User Guide

Identity Message Broker 5.5

ForgeRock AS.
201 Mission St., Suite 2900
San Francisco, CA 94105, USA
+1 415-599-1100 (US)
www.forgerock.com

Abstract

Guide to the ForgeRock® Identity Message Broker.

Table of Contents

Preface

ForgeRock Identity Platform™ is the only offering for access management, identity management, user-managed access, directory services, and an identity gateway, designed and built as a single, unified platform.

The platform includes the following components that extend what is available in open source projects to provide fully featured, enterprise-ready software:

- ForgeRock Access Management (AM)
- ForgeRock Identity Management (IDM)
- ForgeRock Directory Services (DS)
- ForgeRock Identity Gateway (IG)
- ForgeRock Identity Message Broker (IMB)

1. About This Guide

The ForgeRock Identity Message Broker (IMB) is a publish-subscribe broker service that secures and hardens the sending and receiving of messages between an MQTT client and the cloud in Internet of Things (IoT) systems.

This guide describes how to set up and run the IMB, providing examples for the following tasks:

- **Token validation on connect**, to validate an id_token and access_token from AM, and open a connection from an MQTT client to the IMB.
- **Authorization on publish**, to verify the authorization of an authenticated MQTT client to publish messages on a given topic.
- **Authorization on receive**, to verify the authorization of authenticated MQTT clients to send and receive messages on a given topic.
- **Connection closure on invalid access_token**, to verify the validity of the access token at a specified interval, and close the connection if the token becomes invalid.

Although this guide provides instructions to run the examples, it will help to be familiar with the basics of authentication and authorization in ForgeRock Access Management.

2. Formatting Conventions

Most examples in the documentation are created in GNU/Linux or Mac OS X operating environments. If distinctions are necessary between operating environments, examples are labeled with the operating environment name in parentheses. To avoid repetition file system directory names are often given only in UNIX format as in `/path/to/server`, even if the text applies to `C:\path\to\server` as well.

Absolute path names usually begin with the placeholder `/path/to/`. This path might translate to `/opt/`, `C:\Program Files\`, or somewhere else on your system.

Command-line, terminal sessions are formatted as follows:

```
$ echo $JAVA_HOME
/path/to/jdk
```

Command output is sometimes formatted for narrower, more readable output even though formatting parameters are not shown in the command.

Program listings are formatted as follows:

```
class Test {
    public static void main(String [] args)  {
        System.out.println("This is a program listing.");
    }
}
```

3. Accessing Documentation Online

ForgeRock publishes comprehensive documentation online:

- The ForgeRock Knowledge Base offers a large and increasing number of up-to-date, practical articles that help you deploy and manage ForgeRock software.

  While many articles are visible to community members, ForgeRock customers have access to much more, including advanced information for customers using ForgeRock software in a mission-critical capacity.

- ForgeRock product documentation, such as this document, aims to be technically accurate and complete with respect to the software documented. It is visible to everyone and covers all product features and examples of how to use them.

4. Using the ForgeRock.org Site

The ForgeRock.org site has links to source code for ForgeRock open source software, as well as links to the ForgeRock forums and technical blogs.

If you are a *ForgeRock customer*, raise a support ticket instead of using the forums. ForgeRock support professionals will get in touch to help you.

5. Getting Support and Contacting ForgeRock

ForgeRock provides support services, professional services, classes through ForgeRock University, and partner services to assist you in setting up and maintaining your deployments. For a general overview of these services, see https://www.forgerock.com.

ForgeRock has staff members around the globe who support our international customers and partners. For details, visit https://www.forgerock.com, or send an email to ForgeRock at info@forgerock.com.

Chapter 1

About the Identity Message Broker

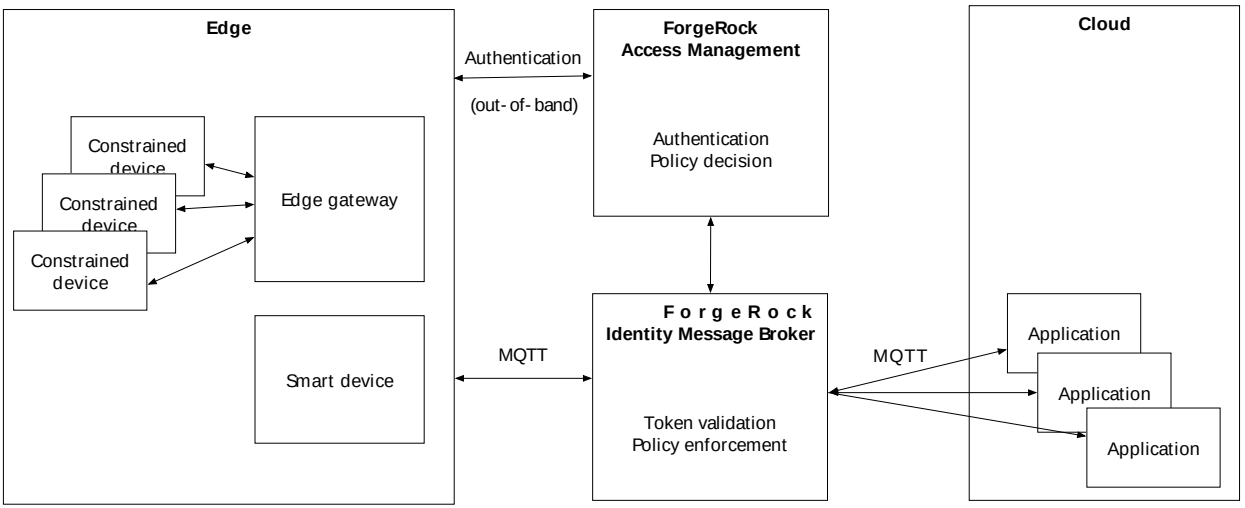
The ForgeRock Identity Message Broker (IMB) is a publish-subscribe broker service that secures and hardens the sending and receiving of messages between an MQTT client and the cloud in Internet of Things (IoT) systems.

The architecture described in this document starts at the edge, with a smart device, or with a constrained device that talks to an edge gateway.

ForgeRock Access Management acts as an authorization server, to authenticate an MQTT client and provide it with an OpenID Connect id_token and access token. The client authenticates to the IMB, using an id_token as its username and an access_token as its password. The IMB validates the tokens, and enforces policies decisions from ForgeRock Access Management.

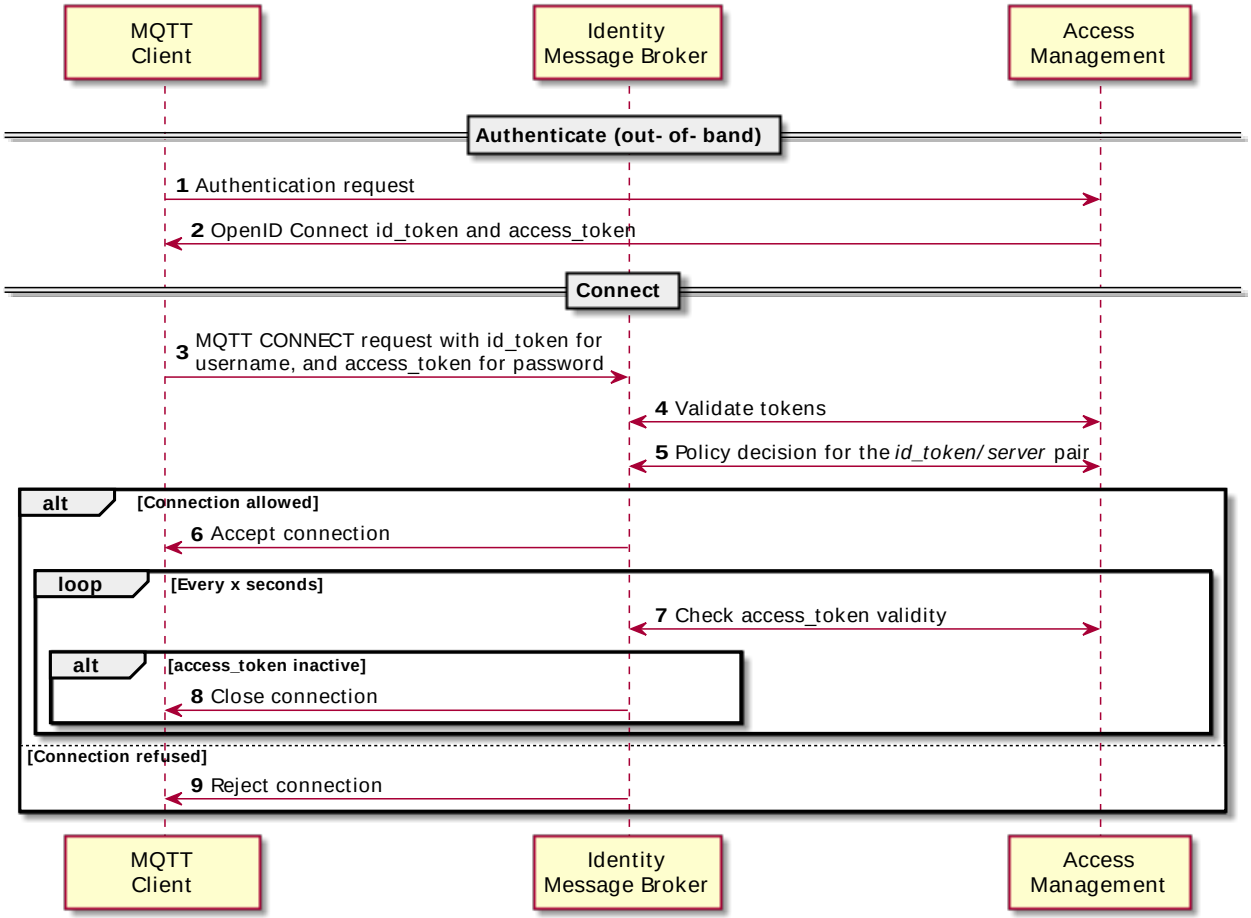
The following figure illustrates the architecture of an IoT system, with the IMB positioned to manage traffic between the edge and the cloud.

