

Memoria actualizada
¡Felicidades por haber completado el proyecto con Spring Boot, Kafka, Elasticsearch, y Kibana! Ahora que has adquirido experiencia trabajando con estas tecnologías, te propongo un desafío que simule una necesidad real en el ámbito de la tecnología, especialmente en empresas que manejan grandes volúmenes de datos y requieren sistemas distribuidos y escalables.

Desafío propuesto: Sistema de Monitoreo en Tiempo Real de Transacciones Bancarias
Enunciado del Desafío:

Desarrolla un sistema distribuido que permita monitorear transacciones bancarias en tiempo real utilizando un flujo de procesamiento basado en eventos. El sistema debe ser capaz de:

Consumir transacciones bancarias en tiempo real (simuladas o a través de una API) y procesarlas para realizar análisis y detección de fraudes.
Indexar las transacciones en Elasticsearch para que puedan ser buscadas y visualizadas en Kibana en tiempo real, permitiendo un análisis detallado de cada transacción.

Notificar anomalías o posibles fraudes utilizando eventos Kafka para comunicar entre servicios de forma asíncrona.

Objetivos del Desafío:

Microservicio de Producción de Transacciones:

Simula o recibe un flujo de transacciones bancarias en tiempo real.

Envía cada transacción a un tópico de Kafka.

Cada transacción debe contener campos como:

id (identificador único)

monto

tipo_transacción (depósito, retiro, transferencia, etc.)

timestamp

origen y destino de la cuenta.

Microservicio de Procesamiento de Transacciones:

Consume las transacciones de Kafka en tiempo real.

Implementa una lógica de negocio para detectar anomalías o posibles fraudes basados en reglas simples, como:

Montos muy altos que superen un umbral.

Frecuencia inusualmente alta de transacciones desde una misma cuenta en un corto período de tiempo.

Cuando se detecta un fraude, envía un evento de notificación a un segundo tópico de Kafka (alerts).

Persistencia en Elasticsearch:

Todas las transacciones se deben indexar en Elasticsearch, para que luego puedan ser consultadas y analizadas en tiempo real desde Kibana.

Usa Kibana para crear dashboards personalizados que muestren:

El flujo de transacciones en tiempo real.

Las posibles alertas de fraude.

Estadísticas agregadas (monto promedio, transacciones por tipo, etc.).

Servicio de Notificaciones (Opcional):

Consume las alertas de fraude desde el tópico de Kafka alerts y envía notificaciones o alertas a un sistema externo (por ejemplo, por email o logs).

Requisitos:

Tecnologías a Usar:

Spring Boot: Para los microservicios de producción y procesamiento de transacciones.
Kafka: Como middleware para manejar los eventos de transacciones y notificaciones de fraude.

ElasticSearch: Para almacenar y permitir búsquedas sobre las transacciones indexadas.

Kibana: Para visualizar las transacciones y alertas de fraude, construyendo dashboards y gráficos.

Docker (Opcional): Puedes orquestar los distintos componentes utilizando contenedores Docker (Kafka, Elasticsearch, Kibana, los microservicios).

Consideraciones de Arquitectura:

Los microservicios deben estar desacoplados y comunicarse exclusivamente a través de Kafka.

Las transacciones deben ser procesadas en tiempo real y persistidas en Elasticsearch para su posterior análisis.

Las alertas de fraude deben ser notificadas de manera asíncrona.

Desafíos Adicionales:

54 Escalabilidad: Haz que los microservicios sean escalables horizontalmente. Por ejemplo, puedes tener múltiples instancias del microservicio de procesamiento de transacciones consumiendo del mismo tópico de Kafka.

55 Tolerancia a Fallos: Implementa una estrategia para manejar errores y garantizar que las transacciones no se pierdan si uno de los servicios falla.

56 Paginar resultados en Elasticsearch: Si el número de transacciones es muy grande, implementa paginación en las consultas a Elasticsearch para poder visualizar los resultados en Kibana de manera eficiente.

57 Seguridad: Agrega autenticación y autorización (JWT o OAuth2) para proteger los endpoints de tus microservicios y el acceso a Elasticsearch.

58 Casos de Uso:

59 Consulta de Transacciones: Los analistas pueden consultar transacciones específicas o filtrar por tipo, monto, o cuentas específicas desde Kibana.

60 Monitoreo de Fraude: Los administradores pueden visualizar alertas de posibles fraudes en tiempo real, con detalles sobre cada transacción sospechosa.

61 Notificaciones de Fraude: Cuando se detecta una transacción sospechosa, los administradores reciben una alerta automática para investigar el caso.

