whitehorses®
member of the human network

WHITEBOOK

Integratie met Apache Kafka

Maart 2017

Auteur:
Roger Goossens
INTEGRATIE SPECIALIST

Inleiding

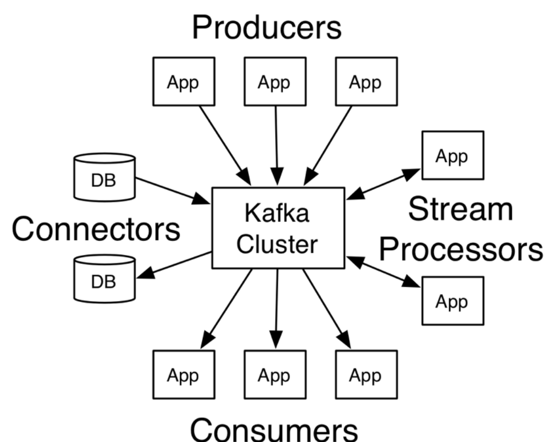
In de integratiewereld hoor ik de laatste tijd steeds vaker mensen over Apache Kafka spreken. Ook het aantal refrenties naar artikelen en blogposts m.b.t. Apache Kafka laat een stijgende lijn zien op mijn twitterfeed. Tijdens het recent Oracle Fusion Middleware Forum in Split bleek ook Oracle het produkt te gebruiken in haar almaar uitdijende Cloud portfolio. Hoog tijd om er zelf eens wat meer over te weten. In dit Whitebook wordt uitgelegd wat Apache Kafka is en waar het voor toegepast kan worden.
Daarnaast wordt er met een integratieblik gekeken naar manieren om met Apache Kafka te kunnen integreren. Hierbij worden een aantal technieken aan de hand van (code-) voorbeelden gedemonstreerd. De codevoorbeelden zijn te downloaden via Github.

Kafka

De website van Apache Kafka beschrijft het produkt als een "distributed streaming platform".

Messaging

Een van de toepassingen van Apache Kafka is voor data- en/of bericht-uitwisseling tussen systemen en applicaties middels het Publish-Subscribe integratie-patroon. In die zin is het vergelijkbaar met traditionele JMS oplossingen. Het grote verschil is dat Apache Kafka van de grond af aan is opgebouwd met schaalbaarheid in het achterhoofd en dat het mede daardoor supersnel is. Analoog aan JMS maakt Kafka gebruik van topics. Vergeleken met traditionele JMS oplossingen zijn deze een combinatie van JMS-queues en -topics waarbij de voordelen van beiden - schaalbaarheid vs. Multi-consumer - zijn gecombineerd. Hierbij wordt gebruikt gemaakt van de Producer API (voor het vastleggen van data op een topic) en de Consumer API (voor het consumeren van de data op een topic).



Streaming
Daarnaast kan Apache Kafka toegepast worden om datastromen te transformeren. Hierbij wordt de data van een of meerdere topic geconsumeerd en getransformeerd naar een of meerdere output topics. Hierbij wordt gebruik gemaakt van de Streaming API.
<< iets meer informatie over messaging (ordering) stream processing en storage (clustering)>>
Voor meer informatie: https://kafka.apache.org

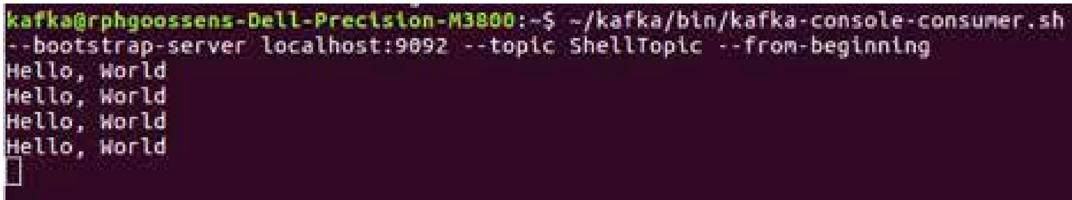
Kafka - installatie

Installeren van Kafka is vrij recht toe recht aan. Voor installatie op Ubuntu wordt verwezen naar onderstaande link:
https://www.digitalocean.com/community/tutorials/how-to-install-apache-kafka-on-ubuntu-14-04

Na installatie kan de kafka server gestart worden:
nohup ~/kafka/bin/kafka-server-start.sh ~/kafka/config/server.properties > ~/kafka/kafka.log 2>&1 &

Shellscripts

Kafka's installatie bevat een tweetal shell scripts waarmee data op een topic geproduceerd, resp. geconsumeerd kan worden.
Via een command prompt kan middels onderstaand commando het consumeren van een topic (genaamd *ShellTopic* in dit voorbeeld) gestart worden.
~/kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic ShellTopic --from-beginning
De topic ShellTopic hoeft niet eerst gecreeerd te worden (je krijgt hier initieel wel een melding over, maar daarna is de topic aanwezig).
Via een tweede command prompt kan vervolgens wat data gepubliceerd worden op diezelfde topic:
echo "Hello, World" | ~/kafka/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic ShellTopic > /dev/null
Elk bericht dat via het producer script wordt aangeboden komt vervolgens in het consumer window tevoorschijn.



