**APLICACIÓN DISTRIBUIDA**

**APLICACIÓN “Sistema Distribuido de video usando Kafka”**

**AUTORES:** Jorge Erazo y Cristian Suntaxi

**TRABAJO FIN DE SEMESTRE – NOVENO SEMESTRE DE LA ASIGNATURA DE PROGRAMACIÓN DISTRIBUIDA**

**UNIVERSIDAD CENTRAL DEL ECUADOR  
FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS.  
INGENIERÍA EN INFORMÁTICA  
QUITO, AGOSTO 2023**

Contenido

[Introducción 3](#_Toc144225385)

[Objetivos 3](#_Toc144225386)

[Objetivo general 3](#_Toc144225387)

[Objetivos específicos 3](#_Toc144225388)

[Desarrollo 3](#_Toc144225389)

[Beneficios del Streaming de Video Distribuido 4](#_Toc144225390)

[Arquitectura de la aplicación 5](#_Toc144225391)

[Guía de uso 6](#_Toc144225392)

[Conclusiones 6](#_Toc144225393)

[Anexos 6](#_Toc144225394)

[Bibliografía 8](#_Toc144225395)

# Introducción

La arquitectura distribuida está de moda en la actualidad. A medida que los programadores se sienten frustrados con los monolitos problemáticos que son sus proyectos heredados, los microservicios y la arquitectura orientada a servicios (SOA) parecen prometer una cura para todos sus problemas**.**

Kafka se ha convertido en una de las tecnologías clave para responder a esta pregunta de cómo transmitir mensajes en tiempo real y registros de eventos a un sistema distribuido y escalado masivamente. En este proyecto, echamos un vistazo a Kafka, lo compararemos con otros intermediarios de mensajes que existen y nos ensuciaremos las manos con un proyecto de transmisión de video.

# Objetivos

## Objetivo general

Analizar, evaluar y comprender en profundidad el funcionamiento de un aplicativo distribuido en un entorno controlado, con el propósito de identificar oportunidades de mejora en términos de rendimiento, escalabilidad y eficiencia.

## Objetivos específicos

* Realizar el levantamiento del aplicativo distribuido en un entorno de prueba, asegurando la configuración adecuada de todos los componentes y la interconexión correcta entre ellos.
* Ejecutar una serie de pruebas, diseñando y ejecutando escenarios de carga y situaciones que simulen condiciones reales de uso, con el fin de identificar posibles cuellos de botella, evaluar el rendimiento del aplicativo bajo diferentes circunstancias y recopilar datos relevantes para el análisis.
* Analizar las características de una aplicación distribuida estudiadas a lo largo del semestre.

# Desarrollo

"Streaming de Video Distribuido con Python y Kafka" se refiere al concepto de utilizar el sistema de mensajería Kafka junto con la programación en Python para habilitar la transmisión eficiente y escalable de contenido de video en sistemas distribuidos. Kafka es una plataforma distribuida de transmisión en tiempo real popular diseñada para manejar grandes volúmenes de datos con alta capacidad de rendimiento, tolerancia a fallos y procesamiento en tiempo real. Se utiliza comúnmente para el enrutamiento de eventos, canalización de datos y análisis en tiempo real.

La combinación de Python y Kafka para el Streaming de video distribuido permite a los desarrolladores crear aplicaciones que pueden abordar los desafíos del procesamiento de datos de video en tiempo real, su distribución y consumo.

A continuación, presentamos una descripción general de cómo funciona esta configuración:

**Fuente de Video**: La fuente de video puede ser una cámara, un archivo de video o un flujo de video de otra aplicación. Python se utiliza para capturar los fotogramas de video y convertirlos a un formato que pueda ser procesado y transmitido.

**Procesamiento en Python**: Bibliotecas de Python como OpenCV se pueden utilizar para procesar los fotogramas de video. Esto podría involucrar tareas como detección de objetos, reconocimiento facial u otras técnicas de procesamiento de imágenes. Una vez que los fotogramas son procesados, están listos para ser enviados para la transmisión.

**Productor de Kafka**: Los fotogramas de video procesados se envían a un tema de Kafka mediante un productor de Kafka implementado en Python. Los productores de Kafka aseguran que los fotogramas sean distribuidos eficientemente a los brokers de Kafka.

