Presentación adicional

ANDRES ARMANDO SANCHEZ MARTIN

HUMBERTO RUEDA CATAÑO

FACULTAD DE INGENIERÍA

ARQUITECTURA DE SOFTWARE

PONTIFICIA UNIVERSIDAD JAVERIANA

BOGOTÁ D.C.

2024

I. INTRODUCCIÓN

El presente documento contiene los lineamientos y directrices para el desarrollo básico de una comparativa entre dos colas. Este documento también tiene como objetivo establecer comparativas entre ambas implementaciones con la razón de establecer diferencias claras. Además, establece una breve introducción a las tecnologías utilizadas en esta presentación como lo son Kafka y RabbitMQ

II. DEFINICIONES, HISTORIA Y EVOLUCIÓN

1. Comunicación asíncrona y conceptos relacionados

La comunicación asíncrona es un tipo de comunicación en la que los procesos se comunican entre sí sin necesidad de estar sincronizados en el tiempo. Esto significa que un proceso puede enviar un mensaje y continuar con su ejecución sin esperar a que el mensaje sea recibido y procesado por el otro proceso. Algunos conceptos relacionados a la comunicación asíncrona son los siguientes:

- Concurrente vs. Paralelo: La programación concurrente se refiere a la ejecución de varias tareas de manera que puedan parecer simultáneas, mientras que la programación paralela se refiere a la ejecución simultánea de múltiples tareas utilizando múltiples procesadores.
- Bloqueante vs. No Bloqueante: Una operación bloqueante hace que el proceso espere hasta que se complete, mientras que una operación no bloqueante permite que el proceso continúe su ejecución.
- Eventos y Callbacks: En la comunicación asíncrona, los eventos y los callbacks son mecanismos comunes para manejar tareas una vez que una operación asíncrona se completa.
- Futuros y Promesas: Son objetos que representan un valor que puede estar disponible en el futuro, útil en la programación asíncrona.

2. ¿Qué es una cola? ¿Qué tipos de cola hay?

Una cola es una estructura de datos lineal que sigue el principio FIFO (First In, First Out), donde el primer elemento en entrar es el primero en salir.

Tipos de Colas

- Cola Simple: Solo permite operaciones de inserción (enqueue) al final y eliminación (dequeue) al principio.
- Cola Doble (Deque): Permite inserción y eliminación de elementos tanto al principio como al final.
- Cola Prioritaria: Cada elemento tiene una prioridad, y los elementos con mayor prioridad son procesados antes que los de menor prioridad.
- Cola Circular: Los elementos se tratan como si estuvieran en un círculo, permitiendo el uso eficiente del espacio.

3. ¿Cómo funcionan las colas?

Las colas poseen algunas operaciones básicas para su funcionamiento, algunas de estas son:

- Enqueue: Agregar un elemento al final de la cola.
- Dequeue: Eliminar y retornar el primer elemento de la cola.
- Front: Obtener el primer elemento sin eliminarlo.
- IsEmpty: Verificar si la cola está vacía.
- Size: Obtener el número de elementos en la cola.

Las colas se pueden implementar usando arrays, listas enlazadas o incluso estructuras más avanzadas como heaps para colas prioritarias. La eficiencia de las operaciones depende de la implementación específica.

4. ¿Para qué sirven las colas?
    - Sistemas de impresión: Manejo de trabajos de impresión en orden de llegada.
    - Administración de Procesos: Gestión de tareas en sistemas operativos.
    - Manejo de Solicitudes en Servidores Web: Procesamiento de solicitudes HTTP en orden de llegada.
    - Interfaz de Usuario: Manejo de eventos de usuario en aplicaciones gráficas.
    - Algoritmos de Grafos: Implementación de algoritmos BFS (Breadth-First Search).
5. Plataformas que implementan colas

- RabbitMQ: Un broker de mensajes que soporta colas para la comunicación entre sistemas distribuidos.

- Apache Kafka: Una plataforma de streaming que utiliza colas para manejar flujos de datos en tiempo real.
- Amazon SQS (Simple Queue Service): Un servicio de colas administrado en la nube por Amazon Web Services.
- Azure Queue Storage: Servicio de colas proporcionado por Microsoft Azure para almacenar grandes volúmenes de mensajes.
- Redis: Aunque principalmente una base de datos en memoria, Redis soporta operaciones de colas.

