Universidad del Valle de Guatemala
Facultad de Ingeniería
Departamento de Ciencias de la Computación
CC3067 Redes

Laboratorio 9

IoT: Estación Meteorológica

1 Objetivos

- Implementar y simular una posible solución comúnmente vista en proyectos de IoT.
- Conocer y utilizar herramientas de software para aplicaciones de IoT y Edge Computing.
- Resolver las necesidades y problemas requeridos por un proyecto en presencia de restricciones fuera de nuestro control.

2 Preámbulo

En los últimos años hemos experimentado un innegable "boom" en la popularidad del internet y las redes, lo cual nos ha obligado a adaptar y modificar muchas de las técnicas, principios y estándares tradicionales… De las más notables y prometedoras son las tecnologías "**Edge**" y el **Internet de las cosas (IoT)**, las cuales juegan un papel importante en el acelerado crecimiento de la información y de generación de contenido. Las tecnologías Edge y el Internet de las Cosas requieren soluciones más escalables, más eficientes, más distribuidas, y orientadas a microservicios.

Una herramienta que cumple con las características mencionadas anteriormente es **Apache Kafka.** Originalmente desarrollada por LinkedIn y luego adoptada por Apache Software Foundation y vuelta Open Source, Kafka es una herramienta de streaming de datos distribuida y escalable que se basa en el Patron Publish/Subscribe (Pub/Sub, similar al "clásico" patrón Observador). Kafka pertenece al mundo de los sistemas e implementaciones MOM (Message-Oriented Middleware), o menos formalmente Message Broker o simplemente Broker. (https://en.wikipedia.org/wiki/Message-oriented_middleware)

Originalmente fue desarrollado e implementado en **Java** y luego en Scala, hoy en dia tiene implementaciones en mas de 10 lenguajes, como C/C++, Node.js, .NET, Lua, Kotlin, Python, y otros mas (https://www.confluent.io/blog/12-programming-languages-walk-into-a-kafka-cluster/).

A continuación unos conceptos básicos de Kafka. Tras bambalinas, se basa en un protocolo TCP de alto rendimiento. Referirse a https://kafka.apache.org/documentation/#introduction para más información:

- **Producer**: Elemento que genera datos y los publica (**Publish**) al Broker.
- **Consumer**: Elemento **suscrito** a mensajes de cierto tipo que provienen de uno o más Producers.
- **Evento**: un suceso o dato de interés en el contexto del problema/solución, el cual se organiza en temas o **Topics.**
- Los topics se pueden **particionar** y distribuir en varios Brokers, para asegurar una alta disponibilidad y tolerancia a fallos.

- En cada partición hay **segmentos** en los que se escribe hasta que se llene y se cree otro. Los mensajes de un topic con el mismo **key** se almacenan en la misma partición en el mismo orden que se escriben.

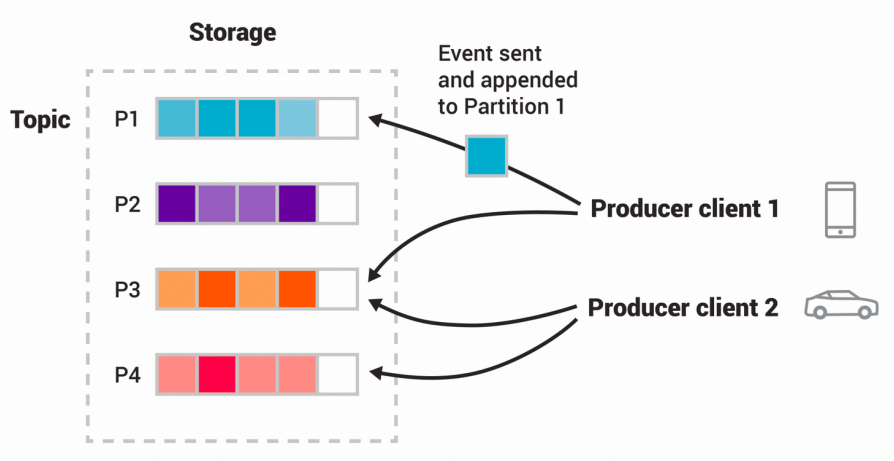

Figura 1: Distintos Producers publicando mensajes en diversas particiones de un topic. (Obtenida de https://kafka.apache.org/documentation/#intro_concepts_and_terms)

3 Desarrollo

El laboratorio será desarrollado en parejas. Toda la evidencia de las fases debe de capturarla y entregarla en un documento PDF. El documento debe incluir explicación de lo que se hizo y capturas de pantalla de los resultados, así como las respuestas a las preguntas que se hagan en el transcurso de la actividad.