**Brokers de Kafka**: Los brokers de Kafka son los componentes centrales del clúster de Kafka. Almacenan y administran los datos en Streaming, asegurando la tolerancia a fallos y la alta disponibilidad.

**Temas de Kafka**: Los temas de Kafka actúan como canales para organizar los flujos de datos. En este caso, los fotogramas de video se envían a un tema de Kafka específico dedicado al Streaming de video.

**Consumidor de Kafka**: En el extremo receptor, otra aplicación Python actúa como consumidor de Kafka. Se suscribe al tema de Kafka y procesa los fotogramas de video entrantes.

**Visualización de Video o Procesamiento Adicional**: La aplicación consumidora en Python puede mostrar el flujo de video en tiempo real, guardarla como archivo de video o incluso realizar un procesamiento adicional antes de mostrar o almacenar los fotogramas.

## Beneficios del Streaming de Video Distribuido

**Escalabilidad**: La naturaleza distribuida de Kafka permite la escalabilidad horizontal, lo que lo hace adecuado para manejar grandes volúmenes de datos de video y garantizar que el sistema pueda crecer según sea necesario.

**Tolerancia a Fallos**: La arquitectura de replicación y tolerancia a fallos de Kafka garantiza que incluso si un bróker falla, los datos permanezcan disponibles y puedan ser recuperados.

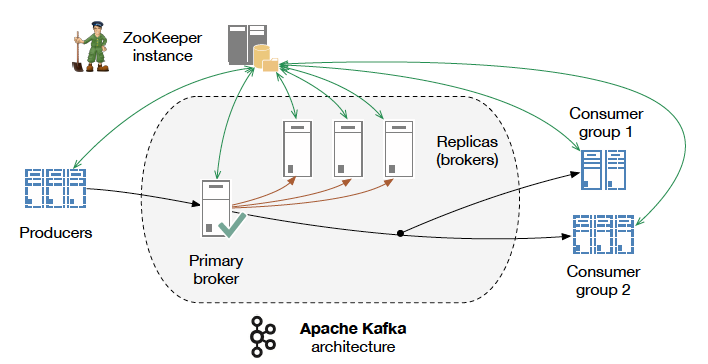
**Procesamiento en Tiempo Real**: La baja latencia y las capacidades en tiempo real de Kafka son adecuadas para aplicaciones que requieren procesamiento y distribución instantáneos o casi instantáneos de contenido de video.

**Flexibilidad**: Las amplias bibliotecas y la sintaxis sencilla de Python facilitan el desarrollo de aplicaciones que procesan y manipulan fotogramas de video.

**Casos de Uso**: Esta configuración se puede utilizar para una variedad de aplicaciones, incluyendo análisis de video en tiempo real, transmisión en vivo, monitoreo de seguridad y más.

Sin embargo, es importante tener en cuenta que la implementación de Streaming de video distribuido requiere una cuidadosa consideración de factores como el ancho de banda de la red, los recursos del hardware y la complejidad del procesamiento de video. Además, garantizar baja latencia y Streaming de alta calidad puede requerir optimizaciones en varios niveles de la arquitectura del sistema.

## Arquitectura de la aplicación



La arquitectura distribuida para el "Streaming de Video Distribuido con Python y Kafka" implica el diseño y la organización de componentes que trabajan juntos para lograr la transmisión eficiente de contenido de video a través de sistemas distribuidos utilizando Python y Kafka. Esta arquitectura se basa en la capacidad de Kafka para gestionar el flujo de datos en tiempo real y en las capacidades de procesamiento y comunicación de Python.

Aquí se describe la arquitectura distribuida para esta aplicación:

**Productor de Video (Python):** En el extremo del video, se tiene un componente escrito en Python que actúa como productor. Su función es capturar los fotogramas de video de una fuente (como una cámara) y procesarlos, si es necesario, antes de enviarlos al sistema de transmisión.

**Sistema de Colas (Kafka):** Se utiliza Kafka como el sistema de colas central. Los productores de video envían los fotogramas procesados a los tópicos de Kafka. Los tópicos son canales de comunicación en los que los datos se organizan y distribuyen a los consumidores.

