encarga de transformar la información y persistir en algún modelo de datos, dando como resultado a una arquitectura cliente/servidor.

Finalmente, llega una gran transición hacia los microservicios, los cuales se encargan de desacoplar las capas de un artefacto de software estableciendo el acoplamiento a un nivel mínimo y permitiendo la escalabilidad, mantenibilidad y despliegue independiente. Los microservicios representan los artefactos más atómicos que pueda tener una aplicación que en conjunto representan y componen los requerimientos del sistema.

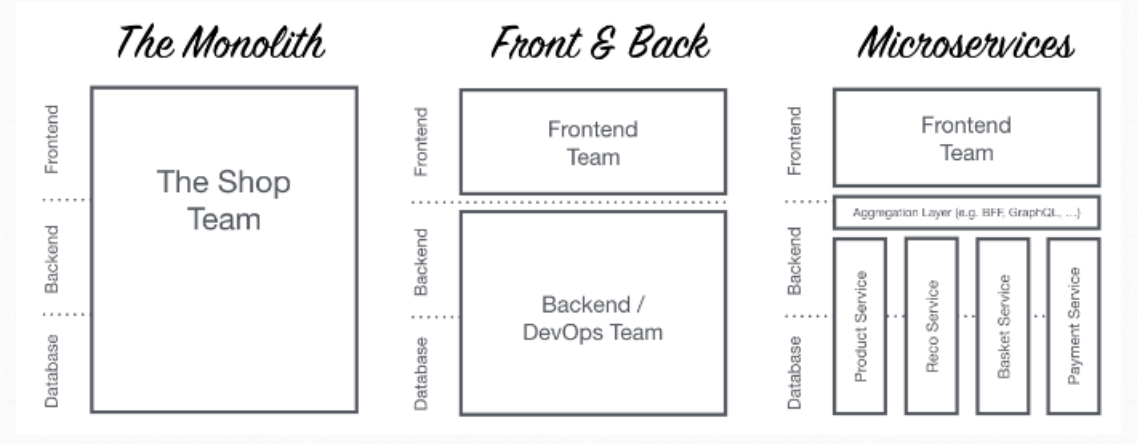

Figure 1: Evolución temporal de las arquitecturas[8].

La arquitectura de microservicios se define como un estilo que estructura una aplicación como un conjunto de servicios implementados de manera independiente[3]. Si bien la filosofía base de esta metodología soluciona un amplio espectro de problemas, a la hora de diseñar una aplicación surge la necesidad de los microservicios de manera independiente y que permita que cada uno de ellos administre sus propios recursos de forma aislada. Por tanto, se evidencia la necesidad del uso de tecnologías como los contenedores que permitan aislar cada micro artefacto de software.

El uso de contenedores ofrece muchas ventajas especialmente su portabilidad e independencia que posibilitan la utilización y el desarrollo de arquitecturas modernas. Sin embargo, cuando se utilizan gran cantidad de ellos, se hace necesario contar con un sistema de gestión y administración de los procesos ejecutados por ellos mismos, es allí donde sistemas como Kubernetes, una plataforma de orquestación de código abierto, cumple un papel muy importante. Kubernetes ofrece un entorno de administración centrado en contenedores, de manera que orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo no dependan directamente del soporte humano de los usuarios, esto quiere decir que Kubernetes proporciona un ecosistema de componentes y herramientas que facilitan el proceso de desplegar, mantener, escalar y administrar aplicaciones[5].