Figure 1.1. IoT System Architecture



The following figures illustrate the data flow when an MQTT client successfully publishes a message on a topic to the cloud, and then receives a message about another topic from the cloud:

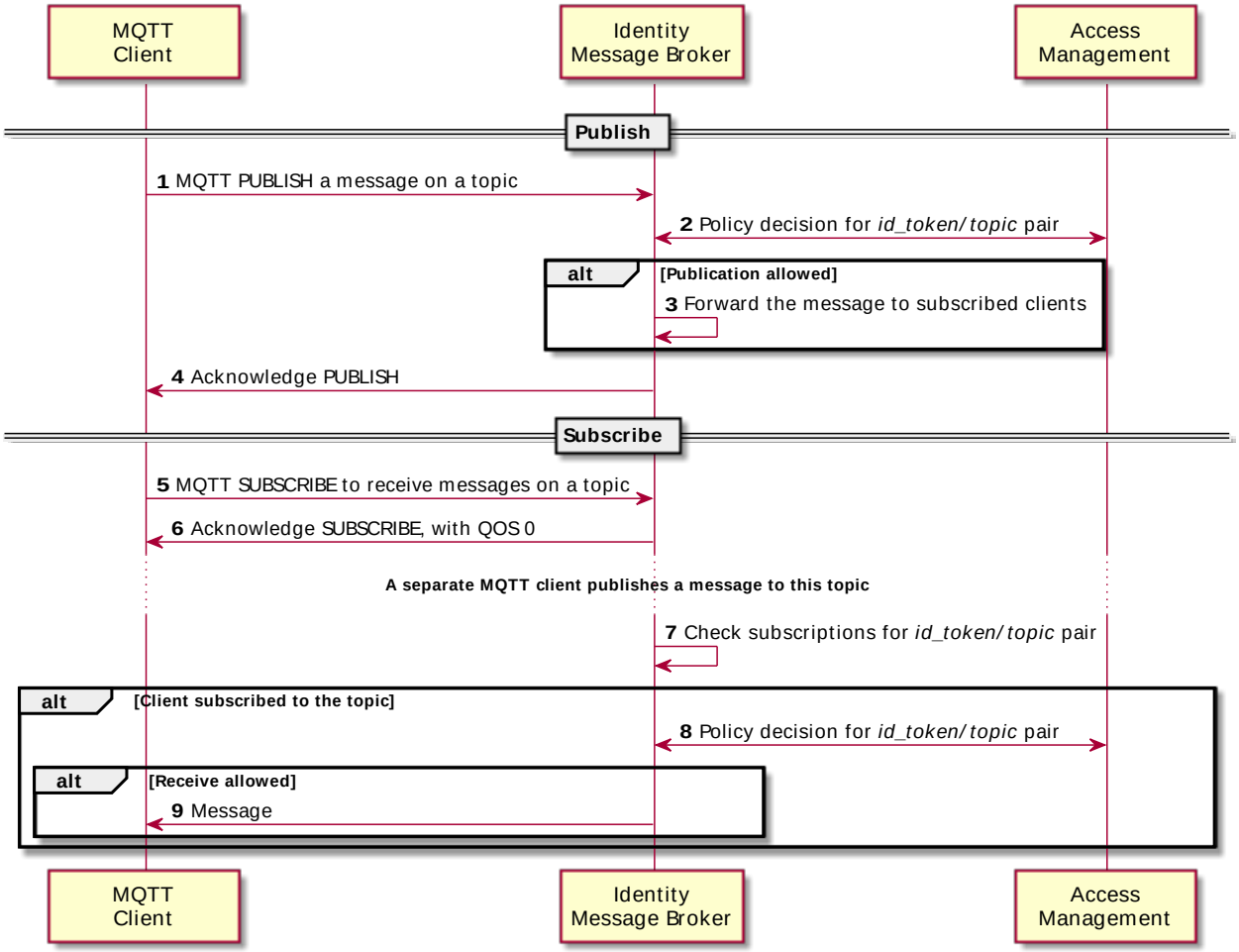
Figure 1.2. IoT Data Flow for Connect



Notice the following points about the data flow:

- The IMB checks the validity of the id_token and access_token presented in the connect request from the MQTT client. If the IMB can validate both tokens, it accepts the connection initiated by the client. Otherwise, it rejects the connection with a message indicating a bad username or password.
- After the IMB accepts a connection, it periodically checks the validity of the access_token. If the token becomes invalid (expires or is revoked), the IMB closes the connection.

Figure 1.3. IoT Data Flow for Publish and Receive



Notice the following points about the data flow:

- In the publish stage, if the MQTT client does not have the correct authorization to publish a messages for the given topic the IMB silently ignores the publish command.
- If there is no client subscribed to receive a message when it is published, the message is lost. The IMB doesn't retain messages.

  Ignored messages are traced at the debug level in the logs.

Chapter 2

Installing and Starting the Identity Message Broker

2.1. Required Software

The following software is required by the IMB:

- **Java 8**
- **ForgeRock Access Management 5.1 or a later version**

  AM is used to authenticate MQTT clients, authenticate the Identity Message Broker, and provide a policy decision point. For information, see the *Access Management documentation*.

The examples in Chapter 3, "*Running MQTT Publish and Receive*" use **Mosquitto MQTT client** for the MQTT client and cloud application. For information and downloads, see https://mosquitto.org.

2.2. Downloading and Installing the IMB

Procedure 2.1. To Download and Install IMB

1. Download `IMB-5.5.1.zip` from the Edge Security area of ForgeRock BackStage download site .
2. Copy (or move) and unzip the file to the installation directory:

   ```
   $ mkdir /IMB
   $ unzip ~/Downloads/IMB-5.5.1.zip -d /IMB
   ```

   The following files and directories are extracted in `/IMB/imb-microservice`:

   - `/bin/startup.sh`, to start the IMB.
   - `/conf/boot/boot.properties`, to configure port numbers and keys for starting the IMB.
   - `/conf/logging.properties`, to configure logging levels for messages to the console and file. By default, messages are logged to console at the level `INFO`, and to file at the level `ALL`.
   - `/logs`, the default location for log files.

- `/examples/imb_mqtt_server-default.json`, the MQTT server configuration file. For more information and an example file, see Section 2.3.1, "Configuration File for MQTT Server".

  If you plan to use MQTTS for the MQTT server connection, this file must contain parameters to require SSL, and to define the path to the keystore file. For an example file, see Example 4.1, "Example Configuration Requiring MQTTS for the MQTT Server Connection".

- `/examples/imb_capture.json`, to configure a print to logs for authorized messages to the IMB. For more information and an example file, see Section 2.3.2, "Configuration File for Capture".

- `/amster`, to provide the AM configuration used in Appendix A, "*Setting Up AM Manually for the Examples*". Instead of configuring AM manually you can import the configuration through Amster, as described in that section.

2.3. Example Configuration Files for the IMB

The IMB is delivered with example configuration files that you can use as they are or use as a template for your configuration. Your configuration must contain files with the same name as these files.

2.3.1. Configuration File for MQTT Server

The example configuration file, `/examples/imb_mqtt_server-default.json`, defines the following properties:

- Host and port number that MQTT clients use to connect to the IMB.
- Information for the IMB to connect with AM, and the policy set and realms to use. These values must match those in Appendix A, "*Setting Up AM Manually for the Examples*":
- The interval at which the IMB checks the validity of the access_token presented as a password for the MQTT command. The default is 15000 ms.

```
{
  "config" : {
    "mqtt" : {
      "host" : "0.0.0.0",
      "port" : 1883
    },
    "security" : {
      "openam-uri" : "http://openam.example.com:8088/openam",
      "policy-set" : "things",
      "operator" : {
        "username" : "imb_oidc_client",
        "password" : "password",
        "realm" : "/"
      }
    }
```

```
  }
}
```

2.3.2. Configuration File for Capture

The example configuration file, `/examples/imb_capture.json` contains the following configuration to capture messages coming into the IMB:

```
{}
```

2.4. Starting the IMB

This section describes how to start up the IMB and then make sure that it started successfully.

Procedure 2.2. To Start the IMB

Before you start, download and install the IMB as described in Section 2.2, "Downloading and Installing the IMB". The example commands are run from the installation directory `/IMB/imb-microservice`.