**Consumidores de Video (Python):** En el extremo del consumidor, existen uno o varios componentes escritos en Python que actúan como consumidores. Estos consumidores se suscriben a los tópicos relevantes en Kafka para recibir los fotogramas de video. Aquí, los consumidores pueden realizar diversas acciones, como mostrar el video en tiempo real, almacenarlo o aplicar procesamientos adicionales.

**Procesamiento Adicional (Python):** En algunos casos, los consumidores pueden realizar procesamientos adicionales en los fotogramas de video recibidos. Esto podría incluir análisis de video en tiempo real, detección de objetos, reconocimiento facial u otras técnicas de procesamiento de imágenes.

**Escalabilidad y Redundancia**: Para garantizar la escalabilidad y la redundancia, la arquitectura se puede expandir utilizando múltiples brokers de Kafka y consumidores distribuidos. Esto asegura que el sistema pueda manejar grandes volúmenes de datos y tolerar fallas en componentes individuales.

**Almacenamiento o Transmisión Adicional:** Dependiendo de las necesidades del sistema, los consumidores pueden almacenar el video en una base de datos, transmitirlo a otras aplicaciones o servicios, o realizar cualquier acción adicional requerida.

# Guía de uso

Para la configuración y su posterior fase de pruebas se detalla a continuación en el link adjunto donde se encuentra nuestra aplicación y su correspondiente guía de uso.

<https://github.com/George241915/DistributedStreamingWithPythonAndKafka>

# Conclusiones

La implementación de "Streaming de Video Distribuido con Python y Kafka" ofrece una solución robusta y escalable para la transmisión eficiente de contenido de video en sistemas distribuidos. A través de la combinación de las capacidades de Kafka y la versatilidad de Python, esta arquitectura permite la gestión efectiva de datos en tiempo real, el procesamiento de video y la distribución a través de múltiples componentes.

Eficiencia en el Procesamiento: La arquitectura aprovecha la capacidad de Python para procesar los fotogramas de video de manera eficiente. Esto permite realizar análisis en tiempo real y aplicar técnicas de procesamiento de imágenes según las necesidades.

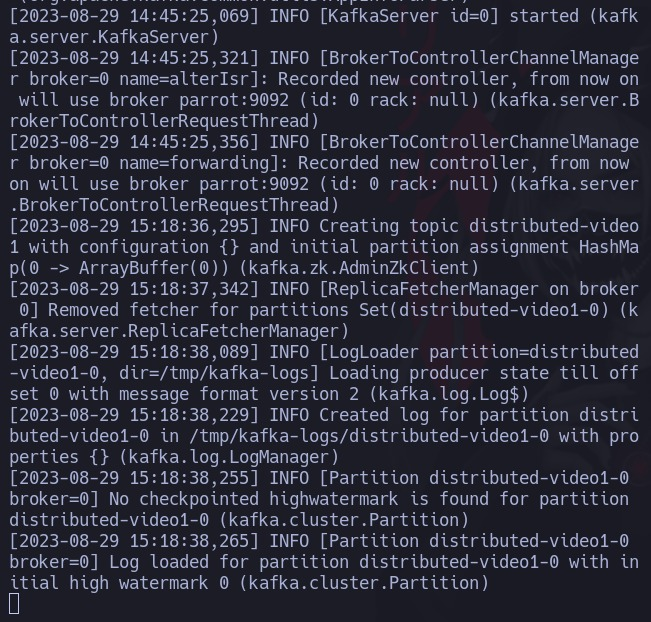
Escalabilidad: La arquitectura distribuida se beneficia de la escalabilidad horizontal de Kafka, lo que permite manejar volúmenes crecientes de datos y tráfico de video sin comprometer el rendimiento.

