# Общие сведения

Kafka представляет из себя распределенную потоковую платформу (distributed streaming platform), которая предоставляет своим пользователям следующие возможности:

* Публиковать и подписываться на потоки записей.
* Хранить потоки записей в отказоустойчивой манере.
* Обрабатывать потоки записей по мере их поступления.

Ключевой особенностью Kafka является отсутствие необходимости в развертывании какой-либо специальной кластерной инфраструктуры. То есть, приложения, которые используют Kafka являются обычными java приложениями.

Для работы Kafka необходимо развернуть Zookeeper, который используется ею для организации и поддержки кластера.

Терминология:

**Запись(Record)** – сущность состоящая из ключа, значения и временной метки.

**Тема(Topic)** - поток записей, конкретнее, имя категории для которой публикуются записи.

Темы поддерживают множество подписчиков. Иными словами, тема может иметь одного и более подписчиков или не иметь подписчиков совсем.

**Продюсер(Producer)** – сущность публикующая записи для конкретных тем.

**Потребитель(Consumer)** – сущность потребляющая записи из конкретных тем, на которые она подписана. Каждому потребителю назначается имя потребительской группы. Если двум потребителям назначено одно и тоже имя потребительской группы, запись будет доставлена только одному из них. Если у обоих потребителей разное имя потребительской группы, то запись будет доставлена каждому из них.

**Брокер(Broker)** – Kafka сервер, который работает в составе Kafka кластера.

**Кластер(Cluster)** – Множество Kafka брокеров работающих как одно целое. Брокеры могут быть развернуты как на одной, так и на множестве физических машин.

Kafka предлагает 4 программных интерфейса:

**Producer API** позволяет приложению публиковать поток записей для одной или нескольких тем.

**Consumer API** позволяет приложению подписываться на одну или более тем.

**Streams API** позволяет приложению выполнять роль обработчика потока, потребляя входящий поток от одной или нескольких тем и производя исходящий поток к одной или нескольким темам, преобразовывая входящие потоки в исходящие потоки.

**Connect API** позволяет строить и выполнять повторно используемых продюсеров и потребителей которые подключают темы к существующим приложениям и системам данных.

# Maven зависимости для работы с интерфейсами.

Для Producer API и Consumer API:

<dependency>

    <groupId>org.apache.kafka</groupId>

    <artifactId>kafka-clients</artifactId>

    <version>2.4.1</version>

</dependency>

Для Streams API:

<dependency>

    <groupId>org.apache.kafka</groupId>

    <artifactId>kafka-streams</artifactId>

    <version>2.4.1</version>

</dependency>

Для Connect API:

<dependency>

<groupId>org.apache.kafka</groupId>

<artifactId>connect-api</artifactId>

<version>2.4.1</version>

</dependency>

# Как работать с Streams API

**Создаем экземпляр StreamsBuilder.**  
StreamsBuilder builder = new StreamsBuilder();

**Создаем входящий поток из темы “Topic”**  
KStream<String, String> input = builder.stream("Topic");

**Создаем исходящий поток используя лямбду, преобразующую входящий поток. В данном случае, мы просто заменяем значение записи “Green” на “Blue” при его наличии.**  
KStream<String, String> outputStream = inputStream.map((s, s2) -> new KeyValue<>(s, s2.equals("Green") ? "Blue" : s2));

**Создаем экземпляр клиента.**  
Properties props = new Properties();  
properties.put(StreamsConfig.APPLICATION\_ID\_CONFIG, "application");  
properties.put(StreamsConfig.BOOTSTRAP\_SERVERS\_CONFIG, "kafka-broker1:9092");  
properties.put(StreamsConfig.DEFAULT\_KEY\_SERDE\_CLASS\_CONFIG, Serdes.String().getClass());  
properties.put(StreamsConfig.DEFAULT\_VALUE\_SERDE\_CLASS\_CONFIG, Serdes.String().getClass());

KafkaStreams streams = new KafkaStreams(builder.build(), properties);

streams.start();

# Как работать с Producer API

Создаем продюсера, отправляем сообщение и закрываем его.

Properties properties = new Properties();

properties.put("bootstrap.servers", "localhost:9092");

properties.put("acks", "all");

properties.put("key.serializer", "org.apache.kafka.common.serialization.StringSerializer");

properties.put("value.serializer", "org.apache.kafka.common.serialization.StringSerializer");

try (Producer<String, String> producer = new

KafkaProducer<>(properties)) {

producer.send(new ProducerRecord<>("test", "key", "value"));

}

# Как работать с Consumer API

Создаем потребителя.

Properties properties = new Properties();

properties.put("bootstrap.servers", "localhost:9092");

properties.put("group.id", "consumer");

properties.put("key.deserializer", StringDeserializer.class.getName());

properties.put("value.deserializer", StringDeserializer.class.getName());

KafkaConsumer<String, String> consumer = new KafkaConsumer<>( properties);

Подписываем его на тему.

consumer.subscribe(Arrays.asList("test"));

Принимаем сообщения.

try {

while (true) {

ConsumerRecords<String, String> records = consumer.poll(10);

for (ConsumerRecord<String, String> record : records)

System.out.println(record.key()+" "+ record.value());

}

} finally {

consumer.close();

}

# Как развернуть Kafka

Запускаем Zookeeper сервер.

kafka\_2.13-2.4.1\bin\windows\zookeeper-server-start.bat config\zookeeper.properties

Запускаем Kafka сервер.

kafka\_2.13-2.4.1\bin\windows\kafka-server-start.bat config\server.properties

Создаем Kafka тему.

kafka\_2.13-2.4.1\bin\windows\kafka-topics.bat --create --bootstrap-server localhost:9092 --replication-factor 1 --partitions 1 --topic test

Выводим список тем.

kafka\_2.13-2.4.1\bin\windows\kafka-topics.bat --list --zookeeper localhost:2181

Выводим описание тем.

kafka\_2.13-2.4.1\bin\windows\kafka-topics.bat --bootstrap-server localhost:9092 --describe

# Примеры продюсера и потребителя

Сначала запускаем серверы Zookeeper и Kafka как описано выше.

Собранный пример продюсера можно найти тут:

kafka\_consumer\target\kafka-producer\ kafka-producer.jar

Запускаем так: java -jar kafka-producer.jar

Собранный пример потребителя можно найти тут:

kafka\_consumer\target\kafka-consumer\ kafka-consumer.jar

Запускаем так: java -jar kafka-consumer.jar

# Некоторые итоги

Что есть у ServiceMix и нет у Kafka:

* Общая платформа для развертывания (в случае с Kafka, независимые приложения подключаются к Kafka и используют ее как брокер).
* Создание цепочек обработки внутри.
* Возможность генерации событий и запуска любого произвольного процесса внутри. (при работе с Kafka, всегда должно быть приложение producer в явном, либо неявном виде (Streams API)).
* Возможность создавать микросервисы внутри такие как, например HTTP и т.п.