Эксплуатация Arenadata Streaming (Kafka, NiFi)

ARENADATA

Arenadata Streaming Control (ADS Control)

Agenda

- Основные операции ADS Control:

  управление и добавление коннекторов (FileStreamSinkConnector, FileStreamSourceConnector, MirrorCheckpointConnector, MirrorHeartbeatConnector, MirrorSourceConnector, пользовательских, Debezium).

  Лабораторная работа.

Основные операции ADS Control

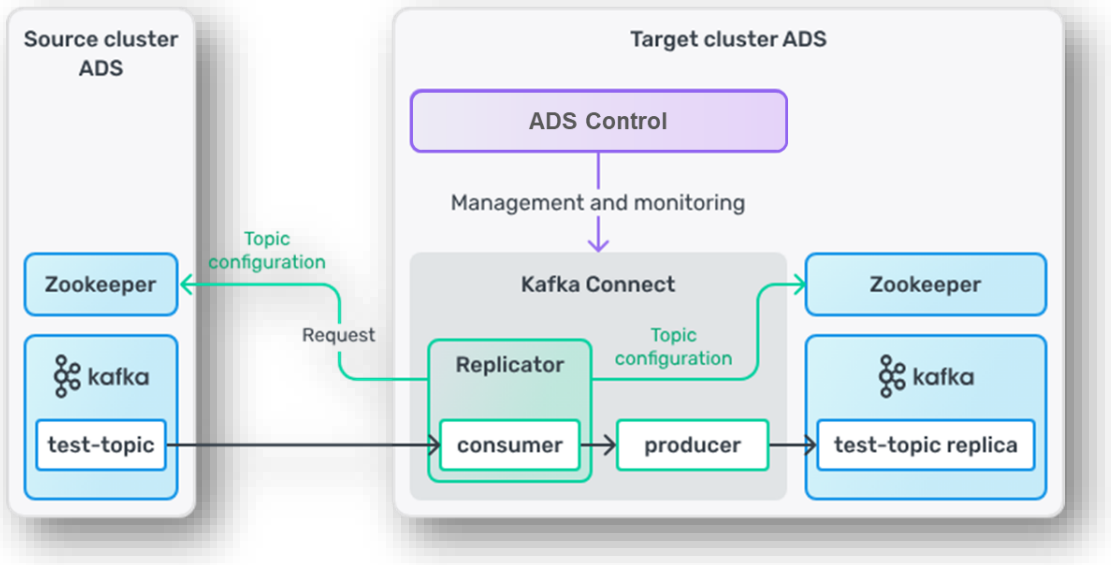
ARENADATA

Mirror’s Connectors

- MirrorMaker 2.0 — основанный на платформе сервиса Kafka Connect механизм репликации данных из исходного кластера на удаленный.
- Active/Active: кластеры получают данные непосредственно от источников данных и реплицированные данные, вводимые из удаленного кластера.
- Active/Standby: целевой кластер находится в пассивном режиме (не имеет подключенных к нему потребителей и производителей) и получает только реплицированные данные.

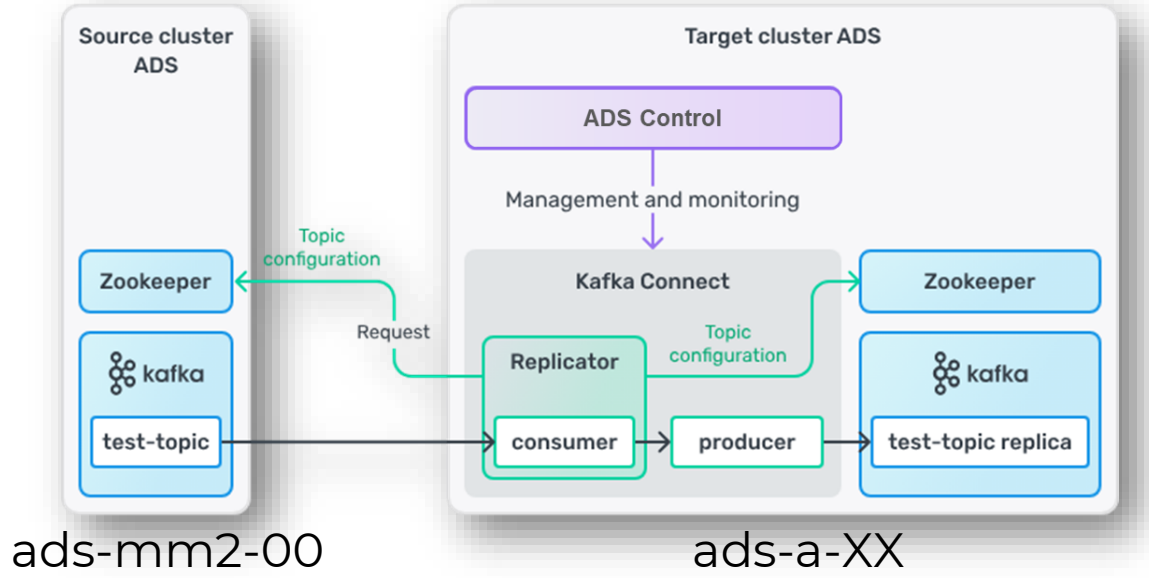
MirrorMaker 2.0 в составе Kafka Connect создает специальные коннекторы:

- MirrorSourceConnector осуществляет репликацию топиков из исходного кластера в целевой кластер.
- MirrorCheckpointConnector создает контрольные точки смещения потребителя и синхронизирует смещение со служебным топиком __consumer_offsets исходного кластера.
- MirrorHeartbeatConnector периодически проверяет подключение между кластерами, создавая сообщения в специальном топике heartbeats в исходном кластере через заданный период времени и считывая их в целевом кластере.



Mirror Maker 2 в ADS Control

- Исходный кластер (source) — кластер, из которого происходит репликация топиков.
- Целевой кластер (target) — кластер, на который происходит репликация топиков.
- Исходный топик — топик, созданный на исходном кластере и предназначенный для репликации в целевой кластер.
- Топик-реплика — топик, автоматически созданный на целевом кластере и имеющий такое же содержимое и метаданные, как и исходный топик.
- В исходном и целевом кластере должен быть установлен и настроен сервис Kafka Connect
- Настроим репликацию из кластера **ads-mm2** в кластер **ads-a**.



