**TAREA 2: SISTEMAS DISTRIBUIDOS**
**SISTEMA PARA LA GESTIÓN DE PEDIDOS DE UN DELIVERY**
**Profesor:** Nicolás Hidalgo
**Ayudantes:** Cristian Villavicencio, Felipe Ulloa y Marcelo Yañez

# LEA EL DOCUMENTO COMPLETO ANTES DE EMPEZAR A DESARROLLAR LA TAREA

## Objetivo

El objetivo de este trabajo es introducir a los estudiantes a los *sistemas de procesamientos basados en eventos de tipo stream*. Para ello, los alumnos deberán trabajar con **Apache Kafka**. Apache Kafka es una plataforma distribuida para la transmisión de datos , de abierto, desarrollado por LinkedIn y donado a la Apache Software Foundation[1]. Kafka se basa en un modelo Publish-Subscribe con *micro-batching*, sus características han catapultado su uso de manera masiva en la industria. En esta tarea, los estudiantes deberán comprender el funcionamiento y aplicar las funcionalidades de Kafka. Para ello deberán configurar la plataforma, y reconocer y analizar las ventajas de este tipo de tecnologías.

## Conceptos previos

**Apache Kafka** es una plataforma distribuida de transmisión de datos que permite *publicar*, *almacenar* y *procesar* flujos de registros, así como *suscribirse* a ellos, de forma inmediata. Está diseñada para administrar los flujos de datos de varias fuentes y distribuirlos a diversos usuarios [2].

Algunos conceptos claves en Kafka son:

- **Broker**: Corresponde a un servidor de Kafka. Pueden existir varios conectados en una red y son capaces de comunicarse entre ellos utilizando un mecanismo propio. Siempre existe un broker líder; en caso de que este se caiga, otro entra a tomar su lugar.
- **Producer**: Es aquel que produce/publica datos en algún flujo.
- **Topic**: Es un canal donde se publica el flujo de datos. Puede relacionarse con una estructura de datos como lo es una cola.
- **Consumer**: Es aquel que consume los datos publicados en un topic. Sin embargo, los datos al ser consumidos **no** son borrados del topic.
- **Partition**: Es parte del flujo generado en un topic. Podría decirse que son *mini-topics* y pueden repartirse entre distintos brokers. Cada partición puede contener información distinta, sin embargo, se utilizan para generar una jerarquía con la información.
- **Consumer group**: Dado que los datos de un topic de Kafka no son borrados, existe este mecanismo para guardar el *offset* de la lectura de los datos. Es decir, que si un consumer group lee la información del consumer group $X$, Kafka se encargará de guardar la posición del último valor leído, para así no perder el orden. Dos *consumer groups* distintos tendrán distintos *offset*.

Para más detalles, revisar la documentación oficial de apache Kafka: `https://kafka.apache.org/`. Por otro lado, se le recomienda revisar el siguiente repositorio: `https://github.com/Naikelin/async-events-kafka`

Recuerde, parte esencial de las tareas radica en investigar sobre la tecnología a trabajar.

[1] https://es.wikipedia.org/wiki/Apache_Kafka
[2] https://www.redhat.com/es/topics/integration/what-is-apache-kafka

## Problema

Desde la pandemia hemos experimentado un creciente aumento en el uso de plataformas de delivery. Esto ha traído consigo un gran cambio en el comercio y el manejo de los propios pedidos. Comprometidos con los avances tecnologicos y los emprendimientos locales, la reconocida *Universidad Deigo Morales* busca apoyar a un emprendimiento local en la implementación su propio sistema de entregas y recepción de pedidos. Para ello, le solicitó a los estudiantes del curso de sistemas distribuidos el diseño de un sistema de manejo de pedidos, escalable y de alta disponibilidad para dar soporte a dicho emprendimiento. Como alumno del curso, le corresponde a su grupo realizar una arquitectura de prueba utilizando un modelo de comunicación indirecta para poder hacer una demostración de la nueva aplicación y las propiedades de esta.

Conociendo las propiedades y la tendencia en el uso de Kafka, se le solicita modelar una arquitectura de micro-servicios que utilicen como medio de comunicación el modelo pub-sub de Kafka bajo 3 servicios que se detallan a continuación:

- **Servicio de solicitud**: Este se encargará de recibir toda solicitud de pedido mediante una petición HTTP POST, esta solicitud deberá ser procesada y enviada al sistema de mensajería.

- **Servicio de procesamiento**: Este servicio deberá consumir las solicitudes y procesarlas. Para ello se debe configurar un sistema automático sin interacción de usuarios (batch), el cual se ejecuta al recibir una solicitud. Dicha solicitud es procesada cambiando su estado. El modelo considera 4 posibles estados: "recibido", "preparando", "entregando" y "finalizado". Al pasar por cada estado este deberá ser encolado nuevamente, y luego de un tiempo definido (parámetro del sistema) pasar al siguiente hasta llegar al estado "finalizado". Así mismo, este servicio al procesar un pedido, y en cada estado, deberá estar en permanente comunicación con el servicio de notificación para indicar el avance del mismo usando como medio de comunicación la misma cola de mensajería.

