Sistema IoT con Apache Kafka, Zookeeper, MongoDB y Flask. (Noviembre 2021)

A. Cárdenas [1], S. Matiz [1], J. Núñez [2], M. Otavo [2]
alejandro.cardenas_g@uao.edu.co, juan.matiz@uao.edu.co, julian.nunez@uao.edu.co, mishell.otavo@uao.edu.co
[1]*Ingeniería Electrónica y telecomunicaciones,* [2]*Ingeniería Informática*
Facultad de Ingeniería, Universidad Autónoma de Occidente, Ala Norte tercer piso, Cali Colombia

***Resumen*—En el presente documento se presenta un breve definición de la tecnología Apache Kafka y se realiza la implementación de un sistema de distribución de datos en IoT a través de las tecnologías de Streaming de eventos Apache Kafka y Apache Zookeeper. Este proceso se lleva a cabo a través de una metodología simple que se compone en la formulación del problema, un proceso de investigación de las tecnologías y su funcionamiento, la implementación de los servicios necesarios independientemente y la conexión entre ellos, terminando con un proceso de validación con soporte de una simulación en Proteus que permite emular el proceso de transmisión y el despliegue de una aplicación web simple en Flask que permite visualizar los datos. Finalmente, se muestran las pruebas realizadas y las conclusiones ligadas al proyecto implementado.**

Palabras claves: Zookeeper; Kafka; IoT;.

***Abstract*—In the following article, a system of data distribution is deployed for IoT application through technologies of events streaming such as Apache Kafka and Apache Zookeeper. The whole process is made by a simple methodology, the which is composed by the problem formulation, a research process about the technologies and their functioning, the deployment of the required services independently and the connection between them, ending up with a validation process supported by a Proteus Simulation that allows to emulate the data transfer process and the deployment of a simple web application made in Flask to visualize the data. Finally, the made tests and the conclusions related to the implemented project are shown.**

Key-Words: Zookeeper; Kafka; IoT;.

I. Introducción

Las tecnologías emergentes y sus nuevas especificaciones han permitido que múltiples aplicaciones en diferentes campos consigan un mejor rendimiento en cuanto a muchos aspectos. Estas tecnologías han permitido que se alcance el concepto de medición, monitoreo, control y demás en tiempo real, debido a la necesidad de conocer el estado de un sistema, una aplicación, un base de datos o un evento a cada momento para poder actuar frente a esto o permitir a otros sistemas interactuar mediante esto. Tecnologías como Apache Kafka y Zookeeper han permitido que empresas grandes en el mundo como Netflix, Linkedin, entre otras utilicen estos servicios en aplicaciones como realizar recomendaciones a sus usuarios en tiempo real o bien monitorear toda una arquitectura en la nube. Asimismo, estas tecnologías se utilizan para muchas otras aplicaciones como el IoT, donde se requiere en muchos casos mantener un tiempo de muestreo constante, rápido y confiable para controlar y conocer los estados de distintos dispositivos alrededor de un contexto físico.

II. Patrón de publicación/subscripción

De acuerdo con AWS [1], la mensajería de publicación/suscripción es una forma de comunicación asincrónica que se utiliza en arquitecturas de microservicios y en la computación sin servidor (serverless).

La mensajería de publicación/suscripción es un patrón que se caracteriza porque el productor (o publicador) de un dato (mensaje) no lo dirige específicamente a un consumidor (o suscriptor), sino que el publicador clasifica el mensaje de alguna manera y ese consumidor se suscribe para recibir cierto tipo de mensajes. Los sistemas de publicación/suscripción suelen tener un intermediario, un punto central donde se publican los mensajes, para facilitar esto (ver figura 1).

