DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
CARRERA DE INGENIERÍA EN SOFTWARE
APLICACIONES DISTRIBUIDAS

Evaluación Conjunta

SISTEMA DE GESTIÓN INTELIGENTE PARA REDES DE MONITOREO AMBIENTAL DISTRIBUIDO

A. Contexto

Una red de sensores ambientales distribuidos en una región montañosa utiliza microservicios para monitorear variables como temperatura, humedad, nivel de contaminantes y movimiento sísmico. Estos datos se recopilan, procesan y analizan en tiempo real para predecir posibles desastres naturales o cambios climáticos significativos.

Sin embargo, debido a la ubicación remota de muchos de estos sensores, es común que haya interrupciones temporales en el servicio de comunicación entre los dispositivos y el sistema central. Por ello, se requiere un sistema escalable, tolerante a fallos, con persistencia garantizada de eventos y alta disponibilidad.

B. Objetivo

Desarrollar una arquitectura de microservicios basada en Spring Cloud y Spring Boot que cumpla con los siguientes requisitos:

- Utilizar un API Gateway para la gestión de rutas y autenticación.
- Implementar un Eureka Server para descubrimiento de servicios.
- Comunicación asíncrona entre microservicios mediante RabbitMQ.
- Persistencia distribuida utilizando CockroachDB, creando nodos replicados en diferentes zonas geográficas.
- Manejo de eventos globales: cuando un evento ocurre en cualquier microservicio, debe notificarse a todos los demás.
- Resiliencia ante caídas de colas: si RabbitMQ se cae temporalmente, los mensajes deben quedar encolados localmente y enviarse automáticamente al restablecerse el servicio.
- Incorporar tareas programadas que realicen análisis periódico de datos y alertas automáticas.

C. Requisitos técnicos

C.1. Arquitectura General

- Implementar 3 microservicios con propósitos claramente definidos:

    1. **SensorDataCollector:** Recibe datos brutos de sensores y los almacena.
    2. **EnvironmentalAnalyzer:** Analiza datos y genera alertas basadas en umbrales.

3. **NotificationDispatcher:** Envía notificaciones a usuarios y sistemas externos.

- Uso obligatorio de:
    - Spring Cloud Gateway como API Gateway.
    - Eureka Server para descubrimiento de servicios.
    - RabbitMQ para comunicación asíncrona.
    - CockroachDB con nodos replicados en al menos 2 zonas geográficas distintas.

C.2. Comunicación y Eventos

- Todos los microservicios deben suscribirse a eventos globales emitidos por cualquier componente. Ejemplos:
    - NewSensorReadingEvent: Nuevo dato de sensor.
    - HighTemperatureAlert: Temperatura > 40°C.
    - SeismicActivityDetected: Actividad sísmica detectada.
    - SystemOfflineEvent: RabbitMQ caído.
    - SystemBackOnlineEvent: RabbitMQ restaurado.
- Los eventos deben ser idempotentes y persistidos en CockroachDB para auditoría.

C.3. Resiliencia ante Fallos

- Simular caídas de RabbitMQ:
    - Si RabbitMQ falla, los microservicios deben almacenar eventos localmente (en memoria o en una base de datos temporal como SQLite).
    - Al restablecerse el servicio, los mensajes pendientes deben ser reenviados automáticamente y eliminados del almacenamiento temporal.
- Validar que no se pierdan eventos durante la caída.

C.4. Tareas Programadas

Implementar las siguientes tareas usando Spring Scheduler o Quartz:

1. **Generación de reportes diarios:** Cada 24 horas, el EnvironmentalAnalyzer debe generar un resumen de tendencias y emitir DailyReportGenerated.
2. **Verificación de sensores inactivos:** Cada 6 horas, detectar sensores sin datos en 24 horas y emitir SensorInactiveAlert.

3. **Limpieza de datos históricos:** Semanalmente, archivar datos antiguos (>6 meses) en CockroachDB.

C.5. Configuración de CockroachDB

- Crear múltiples nodos en diferentes zonas geográficas (simuladas con Docker Compose o máquinas virtuales).
- Configurar replicación automática y tolerancia a fallos (zona primaria/secundaria).
- Validar que los datos se sincronicen correctamente entre nodos.

