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

**Nombre: Priscila Sabino Miranda**

**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:**

Gráfico

El contenido generado por IA puede ser incorrecto.

**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 realistamente:
       - 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 unívocamente el evento

}

Ejemplo de posible código:

importrandom

import time

fromdatetimeimportdatetime, timezone

importuuid

importjson # 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

forserver\_id in SERVER\_IDS:

# Generar métricas simuladas con fluctuaciones

cpu\_percent = random.uniform(5.0, 75.0)

# Añadir un pico ocasional de CPU

ifrandom.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

ifrandom.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

ifrandom.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)

exceptKeyboardInterrupt:

print("\nSimulación detenida por el usuario.")

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

1. **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 forserver\_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 servidorse envió (ej: print(f"Enviada métrica de {server\_id} a Kafka.")).
   * **Fuera del bucle forserver\_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).
2. **Manejo de Errores (Básico):** Incluye try...except para conexión y envío, y finally para cerrar el productor.

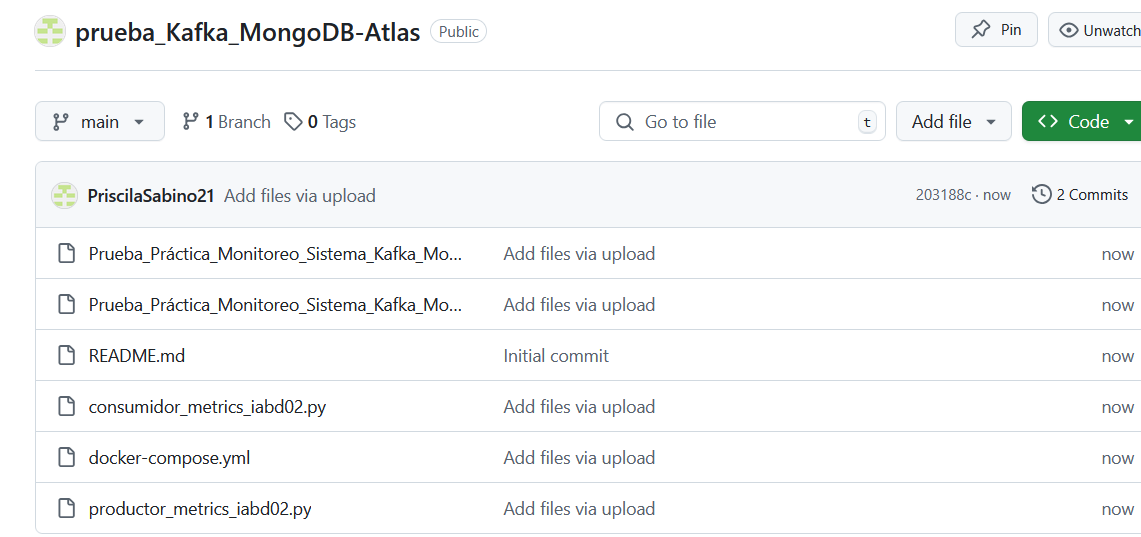
**Parte B: Consumidor, Inserción en MongoDB y Cálculo de KPIs (consumidor\_metrics\_iabdXX.py)**

* 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).

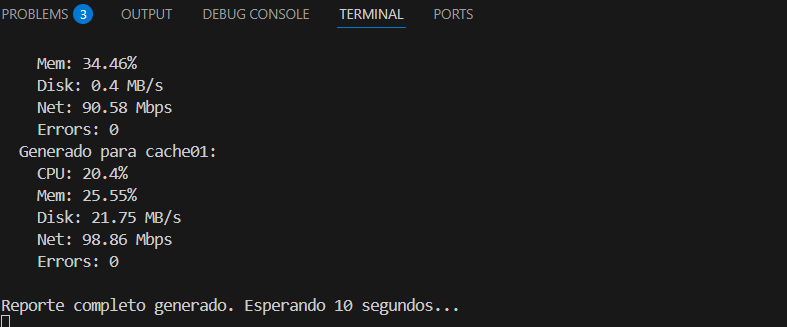
**Entregables:**

1. Código fuente completo de los scripts productor\_metrics\_iabdXX.py y consumidor\_metrics\_iabdXX.py.

<https://github.com/PriscilaSabino21/prueba_Kafka_MongoDB-Atlas/tree/main>

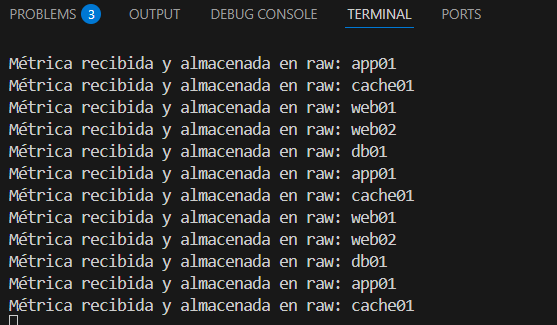


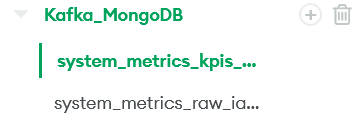
1. Capturas de pantalla que evidencien:
   * La ejecución del productor enviando datos.



* + 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)** | ‑ Produce mensajes con los campos requeridos.  ‑ Configura value\_serializer a JSON.  ‑ Intervalo y IDs de servidor correctos.  ‑ Manejo de errores y cierre ordenado. | **2,50** |
| **B. Consumidor (Kafka → MongoDB Atlas)** | ‑ Se conecta con el group\_id indicado.  ‑ Inserta cada mensaje en **system\_metrics\_raw\_iabdXX** sin pérdidas.  ‑ Maneja reconexión y commit de offset. | **2,50** |
| **C. Cálculo y almacenamiento de KPIs** | ‑ Implementa ventana tumbling de 20 mensajes.  ‑ Calcula correctamente promedios, suma de errores y tasa de mensajes/seg.  ‑ Inserta documento resumen en **system\_metrics\_kpis\_iabdXX** con metadatos de la ventana. | **2,00** |
| **D. Calidad del código** | ‑ Uso de funciones/modularidad.  ‑ Nombre de variables coherente y en inglés o español consistente.  ‑ Comentarios claros, requisitos en un README, uso de .env. | **1,0** |
| **E. Registro y logs** | ‑Logging legible (nivel INFO/ERROR).  ‑ Mensajes informan de envío, recepción y KPI generados. | **0,50** |
| **F. Entregables y evidencias** | ‑ Scripts con nombre correcto.  ‑ Capturas claras (productor, consumidor y ambas colecciones). | **0,50** |
| **G. Buenas prácticas extra (bonus)** | ‑Dockerizar los scripts, uso de variables de entorno, manejo de SIGTERM, documentación Markdown adicional, tests básicos. | **1,00** |
| **TOTAL** |  | **10,00** |