iotagent-mosca Documentation

Release 0.2.0a1

Matheus Magalhaes

Apr 06, 2018

Contents:

IoT agents are responsible for receiving messages from physical devices (directly or through a gateway) and sending them commands in order to configure them. This iotagent-mosca, in particular, receives messages via MQTT with JSON payloads.

CHAPTER 1

Concepts

1.1 MQTT

MQTT is a somewhat simple protocol: it follows a publish/subscriber paradigm and messages are exchanged using topics. These topics are simple strings such as `/admin/cafe/attrs`. A publisher can, well, publish messages by sending them to a MQTT broker using a particular topic and all the subscribers that are listening to that topic will receive a copy of the message.

Subscribers can listen not only to specific topics, but also to topics with wildcards. For instance, one could use a '+' to indicate that any token will match the subscribed topic, such as `/admin/+/attrs` - messages sent to both `/admin/cafe/attrs` and `/admin/4593/attrs`, for instance, will be received by this subscriber. Another possibility is to create a subscription to all remainder tokens in the topic, such as `/admin/#`. All messages sent to topics beginning with `/admin/` will be received by this subscriber.

1.2 Kafka

Kafka is, in fact, a project from the Apache Foundation. It is a messaging system that is similar to MQTT in the sense that both are based on publisher/subscriber. Kafka is way more complex and robust - it deals with multiple subscribers belonging to the same group (and performs load-balancing between them), stores and replays messages, and so on. The side effect is that its clients are not that simple, which could be a heavy burden for tiny devices.

CHAPTER 2

Operation

2.1 Configuration

iotagent-mosca configuration is pretty simple. These are the environment variables used by it:

- BACKEND_HOST, BACKEND_PORT: redis host and port to be used.

2.2 Receiving messages from DeviceManager via Kafka

Messages containing device operations should be in this format:

```
{
  "event": "create",
  "meta": {
    "service": "admin"
  },
  "data": {
    "id": "cafe",
    "attrs" : {

    }
  }
}
```

These messages are related to device creation, update, removal and actuation. For creation and update operations, it contains the device data model to be added or updated. For removal operation, it will contain only the device ID being removed. The actuation operation will contain all attributes previously created with their respective values.

The documentation related to this message can be found in DeviceManager Messages.

2.2.1 Device configuration for iotagent-mosca

The following device attributes are considered by iotagent-mosca. All these attributes are of `meta` type.

Table 2.1: Device attributes for iotagent-mosca

Attribute	Description	Example
topic	Topic to which the device will publish messages.	`/admin/efac/attrs`

Example

The following message serves as an example of a device with all attributes used by iotagent-mosca.

```
{
  "label": "Thermometer Template",
  "attrs": [
    {
      "label": "topic",
      "type": "meta",
      "value_type": "string",
      "static_value": "/agent/main/000BABC80F4A/devinfo"
    },
    {
      "label": "temperature",
      "type": "dynamic",
      "value_type": "float"
    },
    {
      "label": "reset",
      "type": "actuator",
      "value_type": "boolean"
    }
  ]
}
```

2.3 Sending messages to other components via Kafka

When iotagent-mosca receives a new message from a particular device, it must publish the new data to other components. The default subject used to publish this information is "device-data". Check data-broker documentation to check how these subjects are translated into Kafka topics.

The message sent by iotagent-mosca is like this one:

```
{
  "metadata": {
    "deviceid": "efac",
    "protocol": "mqtt",
    "payload": "json"
  },
  "attrs": {
  }
}
```

As previously stated, the "attrs" attribute is the same as the one from DeviceManager Messages.

2.4 Receiving messages from devices via MQTT

Any message payload sent to iotagent-mosca must be in JSON format. Preferably, they should follow a simple key-value structure, such as:

```
{
  "speed": 100.0,
  "weight": 50.2,
  "id": "truck-001"
}
```

If more than one device is supposed to use the same topic, you should set the client ID in all messages sent by devices. Its value should be `service:ID`, such as `admin:efac`.

Should the device send its messages using any other JSON scheme, the user could translate them into simple key-value structures using flows, using flowbuilder for that.

2.4.1 Example

This example uses `mosquitto_pub` tool, available with `mosquitto_clients` package. To send a message to iotagent-mosca via MQTT, just execute this command:

```
mosquitto_pub -h localhost -i "admin:efac" -t /device/data -m '{"temperature" : 10}'
```

This command will send the message containing one value for attribute `speed`. The device ID is `efac` and its service is "admin". `-t` flag sets the topic to which this message will be published and `-i` sets the client ID to be sent.

This command assumes that you are running iotagent-mosca in your machine (it also works if you use dojot's docker-compose).

CHAPTER 3

How to build/update/translate documentation

If you have a local clone of this repository and you want to change the documentation, then you should follow this simple guide.

3.1 Build

The readable version of this documentation can be generated by means of sphinx. In order to do so, please follow the steps below. Those are actually based off ReadTheDocs documentation.

```
pip install sphinx sphinx-autobuild sphinx_rtd_theme sphinx-intl
make html
```

For that to work, you must have pip installed on the machine used to build the documentation. To install pip on an Ubuntu machine:

```
sudo apt-get install python-pip
```

To build the documentation in Brazilian Portuguese language, run the following extra commands:

```
sphinx-intl -c conf.py build -d locale
make html BUILDDIR=build/html-pt_BR O='-d build/doctrees/ -D language=pt_BR'
```

3.2 Update workflow

To update the documentation, follow the steps below:

1. Update the source files for the english version
2. Extract translatable messages from the english version

```
make gettext
```

3. Update the message catalog (PO Files) for pt_BR language

```
sphinx-intl -c conf.py update -p build/gettext -l pt_BR
```

4. Translate the messages in the pt_BR language PO files

This workflow is based on the Sphinx i18n guide.

CHAPTER 4

How does it work

iotagent-mosca depends on a Kafka broker, so that it can receive messages informing it about new devices (and, in extension, about their updates and removals). It listens to device management topics on Kafka and for MQTT messages using its internal broker implemented by Mosca library. For more information about the internals of this mechanism, check iotagent-nodejs documentation.

CHAPTER 5

How to build

As this is a npm-based project, building it is as simple as

```
npm install
```

And that's all.

CHAPTER 6

How to run

As simple as:

```
node index.js
```

Remember that you should already have a Kafka node (with a zookeeper instance).

6.1 How do I know if it is working properly?

Simply put: you won't. In fact you can implement a simple Kafka publisher to emulate the behaviour of a device manager instance and a listener to check what messages it is generating. But it seems easier to get the real components - they are not that hard to start and to use (given that you use dojot's docker-compose). Check also DeviceManager documentation for further information about how to create a new device.