- **Servicio de notificación**: Este servicio en particular deberá cumplir 2 funciones, primeramente ir notificando de forma automática utilizando un sistema de correos SMTP simple, y a su vez mostrar el estado de una solicitud en particular mediante una petición HTTP GET.

## Qué hacer:

- Estudiar apache Kafka y sus parámetros de configuración: 1) para el manejo de topics, 2) groups consumers, y 3) particiones.

- Seleccionar dos conjuntos de datos o *datasets* para realizar las pruebas. Estos pueden ser de cualquier fuente, se recomienda un gran volumen de datos para lograr hacer un análisis lo más acabado posible. (No se requiere de una estructura compleja, podría ser simplemente un dataset que caracterice dos valores, por ejemplo, con nombre distintivo y un precio cualquiera).

- Implementar la arquitectura del sistema: Esta deberá poder completar el flujo del sistema, desde la solicitud del producto, pasando por el sistema automático de procesamiento, y finalmente el sistema de notificación, los cuales según corresponda deberá funcionar ya sea de forma manual (para las peticiones), y también la forma automática (procesamiento y notificaciones) Figure 1.

- Deberá modelar y explicar una forma de manejo de las peticiones utilizando diferentes topics, consumer group y partitions con el objetivo de dar solución a el problema.

- Respecto al modelo generado, debe diseñar dos escenarios para probar la robustez de su sistema. Por ejemplo, un test de carga dada una cantidad de pedidos ingresados en un intervalo de tiempo o una gran cantidad de pedidos ingresados en el mismo momento. Estos escenarios deberán ser justificados con métricas como: el tiempo de respuesta, la número de solicitudes procesadas (no perder solicitudes), entre otras.

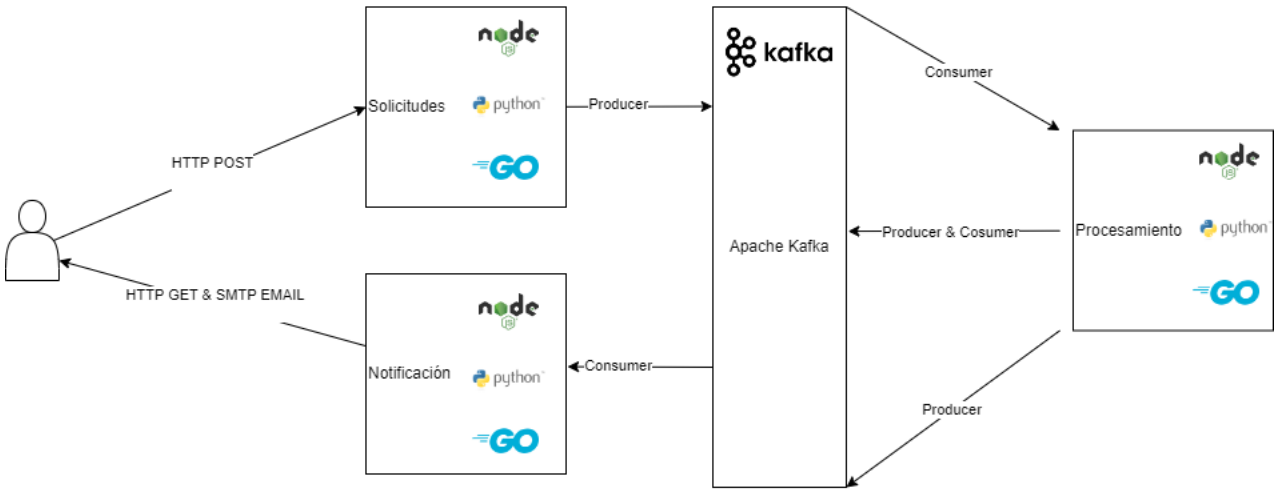


Figure 1: Arquitectura del sistema.

## Ejemplos de procesamiento

Supongamos que nuestro dataset posee una estructur como la presentada a continuación:

```
{
    "nombre": "Macbook air M1",
    "precio": 759.000,
    "correo": "correopersonal@correo.com"
}
```

Se realizará una petición HTTP POST al servicio de Solicitudes, el cual modificará el dato de entrada integrándole un identificador único (ID) resultando:

```
{
    "id": 1,
    "nombre": "Macbook air M1",
    "precio": 759.000,
    "correo": "correopersonal@correo.com"
}
```

Posteriormente, este pasará a ser enviado por la cola para ser consumida por el servicio de procesamiento.

