

Ejercicio Práctico: Pipeline de Procesamiento y Agregación de Métricas de Sistema con Python, Kafka y MongoDB Atlas

Curso: Big Data Aplicado

Contexto:

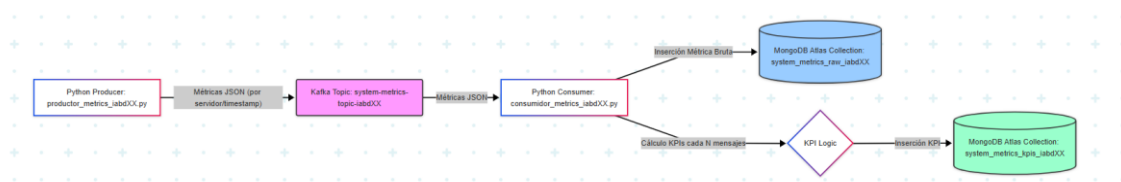
Vamos a simular la monitorización de una infraestructura simple. Varios "servidores" (simulados por nuestro productor) reportan periódicamente sus métricas de rendimiento (CPU, memoria, disco, red) a un sistema centralizado a través de Apache Kafka. Un servicio consumidor recogerá estas métricas, las almacenará en bruto para auditoría o análisis detallado, y además calculará KPIs agregados (promedios, tasas) sobre el estado general del sistema, guardando estos indicadores clave en MongoDB Atlas para una visualización y alerta rápidas.

Objetivo Principal:

Implementar un productor de Kafka en Python que simule el envío de métricas de rendimiento de varios servidores. Luego, implementar un consumidor en Python que:

1. Reciba estas métricas desde Kafka.
2. Almacene las métricas *brutas* de cada servidor/timestamp en una colección de MongoDB Atlas.
3. Calcule periódicamente KPIs agregados (como uso promedio de CPU/Memoria, rendimiento de red/disco, tasa de recepción de métricas).
4. Almacene estos KPIs calculados en una *segunda* colección de MongoDB Atlas.

Arquitectura a Implementar:



Entorno y Herramientas Proporcionadas:

1. **Kafka:** Se te proporciona un archivo docker-compose.yml. Al ejecutar docker-compose up -d, tendrás un servicio de Kafka disponible. El broker principal estará accesible para tus scripts Python en: IPlocal:29092.
2. **MongoDB Atlas:** Tienes una cuenta personal de MongoDB Atlas, con una cadena de conexión (CONNECTION_STRING) para acceder a una base de datos en MongoDB Atlas. Deberás usar esta cadena para conectarte desde tu script consumidor.

Tareas a Realizar:

Parte A: Productor de Métricas de Sistema (productor_metrics_iabdXX.py)

1. Configuración Inicial:

- Crea un nuevo script Python llamado productor_metrics_iabdXX.py.
- Instala las librerías necesarias.
- Define una lista de IDs de servidores simulados: SERVER_IDS = ["web01", "web02", "db01", "app01", "cache01"].

2. Generación de Métricas Simuladas:

- En un bucle infinito (while True):
 - Itera sobre cada server_id en SERVER_IDS.
 - Para cada servidor, genera un conjunto de métricas simuladas con valores que fluctúen realísticamente:
 - cpu_percent: Uso de CPU (ej. random.uniform(5.0, 75.0)).
 - memory_percent: Uso de Memoria (ej. random.uniform(20.0, 85.0)).
 - disk_io_mbps: Lectura/Escritura de disco en MB/s (ej. random.uniform(0.1, 50.0)).
 - network_mbps: Tráfico de red en Mbps (ej. random.uniform(1.0, 100.0)).
 - error_count: Contador de errores simulado (ej. random.randint(0, 2)).