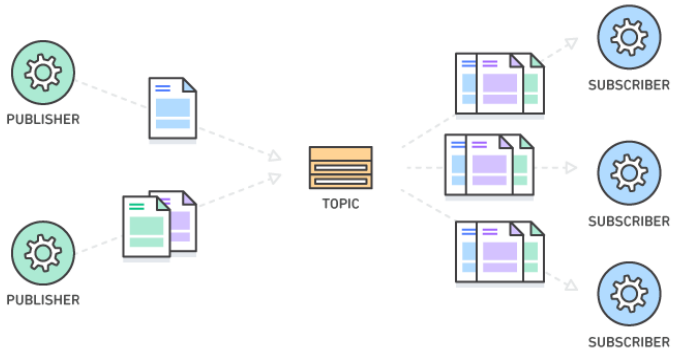

Figura 1: Modelo Pub/Sub
Fuente: [1]

A. Importancia

Antes que las empresas incluyan este patrón de publicación/suscripción se parte de un mismo sitio: una cola de mensajes simple o un canal de comunicación entre procesos. Por ejemplo, se crea una aplicación que necesita enviar información de monitoreo a algún lugar, por lo que se configura una conexión directa desde la aplicación que capta los datos a una aplicación que muestra las métricas en un tablero. Pero a medida que crecen la cantidad de métricas medidas y la cantidad de servidores que reciben estos datos por conexión directa aumenta la complejidad de la

arquitectura del sistema, algo similar a la figura 2.

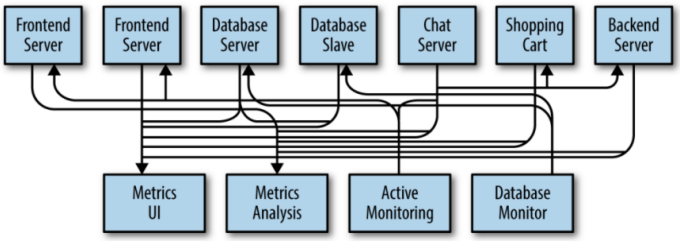

Figura 2: Muchas métricas publicadas, usando conexión directa
Fuente: [2]

El sistema de la figura 2 se vuelve muy complejo, y por lo tanto se hace necesario configurar una sola aplicación que reciba métricas de todas las aplicaciones que existen y proporciona un servidor para consultar esas métricas para cualquier sistema que las necesite. Esto reduce la complejidad de la arquitectura (ver figura 3) y de esta forma se consigue un sistema de mensajería de publicación y suscripción.

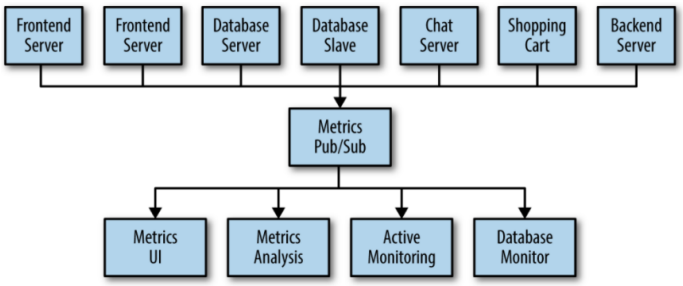

Figura 3: Sistema de publicación / suscripción de métricas
Fuente: [2]

III. APACHE KAFKA

Según [3] Apache Kafka es una plataforma distribuida de transmisión de datos que permite publicar, almacenar y procesar flujos de registros, y suscribirse a ellos, en tiempo real. Está diseñada para manejar flujos de datos de varias fuentes y distribuirlos a los diversos usuarios. Además es de código abierto y de acuerdo a [4] actualmente es utilizada por miles de empresas para canalizaciones de datos de alto rendimiento, análisis de transmisión, integración de datos y aplicaciones de misión crítica, debido a que garantiza un flujo continuo de los datos y la interpretación de estos para que la información correcta esté en el lugar correcto, en el momento adecuado. Algunos ejemplos de aplicaciones con apache kafka son:

- Procesar pagos y transacciones financieras en tiempo real.
- Rastrear y monitorear automóviles, camiones, flotas y envíos en tiempo real.
- Recopilar y reaccionar de inmediato a las interacciones y pedidos de los clientes.
- Monitorear a los pacientes en atención hospitalaria y predecir cambios en la condición para garantizar el tratamiento oportuno en emergencias.

Estas tecnologías tienen características importante como la baja latencia en la transmisión de datos (menor a 10ms), lo que es llamado tiempo real. Sin embargo, esta latencia se debe a que los mensajes no puede soportar una gran cantidad de datos, por lo cual el contenido de un mensaje no puede ser mayor a 1 MB.

A. Eventos

Un evento, registro o mensaje es un dato que puede ser almacenado y consultado en Kafka. La estructura de un evento tiene una clave (significa la partición en la cual se almacenan), un valor, información de cuando fue publicado y metadatos opcionales.