Mirror Maker 2 в ADS Control

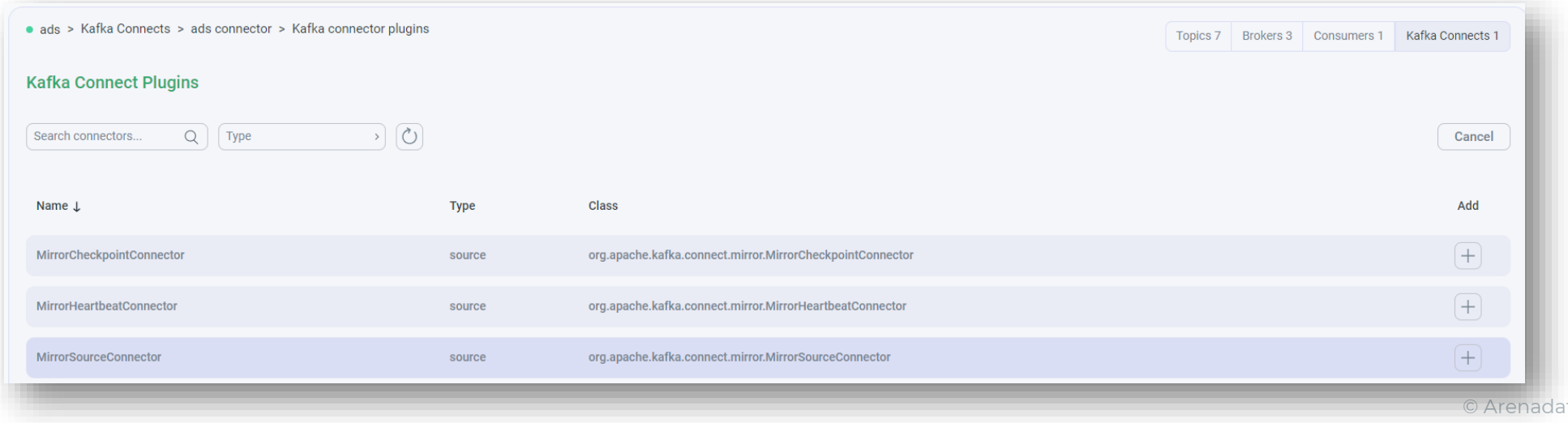
- Создадим топик в исходном (source) кластере (ads-mm2):

/usr/lib/kafka/bin/kafka-topics.sh --create --topic test-active-topic<XX> --bootstrap-server <host_source>:9092

- Запишем несколько сообщений (one, two, three) в топик test-active-topic<XX> :

/usr/lib/kafka/bin/kafka-console-producer.sh --topic test-active-topic<XX> --bootstrap-server <host_source>:9092
>one
>two
>three

- Перейдем в WEB UI ADSC. На странице в списке кластеров Kafka Connect выбрать кластер ADS создать коннектор MSCFromInternalADS(тип: MirrorSourceConnector):