Een lijst met aanwezige topics is ook via een shellscript te raadplegen:
~/kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
Met hetzelfde shell script kunnen topic gecreëerd, bijgewerk en verwijderd worden. Om de topic die hierboven gecreëerd is weer te verwijderen:
~/kafka/bin/kafka-topics.sh --delete --topic ShellTopic --zookeeper localhost:2181

Integratie met Java

Pub/Sub

Shellscript zijn handig om de installatie te testen. Om vanuit applicaties te integreren met Kafka is Java bijvoorbeeld een betere oplossing. Hoewel publish en subscribe op een Kafka topic heel veel weg heeft van integratie met JMS, ondersteunt Kafka niet de JMS API. Kafka komt met een eigen zgn. Producer en Consumer API. Het heeft er alle schijn van dat deze API nog volop in ontwikkeling is. Codevoorbeelden die op internet voorhanden zijn, blijken vaak al achterhaald en gebruik te maken van deprecated code. Onderstaande codevoorbeelden zijn succesvol getest met Kafka versie 0.10.2 .
Zorg dat onderstaande maven dependency aanwezig is om de code te laten compileren.

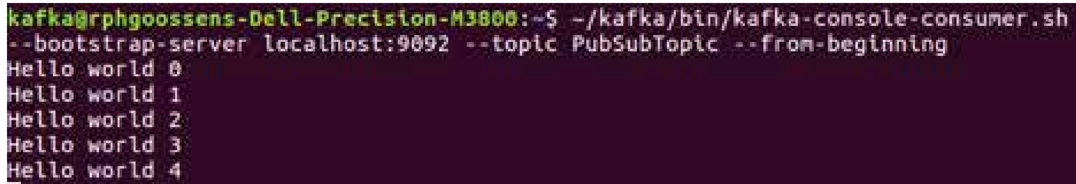
```
<dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.10.2.0</version>
</dependency>
```

Producer API

Middels onderstaand code voorbeeld wordt de String Hello World een aantal malen geproduceerd (het aantal is gelijk aan het 1e input argument).

```
public class DataProducer {

   private static final String BROKERHOST = "127.0.0.1";
   private static final String BROKERPORT = "9092";

   private final String topic;

   public DataProducer(String topic) {
       this.topic = topic;
   }

   public void produce(long events) {
       Properties producerProps = new Properties();
       producerProps.setProperty(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, 
BROKERHOST + ":" + BROKERPORT);
       producerProps.setProperty(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, 
StringSerializer.class.getName());
       producerProps.setProperty(ProducerConfig.VALUE_SERIALIZER_CLASS_
CONFIG, StringSerializer.class.getName());
       try (Producer<String, String> producer = new 
KafkaProducer<>(producerProps)) {
             for (long nEvents = 0; nEvents < events; nEvents++) {
                   String key = "Producer";
                   String value = "Hello world " + nEvents;

                   ProducerRecord<String, String> data = new 
ProducerRecord<>(topic, key, value);
                   producer.send(data);
             }
       }
   }
}
```

Indien voor het testen een consumer shell is gestart die naar dezelfde topic (JavaPubSubTopic) luistert, kun je in bijbehorend command window de message voorbij zien komen om te testen of het werkt.



Het ProducerRecord is in bovenstaande voorbeeld van het type String (key), String (value). Dit kan echter ook bv. Een Integer of een ByteArray zijn. Let er dan wel op dat de properties key.serializer en value.serializer de juiste serializer moeten krijgen. Keys kunnen gebruikt worden voor message correlatie en hoeven niet uniek te zijn,