Además de estos términos, a nivel de arquitectura, se debe tener en cuenta que como propósito del desarrollo se desea tener una arquitectura flexible que pueda adaptarse a los cambios y permita tomar decisiones en tiempo real. Además, se pretende tener desacoplamiento en tiempos de ejecución. Estas necesidades expuestas dan paso a la implementación de las arquitecturas basadas en eventos (event-driven architecture) que ofrecen un enfoque de programación orientado en la captura, la comunicación, procesamiento y persistencia de unidades llamadas eventos[6]. Estas arquitecturas desacoplan los servicios a un nivel más alto garantizando que si un servicio no es tolerante a fallas, el resto siga funcionando. Adicionalmente, ofrecen un enrutador de eventos que actúa como un sistema nervioso que audita y define políticas como restaurar el sistema a una versión estable en caso de ser necesario[7]. Esta característica puede verse en detalle en la Imagen 2, donde se aprecia cómo los eventos producidos en un sistema llegan a un enrutador de eventos que se encarga de distribuir la información en los canales apropiados según los filtros y reglas designadas. Dentro de esta arquitectura, se utilizará el modelo de eventos que se compone por un agente productor y un consumidor, el primero actúa como un generador que inscribe mensajes en un tópico determinado donde llega al segundo y este toma las acciones respectivas para atender y procesar dicha información usando distintos controladores.

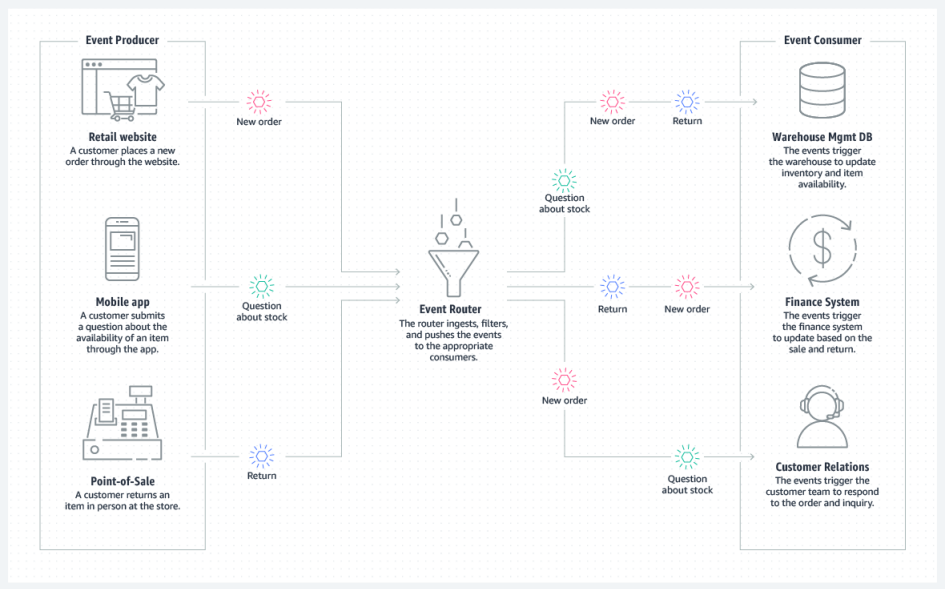

Figure 2: Arquitectura basada en eventos [9]

Metodología

La Imagen 3 representa un diagrama de flujo mostrando las etapas incluidas en el proyecto, cada una, como un hito dentro del marco de las prácticas académicas.

Fase 1: Búsqueda de la literatura.

Se utilizaron diversas fuentes de información para obtener un panorama general de las tecnologías planteadas dentro de la implementación del proyecto. Principalmente, se inicia una búsqueda rigurosa del funcionamiento de los contenedores linux como tecnologías de aislamiento lógico. Posteriormente, basados en los conceptos fundamentales se definió una ruta de aprendizaje que iniciaba desde los contenedores LXC [10] hasta llegar a Docker. Docker permite un modo Cluster que sirve de puente para entender el funcionamiento de los orquestadores de contenedores como Apache Mesos y Kubernetes, llamado Docker Swarm. Finalmente, se hace un estudio a profundidad de infraestructura cloud entendiendo Kubernetes paso a paso y estableciendo una línea de ideas hasta llegar a recursos personalizados que son una forma más clara de utilizar componentes reutilizables y externos como lo es Kafka. Paralelamente al proceso anterior se realizó una ruta para comprender las arquitecturas de software basadas en eventos utilizando Apache Kafka.

Fase 2: Aplicación de las técnicas de Kubernetes enfocadas en Kafka .