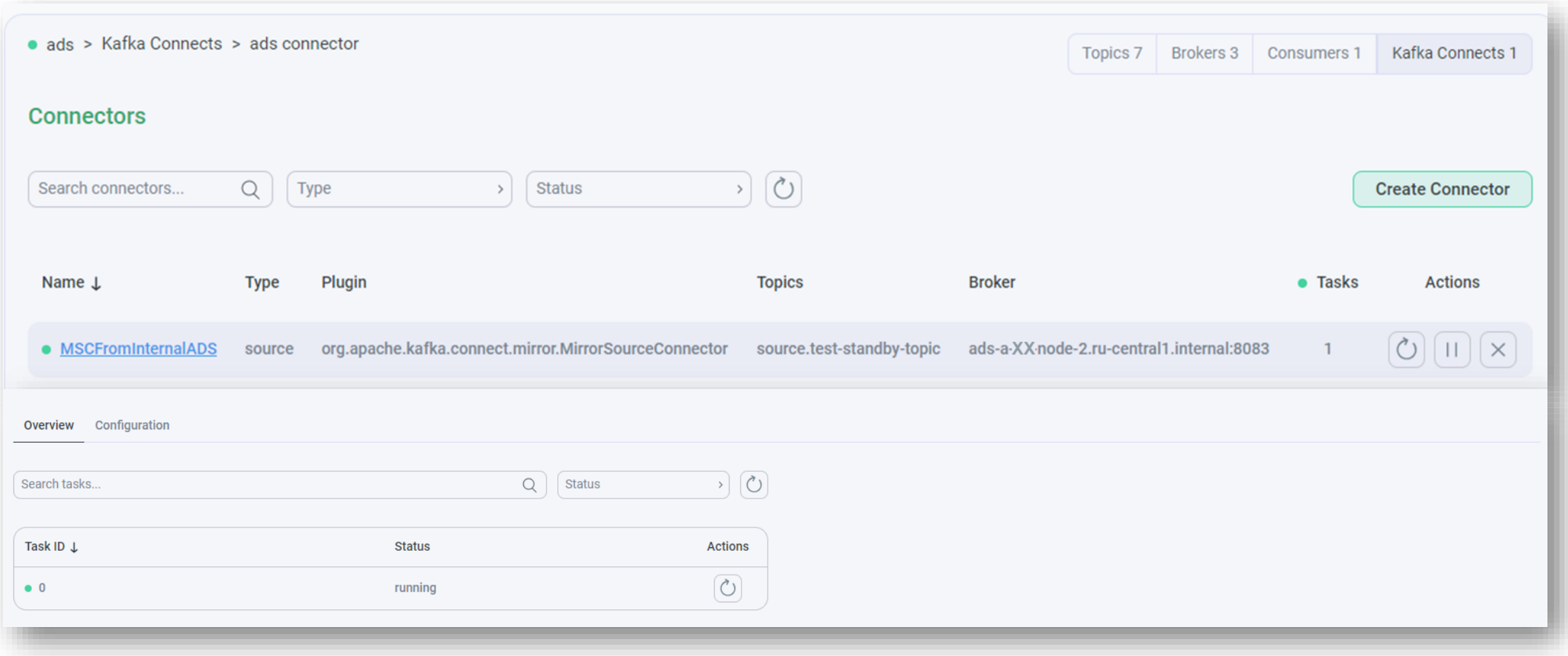
Mirror Maker 2 в ADS Control

- Изменить/добавить свойства коннектора MSCFromInternalADS в формате JSON и нажать кнопку Save:

```
"name": "MSCFromInternalADS",
"source.cluster.bootstrap.servers": "ads-mm2-00-node-1.ru-central1.internal:9092,ads-mm2-00-node-2.ru-central1.internal:9092,ads-mm2-00-node-3.ru-central1.internal:9092",
"target.cluster.bootstrap.servers": "ads-a-XX-node-1.ru-central1.internal:9092,ads-a-XX-node-2.ru-central1.internal:9092,ads-a-XX-node-3.ru-central1.internal:9092",
"target.cluster.alias": "target",
"source.cluster.alias": "standby",
"topics": "test-active-topic<XX>"
```

Mirror Maker 2 в ADS Control

- В случае успешной настройки конфигурации коннектора появится статус Online Connect:



Mirror Maker 2 в ADS Control

- Выведем содержимое топика standby.test-active-topic<XX> на кластере **ads-a**:

```
/usr/lib/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic standby.test-active-topic --from-beginning
one
two
three
```

- Добавим новое сообщение в топик test-active-topic на кластере **ads-mm**:

```
/usr/lib/kafka/bin/kafka-console-producer.sh --topic test-active-topic<XX> --bootstrap-server 10.128.0.100:9092
>one
>two
>three
>four
```

- Считываемые сообщения в целевом(**ads-a**) кластере полностью повторяют записанные в исходном(**ads-mm**) кластере:

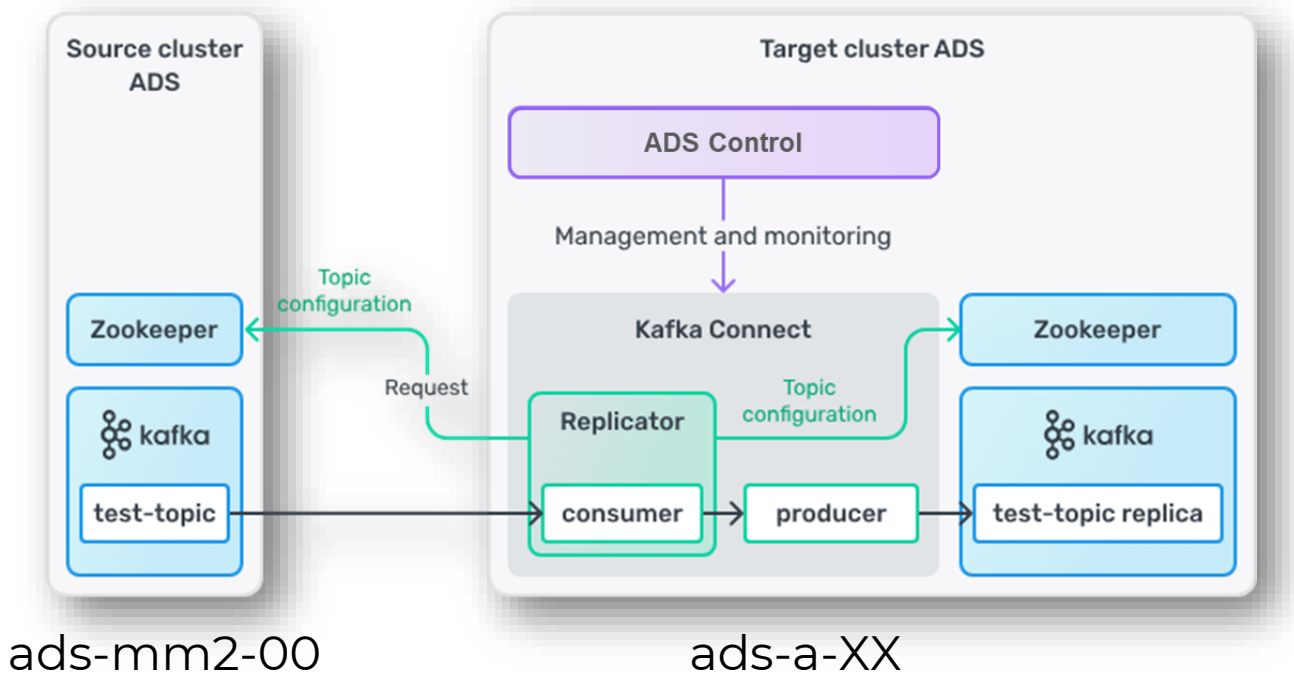
```
/usr/lib/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic standby.test-active-topic<XX> --from-beginning
one
two
three
four
```

Лабораторная работа

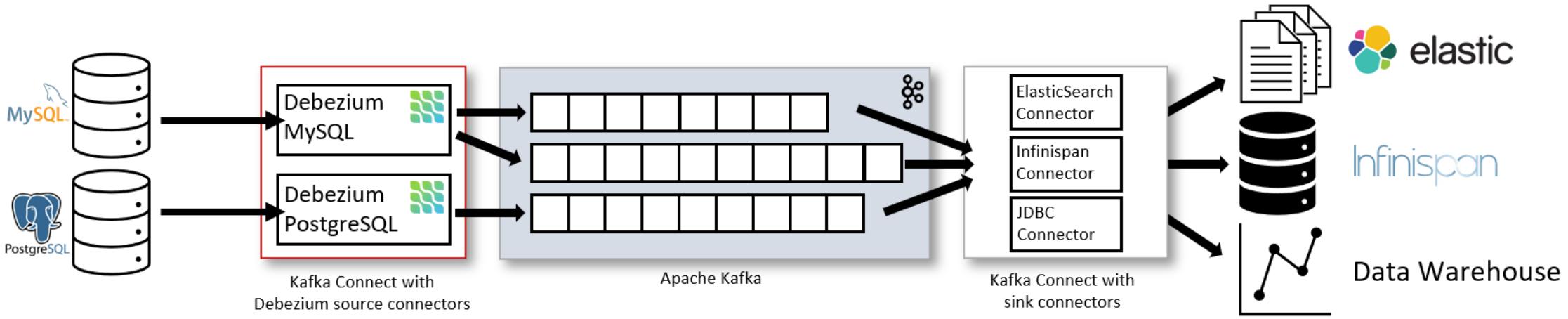
- Настроить коннектор MSCFromInternalADS для топика test-active-topic:

Исходный кластер: ads-mm2-00-node-1.ru-central1.internal:9092,ads-mm2-00-node-2.ru-central1.internal:9092,ads-mm2-00-node-3.ru-central1.internal:9092",

Целевой кластер: "ads-a-XX-node-1.ru-central1.internal:9092,ads-a-XX-node-2.ru-central1.internal:9092,ads-a-XX-node-3.ru-central1.internal:9092"

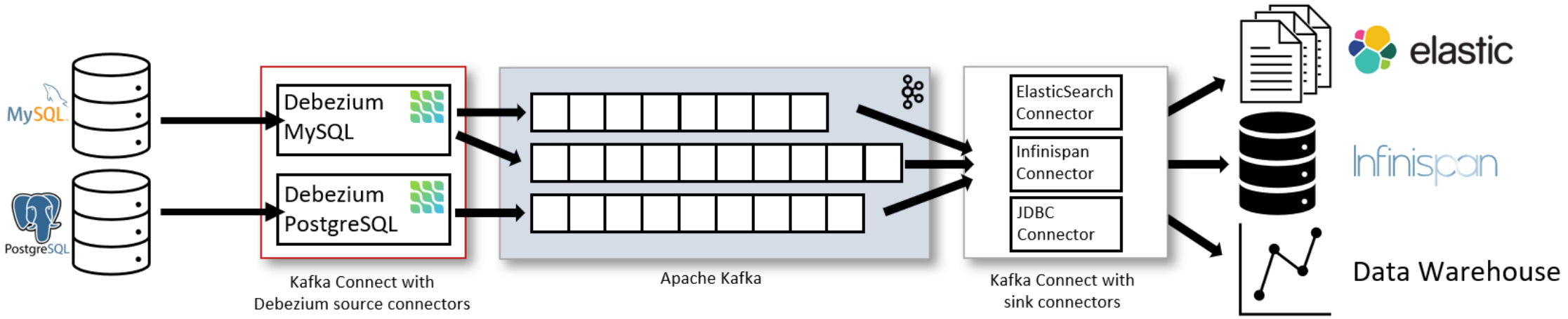


Debezium Architecture



- Debezium — это открытый программный проект, который обеспечивает change data capture (CDC) для СУБД.
- Набор коннекторов для различных СУБД, совместимых с фреймворком Apache Kafka Connect.
- Может отслеживать и передавать изменения, происходящие в СУБД, в реальном времени, что позволяет системам и приложениям реагировать на эти изменения немедленно.
- Это Open Source-проект, использующий лицензию Apache License v2.0 и спонсируемый компанией Red Hat.
- Разработка ведётся с 2016 года и на данный момент в нем представлена официальная поддержка следующих СУБД: MySQL, PostgreSQL, MongoDB, SQL Server, Oracle...

Debezium. Настройка



- **Установите Kafka (Brocker, Connect)**: Debezium использует Apache Kafka для передачи сообщений о изменениях.
- **Добавьте коннекторы Debezium**: Скачайте соответствующие коннекторы Debezium для вашей СУБД и добавьте их в Kafka Connect.
- **Настройте коннекторы**: Создайте файл конфигурации для каждого коннектора Debezium, указав параметры подключения к СУБД: URL, имя пользователя и пароль, а также параметры захвата изменений, ….
- **Запустите Kafka Connect** для запуска коннекторов Debezium с вашей конфигурацией.
- **Проверьте поток данных**: После запуска коннекторов проверьте, что изменения в СУБД корректно отражаются в Kafka.
- **Опционально. Установить Schema Registry и настроить использование схем в коннекторе**
- **Опционально. Настроить сериализацию данных (AVRO) в коннекторе**

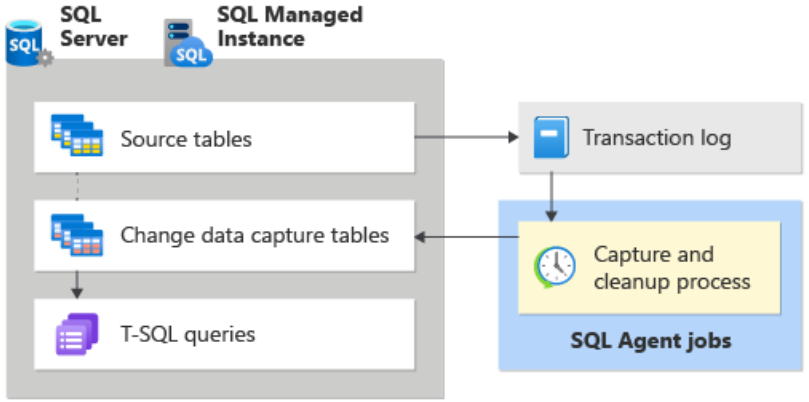
Debezium. Пример использования для MS SQL Server

- Реализуем механизм CDC для схемы:

- Debezium SQL Server фиксирует изменения на уровне строк, которые происходят в схемах базы данных SQL Server
- Версии SQL Server: *Database:* 2017, 2019, 2022, *Driver:* 12.4.2.jre8 (для версии Debezium 1.9 и выше)

Принцип работы:

- **Захват изменений**: Debezium использует функцию SQL Server под названием **Change Data Capture (CDC)**, которая должна быть включена для отслеживаемых таблиц и баз данных. CDC записывает capture row-level INSERT, UPDATE, and DELETE в специальные таблицы журналов изменений.
- **Чтение журналов**: Коннектор Debezium читает журналы изменений и преобразует записи в сообщения Kafka.
- **Отправка сообщений**: Эти сообщения затем отправляются в топики Kafka, соответствующие каждой таблице базы данных.
- **Потребление данных**: Приложения и сервисы читать топики Kafka изменений в Real-time.