1. Run the script to start the IMB:

   ```
   $ bin/startup.sh
   INFO    | com.forgerock.imb.osgi.VertxServiceComponent | Vert.x successfully
    registered
   -> Service ready
   ```

   A message indicates that the microservices framework is ready.

   The bootstrap process tries to activate the IMB components by going through all of the .json files in the `/conf` directory. If no `imb_mqtt_server-****.json` is present, the IMB MQTT Server component isn't activated.

2. If you intend to run the examples, go directly to Appendix A, "*Setting Up AM Manually for the Examples*". Otherwise, perform this step.

   Add the MQTT server configuration file called `imb_mqtt_server-default.json` to the `/conf` directory. For an example file, see Section 2.3.1, "Configuration File for MQTT Server".

   The IMB takes a few seconds to update its configuration automatically, and displays the following message:

   ```
   INFO | com.forgerock.imb.mqtt.server.MqttServerVerticle | MQTT Server is listening on localhost:1883
   ```

Chapter 3

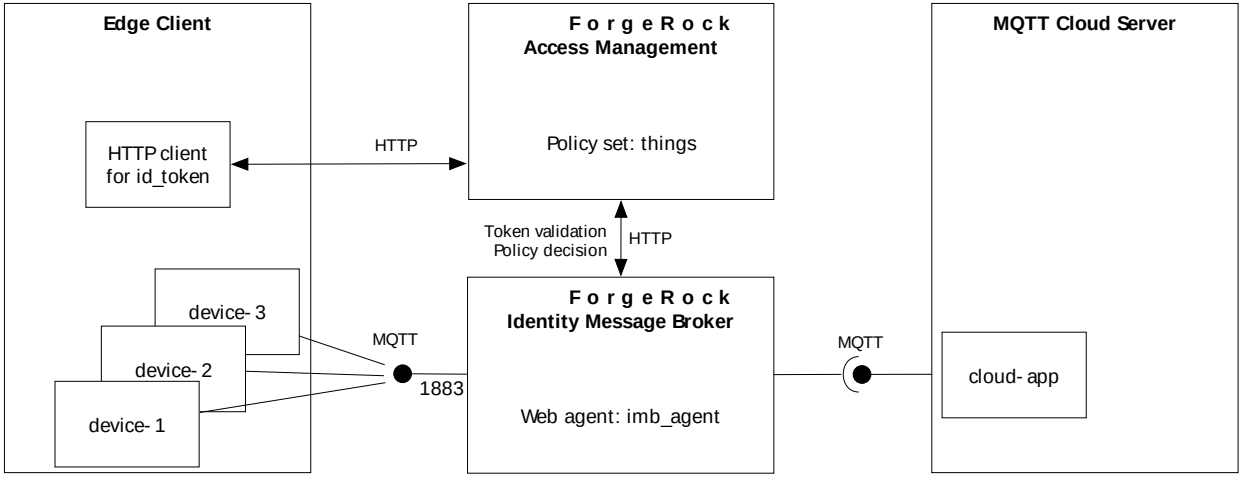
Running MQTT Publish and Receive

This chapter describes how to set up and run examples for the following features of the IMB:

- **Token validation on connect**, to validate an id_token and access_token from AM, and open a connection from an MQTT client to the IMB.
- **Authorization on publish**, to verify the authorization of an authenticated MQTT client to publish messages on a given topic.
- **Authorization on receive**, to verify the authorization of authenticated MQTT clients to send and receive messages on a given topic.
- **Connection closure on invalid access_token**, to verify the validity of the access token at a specified interval, and close the connection if the token becomes invalid.

The following figure illustrates the architecture used in the examples.

Figure 3.1. Example Architecture



The examples in this document use `device-1` and `cloud-app`.

Authentication, resource types, and policies are configured in AM to determine how `device-1` and `cloud-app` authenticate, send messages to each other, and receive messages from each other.

AM is installed on `http://openam.example.com:8088/openam`, and the IMB on port `1883`. If you set up the examples differently, adjust the instructions.

By default, messages are logged to the console at the level `INFO`, and to file at the level `ALL`. To change the logging levels, edit `/conf/boot/logging.properties`.

The examples are tested on macOS and native Linux hosts.

3.1. Preparing for the Examples

This section describes how to prepare the examples. Before you start, install the IMB as described in Chapter 2, "*Installing and Starting the Identity Message Broker*".

3.1.1. Setting Up AM With Amster

This section describes how to set up authentication, resource types, and policies in AM by importing the configuration through Amster. Amster sets up two devices, `device-1` and `device-2`, and the `cloud-app`.

Alternatively, you can set up AM manually, as described in Appendix A, "*Setting Up AM Manually for the Examples*". For simplicity, the manual configuration sets up only one device (`device-1`) and the `cloud-app`.

The configuration is provided with the IMB delivery, in the `/amster` directory. After importing the configuration, you must separately set up subjects in the AM console, as described in Procedure 3.2, "To Set Up the Example Subjects".

For information how to get started and use Amster, see the *Amster User Guide*. For other information about Amster, see the Amster documentation.

Caution

Amster imports overwrite any configuration that already exists in the target AM instance.

Procedure 3.1. To Import the AM Configuration From Amster

- With AM running on `http://openam.example.com:8088/openam`, and Amster running as described in the Amster documentation, run commands similar to the following to import the AM configuration:

```
$ cd /path/to/amster

/path/to/amster$ ./amster

am> connect --interactive http://openam.example.com:8088/openam
Sign in
User Name: amadmin
Password: ********

amster openam.example.com:8088> import-config --path /path/to/IMB/imb-microservice/amster/config-am51/
amster-imb
...
Import completed successfully
```

Procedure 3.2. To Set Up the Example Subjects

1. Log in to the AM console as admin, select the top-level realm, and select Subjects.
2. Add a subject for device 1, with the following properties:
   - ID: `device-1`
   - Last Name: `device-1`
   - Full Name: `device-1`
   - Password: `changeit`
3. Add a subject for the cloud application, with the following properties:
   - ID: `cloud-app`
   - Last Name: `cloud-app`
   - Full Name: `cloud-app`
   - Password: `changeit`

3.1.2. Getting ID Tokens

This section describes how to run a **curl** command to get an id_token and access_token from AM for each of the subjects you created in Procedure 3.2, "To Set Up the Example Subjects". You can alternatively use another grant flow to get the tokens.

1. In a terminal window, run the following **curl** command for `device-1`, where `oidc_client` and `password` are the credentials of the OAuth 2.0 client in AM:

```
$ curl \
-s --insecure --request POST --user oidc_client:password
 \
--header 'Content-Type: application/x-www-form-urlencoded' \
'http://openam.example.com:8088/openam/oauth2/access_token?realm=/
&grant_type=password&scope=openid&username=device-1&password=changeit'

{
"access_token": "c6df8464-. . .",
"scope": "openid",
"id_token": "eyJ0eXAiOiJKV . . .",
"token_type": "Bearer",
"expires_in": 3599
}
```

2. Run a similar **curl** command for `cloud-app`:

```
$ curl \
-s --insecure --request POST --user oidc_client:password
 \
--header 'Content-Type: application/x-www-form-urlencoded' \
'http://openam.example.com:8088/openam/oauth2/access_token?realm=/
&grant_type=password&scope=openid&username=cloud-app&password=changeit'
```

3. Make a note of each of the tokens, which are referred to later as follows:

   - `access_token-device-1`
   - `id_token-device-1`
   - `access_token-cloud-app`
   - `id_token-cloud-app`