Consumer API

Voor het consumeren van data op een topic middels java is dezelfde maven dependency nodig als bij het produceren.

```
public class DataConsumer {

  private static final String BROKERHOST = "127.0.0.1";
  private static final String BROKERPORT = "9092";

  private final String topic;
  private final String group;

  public DataConsumer(String topic, String group) {
      this.topic = topic;
      this.group = group;
  }

  public void consume() {
      Properties consumerProps = new Properties();
      consumerProps.setProperty(ConsumerConfig.GROUP_ID_CONFIG, group);
      consumerProps.setProperty(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG,
BROKERHOST + ":" + BROKERPORT);
      consumerProps.setProperty(ConsumerConfig.KEY_DESERIALIZER_CLASS_
CONFIG, StringDeserializer.class.getName());
      consumerProps.setProperty(ConsumerConfig.VALUE_DESERIALIZER_CLASS_
CONFIG, StringDeserializer.class.getName());
```

```
        try (Consumer<String, String> consumer = new
KafkaConsumer<>(consumerProps)) {
                consumer.subscribe(Arrays.asList(new String[]{topic}));

                while (true) {
                        ConsumerRecords<String, String> records = consumer.
poll(Long.MAX_VALUE);
                        for (ConsumerRecord<String, String> record : records) {
                        System.out.println(group + " " + record.offset() + ": " +
record.key() + ":" + record.value());
                        }
                }

        }
    }
}
```

Bovenstaande code luistert naar dezelfde topic als waar het producer voorbeeld naar produceert. Als de consumer draait en de producer code wordt uitgevoerd, is te zien dat de consumer code zijn werk doet. Als ook de shell consumer nog draait, zie je daar - zoals verwacht in een Pub-Sub integratie - ook de berichten binnenkomen.

Een interessant aspect bij het consumeren is het zgn. **group id.** Indien meerdere consumer processen dezelfde group id gebruiken wordt het record maar door 1 van deze consumers opgepakt. Bij verschillende group ids wordt het record per groep eenmaal verwerkt.

Streaming API

Een interessant onderdeel van Kafka is de streaming API. Data welke binnenkomt op een queue kan hiermee middels streaming direct worden getransformeerd en afgeleverd worden op een andere queue.

Zie onderstaand voorbeeld. De code heeft de volgende Maven dependency nodig
Zorg dat onderstaande maven dependency aanwezig is om de code te laten compileren.

```
<dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-clients</artifactId>
            <version>0.10.2.0</version>
</dependency>
```

De java code transformeert de records die binnenkomen op een input queue naar upper case. Een simpel voorbeeld, maar het laat wel zien hoe krachtig Kafka streams zijn. In de tranformatiestap kun je analoog aan java 8 streams helemaal los gaan met transformaties, filters, aggregaties en wat dies meer zij.

```
public class DataStreamer {

  private static final String BROKERHOST = "127.0.0.1";
  private static final String BROKERPORT = "9092";

  private final String inputTopic;
  private final String outputTopic;

  public DataStreamer(String inputTopic, String outputTopic) {
      this.inputTopic = inputTopic;
      this.outputTopic = outputTopic;
  }

  public void stream() {
      Properties streamerProps = new Properties();
      streamerProps.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-
capitalize");
      streamerProps.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, BROKERHOST +
":" + BROKERPORT);
      streamerProps.put(StreamsConfig.KEY_SERDE_CLASS_CONFIG, Serdes.
String().getClass().getName());
```

```
        streamerProps.put(StreamsConfig.VALUE_SERDE_CLASS_CONFIG, Serdes.
String().getClass().getName());

        final Serde<String> stringSerde = Serdes.String();
        KStreamBuilder builder = new KStreamBuilder();

        KStream<String, String> values = builder.stream(stringSerde,
stringSerde, inputTopic);
        values.mapValues(String::toUpperCase).to(stringSerde, stringSerde,
outputTopic);

        KafkaStreams streams = new KafkaStreams(builder, streamerProps);

        streams.start();

        Runtime.getRuntime().addShutdownHook(new Thread(streams::close));
    }
}
```

Zie hieronder het resultaat van de consumer als zowel de streamer als de consumer gestart zijn, waarbij die laatste luistert naar de output queue van de streamer en de producer wordt gestart die op zijn beurt de input queue van de streamer bedient.

```
Run (kafka-streamer) ×   Run (kafka-consumer) ×   Run (kafka-producer) ×
  Group1 35: Producer:HELLO WORLD 0
  Group1 36: Producer:HELLO WORLD 1
  Group1 37: Producer:HELLO WORLD 2
  Group1 38: Producer:HELLO WORLD 3
  Group1 39: Producer:HELLO WORLD 4
```

