sa.engine EventHub plugin

Stream Analyze Sweden AB
Sweden

Version 2.0
2019-12-07
sa_EventHubApi_1.0.pdf

This document describes how to publish streams to Azure Eventhubs, and consume events from Azure Event Hub using sa.engine.

ContentsW

Introduction

When interfacing sa.engine with Event Hub there are two main actions: publishing object streams to Even Hubs, and subscribing on object streams coming from Event Hubs.

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 Event Hub plugin

Event Hub is installed by unzipping the file `sa_event_hub.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_event_hub.zip` you should at least have the following file tree in sa.engine home:

- bin
  - sa_event_hub.jar
  - azure-eventhubs-2.2.0.jar
  - azure-eventhubs-eph-2.4.0.jar
  - azure-keyvault-core-1.0.0.jar
  - azure-storage-8.0.0.jar
  - commons-lang3-3.4.jar
  - gson-2.8.5.jar
  - guava-20.0.jar
  - jackson-core-2.9.4.jar
  - proton-j-0.31.0.jar
  - qpid-proton-j-extensions-1.1.0.jar
  - slf4j-api-1.7.25.jar
- models
  - event_hub
    - master.osql

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

2. Event Hub interface

The Event Hub plugin consists of three OSQL functions with signatures:

```
event_hub_config(Charstring nm) -> Record

event_hub_subscribe(Record conf) -> Stream of Record
```

```
event_hub_publish(Stream data, Record props) -> Stream
```

The function `event_hub_config` returns the Event Hub configuration named *nm*.
The function `event_hub_subscribe` subscribes to an Event Hub specified by the configuration record *conf*.

The function `event_hub_publish` publishes a stream an Event Hub specified by the configuration record *conf*.

2.1. Configuration records

When publishing and subscribing to Event Hub, sa.engine needs to know how to connect to the Event Hub service in question. This is done with a *configuration record* that specifies configuration properties and their corresponding values. The format of Event Hub configuration records is the following:

```
{
  "namespace": "_EVENT-HUB-NAMESPACE_",
  "event_hub_name": "_EVENT-HUB-NAME_",
  "policy_name": "_POLICY-NAME_",
  "primary_sas_key": "_PRIMARY-SAS-KEY_",
  "storage_connection_string": _STORAGE-CONNECTION-STRING_",
  "storage_container_name": "_STORAGE-CONTAINER-NAME_",
  "host_name_prefix":"_HOST-NAME-PREFIX_"
};
```

Example 1 the configuration format for connecting to Event Hub

2.2. Publishing sa.engine streams to Event Hubs

Publishing streams to an Event Hub is done with the function `event_hub_publish`. To publish an sa.engine stream *s* on an Event Hub service defined as a config named “test”, call:

```
event_hub_publish(s, event_hub_config("test"));
```

Example 2 Publishing a stream s to an Event Hub service where the config för connecting is stored in event_hub_config(“test”)

The `event_hub_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 = event_hub_config("test")
   and y = event_hub_publish(heartbeat(0.5), conf)
   and x = event_hub_publish(y+1, conf)
```

Example 3 Publishing in two streams on Event Hub "test2

2.3. Subscribing to an Event Hub.

Subscribing to an Event Hub service is done with the function `event_hub_subscribe`. To subscribe to the Event Hub service configured in event hub config "test", call:

```
event_hub_subscribe(event_hub_config(test));
```

Example 4 Subscribing to Event Hub "test".

The result from such a subscription is a stream of records where each record has the structure: `{"value": v}`. Here *v* is actual data element received. If you have published the stream `heartbeat(0.5)` to the Event Hub the result would be a stream of records:

```
{
  "value": 0
}

{
  "value": 0.5
}

{
  "value": 1
}

etc…
```

Example 5 Result stream for a subscription to Event Hub "test" 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 Event Hub. If you have a different format than the default JSON on the published Event Hub 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 event_hub_config('test') =
  {
    …
    'serializer': 'my_serializer'
  };

event_hub_publish(heartbeat(1), event_hub_config('test'));
```

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 Event Hub you can add your own deserialization by registering a *deserializer* function *d* with signature `d(Charstring s) -> Object`, which takes a received Event Hub string *s* and converts it to a corresponding OSQL object, for example:

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

set event_hub_config("test") =
  {
    …
    "deserializer":      "my_deserializer"
  };

event_hub _subscribe(event_hub _config(test'));
```

Example 6 How to create, register, and call a custom deserializer function for a consumer to an Event Hub.