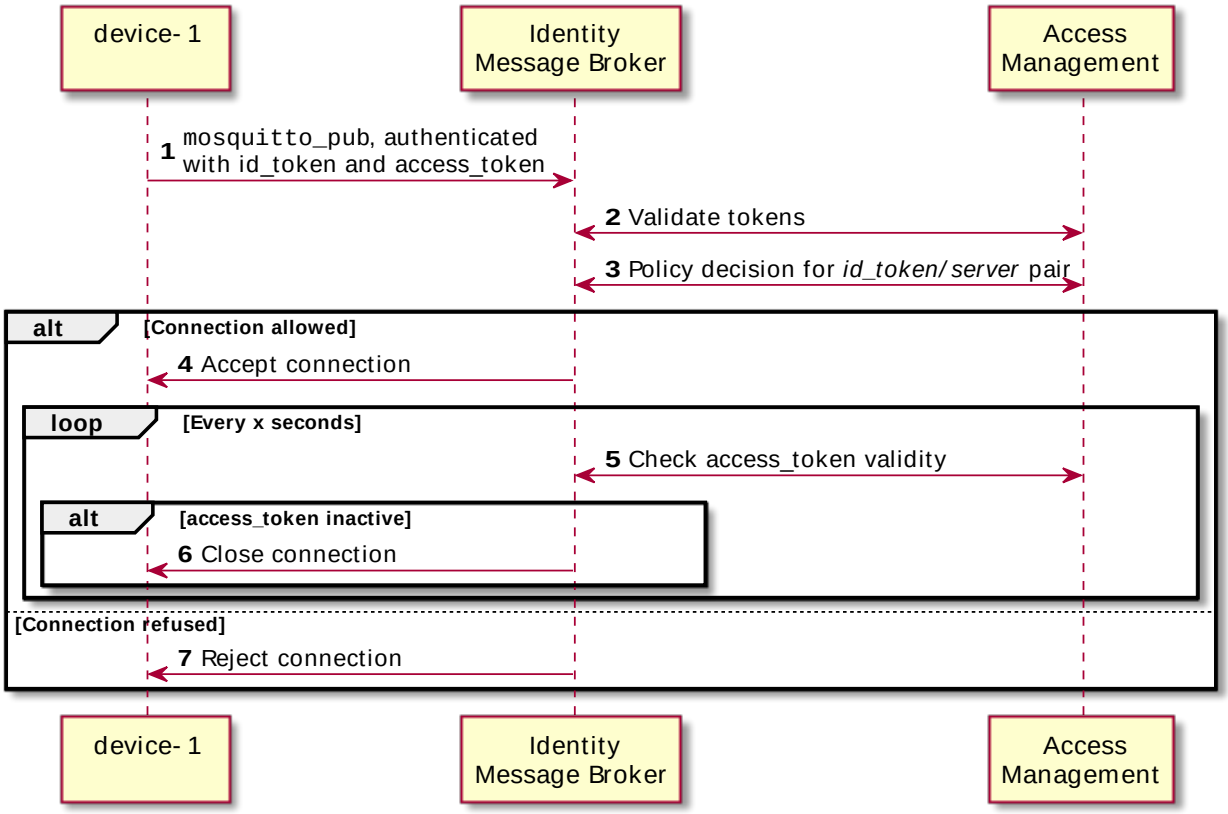
3.2. Validating a Token for Connect

In this example, `device-1` authenticates by using an id_token as its username and an access_token as its password.

If the IMB validates the tokens, it accepts the connection from the `device-1`. Otherwise, it rejects the connection. A message is displayed in the `device-1` terminal.

The following figure illustrates the data flow:

Figure 3.2. Data Flow for Token Validation



Procedure 3.3. To Validate a Token for Connect

Before you start, make sure that you have completed all of the tasks in Section 3.1, "Preparing for the Examples".

1. In a terminal window for IMB, start the IMB:

```
$ /IMB/imb-microservice/bin/startup.sh
INFO    | com.forgerock.imb.osgi.VertxServiceComponent | Vert.x successfully
 registered
-> Service ready
```

2. Include the configuration to set up the connections to the IMB:

   a. Copy `/examples/imb_mqtt_server-default.json` to the `/conf` directory:

```
{
  "config" : {
    "mqtt" : {
      "host" : "0.0.0.0",
      "port" : 1883
    },
    "security" : {
      "openam-uri" : "http://openam.example.com:8088/openam",
      "policy-set" : "things",
      "operator" : {
        "username" : "imb_oidc_client",
        "password" : "password",
        "realm" : "/"
      }
    }
  }
}
```

For information about the file, see Section 2.3.1, "Configuration File for MQTT Server".

b. Check the IMB terminal for an updated message like this:

```
INFO | com.forgerock.imb.mqtt.server.MqttServerVerticle | MQTT Server is listening on 0.0.0.0:1883
```

3. Publish a message anonymously:

a. In a terminal for `device-1`, run the following command:

```
$ mosquitto_pub -h localhost -p 1883 -t "/device-1/messages" -m "1266193804 32" -d
```

`device-1` tries to connect to the IMB without providing credentials.

b. Check the `device-1` terminal for a message to confirm that the connection was refused:

```
Client mosqpub/...- sending CONNECT
Client mosqpub/...- received CONNACK
Connection Refused: bad user name or password.
Error: The connection was refused.
```

4. Publish a message with credentials:

a. Run the following command, replacing *<id_token-device-1>* and *<access_token-device-1>* with the values returned in Section 3.1.2, "Getting ID Tokens":

```
$ mosquitto_pub -h localhost -p 1883 -u "<id_token-device-1>" -P "<access_token-device-1>" -t "/
device-1/messages" -m "1266193804 32" -d
```

b. Check the `device-1` terminal for a message to confirm that the that the IMB accepted the connection:

```
Client mosqpub/...- sending CONNECT
Client mosqpub/...- received CONNACK
Client mosqpub/...- sending PUBLISH (d0, q0, r0, m1, '/device-1/messages', ... (10 bytes))
Client mosqpub/...- sending DISCONNECT
```

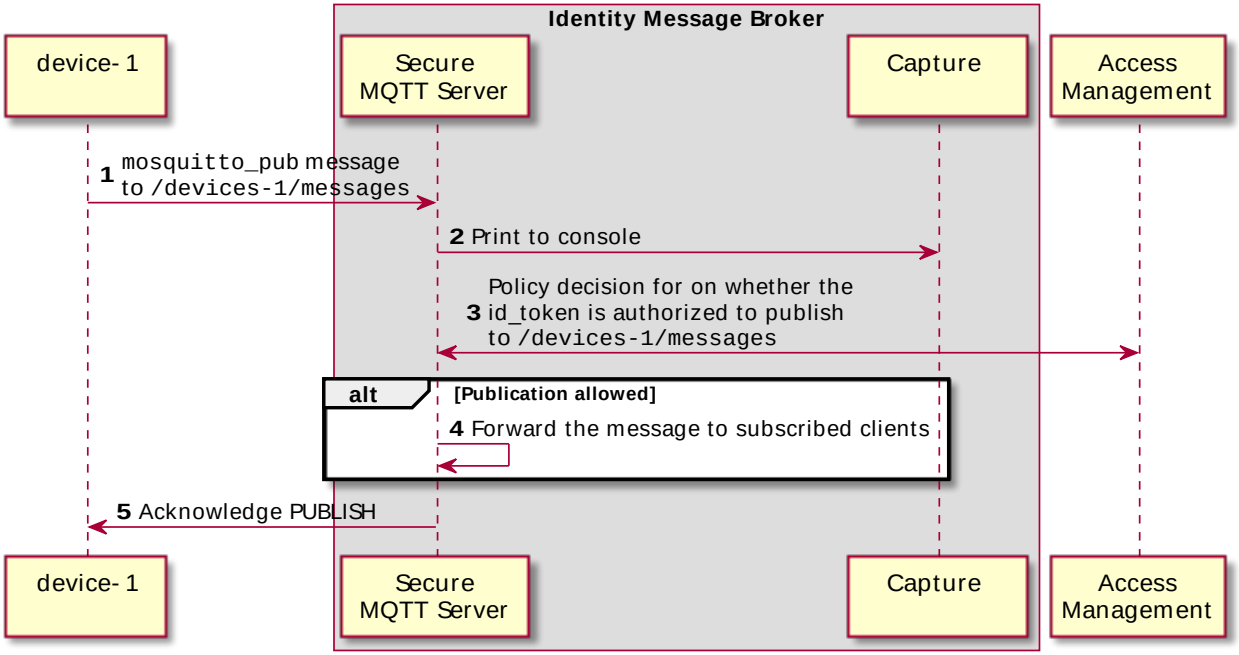
3.3. Authorizing on Publish

This example builds on Section 3.2, "Validating a Token for Connect" to publish a message to a topic and route the message to a capture.

The IMB requests a policy decision from AM on whether the `device-1` is authorized to publish to the topic. If it is authorized, the IMB routes the message to a capture component that prints the message in the IMB terminal.

The following figure illustrates the data flow for this example:

Figure 3.3. Data Flow for Authorization on Publish



Procedure 3.4. To Authorize on Publish

Before you start, work through Section 3.2, "Validating a Token for Connect".

1. Include configuration to capture messages coming into the IMB:

   a. Copy `/examples/imb_capture.json` to the `/conf` directory.

      The file contains the following configuration to capture messages coming into the Identity Message Broker:

```
{}
```

For information about the file, see Section 2.3.2, "Configuration File for Capture".

b. Check the IMB terminal for a message to confirm that the configuration has been updated:

```
INFO | com.forgerock.imb.mqtt.server.MqttServerVerticle | MQTT Server is listening on 0.0.0.0:1883
INFO | com.forgerock.imb.client.text.TextualClientVerticle | Textual Client started>
```

2. Publish a message with authentication, but to an unauthorized topic:

a. In the `device-1` terminal, run the following command, to publish a message to `/temperature`, replacing *<id_token-device-1>* and *<access_token-device-1>* with the values returned in Section 3.1.2, "Getting ID Tokens":

```
$ mosquitto_pub -h localhost -p 1883 -u "<id_token-device-1>" -P "<access_token-device-1>" -t "/
temperature" -m "1266193804 32" -d
Client mosqpub/....- sending CONNECT
Client mosqpub/....- received CONNACK
Client mosqpub/....- sending PUBLISH (d0, q0, r0, m1, '/temperature', ...)
Client mosqpub/....- sending DISCONNECT
```

Although an AM policy authorizes `device-1` to publish on `/device-1/messages` there is no policy to authorize it to publish on `/temperature`.

b. In the IMB terminal, check that there is **no message** to indicate that the IMB accepted the publish.

c. View the IMB logs for a message that the publish was not allowed.