Para el laboratorio pueden utilizar cualquier lenguaje de programación, siempre y cuando tenga una librería o API de Kafka, naturalmente. Estaremos implementando y "emulando" un nodo de sensores en una estación meteorológica. Dichos nodos (las "cosas") enviarán su telemetría periódicamente a un servidor Kafka en el borde (Edge), del cual luego se consumen datos para desplegar y graficar.

El paso cero sería instalar y configurar un servidor con Apache Kafka (https://kafka.apache.org/quickstart), el cual tomará el rol de nuestro Edge Server. Esta parte ya se les provee, por lo que no tienen que instalarlo ustedes; nos enfocaremos en el resto del "stack" IoT de esta implementación.

El servidor se encuentra en **iot.redesuvg.cloud**. Se utiliza el puerto estándar de Kafka (**9092**).

3.1 Simulación de un Sensor

Las estaciones meteorológicas poseen una gran variedad de sensores y datos que miden constantemente. Nuestro nodo tendrá tres tipos de sensores, los cuales poseen los siguientes rangos, tipos y resoluciones de datos:

- Sensor de temperatura (Termómetro)
  - Rango: [0, 110.00]**°C**. **Float** de dos decimales.
- Sensor de Humedad Relativa (Higrómetro)
  - Rango: [0, 100]**%**. **Entero**.
- Sensor de dirección del viento.
  - {N, NO, O, SO, S, SE, E, NE}

En implementaciones reales, dichos sensores son componentes electrónicos, usualmente externos o embebidos, que generan datos en base a principios físicos y sus lecturas. En este laboratorio estaremos simulando los datos mediante generadores de números pseudoaleatorios (random). Para hacerlo más realista y que el despliegue se pueda apreciar mejor, los datos de Temperatura y Humedad Relativa deberán ser generados siguiendo una Distribución Uniforme (Gaussiana) en el intervalo de [0,110]. En otras palabras deberán muestrear de tal distribución para generar datos que están "centrados" en la media elegida (grados y %, respectivamente). La media y la varianza serán valores razonables que ustedes consideren. Pueden usar alguna otra distribución si así desean o gustan, siempre y cuando se respete y evidencie que se cumplen los límites indicados anteriormente.

No hay restricción en la forma en que se generen las mediciones de Dirección de Viento; con una uniforme basta, o la que deseen. Se sugiere fuertemente condensar los datos a una representación tipo SOAP o JSON, dependiendo de su preferencia, para darle mayor modularidad y menos fragilidad al código. Por ejemplo, en el caso de JSON se podría ver algo así:

{"temperatura":56.32, "humedad":51, "direccion_viento":"SO"}

- ➢ *Responda: ¿A qué capa pertenece JSON/SOAP según el Modelo OSI y porque?*
- ➢ *Responda: ¿Qué beneficios tiene utilizar un formato como JSON/SOAP?*

3.2 Envío de Datos al Server Edge

Una vez que puedan generar datos es momento de enviarlos a nuestro Kafka Broker. Enviaremos datos entre cada 15 y 30 segundos. Es válido accionar la generación de datos manualmente mientras se prueba y desarrollan las siguientes partes, pero su "Dispositivo IoT" (el rol que cumple el Producer) deberá ser capaz de quedarse corriendo y mandando hasta ser interrumpido. Nuevamente, el servidor se encuentra en **iot.redesuvg.cloud**. Se utiliza el puerto estándar de Kafka (**9092**).

Para enviar datos deberán crear un **Kafka Producer**, el cual enviará datos al susodicho **bootstrap server**. Cada pareja deberá enviar datos a un **topic** único, para no cruzar datos con otras parejas. Para ello, cada pareja puede usar un número de carné de alguno de sus integrantes como topic **.**

Se debe consultar la documentación del lenguaje de su elección para encontrar la sintaxis específica para ello. Un esqueleto del programa en pseudocódigo puede ser:

```
//importar modulos de kafka
//importar elementos para random, y demas
public static void main(String[]: args){
    //inicializar random seeds, etc
    //crear topic, instanciar demas objetos necesarios
    KafkaProducer producer = new KafkaProducer(server='<host_or_ip>:9092');
    while(corriendo){
        JSONObject data = generarData().toJSON();//{'temperatura':50.1, …}
        producer.send(topic='12345',key='sensor1',…, value=data.toString();
    }
}
```