CDC MS SQL Server

Debezium. Пример использования для MS SQL Server

- Debezium SQL Server для создания текущего состояния СУБД, использует процесс snapshot. Первоначальный снимок фиксирует структуру и данные таблиц в базе данных.

Workflow Debezium SQL Server Snapshot (по умолчанию, используется initial snapshot):

1. Устанавливается соединение с СУБД.
2. Определяются таблицы, которые необходимо захватить. По умолчанию соединитель захватывает все не системные таблицы (для фильтрации используются свойства table.include.list и/или table.exclude.list).
3. Включает блокировку таблиц SQL Server для CDC (snapshot.isolation.mode: repeatable_read по умолчанию. Лучше snapshot или exclusive).
4. Читает позицию максимального порядкового номера журнала (LSN) в журнале транзакций сервера.
5. Фиксирует структуру всех таблиц, предназначенных для записи. Соединитель сохраняет эту информацию в своей теме истории схемы внутренней базы данных. История схемы предоставляет информацию о структуре, которая действует при возникновении события изменения.
6. При необходимости снимаем lock's, полученные на шаге 3.
7. В позиции LSN, считанной на шаге 4, соединитель сканирует таблицы, которые необходимо захватить. В ходе проверки коннектор выполняет следующие этапы:
    a) Подтверждает, что таблица была создана до начала создания snapshot. Если таблица была создана после snapshot, соединитель пропускает таблицу.
    b) Создает событие read для каждой строки, полученной из таблицы. Все read-события содержат одну и ту же позицию LSN, которая получена на шаге 4.
    c) Выдает каждое read-событие в тему Kafka для таблицы.
8. Записывает успешное завершение snapshot в connector offsets.

- Из этого базового состояния коннектор фиксирует последующие изменения.
- После перезапуска коннектора возобновляется потоковая передача изменений с того места, где он ранее остановился на основе номера в LSN.

snapshot.mode: always, initial(default), initial_only, no_data, ...

Debezium. Пример использования для MS SQL Server

Настройка Debezium и MS SQL Server:

- Скачать коннектор (нужной версии и для нужной Java версии).
- Добавить в сервис Kafka Connect (при необходимости добавить зависимости)
- **Включите CDC** в SQL Server для таблиц, которые вы хотите отслеживать (stored procedure sys.sp_cdc_enable_db, stored procedure sys.sp_cdc_enable_table для каждой таблицы).
- **Настройте коннектор Debezium** для SQL Server, указав параметры подключения и таблицы для отслеживания.
- **Запустите коннектор** через Kafka Connect, чтобы начать передачу изменений в Kafka.

Инструкция по настройке debezium documentation sqlserver

Debezium. Пример использования для MS SQL Server

Установка MS SQL Server (используется docker-образ, который будет развернут на узле **ads-a-XX-et**):

```
sudo docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=ads@2015" -e "MSSQL_AGENT_ENABLED=true" -p 1433:1433 --name mssql --hostname sql1
-d mcr.microsoft.com/mssql/server:2019-latest
# Get SQL Script intit_db.sql from https://github.com/debezium/debezium-examples/blob/main/monitoring/inventory.sql
# Deploy init_db.sql for Docker Container
sudo docker cp /tmp/init_db.sql mssql:/tmp/init_db.sql
sudo docker exec -it mssql bash -c '/opt/mssql-tools/bin/sqlcmd -U sa -P $MSSQL_SA_PASSWORD  -i /tmp/init_db.sql'
# Check Complete DDL
sudo docker exec -it mssql bash -c '/opt/mssql-tools/bin/sqlcmd -U sa -P $MSSQL_SA_PASSWORD'
SELECT TABLE_NAME FROM testDB.INFORMATION_SCHEMA.TABLES;
GO
TABLE_NAME
------------------------------------------------------------------------------------------------------------------------------------
systranschemas
change_tables
ddl_history
lsn_time_mapping
captured_columns
…
```

Debezium. Пример использования для MS SQL Server

Настройка Kafka Connect:

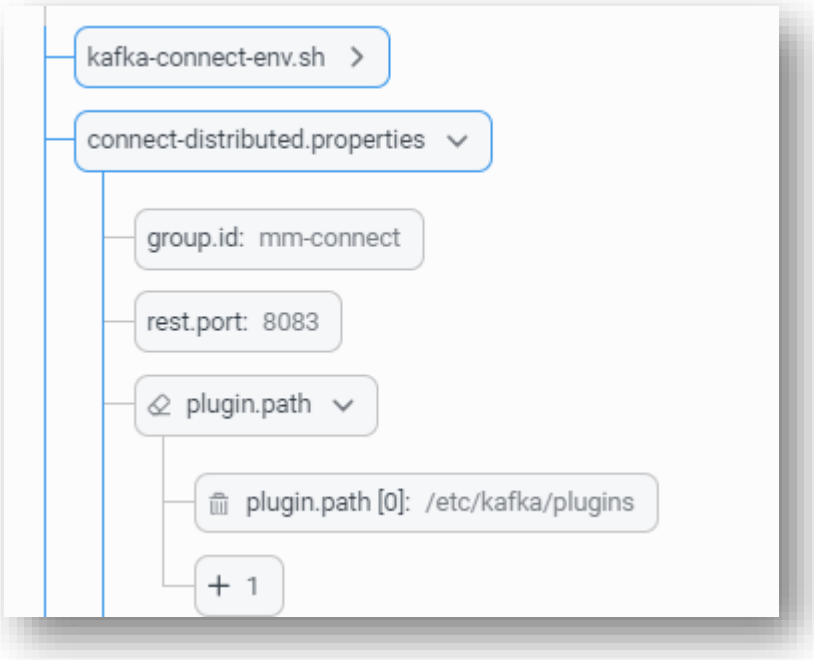
- Необходимо добавить директорию для коннектора в сервисе ADCM:

ADCM -> ADS -> SERVICES -> KAFKA CONNECT -> Configuration:

connect-distributed.properties:

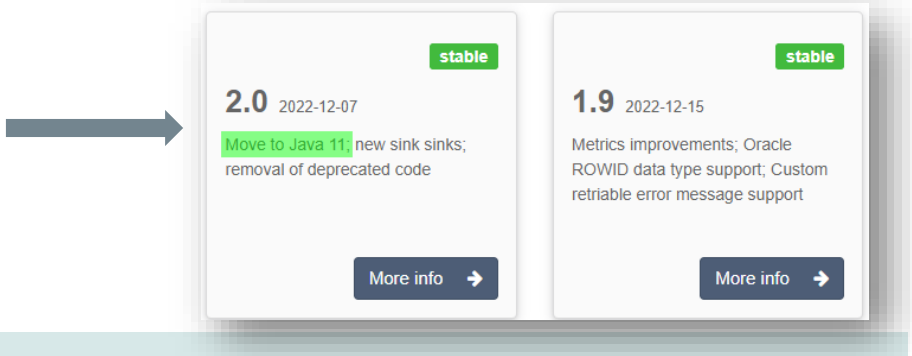
plugin.path:

/etc/kafka/plugins



Debezium. Пример использования для MS SQL Server

https://debezium.io/releases/



Настройка Kafka Connect:

- Скопируем основные зависимости в директорию с kafka plugins:
  - Коннектор Debezium sqlserver-1.9.8 (версия выбирается в соответствии с версией java 1.8)
  - Зависимости для работы с AVRO. Поменяем формат сообщения с JSON на AVRO.
  - Зависимости для работы со Schema Registry
- На всех узлах Kafka Connect Worker кластера подготовим окружение:

```
sudo mkdir -p /etc/kafka/plugins/debezium
curl -L -o /tmp/debezium-connector-sqlserver-1.9.8.Final-plugin.tar.gz  "https://repo1.maven.org/maven2/io/debezium/debezium-connector-
sqlserver/1.9.8.Final/debezium-connector-sqlserver-1.9.8.Final-plugin.tar.gz"
tar -xvf /tmp/debezium-connector-sqlserver-1.9.8.Final-plugin.tar.gz -C /tmp
sudo cp /tmp/debezium-connector-sqlserver/* /etc/kafka/plugins/debezium/
sudo cp /usr/share/java/kafka-serde-tools/*avro*.jar /etc/kafka/plugins/debezium/
sudo cp /usr/share/java/kafka-serde-tools/kafka*schema*.jar /etc/kafka/plugins/debezium/
sudo cp /usr/lib/schema-registry/libs/kafka-schema-registry*jar /etc/kafka/plugins/debezium/
sudo chmod 755 /etc/kafka/plugins/debezium/*.jar
```

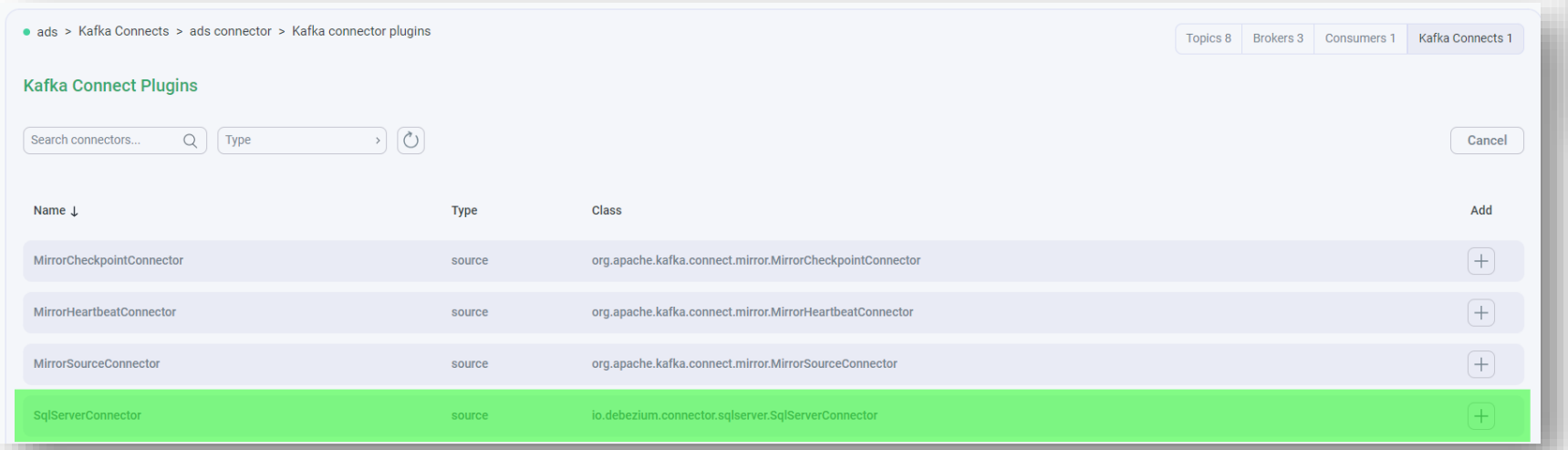
Debezium. Пример использования для MS SQL Server