3. Publish a message with authentication, but to an authorized topic:

a. Run the following command to publish a message to `/device-1/messages`:

```
$ mosquitto_pub -h localhost -p 1883 -u "<id_token-device-1>" -P "<access_token-device-1>" -t "/
device-1/messages" -m "1266193804 32" -d
Client mosqpub/....- sending CONNECT
Client mosqpub/....- received CONNACK
Client mosqpub/....- sending PUBLISH (d0, q0, r0, m1, '/device-1/messages', ... (10 bytes))
Client mosqpub/....- sending DISCONNECT
```

b. In the IMB terminal, check that there **is** a message to indicate that the IMB received the publish message from `device-1`, and the capture component was able to display it:

```
Message received on /device-1/messages
* issuer: device-1
```

3.4. Authorizing on Receive

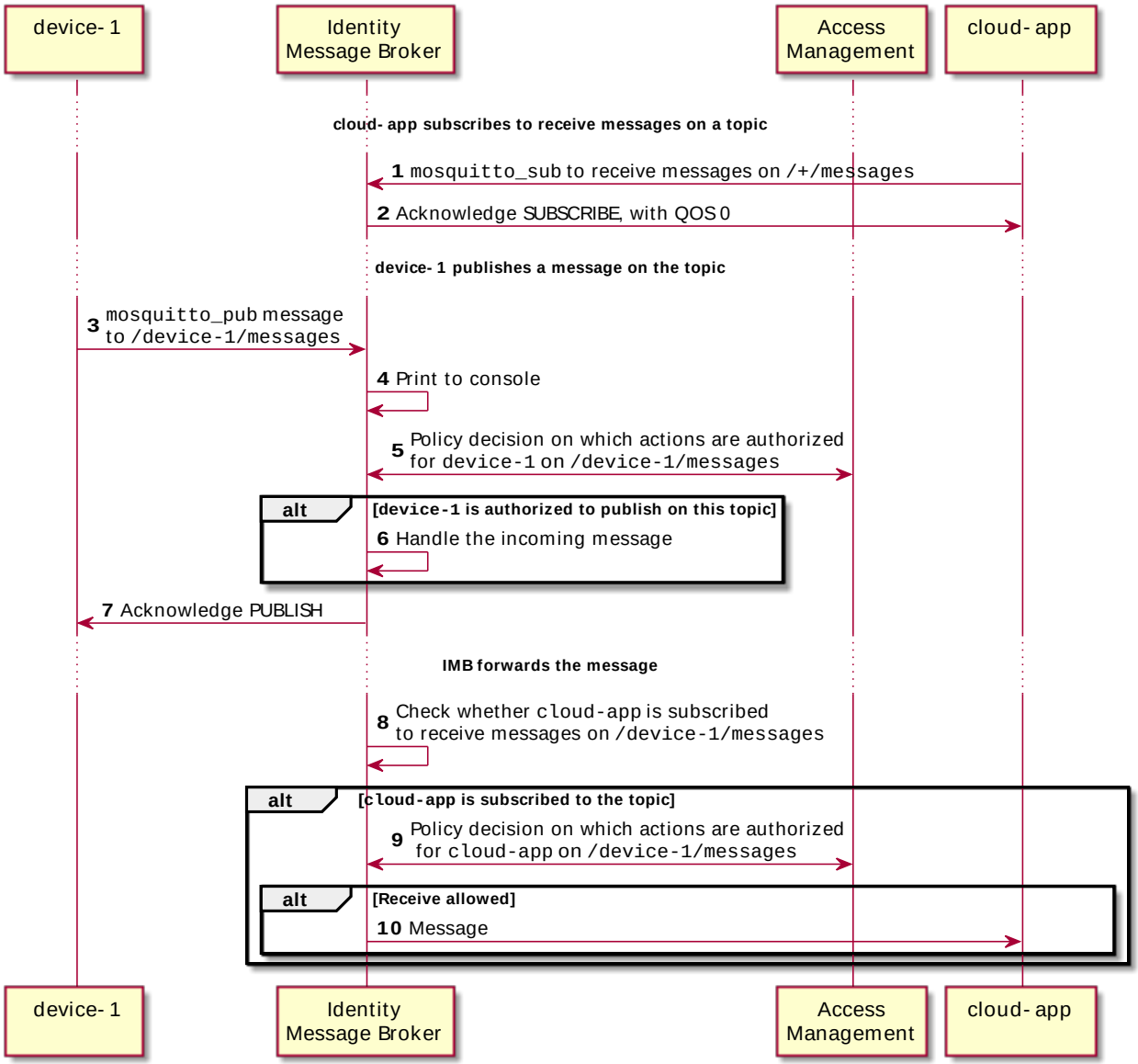
This example builds on Section 3.3, "Authorizing on Publish", where `cloud-app` subscribes to receive messages on the topic `/+/messages`, and `device-1` subscribes to receive messages on `/+/actions`.

3.4.1. Receiving Messages on /device-1/messages

In this section, `cloud-app` subscribes to receives messages on the topic `/+/messages`. When `device-1` publishes a message on `/device-1/messages`, `cloud-app` receives the message.

The following figure illustrates the data flow for this example:

Figure 3.4. Data Flow for Authorization on Receive



Procedure 3.5. To Receive Messages on /device-1/messages

1. In a terminal for `cloud-app`, run the following **subscribe** command, replacing *<id_token-cloud-app>* and *<access_token-cloud-app>* with the values returned in Section 3.1.2, "Getting ID Tokens":

```
$ mosquitto_sub -h localhost -p 1883 -u "<id_token-cloud-app>" -P "<access_token-cloud-app>" -t "/+/
messages" -d
Client mosqsub... sending CONNECT
Client mosqsub/... received CONNACK
Client mosqsub/... sending SUBSCRIBE (Mid: 1, Topic: /+/messages, QoS: 0)
Client mosqsub/... received SUBACK
Subscribed (mid: 1): 0
```

2. In the `device-1` terminal, run the following command to publish the example message `1266193804 32`. This message contains a time stamp and a temperature reading:

```
$ mosquitto_pub -h localhost -p 1883 -u "<id_token-device-1>" -P "<access_token-device-1>" -t "/device
-1/messages" -m "1266193804 32" -d
Client mosqpub/...- sending CONNECT
Client mosqpub/...- received CONNACK
Client mosqpub/...- sending PUBLISH (d0, q0, r0, m1, '/device-1/messages', ... (10 bytes))
Client mosqpub/...- sending DISCONNECT
```

3. In the IMB terminal, check that the IMB received the publish message from `device-1`

```
Message received on /device-1/messages
* issuer: device-1
```

4. In the `cloud-app` terminal, confirm that `cloud-app` received the message `1266193804 32`:

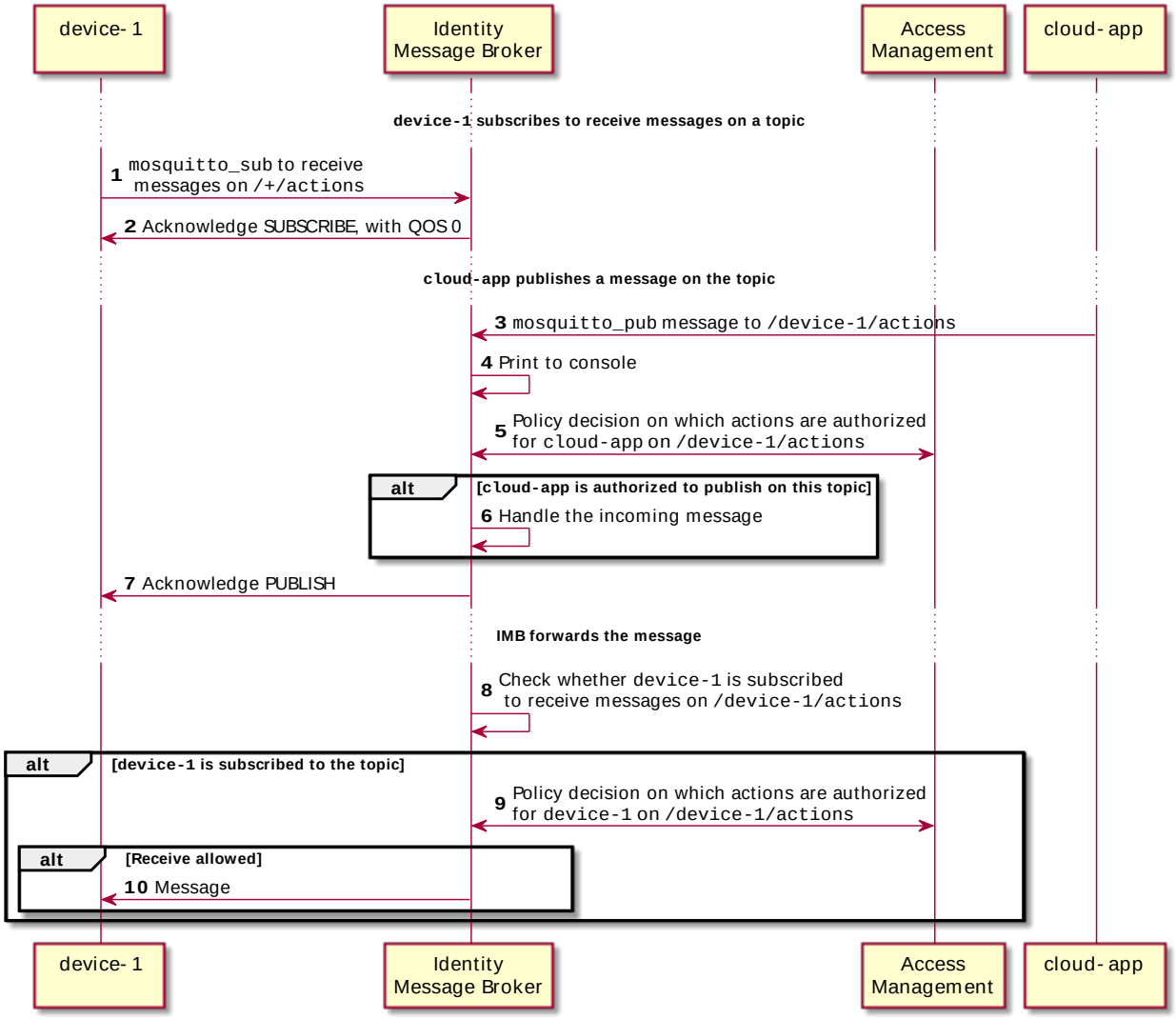
```
Client mosqsub/... received PUBLISH (d0, q0, r0, m0, '/device-1/messages', ... (13 bytes))
1266193804 32
```