B. Servidores y clientes

Los servidores de kafka funcionan como un clúster que puede abarcar varios centros de datos o regiones de la nube. Algunos de estos servidores forman la capa de almacenamiento, denominados Brokers. Otros servidores ejecutan Kafka Connect para importar y exportar datos continuamente como flujos de eventos para integrar Kafka con los sistemas existentes, que pueden ser bases de datos relacionales y otros clústeres de Kafka.

Los clientes pueden ser de 2 tipos: Productor o suscriptor. Los productores, como su nombre lo indica, son los encargados de suministrar (escribir) eventos a Kafka, y los consumidores son aquellos que se suscriben, leen y procesan esos eventos que se encuentran en Kafka. En este sistema, los productores y suscriptores no se conocen entre sí, ni tampoco es relevante que se conozcan, ya que no están conectados directamente. Gracias a lo anterior Kafka permite una gran escalabilidad.

C. Temas (Topic o Tópico)

Los eventos se almacenan en temas o tópicos, y se puede asemejar a un directorio o carpeta, donde se guardan archivos (eventos) relacionados entre sí. Además, estos temas (tópicos) permiten conservar los eventos dentro de ellos por mucho tiempo, hasta el momento que se decida deshacerse de los eventos más viejos, y el rendimiento de Kafka no se ve afectado por la cantidad de datos almacenados en él.

D. Particiones

Cada tema se encuentra particionado, lo que se traduce en que cada tópico se puede distribuir en varios Brokers de Apache. Esto es lo que permite su escalabilidad, y permite que muchos productores y consumidores escriban y lean eventos al mismo tiempo y sin problema. Los eventos con una misma clave son almacenados en una misma partición (ver figura).

E. Replicas (opcional)

Cuando el sistema en donde se usa es muy grande, es de gran importancia y se ve la necesidad de tener alta disponibilidad, se pueden generar réplicas de los tópicos en diferentes zonas geográficas, para que sea tolerable a los fallos y pueda proveer el servicio todo el tiempo el la mayoría de los casos. Lo más común es tener un factor de réplica igual a 3.

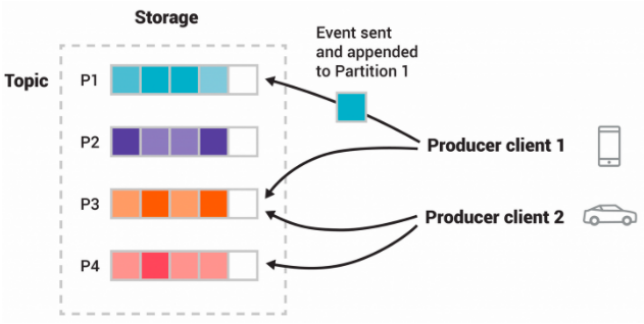

Figura 4: Ejemplo de particiones de un tópico en Apache Kafka
Fuente: [4]

IV. ZOOKEEPER.

Esta tecnología permite manejar o administrar sistemas distribuidos para coordinarlos correctamente. Zookeeper es una solución para estas necesidades debido a su simplicidad en la coordinación de procesos similar a un sistema de archivos, capacidad de escalabilidad, velocidad y fiabilidad o tolerancia a fallos. [5]. Normalmente este software es utilizado para sistemas distribuidos, big data usando otras herramientas como Hadoop o Kafka.

La terminología de Zookeeper se basa en nodos, los cuales son conocidos como Znodos que se caracterizan por tener un path asociado o ruta, por almacenar datos, entre otras, sin embargo, también se caracteriza en que el Znodo puede ser efímero o persistente, es decir que puede persistir o no los datos almacenamos al apagar el proceso [6].

Como se observa en la siguiente figura, la arquitectura de Zookeeper se basa en manejar múltiples servidores, lo que normalmente se traduce en un clúster de znodos que manejan y gestionan las diferentes operaciones entre los brokers de Apache Kafka.