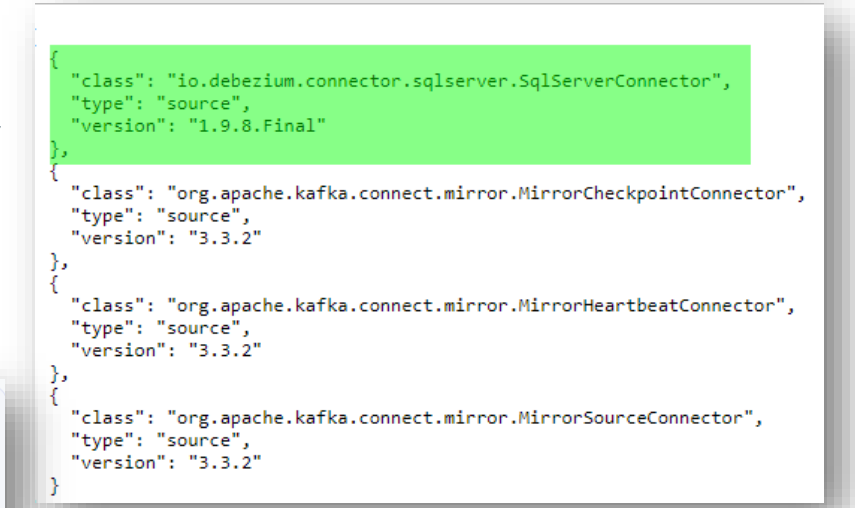
Настройка Kafka Connect:

- Restart Kafka Connect Service
- Проверим установленные коннекторы:

REST API Kafka Connect: http://ads-a-XX-node-1:8083/connector-plugins

ADS Control: ads -> Kafka Connects -> ads connector -> Kafka connector plugins





```
{
  "class": "io.debezium.connector.sqlserver.SqlServerConnector",
  "type": "source",
  "version": "1.9.8.Final"
},
{
  "class": "org.apache.kafka.connect.mirror.MirrorCheckpointConnector",
  "type": "source",
  "version": "3.3.2"
},
{
  "class": "org.apache.kafka.connect.mirror.MirrorHeartbeatConnector",
  "type": "source",
  "version": "3.3.2"
},
{
  "class": "org.apache.kafka.connect.mirror.MirrorSourceConnector",
  "type": "source",
  "version": "3.3.2"
}
```

Debezium. Пример использования для MS SQL Server

Настройка ADS Control:

- Добавим новый коннектор SqlServerConnector со след. настройками:

```
{
  "connector.class": "io.debezium.connector.sqlserver.SqlServerConnector",
  "transforms.unwrap.delete.handling.mode": "rewrite",
  "tasks.max": "1",
  "database.history.kafka.topic": "schema-changes.mssql-inventory",
  "transforms": "unwrap",
  "output.data.format": "AVRO",
  "topic.prefix": "mssqlserver",
  "schema.history.internal.kafka.topic": "schema-changes.mssql-inventory",
  "transforms.unwrap.type": "io.debezium.transforms.ExtractNewRecordState",
  "value.converter": "io.confluent.connect.avro.AvroConverter",
  "key.converter": "io.confluent.connect.avro.AvroConverter",
  "database.encrypt": "false",
  "database.user": "sa",
  "database.names": "testDB",
…
```

+

```
…
  "topic.creation.default.replication.factor": "3",
  "topic.creation.default.partitions": "5",
  "topic.creation.enable": "true",
  "database.history.kafka.bootstrap.servers": "ads-a-XX-node-1:9092",
  "database.server.name": "ads-a-XX-et",
  "schema.history.internal.kafka.bootstrap.servers": "ads-a-XX-node-1:9092",
  "database.port": "1433",
  "key.converter.schemas.enable": "false",
  "value.converter.schema.registry.url": "http://ads-a-XX-node-3:8081",
  "database.hostname": "ads-a-XX-et",
  "database.password": "ads@2015",
  "name": "mssql",
  "value.converter.schemas.enable": "false",
  "transforms.unwrap.add.fields": "op,table,source.ts_ms,ts_ms",
  "key.converter.schema.registry.url": "http://ads-a-XX-node-3:8081"
}
```

Debezium. Пример использования для MS SQL Server

Настройка ADS Control:

- transforms.unwrap.add.fields – добавляем в сообщение ряд системных полей, используя функционал коннектора transformations. Данный встроенный в коннектор функционал позволяет налету менять сообщения перед публикацией их в топик Kafka. Воспользуемся типом трансформации New Record State Extraction и добавим в сообщение ряд системных полей: op, table, lsn, source.ts_ms, ts_ms
- [key, value].converter.schema.registry – настройка для использования Schema Registry
- После запуска коннектора проверить статус:

http://ads-a-XX-node-1:8083/connectors/mssql/status

- Проверить топики:

http://ads-a-XX-node-1.ru-central1.internal:9000/clusters/ADS/topics

```
…
  "topic.creation.default.replication.factor": "3",
  "topic.creation.default.partitions": "5",
  "topic.creation.enable": "true",
  "database.history.kafka.bootstrap.servers": "ads-a-XX-node-1:9092",
  "database.server.name": "ads-a-XX-et",
  "schema.history.internal.kafka.bootstrap.servers": "ads-a-XX-node-1:9092",
  "database.port": "1433",
  "key.converter.schemas.enable": "false",
  "value.converter.schema.registry.url": "http://ads-a-XX-node-3:8081",
  "database.hostname": "ads-a-XX-et",
  "database.password": "ads@2015",
  "name": "mssql",
  "value.converter.schemas.enable": "false",
  "transforms.unwrap.add.fields": "op,table,source.ts_ms,ts_ms",
  "key.converter.schema.registry.url": "http://ads-a-XX-node-3:8081"
}
```

Debezium. Пример использования для MS SQL Server

Настройка ADS Control:

- Проверить созданные схемы: http://ads-a-XX-node-3:8081/schemas

```
...
{
    "subject": "ads-a-XX-et.testDB.dbo.products-key",
    "version": 1,
    "id": 8,
    "schema": "{\"type\":\"record\",\"name\":\"Key\",\"namespace\":\"ads_a_XX_et.testDB.dbo.products\",\"fields\":[{
}
```

- Проверить сообщения в топике:

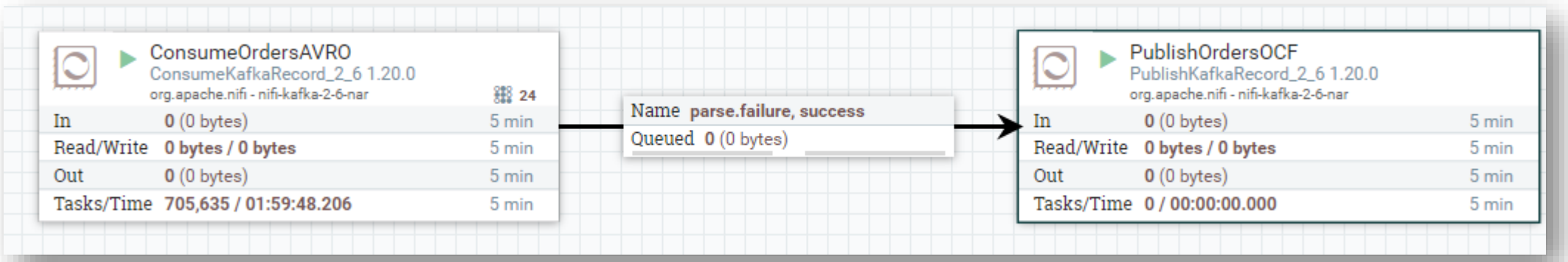
/usr/bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --topic ads-a-XX-et.testDB.dbo.products --from-beginning --property schema.registry.url=http://ads-a-**XX**-node-3:8081

```
{
    "id": 104,
    "name": "hammer",
    "description": {
        "string": "12oz carpenter's hammer"
    },
    "weight": {
        "double": 0.75
    },
    "__op": {
        "string": "r"
    },
    "__table": {
        "string": "products"
    },
    "__source_ts_ms": {
        "long": 1718566298650
    },
    "__ts_ms": {
        "long": 1718566298656
    },
    "__deleted": {
        "string": "false"
    }
}
..
```

Debezium. Пример использования для MS SQL Server

Настройка NiFi:

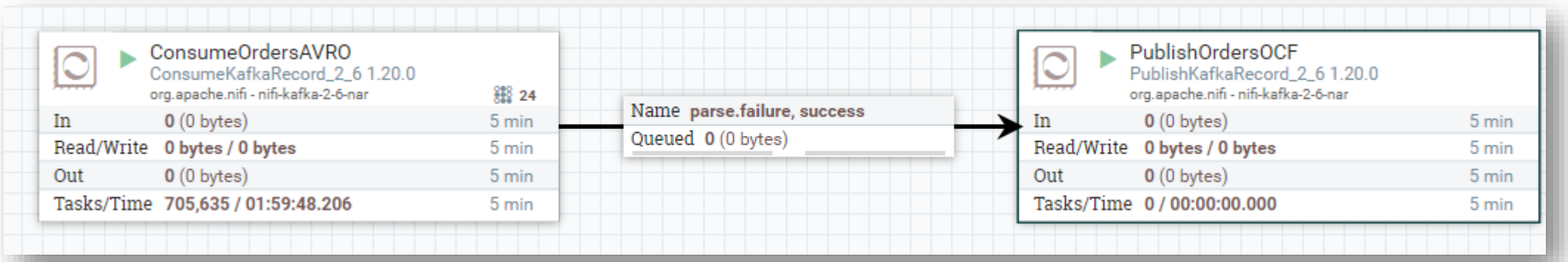
- Для загрузки сообщений из Kafka в ADB используется коннектор Kafka2ADB из состава Enterprise версии ADB.
- Коннектор читает AVRO сообщения только в OCF формате (когда схема записывается в само сообщение).
- Debezium-коннектор пишет AVRO сообщения со ссылкой на Schema Registry.
- Необходимо реализовать преобразования в AVRO OCF, с помощью NiFi Flow



Debezium. Пример использования для MS SQL Server

Настройка NiFi:

- Для загрузки сообщений из Kafka в ADB используется коннектор Kafka2ADB из состава Enterprise версии ADB.
- Коннектор читает AVRO сообщения только в OCF формате (когда схема записывается в само сообщение).
- Debezium-коннектор пишет AVRO сообщения со ссылкой на Schema Registry.
- Необходимо реализовать преобразования в AVRO OCF, с помощью NiFi Flow



Debezium. Пример использования для MS SQL Server

Настройка ADB:

- Создать Сервер kafka_server
- Создать Foreign-таблицу kafka_table (структура таблицы должна соответствовать используемой AVRO-схеме)
- Добавить данные в MS SQL Server и проверить данные в Foreign-таблице ADB.

Microsoft SQL Server

```
1> INSERT INTO products(name,description,weight) VALUES ('MyProduct1','Small',100.14);
2> go

(1 rows affected)
1> INSERT INTO products(name,description,weight) VALUES ('MyProduct10','Small',100.14);
2> go

(1 rows affected)
1>
```

AD.B

```
CREATE FOREIGN TABLE
adb=# SELECT * FROM ka_products_cdc;
NOTICE:  Kafka-ADB: Offset for partition 0 is not known, and is set to default value 0  (seg0 slice1 10.128.0.26:10000 pid=16599)
NOTICE:  Kafka-ADB: Offset for partition 1 is not known, and is set to default value 0  (seg0 slice1 10.128.0.26:10000 pid=16599)
NOTICE:  Kafka-ADB: Offset for partition 2 is not known, and is set to default value 0  (seg1 slice1 10.128.0.26:10001 pid=16598)
NOTICE:  Kafka-ADB: Offset for partition 3 is not known, and is set to default value 0  (seg2 slice1 10.128.0.65:10000 pid=22118)
NOTICE:  Kafka-ADB: Offset for partition 4 is not known, and is set to default value 0  (seg3 slice1 10.128.0.65:10001 pid=22119)
 id  |    name     | description | weight | __op | __table  | __source_ts_ms |     __ts_ms     | __deleted
-----+-------------+-------------+--------+------+----------+----------------+-----------------+-----------
 115 | MyProduct10 | Small       | 100.14 | c    | products |  1718575440640 | 1718575442175 | false
 114 | MyProduct1  | Small       | 100.14 | c    | products |  1718572921443 | 1718572926173 | false
(2 rows)
```

AD