D. Requisitos Detallados por Microservicio

a. Microservicio: SensorDataCollector

Propósito: Recibir y almacenar datos brutos de sensores ambientales (temperatura, humedad, etc.).

Requisitos Técnicos

- **Endpoints RESTful:**
  - POST /sensor-readings: Recibe datos de sensores en formato JSON.

```
{
  "sensorId": "S001",
  "type": "temperature",
  "value": 38.5,
  "timestamp": "2024-04-05T12:00:00Z"
}
```

  - GET /sensor-readings/{sensorId}: Devuelve historial de lecturas de un sensor.
- **Persistencia:**
  - Almacenar datos en CockroachDB en la tabla sensor_readings.
    - Campos sugeridos: id, sensor_id, type, value, timestamp.
  - Replicación geográfica: Datos deben estar disponibles en todos los nodos de CockroachDB.
- **Eventos:**
  - Emitir evento global NewSensorReadingEvent cada vez que se recibe una lectura.

- Ejemplo de evento:

```
{
  "eventId": "EVT-001",
  "sensorId": "S001",
  "type": "temperature",
  "value": 38.5,
  "timestamp": "2024-04-05T12:00:00Z"
}
```

- **Resiliencia:**
  - Si RabbitMQ está caído, almacenar eventos localmente (en memoria o SQLite).
  - Reintentar envío al restablecer conexión, hasta 3 veces con retroceso exponencial.
- **Validaciones:**
  - Verificar formato de datos entrantes.
  - Rechazar lecturas con valores fuera de rango (ej.: temperatura > 60°C).

b. Microservicio: EnvironmentalAnalyzer

Propósito: Analizar datos de sensores y generar alertas basadas en umbrales.

Requisitos Técnicos

- Lógica de Análisis:
  - Escuchar eventos NewSensorReadingEvent desde RabbitMQ.
  - Aplicar reglas de negocio para detectar anomalías:
    - Temperatura alta: > 40°C.
    - Humedad baja: < 20%.
    - Actividad sísmica: Valor > 3.0 en escala Richter.
- Eventos Generados:
  - Emitir eventos como HighTemperatureAlert, LowHumidityWarning, o SeismicActivityDetected.
  - Ejemplo de evento:

```
{
 "alertId": "ALT-001",
 "type": "HighTemperatureAlert",
 "sensorId": "S001",
 "value": 42.0,
 "threshold": 40.0,
 "timestamp": "2024-04-05T12:05:00Z"
}
```


- Tareas Programadas:
    - Generación de reportes diarios:
        - Cada 24 horas, calcular promedios, máximos y mínimos por tipo de sensor.
        - Emitir evento DailyReportGenerated.
    - Verificación de sensores inactivos:
        - Cada 6 horas, identificar sensores sin lecturas en las últimas 24 horas.
        - Emitir evento SensorInactiveAlert.
- Persistencia:
    - Almacenar alertas en CockroachDB en la tabla alerts.
        - Campos sugeridos: alert_id, type, sensor_id, value, threshold, timestamp.

c. Microservicio: NotificationDispatcher

Propósito: Enviar notificaciones a usuarios o sistemas externos basadas en eventos críticos.

Requisitos Técnicos

- Escucha de Eventos:
    - Suscribirse a todos los eventos globales (HighTemperatureAlert, SeismicActivityDetected, etc.).
- Tipos de Notificaciones:
    - Enviar alertas por:
        - Correo electrónico (simular con un servicio de logs).

- SMS (simular con un endpoint dummy).
- Push notifications (simular con un mensaje en consola).

- Priorización de Alertas:
  - Clasificar alertas por gravedad (ej.: CRITICAL, WARNING, INFO).
  - Enviar notificaciones de alta prioridad inmediatamente.
  - Agrupar notificaciones de baja prioridad y enviar cada 30 minutos.
- Persistencia:
  - Registrar todas las notificaciones enviadas en CockroachDB en la tabla notifications.
    - Campos sugeridos: notification_id, event_type, recipient, status, timestamp.
- Resiliencia:
  - Si RabbitMQ falla, almacenar eventos pendientes y reintentar envío al recuperarse.
- Simulación de APIs Externas:
  - Implementar endpoints dummy para simular servicios de correo/SMS.
  - Ejemplo: POST /mock-email para recibir notificaciones simuladas.