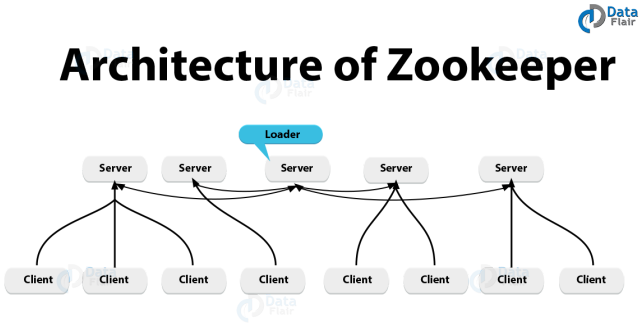

Figura 5: La arquitectura de Zookeeper.
Fuente: [2, Fig. 1]

Como se menciona anteriormente, zookeeper es este servicio que permite coordinar aplicaciones distribuidas a través de la gestión de los subsistemas existentes, siendo finalmente un centro de control.

Las funciones de Zookeeper son extensas en cualquier aplicación que le requiere, pero específicamente para Kafka, este servicio se encarga de intercambiar información con los brokers de Apache Kafka, los productos y consumidores. La información que se intercambia en estas transacciones son las direcciones de los brokers, metadatos para la detección de la carga de trabajo, para balancear esta eficientemente, realizar procesos de descubrimiento y control de los brokers en el clúster , entre otros [7].

En las versiones 2.x de Kafka, Zookeeper es un servicio totalmente necesario, por lo que si este no pertenece al sistema, Apache Kafka no funcionará. Sin embargo, para nuevas versiones de Kafka (3.x en adelante), Zookeeper es reemplazado por otras tecnologías de administración como Kafka Raft u otras tecnologías emergentes.

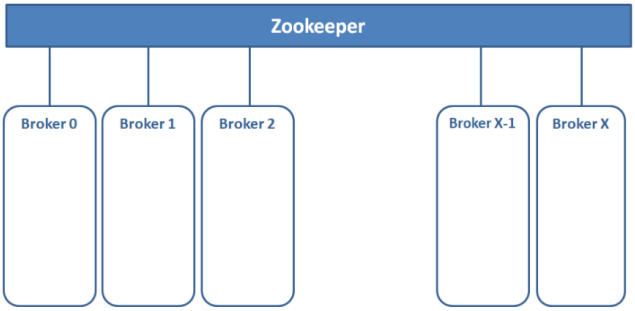

Figura 6: Zookeeper con Brokers de Kafka.
Fuente: [7, Sec. Zookeeper]

El sistema mostrado anteriormente, refleja lo que significa un clúster, el cuál se compone de Apache Zookeeper manejando los hilos en cada broker de Apache Kafka. El conjunto de brokers se denomina un clúster de Kafka. Como se menciona en la sección de Apache Kafka de este documento, existen conceptos como particiones, mensajes, replicación entre otras, esto realmente es manejado por todo este sistema mostrado. Realmente el objetivo de un Clúster es almacenar un grupo de registros por una identificación denominada tópico. Por otra parte, en los conceptos de replicación y partición es importante tener en cuenta que la partición siempre tiene un leader y múltiples followers, debido a un broker es dueño de una partición, quien es la instancia que se comunicará con los consumidores o bien los productores, por otra parte, la replicación es un factor que alude a la tolerancia a fallos y alta disponibilidad, ya que se puede implementar un sistema con múltiples clusters, sin embargo para ello, es necesario tener en cuenta algunas recomendaciones para gestionar correctamente el rendimiento

y eficiencia del sistema como el número de brokers de Kafka por cada Nodo de Zookeeper y parámetros de red de cada broker.

V. Tecnologías para el proyecto

Las herramientas a emplear en este proyecto son:

- Software Proteus Design Suite
- Aplicación Arduino IDE
- Lenguajes de programación Python
- Framework de Python: Flask
- Cuenta de Microsoft Azure para la creación de una máquina virtual con sistema operativo Ubuntu 20.04
- Software Free Virtual Serial Ports Emulator o similares.
- Base de datos MongoDB

VI. Desarrollo del proyecto

A continuación se presenta la implementación de un proyecto de IoT para entender y poner en práctica el funcionamiento de Apache Kafka y Zookeeper. El proyecto integra las herramientas antes mencionadas.