Luego del análisis de la literatura propuesta se desarrolló una prueba de concepto enfocada en introducir y adquirir conocimientos relacionados a kubernetes, para esto, se habilitó un clúster en Google Cloud[11] donde se debe realizar la configuración, autenticación, despliegue y monitoreo de este. Se exploraron conceptos tales como la estructura de un archivo que describe el comportamiento imperativo de los componentes de Kubernetes, la implementación de unidades mínimas dentro del orquestador llamadas pods, uso de servicios para establecer comunicaciones entre componentes internos y externos, descripción de despliegues quienes controlan las réplicas de un artefacto y su

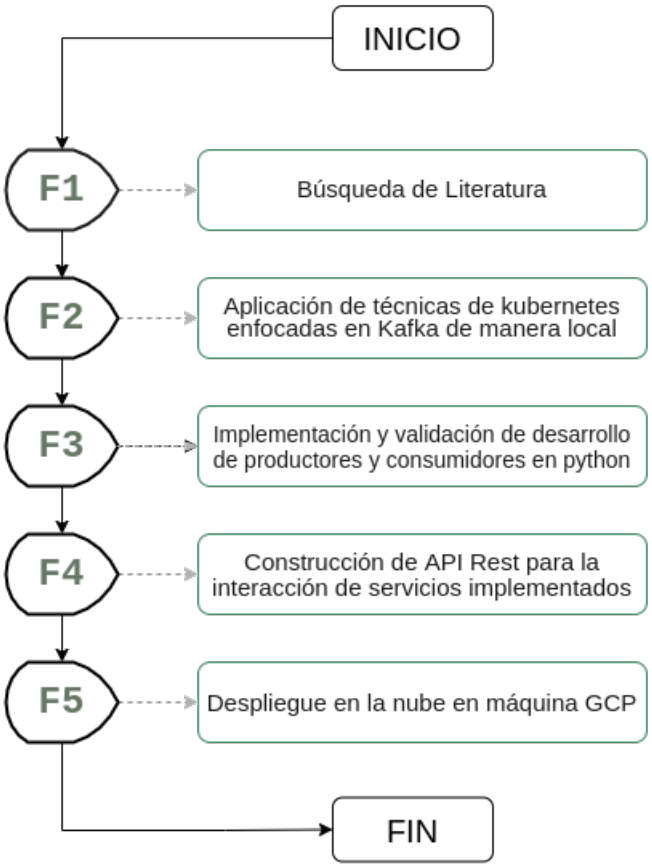

Figure 3: Diagrama de flujo metodología implementada.

comportamiento en general, credenciales y secretos como capa adicional de seguridad y volúmenes para el manejo de la persistencia física de las aplicaciones. Adicional, se usó un laboratorio de manera local a través de herramientas que simulan un cluster mediante nodos de docker tales como Kind[12].

Existen multitud de estrategias para instalar una aplicación dentro de Kubernetes, sin embargo una de las más referenciadas desde las fuentes oficiales es Helm, el cual funciona como gestor de paquetes basado en configuraciones que denomina como Helm Charts. Estas descripciones utilizan manifiestos nativos de Kubernetes para establecer relaciones entre componentes y lograr despliegues limpios basados en releases. Desde el punto de vista de productividad, utilizar Helm proporcionó un incremento sustancial en la efectividad del proyecto despreocupado al equipo de instalaciones de software como Kafka o bases de datos que de manera manual suelen ser complicadas de realizar.

Para introducir Kafka al orquestador Kubernetes, se utilizó el operador Strimzi[14] haciendo uso del gestor de Helm. Este operador como lo denota la Imagen 4 proporciona imágenes de contenedores y operadores para ejecutar Kafka en Kubernetes, además de simplificar el proceso, permite implementar, ejecutar y configurar clústeres y componentes de kafka proporcionando la administración de tópicos y usuarios[13]. Durante esta implementación se tuvieron en cuenta las siguientes configuraciones:

- **TLS Cifrado:** La implementación de este componente nos permite establecer una sesión