Para este desarrollo, se implementaron dos colas usando las tecnologías de Apache Kafka y RabbitMQ.

Apache Kafka

Apache Kafka es una plataforma de streaming de eventos distribuida y de código abierto, utilizada para construir pipelines de datos en tiempo real y aplicaciones de streaming. Es capaz de manejar grandes volúmenes de datos en tiempo real y permite la publicación, suscripción, almacenamiento y procesamiento de flujos de eventos.

Historia y Evolución

- Origen: Kafka fue desarrollado originalmente por LinkedIn en 2011 para manejar sus flujos de datos en tiempo real. Fue diseñado como una solución de alto rendimiento y alta disponibilidad para la gestión de datos de eventos en tiempo real.
- Apache Software Foundation: En 2012, Kafka fue donado a la Apache Software Foundation y se convirtió en un proyecto de código abierto.
- Evolución: Desde su creación, Kafka ha evolucionado significativamente:
- Kafka Connect: Introducido en la versión 0.9, permite conectar Kafka con sistemas externos para importar y exportar datos.
- Kafka Streams: Añadido en la versión 0.10, es una biblioteca de procesamiento de streams que permite crear aplicaciones de procesamiento de datos en tiempo real.
- KRaft: Kafka ha estado moviéndose hacia una arquitectura sin Zookeeper (KRaft) para mejorar la eficiencia y escalabilidad.

Kafka se ha convertido en una herramienta fundamental en muchas empresas grandes, incluidas más del 80% de las empresas de Fortune 100, debido a su capacidad para manejar datos a gran escala con baja latencia y alta disponibilidad.

RabbitMQ

RabbitMQ es un sistema de mensajería orientado a mensajes de código abierto que implementa el protocolo Advanced Message Queuing Protocol (AMQP). Es utilizado para la intermediación de mensajes entre sistemas, permitiendo la comunicación asíncrona y desacoplada entre aplicaciones.

Historia y Evolución

- Origen: RabbitMQ fue desarrollado en 2007 por Rabbit Technologies Ltd., y posteriormente adquirido por SpringSource (una división de VMware) en 2010.
- Apache Software Foundation: Aunque no es un proyecto oficial de la Apache Software Foundation, RabbitMQ es un proyecto ampliamente utilizado y mantenido por la comunidad open-source.

Evolución:

- Soporte de AMQP: Inicialmente, RabbitMQ implementaba el protocolo AMQP 0-9-1.
- Plugins: RabbitMQ permite la extensión de sus funcionalidades mediante plugins, que pueden añadir soporte para otros protocolos como MQTT, STOMP y HTTP.
- Alta Disponibilidad y Clustering: Con el tiempo, RabbitMQ ha mejorado su soporte para la alta disponibilidad y clustering, permitiendo la replicación de colas y la tolerancia a fallos.

RabbitMQ es conocido por su simplicidad de instalación y configuración, y es ampliamente utilizado en sistemas de microservicios para la gestión de colas de mensajes y la coordinación de tareas

III. Casos de estudio

Kafka:

- Uso: LinkedIn utiliza Kafka para su infraestructura de datos en tiempo real, gestionando más de 1.2 billones de mensajes al día. Kafka ayuda a LinkedIn a proporcionar análisis en tiempo real y personalización a sus usuarios.
- Caso de Éxito: Kafka ha permitido a LinkedIn manejar grandes volúmenes de datos en tiempo real, mejorando la experiencia del usuario y la toma de decisiones basada en datos.

- Uso: Uber emplea Kafka para la coincidencia en tiempo real entre pasajeros y conductores, así como para la monitorización de sus sistemas.
- Caso de Éxito: Kafka permite a Uber gestionar la demanda y oferta en tiempo real, optimizando rutas y tiempos de espera para los usuarios.
- Uso: Netflix utiliza Kafka para monitorear eventos y logs en tiempo real, ayudando en la detección de anomalías y la resolución de problemas de manera eficiente.
- Caso de Éxito: Kafka ha mejorado la capacidad de Netflix para detectar y resolver problemas rápidamente, garantizando una experiencia de transmisión ininterrumpida para los usuarios.