- Crea un diccionario para el mensaje de métricas:

```
metric_message = {
    "server_id": server_id,
    "timestamp_utc": datetime.utcnow().isoformat(),
    "metrics": {
        "cpu_percent": round(cpu_percent, 2),
        "memory_percent": round(memory_percent, 2),
        "disk_io_mbps": round(disk_io_mbps, 2),
        "network_mbps": round(network_mbps, 2),
        "error_count": error_count
    },
    "message_uuid": str(uuid.uuid4()) #Para identificar univocamente el evento
}
```

Ejemplo de posible código:

```
import random
import time
from datetime import datetime, timezone
import uuid
import json # Añadido para poder imprimir el JSON si se desea

# --- Configuración ---
SERVER_IDS = ["web01", "web02", "db01", "app01", "cache01"]
REPORTING_INTERVAL_SECONDS = 10 # Tiempo entre reportes completos de todos los servers

# --- Inicio de la Lógica de Generación (Parte A, Paso 2) ---

if __name__ == "__main__":
    print("Iniciando simulación de generación de métricas...")
    print(f"Servidores simulados: {SERVER_IDS}")
```

```

print(f"Intervalo de reporte: {REPORTING_INTERVAL_SECONDS} segundos")
print("-" * 30)

try:
    while True:
        print(f"\n(datetime.now()): Generando reporte de métricas...")

        # Iterar sobre cada servidor para generar sus métricas
        for server_id in SERVER_IDS:

            # Generar métricas simuladas con fluctuaciones
            cpu_percent = random.uniform(5.0, 75.0)
            # Añadir un pico ocasional de CPU
            if random.random() < 0.1: # 10% de probabilidad
                cpu_percent = random.uniform(85.0, 98.0)

            memory_percent = random.uniform(20.0, 85.0)
            # Añadir un pico ocasional de memoria
            if random.random() < 0.05: # 5% de probabilidad
                memory_percent = random.uniform(90.0, 99.0)

            disk_io_mbps = random.uniform(0.1, 50.0)
            network_mbps = random.uniform(1.0, 100.0)
            # Errores deben ser poco frecuentes
            error_count = 0
            if random.random() < 0.08: # 8% probabilidad de tener algún error
                error_count = random.randint(1, 3)

            # Crear el diccionario del mensaje de métricas
            metric_message = {
                "server_id": server_id,
                "timestamp_utc": datetime.now(timezone.utc).isoformat(), # Usar timezone.utc
                "metrics": {
                    "cpu_percent": round(cpu_percent, 2),
                    "memory_percent": round(memory_percent, 2),
                    "disk_io_mbps": round(disk_io_mbps, 2),
                    "network_mbps": round(network_mbps, 2),
                    "error_count": error_count
                },
                "message_uuid": str(uuid.uuid4()) # Identificador único del mensaje
            }

            # --- Punto de Integración ---
            # Aquí es donde, en el script completo, enviarías `metric_message` a Kafka.
            # Por ahora, solo lo imprimimos para ver qué se genera.
            print(f" Generado para {server_id}:")
            # Imprimir de forma legible (opcional)
            print(f" CPU: {metric_message['metrics']['cpu_percent']}%")
            print(f" Mem: {metric_message['metrics']['memory_percent']}%")
            print(f" Disk: {metric_message['metrics']['disk_io_mbps']} MB/s")
            print(f" Net: {metric_message['metrics']['network_mbps']} Mbps")
            print(f" Errors: {metric_message['metrics']['error_count']}")
            # O imprimir el JSON completo (más útil para ver la estructura final)
            # print(json.dumps(metric_message, indent=2))
            # print("-" * 10)

            # Esperar antes de generar el siguiente reporte completo
            print(f"\nReporte completo generado. Esperando {REPORTING_INTERVAL_SECONDS} segundos...")
            time.sleep(REPORTING_INTERVAL_SECONDS)

        except KeyboardInterrupt:
            print("\nSimulación detenida por el usuario.")

# --- Fin de la Lógica de Generación ---

```

3. Conexión y Envío a Kafka:



- Establece conexión con Kafka en la ip:puerto adecuados (kafka-python).
- Configura serializador de valor a JSON.
- **Dentro del bucle while True y el bucle for server_id...:**
 - Envía el metric_message como **un único mensaje** al topic de Kafka denominado system-metrics-topic-iabdXX.