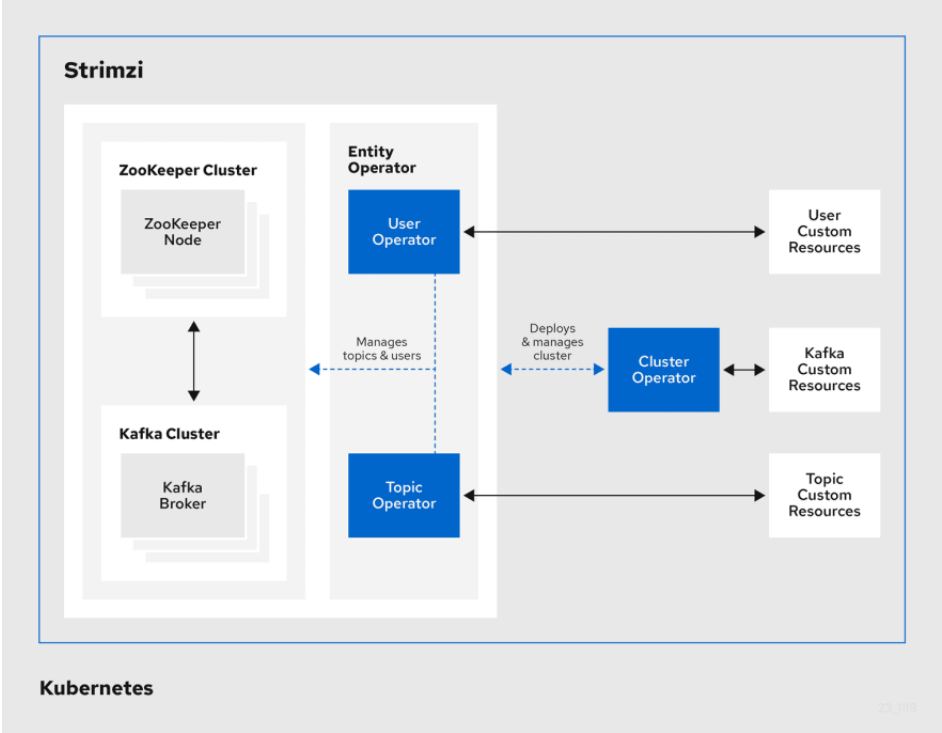

Figure 4: Arquitectura operador Strimzi [13]

cifrada entre el cliente y el servidor, realizando verificación de la identidad de cada conexión permitiendo que la información pueda transmitirse de manera segura a través de un canal.

- **Autenticación TLS:** Esta característica ofrece una capa de protección a la información que se maneja a nivel de tópico, permite que exista una autenticación bidireccional entre el servidor y el cliente basado en un identificador único de usuario.
- **Persistencia de la información:** Para garantizar la información contenida en los tópicos necesarios para el desarrollo, se exploraron los componentes Persistent Volume y Persistent Volume Claim quienes se encargan de usar estrategias de almacenamiento buscando la mayor tolerancia a fallos.

Fase 3: Implementación y validación de desarrollo de productores y consumidores en Python.

Una vez creada la infraestructura cloud necesaria en Kubernetes, se desarrollaron los módulos de la arquitectura PUB/SUB que producen la información que se envía a los tópicos y que la consume. Todo esto fue ejecutado haciendo uso del paquete kafka-python que ofrece una interfaz limpia y transparente para la conexión con Kafka.

Para este caso, se implementó un módulo productor, el cual con la ayuda de la librería o365[18] establece una conexión al servicio de Outlook y obtiene un objeto de tipo Email que contiene toda la información de cada correo electrónico recibido a una bandeja específica. Esta información es procesada y tabulada, es decir, pasa de ser data no estructurada a estructurada bajo un proceso cognitivo interno llamado Hephaestus encargado de organizar y establecer relaciones con la información, luego de dar formato a los datos son enviados al módulo consumidor publicando un

mensaje en el tópico correspondiente. La idea principal es tener un flujo constante de información que permite tener correos electrónicos casi que en tiempo real, por ellos, el proceso anteriormente descrito se hace con una frecuencia alta.