RabbitMQ

- Uso: Instagram utiliza RabbitMQ para gestionar la entrega de mensajes en su sistema de chat, garantizando que los mensajes se entreguen de manera confiable y eficiente.
- Caso de Éxito: RabbitMQ ha ayudado a Instagram a escalar su sistema de mensajería, asegurando que los usuarios puedan enviar y recibir mensajes sin interrupciones.

- Uso: Mozilla emplea RabbitMQ para la sincronización de datos en sus servicios de sincronización de Firefox, permitiendo a los usuarios mantener sus datos actualizados en múltiples dispositivos.
- Caso de Éxito: RabbitMQ ha permitido a Mozilla ofrecer un servicio de sincronización rápido y confiable, mejorando la experiencia del usuario.

- Uso: lastminute.com utiliza RabbitMQ para mejorar la escalabilidad de su sistema de búsqueda de soluciones de viaje, integrando microservicios y procesamiento asíncrono.
- Caso de Éxito: RabbitMQ ha mejorado la eficiencia y escalabilidad de lastminute.com, permitiendo manejar aproximadamente 50 millones de solicitudes diarias con un uso optimizado de recursos.

IV. Relación entre los temas

1. **Comunicación Asíncrona y Colas de Mensajes**

Definición: La comunicación asíncrona permite a los sistemas intercambiar mensajes sin esperar respuestas inmediatas, usando colas de mensajes para gestionar la transmisión.
Aplicación: Tanto Kafka como RabbitMQ son herramientas de mensajería asíncrona que utilizan colas para transmitir datos entre productores y consumidores.

2. **Kafka y RabbitMQ**

Historia y Evolución: Ambos sistemas tienen una rica historia de desarrollo y evolución, adaptándose a las necesidades de grandes volúmenes de datos y alta disponibilidad.
Ventajas y Desventajas: Comparar las capacidades de Kafka y RabbitMQ ayuda a entender sus fortalezas y debilidades en diferentes escenarios de uso.
Casos de Uso y Éxito: Empresas como LinkedIn, Uber, Instagram y Mozilla utilizan estas tecnologías para manejar grandes volúmenes de datos y garantizar la entrega fiable de mensajes.

V. Situaciones y/o problemas donde se podrían aplicar los temas

1. **Procesamiento de Datos en Tiempo Real**

Situación:

Una empresa de comercio electrónico quiere monitorear el comportamiento del usuario en su sitio web en tiempo real para personalizar recomendaciones de productos y optimizar la experiencia del usuario.

Solución con Kafka:

Kafka puede recolectar y procesar datos de clics, vistas de productos y búsquedas en tiempo real. Estos datos se pueden usar para alimentar sistemas de recomendación y análisis en tiempo real, mejorando la experiencia del usuario y aumentando las ventas.

Solución con RabbitMQ:

RabbitMQ puede gestionar la comunicación entre microservicios encargados de recolectar datos de usuario y los sistemas de recomendación. RabbitMQ asegura la entrega fiable de mensajes entre estos componentes, permitiendo una integración fluida y eficiente.

2. **Coordinación de Microservicios**

Situación:

Una aplicación basada en microservicios necesita coordinar múltiples servicios que deben interactuar para completar transacciones complejas, como la gestión de pedidos en una tienda en línea.

Solución con Kafka:

Kafka se puede utilizar como un bus de eventos, donde cada microservicio publica y suscribe eventos. Esto permite una coordinación eficiente y la posibilidad de reintentar operaciones fallidas, asegurando la consistencia y durabilidad de los datos.

Solución con RabbitMQ:

RabbitMQ es ideal para la orquestación de microservicios utilizando colas de mensajes. Cada paso en la gestión de pedidos (validación de inventario, procesamiento de pagos, generación de facturas) puede comunicarse de manera asíncrona a través de colas, garantizando la entrega ordenada y fiable de mensajes.

3. **Monitoreo y Alertas de Sistemas**

Situación:

Una empresa de servicios en la nube necesita monitorear el estado y rendimiento de sus servidores y servicios para detectar fallos y generar alertas en tiempo real.

Solución con Kafka:

Kafka puede recolectar métricas de rendimiento y logs de diversos servicios en tiempo real. Estos datos se pueden procesar para detectar anomalías y enviar alertas, permitiendo una respuesta rápida a incidentes y minimizando el tiempo de inactividad.

