EVD Platform V Synapse Event Transfer Service

EVTD Сервис передачи сообщений

Руководство по эксплуатации

ОГЛАВЛЕНИЕ

Руководство по эксплуатации

Термины и сокращения

Термин/Аббревиатура	Определение
УЦ	Удостоверяющий центр
Jenkins	Программная система для обеспечения процесса непрерывной интеграции программного обеспечения
mTLS	Mutual Transport Layer Security — криптографический протокол взаимной (двусторонней) аутентификацией клиента и брокера, который обеспечивает безопасность транспортного уровня (Transport Layer Security)
TLS-сертификат	Transport Layer Security — криптографический протокол защиты транспортного уровня, предназначенный для защиты обмена данными в сети
CLI	Command Line Interface — интерфейс командной строки

Руководство прикладного разработчика

Системные требования

Зависят от выбранного языка реализации клиентской библиотеки.

Подключение и конфигурирование

Перед разработкой необходимо подключить к проекту необходимые зависимости:

Для Java: `org.apache.kafka:kafka-clients:2.4.0`

Для Python: kafka-python

Для C/C++: librdkafka

Для Go(GoLang): sarama

Для .Net: confluent-kafka-dotnet

Миграция на текущую версию

Общий подход

Сохраняется совместимость с предыдущими версиями.

Разработка первого приложения с использованием программного продукта

Для передачи событий через Kafka необходимо реализовать отправку и получение на стороне клиентов: продюсер на стороне отправителя и консьюмер на стороне получателя.

Публикация событий

Публикация событий - процесс отправки сообщений в определенный топик определенного кластера Apache Kafka.

Пререквизиты:

1. Подготовить клиентский сертификат для подключения приложения к Apache Kafka.
2. Определить кластер Apache Kafka, в который будет производиться запись.
3. Определить наименование топика, в который будут отправляться события в соответствии с правилом наименования: <индентификатор системы-поставщика>.<тип события>EVENT.V<версия формата события>

AS1.ENTITYCHANGEDEVENT.V1

1. Запросить у администратора доступа создание топика с получившимся наименованием на кластере и выдачу на него прав на запись для полученного сертификата.

Шаги разработки:

Примеры кода приводятся для языка Java.

1. Подключить в зависимости проекта библиотеку `"org.apache.kafka:kafka-clients:2.4.0"`.
2. В классе произвести заполнение параметров подключения:

```
Properties props = new Properties();
props.put("bootstrap.servers", "192.168.1.1:9093,192.168.1.2:9093,192.168.1.3:9093");
props.put("acks", "all");
```

```
props.put("batch.size", 16384);
props.put("linger.ms", 1);
props.put("buffer.memory", 33554432);
props.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");
props.put("security.protocol", "SSL");
props.put("ssl.keystore.location", "<полный путь до хранилища ключа>" );
props.put("ssl.truststore.location", "<полный путь до хранилища ключа>");
props.put("ssl.keystore.password", "<пароль хранилища ключа>");
props.put("ssl.truststore.password", "<пароль хранилища сертификата ЦС>");
props.put("ssl.enabled.protocols", "TLSv1.2");
props.put("ssl.key.password", "<пароль закрытого ключа>");
props.put("ssl.endpoint.dentification.algorithm", "");
```

где

- `bootstrap.servers` - список серверов кластера Apache Kafka с указанием портов через запятую;
- `acks` - необходимое количество ответов от брокеров Apache Kafka, для того чтобы считать запись успешной.
  Возможные значения:
  `0 - ответ не ожидается;`

  `1 - достаточно ответа от брокера-лидера;`

  `all - ожидается ответ от всех реплик что они получили сообщение.`
- `batch.size` - размер пачки в байтах, в которые будут группироваться события в случае отправки нескольких событий.
- `linger.ms` - количество миллисекунд, которое producer ждет получения возможных новых событий для пачки перед отправкой.
- `buffer.memory` - общее количество памяти выделяем для буферизации отправляемых событий.
- `key.serializer` и `value.serializer`- классы-сериализатор ключа и значения переданных в ProducerRecord. В комплекте с клиентом идут два стандартных класса: ByteArraySerializer и StringSerializer.

1. Добавить непосредственно отправку события:

```
Producer<String, String> producer = new KafkaProducer<>(props);
ProducerRecord<String,String> record = new ProducerRecord<String,String>("my-topic", key, value);
producer.send(record);
producer.close();
```

где

- `props` - предзаполненные параметры подключения;
- `my-topic` - наименование топика в который производится отправка;

`key` - ключ события;
`value` - текст события.
Непосредственно отправка возможна одним из трех вариантов:

- **Блокирующий синхронный, с ожиданием ответа об успешной отправке:**
  `producer.send(record).get();`
- **Асинхронный без ожидания ответа:**
  `producer.send(record)`
- **Асинхронный с ожиданием ответа:**
  `producer.send(record, new Callback() { public void onCompletion(RecordMetadata metadata, Exception e) { if(e != null) { e.printStackTrace(); } else { System.out.println("The offset of the record we just sent is: " + metadata.offset()); } } });`

Подписка на события

Подписка на события - процесс получения сообщений из определенного топика определенного кластера Apache Kafka.

Пререквизиты:

1. Подготовить клиентский сертификат для подключения приложения к Apache Kafka.
2. Определить кластер Apache Kafka, в который публикуются события, к которым необходимо получить доступ.
3. Определить наименование топика, в который публикуются события, к которым необходимо получить доступ.
4. Запросить у администратора доступа выдачу прав на подписку к указанному топику для полученного сертификата.

Шаги разработки:

Примеры кода приводятся для языка Java.

1. Подключить в зависимости проекта библиотеку `org.apache.kafka:kafka-clients:2.4.0`.
2. В классе произвести заполнение параметров подключения:

```
Properties props = new Properties();
props.put("bootstrap.servers", "192.168.1.1:9093,192.168.1.2:9093,192.168.1.3:9093");
props.put("group.id", "test");
```

```
props.put("enable.auto.commit", "true");
props.put("auto.commit.interval.ms", "1000");
props.put("key.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("value.deserializer", "org.apache.kafka.common.serialization.StringDeserializer");
props.put("security.protocol", "SSL");
props.put("ssl.keystore.location", "<полный путь до хранилища ключа>" );
props.put("ssl.truststore.location", "<полный путь до хранилища ключа>");
props.put("ssl.keystore.password", "<пароль хранилища ключа>");
props.put("ssl.truststore.password", "<пароль хранилища сертификата ЦС>");
props.put("ssl.enabled.protocols", "TLSv1.2");
props.put("ssl.key.password", "<пароль закрытого ключа>");
props.put("ssl.endpoint.dentification.algorithm", "");
```

где

- `bootstrap.servers` - список серверов кластера Apache Kafka с указанием портов через запятую;
- `group.id` - наименования группы потребителя, все потребители в рамках одной группы считаются одним потребителем;
- `enable.auto.commit` - установка в true означает что обработанные оффсеты коммитятся автоматически, интервал коммита устанавливается в параметре `auto.commit.interval.ms`;
- `key.deserializer` и `value.deserializer`- класс-десериализатор ключа и значения полученных в ConsumerRecords. В комплекте с клиентом идут два стандартных класса: *ByteArrayDeserializer* и *StringDeserializer*.

1. Добавить подключение к кластеру:

```
KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);
consumer.subscribe(Arrays.asList("my-topic", "your-topic"));
```

где

- `props` - предзаполненные параметры подключения;
- `my-topic`, `your-topic` - наименование топиков к которым производится подключение.

1. Добавить непосредственно вычитку:

```
while (true) {
    ConsumerRecords<String, String> records = consumer.poll(100);
    for (ConsumerRecord<String, String> record : records)
        System.out.printf("offset = %d, key = %s, value = %s%n", record.offset(), record.key(), record.value());
```

```
}
```

Использование программного продукта

Основной функционал продукта - предоставление транспортного слоя и возможностей отправки и получения событий.

Дополнительно предоставляется функционал классов-перехватчиков, которые отрабатывают перед передачей управления в основной код, и входят в стандартные механизмы подключения к Apache Kafka.

Общей задачей классов-перехватчиков служит обработка каждой исходящей/входящей записи на уровне транспорта, до передачи ее в прикладной код. Таким образом осуществляется независимая от прикладного кода функциональность транспортного слоя. Вмешательство в работу перехватчика со стороны прикладного кода, если перехватчик загружен в клиента kafka, невозможно. Только переинициализация клиента с выгрузкой перехватчика.

Часто встречающиеся проблемы и пути их устранения

Не выявлено.

Руководство оператора

Доступ к приложению

Доступ к приложению осуществляется через CLI или с помощью Jenkins.

Оператору необходимо самостоятельно определить методы защиты информации исходя из уровня ее конфиденциальности и рекомендаций информационной безопасности.

Использование приложения оператором

Сценарии использования

Предусловия

1. Наличие сертификатов TLS для администратора доступа. Перед началом действий поместить сертификаты в домашнюю директорию на сервере, где расположен продукт EVTD.
2. На сервере, где расположен EVTD, создать файл **ssl_adm.properties**. Наполнить рекомендуемыми значениями:

```
security.protocol = SSL;
ssl.keystore.location – полный путь до хранилища ключа.jks;
ssl.truststore.location – полный путь до хранилища ключа.jks;
ssl.keystore.password – пароль хранилища ключа;
ssl.truststore.password – пароль хранилища сертификата ЦС;
ssl.enabled.protocols=TLSv1.2;
ssl.key.password – пароль закрытого ключа;
ssl.endpoint.dentification.algorithm.
```

Создание новых потоков событий

На сервере, где расположен EVTD, выполнить команду:

```
/opt/Apache/kafka/bin/kafka-topics.sh --bootstrap-server hostname -f:<порт> --create --command-config ~/ssl_adm.properties --topic
<наименование потока> --partitions <количество партиций> --replication-factor <фактор репликации> --config retention.ms=<сколько хранить
события> --config retention.bytes=<максимальный размер партиции> --config max.message.bytes=<максимальный размер сообщения>
```

где `topic` - наименование потока событий в соответствии с правилом наименования: <индентификатор системы-поставщика>.<тип события>EVENT.V<версия формата события>
Например: AS1.ENTITYCHANGEDEVENT.V1
`partitions` - количество партиций, рассчитывается исходя из предполагаемой нагрузки; `replication-factor` - фактор репликации, отвечает за отказоустойчивость потока, обычно указывается как количество брокеров Apache Kafka - 1; `config retention.ms` - время хранения событий в потоке в миллисекундах, по истечении данного времени события будут удалены; `config retention.bytes` - максимальный размер партиции в байтах, при его превышении самые старые события будут удалены; `config max.message.bytes` -

максимальный размер события в потоке в байтах. При попытке публикации события, превышающего данное значение, будет выдана ошибка.