El segundo módulo implementado contiene el consumidor. Una vez el productor envía mensajes a un tópico, el consumidor se encarga de escucharlo y tomarlo para su posterior uso, a esto se le llama una suscripción. Al momento de tomar un mensaje encolado en un tópico se hace una petición a un servicio ofrecido por el equipo de ciencia de datos de Guane Enterprises encargado de usar algoritmos de Machine Learning y Deep Learning enfocados en el procesamiento de lenguaje natural. Este paso es fundamental y necesario debido a que nuevamente los datos son transformados.

Fase 4: Construcción de API Rest para la interacción de servicios implementados.

La comunicación entre servicios es de gran relevancia para el proceso que se intenta llevar a cabo. Dentro de la pila tecnológica empresarial se encuentra establecido FastAPI como Framework de la capa de aplicación para proporcionar interfaces web. Este marco de desarrollo está orientado a Python utilizando estrategias novedosas en el lenguaje como lo son las funciones asíncronas y tipado de variables, validaciones de tipos de datos como Pydantic y Typing, que son incorporadas en el ecosistema de FastAPI logrando un alto desempeño de RPM (response per minute). Todo esto garantiza el máximo aprovechamiento de los recursos del servidor físico.

Para lanzar un flujo completo de servicios web con Python, debemos apoyarnos de un servidor WSGI (Web Server Gateway Interface) que tome los requerimientos que ingresan a la máquina y las conduzca en una estructura legible hacia el framework web que se está usando.

Últimamente Python ha avanzado enormemente en su desarrollo interno; ahora es capaz de manejar hilos verdes(co-rutinas) usados en otros lenguajes para hacer uso de la concurrencia de procesos a nivel de usuario, por ello es necesario utilizar servidores ASGI (Asynchronous Server Gateway Interface). En el proyecto, para este propósito se utiliza como servidor WSGI: gunicorn utilizando trabajadores ASGI de Uvicorn.

Fase 5: Despliegue en la nube en máquina GCP.

Para llevar el proyecto a la fase de staging se utilizaron varios pasos que permitieron un despliegue exitoso y ordenado mediante las funcionalidades ofrecidas por los proveedores de la nube, particularmente los pasos más relevantes son:

- **Construcción de imágenes de aplicación usando docker:** Se construyeron las imágenes necesarias para el funcionamiento del proyecto, es decir la API REST construida en FastAPI. Estas imágenes, se almacenaron en el container registry ofrecido por GCloud (gcr) que funciona como un repositorio o almacén de imágenes.

- **Despliegue de Clúster:** Para este paso se realizó un despliegue de un clúster de producción el cual cuenta con características que garantizan la disponibilidad y el buen funcionamiento de los aplicativos expuestos. Este clúster cuenta con 3 tipos de máquinas, cada una composición de 5 nodos, una máquina es de tipo n1-standard-2 la cual se compone de 2 CPU virtuales, 8 GB de memoria ram y almacenamiento tipo ssd, con un ancho de banda de 10 Gbps, la segunda máquina es una e2-medium la cual contiene 2 CPU virtuales, 4 GB de memoria ram, disco mecánico para almacenamiento de información y un ancho de banda de hasta 2 Gbps. Finalmente la tercer máquina es un e2-standard-4 la cual se caracteriza por tener 4 CPU

virtuales, 16 GB de memoria ram, disco mecánico para almacenamiento de la información y un ancho de banda de hasta 8 Gbps.

