TAREA 2: SISTEMAS DISTRIBUIDOS
Colas de Mensajería
Profesor: Nicolás Hidalgo
Ayudantes: Cristián Villavicencio, Joaquín Fernández, Felipe Ulloa y Nicolás Núñez

LEA EL DOCUMENTO COMPLETO ANTES DE EMPEZAR A DESARROLLAR LA TAREA

Objetivo

El propósito de esta actividad es que los estudiantes adquieran conocimientos y experiencia práctica en el uso de tecnologías de brókers de mensajería. Para ello, se les pedirá que desarrollen un sistema que simule dispositivos *IoT* y transmitan la información a un servidor central mediante el uso de dos colas de mensajería distintas: Kafka y RabbitMQ. Además, se espera que los estudiantes realicen un análisis comparativo de ambas tecnologías, identifiquen sus limitaciones y propongan mejoras para optimizar el desempeño del sistema.

Enunciado

Una noche, analizando su propia existencia, te diste cuenta sobre la evolución de la tecnología y la cantidad de información que se comparte día a día. Inmediatamente, al ver las estrellas, recordó que existen dispositivos **IoT** que están constantemente transmitiendo información a través de la red. Entonces se preguntó: ¿Cómo es posible procesar tal cantidad de información sin perder datos en el camino?

Al investigar, descubriste que existen tecnologías diseñadas para manejar grandes volúmenes de información y asegurar la entrega de los datos, como las colas de mensajería. Inspirado por esta idea, decidiste simular dispositivos **IoT** que son capaces de enviar información constantemente.

Además, te diste cuenta de que las colas de mensajería pueden implementarse de diferentes maneras y que la elección de una tecnología específica depende del contexto del problema. Por lo tanto, una vez que hayas desarrollado una forma de simular los dispositivos, deberás comparar el rendimiento de diferentes tecnologías de colas de mensajería, ajustando ciertas variables para analizar su comportamiento en distintos escenarios.

Observaciones

- La tarea se debe de realizar en parejas. Sólo en casos especiales se aceptarán grupos de 3 integrantes considerando una extensión del problema para hacer equivalente la dificultad con el resto de grupos (leer detenidamente las instrucciones).

Instrucciones

1. **Planificación del sistema**: Usted debe inventar un caso donde pueda aplicar el problema. No existen restricciones más allá de los pasos siguientes, por lo que usted es libre para generar un escenario a su imaginación.

2. (**20 pts**) Implemente un sistema que sea capaz de simular n dispositivos **IoT**, donde n es un parámetro en el sistema que usted puede variar a medida que haga pruebas. Estos enviarán información cada cierto tiempo $\boldsymbol{\Delta t}$, un parámetro que usted debe ser capaz de controlar (es decir, es una variable que debe ser capaz de modificar para cada dipositivo), lo que es natural puesto la naturaleza que tienen estos dispositivos. La información deberá ser enviada en formato *JSON* y debe contener:

   - **Timestamp**: un tiempo que marque en que momento se envió la información.
   - **Values**: un valor de información que puede ser aleatorio, pero de tamaño aleatorio. La idea que el tamaño de la información que usted transmita pueda ser variable a partir de un parámetro que usted indique. El fin de esto es que haya información más *pesada* en algunos dispositivos.

   Hasta este punto deberá mostrar el output de cada dispositivo por terminal, dando un *log* de qué dispositivos están transmitiendo información.

Nota: Para la generación de flujos de datos (***Producers***), se recomienda el uso de *hilos*. Cada dispositivo IoT es un hilo, tal como lo hizo en algún momento en *Sistemas Operativos*.

3. (**10 pts**) Implemente un sistema que sea capaz de recibir la información a través de m (***Consumers***), un parámetro que usted debe ser capaz de variar. Este sistema deberá mostrar la información en el orden que los datos son enviados (desde el ***Producer***).
   Un ejemplo

```
Device 0 sending: {"timestamp": 1652224056.2431426, "value": {"data": "m8LPcUxltBnck"}}
Device 1 sending: {"timestamp": 1652224056.2431426, "value": {"data": "5vuxAeS7pY"}}
Device 2 sending: {"timestamp": 1652224056.2431426, "value": {"data": "GKj52dI7HRu9XzvhgQUOBeLb"}}
Device 0 sending: {"timestamp": 1652224061.246106, "value": {"data": "3fKc7OrBwLN"}}
Device 1 sending: {"timestamp": 1652224061.246106, "value": {"data": "o23n"}}
Device 2 sending: {"timestamp": 1652224061.246106, "value": {"data": "9ybFLMxK"}}
Device 0 sending: {"timestamp": 1652224064.248211, "value": {"data": "XjBO5x2zAIvO6"}}
Device 1 sending: {"timestamp": 1652224064.248211, "value": {"data": "KrW1cSv"}},
Device 2 sending: {"timestamp": 1652224064.248211, "value": {"data": "rskf8UEi3qbCn0YmxNVt"}}
```

4. (**70**) Implemente los siguientes sistemas, donde los dispositivos *IoT* deberán insertar información en los sistemas de mensajería de colas:

   - **RabbitMQ**: se le recomienda familiarizarse con este sistema a través de los siguientes links: `https://www.rabbitmq.com/getstarted.html`, `https://hub.docker.com/_/rabbitmq`. De igual manera se le dará una mirada en las ayudantías.
   - **Kafka**: se le recomienda familiarizarse con este sistema a través de los siguientes links: `https://github.com/Naikelin/async-events-kafka`, `https://hub.docker.com/r/bitnami/kafka`. De igual manera se le dará una mirada en las ayudantías.

