Hitachi NEXT 2018
IoT Analytics Using Streaming Data

Contents
Page 2: Introduction to MQTT Protocol
Page 4: Guided Demonstration – MQTT Overview
Page 9: Guided Demonstration MQTT – Telemetry (IoT)
Page 22: Use Case – Racing cars
Introduction to MQTT Protocol

MQTT stands for MQ Telemetry Transport. It is a publish or subscribe, extremely simple and lightweight messaging protocol, designed for constrained devices and low-bandwidth, high-latency or unreliable networks. The design principles are to minimize network bandwidth and device resource requirements whilst also trying to ensure reliability and some degree of assurance of delivery. These principles also turn out to make the protocol ideal of the emerging “machine-to-machine” (M2M) or “Internet of Things” world of connected devices, and for mobile applications where bandwidth and battery power are at a premium.

Use Cases

MQTT excels in scenarios where reliable message delivery is crucial for an application but a reliable network connection is not necessarily available. For example, mobile networks. Typical use cases of MQTT include:

- Telemetry
- Automotive
- Smart Home
- Energy Monitoring
- Chat Applications
- Notification Services
- Healthcare Applications

MQTT Message Pattern

As mentioned, MQTT implements the Publisher - Subscriber paradigm. This paradigm decouples a client that publishes a message (“publisher”) to other clients that receive the message (“subscribers”). Moreover, MQTT is an asynchronous protocol, which means that it does not block the client while it waits for the message. Of course, that’s in contrast to HTTP, which is mainly a synchronous protocol. Another interesting property of MQTT protocol is that it does not require that the client (“subscriber”) and the publisher are connected at the same time.
MQTT Publisher-Subscriber Pattern

As described above, MQTT is a message-based protocol that uses Publisher-Subscriber pattern. The key component in MQTT is the MQTT broker. The main task of MQTT broker is dispatching messages to the clients (“Subscribers”). In other words, it receives messages from Publisher and dispatches these messages to the Subscribers. While it dispatches messages, the MQTT broker uses a **Topic** to filter the clients that will receive the message. The Topic is a string, and it is possible to combine the Topics creating topic levels.

A Topic is like a virtual channel that connects a Publisher to its Subscribers. This Topic is managed by the MQTT broker. Through this virtual channel, the Publisher is decoupled from the Subscribers, and the clients (Publishers or Subscribers) do not have to know each other. This makes this protocol highly scalable without a direct dependency from the message Producer (“Publisher”) and the message Consumer (“Subscriber”).

**Topic**

A Topic is a UTF-8 string, which is used by the broker to filter messages for each connected client. It consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).

In comparison to a message queue, a Topic is very lightweight. There is no need for a client to create the desired topic before publishing or subscribing to it, because a broker accepts each valid Topic without any prior initialization.
Guided Demonstration: MQTT - Overview

Introduction  In this guided demonstration, you will configure and start a Mosquitto MQTT Broker (EMQ) on a Microsoft Windows OS.

Objectives  In this guided demonstration, you will:

- Create Topics
- Publish/Subscribe messages to some Topics

MQTT.fx

MQTT.fx is a MQTT Client written in Java based on Eclipse Paho.

1. Double-click in the **MQTT.fx** icon.

2. Select: **Extras > Edit Connection Profiles**.

   ![MQTT.fx interface](image)

   **Figure 1:** MQTT.fx interface

   **Figure 2:** Edit Connection Profiles option

HITACHI is a trademark or registered trademark of Hitachi, Ltd.
Broker Connection

To configure a connection to the Broker:

1. Click on the cog-wheel icon (A default local mosquitto connection may already be defined.)
2. Check connection details.

3. Click Cancel.
4. Click on Connect.
Publish / Subscribe

Once connected, you can Subscribe or Publish to Topics.

1. Click on the **Subscribe** option in the main menu and then enter: **test** in the drop-down box.

2. Next, click **Subscribe** to the right.

You are now subscribing to the Topic: **test**

3. Click on the **Publish** option in the main menu.
4. Select **test** from the drop-down menu and type the message.

5. Click on **Publish** button to the right.
6. Click on the **Subscribe** option in the main menu.

The message will appear in Subscribe.

- Look at the Log to examine the stream.
- The Topics Collector can scan for Topics in the stream(s).
- Each Topic can be colour coded.
Guided Demonstration: MQTT – Telemetry (IoT)

**Introduction**
In this Guided Demonstration, you will be analyzing ‘IoT data’ around a moving ‘thing’, for example, GPS co-ordinates.