Назначение прав на публикацию событий

На сервере, где расположен EVTD, выполнить команду:

```
/opt/Apache/kafka/bin/kafka-acls.sh --bootstrap-server hostname -f:9093 --add --allow-principal <User:полное наименование сертификата (DN) без пробелов при разделении категорий> --producer --topic <наименование потока> --command-config ~/ssl_adm.properties
```

Назначение прав на подписку на события

На сервере, где расположен EVTD, выполнить команду:

```
/opt/Apache/kafka/bin/kafka-acls.sh --bootstrap-server \hostname -f:9093 --add --allow-principal <User:полное наименование сертификата (DN) без пробелов при разделении категорий> --consumer --topic <наименование потока> --group <наименование consumerGroup потребителя> --command-config ~/ssl_adm.properties
```

Часто встречающиеся проблемы и пути их устранения

Тип ошибки	Причина	Возможное решение
Ошибка аутентификации - SSL handshake failed	1. Не корректно сформировано хранилище сертификатов 2. Отсутствует параметр ssl.endpoint.identification.algorithm=	1. Получить вывод команды keytool -v -list -keystore <имя файла кейстора>, проверить в выводе, что в keystore есть PrivateKeyEntry и это именно тот сертификат, для которого выдавались права, а также что

Тип ошибки	Причина	Возможное решение
		данный private key entry подписан правильным Удостоверяющим центром (далее - УЦ) (issuer) 2. Убедиться, что параметр ssl.endpoint.identification.algorithm= установлен в конфигурации клиента
Ошибка авторизации - Not authorized to access topics	1. Используется некорректный Distinguished Name (далее - DN) сертификата 2. Выполняется подключение не к тем брокерам Kafka. 3. Клиент подключается как продьюсер, а должен как консьюмер. И наоборот	1. Необходимо сверить DN сертификата с которым производится подключение, с тем который указывали при выдаче прав. Проверить отсутствие пробелов, русских букв 2. Проверить настройки подключения (bootstrap.servers) 3. Проверить корректность подключения и сверить с тем, какие права выдавались
Ошибка авторизации - Not authorized to	Используется некорректная группа консьюмеров.	Необходимо сверить наименование consumer group в настройках с той, для которой

Тип ошибки	Причина	Возможное решение
access group		выдавались права
Topic <topic name> not present in metadata after 60000 ms	Топик не существует	Проверить наименование топика и корректность указанных брокеров Kafka
Connection to node ... terminated during authentication	Некорректная настройка окружения	1. Проверить настройки Firewall 2. Проверить настройки service entry при обращении из OpenShift
Group coordinator ... is unavailable or invalid, will attempt rediscovery	Не доступен координатор группы	1. Проверить что брокер-координатор не остановился 2. Проверить сетевой доступ

Параметры настройки

Настройки продукта задаются администратором и указаны в руководстве по установке.

Правила эксплуатации

Дополнительные правила эксплуатации отсутствуют.

Классы-перехватчики

Общей задачей классов-перехватчиков служит обработка каждой исходящей/входящей записи на уровне транспорта, до передачи ее в прикладной код. Таким образом, осуществляется независимая от прикладного кода функциональность транспортного слоя.

Вмешательство в работу перехватчика со стороны прикладного кода, если перехватчик загружен в клиента Kafka, невозможно. Только переинициализация клиента с выгрузкой перехватчика.

Перехватчики, обеспечивающие фильтрацию и валидацию событий

Разные системы имеют разный уровень доступа к данным. Для предотвращения доступа к данным, не предназначенным для системы, можно использовать соответствующие перехватчики.

В таких случаях перехватчик - это модуль устанавливаемый на стороне потребителя, предназначен для валидации сообщений, а также для фильтрации полей, которые не предназначены для данного потребителя.

Запуск/остановка перехватчика фиксируется в логах потребителя, что позволяет определить использовался ли перехватчик потребителем при чтении событий.

На данный момент разработаны несколько видов интерсепторов:

1. Тип `VALIDATOR` (для валидации сообщений):
    - **validator-interceptor** - позволяет валидировать сообщения в форматах json/xml/avro по схемам, указанным в конфигурационном файле.
2. Тип `SIGNATURE` (для контроля целостности сообщений):
    - **kafka-certificate-signature-interceptor** - предназначен для подписания сообщения сертификатом.

3. Тип LATENCY:
    - **kafka-timestamp-interceptor** - используется для замера задержки (latency) при прохождении через событийный сегмент.
4. Тип OTHER (другие):
    - **dsl-interceptor** - позволяет преобразовывать тело сообщений при помощи языка dsl. На вход и выход ожидается одно сообщение;
    - **json-consumer-interceptor** - модуль, который выполняет фильтрацию json-сообщений, используя jsonPath;
    - **json-extractor-interceptor** - может достать поле из сообщения в json формате и добавить его в заголовок.

Все интерсепторы собраны и протестированы на версии kafka-clients, которая указана в версии интерсептора, указанной в блоке <dependency>. Например: 2.7.1-0.0.3, где 2.7.1 - версия Apache Kafka. Работа с другими версиями возможна, но не гарантирована.

Актуальные версии интерсепторов

Для каждого интерсептора в папке дистрибутивов присутствуют следующие артефакты:

- **<имя_интерсептора>.jar** - без зависимостей, для подключения к проектам через системы сборки.
- **<имя_интерсептора>-fatjar.jar** - включает в себя все зависимости, для подключения с помощью копирования библиотеки в *classpath*.
- **<имя_интерсептора>-sources.jar** - исходный код (может потребоваться scala-плагин для IDE).

Имена/версии интерсепторов используют следующий шаблон: <имя_интерсептора>_<версия_scala>:<версия_Apache_Kafka>-<версия_интерсептора>

Только минорные версии scala совместимы между собой (например 2.13.0 и 2.13.1). Остальные версии (например 2.12 и 2.13 не совместимы).

Интерсепторы протестированы только на версии Apache Kafka, указанной в версии интерсептора, но в большинстве случаев будут работать и с более старыми версиями клиентов.

kafka-interceptors-bundle

Включает в себя все доступные интерсепторы.

Артефакт kafka-interceptors-bundle-fatjar предназначен для деплоя в директорию /libs kafka-брокера (например для использования с kafka connect), из него вырезаны все зависимости, которые уже есть в classpath'e kafka.

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>kafka-interceptors-bundle_2.13</artifactId>
  <version>2.7.1-1.1.3</version>
</dependency>
```

validator-interceptor

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>validator-interceptor_2.13</artifactId>
  <version>2.7.1-1.0.2</version>
</dependency>
```

---

json-consumer-interceptor

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>json-consumer-interceptor_2.13</artifactId>
  <version>2.7.1-0.0.5</version>
</dependency>
```

json-extractor-interceptor

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>json-extractor-interceptor_2.13</artifactId>
  <version>2.7.1-0.1.1</version>
</dependency>
```

kafka-certificate-signature-interceptor

```
<dependency>
```

```
  <groupId>ru.sbt.ss</groupId>
  <artifactId>kafka-certificate-signature-interceptor_2.13</artifactId>
  <version>2.7.1-0.2.2</version>
</dependency>
```

---

kafka-timestamp-interceptor

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>kafka-timestamp-interceptor_2.13</artifactId>
  <version>2.7.1-0.7.2</version>
</dependency>
```

dsl-interceptor

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>dsl-interceptor_2.13</artifactId>
  <version>2.7.1-0.0.5</version>
</dependency>
```

Validator-interceptor

Описание модуля

Реализует интерфейсы ConsumerInterceptor/ProducerInterceptor и позволяет валидировать сообщения в форматах *json/xml/avro* по схемам, указанным в конфигурационном файле.

- Интерсепторы позволяют использовать разные схемы для разных топиков.
- Для форматов json и xml поддерживается валидация схем.
- Интерсепторы могут работать с сообщениями типа String или Array.

Все используемые Runtime и Provided dependencies представлены в репозитории интерсептора.

Подключение к Kafka Client

Примеры конфигурации

Пример конфигурации Producer

```
...
interceptor.classes=ru.sbt.ss.kafka.validator.interceptor.ValidatorProducerInterceptor
interceptor.validator.config=path/to/validator.conf
# interceptor.validator.default.topic=*
# interceptor.validator.schema.validation.enabled=true
```

Пример конфигурация Consumer

```
...
interceptor.classes=ru.sbt.ss.kafka.validator.interceptor.ValidatorConsumerInterceptor
interceptor.validator.config=path/to/validator.conf
# interceptor.validator.default.topic=*
# interceptor.validator.mode = failOnValue
# interceptor.validator.schema.validation.enabled=true
```

Пример конфигурация файла со списком путей до схем

Для каждого топика можно задать отдельную схему, а также схему по умолчанию с именем топика. Имя топика, который будет использоваться как топик по умолчанию можно задать с помощью настройки `interceptor.validator.default.topic`. Если схема по умолчанию отсутствует и сообщение было отправлено/получено из топика, для которого не указана схема - сообщение считается НЕВАЛИДНЫМ.

```
schemas: {
 "topic-1": {
    type: "json",
    schema: "path/to/schema.json"
 }

 "topic-2": {
    type: "xml"
    schema: "path/to/schema.xml"
 }

 "DEFAULT": {
    type: "avro"
```

```
    schema: "path/to/schema.avsc"
 }
}
```

Пример схемы для валидации json-сообщения

```
{
         "rq_tm" : "2021-11-23T22:12:12",
         "event_type" : "type",
         "epk_id" : "1234",
         "client_id" : 12356712567412,
         "pan" : "pan",
         "is_main_card" : true,
         "product_way4_code" : "dkjahskdjf",
         "ips_product_name" : "product_name",
         "card_type" : "DC",
         "pprb_status" : "status",
         "reason" : "why"
}
```

Пример схемы для валидации xml-сообщения

```
<?xml version="1.0" encoding="UTF-8"?>

<shiporder orderid="889923"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:noNamespaceSchemaLocation="shiporder.xsd">
    <orderperson>John Smith</orderperson>
    <shipto>
        <name>Ola Nordmann</name>
        <address>Langgt 23</address>
        <city>4000 Stavanger</city>
        <country>Norway</country>
    </shipto>
    <item>
        <title>Empire Burlesque</title>
        <note>Special Edition</note>
        <quantity>1</quantity>
        <price>10.90</price>
    </item>
    <item>
```

```
        <title>Hide your heart</title>
        <quantity>1</quantity>
        <price>9.90</price>
    </item>
</shiporder>
```

Пример схемы для валидации avro-сообщения

```
14 66 69 72 73 74 5F 6A 73 6F 6E 0A 48 65 6C 6C 6F 02 08 31 41 42 43
```

В случае если сообщение не прошло валидацию при отправке (producer.send()) будет выброшено исключение ru.sbt.ss.kafka.interceptors.ProducerInterceptorError с сообщением:

```
Error while processing record(topic: topic): *ошибка валидации в зависимости от формата сообщения*.
```

В случае, если сообщение не прошло валидацию при получении (consumer.poll()) поведение настраивается параметром *'interceptor.validator.mode'*:

1. *failOnValue (default)* - клиент получит сообщение-заглушку вместо невалидного сообщения, которое содержит *null* вместо *value* и выбросит исключение при вызове *record.value()*.
2. *filter* - сообщение об ошибке будет залогировано в *error*, клиент не получит невалидное сообщение.
3. *failOnConsume* - метод *consumer.poll()* выбросит исключение *ru.sbt.ss.kafka.interceptors.ConsumerInterceptorError*, клиент не получит ни одного сообщения из пачки.