Solución con RabbitMQ:

RabbitMQ puede gestionar la comunicación entre agentes de monitoreo y el sistema central de alertas. Cada agente envía mensajes de estado a una cola central, desde donde se procesan y generan alertas en caso de fallos o anomalías detectadas.

4. **Integración de Sistemas Heterogéneos**

Situación:

Una empresa de logística necesita integrar varios sistemas heredados y nuevos, incluyendo ERP, CRM, y sistemas de seguimiento de envíos, para proporcionar una visión unificada y en tiempo real de sus operaciones.

Solución con Kafka:

Kafka puede actuar como un hub central de datos, donde todos los sistemas publican y suscriben eventos. Esto facilita la integración de sistemas heterogéneos, permitiendo la transformación y análisis de datos en tiempo real.

Solución con RabbitMQ:

RabbitMQ puede facilitar la integración mediante la comunicación asíncrona entre sistemas. Las colas de mensajes aseguran que los datos se entreguen de manera fiable entre sistemas, incluso en presencia de fallos temporales.

5. **Procesamiento de Pagos y Transacciones Financieras**

Situación:

Un banco necesita procesar pagos y transacciones financieras de manera fiable y en tiempo real, garantizando la integridad y consistencia de los datos.

Solución con Kafka:

Kafka puede manejar el flujo continuo de transacciones financieras, proporcionando almacenamiento duradero y permitiendo la reanudación y reintento en caso de fallos. Su capacidad de manejar grandes volúmenes de datos lo hace ideal para aplicaciones financieras.

Solución con RabbitMQ:

RabbitMQ puede gestionar la orquestación de procesos de pagos, asegurando que cada paso en el proceso de transacción (autorización, liquidación, notificación) se ejecute de manera fiable y ordenada. Su soporte para transacciones distribuidas asegura la consistencia de datos en sistemas complejos.

VI. Ventajas y desventajas

Apache Kafka

Ventajas:

- Alta Throughput: Kafka puede manejar millones de mensajes por segundo, ideal para aplicaciones de alto rendimiento.
- Baja Latencia: Ofrece latencias muy bajas, esencial para aplicaciones en tiempo real.
- Escalabilidad: Escala horizontalmente añadiendo más brokers al cluster.
- Persistencia y Durabilidad: Almacena datos de manera distribuida y duradera, permitiendo retención a largo plazo.
- Alta Disponibilidad: Kafka es altamente disponible, soportando replicación y particionamiento para evitar la pérdida de datos.

Desventajas:

- Complejidad de Configuración: Requiere configuración y gestión compleja, especialmente en grandes clusters.

- Dependencia de Zookeeper: Históricamente ha dependido de Zookeeper para la coordinación.
- Retención de Datos: Puede consumir mucho espacio de almacenamiento debido a la retención de datos.

RabbitMQ

Ventajas:

- Facilidad de Uso: Fácil de instalar y configurar, con una curva de aprendizaje suave.
- Interoperabilidad: Soporta múltiples protocolos de mensajería como AMQP, MQTT y STOMP.
- Plugins y Extensibilidad: Permite la extensión de funcionalidades a través de plugins.
- Alta Disponibilidad y Clustering: Soporta replicación de colas y clustering para alta disponibilidad y tolerancia a fallos.

Desventajas:

- Rendimiento Menor: Menos adecuado para aplicaciones de alto rendimiento comparado con Kafka.
- Escalabilidad: Requiere configuración más compleja para manejar grandes volúmenes de datos.
- Persistencia de Datos: Menos eficiente en la persistencia de datos en comparación con Kafka.

VII. ALCANCE

El proyecto tiene como objetivo principal diseñar, desarrollar e implementar dos sistemas de colas, con su consumidor y productor, para hacer una comparativa de su rendimiento. Sé evaluará la capacidad de enviar y recibir mensajes de ambas colas. Las tecnologías seleccionadas son Apache Kafka y RabbitMQ. Para la facilidad del proyecto, se integrará también Docker Compose para su fácil ejecución y el código de las colas se implementará en Python.

VIII. REQUERIMIENTOS FUNCIONALES

1. Módulo de productores

Este módulo se centra en establecer los parámetros de cada uno de los productores de las colas.

