Kafka Connect: что за зверь этот ваш Single Message Transform?

Даниэл Рачич

Привет всем

Какой у нас план

- Поговорим про Kafka и Kafka Connect
- Расскажу про Single Message Transform
- Вместе реализуем SMT для шифрования чувствительных данных
- Обсудим вопросы интеграции SMT в инфраструктуру, а также CI/CD
- Отвечу на ваши вопросы

Kafka и Kafka Connect

Обзор

- Apache Kafka – распределённый программный брокер сообщений
- Apache Kafka Connect – фреймворк для подключения, импорта и экспорта данных в любую внешнюю систему или из нее, через кластер Kafka. Он обеспечивает стандартизированный способ подключения к разнообразным системам и позволяет легко передавать данные в и из Kafka топиков

Kafka и Kafka Connect

Роль

- Интеграция данных
- Реактивная архитектура
- Микросервисная архитектура
- Аналитические платформы

Kafka и Kafka Connect

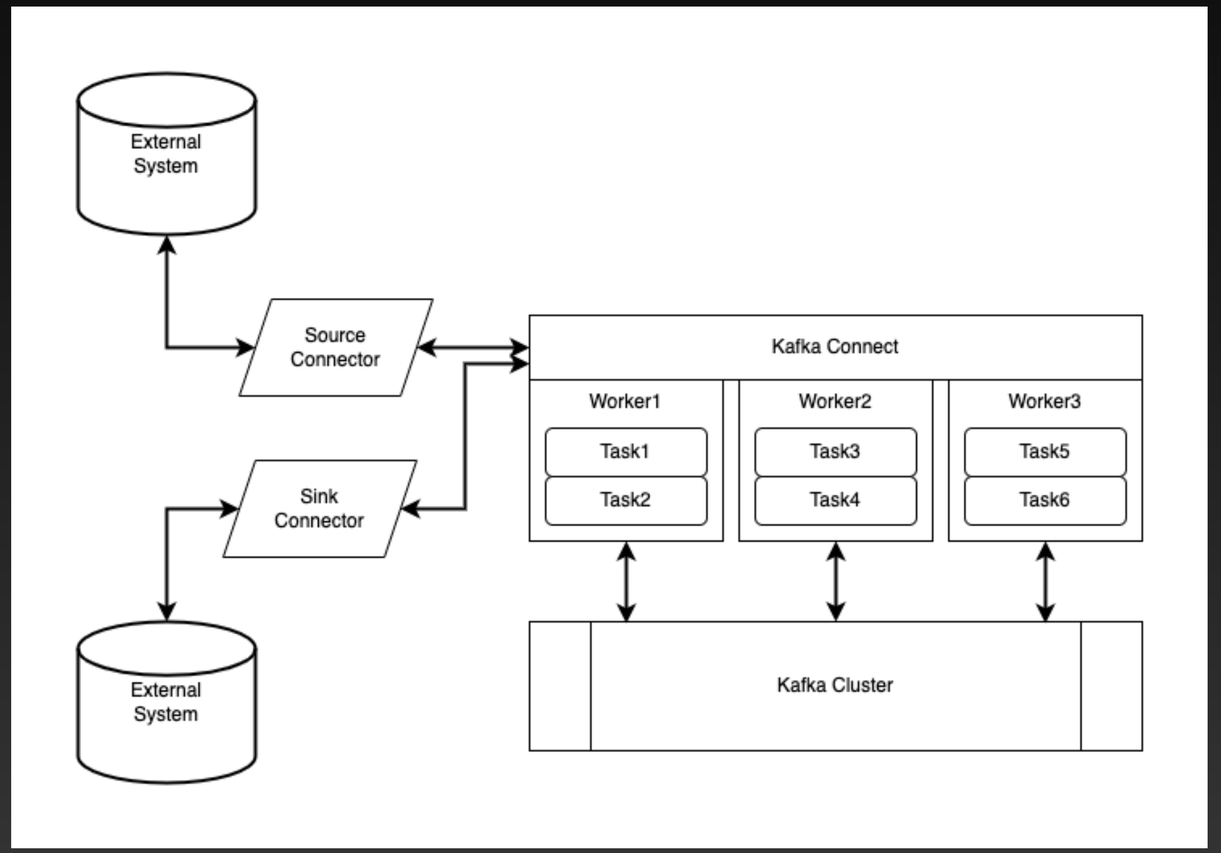
Преимущества

- Стандартизация
- Упрощение интеграции
- Масштабируемость и отказоустойчивость
- Простота управления