Respecto al servicio de procesamiento este debe leer toda solicitud entregándole información sobre el estado. Inicialmente este estado será “recibido”, obteniendo así la estructura de esta forma:

```
{
    "id": 1,
    "nombre": "Macbook air M1",
    "precio": 759.000,
    "correo": "correopersonal@correo.com",
    "estado": "recibido"
}
```

Una vez realizado esto, se debe volver enviar a la cola de forma automática en un tiempo $\boldsymbol{N}$ (parámetro del sistema), con el fin de reflejar el procesamiento de forma automática. Este volverá a ser leído y actualizará su estado hasta llegar a ser estado “finalizado”.

En paralelo, y en cada actualización, de estado el servicio de notificaciones deberá estar recibiendo también la información de la cola para poder ir enviando información de forma periódica y automática por cada cambio de estado y notificando por correo.

A su vez la información podrá ser consultada mediante una petición HTTP GET al servicio de notificación sólo con el identificador o ID, el cual entregará como respuesta un JSON con la última actualización de la petición, por ejemplo:

```
{
    "id": 1,
    "nombre": "Macbook air M1",
    "precio": 759.000,
    "correo": "correopersonal@correo.com",
    "estado": "finalizado"
}
```

**Cabe destacar que la forma en como se separan cada mensaje, es decir topics, consumer groups o particiones, deberá ser modelada por cada estudiante según su criterio.**

## Entregables:

- **Justificación del modelo de datos**: Un análisis detallado de su modelo de datos en donde explique el dataset a usar y la forma en que configuró Apache Kafka para gestionar sus respectivos topics, consumer groups y particiones.
- **Reporte**: Un documento que compare los escenarios y muestre el comportamiento de ellos. Debe incluir métricas, gráficos , y conclusiones.
- Responder a las siguientes preguntas en la entrega:
    1. Describa en detalle cómo la arquitectura de Apache Kafka permite brindar tolerancia a fallos en el sistema propuesto.
    2. Considerando el sistema propuesto, ¿cómo se aseguraría de que el sistema es capaz de escalar para manejar un aumento significativo en el número de solicitudes? Describa cualquier ajuste de configuración o estrategia de escalado que implementaría.
    3. Defina métricas (latencia, througput, entre otras) y en base a ellas evalúe el rendimiento del sistema bajo diferentes cargas de trabajo. ¿Cómo afecta el aumento de la cantidad de mensajes en los tópicos a la latencia y al throughput del sistema? Presente gráficos o tablas que respalden su respuesta. Se recomienda que se realicen pruebas automatizadas con scripts.
    4. Dada la naturaleza variable de la demanda ¿cómo podría optimizar el uso de recursos del sistema durante los períodos de baja demanda, y cómo prepararía el sistema para los picos de demanda? Explique las estrategias de optimización y *autoscaling* que podría emplear.
    5. Considerando la diversidad de fuentes de datos en el sistema propuesto (Solicitud de producto, gestión de estados de solicitud, notificación de estado), ¿cómo se aseguraría de que los datos se integran y gestionan de manera coherente y unificada en Apache Kafka? Describa los posibles desafíos y soluciones para la integración de datos y la gestión de esquemas en los tópicos, consumer groups y particiones de Kafka.

## Observaciones

Se recomienda crear una cuenta de pruebas en Google, para realizar uso del servicio SMTP y enviar correos de pruebas.

## Aspectos formales de entrega

- **Fecha de entrega:** 31 de Mayo hasta las 23:59 hrs.
- **Número de integrantes:** Grupos de 2 personas las cuales deben estar claramente identificadas en la tarea.
- **Lenguaje de Programación:** No existe limitación. Procure utilizar el que le brinde mayor confianza en cualquier caso.
- **Rúbrica**: `https://docs.google.com/spreadsheets/d/1XxCEbyiXjO19kZH_tZ1DFgsI2v4LQGokTqLxq95qoYM/edit?usp=sharing`
- **Formato de entrega:** Repositorio público (Github o Gitlab), video de funcionamiento e informe en formato PDF.
- **Tecnologías complementarias:** En caso de usar tecnologías complementarias, añadir una descripción en el informe.
- **Contenedores de Kafka**: se recomienda utilizar las siguientes imágenes: `https://hub.docker.com/r/bitnami/kafka/`. En caso de utilizar otra imagen, deberá especificarlo en el informe.
- Las copias de código serán penalizadas con nota mínima. Referencie apropiadamente todo segmento de código que no sea de su autoría.
- No se debe implementar un front o interfaz, solo basta con implementar una API REST o una interfaz interactiva en la terminal.
- Consultas sobre la tarea: **cristian.villavicencio1@mail.udp.cl** y **marcelo.araya1@mail.udp.cl**