Валидация схем

При настройке *interceptor.validator.schema.validation.enabled=true json/xsd* схемы будут провалидированы при загрузке.

Json

Механизм валидации аналогичен валидации обычных сообщений, в качестве схемы используется json метасхема версии draft-04. Метасхема находится в /resources, при загрузке проверяется ее hash.

Метасхема дополнена следующими ограничениями:

1. Для любого объекта схемы обязательно хотя бы одно поле. (ограничивает использование пустых объектов *{}* в схеме).
2. Временно ограничено использование ссылок *$ref*.

3. Для массивов *("type": "array")* обязательны поля: *maxItems*, *additionalItems=false*, *uniqueItems=true*.
4. Для объектов *("type": "object")* обязательно поле *additionalProperties*. Поле может принимать значение *false*, либо *{"type":"string"}* с указанием *maxProperties*.
5. Для чисел (*"type": "number"* или *"type": "integer"*) обязательны поля *minimum* и *maximum*.
6. Для строк (*"type": "string"*) обязательно поле *maxLength*.
7. При указании нескольких типов для поля (*"type": ["object", "array"]*) проверяются ограничения для всех присутствующих типов.

При неуспешной валидации будет выброшено исключение *Schema '<Имя схемы>' doesn't match meta schema:* + список ошибок валидации.

Также метасхема дополнена следующими некритичными ограничениями, не влияющими на прохождение валидации:

1. Для любого объекта, имеющего тип (*type*), требуется аннотация *description*.
2. Максимальная длина строк (*"type": "string"*) не должна быть больше 250 символов (*maxLength* <= 250)

Некритичные ограничения не влияют на прохождение схемой валидации, ошибки будут залогированы в WARN.

XSD

XSD схема будет проверена на соответствие следующим ограничениям:

1. Для представления числовой информации нужно использовать ограничение *“totalDigits”* (*xs:decimal* и все типы, производные от него: *xs:integer*, *xs:negativeInteger*, *xs:nonNegativeInteger*,*xs:nonPositiveInteger*, *xs:positiveInteger*, *xs:byte*, *xs:long*, *xs:int*, *xs:short*, *xs:unsignedLong*, *xs:unsignedInt*, *xs:unsignedShort*, *xs:unsignedByte*).
2. Запрещено использовать *any* для описания элементов (*xs:any*).
3. Не допускается неограниченная длина элементов схемы (*<xs:element maxOccurs="unbounded"/>*).

При неуспешной валидации будет выброшено исключение: *XML Schema '$schema' validation failed: список ошибок валидиации*.

Также будут проверены следующие некритичные ограничения, не влияющие на прохождение валидации:

1. Все элементы схем должны быть аннотированы.
2. Максимальная длина строк не должна быть больше 250 символов.

Некритичные ограничения не влияют на прохождение схемой валидации, ошибки будут залогированы в WARN.

Avro

Валидация avro-схем не поддерживается.

JMX метрики

При успешной загрузке интерсептора в jmx будут добавлены метрики с именем/типом схемы для каждого топика и флагом прохождения схемой валидации по пути в атрибуте *Value*:

```
kafka.consumer:type=consumer-interceptor-metrics,client-id=<client-id>,interceptor=ValidatorInterceptor,topic=<topic-name>,name=schemaName
kafka.consumer:type=consumer-interceptor-metrics,client-id=<client-id>,interceptor=ValidatorInterceptor,topic=<topic-name>,name=schemaType
kafka.consumer:type=consumer-interceptor-metrics,client-id=<client-id>,interceptor=ValidatorInterceptor,topic=<topic-name>,name=validated
```

или

```
kafka.producer:type=producer-interceptor-metrics,client-id=<client-id>,interceptor=ValidatorInterceptor,topic=<topic-name>,name=schemaName
kafka.producer:type=producer-interceptor-metrics,client-id=<client-id>,interceptor=ValidatorInterceptor,topic=<topic-name>,name=schemaType
kafka.producer:type=producer-interceptor-metrics,client-id=<client-id>,interceptor=ValidatorInterceptor,topic=<topic-name>,name=validated
```

Для схемы по умолчанию в имени метрики будет отсутствовать имя топика `topic=`.

Значения флага *validated*:

- **true** - валидация схем включена и успешно пройдена;
- **false** - валидация схем выключена.

client-id берется из конфигурации консьюмера *client.id*, при отсутствии в конфигурации генерируется автоматически, *topic* - имя топика или DEFAULT для схемы по умолчанию.

Относительные пути в конфигурации

Относительные пути до файлов в конфигурации обычно разрешаются относительно директории запуска приложения.

В некоторых случаях может быть полезно явно указать root директорию, относительно которой будут разрешаться относительные пути:

- `interceptor.config.root.dir` (общая для всех интерсепторов с подобной настройкой)
- `interceptor.validator.config.root.dir` (переопределяет общую настройку)

Пример конфигурации:

```
interceptor.validator.config.root.dir=/full/path/to
# interceptor.config.root.dir=/full/path/to

# /full/path/to/validator.conf
interceptor.validator.config=validator.conf
```

Также работает и для путей до схем в файле конфигурации валидатора:

```
schemas: {
 "topic-1": {
    type: "json",

    # /full/path/to/schema.conf
    schema: "schema.json"
 }

 "topic-2": {
    type: "xml"

    # /full/path/to/schema.xml
    schema: "schema.xml"
 }
```

Json-Consumer-Interceptor

Описание модуля

Json-Consumer-Interceptor - модуль реализует интерфейс *ConsumerInterceptor*, позволяет фильтровать поля json'a при помощи языка запросов *JsonPath*.

Интерсептор пока не работает с массивами, вложенными в массивы.

Все используемые Runtime и Provided dependencies представлены в репозитории интерсептора.

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>json-consumer-interceptor_2.13</artifactId>
  <version>2.7.1-0.0.3</version>
</dependency>
```

Подключение к Kafka Consumer

1. Необходимо добавить *jar* в *classpath*.
2. В конфигурации Consumer'a указать путь до Interceptor'a.
3. Указать путь до конфигурируемого файла, содержащий список необходимых полей на языке запросов JsonPath.
4. Сконфигурировать файл со списком необходимых полей.

Пример конфигурации Consumer'a

```
...
interceptor.classes=ru.sbt.ss.kafka.clients.consumer.FilterInterceptor
interceptor.json.config=/users/test/kafka/json-interceptor/data/valid_json_path.conf
```

Пример конфигурации файла со списком необходимых полей

```
paths: [
  "$['requestChainUid']",
  "$['table']['id', 'addressData', 'deliveryChannel', 'messageDetails', 'messageString', 'subject', 'nodeIdent']",
  "$['operation']['type', 'tranAmount', 'tranCurrency', 'sicCode', 'docId']",
  "$['card']['cardBalance', 'cardCurrency', 'cardDateClose']"
  "$['client']['clientITN', 'type']"
```

Пример фильтрации полей JSON'a

Исходный JSON:

```
{
  "requestChainUid":"W4PRIM00000000000001296291933780",
  "table":{
    "id":1296291933780,
    "usageActionOid":159761923680,
    "addressData":null,
    "deliveryChannel":"SBRF_R",
    "code":null,
    "dateFrom":"2020-08-13 07:50:45",
    "dateTo":"2100-01-01 00:00:00",
    "messageDetails":null,
    "messageString":"A^4276380045468134^52.50^RUR^2020-08-13 07:50:44^206197^0^AD^P2P_byphone_Tinkoff-Bank^ABP0010^1 806
899.21^RUR^0.00^^98ec55d180eb47bd9b1eac6cd77d8ef9^^290945056960^^BPWWW000",
```

```
  "messageTemplate":2744495830,
  "status":"P",
  "sendingChannel":null,
  "sendingDate":null,
  "sendingDetails":null,
  "refNumber":null,
  "subject":null,
  "priority":0,
  "groupNumber":0,
  "nodeIdent":"W4PRIM"
},
"operation":{
  "type":"A",
  "cardNumber":"4276380045468134",
  "tranAmount":5250,
  "tranCurrency":"RUR",
  "tranTime":"2020-08-13 07:50:44",
  "authCode":"206197",
  "tranType":"AD",
  "replyCode":"0",
  "merchant":"P2P_byphone_Tinkoff-Bank",
  "authType":"ABP0010",
  "cardBalance":180689921,
  "cardCurrency":"RUR",
  "commissionAmount":0,
  "tranId":null,
  "paymentId":null,
  "postpone":null,
  "sourceRegNum":"98ec55d180eb47bd9b1eac6cd77d8ef9",
  "clientITN":null,
  "sourceNumber":"BPWWW000",
  "sicCode":null,
  "docId":"290945056960",
  "cardCreditLimit":null,
  "lockoutCode":null,
  "cardStatus":null,
  "cardsClient":null
},
"card":{
  "cardBalance":180689921,
  "cardCurrency":"RUR",
  "rbsNumber":null,
```

```
    "cardStatus":null,
    "prevCardNumber":null,
    "product":null,
    "cardExpire":null,
    "contractNumber":null,
    "tokenReferenceID":null,
    "cardDateOpen":null
  },
  "message":null,
  "client":{
    "clientITN":null,
    "clientWayId":null,
    "cardsClient":null,
    "regNumber":null,
    "lastName":null,
    "firstName":null,
    "middleName":null,
    "birthDate":null,
    "gender":null
  }
}
```


JSON после фильтрации полей:


```
{
  "requestChainUid":"w4prim|MSG00000000001296291924510",
  "client":{
    "clientITN":"1117810202007100204",
    "type":null
  },
  "operation":{
    "type":"F",
    "tranAmount":6000,
    "tranCurrency":"RUR",
    "sicCode":null,
    "docId":"290944991820"
  },
  "table":{
    "id":1296291924510,
    "addressData":"10 02 210399",
    "deliveryChannel":"KF550-BALANCE",
    "messageDetails":null,
```

```
    "messageString":"F^4276180592767155^60.00^RUR^2020-08-12 11:30:46^271888^00^R^MOBILE BANK:
KOMISSIYA^167.00^RUR^+0.00^^^900120820WAPD0000000000000136503^^290944991820^1117810202007100204^BPPOS000",
    "subject":null,
    "nodeIdent":"w4prim|MSG"
  },
  "card":{
    "cardBalance":16700,
    "cardCurrency":"RUR",
    "cardDateClose":null
  }
}
```

В случае, если сообщение не прошло валидацию ни по одной из схем (указанных в конфигурации интерсептора), будет выброшено исключение с сообщением:

```
Avro message:$avro is incorrect. It does not match the schemes specified in the interceptor config file!
```

Kafka-certificate-signature-interceptor

Описание модуля

Предназначен для подписания сообщения сертификатом.

Логика работы

Интерсептор устанавливается на продьюсер и консьюмер, перехватывает отправляемое сообщение, добавляет цифровую подпись и открытый ключ в заголовки сообщения, используя указанный в настройках сертификат. Далее на консьюмере интерсептор проверяет подпись, используя открытый ключ.

Все используемые Runtime и Provided dependencies представлены в репозитории интерсептора.

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>kafka-certificate-signature-interceptor_2.13</artifactId>
  <version>2.7.1-0.2.0</version>
</dependency>
```

Подключение к Kafka-клиентам

1. Добавить *jar* в *classpath*.
2. Сконфигурировать kafka-клиент в соответствии с примером.