   Deberá elaborar una comparativa con datos estadísticos de sistema entre:

   (a) **Escalabilidad**: ¿Cómo manejan Kafka y RabbitMQ el crecimiento en el número de productores (dispositivos) y consumidores de datos? Implemente dos mejoras que muestren cambios en la escalabilidad del problema. Debe implementar una mejora para Kafka y una para RabbitMQ. *Hints: Revise los grupos de consumidores en Kafka y patrones de intercambios de mensaje en RabbitMQ.*

   (b) **Tolerancia a fallos**: ¿Qué mecanismos ofrecen Kafka y RabbitMQ para garantizar la disponibilidad y la recuperación ante errores? Entre Kafka y RabbitMQ ¿Cuál entrega mejores herramientas para el manejo de errores? ¿De qué depende? Compare estadísticamente esto.

   (c) **Latencia y rendimiento**: ¿Cómo se comportan Kafka y RabbitMQ en términos de latencia y rendimiento en un escenario de alta carga y tráfico de mensajes? *Hints: aumente la cantidad de mensajes que envían los dispositivos.*

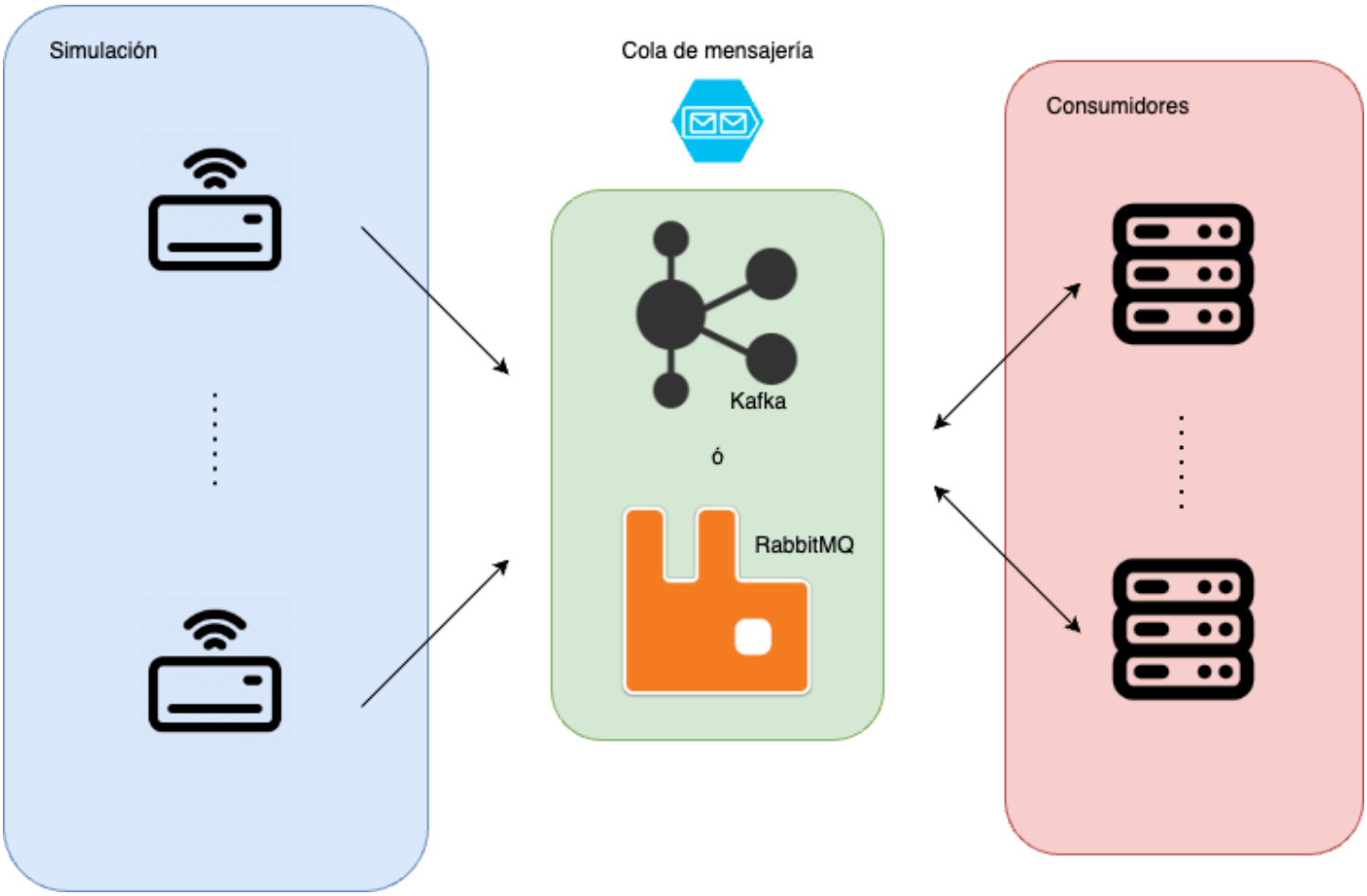
   (d) **Persistencia de mensajes**: ¿Qué opciones ofrecen ambos sistemas para el almacenamiento y recuperación de mensajes?