E. Requisitos comunes a todos los Microservicios

1. Comunicación con API Gateway:

- Registrar automáticamente en Eureka Server.
- Acceder a los endpoints a través del API Gateway (ej.: GET /sensor-readings).
- Todos los endopoints saldrán por la ruta base **/api/conjunta/2p/**

2. Logging y Monitorización:

- Registrar logs detallados (nivel DEBUG/INFO/ERROR).
- Exponer métricas de rendimiento (ej.: número de eventos procesados).

3. Configuración Centralizada:

- Usar Spring Cloud Config para gestionar propiedades (ej.: umbrales de alerta).

4. Pruebas Unitarias y de Integración:

- Incluir pruebas automatizadas para endpoints, lógica de negocio y manejo de errores.

F. ¿Por qué los eventos llegan a todos los microservicios?

En una arquitectura basada en eventos (event-driven architecture), la difusión de eventos a todos los microservicios es un patrón clave para garantizar que cada componente pueda reaccionar de forma independiente a cambios en el sistema. A continuación, se explica el propósito y los beneficios de este diseño:

1. Propósito de la difusión de eventos

- **Desacoplamiento funcional:** Cada microservicio opera de forma autónoma sin depender directamente de otros. Al recibir todos los eventos, pueden actuar según su responsabilidad específica sin necesidad de coordinación explícita.

  Ejemplo: Cuando ocurre un evento HighTemperatureAlert, el NotificationDispatcher envía una alerta, mientras que el EnvironmentalAnalyzer actualiza su modelo predictivo.

- **Reactividad y escalabilidad:** Al no tener dependencias síncronas entre servicios, el sistema puede escalar horizontalmente y responder a eventos en tiempo real.

- **Auditoría y consistencia eventual:** Todos los microservicios tienen visibilidad de los eventos, lo que permite registrar acciones críticas (ej.: alertas, fallos) y garantizar que eventualmente se **alcance un estado consistente en todo el sistema.**

2. Casos de Uso Concretos

Los eventos globales permiten que cada microservicio responda a situaciones específicas:

EVENTO	ACCIÓN DEL SENSORDATACOLLECTOR	ACCIÓN DEL ENVIRONMENTALANALYZER	ACCIÓN DEL NOTIFICATIONDISPATCHER
NewSensorReadingEvent	Almacena datos brutos en CockroachDB.	Analiza tendencias y genera alertas.	No actúa directamente.
HighTemperatureAlert	Registra el evento para auditoría.	Actualiza modelos predictivos.	Envia notificaciones por correo/SMS.
SystemOfflineEvent	Almacena eventos localmente hasta que RabbitMQ se restaure.	Detiene análisis temporalmente.	Detiene el envío de notificaciones.
SystemBackOnlineEvent	Reenvía eventos almacenados a RabbitMQ.	Reanuda análisis con datos acumulados.	Reanuda envío de notificaciones pendientes.
SensorInactiveAlert	Verifica si el sensor está caído.	No actúa directamente.	Notifica al operador sobre el sensor inactivo.

G. Rúbrica de Evaluación

ASISTENCIA	PUNTOS	OBTENIDOS
Asiste puntualmente a clases	2	
REQUISITOS GENERALES		
Spring Cloud Gateway como API Gateway.	1	
Eureka Server para descubrimiento de servicios.	1	
CockroachDB con nodos replicados en al menos 2 zonas geográficas distintas.	1	
MICROSERVICIOS		
SensorDataCollector: Recibe datos brutos de sensores y los almacena.	5	
EnvironmentalAnalyzer: Analiza datos y genera alertas basadas en umbrales.	5	
NotificationDispatcher: Envia notificaciones a usuarios y sistemas externos.	5	
TOTAL	**20**	