Пример конфигурации Producer:

```
### Kafka settings ###

bootstrap.servers = localhost:9092

key.serializer=org.apache.kafka.common.serialization.ByteArraySserializer
# Interceptor works only with values of types String/ByteArray
value.serializer=org.apache.kafka.common.serialization.ByteArraySserializer

security.protocol = SSL
ssl.keystore.location = ssl/keystore.jks
ssl.keystore.password = password
ssl.truststore.location = ssl/truststore.jks
ssl.truststore.password = password


### Interceptor settings ###
# Enable signature interceptor
interceptor.classes = ru.sbt.ss.kafka.interceptors.KafkaSignatureInterceptor

# Set signature service class
interceptor.signature.service.class = ru.sbt.ss.kafka.interceptors.X509SignatureService

# Set signature header name
# interceptor.signature.header = signature

# Set prefix for custom signature attributes
# interceptor.signature.attributes.prefix = signature.

# Fail send with NPE if failed to sign record
interceptor.signature.stop.send.on.failure = false


### X509 signature service settings ###

# Signature generation algorithm
# interceptor.signature.certificate.algorithm=SHA256WithRSA
```

```
# Alias of keystore entry, used for signing records (may be left empty if there's only one entry in keystore)
# interceptor.signature.certificate.key.alias =

# Set signature algorithm header name
# interceptor.signature.certificate.algorithm.header=algorithm
# Set pulic key header name
# interceptor.signature.certificate.public.key.header=public.key

### Overriding SSL Settings ###
# This settings can override kafka ssl settings

# interceptor.signature.certificate.ssl.provider =
# interceptor.signature.certificate.ssl.keystore.location = ssl/keystore.jks
# interceptor.signature.certifficate.ssl.keystore.password = password
# interceptor.signature.certificate.ssl.key.password = password
```

Пример конфигурации Consumer

```
### Kafka settings ###

bootstrap.servers = localhost:9092

group.id = test-group

key.deserializer=org.apache.kafka.common.serialization.ByteArrayDeserializer

# Interceptor works only with values of types String/ByteArray

value.deserializer=org.apache.kafka.common.serialization.ByteArrayDeserializer

security.protocol = SSL
ssl.keystore.location = ssl/keystore.jks
ssl.keystore.password = password
ssl.truststore.location = ssl/truststore.jks
ssl.truststore.password = password

### Interceptor settings ###

# Enable signature interceptor
interceptor.classes = ru.sbt.ss.kafka.interceptors.KafkaSignatureInterceptor
```

```
# Set signature header name
# interceptor.signature.header = signature

# Set prefix for custom signature attributes
# interceptor.signature.attributes.prefix = signature.

# Remove token headers on consume
# interceptor.signature.remove.headers = false

### X509 signature service settings ###

# Set signature algorithm header name
# interceptor.signature.certificate.algorithm.header = algorithm

# Set pulic key header name
# interceptor.signature.certificate.public.key.header = public.key

### Overriding SSL Settings ###

# This settings can override kafka ssl settings
# interceptor.signature.certificate.ssl.provider =
```

Kafka-timestamp-interceptor

Описание модуля

Данные интерсепторы используются для замера задержки (latency) при прохождении через событийный сегмент.

Все используемые Runtime и Provided dependencies представлены в репозитории интерсептора.

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>kafka-timestamp-interceptor_2.13</artifactId>
  <version>2.7.1-0.7.0</version>
</dependency>
```

Producer Interceptor

Функции интерсептора:

1. Добавляет заголовок с текущим временем в сообщения;
2. Добавляет заголовок с уникальным идентификатором (UUID) в сообщения;
3. Рассчитывает latency записи сообщения в топик (от времени создания сообщения до времени подтверждения записи от брокера) и передает настраиваемым репортерам (подробнее о репортерах в примере конфигурации).

Подключение интерсептора

1. Добавить интерсептор в зависимости проекта.
2. Добавить настройки интерсептора к настройкам продьюсера в соответствии с примером:

Конфигурация Producer

```
# 1) Подключить интерсептор
interceptor.classes = ru.sbt.ss.kafka.clients.timestamp.TimestampProducerInterceptor

# 2) Настроить список топиков. Интерсептор будет обрабатывать только сообщения, отправленные в топики из этого списка.
# Список можно оставить пустым - тогда интерсептор будет обрабатывать все сообщения.
# interceptor.timestamp.topics = test, test-2

# 3) Настроить заголовки, добавляемые интерсептором в сообщение:
# Имя заголовка с временем отправки сообщения продьюсером
interceptor.timestamp.create.time.header = Latency.Replicator.CreateTime

# Имя заголовка с сгенерированным UUID сообщения.
interceptor.timestamp.message.id.header = Latency.MessageId

# 3) ОПЦИОНАЛЬНО Настроить репортеры

# Настроить buffered-logger reporter, подробнее ниже
interceptor.timestamp.reporters = buffered-logger

# Настроить режим работы репортера, отключающий рассчет latency.
# Продьюсер может рассчитывать только latency от времени сообщения до времени получения подтверждения об отправке этого сообщения.
# С настройкой топика message.timestamp.type=LogAppendTime время сообщения = время записи сообщения в топик, поэтому рассчитанная latency
будет задержкой от времени записи в топик до времени получения подтверждения об отправке на клиенте (довольно бесполезная величина). Вместо
latency в репортер отправляется время записи сообщения в топик.
interceptor.timstamp.reporter.mode = timestamp

# Имя заголовка с временем получения подтверждения отправки сообщения от брокера kafka
```

```
# Заголовком называется условно, в заголовки сообщения не попадает (сообщение уже отправлено), передается только как latency заголовок
репортерам.
interceptor.timestamp.acknowledge.header = Latency.AckTime

# 4) ОПЦИОНАЛЬНО Настроить buffered-logger репортер. Данный репортер для каждого сообщения логирует:
# - метаданные сообщения (топик, партиция, оффсет)
# - acknowledge.header в формате timestamp (время получения подтверждения об отправке сообщения)

# Имя логгера, в который будет логироваться latency. Также с помощью изменения уровня этого логгера можно управлять интерсептором
interceptor.timestamp.logger.name = <имя логгера>
# Размер in-memory буфера, в который репортер сохраняет свои сообщения. Сообщения логируются одной пачкой при полном заполнении буфера.
interceptor.timestamp.logger.buffer.size = 1

# 5) Использовать window-metrics репортер как в упрощенной схеме взаимодействия не имеет смысла, т.к. репортерам вместо latency передается
timestamp (время получения подтверждения об отправке сообщения).
```

Пример сообщения *buffered-logger* репортера для продьюсера

```
{
  "message": {
    "topic": "test",
    "partition": "1",
    "offset": "0",
    "timestamp": "1620981646442"
  },
  "attributes": { "AckTime": "1620981646444" }
}
```

Consumer Interceptor

Функции интерсептора:

1. Добавляет в запись заголовок со сгенерированным id сообщения (UUID).
2. Добавляет заголовок с временем сообщения (record.timestamp()).
3. Добавляет заголовок с временем чтения сообщения консюмером.
4. Рассчитывает latency по настраиваемым заголовкам (от времени в заголовке сообщения до времени чтения сообщения) и передает настраиваемым репортерам (подробнее о репортерах в примере конфигурации).

Подключение интерсептора

1. Добавить интерсептор в зависимости проекта.
2. Добавить настройки интерсептора к настройкам консюмера в соответствии с примером:

Конфигурация Consumer

```
# 1) Подключить интерсептор
interceptor.classes = ru.sbt.ss.kafka.clients.timestamp.TimestampConsumerInterceptor

# 2) Настроить заголовки, добавляемые интерсептором в сообщение:

# Отключить добавление заголовка с идентификатором сообщения (уже добавлен консюмером или продьюсером поставщика)
interceptor.timestamp.include.message.id.header = false
# Имя заголовка с временем из сообщения (record.timestamp()). При настройке топика message.timestamp.type=LogAppendTime в сообщении будет
время записи сообщения в топик, добавленное брокером.
interceptor.timestamp.message.time.header = Latency.SinkCluster.LogAppendTime
# Имя заголовка с временем чтения сообщения консюмером
interceptor.timestamp.consume.time.header = Latency.ConsumerSystem.ConsumeTime

# 3) ОПЦИОНАЛЬНО Настроить репортеры

# Настроить два репортера, подробнее про каждый ниже
interceptor.timestamp.reporters = buffered-logger, window-metrics
# Настроить имена заголовков, которые будут переданы репотерам в качестве идентификаторов сообщения (можно списком через ',')
interceptor.timestamp.message.id.headers = Latency.MessageId
# Настроить имена заголовков, которые будут переданы репортерам для расчета latency (можно списком через ',')
interceptor.timestamp.latency.headers = Latency.ProducerSystem.CreateTime, Latency.SourceCluster.LogAppendTime,
Latency.Replicator.ConsumeTime, Latency.Replicator.CreateTime, Latency.SinkCluster.LogAppendTime

# 4) ОПЦИОНАЛЬНО Настроить buffered-logger репортер. Данный репортер для каждого сообщения логирует:
# - метаданные сообщения (топик, партиция, оффсет)
# - время чтения сообщения
# - список идентификаторов сообщения (значения заголовков из message.id.headers)
# - рассчитанную latency («от значения заголовка до времени чтения» для каждого заголовка из latency.headers)

# Имя логгера, в который будет логироваться latency. Также с помощью изменения уровня этого логгера можно управлять интерсептором
interceptor.timestamp.logger.name = <имя логгера>
# Размер in-memory буфера, в который репортер сохраняет свои сообщения. Сообщения логируются одной пачкой при полном заполнении буфера.
interceptor.timestamp.logger.buffer.size = 1

# 5) ОПЦИОНАЛЬНО Настроить window-metrics репортер. Данный репортер добавляет значение latency в JMX метрику, агрегируя среднее/максимальное
и другие значения за указанный временной интервал.
```

```
# Также позволяет настроить лимиты для среднего и максимального значения за временной интервал. Если лимиты превышены - будет залогировано предупреждение.