3.3 Consumir y Desplegar Datos Meteorológicos

Una vez se genere data y se esté publicando en Topics de nuestro Kafka Broker es momento de consumirlos y darles utilidad. Para ello usaremos un **Kafka Consumer**, el cual se suscribe a un Topic y escuchara por nuevos mensajes entrantes. El Consumer juega el rol de los elementos que están detrás del Edge, posiblemente en el core o en algún otro componente Edge, o bien alguna herramienta externa. Un pseudocódigo ejemplo del Consumer es el siguiente:

```
from kafka import KafkaConsumer
from kafka import …
#import random y demas utilidades, modulos, etc.

consumer = KafkaConsumer('12345',group_id='foo2', …, bootstrap_server='...')

for mensaje in consumer:
       print(mensaje)
       payload = procesarMensaje(mensaje)
       all_temp.append(payload['temperatura'])
       all_hume.append(payload['humedad'])
       all_wind.append(payload['direccion_viento'])

       #graficar, plotear, analizar, etc.
       plotAllData(all_temp, all_hume, all_wind)
```

Se sugiere explorar sobre los distintos parámetros del KafkaConsumer y el KafkaServer. Mediante se vaya recibiendo data, se debe ir **graficando** la telemetría pasada y entrante en vivo, actualizando el gráfico cada vez que entre un nuevo dato (que será aprox cada 15-30 segundos). Puede apalancarse de cualquier librería para graficar (matplot, chart.js, etc).

- ➢ *Responda: ¿Qué ventajas y desventajas considera que tiene este acercamiento basado en Pub/Sub de Kafka?*
- ➢ *Responda: ¿Para qué aplicaciones tiene sentido usar Kafka? ¿Para cuáles no?*

3.4 IoT en Entornos con Restricciones

En algunos entornos más complicados o remotos, como es el caso de nuestra estación, surgen diversos retos y restricciones que determinan fuertemente el diseño de nuestro protocolo/estructura de mensaje. En redes IoT como LoRa y Sigfox una de esas limitaciones es el tamaño de la Carga Útil de un paquete enviado (payload).

Modifique su código para adaptarlo a la siguiente restricción: el tamaño máximo de mensaje (payload) a enviar es de **3 bytes (24 bits)**. Recordemos que un Char pesa un byte, por lo que nuestro mensaje debe caber dentro de 3 caracteres (“aF!”, “=~Z”, etc…). Debemos entonces **codificar** y **decodificar** nuestros mensajes antes de enviarlos y luego de recibirlos.

Debe evidenciar en su documento que se logró lo mismo de los pasos anteriores (envio, consumo, despliegue gráfico) pero ahora con la restricción de payload .

Como podrán ver, ahora “cada bit cuenta”. A continuación algunos tips muy importantes:

- Implementar una función Encode (JSON to Bytes) y una Decode (Bytes to JSON) para el propósito.
- La temperatura en punto flotante es un gran problema. Solo el tipo de dato float ocupa 4 bytes (!), por lo que hay que usar nuestro ingenio para lograr que quepa…
- La humedad es un dato entero, entre 0-100, por lo que cabría en 7 bits mínimo.
- La dirección del viento es un valor categórico con 8 posibles opciones. Cabe en 3 bits mínimo.
- De los 24 bits nos quedan ahora 14… aca viene la pista importante:
    - Cuanto es 2^14?
    - Observe detenidamente el rango de valores posibles de temperatura, especialmente el máximo valor posible.
    - Vuelva a observar ese valor máximo detenidamente y compárelo con los 14 bits que nos quedan. ¿Cómo podemos hacer que entre ahí?

➢ *Responda: ¿Qué complejidades introduce el tener un payload restringido (pequeño)?*
➢ *Responda: ¿Cómo podemos hacer que el valor de temperatura quepa en 14 bits?*
➢ *Responda: ¿Qué sucedería si ahora la humedad también es tipo float con un decimal? ¿Qué decisiones tendríamos que tomar en ese caso?*
➢ *Responda: ¿Qué parámetros o herramientas de Kafka podrían ayudarnos si las restricciones fueran aún más fuertes?*

4 Entregar en Canvas

- La presentación con la evidencia de las fases, explicaciones, capturas de pantalla y respuesta a las preguntas, en formato PDF.
- Todo el código implementado en el Laboratorio.