Kafka и Kafka Connect

Архитектура

- Коннекторы и Задачи
- Конфигурирование и управление
- Кластер и масштабирование



Kafka и Kafka Connect

Коннекторы и задачи

- Коннекторы - это модули, которые определяют способы подключения к источникам и приемникам данных. Они абстрагируют детали общения с конкретной системой, позволяя сосредоточиться на том, какие данные нужно передавать и как обрабатывать результаты
- Задачи - это отдельные процессы, которые фактически выполняют передачу данных между источником и Kafka топиками

Kafka и Kafka Connect

Конфигурирование и управление

- REST API для управления
- Конфигурационные файлы
- Метрики и журналы
- Интеграция с мониторингом
- Обработка ошибок
- Стратегии ретраев

Kafka и Kafka Connect

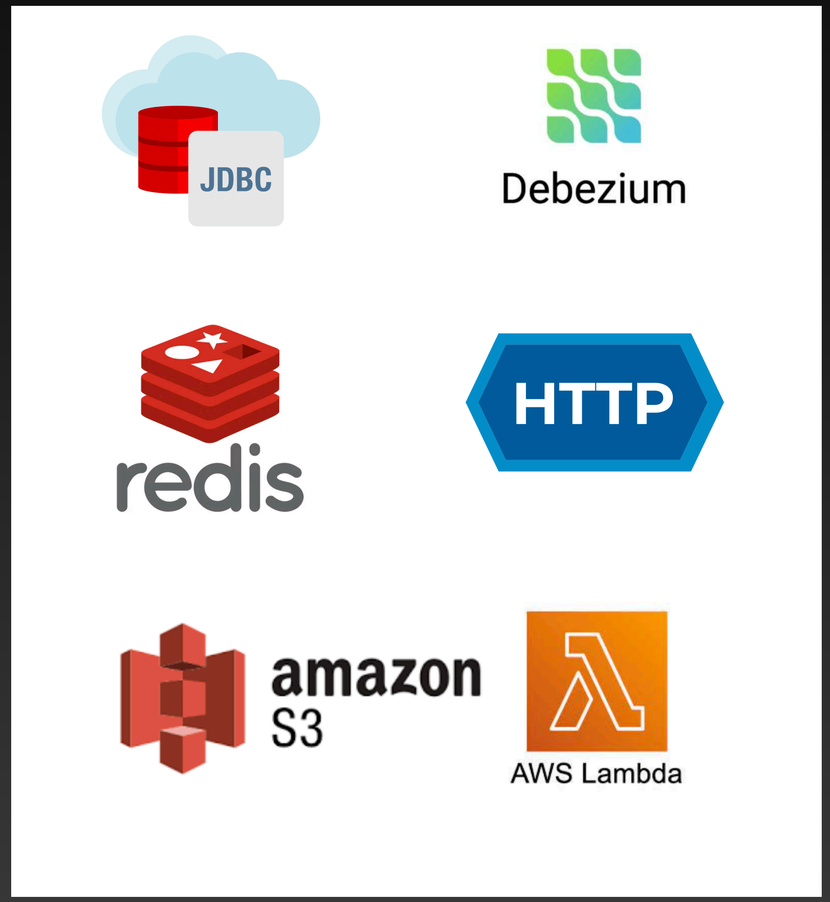
Масштабируемость и отказоустойчивость

- Горизонтальное масштабирование
- Параллельная обработка
- Перезапуск задач
- Репликация данных
- Автоматическое восстановление
- Управление состоянием