# Имя заголовка, с которым будет работать репортер.
interceptor.timestamp.window-metrics.header.name = Latency.SourceCluster.LogAppendTime
# Временной интервал агрегации значений и проверки лимитов, в секундах
interceptor.timestamp.window-metrics.window.s = 60
# Лимиты среднего и максимального значения в ms
interceptor.timestamp.window-metrics.limits = avg=0, max=0
# Имя логгера, в который будет залогировано предупреждение о превышении лимитов
interceptor.timestamp.window-metrics.limits.logger = <имя логгера>
```

Пример сообщения *buffered-logger* репортера для консьюмера

```
{
  "message": {
    "tags": { "MessageId": "ffcb4931-46eb-47ce-8bbf-8246e88272ce" },
    "topic": "test",
    "partition": "1",
    "offset": "0",
    "consumedAt": "1620981646444"
  },
  "attributes": { "CreateTime_latency": "222", "LogAppendTime_latency": "2" }
}
```

Window-metrics репортер

Записывает latency, рассчитанную по указанному заголовку (*interceptor.timestamp.window-metrics.header.name*) для последних *'interceptor.timestamp.window-metrics.window.s'* секунд в jmx-метрику.

Для jmx-метрики доступны следующие атрибуты:

- 50thPercentile
- 75thPercentile
- 95thPercentile
- 98thPercentile
- 999thPercentile
- 99thPercentile
- Count (общее количество образцов с начала)

- Max
- Mean (среднее для текущего окна)
- Min
- SnapshotSize (количество образцов, используемых для вычисления статистики текущего окна)

```
interceptor.timestamp.reporter = window-metrics

# Настройки для 'window-metrics' репортера:
# Метрики будут представлены для этого заголовка
# interceptor.timestamp.window-metrics.header.name = AckTime

# Размер окна в секундах
# interceptor.timestamp.window-metrics.window.s = 60
#
# Список лимитов метрик через запятую, в limitName=limitValueMs формате. Поддерживаемые лимиты 'avg', 'min', 'max'.
# Каждая window.s интервальная метрика будет проверена, и если один из лимитов превышен - предупреждение будет залогировано.
# interceptor.timestamp.window-metrics.limits = avg=50, max=150
#
# Имя логгера для предупреждений по лимитам
# interceptor.timestamp.window-metrics.limits.logger = ru.sbt.ss.utils.metrics.HistogramLimits
```

Репортер может логировать предупреждения о метриках, превышающих установленные лимиты. Лимиты проверяются каждые *interceptor.timestamp.window-metrics.window.s* секунд.

Пример для *interceptor.timestamp.window-metrics.limits = avg=0*:

```
[WARN ] [r.s.ss.utils.metrics.HistogramLimits] 'type=consumer-interceptor-metrics,client-id=consumer-test_consumer-
1,interceptor=TimestampInterceptor,topic=test,name=latency' 'AVERAGE' limit exceeded: 5.5 > 0.
```

Если указанный заголовок *interceptor.timestamp.window-metrics.header.name* отсутствует в сообщении или содержит неправильные данные - вместо него будет подставлено значение *'MaxLong'*.

Управление интерсепторами через уровни логирования

Интерсепторами можно управлять с помощью изменения уровня логирования:

Уровень логирования	Добавление заголовков в сообщение	Агрегация в JMX метриках (window-metrics reporter)	Логирование данных по каждому сообщению (buffered-logger reporter)
OFF	-	-	-
ERROR	+	-	-
WARN	+	-	-
INFO	+	+	-
DEBUG	+	+	+

Пример конфигурации логгера (logback):

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Enable configuration updates check every 30 seconds -->
<configuration scan="true" scanPeriod="30 seconds">

    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%X{ott.req.id} %d{HH:mm:ss.SSS} [%-5level] [%logger{36}] [%thread] - %msg%n</pattern>
```

```
        </encoder>
    </appender>

    <root level="info">
        <appender-ref ref="STDOUT"/>
    </root>

    <!-- level="debug" - process latency data and write it to log -->
    <!-- level="info" - process latency data, don't write it to log (write it to other reporters, if present) -->
    <!-- level="off" - don't process latency data -->
    <logger name="ru.sbt.ss.kafka.clients.timestamp.reporter.LoggerReporter" level="debug"/>

</configuration>
```

Производительность логгера

Интенсивное логирование может снизить производительность продьюсера и консьюмера. Для уменьшения влияния логирования на обработку сообщений можно:

- Использовать *ch.qos.logback.classic.AsyncAppender*, который будет обрабатывать все вызовы логгера в отдельном потоке, не блокируя обработку сообщений. Ниже пример конфигурации с использованием AsyncAppender:

```
<configuration>
  <appender name="FILE" class="ch.qos.logback.core.FileAppender">
    <file>myapp.log</file>
    <encoder>
      <pattern>%logger{35} - %msg%n</pattern>
    </encoder>
  </appender>

  <appender name="ASYNC" class="ch.qos.logback.classic.AsyncAppender">
    <appender-ref ref="FILE" />
      <!-- IMPORTANT! If not set to 0 - logger will discard messages if buffer overflows. -->
      <discardingThreshold>0</discardingThreshold>
  </appender>

  <root level="DEBUG">
    <appender-ref ref="ASYNC" />
  </root>
</configuration>
```

- Использовать репортер **buffered-logger**, который накапливает записи в буфере (размер указывается параметром *interceptor.timestamp.logger.buffer.size*) и записывает записи в виде массива, когда буфер заполнен.

Пример конфигурации:

```
interceptor.timestamp.reporter = buffered-logger
interceptor.timestamp.logger.buffer.size = 256
```

Пример массива сообщений:

```
{
   "messages": [
      {
         "message": {
            "topic": "test",
            "partition": "9",
            "offset": "0",
            "timestamp": "1621591467123"
         },
         "attributes": { "AckTime": "1621591467124" }
      },
      {
         "message": {
            "topic": "test",
            "partition": "9",
            "offset": "1",
            "timestamp": "1621591467123"
         },
         "attributes": { "AckTime": "1621591467129" }
      },
      {
         "message": {
            "topic": "test",
            "partition": "8",
            "offset": "3",
            "timestamp": "1621591468121"
         },
         "attributes": { "AckTime": "1621591468124" }
      }
   ]
}
```

DSL-Interceptor

Описание модуля

Модуль реализует интерфейсы **ConsumerInterceptor** и **ProducerInterceptor**, позволяет преобразовывать тело сообщений при помощи dsl. Важно отметить, что на вход и выход ожидается одно сообщение

Подключение к Kafka Consumer

1. Добавить актуальную версию интерсептора в зависимости проекта.
2. Добавить настройки интерсептора к настройкам kafka-клиента.
3. Сконфигурировать трансформацию с помощью dsl.

Пример конфигурации

Пример конфигурации Kafka Client

```
...
# 1) Подключить интерсептор
interceptor.classes=ru.sbt.ss.kafka.clients.dsl.DslInterceptor

# 2) Указать путь до конфигурационного файла dsl
interceptor.dsl.config=/users/test/kafka/dsl-interceptor/data/interceptor.conf
```

Пример конфигурационного файла dsl

```
{
    dsl: "simple.tr"
    environmentVariables: {
        envName: "envValue"
    }
    input: {
        type: "avro"
        schemaName: "simpleSchema.avsc"
    }
    output: {
```

```
        type: "json"
    }
}
```

1. `dsl` - расположение правил трансформации
2. `environmentVariables` - используемые переменные среды
3. `input` - формат входящего события
4. `output` - формат исходящего события

Значения полей input и output могут иметь значения:

1. событие в формате `json`:
    - `type` - со значением `json`
    - `encoding` - используемая кодировка
2. событие в формате `xml`:
    - `type` - со значением `xml`
    - `encoding` - используемая кодировка
3. событие в формате `csv`:
    - `type` - со значением `csv`
    - `encoding` - используемая кодировка
    - `columnSeparator` - разделитель полей события
    - `fieldNames` - список имен полей события для использования при трансформации, также определяет порядок полей
4. объект с полями:
    - `type` со значением "avro"
    - `schema` - список схем с указанием пути в `path`, объект имеющий логическое поле `default` со значением `true`, будет считаться схемой по умолчанию

Json-extractor-producer-interceptor

Описание модуля

Интерсептор для добавления в заголовки сообщения полей, извлеченных из тела json-сообщения с помощью jsonPath.

Работает с разными типами данных :

- String - использует как есть
- Byte array - преобразует в строку с настраиваемой кодировкой
- Любой другой объект - вызвыает метод toString.

Под капотом использует библиотеку JsonPath (https://github.com/json-path/JsonPath).

Все используемые Runtime и Provided dependencies представлены в репозитории интерсептора.

```
<dependency>
  <groupId>ru.sbt.ss</groupId>
  <artifactId>json-extractor-interceptor_2.13</artifactId>
  <version>2.7.1-0.1.1</version>
</dependency>
```

Подключение интерсептора

1. Необходимо добавить jar в classpath
2. В конфигурации Producer'a указать путь до Interceptor'a
3. Указать путь до конфигурируемого файла, содержащий список необходимых полей
4. Сконфигурировать файл со списком необходимых полей

Конфигурация

Пример json:

```
{
  "message": {
    "id": "someId",
    "body": "messageBody"
  }
}
```

Пример конфигурации для извлечения поля *"id"* и добавления его в заголовок под именем *MessageID*:

```
# Подключение интерсептора
interceptor.classes = ru.sbt.ss.kafka.clients.producer.JsonExtractorInterceptor

# Путь до нужного нам поля
```

```
interceptor.json.extractor.path = $['message']['id']

# Имя хедера
interceptor.json.extractor.header.name = MessageID

# Тип кодировки (по умолчанию UTF-8)
# interceptors.json.extractor.encoding = UTF-8
```

Руководство по системному администрированию

Сценарии администрирования

Предусловия для ручных вариантов администрирования

1. Наличие сертификатов TLS для администратора доступа. Перед началом действий поместить сертификаты в домашнюю директорию на сервере, где расположен EVTD.
2. На сервере, где расположен EVTD, в домашней директории создать файл **ssl_adm.properties**. Наполнить следующими значениями:

```
security.protocol = SSL
ssl.keystore.location = полный путь до хранилища ключа.jks
ssl.truststore.location = полный путь до хранилища ключа.jks
ssl.keystore.password = пароль хранилища ключа
ssl.truststore.password = пароль хранилища сертификата ЦС
ssl.key.password = пароль закрытого ключа
ssl.endpoint.dentification.algorithm =
```

1. Все команды выполняются под учетной записью пользователя с правами администратора доступа.

Создание архивных копий компонентов EVTD

Предусловия: необходимо заполнить параметры в конфигурационном файле **vars.yml**.

- в разделе `kafka`:

    - `backup_installdir_to:` — директория создания архивных копий для директории установки EVTD;
- в разделе `zookeeper`:
    - `backup_installdir_to:` — директория создания архивных копий для директории установки Zookeeper.

Ручной способ

На сервере, с которого производилась установка выполнить команду:

```
ansible-playbook -i inventories/<ID>/inventory zk_and_kafka.yml --ask-vault-pass -t backup -l <host>
```

, где `ID` - имя созданного inventory, `host` - доп. параметр для указания конкретных хостов, если требуется выполнение на конкретных хостах. Данной операцией создается архивная копия для всех компонентов EVTD. Для создания архивной копии только EVTD используется аналогичная команда с выбором playbook `kafka.yml`, для создания архивной копии только Zookeeper используется аналогичная команда с выбором playbook `zookeeper.yml`.