**Objectives**
In this guided demonstration, the instructor will guide you through:

- Create Topics
- Publish / Subscribe messages

**Step 1 - IoT Streaming Data (Transformation)**

1. Open the Transformation:
   
   C:\NEXT-2018
   
   \Next - IoT Analytics using Streaming Data
   
   \Guided Demo - IoT
   
   \tr_iot_device_emulator.ktr

   ![Diagram of MQTT transformation process]

   **MQTT Producer**
   
   Connection: localhost: 1883
   
   Client ID: ${ClientId}
   
   Topic: ${Topic}
   
   QoS: 0
   
   Message field: message
Generate Rows

The Generate Rows step, sets the ‘MQTT Message fields’. The variable `_${NumberOfMessages}_` picks up the parameter value: 500000.

User defined Java Expression (UDJE) Step - Randomize

This step populates some of the MQTT message fields.
Formula Step

This step calculates the LONG and LAT ‘MQTT Message fields’.

![Formula Step Image]

Populate message fields - Replace in string

This step populates the MQTT message fields by replacing the values in the message string field.

![Populate message fields - Replace in string Image]
Select

As the MQTT message fields are now populated, you just need to select the data stream ‘message’ field.

![Select / Rename values](image)

Delay

For each input row, the "Delay row" step will wait the indicated timeout period before giving the row to the next step.

Use this step if you deliberately want to slow down your transformation.

![Delay row](image)

There’s a delay of 500 milliseconds.
MQTT Publisher

The MQTT Publisher step, ‘publishes’ the messages via the Topic, IoTDevices, to the MQTT Broker. This simulates our sensor data.

To configure the MQTT Publisher:

1. Drag and drop the MQTT Producer step onto the canvas.
2. Double-click to set the properties as outlined below:
The parameters have been set in the Transformation Properties > Parameters tab.

<table>
<thead>
<tr>
<th>SETTING</th>
<th>DEFAULT VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>localhost:1883</td>
<td>URL of Broker</td>
</tr>
<tr>
<td>Topic Name</td>
<td>${Topic}</td>
<td>Topic - IoTDevices</td>
</tr>
<tr>
<td>Message field</td>
<td>message</td>
<td>The data stream message field</td>
</tr>
<tr>
<td>Client ID</td>
<td>${ClientID}</td>
<td>ID of IoT Device</td>
</tr>
<tr>
<td>Connection timeout</td>
<td>30</td>
<td>Timeout</td>
</tr>
<tr>
<td>QoS</td>
<td>0</td>
<td>Message is sent once no retransmission</td>
</tr>
</tbody>
</table>

Do not RUN the Transformation until the MQTT Consumer has been configured.

**Step 2- MQTT Consumer**

The MQTT Consumer step batches the incoming stream.

1. Open the Transformation:

   C:\NEXT –2018

   \IoT Analytics using Streaming Data

   \Guided Demo - IoT

   \tr_mqtt_consumer.ktr
To configure the MQTT Consumer:

1. Drag and drop the MQTT Consumer step onto the canvas.
2. Double-click to set the properties as outlined below:

<table>
<thead>
<tr>
<th>SETTING</th>
<th>DEFAULT VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>localhost:1883</td>
<td>URL of Broker</td>
</tr>
<tr>
<td>Topic Name</td>
<td>IoTDevices</td>
<td>Topic - IoTDevices</td>
</tr>
<tr>
<td>Input Message field</td>
<td>message</td>
<td>The incoming data stream message field</td>
</tr>
<tr>
<td>Output Message field</td>
<td>message</td>
<td>The outgoing data stream message field</td>
</tr>
<tr>
<td>Input Topic field</td>
<td>topic</td>
<td>The incoming data stream topic field</td>
</tr>
<tr>
<td>output Topic field</td>
<td>Topic</td>
<td>The outgoing data stream Topic field</td>
</tr>
<tr>
<td>Connection timeout</td>
<td>30</td>
<td>Timeout</td>
</tr>
<tr>
<td>QoS</td>
<td>0</td>
<td>Message is sent once no retransmission</td>
</tr>
</tbody>
</table>
Step 3 - IoT Streaming Data (Transformation)

With this transformation, you ‘Subscribe’ to the IoT topic, conduct some PDI transformation Logic and finally ‘Publish’ back to the MQTT Broker.

1. Open the Transformation:
   
   C:\NEXT -2018
   \IoT Analytics using Streaming Data
   \Guided Demo - IoT
   \tr_datastream.ktr

   Analyzing ‘IoT data’ around a moving ‘thing’, e.g. GPS co-ordinates.

2. Drag and drop the MQTT Get records from stream step onto the canvas.
3. Double-click to set the properties as outlined below:

![Get records from stream](image)

4. Enable: **Allow messages of**.
5. Enter the Topic: **IoTDevices**.

**Get Variables**

The messages are being grouped by Time and Rows. These next steps illustrate how you can apply some transformational logic to the messages.