3.4.2. Receiving Messages on /device-1/actions

This section sets out the inverse data flow of the previous section. Here, `device-1` subscribes to receive messages on the topic `/+/actions`. When `cloud-app` publishes a message on `/device-1/actions`, `device-1` receives the message.

The following figure illustrates the data flow for this example:

Figure 3.5. Data Flow for Authorization on Receive



Procedure 3.6. To Receive Messages on `/device-1/actions`

1. In the `device-1` terminal, run the following subscribe command to receive messages on the topic `/+/actions`:

```
$ mosquitto_sub -h localhost -p 1883 -u "<id_token-device-1>" -P "<access_token-device-1>" -t "/+/
actions" -d
Client mosqsub... sending CONNECT
Client mosqsub/... received CONNACK
Client mosqsub/... sending SUBSCRIBE (Mid: 1, Topic: /+/actions, QoS: 0)
Client mosqsub/... received SUBACK
Subscribed (mid: 1): 0
```

2. In the `cloud-app` terminal, run the following command to publish the example message `***stop***` to the topic `/device-1/actions`. This message contains an instruction to the device:

```
$ mosquitto_pub -h localhost -p 1883 -u "<id_token-cloud-app>" -P "<access_token-cloud-app>" -t "/
device-1/actions" -m "***stop***" -d
Client mosqpub/...- sending CONNECT
Client mosqpub/...- received CONNACK
Client mosqpub/...- sending PUBLISH (d0, q0, r0, m1, '/device-1/actions', ... (10 bytes))
Client mosqpub/...- sending DISCONNECT
```

3. In the IMB terminal, check that the IMB received the publish message from `cloud-app`:

```
Message received on /device-1/actions
* issuer: cloud-app
```

4. In the `device-1` terminal, confirm that `device-1` received the message `***stop***`:

```
Client mosqsub/... received PUBLISH (d0, q0, r0, m0, '/device-1/actions', ... (13 bytes))
***stop***
```

3.5. Closing Connections on Invalid access_token

When the IMB is connected to an MQTT client, the IMB regularly checks the validity of the access_token used to create the connection. If the token becomes invalid, for example it expires or is revoked, the IMB closes the connection to the client.

The IMB checks the validity of access_tokens at the interval specified by the `authentication-check-interval` property in `imb_mqtt_server-default.json`. The default is 15000 ms.

In the previous examples, the MQTT clients used an access_token as a password for publish and subscribe commands. In this example, the token is revoked. The IMB automatically detects that the token has become invalid, and immediately closes the connection to the client. The client can no longer publish or subscribe with the token.

This example uses an OAuth 2.0 client in AM with the scope to introspect tokens.

Procedure 3.7. To Revoke an access_token for Connection Closure

Before you start, work through Section 3.3, "Authorizing on Publish".

1. In a terminal window, run the following command to introspect the access_token for `device-1`:

```
$ http POST openam.example.com:8088/openam/oauth2/realms/root/introspect client_id==imb_oidc_client
 client_secret==password token==<access_token-device-1>
HTTP/1.1 200
 OK
...
{
  "active": true,
  "client_id": "oidc_client",
  "exp": 1505386947,
  "iss": "http://openam.example.com:8088/openam/oauth2",
  "scope": "openid",
  "sub": "device-1",
  "token_type": "access_token",
  "user_id": "device-1"
}
```

Note that the token is active.

2. Publish a message from `device-1`:

   a. In a terminal for `device-1` publish a message to `/device-1/messages`:

   ```
   $ mosquitto_pub -h localhost -p 1883 -u "<id_token-device-1>" -P "<access_token-device-1>" -t "/
   device-1/messages" -m "1266193804 32" -d
   Client mosqpub/....- sending CONNECT
   Client mosqpub/....- received CONNACK
   Client mosqpub/....- sending PUBLISH (d0, q0, r0, m1, '/device-1/messages', ... (10 bytes))
   Client mosqpub/....- sending DISCONNECT
   ```

   b. In the IMB terminal, check that the IMB received the message:

   ```
   Message received on /device-1/messages
   * issuer: device-1
   ```

3. Revoke the access_token for `device-1`:

   a. In a terminal window, run the following command to revoke the access_token:

   ```
   $ http POST openam.example.com:8088/openam/oauth2/token/revoke client_id==oidc_client
    client_secret==password token==<access_token-device-1>
   HTTP/1.1 200
    OK
   ...
   {}
   ```

   b. Introspect the token to check that it is revoked:

   ```
   $ http POST openam.example.com:8088/openam/oauth2/realms/root/introspect
    client_id==imb_oidc_client client_secret==password token==<access_token-device-1>
   HTTP/1.1 200
    OK
   ...
   {
     "active": false
   }
   ```

Note that the token is not active.

4. Try to publish a message from `device-1`:

   a. In a terminal for `device-1` publish a message to `/device-1/messages`:

   ```
   $ mosquitto_pub -h localhost -p 1883 -u "<id_token-device-1>" -P "<access_token-device-1>" -t "/
   device-1/messages" -m "1266193804 32" -d
           Client mosqpub|... sending CONNECT
   Client mosqpub|... received CONNACK
   Connection Refused: bad user name or password.
   Error: The connection was refused.
   ```

   b. In the IMB terminal, check for an error message:

   ```
   | SEVERE  | com.forgerock.imb.mqtt.server.MqttServerVerticle | An error occurred while resolving
    the access_token
   ```

   The log files include other notifications that the connection was rejected.

5. If you set up AM using Amster, consider using `device-2` to test the policies. For example:

   - Check that `device-2` does not receive messages on the topic `/device-1/messages`.
   - Check that when `cloud-app` publishes a message to `/device-1/actions`, `device-2` does not receive it.

Chapter 4

Securing Communications Between MQTT Clients and the IMB

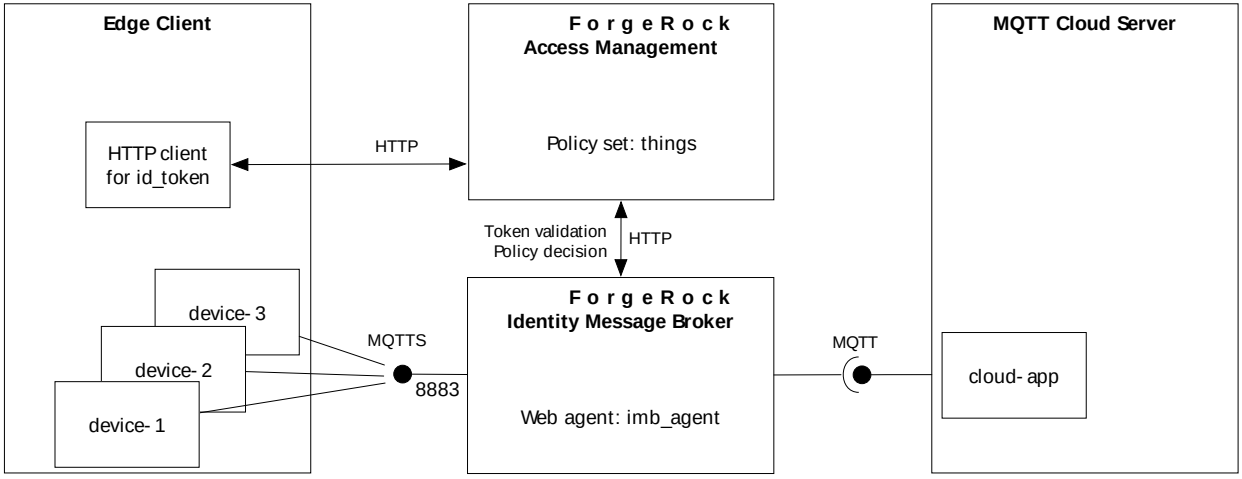
This chapter describes how set up secure communications between MQTT clients and the IMB by using MQTTS for the MQTT server connection, and requiring mutual authentication between MQTT clients and the IMB.