Requerimientos Funcionales:

Kafka Producer

- Enviar mensajes a un tópico específico.
- Serializar mensajes en formato JSON.
- Manejar errores de conexión y reintentar el envío.

RabbitMQ Producer

- Enviar mensajes a una cola específica.
- Serializar mensajes en formato JSON.
- Manejar errores de conexión y reintentar el envío.

2. Módulo de consumidores

Este módulo abarca los parámetros para los consumidores de ambas colas.

Requerimientos Funcionales:

Kafka Consumer

- Recibir mensajes de un tópico específico.
- Deserializar mensajes en formato JSON.
- Calcular y registrar la latencia de cada mensaje.
- Manejar errores de conexión y reintentar la recepción.

RabbitMQ Consumer

- Recibir mensajes de una cola específica.
- Deserializar mensajes en formato JSON.
- Calcular y registrar la latencia de cada mensaje.
- Manejar errores de conexión y reintentar la recepción.

3. Módulo de mensajería

Este módulo abarca los brokers de ambos Kafka y RabbitMQ para su funcionamiento.

Requerimientos funcionales:

Kafka Broker

- Almacenar y gestionar mensajes en tópicos.
- Proporcionar particionamiento y replicación de datos.
- Manejar múltiples productores y consumidores concurrentes.

RabbitMQ Broker

- Almacenar y gestionar mensajes en colas.
- Proporcionar intercambio de mensajes y enrutamiento.
- Manejar múltiples productores y consumidores concurrentes.

4. Módulo de monitoreo y estadísticas

Requerimientos Funcionales:

- Registrar y calcular la latencia promedio, máxima y mínima de los mensajes.
- Proporcionar métricas de rendimiento, como throughput y tasa de entrega de mensajes.
- Generar informes y alertas basados en los datos de rendimiento.

IX. *REQUERIMIENTOS NO FUNCIONALES*

1. **Rendimiento y Tiempo de Respuesta:** Garantizar que las colas proporcionen una baja latencia en el envío y la llegada de mensajes

2. **Alta disponibilidad:** Garantizar que sea posible ejecutar tanto los productores como los consumidores en cualquier momento.

3. **Facilidad de Mantenimiento:** El código de la página debe ser limpio, bien documentado y organizado de manera que permita actualizaciones y mantenimiento eficiente.

4. **Precisión en métricas y estadísticas:** la implementación de cada cola debe contener una serie de métricas y estadísticas clave para su comparación sin perjudicar el rendimiento.

X. RESTRICCIONES

- Se tiene que implementar un sistema de cola con productor y consumidor con Kafka
- Se tiene que implementar un sistema de cola con productor y consumidor con RabbitMQ
- Para el funcionamiento de Kafka, es necesario utilizar Zookeeper.
- Se utilizará Docker Compose para ejecutar los brockers de Kafka y RabbitMQ, pero también el contenedor de Zookeeper.

XI. RIESGOS

- La implementación puede quedar muy poco detallada pues el proyecto es realizado por una sola persona
- Es posible que no se noté una diferencia clara entre ambas colas.
- Docker es una herramienta útil, pero puede generar fallas u algún otro error con sus contenedores.

X. PRINCIPIOS SOLID APLICADOS

- Principio de responsabilidad única: las clases KafkaProducer, RabbitMQProducer, KafkaConsumer, y RabbitMQConsumer tienen responsabilidades únicas y claramente definidas: enviar mensajes y recibir mensajes, respectivamente. Cada clase se encarga únicamente de su propio rol, lo que facilita el mantenimiento y la comprensión del código.

- Principio de inversión de dependencias: la estructura permite que las clases de productores y consumidores puedan ser fácilmente cambiadas o extendidas, lo que podría facilitar la aplicación de este principio en el futuro.