- **Creación y despliegue de manifiestos:** Todos los archivos que describen la operación y aplicación de la plataforma se alojaron en github, para su construcción se crearon pods, servicios y archivos de configuración que permiten el correcto funcionamiento del proceso. Estos manifiestos se componen de una estructura esencial compuesta por:
  - **apiVersion:** Este campo contenido dentro de lo manifiestos de Kubernetes indica la versión de la API de kubernetes que se está usando para crear el objeto[15]
  - **Kind:** Indica la clase de objeto que se quiere crear, estos pueden ser objetos de tipo pod, service, deployment entre otras muchas que nos ofrece kubernetes, para el caso de kafka que utiliza el operador de Strimzi nos ofrece diferentes tipos de objetos personalizados como Kafka, kafkaUser y kafkaTopic, los cuales nos permiten desplegar la infraestructura de kafka necesaria en el clúster.
  - **Metadata:** Datos que se le asignan a un objeto los cuales permiten identificar unívocamente al objeto sobre los demás.
  - **Spec:** A través de este campo especificamos que exactamente queremos indicarle a Kubernetes que construya[16], su configuración es diferente según el tipo de objeto que se haya definido en el campo kind.

```yaml
apiVersion: kafka.strimzi.io/v1beta1
kind: Kafka
metadata:
  name: my-kafka-cluster
spec:
  kafka:
    version: 2.5.0
    replicas: 1
    listeners:
      plain: {}
      external:
        type: loadbalancer
        tls: true
    config:
      offsets.topic.replication.factor: 1
      transaction.state.log.replication.factor: 1
      transaction.state.log.min.isr: 1
      log.message.format.version: "2.4"
    storage:
      type: ephemeral
  zookeeper:
    replicas: 1
    storage:
      type: ephemeral
```

Figure 5: Estructura manifiesto en despliegue de objeto tipo Kafka

La Imagen 5 describe la aplicación de la estructura mencionada anteriormente para el despliegue de un objeto tipo kafka, este objeto se puede crear ya que el operador strimzi importado en el campo apiVersion lo permite. En el campo metadata se identifica el nombre

por el cual podrá ser diferenciado este objeto sobre los demás y en los specs observamos las configuraciones necesarias para el objeto a crear, así como las imágenes necesarias y el tipo de persistencia que se le designará a cada imagen.

Implementación de Traefik: A medida que se avanzó en el desarrollo se identificó la necesidad de tener un servidor ingress que funcione como proxy inverso para redirigir las peticiones que se hacen a los diferentes subdominios a los servicios correctos y de esa manera únicamente exponer un balanceador de carga en nuestro clúster.

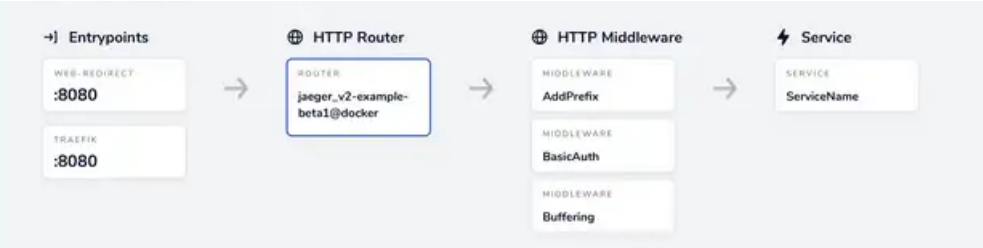

Figure 6: Traefik dashboard

Traefik ofrece una solución a este requerimiento, bajo su concepto de balanceador de carga y proxy inverso facilita la implementación de microservicios integrando los componentes de su infraestructura existente y su configuración dinámica y automática[17].

Otra de las características de Traefik como se observa en la Imagen 6, es que ofrece una interfaz amigable donde se nos presenta información relevante sobre el flujo de las peticiones que entran a nuestros servicios expuestos en los diferentes subdominios, este punto es importante porque facilita el manejo de implementaciones grandes y complejas en protocolos de nubes públicas, privadas e híbridas.

Trabajo futuro

Para próximos lanzamientos del prototipo se espera la incorporación de esta arquitectura basada en eventos en otros procesos internos de los aplicativos, para ello, se evaluará el prototipo entregado y se iniciará un proceso de migración hacia este tipo de arquitecturas. Adicionalmente, se plantea la incorporación de kubernetes para todos los aplicativos desplegados en producción, debido a que algunos se encuentran desplegados por medio de Docker Swarm a través de archivos docker-compose.