Before you start, install the IMB as described in Chapter 2, "*Installing and Starting the Identity Message Broker*" and Appendix A, "*Setting Up AM Manually for the Examples*". IMB and AM must be running.

4.1. Using MQTTS for the MQTT Server Connection

This section describes how to configure MQTTS for the MQTT server connection. The following figure highlights the position of the MQTT server connection in the example:

Figure 4.1. Example Architecture With MQTTS for the MQTT Server Connection



Procedure 4.1. To Set up MQTTS for an MQTT Server Connection

Before you start, complete the setup and examples in Chapter 3, "*Running MQTT Publish and Receive*".

1. In AM, make sure that the policies include resources for the MQTTS port number:
   - If you configured AM with the provided Amster files, the policies are configured.
   - If you configured AM manually, see Section A.4, "Configuring a Policy Set and Policies" to make sure that the `default-mqtt-server` policy contains the resource `mqtt+server://*:8883`.
2. Generate a keystore using the Java **keytool**, replacing where necessary the example values with your own values:
   a. Generate an SSL key pair and keystore:

```
$ keytool \
-genkeypair
 \
-alias imb
 \
-keyalg rsa
 \
-keystore /path/to/server_keystore.jks
 \
-storepass password
 \
-keypass password
 \
-dname "CN=Unknown, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown"
```

   Note

   Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

   A file `/path/to/server_keystore.jks` is created.

   b. Export the public certificate from the keystore:

```
$ keytool \
-exportcert
 \
-alias imb
 \
-keystore /path/to/server_keystore.jks
 \
-keypass password
 \
-rfc
 \
-file server_cert.pem
Enter keystore password: password
Certificate stored in file <server_cert.pem>
```

A file `/path/to/server_cert.pem` is created. MQTT clients must use this certificate to connect to the IMB.

3. Update the configuration file `imb_mqtt_server-default.json` to require SSL, and to set the SSL endpoint and path to the keystore file. Use the following file as an example:

Example 4.1. Example Configuration Requiring MQTTS for the MQTT Server Connection

```
{
  "config" : {
    "mqtt" : {
      "host" : "0.0.0.0",
      "port" : 8883,
      "ssl" : true,
      "keyStoreOptions" : {
        "path" : "/path/to/server_keystore.jks",
        "password" : "password"
      }
    },
    "security" : {
      "openam-uri" : "http://openam.example.com:8088/openam",
      "policy-set" : "things",
      "operator" : {
        "username" : "imb_oidc_client",
        "password" : "password",
        "realm" : "/"
      }
    },
    "authentication-check-interval" : 15000
  }
}
```

The message on the IMB terminal should confirm that it is listening on the secure port `8883`:

```
INFO | com.forgerock.imb.mqtt.server.MqttServerVerticle | MQTT Server is listening on 0.0.0.0:8883
```

Compared to Section 2.3.1, "Configuration File for MQTT Server", this file uses a different port number, includes the `ssl` boolean, and defines the path and password for the keystore file.

Procedure 4.2. To Test the Setup

1. In the `device-1` terminal, run the following command to publish a message without using the certificate file `server_cert.pem` to secure the connection:

```
$ mosquitto_pub -h localhost -p 8883 -u "OIDC token" -P "<id_token-device-1>" -t "/device-1/messages"
-m "1266193804 32" -d
Client mosqpub/... sending CONNECT
Error: The connection was lost.
```

   The message on the IMB terminal should confirm that the client failed to connect over SSL:

```
Client from origin /... failed to connect over ssl:...
```

2. Now run the following command, using the certificate file `server_cert.pem` to secure the connection:

```
$ mosquitto_pub -h localhost -p 8883 --cafile /path/to/server_cert.pem --insecure -u "<id_token-device
-1>" -P "<access_token-device-1>" -t "/device-1/messages" -m "1266193804 32" -d
Client mosqpub/... sending CONNECT
Client mosqpub/... received CONNACK
Client mosqpub/... sending PUBLISH (d0, q0, r0, m1, '/device-1/messages', ... (13 bytes))
Client mosqpub/... sending DISCONNECT
```

   The message on the IMB terminal should confirm that it received the message over the secure port:

```
Message received on /device-1/messages
* issuer: device-1
```

4.2. Authenticating MQTT Clients

This section describes how to configure mutual authentication, so that an MQTT client is required to prove its identity to the IMB through an SSL certificate before it can publish or receive messages.

Procedure 4.3. To Configure Mutual Authentication

Before you start, complete the steps in Procedure 4.1, "To Set up MQTTS for an MQTT Server Connection".

1. Generate a keystore for the MQTT client, device-1:

   a. Generate a keystore and keypair:

```
$ keytool \
-genkeypair
 \
-alias imb
 \
-keyalg rsa
 \
-keystore /path/to/client_keystore.jks
 \
-storepass password
 \
-keypass password
 \
-dname "CN=device-1, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown"
```

Note

Because KeyStore converts all characters in its key aliases to lower case, use only lowercase in alias definitions of a KeyStore.

A file `/path/to/client_keystore.jks` is created.

b. Export the keystore to PKCS12 format:

```
$ keytool \
-importkeystore
 \
-srckeystore client_keystore.jks
 \
-destkeystore client_keystore.p12
 \
-deststoretype PKCS12

Enter destination keystore password: password
Re-enter new password:  password
Enter source keystore password:  password
Entry for alias imb successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled
```

The files `client_keystore.jks` and `client_keystore.p12` are created.

c. Convert the PKCS12 file containing the private key and certificates to PEM, and output the client certificate:

```
$ openssl pkcs12 -in client_keystore.p12 -nokeys -out client_cert.pem
Enter Import Password: password
MAC verified OK
```

The file `client_cert.pem` is created.

d. Convert the PKCS12 file containing the private key and certificates to PEM, and output the client private key:

```
$ openssl pkcs12 -in client_keystore.p12 -nodes -nocerts -out client_key.pem
Enter Import Password: password
MAC verified OK
```

The file `client_key.pem` is created.

2. Import the client certificate into the server's trust store:

```
keytool -importcert -file client_cert.pem -keystore /path/to/server_truststore.jks -alias "device-1"
Enter keystore password: password
Re-enter new password:  password
Owner: CN=device-1, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown
Issuer: CN=device-1, OU=Unknown, O=Unknown, L=Unknown, ST=Unknown, C=Unknown

...

Trust this certificate? [no]:yes
Certificate was added to keystore
```

The file `server_truststore.jks` is created.

3. Update the configuration file `imb_mqtt_server-default.json` to require MQTT clients to authenticate to the IMB, and to set the path and password for the truststore. Use the following file as an example:

Example 4.2. Example Configuration Requiring MQTT Clients to Authenticate

```
{
  "config" : {
    "mqtt" : {
      "host" : "0.0.0.0",
      "port" : 8883,
      "ssl" : true,
      "keyStoreOptions" : {
        "path" : "/path/to/server_keystore.jks",
        "password" : "password"
      },
      "clientAuthRequired" : true,
      "trustStoreOptions" : {
        "path" : "/path/to/server_truststore.jks",
        "password" : "password"
      }
    },
    "security" : {
      "openam-uri" : "http://openam.example.com:8088/openam",
      "policy-set" : "things",
      "operator" : {
        "username" : "imb_oidc_client",
        "password" : "password",
        "realm" : "/"
      }
    },
    "authentication-check-interval" : 15000
  }
```

```
}
```

Compared to Example 4.1, "Example Configuration Requiring MQTTS for the MQTT Server Connection", this file includes the client authentication boolean, and the path and password for the truststore.

The message on the IMB terminal should confirm that the configuration is updated:

```
INFO    | com.forgerock.imb.mqtt.server.MqttServerVerticle | MQTT Server [0.0.0.0:8883] stopped
INFO    | com.forgerock.imb.mqtt.server.MqttServerVerticle | MQTT Server is listening on 0.0.0.0:8883
```

Procedure 4.4. To Test the Setup

1. In the `device-1` terminal, run the following command to publish a message without providing a client certificate to authenticate the MQTT client:

```
$ mosquitto_pub -h localhost -p 8883 --cafile /path/to/server_cert.pem --insecure -u "<id_token-device
-1>" -P "<access_token-device-1>" -t "/device-1/messages" -m "1266193804 32" -d
Client mosqpub/... sending CONNECT
Error: The connection was lost.
```

   The message on the IMB terminal should confirm that the client failed to connect over SSL:

```
Client from origin /... failed to connect over ssl:...
```

2. Now run the following command that provides a client certificate to authenticate the MQTT client:

```
$ mosquitto_pub -h localhost -p 8883 --cafile /path/to/server_cert.pem --insecure --cert /path/to/
client_server_cert.pem --key /path/to/client_key.pem -u "<id_token-device-1>" -P "<access_token-device
-1>" -t "/device-1/messages" -m "1266193804 32" -d

Client mosqpub/... sending CONNECT
Client mosqpub/... received CONNACK
Client mosqpub/... sending PUBLISH (d0, q0, r0, m1, '/device-1/messages', ... (13 bytes))
Client mosqpub/... sending DISCONNECT
```

   The message on the IMB terminal should confirm that it received the message:

```
Message received on /device-1/messages
* issuer: device-1
```

Appendix A. Setting Up AM Manually for the Examples

This section describes how to set up authentication, resource types, and policies through the AM console. Alternatively, you can set up AM through Amster as described in Section 3.1.1, "Setting Up AM With Amster".