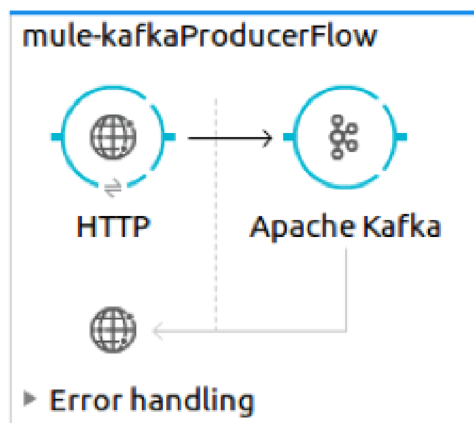
Integratie met een platform - Mule

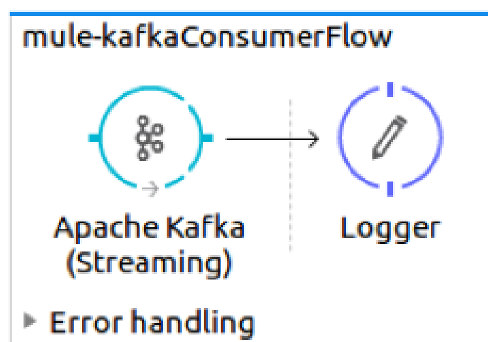
Inmiddels zijn er al een aantal middleware platformen die integratie met Kafka ondersteunen. Zo kan er bv. via de Oracle Service Bus - zonder officiële ondersteuning van Oracle overigens - data met Kafka uitgewisseld worden. Zie http://www.ateam-oracle.com/osb-transport-for-apache-kafka-part-1/.

Ook Mule kent een adapter voor Kafka.
Zie onderstaande link voor de installatie

- Install new software
- Anypoint Connectors Update Site
- Apache Kafka Connector (Mule 3.7+) 1.0.1.201608240947

Voor meer informatie: https://docs.mulesoft.com/mule-user-guide/v/3.8/kafka-connector#install-the-kafka-connector
https://docs.mulesoft.com/mule-user-guide/v/3.8/kafka-connector





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

<mule xmlns:tracking="http://www.mulesoft.org/schema/mule/ee/
tracking" xmlns:http="http://www.mulesoft.org/schema/mule/http"
xmlns:apachekafka="http://www.mulesoft.org/schema/mule/apachekafka"
xmlns="http://www.mulesoft.org/schema/mule/core" xmlns:doc="http://www.
mulesoft.org/schema/mule/documentation"
    xmlns:spring="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans http://
www.springframework.org/schema/beans/spring-beans-current.xsd
http://www.mulesoft.org/schema/mule/core http://www.mulesoft.org/schema/
mule/core/current/mule.xsd
http://www.mulesoft.org/schema/mule/http http://www.mulesoft.org/schema/
mule/http/current/mule-http.xsd
http://www.mulesoft.org/schema/mule/apachekafka http://www.mulesoft.org/
schema/mule/apachekafka/current/mule-apachekafka.xsd
http://www.mulesoft.org/schema/mule/ee/tracking http://www.mulesoft.org/
schema/mule/ee/tracking/current/mule-tracking-ee.xsd">
  <http:listener-config name="HTTP_Listener_Configuration" host="0.0.0.0"
port="8081" doc:name="HTTP Listener Configuration"/>
  <apachekafka:config name="Apache_Kafka__Configuration"
bootstrapServers="localhost:9092" consumerPropertiesFile="consumer.
properties" doc:name="Apache Kafka: Configuration"
producerPropertiesFile="producer.properties"/>
  <flow name="mule-kafkaProducerFlow">
      <http:listener config-ref="HTTP_Listener_Configuration" path="/mule-
kafka" allowedMethods="PUT" doc:name="HTTP"/>
      <apachekafka:producer config-ref="Apache_Kafka__Configuration"
topic="MuleTopic" key="&quot;MuleProducer&quot;" doc:name="Apache Kafka"/>
  </flow>
  <flow name="mule-kafkaConsumerFlow">
      <apachekafka:consumer config-ref="Apache_Kafka__Configuration"
topic="MuleTopic" partitions="1" doc:name="Apache Kafka (Streaming)"/>
      <logger message="#[payload]" level="INFO" doc:name="Logger"/>
  </flow>
</mule>
```

Consumer.properties bevat group.id=mule
Logger en Kafka producer produceert #[payload] naar MuleTopic
Consumer leest van MuleTopic
Na opstarten en PUT middels postman van tekst levert de volgende logregel op in consolue:

```
**********************************************************************************************
*             - - + APPLICATION + - -              *        - - + DOMAIN + - -       * - - + STATUS + - - *
**********************************************************************************************
* mule-kafka                                       * default                         * DEPLOYED           *
**********************************************************************************************

INFO  2017-04-30 15:36:57,727 [[mule-kafka].mule-kafkaConsumerFlow.stage1.02] org.mule.api.processor.LoggerMessageProcessor: Hello Mule with consumer!
```

Conclusie

- Geen JMS
- Java code vrij eenvoudig
- Clustering en scaling apart Whitebook nodig
- Veel andere talen worden ondersteund

https://kafka.apache.org