Kafka и Kafka Connect

Примеры коннекторов

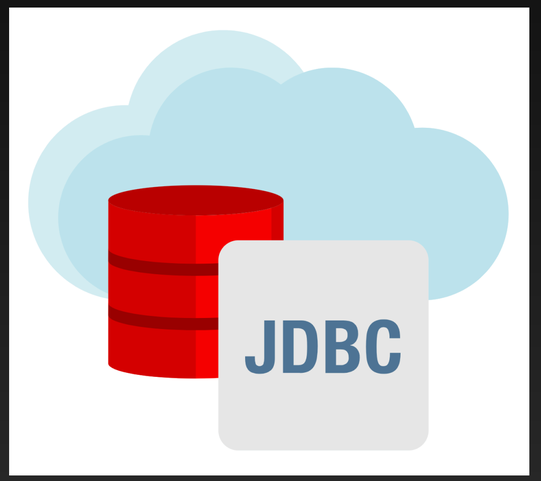
- Source
  - JDBC
  - FileStream
  - HTTP
  - Debezium
  - …
- Sink
  - JDBC
  - S3
  - Redis
  - Lambda
  - …



Kafka и Kafka Connect

JDBC Connector и Debezium

- JDBC Connector
  - Периодические выгрузки
  - Кастомный SQL
- Debezium
  - CDC
  - Real-time приложения



VS

Kafka и Kafka Connect

Преимущества JDBC Connector

- Batch обработка
- Собственная логика SQL
- Загрузка в DWH
- Исторические данные и снапшоты
- Проще и меньше оверхед внедрения

Kafka и Kafka Connect

Преимущества Debezium

- Real-time стриминг
- Более эффективный (использует нативные CDC механизмы)
- Поддержка схем и их изменений
- Порядок изменений
- Относительная независимость от БД

Single Message Transform

Обзор

- Single Message Transform (SMT) - механизм обработки и преобразования отдельных сообщений в Kafka Connect перед тем, как они будут записаны в Kafka топик или после чтения из топика. SMT предоставляет мощные инструменты для манипулирования данными в потоках и позволяет применять различные операции на уровне отдельных сообщений

Single Message Transform

Роль

- Преобразование данных
- Преобразование метаданных
- Нормализация и стандартизация
- Расширение возможностей коннекторов

Single Message Transform

Примеры

- Изменение формата и структуры данных
- Обогащение данных и метаданных
- Фильтрация по условиям
- Валидация данных
- Шифрование конфиденциальных данных

SMT для шифрования чувствительных данных

Проблема безопасности чувствительных данных

- Утечка данных
- Нарушение конфиденциальности
- Законодательные требования и штрафы

SMT для шифрования чувствительных данных

Интерфейс

- Transformation<R>

```java
public interface Transformation<R extends ConnectRecord<R>> {
    /**
     * Возвращает конфигурацию, необходимую для преобразования.
     * @return Конфигурация преобразования.
     */
    ConfigDef config();

    /**
     * Конфигурирует преобразование на основе переданных параметров конфигурации.
     * @param configs Параметры конфигурации для преобразования.
     */
    void configure(Map<String, ?> configs);

    /**
     * Применяет преобразование к записи.
    * @param record Запись, к которой применяется преобразование.
     * @return Новая запись с примененным преобразованием.
     */
    R apply(R record);

    /**
     * Освобождает ресурсы, используемые преобразованием.
     */
    void close();
}
```

SMT для шифрования чувствительных данных

Константы и config()

```
public static final String FIELDS_CONFIG = "fields";
public static final String ALGORITHM_CONFIG = "algorithm";
public static final String KEY_CONFIG = "key";

private static final ConfigDef CONFIG_DEF = new ConfigDef()
    .define(FIELDS_CONFIG, ConfigDef.Type.LIST, ConfigDef.Importance.HIGH, "Fields to encrypt")
    .define(ALGORITHM_CONFIG, ConfigDef.Type.STRING, ConfigDef.Importance.HIGH, "Encryption algorithm")
    .define(KEY_CONFIG, ConfigDef.Type.STRING, ConfigDef.Importance.HIGH, "Encryption key");

private List<String> fields;
private String algorithm;
private String key;
private Key secretKey;

@Override
public ConfigDef config() {
    return CONFIG_DEF;
}
```