62 Retos Reales que Simula este Desafío:

63 Procesamiento en Tiempo Real: Procesar grandes volúmenes de datos de manera eficiente y en tiempo real.

64 Manejo de Eventos Distribuidos: Utilizar Kafka para manejar flujos de eventos y asegurar una comunicación desacoplada entre microservicios.

65 Indexación y Búsqueda Escalable: Utilizar Elasticsearch para almacenar y buscar grandes volúmenes de datos de manera eficiente.

66 Análisis y Visualización: Crear dashboards interactivos en Kibana para facilitar la visualización y análisis de los datos en tiempo real.

67 Beneficios de este Desafío:

68 Este proyecto simula un caso real en las fintech o en empresas que manejan transacciones y necesitan procesarlas y monitorearlas en tiempo real. Al desarrollar este desafío, consolidarás los conocimientos sobre Kafka, Elasticsearch, Spring Boot, y la capacidad de construir un ecosistema distribuido, escalable y tolerante a fallos.

69

70 Si te interesa más algún aspecto particular, como la escalabilidad o la seguridad, puedes profundizar en esas áreas durante el desafío. ¡Espero que disfrutes consolidar tus conocimientos con este reto! Si tienes alguna duda o necesitas ayuda en algún paso, no dudes en pedírmelo. ¡Estoy aquí para ayudarte!

71

72

73

74

75

76

77 Dijiste:

78 compárteme la estructura básica de cada componente

79 ChatGPT dijo:

80 ChatGPT

81 ¡Claro! A continuación te doy una estructura básica de cada componente del desafío propuesto para desarrollar un Sistema de Monitoreo en Tiempo Real de Transacciones Bancarias utilizando Spring Boot, Kafka, Elasticsearch, y Kibana.

82

83 1. Microservicio de Producción de Transacciones (Producer)

84 Este microservicio genera las transacciones y las envía a un tópico de Kafka.

85

86 Estructura Básica:

87 bash

88 Copiar código

89 /transaction-producer

```
90 |
91 |   src
92 |   |   main
93 |   |   |   java
94 |   |   |   |   com.kafka.poc.transactionproducer
95 |   |   |   |   |   TransactionProducerApplication.java
96 |   |   |   |   |   config
97 |   |   |   |   |   |   KafkaProducerConfig.java
98 |   |   |   |   |   controller
99 |   |   |   |   |   |   TransactionController.java
100 |   |   |   |   |   model
101 |   |   |   |   |   |   Transaction.java
102 |   |   |   |   |   service
103 |   |   |   |   |   |   TransactionService.java
104 |   |   |   resources
105 |   |   |   |   application.yml
106 |   pom.xml
```

```

106 Archivos clave:
107 TransactionProducerApplication.java: Clase principal de Spring Boot.
108 KafkaProducerConfig.java: Configuración del productor de Kafka.
109 TransactionController.java: Controlador que recibe peticiones HTTP para simular
    transacciones.
110 TransactionService.java: Lógica para crear y enviar transacciones a Kafka.
111 Transaction.java: Modelo de transacción.
112 Ejemplo de Transaction.java (modelo):
113 java
114 Copiar código
115 public class Transaction {
116     private String id;
117     private String sourceAccount;
118     private String destinationAccount;
119     private double amount;
120     private String type;
121     private String timestamp;
122
123     // Getters y setters
124 }
125 Ejemplo de TransactionService.java:
126 java
127 Copiar código
128 @Service
129 public class TransactionService {
130
131     @Autowired
132     private KafkaTemplate<String, Transaction> kafkaTemplate;
133
134     private static final String TOPIC = "transactions";
135
136     public void sendTransaction(Transaction transaction) {
137         kafkaTemplate.send(TOPIC, transaction);
138     }
139 }
140 2. Microservicio de Procesamiento de Transacciones (Consumer)
141 Este microservicio consume las transacciones de Kafka, las analiza, detecta fraudes,
    y las indexa en Elasticsearch.
142
143 Estructura Básica:
144 bash
145 Copiar código
146 /transaction-consumer
147 └─ src
148     └─ main
149         └─ java
150             └─ com.kafka.poc.transactionconsumer
151                 └─ TransactionConsumerApplication.java
152                 └─ config
153                     └─ KafkaConsumerConfig.java
154                 └─ service
155                     └─ TransactionService.java
156                 └─ elasticsearch
157                     └─ ElasticSearchConfig.java
158                     └─ ElasticSearchService.java
159             └─ resources
160                 └─ application.yml
161 └─ pom.xml
162 Archivos clave:
163 TransactionConsumerApplication.java: Clase principal de Spring Boot.
164 KafkaConsumerConfig.java: Configuración del consumidor de Kafka.
165 TransactionService.java: Lógica para consumir las transacciones y procesarlas.
166 ElasticSearchConfig.java: Configuración del cliente de Elasticsearch.
167 ElasticSearchService.java: Servicio para indexar las transacciones en Elasticsearch.
168 Ejemplo de TransactionService.java (consumidor):
169 java
170 Copiar código
171 @Service
172 public class TransactionService {
173
174     @Autowired
175     private ElasticSearchService elasticSearchService;
176