С помощью Jenkins (опциональный способ)

1. Для создания архивной копии директории установки EVTD необходимо запустить задание Jenkins **SYN_custom_kafka** (поставляется в дистрибутиве) с выбором playbook `kafka.yml` с тегом *backup*. Результатом выполнения задачи Jenkins является архив с названием *kafka_backup.tar.gz*, расположенный в директории, указанной в параметрах конфигурационного файла **vars.yml**.
2. Для создания архивной копии Zookeeper необходимо запустить задание Jenkins **SYN_custom_kafka** (поставляется в дистрибутиве) с выбором playbook `zookeeper.yml` с тегом *backup*. Результатом выполнения задачи Jenkins является создание архива *zookeeper_backup.tar.gz*, расположенного в директории, указанной в параметрах конфигурационного файла **vars.yml**.
3. Для создания архивных копий всех компонентов EVTD необходимо запустить задание Jenkins **SYN_custom_kafka** (поставляется в дистрибутиве) с выбором playbook `zk_and_kafka.yml` с тегом *backup*. Результатом выполнения задачи Jenkins является создание архивов *kafka_backup.tar.gz*, *zookeeper_backup.tar.gz*, расположенных в директориях, указанных в параметрах конфигурационного файла **vars.yml**.

Созданные архивные копии хранятся в директориях в единичном экземпляре для предотвращения переполнения директории хранения архивов. При попытке создания архивных копий компонентов повторно выполнение задания Jenkins будет прерываться, если в указанной директории уже создан ранее архив. Поэтому необходимо перед созданием нового архива предварительно удалить существующие архивные копии из данных директорий. Для этого необходимо запустить задание Jenkins **SYN_custom_kafka** с выбором соответствующего playbook с тегом *backup_remove*.

Теги *backup_remove* и *backup* можно совмещать в одном запуске задания Jenkins для экономии времени.

Восстановление из архивных копий компонентов EVTD

Предусловия: Необходимо заполнить параметры в конфигурационном файле **vars.yml**.

- в разделе `kafka`:
  - `backup_installdir_to:` — директория создания архивных копий для директории установки EVTD;
- в разделе `zookeeper`:
  - `backup_installdir_to:` — директория создания архивных копий для директории установки Zookeeper.

Ручной способ

На сервере, с которого производилась установка выполнить команду:

```
ansible-playbook -i inventories/<ID>/inventory zk_and_kafka.yml --ask-vault-pass -t backup_restore -l <host>
```

, где `ID` - имя созданного inventory, `host` - доп. параметр для указания конкретных хостов, если требуется выполнение на конкретных хостах. Данной операцией создается архивная копия для всех компонентов EVTD. Для создания архивной копии только EVTD используется аналогичная команда с выбором playbook `kafka.yml`, для создания архивной копии только Zookeeper используется аналогичная команда с выбором playbook `zookeeper.yml`.

С помощью Jenkins (опциональный способ)

1. Для восстановления EVTD из архивной копии необходимо запустить задание Jenkins **SYN_custom_kafka** (поставляется в дистрибутиве) с выбором playbook `kafka.yml` с тегом *backup_restore*. Результатом будет являться восстановление директории установки EVTD из архива, помещенного в указанную в файле **vars.yml** директорию.
2. Для восстановления Zookeeper из архивной копии необходимо запустить задание Jenkins **SYN_custom_kafka** (поставляется в дистрибутиве) с выбором playbook `zookeeper.yml` с тегом *backup_restore.* Результатом будет являться восстановление директории установки Zookeeper из архива, помещенного в указанную в файле **vars.yml** директорию.
3. Для восстановления из архивных копий всех компонентов EDMS необходимо запустить задание Jenkins **SYN_custom_kafka** (поставляется в дистрибутиве) с выбором playbook `zk_and_kafka.yml` с тегом *backup_restore*. Результатом будет являться восстановление данных EVTD из архива, помещенного в указанную в файле **vars.yml** директорию.

Получить список топиков на кластере

Ручной вариант

```
bin/kafka-topics.sh --bootstrap-server `hostname -f`:<порт> --list --command-config ~/ssl_adm.properties
```

Более подробную информацию по конкретному топику можно получить используя команду:

```
bin/kafka-topics.sh --bootstrap-server `hostname -f`:<порт> --topic <наименование топика> --describe --command-config ~/ssl_adm.properties
```

С помощью Jenkins (опциональный способ)

Получить список топиков на кластере можно с помощью задания Jenkins **kafka_config_deploy** с выбором playbook **kafka_topics_acls.yml** с тегом *print_topics* (чтобы получить список топиков) или *print_topics_with_configs* (чтобы получить список топиков и их конфигурацию) с выбором сертификата администратора в поле *admin_jks_file*.

Настраиваемые параметры:

- *inventory* - выбрать кластер, для которого необходимо получить список топиков.

Просмотреть конфигурацию топика

Ручной вариант

```
bin/kafka-configs.sh --bootstrap-server `hostname -f`:<порт> --entity-type topics --entity-name <наименование топика> --describe --command-config ~/ssl_adm.properties
```

С помощью Jenkins (опциональный способ)

Просмотреть конфигурацию топиков на кластере можно с помощью задания Jenkins **kafka_config_deploy** с выбором playbook **kafka_topics_acls.yml** с тегом *print_topics_with_configs* (чтобы получить список топиков и их конфигурацию) с выбором сертификата администратора в поле *admin_jks_file*.

Настраиваемые параметры:

- *inventory* - выбрать кластер, для которого необходимо получить список топиков.

Проверить список прав на топике

Ручной вариант

```
/opt/Apache/kafka/bin/kafka-acls.sh --bootstrap-server `hostname -f`:<порт> --list --topic <наименование топика> --command-config ~/ssl_adm.properties
```

С помощью Jenkins (опциональный способ)

Проверить список прав на топике можно с помощью задания Jenkins **kafka_config_deploy** с выбором playbook **kafka_topics_acls.yml** с тегом *print_acls* с выбором сертификата администратора в поле *admin_jks_file*.

Настраиваемые параметры:

- *inventory* - выбрать кластер, для которого необходимо получить список топиков.

Проверить лаг на топике (отставаний consumerGroup по топикам)

Ручной вариант

```
bin/kafka-consumer-groups.sh --bootstrap-server `hostname -f`:<порт> --group <наименование группы> --topic <наименование топика> --reset-offsets --to-current --command-config ~/ssl_adm.properties
```

С помощью Jenkins (опциональный способ)

Проверить лаг на топике можно с помощью задания Jenkins **kafka_config_deploy** с выбором playbook **kafka_topics_acls.yml** с тегом *check_consumer_groups*(покажет список ConsumerGroup с лагом) с выбором сертификата администратора в поле *admin_jks_file*.

Настраиваемые параметры:

- *inventory* - выбрать кластер, для которого необходимо получить список топиков.

Проверить список ConsumerGroup

Ручной вариант

```
bin/kafka-consumer-groups.sh --bootstrap-server `hostname -f`:<порт> --all-groups --list --command-config ~/ssl_adm.properties
```

С помощью Jenkins (опциональный способ)

Проверить список ConsumerGroup можно с помощью задания Jenkins **kafka_config_deploy** с выбором playbook **kafka_topics_acls.yml** с тегом *check_consumer_groups* (покажет список ConsumerGroup) с выбором сертификата администратора в поле *admin_jks_file*.

Настраиваемые параметры:

- *inventory* - выбрать кластер, для которого необходимо получить список.

Сброс текущей позиции для consumerGroup по топику

Используется в случае, если необходимо прочитать предыдущие сообщения.

Ручной вариант

Сбрасывает текущую позицию для consumerGroup по топику на последнюю запись.

```
bin/kafka-consumer-groups.sh --bootstrap-server `hostname -f`:<порт> --group <наименование группы> --topic <наименование топика> --reset-offsets --to-latest --command-config ~/ssl_adm.properties
```

Варианты использования:

- для операции по всем топикам к которым подключалась данная consumerGroup: `--all-topics`
- для операции по определенному топику: `--topic <наименование топика>`
- для операции по определенной партиции топика: `--topic <наименование топика>:<partition>`

Варианты перемещения текущей позиции:

- на запись в указанный момент времени: `--to-datetime <YYYY-MM-DDTHH:mm:SS.sss>`
- сдвиг на указанный промежуток времени от текущего: `--by-period <PnDTnHnMnS>`
- на самую ранню из доступных записей: `--to-earliest`
- на последнюю запись: `--to-latest`
- сдвиг на указанное число позиций: `--shift-by`

Перезапуск компонентов EVTD

Ручной вариант

Зайти на узел на котором будет производиться перезапуск

- Если требуется перезапусть Kafka:
  - остановить сервис: `sudo systemctl stop kafka.service`
  - остановить брокер Kafka: `opt/Apache/kafka/bin/kafka-server-stop.sh`
  - запустить брокер Kafka: `opt/Apache/bin/kafka-server-start.sh -daemon opt/Apache/config/server.properties`
  - запустить сервис: `sudo systemctl start kafka.service`
- Если требуется перезапусть Zookeeper:
  - остановить сервис: `sudo systemctl stop zookeeper.service`
  - остановить Zookeeper: `opt/Apache/kafka/bin/zookeeper-server-stop.sh`
  - запустить Zookeeper: `opt/Apache/bin/zookeeper-server-start.sh -daemon opt/Apache/config/zookeeper.properties`
  - запустить сервис: `sudo systemctl start zookeeper.service`

С помощью Jenkins (опциональный способ)

Для перезапуска компонент с помощью задания Jenkins используем **SYN_custom_kafka**:

- с выбором playbook **zk_and_kafka.yml** с тегами *stop, start* (перезапустит обе компоненты Zookeeper и Kafka);
- с выбором playbook **kafka.yml** с тегами *stop, start* (перезапустит Kafka);
- с выбором playbook **zookeeper.yml** с тегами *stop, start* (перезапустит Zookeeper).

Настраиваемые параметры:

- *inventory* - выбрать кластер, в котором необходимо перезапустить компоненты;

- *only_on_host* - отметить галочками нужные хост(ы) из списка (список соответствует выбранному inventory), если необходимо перезапустить компоненты только для данного хоста(ов) выбранного кластера;
- *install_all_hosts* - выбрать параметр, если необходимо перезапустить все хосты из данного inventory.

Если не выбран ни один из параметров *only_on_host*, *install_all_hosts* выполнение задания Jenkins прервется с ошибкой *Не выбраны хосты*.

Использование скриптов автоматизации

Перезапуск компонентов техсервиса

На сервере с которого производилась установка выполнить команду:

```
ansible-playbook -i inventories/<ID>/inventory zk_and_kafka.yml --ask-vault-pass -t start,stop
```

где `ID` - имя созданного inventory. для перезапуска только Kafka заменить `zk_and_kafka.yml` на `kafka.yml` для перезапуска только Zookeeper заменить `zk_and_kafka.yml` на `zookeeper.yml` Для указания конкретных узлов, использовать флаг `-l <fqdn узла>`
При использовании Jenkins выбираем нужный контур, playbook `zk_and_kafka.yml`/`kafka.yml`/`zookeeper.yml` флаги `start` и `stop`, узлы на которых будет производиться операция, или оставляем пустыми для выполнения на всех узлах

Работа с сущностями (топики и ACL)

Ручной способ

Заполнить в соответствующем *inventory* файл **group_vars/all/vars.yml**:

```
kafka_topics:
  list:
    - name: <имя создаваемого топика>
      partitions: <количество партиций>
      configs: # дополнительные конфигурации топика
        - retention.ms=<очистка топиков по времени в мс>

kafka_acls:
  - principal: <DN сертификата клиента>
    producer: true # в данном случае клиент выступает поставщиком. В случае потребителя указывается «consumer: true»
    topics: <имя топика, к которому подключается клиент>
```

Для создания сущностей используется команда:

```
ansible-playbook -i inventories/<ID>/inventory kafka_topics_acls.yml --ask-vault-pass
```

где ID - имя созданного inventory.