![Get Variable](image)
Switch / case

In our case, we route rows of data to a Transformation for grouping on ROWS and TIME.

MQTT Producer

The MQTT Producer publishes the results back to the Broker.
To configure the MQTT Producer:

1. Drag and drop the **MQTT Producer** step onto the canvas.
2. Double-click to set the properties as outlined below:

<table>
<thead>
<tr>
<th>SETTING</th>
<th>DEFAULT VALUE</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connection</td>
<td>${publisherBroker}</td>
<td>Points to a parameter value: tcp://localhost:1883</td>
</tr>
<tr>
<td>Client ID</td>
<td>Dest-e6679942</td>
<td>ID of Publisher</td>
</tr>
<tr>
<td>Topic Name</td>
<td>PDIDataStream</td>
<td>Topic</td>
</tr>
<tr>
<td>Message field</td>
<td>message</td>
<td>The data stream message field</td>
</tr>
<tr>
<td>Connection timeout</td>
<td>30</td>
<td>Timeout</td>
</tr>
<tr>
<td>QoS</td>
<td>0</td>
<td>Message is sent once no retransmission</td>
</tr>
</tbody>
</table>
Step 3 – RUN Transformation

Now that everything is place, let’s RUN the Transformations.

1. Run the Transformation:
   
   tr_iot_device_emulator.ktr

   This Publishes telemetry co-ordinates to a Topic IoTDevices.

2. Open MQTT.fx.

3. Connect to the broker: local mosquito.

4. Click Subscribe.

5. Click Scan to scan for Topics.


Notice the IoT device messages stream to the Mosquitto Broker.
7. Click on the **Log** button to display the stream.

![Image of MQTT Client Log](image)

8. Next **RUN** the transformation:
   
   `tr_datastream.ktr`

   This Publishes augmented telemetry co-ordinates to a Topic: **PDIDataStream**.

9. Click the **Scan** button to scan for Topics.

![Image of Topics Collector](image)

10. Double-click on the **Topic: IoTDevices**.
    
    Notice the messages stream to the Mosquitto Broker. You can also colour code the Topics.

![Image of MQTT Client Subscribe](image)
MQTT Use Case: Racing Cars

This use case illustrates a solution for visualizing streaming telemetric MQTT data from 3 racing cars.

Step 1: Producing the Data Stream
The **MQTT_RC_Producer_Sensor_Data.ktr** generates the racing telemetric data for each of the drivers:

- Latitude G Force
- Longitude G Force
- Lap
- Kmh
- Gear
- Rpm
- Accelerator
- Brake

The resulting JSON output stream is published by the MQTT Producer step to the Mosquitto Broker in the Topic: *dashboard-sample*.

**Step 2: Consuming the Data Stream**

The streaming data is consumed from the Mosquitto Broker by subscribing to the Topic: *dashboard-sample*.

A MQTT_streaming Data Service is created which is defined as the datasource for the CDF dashboard.

At some stage the name of each driver will need to be pushed down into the query to define the dataset in the Data Service. This is achieved by passing a parameter, `${USERNAME_QUERY}`, in the WHERE clause which is equal to another variable value: `${username}`. This value is set in the Race Car Dashboard.cda.
Step 3: Race Car Data sources

The Race Car dashboard.cda defines 2 datasources based on the Data Service:

- realtime_table
- realtime_chart

Each data source runs a query on the MQTT_streaming data service, with a constraint (WHERE clause) on username. The username variable is populated by retrieving the session name: ${[session:name]}

The query is refreshed every 1 second (Component Refresh Period) and return 10 rows for the table and 200 for the chart.

```xml
<DataAccess access="public" connection="realtime_table" id="realtime_table" type="streaming">
  <Name>realtime_table</Name>
  <Columns/>
  <ComponentRefreshPeriod>1</ComponentRefreshPeriod>
  <DataServiceQuery>SELECT increaseDate, Lap, latG, longG, Kmh, Gear, Rpm, Accelerator, Brake
FROM "MQTT_streaming"
WHERE username = '${[username]}'</DataServiceQuery>
  <Parameters>
    <Parameter default='${[session:name]}'>${[username]}</Parameter>
  </Parameters>
</DataAccess>
```
Step 4: Start PDI Transformations

1. In PDI, run the `MQTT_RC_Producer_Sensor_Data.ktr`.
2. Check that data is being streamed from the MQTT Producer step.
3. In the PUC, double click on the **CDE Race Car Dashboard**. The admin driver telemetrics are displayed.
To view the other drivers’ telemetrics:

4. In another browser, log on with the credentials for the user: **Suzy**.
5. RUN another instance of **the CDE Race Car Dashboard**.

End of Lab