sa.engine Kafka plugin

Stream Analyze Sweden AB
Sweden

Version 2.0
2019-12-07
sa_KafkaAPI_1.0.pdf

This document describes how to publish streams to Kafka topics, and subscribe to topics from Kafka using sa.engine.

Table of contents

1. Introduction

When interfacing sa.engine with Kafka there are two main actions: publishing object streams to Kafka topics, and subscribing on object streams coming from Kafka topics.

This introduction assumes that you have some experience on using sa.engine. It is expected that you have done the tutorial in the documentation of the visual analyzer.

1.1. Installing the Kafka plugin

Kafka is installed by unzipping the file `sa_kafka.zip` under the sa.engine home directory. The sa.engine home directory is `Documents/SA` on windows and OSX and `~/SA` on Linux based systems. If you are unsure where to find the sa home directory you can start sa.engine and call the function `sa_home()`.

After unzipping `sa_kafka.zip` you should at least have the following file tree in sa.engine home:

- bin
  - sa_kafka.jar
  - lz4-1.3.0.jar
  - slf4j-api-1.7.21.jar
  - slf4j-nop-1.7.21.jar
  - snappy-java-1.1.2.6.jar
  - kafka-clients-0.10.2.0.jar
- models
  - kafka
    - master.osql

Once you've unzipped the contents of sa_kafka.zip correctly you can start sa.engine and load the kafka model with `load_model("kafka")`.

2. Kafka interface

The Kafka plugin consists of three OSQL functions with signatures:

```
kafka_config(Charstring nm) -> Record

kafka_subscribe(Record conf, Charstring tp) -> Stream of Record

kafka_publish(Stream s, Charstring tp, Record conf) -> Stream
```

The function `kafka_config` returns the Kafka configuration named *nm*.
The function `kafka_subscribe` subscribes to a topics *tp* from a Kafka server specified by the configuration record *conf*.

The function `kafka_publish` publishes a stream on topic *tp* for a Kafka server specified by the configuration record *conf*.

2.1. Configuration records

When publishing and subscribing to Kafka, sa.engine needs to know how to connect to the Kafka service in question. This is done with a *configuration record* that specifies configuration properties and their corresponding values. The format of Kafka configuration records is documented in https://kafka.apache.org/documentation/#configuration.

Below follows two examples of configuration records, one for subscribing and one for publishing to a Kafka service running with default settings on the same computer as where sa.engine is running.

The configuration record for a local consumer retrieved by calling `kafka_config("local-consumer")` could look like this:

```
{
  "bootstrap.servers": "localhost:9092",
  "group.id": "None",
  "auto.offset.reset": "earliest"
}
```

Example 1 the configuration record stored as kafka_config("local-consumer").

The configuration record for a local producer `kafka_config("local-producer")` could look like:

```
{
  "bootstrap.servers": "localhost:9092"
}
```

Example 2 the configuration record stored as kafka_config("local-producer").

2.2. Publishing sa.engine streams to Kafka topics

Publishing streams to a Kafka topic is done with the function `kafka_publish`. To publish an sa.engine stream *s* on a topic named *tp* to a Kafka service running on the local machine with standard settings, call:

```
kafka_publish(s, tp, kafka_config("local-producer"));
```

Example 3 Publishing a stream s to topic tp on a local kafka service with default settings

The `kafka_publish` function returns a stream, which allows you to publish stream elements inside continuous queries, for example:

```
select x
  from Stream of Number x, Stream of Number y, Record conf
 where conf = kafka_config("local-producer")
   and y = kafka_publish(heartbeat(0.5),"clock_tick", conf)
   and x = kafka_publish(y+1,"addition", conf)
```

Example 4 Publishing two steps to different topics. First raw data elements are published on topic "clock_tick" and then one is added to each raw data element is published on topic "addition".

2.3. Subscribing to a topic from a Kafka service.

Subscribing to a topic from a Kafka service is done with the function `kafka_subscribe`. To subscribe to the topic named *tp* on a Kafka service running on the local machine with standard settings, call:

```
kafka_subscribe(kafka_config('local-consumer'), tp);
```

Example 5 Subscribing to topic tp on the local Kafka service.

The result from such a subscription is a stream of records where each record has the structure: `{"offset": o, "topic": t, "value": v}`. Here, *o* is the offset number of the record received from Kafka, *t* is the topic of the record, and *v* is actual data element received. If you have published the stream `heartbeat(0.5)` to the topic `"clock_tick"` the result would be a stream of records:

```
{
  "offset": 0,
  "topic": "clock_tick",
  "value": 0
}

{
  "offset": 1,
  "topic": "clock_tick",
  "value": 0.5
}

{
  "offset": 2,
  "topic": "clock_tick",
  "value": 1
}
```


etc…

Example 6 Result stream for a subscription to topic "clock" if heartbeat(0.5) has previously been published to it.

3. Customized data encoding

By default sa.engine will publish data as a stream of JSON objects and the subscription will likewise receive a stream of JSON object for a topic. The system supports other formats as well, as described for CSV below and the user can define own (de)serialization OSQL functions for other formats.

3.1. Serialization functions

Data element must be converted to a linear format (usually strings) when being published with Kafka. If you have a different format than the default JSON on the published Kafka data you can add your own serialization by registering to sa.engine a *serializer* function *s* with signature `s(Object o)->Charstring` that takes an OSQL object *o* as argument and converts it to a string. Below is an example on how to create and register your own serializer function for CSV and then using it when publishing a stream:

```
create function my_serializer(Object o) -> Charstring
  as stringify_csv(o);

set kafka_config('my-local-producer') =
  {
    'bootstrap.servers': 'localhost:9092',
    'serializer': 'my_serializer'
  };

kafka_publish(heartbeat(1), 'testTopic1', kafka_config('my-local-producer'));
```

Example How to create, register, and call a custom serializer of a stream.

3.2. Deserialization functions

If you have a different format than JSON for data subscribed to from Kafka you can add your own deserialization by registering a *deserializer* function *d* with signature `d(Charstring s) -> Object`, which takes a received Kafka string *s* and converts it to a corresponding OSQL object, for example:

```
create function my_deserializer(Charstring c) -> Object
  as unstringify_json(c);

set kafka_config("my-local-consumer") =
  {
    "bootstrap.servers": "localhost:9092",
    "group.id":          "None",
    "auto.offset.reset": "earliest",
    "deserializer":      "my_deserializer"
  };

kafka_subscribe(kafka_config('my-local-consumer'), 'my_topic');
```

Example 7 How to create, register, and call a custom deserializer function for a consumer to a Kafka topic.