Для удаления сущностей модифицируем **vars.yml**:

- для удаления топиков добавляем список топиков в поле *topicsToDelete*:

```
kafka_topics:
  topicsToDelete:
    - <имя удаляемого топика1>
    - <имя удаляемого топика2>
```

и используем команду:

```
ansible-playbook -i inventories/<ID>/inventory kafka_topics_acls.yml --ask-vault-pass -t erase_topics
```

где ID - имя созданного inventory.

- для удаления ACLs добавляем флаг *erase: true* к удаляемой сущности

```
kafka_acls:
  - principal: <DN сертификата клиента>
    producer: true # в данном случае клиент выступает поставщиком. В случае потребителя указывается «consumer: true»
    topics: <имя топика, к которому подключается клиент>
    erase: true # флаг удаления ACL
```

и используем команду:

```
ansible-playbook -i inventories/<ID>/inventory kafka_topics_acls.yml --ask-vault-pass -t erase_acls
```

где ID - имя созданного inventory.

При помощи Jenkins (опциональный способ)

Работа с сущностями ведется посредством конфигурационных дистрибутивов.

Создание конфигурационных дистрибутивов осуществляется с помощью задания Jenkins **Kafka_config_create**.

Необходимо заполнить параметры:

- *NexusUrl* — url-ссылка на папку, в которую необходимо поместить дистрибутив;
- *NexusCred* — ID креденшнл для публикации дистрибутива в указанной папке Nexus.

При запуске задания Jenkins с указанными параметрами дистрибутив EVTD будет создан в указанной папке.

Для создания сущностей используется задание Jenkins **kafka_config_deploy** с выбором соответствующего конфигурационного дистрибутива, playbook **kafka_topics_acls.yml** без тегов и выбором сертификата администратора в поле *admin_jks_file*.

Для удаления сущностей используется задание Jenkins **kafka_config_deploy** с выбором соответствующего конфигурационного дистрибутива, playbook **kafka_topics_acls.yml** с тегами *erase_topics* и/или *erase_acls* и выбором сертификата администратора в поле *admin_jks_file*.

Автоматизированная замена TLS-сертификатов

TLS-сертификаты для сетевого взаимодействия между компонентами продукта EVTD могут быть автоматически обновлены на новые в случае окончания их срока действия.

Предварительно перед обновлением сертификтов необходимо выполнить следующие действия:

- выпустить новые сертификаты. DN новых сертификатов для конкретной компоненты продукта должны совпадать с DN старых сертификатов этой компоненты.
- скопировать новые сертификаты в директорию хранения сертификатов согласно настройке, указанной в конфигурационном файле **vars.yml**:

```
kafka:
    trustStorePath: <путь до сертификата truststore>
    trustStorePassword: <пароль truststore>
    keyStorePath: <путь до сертификата keystore>
    keyStorePassword: <пароль keystore>
```

```
    keyPassword: <пароль от ключа в хранилище>
...
zookeeper:
    trustStorePath: <путь до сертификата truststore>
    trustStorePassword: <пароль truststore>
    keyStorePath: <путь до сертификата keystore>
    keyStorePassword: <пароль keystore>
    keyPassword: <пароль от ключа в хранилище>
```

- в случае изменения настроек для ssl-сертификатов (директория хранения, имя сертификата, пароли) необходимо изменить соответствующие настройки в конфигурационном файле **vars.yml**. Сохранить эти изменения.

Для автоматической замены сертификатов необходимо запустить задание Jenkins **SYN_custom_kafka** с выбором параметра *playbook ssl_rolling_update.yml*. После запуска поочередно на каждом хосте кластера будет произведена замена сертификатов, обновятся настройки сертификатов на серверах в файлах *.properties, а также перезапустятся компоненты продукта (EVTD, Zookeeper) с новыми сертификатами на каждом сервере кластера поочередно. В рамках работы задания Jenkins осуществляется проверка корректного запуска задействованных компонент.

События системного журнала

Файлы хранения логов системного журнала:

- `$kafka_logs_dir/server.log` - содержит ошибки подключения пользователей, события смены лидера, изменение количества синхронизированных реплик;
- `$kafka_logs_dir/kafka-authorizer.log` - содержит ошибки авторизации пользователей. Для включения логирования необходимо прописать в `config/log4j.properties` строчку `log4j.logger.kafka.authorizer.logger=INFO`;
- `$kafka_logs_dir/controller.log` - содержит события смены лидера партиции и события с брокерами в кластере.

Наиболее часто встречающиеся события типа *ERROR*:

1. Неверный пароль от хранилища (ssl.keystore.password или ssl.truststore.password):

```
   [2022-02-11 12:57:22,842] ERROR [KafkaServer id=1] Fatal error during KafkaServer startup. Prepare to shutdown (kafka.server.KafkaServer)
org.apache.kafka.common.KafkaException: Failed to load SSL keystore *** of type JKS
```

```
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory$FileBasedStore.load(DefaultSslEngineFactory.java:377)
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory$FileBasedStore.<init>(DefaultSslEngineFactory.java:349)
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory.createKeystore(DefaultSslEngineFactory.java:299)
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory.configure(DefaultSslEngineFactory.java:161)
        at org.apache.kafka.common.security.ssl.SslFactory.instantiateSslEngineFactory(SslFactory.java:138)
        at org.apache.kafka.common.security.ssl.SslFactory.configure(SslFactory.java:95)
        at org.apache.kafka.common.network.SslChannelBuilder.configure(SslChannelBuilder.java:71)
        at org.apache.kafka.common.network.ChannelBuilders.create(ChannelBuilders.java:157)
        at org.apache.kafka.common.network.ChannelBuilders.serverChannelBuilder(ChannelBuilders.java:97)
        at kafka.network.Processor.<init>(SocketServer.scala:790)
        at kafka.network.SocketServer.newProcessor(SocketServer.scala:415)
        at kafka.network.SocketServer.$anonfun$addDataPlaneProcessors$1(SocketServer.scala:288)
        at kafka.network.SocketServer.addDataPlaneProcessors(SocketServer.scala:287)
        at kafka.network.SocketServer.$anonfun$createDataPlaneAcceptorsAndProcessors$1(SocketServer.scala:254)
        at kafka.network.SocketServer.$anonfun$createDataPlaneAcceptorsAndProcessors$1$adapted(SocketServer.scala:251)
        at scala.collection.IterableOnceOps.foreach(IterableOnce.scala:553)
        at scala.collection.IterableOnceOps.foreach$(IterableOnce.scala:551)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:920)
        at kafka.network.SocketServer.createDataPlaneAcceptorsAndProcessors(SocketServer.scala:251)
        at kafka.network.SocketServer.startup(SocketServer.scala:125)
        at kafka.server.KafkaServer.startup(KafkaServer.scala:303)
        at kafka.server.KafkaServerStartable.startup(KafkaServerStartable.scala:44)
        at kafka.Kafka$.main(Kafka.scala:82)
        at kafka.Kafka.main(Kafka.scala)
Caused by: java.io.IOException: Keystore was tampered with, or password was incorrect
        at java.base/sun.security.provider.JavaKeyStore.engineLoad(JavaKeyStore.java:795)
        at java.base/sun.security.util.KeyStoreDelegator.engineLoad(KeyStoreDelegator.java:222)
        at java.base/java.security.KeyStore.load(KeyStore.java:1479)
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory$FileBasedStore.load(DefaultSslEngineFactory.java:374)
        ... 23 more
Caused by: java.security.UnrecoverableKeyException: Password verification failed
        at java.base/sun.security.provider.JavaKeyStore.engineLoad(JavaKeyStore.java:793)
        ... 26 more
```

1. Неверный пароль от ключа в хранилище (значение *ssl.key.password*):

```
   [2022-02-11 12:58:45,350] ERROR [KafkaServer id=1] Fatal error during KafkaServer startup. Prepare to shutdown (kafka.server.KafkaServer)
org.apache.kafka.common.KafkaException: java.security.UnrecoverableKeyException: Cannot recover key
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory.createSSLContext(DefaultSslEngineFactory.java:268)
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory.configure(DefaultSslEngineFactory.java:173)
        at org.apache.kafka.common.security.ssl.SslFactory.instantiateSslEngineFactory(SslFactory.java:138)
```

```
        at org.apache.kafka.common.security.ssl.SslFactory.configure(SslFactory.java:95)
        at org.apache.kafka.common.network.SslChannelBuilder.configure(SslChannelBuilder.java:71)
        at org.apache.kafka.common.network.ChannelBuilders.create(ChannelBuilders.java:157)
        at org.apache.kafka.common.network.ChannelBuilders.serverChannelBuilder(ChannelBuilders.java:97)
        at kafka.network.Processor.<init>(SocketServer.scala:790)
        at kafka.network.SocketServer.newProcessor(SocketServer.scala:415)
        at kafka.network.SocketServer.$anonfun$addDataPlaneProcessors$1(SocketServer.scala:288)
        at kafka.network.SocketServer.addDataPlaneProcessors(SocketServer.scala:287)
        at kafka.network.SocketServer.$anonfun$createDataPlaneAcceptorsAndProcessors$1(SocketServer.scala:254)
        at kafka.network.SocketServer.$anonfun$createDataPlaneAcceptorsAndProcessors$1$adapted(SocketServer.scala:251)
        at scala.collection.IterableOnceOps.foreach(IterableOnce.scala:553)
        at scala.collection.IterableOnceOps.foreach$(IterableOnce.scala:551)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:920)
        at kafka.network.SocketServer.createDataPlaneAcceptorsAndProcessors(SocketServer.scala:251)
        at kafka.network.SocketServer.startup(SocketServer.scala:125)
        at kafka.server.KafkaServer.startup(KafkaServer.scala:303)
        at kafka.server.KafkaServerStartable.startup(KafkaServerStartable.scala:44)
        at kafka.Kafka$.main(Kafka.scala:82)
        at kafka.Kafka.main(Kafka.scala)
Caused by: java.security.UnrecoverableKeyException: Cannot recover key
        at java.base/sun.security.provider.KeyProtector.recover(KeyProtector.java:304)
        at java.base/sun.security.provider.JavaKeyStore.engineGetKey(JavaKeyStore.java:144)
        at java.base/sun.security.util.KeyStoreDelegator.engineGetKey(KeyStoreDelegator.java:90)
        at java.base/java.security.KeyStore.getKey(KeyStore.java:1057)
        at java.base/sun.security.ssl.SunX509KeyManagerImpl.<init>(SunX509KeyManagerImpl.java:145)
        at java.base/sun.security.ssl.KeyManagerFactoryImpl$SunX509.engineInit(KeyManagerFactoryImpl.java:70)
        at java.base/javax.net.ssl.KeyManagerFactory.init(KeyManagerFactory.java:271)
        at org.apache.kafka.common.security.ssl.DefaultSslEngineFactory.createSSLContext(DefaultSslEngineFactory.java:251)
        ... 21 more
```