SMT для шифрования чувствительных данных

Реализация configure()

```
@Override
public void configure(Map<String, ?> configs) {
    final SimpleConfig config = new SimpleConfig(CONFIG_DEF, configs);
    this.fields = config.getList(FIELDS_CONFIG);
    this.algorithm = config.getString(ALGORITHM_CONFIG);
    this.key = config.getString(KEY_CONFIG);

    byte[] decodedKey = Base64.getDecoder().decode(key);
    this.secretKey = new SecretKeySpec(decodedKey, 0, decodedKey.length, algorithm);
}
```

SMT для шифрования чувствительных данных

Реализация apply()

```
@Override
public R apply(R record) {
    Struct value = (Struct) record.value();

    for (String field : this.fields) {
        String originalValue = value.getString(field);
        String encryptedValue = encrypt(originalValue);
        value.put(field, encryptedValue);
    }

    return record.newRecord(
        record.topic(),
        record.kafkaPartition(),
        record.keySchema(),
        record.key(),
        value.schema(),
        value,
        record.timestamp()
    );
}
```

```
private String encrypt(String data) {
     try {
         Cipher cipher = Cipher.getInstance(this.algorithm);
         cipher.init(Cipher.ENCRYPT_MODE, this.secretKey);
         byte[] encryptedBytes=cipher
           .doFinal(data.getBytes(StandardCharsets.UTF_8));
         return Base64.getEncoder().encodeToString(encryptedBytes);
     } catch (Exception e) {
         throw new RuntimeException("Encryption error: " +
            e.getMessage());
     }
 }
```

SMT для шифрования чувствительных данных

Реализация apply() для Debezium

```
@Override
public R apply(R record) {

   Schema recordSchema = record.valueSchema();
   Struct recordValue = requireStruct(record.value(), "Encrypting");

   List<String> sensitiveFields = this.sensitiveDataRepository.listSensitiveFields();
   Schema newSchema = updateSchema(recordSchema, sensitiveFields);
   Struct newValue = updateValue(recordValue, recordSchema, newSchema, sensitiveFields);

   return record.newRecord(record.topic(), record.kafkaPartition(), record.keySchema(), record.key(),
        newSchema, newValue, record.timestamp(), record.headers());
}
```

SMT для шифрования чувствительных данных

Реализация apply() для Debezium

```
private Schema updateSchema(Schema recordSchema, List<String> sensitiveFields) {

    SchemaBuilder newSchemaBuilder = copySchema(recordSchema, List.of(AFTER_FIELD, BEFORE_FIELD));

    Schema encryptedFieldDataSchema = SchemaBuilder.struct()
          .field(ORIGINAL_TYPE_FIELD, Schema.STRING_SCHEMA)
          .field(KEY_FIELD, Schema.STRING_SCHEMA)
          .build();

    SchemaBuilder encryptedBeforeSchemaBuilder = updateDataSchemas(
          recordSchema, encryptedFieldDataSchema, newSchemaBuilder, BEFORE_FIELD, sensitiveFields
    );
    SchemaBuilder encryptedAfterSchemaBuilder = updateDataSchemas(
          recordSchema, encryptedFieldDataSchema, newSchemaBuilder, AFTER_FIELD, sensitiveFields
    );

    Schema encryptedFieldsSchema = SchemaBuilder.struct()
          .field(BEFORE_FIELD, encryptedBeforeSchemaBuilder.build())
          .field(AFTER_FIELD, encryptedAfterSchemaBuilder.build())
          .build();
    newSchemaBuilder.field(ENCRYPTED_FIELDS_FIELD, encryptedFieldsSchema);

    return newSchemaBuilder.build();
}
```