Este sistema de IoT consta de un nodo de IoT simulado con proteus y un programa en arduino (ver figura 7) que imita la captura de diferentes variables ambientales: humedad, temperatura, sensación térmica y radiación solar. Estas variables serán enviadas a través de puerto serial con ayuda del software "Free Virtual Serial Ports Emulator" en formato Json, a un cliente publicador de Apache Kafka, ubicando en la misma máquina de la simulación de proteus.

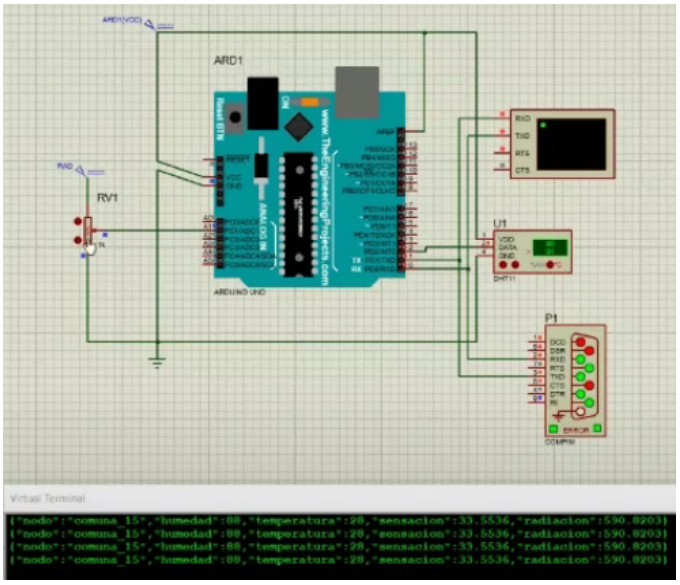

Figura 7: Simulación nodo de IoT con Proteus
Fuente: Propia

Este cliente publicador es un programa en Python que se encargará de recepcionar los datos entregados por el nodo de IoT y enviará estos datos a un tópico llamado "Prueba" del servidor de apache kafka, el cual se encuentra hospedado remotamente y en la nube en una máquina virtual de Microsoft Azure. Una vez que el aplicativo de Python hace una publicación de un nuevo evento con la información de las variables, el servidor de apache kafka almacenará este mensaje en forma de evento al tópico "Prueba".

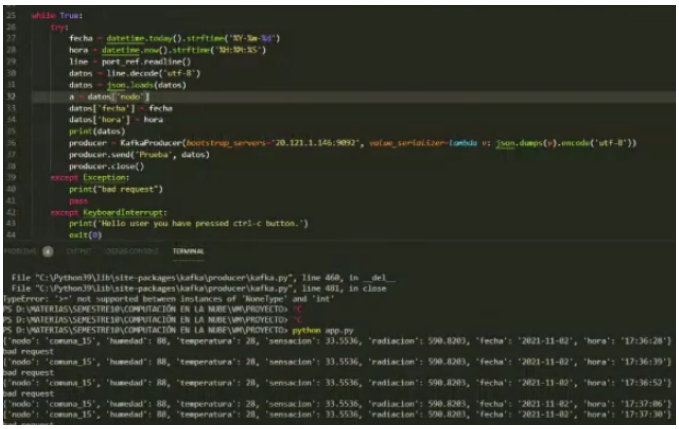

Figura 8: Código en Python de productor de eventos de Apache Kafka
Fuente: Propia

Posteriormente, la misma máquina virtual que tiene el servicio que provee apache kafka también funcionará como un suscriptor (consumidor) de Apache Kafka gracias a un aplicativo en python, y a su vez, este aplicativo insertará los nuevos datos leídos a una base de datos en MongoDB, ubicada también en el servidor. Además, previamente en la máquina virtual se habrá instalado un servidor web usando framework de python Flask, el cual desplegará un aplicativo web que permitirá listar todos los tópicos existentes en apache kafka que han sido guardados en la base de datos de MongoDB, y permitirá obtener información detallada de cada una de las variables almacenadas en cada tópico (para está prueba solo se inserto en el topico "Prueba").