XII. ATRIBUTOS DE CALIDAD

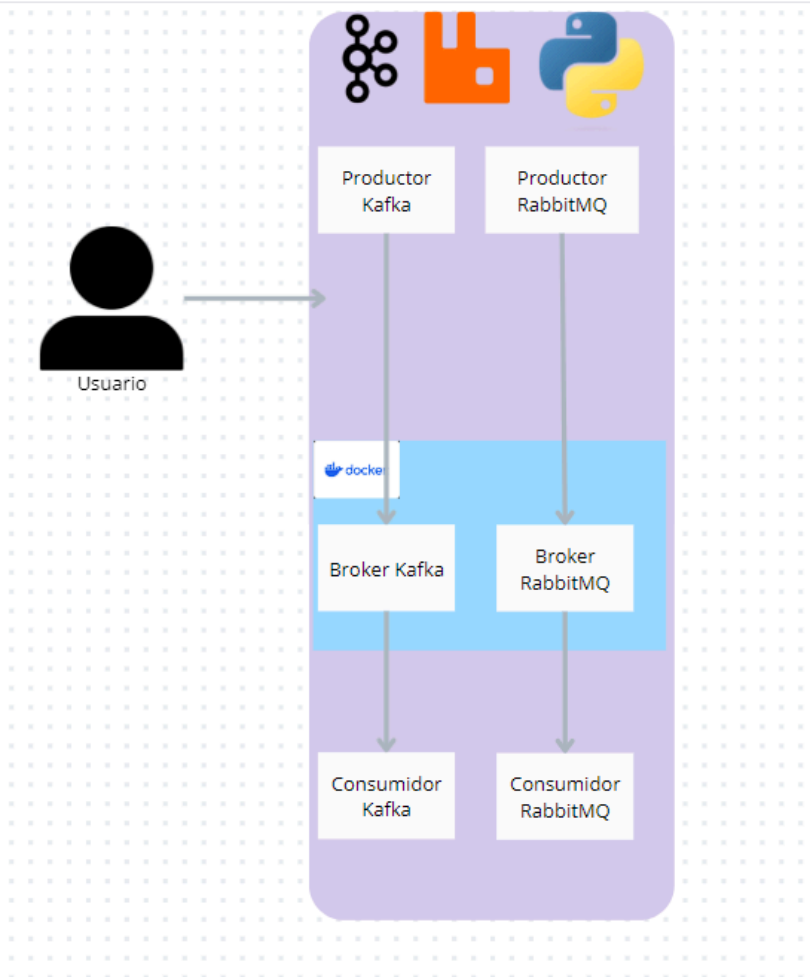
1. **Rendimiento**
2. **Fiabilidad**
3. **Escalabilidad**
4. **Disponibilidad**
5. **Mantenibilidad**