```

```

177     @KafkaListener(topics = "transactions", groupId = "transaction_group")
178     public void consumeTransaction(Transaction transaction) {
179         // Lógica de detección de fraude
180         if (transaction.getAmount() > 10000) {
181             // Enviar una alerta de fraude a otro tópico, opcionalmente
182         }
183
184         // Indexar la transacción en Elasticsearch
185         elasticSearchService.indexTransaction(transaction);
186     }
187 }

```

188 Ejemplo de ElasticSearchService.java (indexación):

```

189 java
190 Copiar código
191 @Service
192 public class ElasticSearchService {
193
194     @Autowired
195     private RestHighLevelClient client;
196
197     public void indexTransaction(Transaction transaction) {
198         IndexRequest request = new IndexRequest("finytec-transactions");
199         request.source(transaction.toMap(), XContentType.JSON);
200
201         client.indexAsync(request, RequestOptions.DEFAULT, new
202             ActionListener<IndexResponse>() {
203                 @Override
204                 public void onResponse(IndexResponse indexResponse) {
205                     // Transacción indexada exitosamente
206                 }
207
208                 @Override
209                 public void onFailure(Exception e) {
210                     // Manejar error de indexación
211                 }
212             });
213     }
214 }

```

214 3. Kibana (Visualización)

215 Kibana no requiere estructura de código, pero sí una configuración mínima para conectarlo a tu instancia de Elasticsearch. Puedes configurar dashboards personalizados en Kibana para visualizar las transacciones y posibles alertas de fraude.

216 Configuración de Kibana (opcional en kibana.yml):

```

217 yaml
218 Copiar código
219 server.host: "localhost"
220 elasticsearch.hosts: ["http://localhost:9200"]

```

222 Ejemplo de Dashboard:

223 Dashboard de Transacciones: Visualiza transacciones en tiempo real, su monto, tipo y alertas de fraude.

224 Alertas de Fraude: Un gráfico con las transacciones que superan los umbrales definidos como riesgosos.

225 4. Servicio de Notificaciones de Fraude (Opcional)

226 Este microservicio consume las alertas de fraude desde un tópico de Kafka y envía notificaciones o logs.

227 Estructura Básica:

```

228 bash
229 Copiar código
230 /fraud-alert-service
231 └─ src
232     └─ main
233         └─ java
234             └─ com.kafka.poc.fraudalertservice
235                 └─ FraudAlertServiceApplication.java
236                 └─ config
237                     └─ KafkaConsumerConfig.java
238                 └─ service
239                     └─ AlertService.java
240             └─ resources
241                 └─ application.yml
242

```

```
243   └─ pom.xml
244 Ejemplo de AlertService.java:
245 java
246 Copiar código
247 @Service
248 public class AlertService {
249
250     @KafkaListener(topics = "fraud-alerts", groupId = "alert_group")
251     public void consumeAlert(FraudAlert alert) {
252         // Lógica para enviar notificaciones o manejar la alerta
253         System.out.println("Alerta de fraude detectada: " + alert);
254     }
255 }
256 Resumen de los Componentes:
257 Productor de Transacciones: Simula transacciones y las envía a Kafka.
258 Consumidor de Transacciones: Procesa las transacciones, detecta fraudes, y las indexa
    en Elasticsearch.
259 Kibana: Dashboards para visualizar transacciones y alertas de fraude.
260 Servicio de Notificaciones (Opcional): Consume alertas de fraude y las notifica.
261 Este esquema es la base de un sistema distribuido que simula un entorno de monitoreo
    de transacciones en tiempo real. Puedes expandirlo con más funcionalidad, como
    autenticación, análisis avanzado de datos, o integración con otros sistemas. Si
    tienes más preguntas o necesitas ayuda en algún aspecto, ¡estoy aquí para ayudarte!
```

262

263

264

265

266

267

268 Dijiste:

269 Me puedes generar toda la estructura completa en un archivo PDF

270 ChatGPT dijo:

271 ChatGPT

272 He generado el archivo PDF con la estructura básica de cada componente del proyecto.
Puedes descargarlo desde el siguiente enlace:

273

274 [Descargar Project_Structure_Kafka_Elastic_Kibana.pdf](#)

275

276 Si necesitas más ajustes o detalles, no dudes en avisarme. ¡Estoy aquí para ayudarte!

277

278

279

280

281

282

283

284

285

286

287