SAPL-MQTT-PEP

Overview

The sapl-mqtt-pep is an extension for the HiveMQ mqtt broker. It is used for enforcing PDP decisions for mqtt client connections, subscriptions and publishes via the mqtt broker. Furthermore, sent mqtt messages could be altered according to the specified policies through constraint handling.

Setup

Extensions will be started from the 'extensions' folder of the HiveMQ broker. To use the sapl-mqtt-pep extension use the following steps:

1. Clone this repository into a maven project of at least Java 11.
2. Run the Maven command `mvn package` to build the packaged extension. A file called "sapl-mqtt-pep-2.1.0-SNAPSHOT-distribution.zip" should now be located in the "target" directory of your project.
3. Unpack this file, and you got the "sapl-mqtt-pep" folder as built extension ready to run.
4. Now move the extension in the "HIVEMQ_HOME/extensions" directory. "HIVEMQ_HOME" stands for the path to your HiveMQ broker.
5. If you are using the HiveMQ enterprise broker the sapl-mqtt-pep extension will be automatically hot-loaded by the broker. In case you are using the HiveMQ community edition broker the broker needs to be restarted and your extension will be loaded automatically on startup of the broker.

Alternative setup

Also, it is possible to start the extension via an embedded HiveMQ broker. To do this, follow these steps:

1. Import the following maven dependency into your project:

```
<dependencies>
    <dependency>
        <groupId>io.sapl</groupId>
        <artifactId>sapl-mqtt-pep</artifactId>
        <version>2.1.0-SNAPSHOT</version>
    </dependency>
    <dependency>
        <groupId>com.hivemq</groupId>
        <artifactId>hivemq-community-edition-embedded</artifactId>
        <version>2022.1</version>
    </dependency>
</dependencies>
```

2. Use the `EmbeddedExtension` builder to build the extension with the `HivemqPepExtensionMain` class of the sapl-mqtt-pep project. When initializing the `HivemqPepExtensionMain` object provide at least the path to the `sapl-extension-config.xml` file for the constructor. Then, you can specify the path to the embedded pdp policies in this file or provide the path to the constructor alternatively.
3. Use the `EmbeddedHiveMQ` builder to configure the embedded HiveMQ mqtt broker with the sapl-mqtt-pep extension built in the last step.
4. At last, you can start the broker and the sapl-mqtt-pep extension will be up and running.

Configuration

For configuration purposes you find in sapl-mqtt-pep extension folder the `sapl-extension-config.xml` file. There you can specify certain settings within the `sapl-extension-config` tag, for example:

```
<sapl-extension-config>
    <connection-enforcement-timeout-millis>5000</connection-enforcement-timeout-
millis>
    <subscription-enforcement-timeout-millis>5000</subscription-enforcement-timeout-
millis>
    <publish-enforcement-timeout-millis>5000</publish-enforcement-timeout-millis>
</sapl-extension-config>
```

The following settings respectively tags are possible:

- `connection-enforcement-timeout-millis`: While initializing the connection the HiveMQ broker needs a timeout in milliseconds. If the PDP did not decide within this interval whether the connection is allowed or not the connection will be denied.
- `subscription-enforcement-timeout-millis`: While initializing a subscription the HiveMQ broker needs a timeout in milliseconds. If the PDP did not decide within this interval whether the subscription is allowed or not the subscription will be denied.
- `publish-enforcement-timeout-millis`: While publishing a message to the HiveMQ broker a timeout in milliseconds needs to be set. If the PDP did not decide within this interval whether

the publishing is allowed or not the subscription will be denied.

- `authz-subscription-timeout-millis`: This value specifies a timeout in milliseconds. It is used to cancel existing SAPL authorization subscriptions to the PDP that were no longer actively used during the defined interval and that are no longer necessary for an enforcement of a mqtt action.
- `pdp-implementation`: Specifies the type of PDP used. It is possible to use an `embedded` PDP or a `remote` PDP.
- `embedded-pdp-policies-path`: Specifies where the embedded PDP can find your policies.
- `embedded-pdp-policies-path-is-relative-to-extension-home`: Specifies whether the path of your policies is to interpret from the sapl-mqtt-pep extension folder or is given as an absolute file system path.
- `remote-pdp-base-url`: Specifies the URL to your remote PDP, e.g.: "https://localhost:8080".
- `remote-pdp-client-key`: The client key for establishing the connection to the remote PDP.
- `remote-pdp-client-secret`: The client secret for establishing the connection to the remote PDP.
- `remote-pdp-first-back-off-millis`: When losing the connection to the remote PDP this time in milliseconds will be awaited before trying to reconnect.
- `remote-pdp-max-back-off-millis`: On losing the connection to the remote PDP this value specifies the maximum duration awaited before trying to reestablish connection.
- `remote-pdp-back-off-factor`: When trying to reconnect to the remote PDP this factor specifies how great the first backoff duration is prolonged till the max backoff duration is reached.

Policies

The policies do have the following basic format:

```
policy "weather_policy"
permit
    subject.clientId == "weather_station"
where
    action.type == "mqtt.subscribe";
    resource.topic == "outside_temperature";
```

With an exception for the 'environment' identifier, the identifiers are used as objects. Therefore, via key step it is possible to access the different object values.