SMT для шифрования чувствительных данных

Реализация apply() для Debezium

```
private SchemaBuilder updateDataSchemas(Schema recordSchema, Schema encryptedFieldDataSchema, SchemaBuilder newSchemaBuilder,
   String dataField, List<String> sensitiveFields) {

   SchemaBuilder dataFieldSchemaBuilder = copySchema(recordSchema.schema().field(dataField).schema(), sensitiveFields);
   SchemaBuilder encryptedDataFieldSchemaBuilder = SchemaBuilder.struct().optional();

   for (Field field: recordSchema.field(dataField).schema().fields()) {
      if (!sensitiveFields.contains(field.name())) {
         continue;
      }
      SchemaBuilder newFieldSchema = copySchema(field.schema(), Schema.Type.STRING, List.of());
      dataFieldSchemaBuilder.field(field.name(), newFieldSchema.build());
      encryptedDataFieldSchemaBuilder.field(field.name(), encryptedFieldDataSchema);
   }
   newSchemaBuilder.field(dataField, encryptedDataFieldSchemaBuilder());

   return encryptedBeforeAfterSchemaBuilder;
}
```

SMT для шифрования чувствительных данных

Конфигурирование

- Source Connector
- Sink Connector
- SMT

```
name=my-secure-connector
connector.class=io.confluent.connect.jdbc.JdbcSourceConnector
tasks.max=1
connection.url=jdbc:mysql://your-database-url
connection.user=db-username
connection.password=db-password
table.whitelist=your-table-name

transforms=encrypt
transforms.encrypt.type=org.example.EncryptionTransform
transforms.encrypt.fields=field1,field2
transforms.encrypt.algorithm=AES
transforms.encrypt.key=your-encryption-key

topics=your-topic
```

SMT для шифрования чувствительных данных

Тестирование

```
public class EncryptionTransformTest {

    private Transformation<SinkRecord> transform;

    @Before
    public void setUp() {
        transform = new EncryptionTransform<>();
        Map<String, Object> props = new HashMap<>();
        props.put(EncryptionTransform.FIELDS_CONFIG, "field1,field2");
        props.put(EncryptionTransform.ALGORITHM_CONFIG, "AES");
        props.put(EncryptionTransform.KEY_CONFIG, "your-encryption-key");

        ConfigDef configDef = new EncryptionTransform<>().config();
        transform.configure(new SimpleConfig(configDef, props).originals());
    }

    . . .

}
```

SMT для шифрования чувствительных данных

Тестирование

```
@Test
public void testEncryption() {
    Schema schema = SchemaBuilder.struct()
            .field("field1", Schema.STRING_SCHEMA)
            .field("field2", Schema.STRING_SCHEMA)
            .build();

    Struct value = new Struct(schema);
    value.put("field1", "SensitiveData1");
    value.put("field2", "SensitiveData2");

    SinkRecord record = new SinkRecord("test-topic", 0, null, null, schema, value, 0);

    SinkRecord transformedRecord = transform.apply(record);

    Struct transformedValue = (Struct) transformedRecord.value();
    String encryptedField1 = transformedValue.getString("field1");
    String encryptedField2 = transformedValue.getString("field2");

    // реализовать дешифрование
    String decryptedField1 = decrypt(encryptedField1);
    String decryptedField2 = decrypt(encryptedField2);

    assertEquals("SensitiveData1", decryptedField1);
    assertEquals("SensitiveData2", decryptedField2);
}
```

SMT для шифрования чувствительных данных

CI/CD

- Docker
- K8s
- Mesos/Nomad/ECS и др

SMT для шифрования чувствительных данных

CI/CD Docker

- Dockerfile

```
# Используем базовый image KafkaConnect (не забываем фиксировать версию)
FROM confluentinc/cp-kafka-connect:7.5.0

# Копируем наш SMT
COPY EnryptionTransform.jar /usr/local/share/kafka/plugins/

# Остальные настройки и команды Dockerfile
```

```
# В настройках передаём config param:

plugin.path=/usr/local/share/kafka/plugins/
```