Conclusiones

La implementación de una arquitectura basada en eventos como kafka, no sólo garantiza que la información siempre esté disponible, sino que también optimiza procesos que a través de llamados continuos de http no lograrían tener la misma eficiencia que los procesos de Kafka.
La incorporación de contenedores son de carácter fundamental en el desarrollo de software, ya que genera instancias ligeras y mejora el tiempo de despliegue de aplicativos a producción. Además permite manejar diferentes entornos, por lo que facilita mantener entornos de prueba y producción. Kubernetes ofrece un entorno robusto y eficiente para el despliegue de aplicativos en producción, todo esto a través del manejo de unos manifiestos estructurados que permiten tener el control de lo

que sucede en nuestro ambiente de producción y nos permite hacer cambios de una manera sencilla y óptima.
Los servicios en la nube tales como GCloud, aws o Azure, ofrecen una infraestructura que permite tener una alta disponibilidad de los aplicativos alojados, así mismo una elasticidad en requerimientos físicos de las máquinas que disponen, por tanto son de gran utilidad para despliegues a producción.

References

[1] What is Extract, Transform, Load? Definition, Process and Tools. (2020, August 12). Retrieved 14 October 2020, from https://www.talend.com/resources/what-is-etl/

[2] Wheatley, M. (2020, June 26). Gartner says container adoption will grow rapidly, but it won't be that profitable. Retrieved 14 October 2020, from https://siliconangle.com/2020/06/25/gartner-says-container-adoption-will-grow-rapidly-wont-profitable/

[3] Microservices. (2014). Retrieved 13 October 2020, from https://martinfowler.com/articles/microservices.html

[4] Containers. (2020, June 1). Retrieved 13 October 2020, from https://www.ibm.com/cloud/learn/containers

[5] What are Containers and their benefits —. (n.d.). Retrieved 13 October 2020, from https://cloud.google.com/containers

[6] What is Kubernetes? (2020, June 17). Retrieved 13 October 2020, from https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

[7] What is event-driven architecture? (n.d.). Retrieved 13 October 2020, from https://www.redhat.com/en/topics/integration/what-is-event-driven-architecture

[8] Micro Frontends - Extendiendo la idea de microservicio al desarrollo frontend. (n.d.). Retrieved 14 October 2020, from https://micro-frontends-es.org/

[9] Event-Driven Architecture. (n.d.). Retrieved 14 October 2020, from https://aws.amazon.com/es/event-driven-architecture/

[10] ¿Qué son los contenedores Linux (LXC)? (n.d.-b). Redhat. Retrieved January 29, 2021, from https://www.redhat.com/es/topics/containers/whats-a-linux-container

[11] Google Cloud. (n.d.). Cloud Computing Services —. Retrieved January 29, 2021, from https://cloud.google.com/

[12] kind. (n.d.). Kind. Retrieved January 29, 2021, from https://kind.sigs.k8s.io/

[13] Gupta, A. (2020, June 9). Kafka on Kubernetes, the Strimzi way! (Part 1). DEV Community. https://dev.to/azure/kafka-on-kubernetes-the-strimzi-way-part-1-57g7

[14] Strimzi - Apache Kafka on Kubernetes. (n.d.). Strimzi. Retrieved January 29, 2021, from https://strimzi.io/

[15] Entender los Objetos de. (2020, May 30). Kubernetes. https://kubernetes.io/es/docs/concepts/overview/working-with-objects/kubernetes-objects/

[16] Rising, B. (2018, September 17). 5 Constructs you must know to get started with Kubernetes. Medium. https://medium.com/slalom-technology/5-constructs-you-must-know-to-get-started-with-kubernetes/

[17] Traefik, The Cloud Native Application Proxy — Traefik Labs. (n.d.). Traefik Labs: Makes Networking Boring. Retrieved January 29, 2021, from https://traefik.io/traefik/

[18] Mwai, E. (2020, February 7). Querying Microsoft Graph API with Python - Towards Data Science. Medium. https://towardsdatascience.com/querying-microsoft-graph-api-with-python-269118e8180c