For all of the procedures in this section, log in to the AM console as administrator.

A.1. Setting Up the Example Subjects

Set up the example subjects as described in Procedure 3.2, "To Set Up the Example Subjects".

A.2. Configuring an OpenID Connect Provider

In this section you register OAuth 2.0 clients to request and introspect tokens:

1. In the top-level realm, select Applications > OAuth 2.0.
2. Add an OAuth 2.0 client with the scope to make an OpenID Connect request to AM, so that MQTT clients can use it to get an id_token:
   - Name: `oidc_client`
   - Password: `password`
   - Scope(s): `openid`

- ID Token Signing Algorithm: `HS256`, `HS384`, or `HS512`.

  Because the algorithm must be HMAC, the value of `RS256` is not okay.

3. Add an OAuth 2.0 client with the scope to introspect tokens, so that the IMB can use it when its interactions with AM require authentication:

   - Name: `imb_oidc_client`
   - Password: `password`
   - Scope(s): `am-introspect-all-tokens`

A.3. Configuring Resource Types

In this section you configure resource types for MQTT servers and MQTT topics, and their default permissions. The resource types are used in Section A.4, "Configuring a Policy Set and Policies":

1. In the top-level realm, select Authorization > Resource Types.
2. Add the following resource type for MQTT server, to define the hostname and port number to which MQTT clients can connect:

   - Name: `MQTT Server`
   - Patterns: `mqtt+server://*:*`
   - Actions: `CONNECT`, `Deny`.

3. Add the following resource type to define the message pattern to which MQTT clients and MQTT servers can publish or subscribe:

   - Name: `MQTT Topic`
   - Patterns: `mqtt+topic://*`
   - Actions: `RECEIVE`, `Deny` and `PUBLISH`, `Deny`.

A.4. Configuring a Policy Set and Policies

In this section you configure a policy set and policies that authorize `device-1` and `cloud-app` to publish and receive messages on topics:

1. In the top-level realm, select Authorization > Policy Sets.
2. Add a policy set with the following values, and then select Create:

- Id: `things`
- Name: `things`
- Resource Types: `MQTT Server` `MQTT Topic`

3. Add the following policy to authorize `device-1` to publish messages on the topic `/device-1/messages`:
   - Name: `device-1_messages_PUBLISH`
   - Resource Type: `MQTT Topic`
   - Resources: `mqtt+topic:///device-1/messages`
   - Actions: `PUBLISH`, `Allow`
   - Subject:
     - Add a constraint to make sure that the id_token is issued by the expected AM instance:
       - Type: `OpenID Connect/JWT Claim`
       - Claim Name: `iss`
       - Claim Value: `http://openam.example.com:8088/openam/oauth2`
     - Add a constraint to make sure that the id_token is received by the expected client:
       - Type: `OpenID Connect/JWT Claim`
       - Claim Name: `aud`
       - Claim Value: `oidc_client`
     - Add a constraint to make sure that `device-1` is the subject of the id_token:
       - Type: `OpenID Connect/JWT Claim`
       - Claim Name: `sub`
       - Claim Value: `device-1`
4. Add the following policy to authorize `cloud-app` to receive messages on the topic `/device-1/messages`:
   - Name: `device-1_messages_RECEIVE`
   - Resource Type: `MQTT Topic`
   - Resources: `mqtt+topic:///device-1/messages`
   - Actions: `RECEIVE`, `Allow`

- Subject:
  - Add a constraint to make sure that the id_token is issued by the expected AM instance:
    - Type: `OpenID Connect/JWT Claim`
    - Claim Name: `iss`
    - Claim Value: `http://openam.example.com:8088/openam/oauth2`
  - Add a constraint to make sure that the id_token is received by the expected client:
    - Type: `OpenID Connect/JWT Claim`
    - Claim Name: `aud`
    - Claim Value: `oidc_client`
  - Add a constraint to make sure that `cloud-app` is the subject of the id_token:
    - Type: `OpenID Connect/JWT Claim`
    - Claim Name: `sub`
    - Claim Value: `cloud-app`

5. Add the following policy to authorize `cloud-app` to publish messages on the topic `/device-1/actions`:
   - Name: `device-1_actions_PUBLISH`
   - Resource Type: `MQTT Topic`
   - Resources: `mqtt+topic:///device-1/actions`
   - Actions: `PUBLISH`, `Allow`
   - Subject:
     - Add a constraint to make sure that the id_token is issued by the expected AM instance:
       - Type: `OpenID Connect/JWT Claim`
       - Claim Name: `iss`
       - Claim Value: `http://openam.example.com:8088/openam/oauth2`
     - Add a constraint to make sure that the id_token is received by the expected client:
       - Type: `OpenID Connect/JWT Claim`

- Claim Name: `aud`
- Claim Value: `oidc_client`

- Add a constraint to make sure that `cloud-app` is the subject of the id_token:
  - Type: `OpenID Connect/JWT Claim`
  - Claim Name: `sub`
  - Claim Value: `cloud-app`

6. Add the following policy to authorize `device-1` to receive messages on the topic `/device-1/actions`:
   - Name: `device-1_actions_RECEIVE`
   - Resource Type: `MQTT Topic`
   - Resources: `mqtt+topic:///device-1/actions`
   - Actions: `RECEIVE`, `Allow`
   - Subject:
     - Add a constraint to make sure that the id_token is issued by the expected AM instance:
       - Type: `OpenID Connect/JWT Claim`
       - Claim Name: `iss`
       - Claim Value: `http://openam.example.com:8088/openam/oauth2`
     - Add a constraint to make sure that the id_token is received by the expected client:
       - Type: `OpenID Connect/JWT Claim`
       - Claim Name: `aud`
       - Claim Value: `oidc_client`
     - Add a constraint to make sure that `device-1` is the subject of the id_token:
       - Type: `OpenID Connect/JWT Claim`
       - Claim Name: `sub`
       - Claim Value: `device-1`
7. Add the following policy to authorize `device-1` and `cloud-app`, authenticated with valid id_tokens and access_tokens, to connect to the IMB on port `1883` or `8883`:

- Name: `default-mqtt-server`
- Resource Type: `MQTT Server`
- Resources: `mqtt+server://*:1883` and `mqtt+server://*:8883`
- Actions: `CONNECT`, `Allow`
- Create the following nested subject condition, with help from the AM *Authorization Guide* :

```
{
  "type": "AND",
  "subjects": [
    {
      "type": "AND",
      "subjects": [
        {
          "type": "JwtClaim",
          "claimName": "iss",
          "claimValue": "http://openam.example.com:8088/openam/oauth2"
        },
        {
          "type": "JwtClaim",
          "claimName": "aud",
          "claimValue": "oidc_client"
        }
      ]
    },
    {
      "type": "OR",
      "subjects": [
        {
          "type": "JwtClaim",
          "claimName": "sub",
          "claimValue": "device-1"
        },
        {
          "type": "JwtClaim",
          "claimName": "sub",
          "claimValue": "cloud-app"
        }
      ]
    }
  ]
}
```

Glossary

broker	See Identity Message Broker.
connect	The act of opening a connection between an MQTT client and the IMB. Uses the **MQTT CONNECT** command. See also publish, receive.
constrained device	A device that cannot communicate directly to the IMB. Constrained devices communicate with the IMB through an edge gateway. See also smart device.
device	An item embedded with electronics, software, sensors, or actuators, and network connectivity that enables it to collect and exchange data. See also constrained device, smart device.
edge	The position of sensors, actuators, devices, or an edge gateway in a network of internet-connected things. See also edge client.
edge client	An MQTT client that communicates with the Identity Message Broker. An edge client can be a constrained device at the edge that connects to an edge gateway, an edge gateway itself, or a smart device at the edge. See also MQTT client.
edge gateway	Positioned between devices and the IMB to enable a constrained device to communicate with the IMB.
gateway	See edge gateway.

Identity Message Broker	A publish-subscribe broker service that secures and hardens the sending and receiving of messages between the edge and the cloud, in IoT systems.
MQTT bridge	A relay between two MQTT servers. See also MQTT client, MQTT server.
MQTT client	A client on the edge or in the cloud that communicates with the IMB over MQTT. See also edge client, MQTT server.
MQTT server	A server in the IMB or in the cloud that communicates with an MQTT client over MQTT. See also MQTT bridge, MQTT client.
publish	The act of sending a message to an MQTT client that has registered to receive messages for that topic. Uses the **MQTT PUBLISH** command. See also connect, receive.
receive	The act of sending a message to an MQTT client that has registered to receive messages for that topic. Uses the **MQTT RECEIVE** command. See also connect, publish.
smart device	An electronic device that can connect to other devices or networks through MQTT. See also constrained device.