SMT для шифрования чувствительных данных

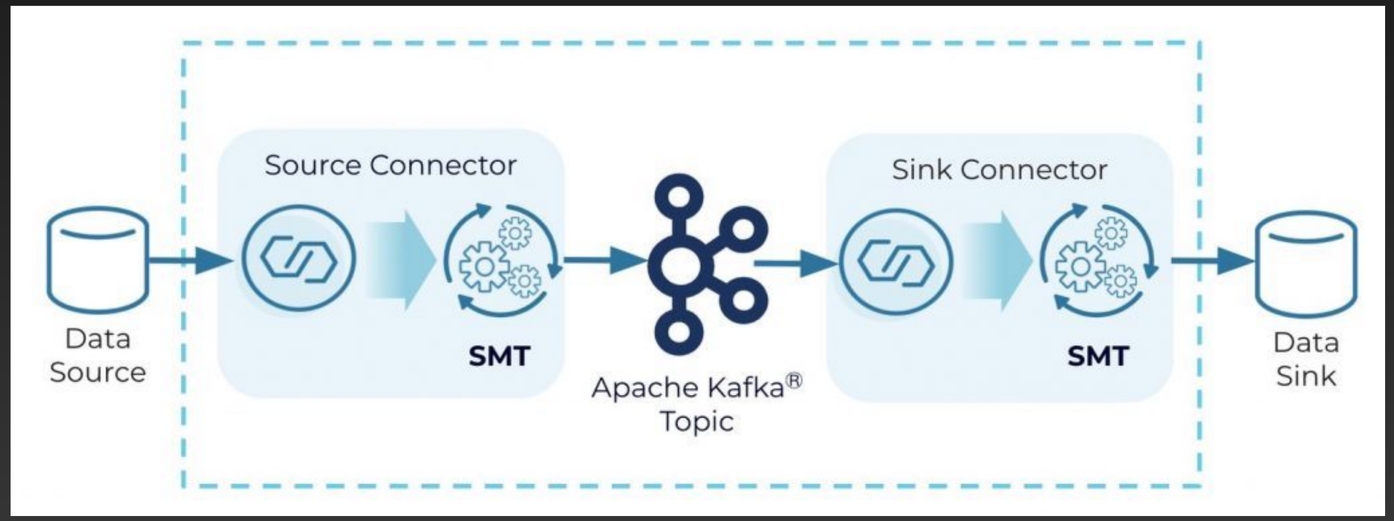
CI/CD K8s

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: kafka-connect
spec:
  replicas: 1
  template:
    metadata:
      labels:
        app: kafka-connect
    spec:
      containers:
        - name: kafka-connect
          image: confluentinc/cp-kafka-connect:7.5.0
          ports:
            - containerPort: 8083
          volumeMounts:
            - name: kafka-connect-plugins
              mountPath: /usr/share/java/kafka-connect/smt/
      initContainers:
        - name: clone-smt-from-git
          image: alpine/git:2.40.1
          command: ["/bin/sh", "-c"]
          args:
            - git clone <your-git-repo-url> /usr/share/java/kafka-connect/smt/
          volumeMounts:
            - name: kafka-connect-plugins
              mountPath: /usr/share/java/kafka-connect/smt/
      volumes:
        - name: kafka-connect-plugins
          emptyDir: {}
```

SMT для шифрования чувствительных данных

Использование

- Шифрование данных перед Kafka
- Дешифрование данных после Kafka



SMT для шифрования чувствительных данных

Преимущества

- Простая интеграция
- Отказоустойчивость и масштабируемость
- Независимость источника и назначения
- Централизация безопасности
- Прозрачность для приложений

SMT для шифрования чувствительных данных

Недостатки

- Производительность
- Зависимость от инфраструктуры
- Задержки обработки потока
- Возможность обработки сообщений только по одному (S в SMT)

Заключение

Вопросы