XIII. ESCENARIOS DE CALIDAD

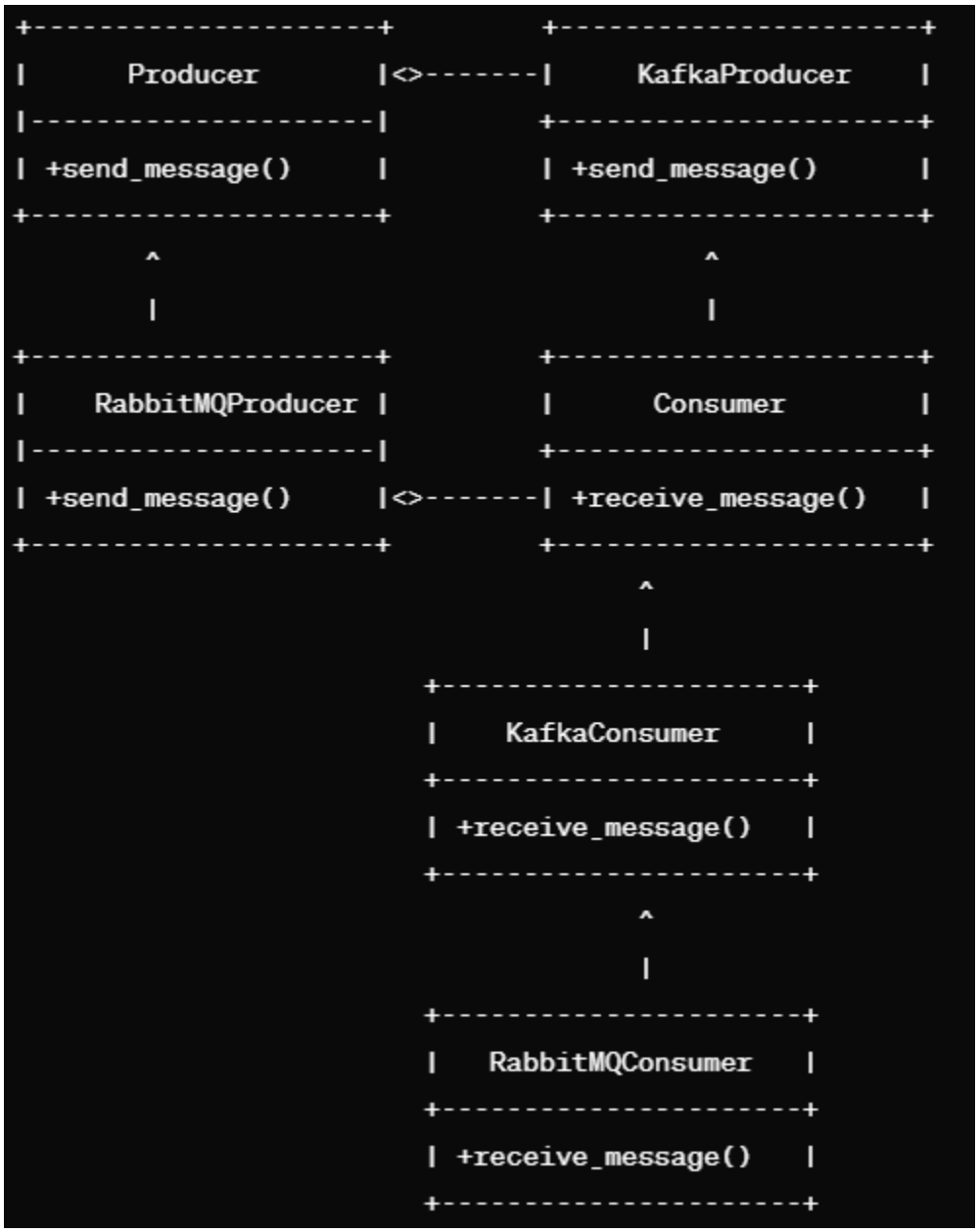
Atributo de Calidad	Escenario de Calidad	Métricas de Rendimiento	Prioridad (Desarrollador)	Prioridad (Product Owner)
Rendimiento	Procesamiento de mensajes	Tiempo promedio de latencia < 100ms	Alta	Alta
		Cantidad de mensajes > 1000 msg/s	Alta	Alta
Fiabilidad	Entrega de mensajes	Tasa de entrega de mensajes > 99.99%	Alta	Alta

	Recuperación ante fallos	**Tiempo de recuperación < 1s**	**Media**	**Alta**
Escalabilidad	**Aumento de carga**	**Manejo de 10x carga sin degradación significativa**	**Alta**	**Media**
Disponibilidad	**Tiempo de actividad**	**Uptime > 99.95%**	**Alta**	**Alta**
Mantenibilidad	**Facilidad de actualización**	**Tiempo de implementación de actualizaciones < 1 hora**	**Media**	**Media**
	Facilidad de diagnóstico	**Tiempo de resolución de problemas < 2 horas**	**Alta**	**Media**