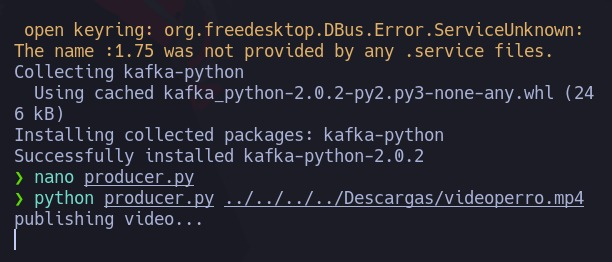
Distribución Equitativa: Kafka distribuye datos equitativamente entre los consumidores, lo que asegura que todos los consumidores tengan acceso a la misma información y facilita el procesamiento paralelo.

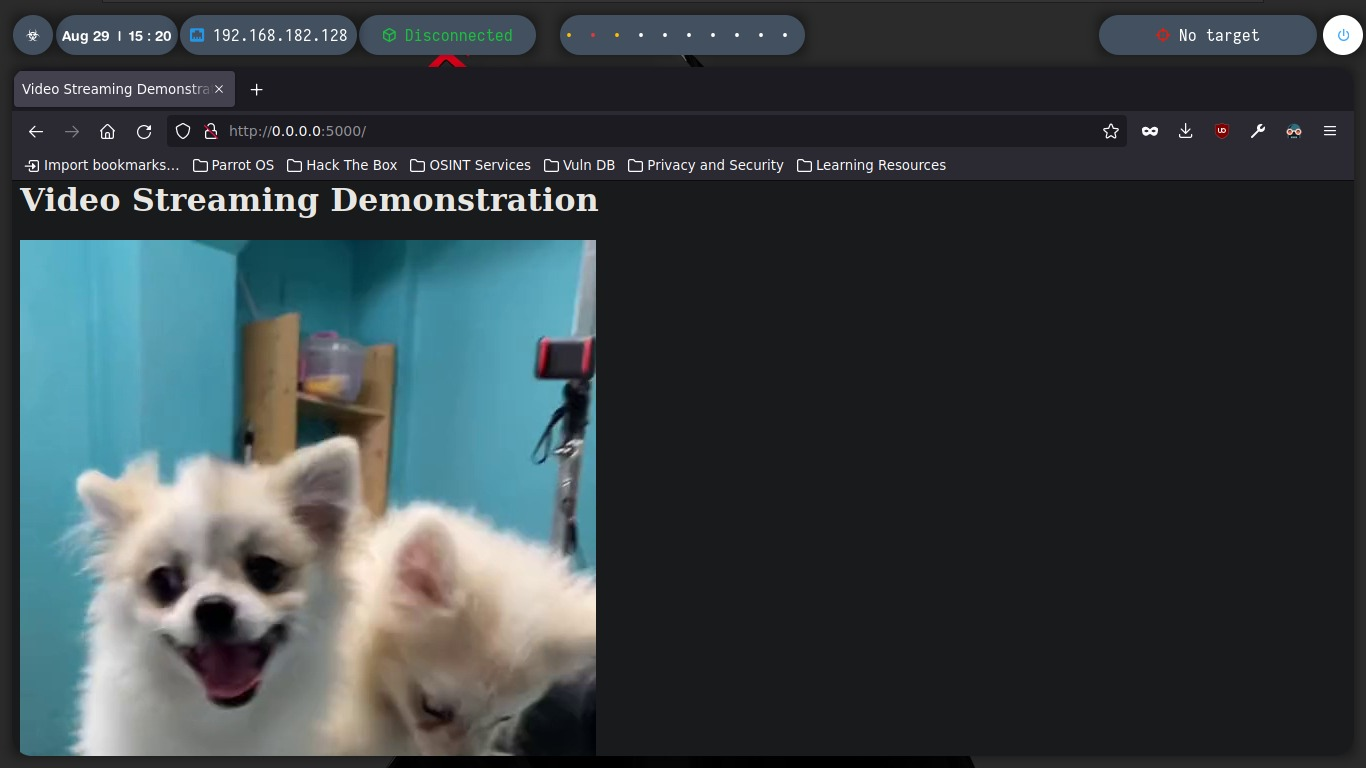
# Anexos

Dentro de esta sección adjuntamos algunas capturas de pantalla entre ellas algunas configuraciones que realizamos y finalmente la aplicación de Streaming levantada.







****

# Bibliografía

* Link del repositorio

<https://github.com/George241915/DistributedStreamingWithPythonAndKafka>