1. Ключом из *conf/encrypt.pass* невозможно расшифровать любое зашифрованное значение вида *${decode:C5eoLl0iVSaYIzTMYFs+DQ==}*

```
 [2022-02-11 12:54:44,756] ERROR Exiting Kafka due to fatal exception (kafka.Kafka$)
org.jasypt.exceptions.EncryptionOperationNotPossibleException
        at org.jasypt.encryption.pbe.StandardPBEByteEncryptor.decrypt(StandardPBEByteEncryptor.java:1055)
        at org.jasypt.encryption.pbe.StandardPBEStringEncryptor.decrypt(StandardPBEStringEncryptor.java:725)
        at org.jasypt.util.text.BasicTextEncryptor.decrypt(BasicTextEncryptor.java:112)
        at ru.sbt.ss.password.decoder.SimpleTextPasswordDecoder.decode(SimpleTextPasswordDecoder.java:57)
        at ru.sbt.ss.kafka.DecryptionConfigProvider.$anonfun$get$1(DecryptionConfigProvider.scala:51)
        at scala.collection.StrictOptimizedIterableOps.map(StrictOptimizedIterableOps.scala:99)
```

```
        at scala.collection.StrictOptimizedIterableOps.map$(StrictOptimizedIterableOps.scala:86)
        at scala.collection.convert.JavaCollectionWrappers$JSetWrapper.map(JavaCollectionWrappers.scala:180)
        at ru.sbt.ss.kafka.DecryptionConfigProvider.get(DecryptionConfigProvider.scala:51)
        at org.apache.kafka.common.config.ConfigTransformer.transform(ConfigTransformer.java:103)
        at org.apache.kafka.common.config.AbstractConfig.resolveConfigVariables(AbstractConfig.java:495)
        at org.apache.kafka.common.config.AbstractConfig.<init>(AbstractConfig.java:107)
        at org.apache.kafka.common.config.AbstractConfig.<init>(AbstractConfig.java:142)
        at kafka.server.KafkaConfig.<init>(KafkaConfig.scala:1309)
        at kafka.server.KafkaConfig.<init>(KafkaConfig.scala:1312)
        at kafka.server.KafkaServerStartable$.fromProps(KafkaServerStartable.scala:34)
        at kafka.Kafka$.main(Kafka.scala:68)
        at kafka.Kafka.main(Kafka.scala)
```

1. Невозможно подключиться к Zookeeper:

```
   [2022-02-11 13:01:14,247] INFO future isn't success, cause: (org.apache.zookeeper.ClientCnxnSocketNetty)
io.netty.channel.AbstractChannel$AnnotatedConnectException: Connection refused: hostname/ip:2181
Caused by: java.net.ConnectException: Connection refused
        at java.base/sun.nio.ch.SocketChannelImpl.checkConnect(Native Method)
        at java.base/sun.nio.ch.SocketChannelImpl.finishConnect(SocketChannelImpl.java:777)
        at io.netty.channel.socket.nio.NioSocketChannel.doFinishConnect(NioSocketChannel.java:330)
        at io.netty.channel.nio.AbstractNioChannel$AbstractNioUnsafe.finishConnect(AbstractNioChannel.java:334)
        at io.netty.channel.nio.NioEventLoop.processSelectedKey(NioEventLoop.java:707)
        at io.netty.channel.nio.NioEventLoop.processSelectedKeysOptimized(NioEventLoop.java:655)
        at io.netty.channel.nio.NioEventLoop.processSelectedKeys(NioEventLoop.java:581)
        at io.netty.channel.nio.NioEventLoop.run(NioEventLoop.java:493)
        at io.netty.util.concurrent.SingleThreadEventExecutor$4.run(SingleThreadEventExecutor.java:989)
        at io.netty.util.internal.ThreadExecutorMap$2.run(ThreadExecutorMap.java:74)
        at io.netty.util.concurrent.FastThreadLocalRunnable.run(FastThreadLocalRunnable.java:30)
        at java.base/java.lang.Thread.run(Thread.java:829)
...
[2022-02-11 13:01:33,343] ERROR Fatal error during KafkaServer startup. Prepare to shutdown (kafka.server.KafkaServer)
kafka.zookeeper.ZooKeeperClientTimeoutException: Timed out waiting for connection while in state: CONNECTING
        at kafka.zookeeper.ZooKeeperClient.waitUntilConnected(ZooKeeperClient.scala:262)
        at kafka.zookeeper.ZooKeeperClient.<init>(ZooKeeperClient.scala:119)
        at kafka.zk.KafkaZkClient$.apply(KafkaZkClient.scala:1881)
        at kafka.server.KafkaServer.createZkClient$1(KafkaServer.scala:441)
        at kafka.server.KafkaServer.initZkClient(KafkaServer.scala:466)
        at kafka.server.KafkaServer.startup(KafkaServer.scala:233)
        at kafka.server.KafkaServerStartable.startup(KafkaServerStartable.scala:44)
        at kafka.Kafka$.main(Kafka.scala:82)
```

```
        at kafka.Kafka.main(Kafka.scala)
```

1. Не совпадают значения **broker.id** в файле конфигурации и в данных EVTD в файле **meta.properties**:

```
 [2022-02-11 13:03:15,339] ERROR Fatal error during KafkaServer startup. Prepare to shutdown (kafka.server.KafkaServer)
kafka.common.InconsistentBrokerIdException: Configured broker.id 4 doesn't match stored broker.id 1 in meta.properties. If you moved your
data, make sure your configured broker.id matches. If you intend to create a new broker, you should remove all data in your data directories
(log.dirs).
        at kafka.server.KafkaServer.getOrGenerateBrokerId(KafkaServer.scala:842)
        at kafka.server.KafkaServer.startup(KafkaServer.scala:255)
        at kafka.server.KafkaServerStartable.startup(KafkaServerStartable.scala:44)
        at kafka.Kafka$.main(Kafka.scala:82)
        at kafka.Kafka.main(Kafka.scala)
```

1. Не удается создать директории для данных:

```
[2022-02-11 13:04:29,163] ERROR Failed to create or validate data directory /KAFKADATA1 (kafka.server.LogDirFailureChannel)
java.io.IOException: Failed to create data directory /KAFKADATA1
        at kafka.log.LogManager.$anonfun$createAndValidateLogDirs$1(LogManager.scala:156)
        at scala.collection.IterableOnceOps.foreach(IterableOnce.scala:553)
        at scala.collection.IterableOnceOps.foreach$(IterableOnce.scala:551)
        at scala.collection.AbstractIterable.foreach(Iterable.scala:920)
        at kafka.log.LogManager.createAndValidateLogDirs(LogManager.scala:147)
        at kafka.log.LogManager.<init>(LogManager.scala:81)
        at kafka.log.LogManager$.apply(LogManager.scala:1212)
        at kafka.server.KafkaServer.startup(KafkaServer.scala:290)
        at kafka.server.KafkaServerStartable.startup(KafkaServerStartable.scala:44)
        at kafka.Kafka$.main(Kafka.scala:82)
        at kafka.Kafka.main(Kafka.scala)
[2022-02-11 13:04:29,168] ERROR Shutdown broker because none of the specified log dirs from /KAFKADATA1 can be created or validated
(kafka.log.LogManager)
```

События мониторинга

Мониторинг производится:

- Самостоятельно через преднастроенный JMX-порт;

- С помощью системы мониторинга Mayak компонента Platform V Synapse Event-domain management (EDM). Настройка подключения описана в документации по мониторингу. Подробное описание использования: *Руководство пользователя* документации *Система мониторинга Mayak*.

Отслеживаются следующие метрики:

Область действия метрики	**Метрика**	**Триггер**	**MBEAN Name**
Кластер	Список брокеров кластера	не должно изменяться	describeCluster() KafkaClusterBrokersCount, KafkaClusterBrokersList
Кластер	Список живых брокеров	не должно изменяться	Варианты: 1) скрипт `./bin/zookeeper-shell.sh localhost:2181 ls /brokers/ids` возвращает список живых брокеров 2) также можно использовать любую jmx метрику, например: `kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec` - если она обновляется - значит брокер работает 3) есть jmx метрика `kafka.server:type=KafkaServer,name=BrokerState` она возвращает зачения Broker State = 0: Not Running Broker State = 1: Starting Broker State = 2: Recovering from Unclean Shutdown Broker State = 3: Running as Broker Broker State = 4: Running as Controller Broker State = 5: Pending Controlled Shutdown

Область действия метрики	Метрика	Триггер	MBEAN Name
			Broker State = 6: Broker Shutting Down
Кластер	Наличие активного лидера кластера	должен быть 1	`kafka.controller:type=KafkaController,name=ActiveControllerCount` суммируются значения метрики по всем брокерам кластера
Брокер	Загрузка DRIVE	Если максимальное значение больше 80% - предупреждение, 90% - ошибка	Настроить мониторинг свободного места на узле EVTD
Брокер	Количественные показатели работы брокера в байтах	должно быть больше 0	`kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec` `kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec`
Брокер	Партиции не полностью среплицированные	не должно увеличиваться	`kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions`
Брокер	Партиции не	не должно быть	`kafka.controller:type=KafkaController,name=OfflinePartitionsCount`

Область действия метрики	Метрика	Триггер	MBEAN Name
	доступные		
Брокер	Количество запросов к брокеру-лидеру	должно быть > 0	`kafka.network:type=RequestMetrics,name=RequestsPerSec,request=FetchFollower`
Брокер	Количество неудачных запросов на брокер	не должно увеличиваться	`kafka.server:type=BrokerTopicMetrics,name=FailedProduceRequestsPerSec` `kafka.server:type=BrokerTopicMetrics,name=FailedFetchRequestsPerSec`
Брокер	Размеры очередей	не должны увеличиваться	`kafka.network:type=RequestChannel,name=RequestQueueSize` `kafka.network:type=RequestChannel,name=ResponseQueueSize`
Топик	Нагрузка на топики в tps – сколько сообщений поступает в топик за секунду в шт.	Среднее должно быть >0	`kafka.server:type=BrokerTopicMetrics,name=TotalFetchRequestsPerSec,topic=`
Топик	Нагрузка на топики – какой	Среднее должно	`kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec,topic=`

Область действия метрики	Метрика	Триггер	MBEAN Name
	объем данных записывается	быть >0	
Топик	Нагрузка на топики – какой объем данных вычитывается	Среднее должно быть >0	`kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec,topic=`
Подписчик	Лаг на топиках по consumer group – показывает какая группа плохо читает/перестала читать	Не должен увеличиваться	Лаги и вся информация о консьюмер оффсетах хранится в служебном топике `__consumer_offsets`, поэтому получить информацию о консьюмер группах можно только оттуда. Что можно сделать: - прочитать и распарсить самим метрики из jmx kafka.log.Log.LogStartOffset и LogEndOffset - вычесть одно из другого, чтобы получить лаг. - сделать запрос через API Apache Kafka напрямую describeConsumerGroups и получить информацию по tcp в уже конкретных объектах

Часто встречающиеся проблемы и пути их устранения

Не выявлено.