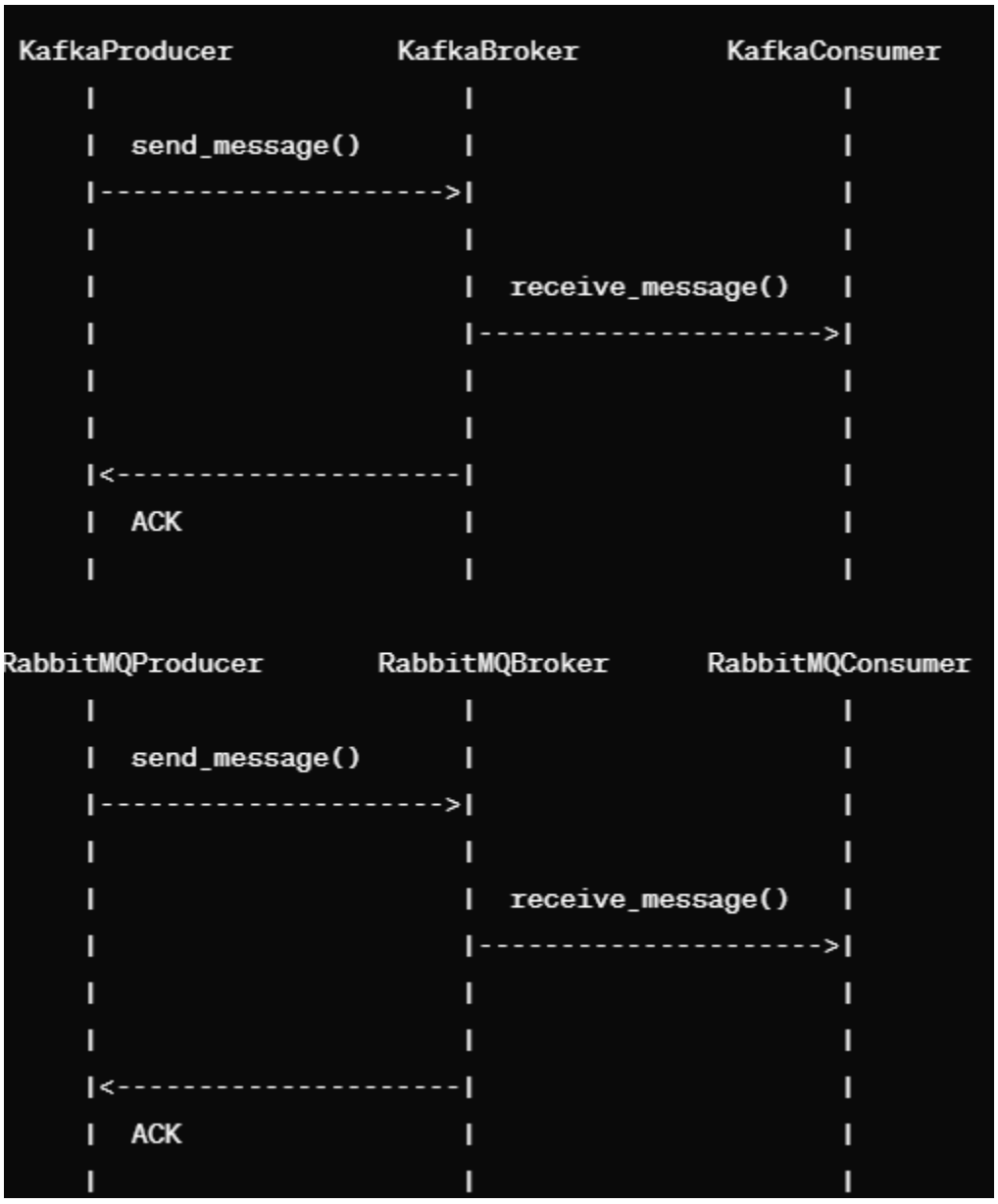
XIV. ARQUITECTURA DE ALTO NIVEL



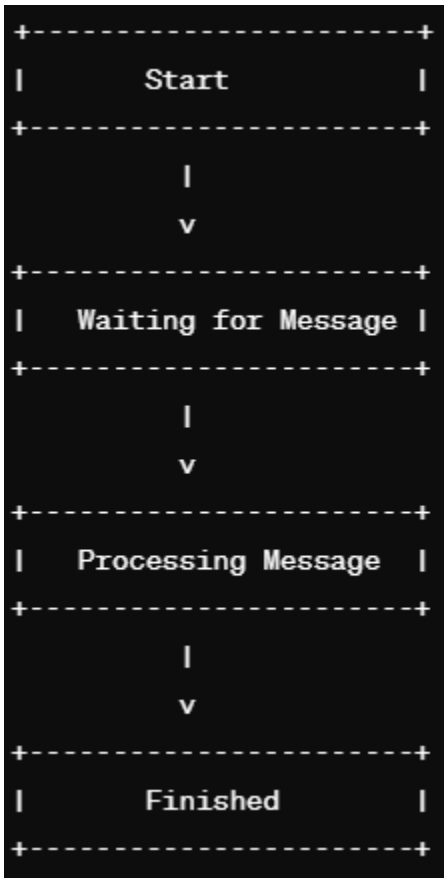
XV. DIAGRAMA DE CLASES



XVI. DIAGRAMA DE SECUENCIA



XVII. DIAGRAMA DE ESTADOS


Start
Waiting for Message
Processing Message
Finished