Identifier 'subject'

The key steps on the 'subject' identifier:

- `userName`: The name specified by the user for authentication purposes. If no username was given then the username will be 'anonymous'.
- `cliendId`: The unique id of the mqtt client.

Identifier 'action'

These key steps are possible for enforcements of connections :

- `type`: The type of action to enforce. In cases of connections it will be 'mqtt.connect'.
- `isCleanSession`: Whether the client wants to establish a persistent session or not.
- `lastWillTopic`: Topic of the notification message (last will and testament) when client disconnects ungracefully.
- `lastWillQos`: Quality of service level of the notification message when client disconnects ungracefully.
- `lastWillFormatIndicator`: Specifies whether the notification message payload is 'unspecified' or of 'utf-8' encoding.
- `lastWillContentType`: Specifies the type of encoded notification message payload of the message to publish.
- `lastWillPayload`: The payload of the notification message. It is textual in case the last will format indicator indicates a UTF-8 encoding. In other cases it contains the binary payload.

These key steps are possible for enforcements of subscriptions:

- `type`: The type of action to enforce. In cases of subscriptions it will be 'mqtt.subscribe'.
- `qos`: The quality of service level under which the client wants to subscribe for messages.

These key steps are possible for enforcements of publishes:

- `type`: The type of action to enforce. In cases of publishes it will be 'mqtt.publish'.
- `qos`: The quality of service level of the message to publish.
- `isRetain`: Whether the message is saved as the last known value of the specific topic by the broker.

Identifier 'resource'

These key steps are possible for enforcements of connections:

- `brokerName`: The name of the broker that is why it is always 'HiveMQ'.
- `brokerEdition`: The license edition is one of the following: 'community', 'trial', 'professional', 'enterprise'
- `brokerVersion`: The version of the HiveMQ broker. The syntax differs depending on the HiveMQ edition in use. For the community edition it’s "year.release-number", so for example 2019.1 (first release in 2019). For the enterprise edition it’s "major.minor.patch", so for example 4.5.10.

In addition to the key steps for enforcing connections the following key step is also possible for enforcing subscriptions:

- `topic`: The specific topic under which the client wants to subscribe for messages.

In addition to the key steps for enforcing connections the following key steps are also possible for enforcing publish attempts :

- `topic`: The specific topic of the message to publish.
- `formatIndicator`: Specifies whether the payload is 'unspecified' or of 'utf-8' encoding.
- `contentType`: Optionally specifies the type of UTF-8 encoded payload of the message to publish.

Identifier 'environment'

It is possible to use the environment identifier when enforcing a connection. It is just textual and contains the mqtt version of the client (`V_3_1`, `V_3_1_1` or `V_5`).

Constraints

Constraints can be specified in the following format:

```
policy "temperature_policy"
permit
    subject.clientId == "outdoor_thermometer"
where
    action.type == "mqtt.publish";
    resource.topic == "outside_temperature";
obligation
    {
        "type" : "setQos",
            "qosLevel" : 2
    }
```

Constraints are possible to use when a client interacts with the sapl-mqtt-pep. They are objects of one or multiple entries.

When connecting to a broker the connection time can be limited via a constraint of type `limitMqttActionDuration`. With the constraint parameter `timeLimit` the maximal connection time in seconds will be provided.

For subscriptions to the broker the following constraints are possible:

- `limitMqttActionDuration`: Limit the maximal duration of the subscription. The time limit is specified via the parameter `timeLimit` in seconds.
- `resubscribeMqttSubscription`: Whether the sapl-mqtt-pep reestablishes the client subscription by itself when access gets permitted after initial deny. By default, the client stays unsubscribed so that the broker does not reestablish the subscription. Use the constraint parameter `status` to set the status to 'enabled' or 'disabled'. The former status leads to a resubscription.

When publishing a message the following constraints can alter the message send:

- `setQos`: Changing the quality of service level of the message sent with the constraint parameter

`qosLevel`. Possible quality of service levels are 0, 1 and 2.

- `retainMessage`: Altering whether the message should be saved as the last known value of a specific topic or not. With the constraint parameter `status` the retaining can be 'enabled' or 'disabled'.
- `replaceMessageExpiryInterval`: Specifies how long the message will be stored / retained by the broker for further subscribers. Use the `timeInterval` constraint parameter to set the new interval time in seconds.
- `replaceContentType`: Changing the type for the encoded payload of the message to publish. Use the `replacement` constraint parameter to specify the new content type.
- `replacePayload`: Setting a new text for the payload of the message to publish. Use the `replacement` constraint parameter to specify the new payload.
- `blackenPayload`: This constraint is possible in case the payload is a UTF-8 encoded string. It replaces the characters of the string with a specified character. By default, the replacement character is 'X'. Via the parameter 'replacement' you can specify an alternative character. Furthermore, with usage of the parameters 'discloseLeft' and 'discloseRight' you can define the amount of characters from the right and left site that are kept. By default, every character will be replaced.