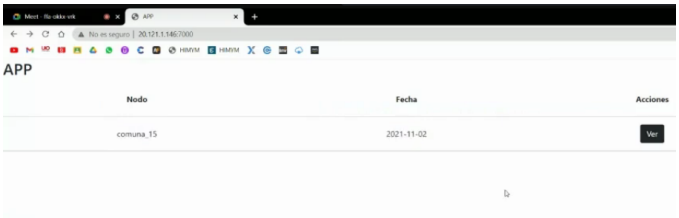

Figura 9: Página web en el servidor (MV) y despliegue de tópicos de Kafka
Fuente: Propia

La página desplegada por la máquina virtual puede ser consultada con la dirección IP pública de la máquina virtual (o dominio sí se ha configurado así) por el puerto 7000.

Figura 10: Página web y vista de información de un nodo específico. Fuente: Propia

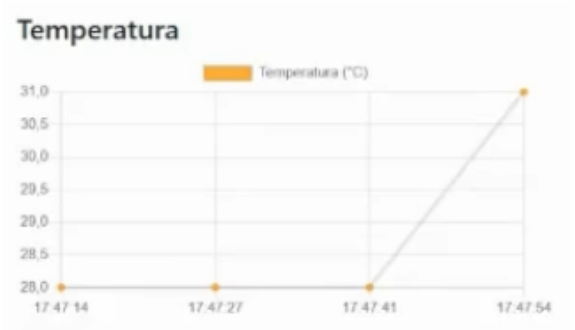

Figura 11: Página web y vista de información de un nodo específico. Fuente: Propia

VII. Conclusiones

En este documento se ha presentado brevemente la definición de Apache Kafka y sus componentes, y se ha implementado un ejemplo práctico de su implementación en un sistema IoT simulado.

Gracias a esta práctica nos damos cuenta de la importancia de esta herramienta en el mundo real, donde la complejidad de las redes se va volviendo cada vez más grande y difícil de entender, y la tecnología de apache Kafka logra solucionar satisfactoria los requerimientos que las grandes empresas tiene para solventar la gran demanda constante que los usuarios tienen, al igual que permite una mayor disponibilidad de los servicios.

Las tecnologías de Apache Kafka y Zookeeper son herramientas que permiten implementar sistemas complejos de monitoreo y transacciones de estado entre múltiples aplicaciones bajo el modelo publicador/suscriptor en tiempo real debido a la baja latencia se maneja en Kafka. Por otra parte, estas tecnologías ofrecen atributos importantes en el despliegue de cualquier aplicación grande como la alta disponibilidad, la escalabilidad y la tolerancia a fallos.

Finalmente, estas tecnologías permiten implementar sistemas distribuidos complejos y robustos con la limitante de que los mensajes tienen un tamaño máximo muy bajo, lo que no permite gestionar gran cantidad de datos por transacción, pero que en contraste se ve compensado por su baja latencia para trabajos en tiempo real.

El código fuente de este proyecto se encuentra en un repositorio público de github: https://github.com/janumejia/CompuNube

Referencias

[1] "What is Pub/Sub Messaging?" (2019) AWS. [En línea]. Disponible en: https://aws.amazon.com/es/pub-sub-messaging/ , accedido en noviembre 5, 2021

[2] Narkhede N. & Shapira G. & Palino T, "Kafka: The Definitive Guide" (2017) O'Reilly Media, Inc. [En línea]. Disponible en: https://www.confluent.io/resources/kafka-the-definitive-guide/ , accedido en noviembre 6, 2021.

[3] "¿Qué es Apache Kafka?" (2020) redhat. [En línea]. Disponible en: https://www.redhat.com/es/topics/integration/what-is-apache-kafka , accedido en noviembre 6, 2021.

[4] "INTRODUCTION" Apache Kafka. [En línea]. Disponible en: https://kafka.apache.org/ , accedido en noviembre 6, 2021.

[5] "What is ZooKeeper & How Does it Support Kafka?" (2021). [En línea]. Disponible en: https://dattell.com/data-architecture-blog/what-is-zookeeper-how-does-it-support-kafka/ , accedido en noviembre 6, 2021.

[6] "Apache Zookeeper Architecture – Diagrams & Examples". [En línea]. Disponible en: https://data-flair.training/blogs/zookeeper-architecture/ , accedido en noviembre 6, 2021.

[7] Victor M. (2018). "Aprendiendo apache kafka (Parte 2): Conceptos básicos". [En línea]. Disponible en: https://enmilocalfunciona.io/aprendiendo-apache-kafka-parte-2-2/