   (e) **Facilidad de uso y administración**: ¿Cuál de los dos sistemas es más fácil de configurar y administrar?

5. (30 pts) Imagine que los dispositivos **IoT** tienen 5 categorías distintas (asígnelas usted). Cada dispositivo debe enviar información a través de un canal de comunicación distinto, dependiendo de la categoría que se encuentre el dispositivo. ¿Cómo puede enviar esta información teniendo un sólo *broker*? Haga una comparativa de rendimiento entre Kafka y RabbitMQ. *hint: Revise los topics en Kafka y los patrones de intercambio en RabbitMQ.*

6. (40 pts) **Grupo de 3 personas**: En caso de reconocer diferencias entre ambas tecnologías, modifique una de las tecnologías haciendo uso de Redis para generar funcionalidades similares. Muestre sus resultados utilizando tablas o gráficos.

Diagrama



Entrega

Cada estudiante deberá hacer un vídeo que muestre la implementación del sistema, las pruebas de rendimiento realizadas y la evaluación de la escalabilidad; La grabación debe ser explicativa y debe mostrar el proceso de implementación paso a paso. El vídeo debe ser acompañado de un informe escrito que incluya el código fuente (subido a github o gitlab), la documentación y los resultados de las pruebas de rendimiento.

Glosario y definciones relevantes

- ***Kafka:***
  - **Broker**: Corresponde a un servidor en 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 al mando.
  - **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 ordenar 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*. (consume los datos en la cola)

- ***Rabbit MQ:***
  - **Broker**: Servidor responsable de almacenar y enrutar los mensajes a través de diferentes colas (también conocidas como "colas de mensajes") según las reglas de enrutamiento definidas. El broker también se encarga de garantizar la entrega de los mensajes a los consumidores, incluso en caso de fallos o desconexiones temporales.
  - **Cola**: es una estructura de datos que almacena los mensajes en RabbitMQ. Los productores envían mensajes a una cola y los consumidores leen los mensajes de una cola. Las colas pueden ser configuradas con diferentes políticas de encolamiento y desencolamiento.
  - **Producer**: Es aquel que produce/publica datos en algún flujo (envía mensajes a una cola en RabbitMQ).
  - **Consumer**: Es aquel que consume los datos publicados la cola.
  - **Exchange**: es un componente de RabbitMQ que permite el enrutamiento de mensajes a una o varias colas. Los productores envían mensajes a un exchange y el exchange es el encargado de enrutar los mensajes a las colas correspondientes, según la configuración establecida.
  - **Binding**: es la relación entre un exchange y una cola en RabbitMQ. Un binding establece cómo los mensajes enviados a un exchange son enrutados a una cola determinada.

Aspectos formales de entrega

- **Fecha de entrega:** 22 de Mayo 23:59 hrs.
- **Número de integrantes:** Grupos de 2 o 3 personas las cuales deben estar claramente identificadas en la tarea.
- **Pauta de evaluación:** La pauta se puede encontrar en el siguiente link `Enpreparación...`.
- **Lenguaje de Programación:** para la implementación debe escoger entre los siguientes lenguajes: **Python**, **Java**, **Golang** o **JavaScript**.
- **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 RabbitMQ**: `https://hub.docker.com/_/rabbitmq` `https://hub.docker.com/r/bitnami/rabbitmq/`
- **Contenedores de Kafka**: `https://hub.docker.com/r/bitnami/kafka`
- **Contenedores de Redis bitnami y Redis oficial**: `https://hub.docker.com/r/bitnami/redis/`, `https://hub.docker.com/_/redis`. En caso de utilizar otra imagen, deberá especificarlo en el README.
- Las copias de código serán penalizadas con nota mínima. Referente apropiadamente todo segmento de código que no sea de su autoría.
- No es necesario implementar un front o interfaz, solo basta con implementar un **Cliente - Servidor**.
- Consultas: **nicolas.nunez2@mail.udp.cl** o **nk#2258**, **joaquin.fernandez1@mail.udp.cl** o **Jøacø#9352**