- Imprime un mensaje en consola indicando qué métrica de qué servidor se envió (ej: `print(f"Enviada métrica de {server_id} a Kafka.")`).
 - **Fuera del bucle for server_id... pero dentro del while True:**
 - Introduce una pausa después de enviar las métricas de todos los servidores (ej. `time.sleep(10)` para enviar un reporte completo cada 10 segundos).
4. **Manejo de Errores (Básico):** Incluye `try...except` para conexión y envío, y `finally` para cerrar el productor.

Parte B: Consumidor, Inserción en MongoDB y Cálculo de KPIs (consumidor_metrics_iabdXX.py)

- Consumir mensajes del topic system-metrics-topic-iabdXX utilizando un group_id único "grupo_iabdXX_id".
- Conectar a MongoDB Atlas usando la CONNECTION_STRING.
- Para cada mensaje recibido:
 - Almacenar el documento JSON completo (métrica bruta) en la colección system_metrics_raw_iabdXX.
- Implementar lógica para calcular KPIs agregados cada N=20 mensajes recibidos: al completar 20 registros se calculan KPIs y se reinicia el contador
 - **KPIs Requeridos:** Calcular para la ventana de N mensajes:
 - Promedio de cpu_percent.
 - Promedio de memory_percent.
 - Promedio de disk_io_mbps.
 - Promedio de network_mbps.
 - Suma de error_count.
 - Tasa de procesamiento (mensajes/segundo).
 - Almacenar los KPIs calculados como un **único documento** por ventana en la colección system_metrics_kpis_iabdXX. Este documento debe incluir:
 - Timestamp del cálculo.
 - Duración de la ventana.
 - Número de mensajes en la ventana.
 - Los KPIs calculados.
- Implementar manejo básico de errores y cierre adecuado de recursos (Kafka, MongoDB).

Entregables:

1. Código fuente completo de los scripts productor_metrics_iabdXX.py y consumidor_metrics_iabdXX.py.
2. Capturas de pantalla que evidencien:
 - La ejecución del productor enviando datos. 
 - La ejecución del consumidor recibiendo datos, insertando en system_metrics_raw_iabdXX y calculando/insertando KPIs en system_metrics_kpis_iabdXX. 

- Contenido de ejemplo en la colección `system_metrics_raw_iabdXX` en MongoDB Atlas. 
- Contenido de ejemplo en la colección `system_metrics_kpis_iabdXX` en MongoDB Atlas. 

Criterio	Descripción detallada	Puntos
A. Productor (Kafka)	<ul style="list-style-type: none"> - Produce mensajes con los campos requeridos. - Configura <code>value_serializer</code> a JSON. - Intervalo y IDs de servidor correctos. - Manejo de errores y cierre ordenado. 	2,50
B. Consumidor (Kafka → MongoDB Atlas)	<ul style="list-style-type: none"> - Se conecta con el <code>group_id</code> indicado. - Inserta cada mensaje en <code>system_metrics_raw_iabdXX</code> sin pérdidas. - Maneja reconexión y commit de offset. 	2,50
C. Cálculo y almacenamiento de KPIs	<ul style="list-style-type: none"> - Implementa ventana tumbling de 20 mensajes. - Calcula correctamente promedios, suma de errores y tasa de mensajes/seg. - Inserta documento resumen en <code>system_metrics_kpis_iabdXX</code> con metadatos de la ventana. 	2,00
D. Calidad del código	<ul style="list-style-type: none"> - Uso de funciones/modularidad. - Nombre de variables coherente y en inglés o español consistente. - Comentarios claros, requisitos en un README, uso de <code>.env</code>. 	1,0
E. Registro y logs	<ul style="list-style-type: none"> - Logging legible (nivel INFO/ERROR). - Mensajes informan de envío, recepción y KPI generados. 	0,50
F. Entregables y evidencias	<ul style="list-style-type: none"> - Scripts con nombre correcto. - Capturas claras (productor, consumidor y ambas colecciones). 	0,50
G. Buenas prácticas extra (bonus)	<ul style="list-style-type: none"> - Dockerizar los scripts, uso de variables de entorno, manejo de SIGTERM, documentación Markdown adicional, tests básicos. 	1,00
TOTAL		10,00