Using a CCDT in JSON with SSL/TLS with an IBM MQ client using Python PyMQI

https://www.ibm.com/support/pages/node/6955621

Date last updated: 14-Feb-2023

Angel Rivera
IBM MQ Support
https://www.ibm.com/products/mq/support
Find all the support you need for IBM MQ

+++ Objective +++

The objective of this tutorial is to describe how to use a CCDT in JSON with SSL/TLS with an IBM MQ client program that uses Python PyMQI: simple-mqput.py

The approach taken here is conservative and gradual. The idea is to begin with a solid baseline and subsequent steps are discreet and narrow in scope. In case that there is an error, it needs to be fixed before taking the next step.
Why? Because if you try to implement everything at once and then you encounter an error, where do you begin the troubleshooting task? And the answer to this rhetorical question is: well, it is difficult to troubleshoot such scenario when there are many possible candidates!

One important aspect to consider is that the MQ Client application (C-based or Python PyMQI) does NOT need to be aware if SSL is being used or not used: the underlying MQ client shared-libraries are the ones that will handle the SSL aspects!
Of course, the configuration of the SSL certificates and SSL-enabled channels need to be done by the user and the MQ administrator, and this is the part that is the most confusing to do correctly.

+ Strategy

Only when you can run successfully the MQ sample amqsputc as described in the tutorial about amqsputc then you can proceed to test with your Python PyMQI client application.

First, we need to ensure that non-SSL server-connection channels can be used with the MQ simple PyMQI client application "simple-mqput.py", beginning with using only the simple environment variable MQSERVER and then a CCDT in JSON that does not use SSL.

Secondly, only after all errors encountered with the first step are fixed, then we test the SSL/TLS server-connection channel using an CCDT in JSON that uses SSL.

At this point, if simple-mqput.py can run successfully but if your more complex PyMQI client application encounters errors, then these errors will most likely be contained with that client application and not with the certificates, the SSL channel, and the CCDT.

+ Requirements:

This document is in reality "Part 2", which has as a requirement the tasks mentioned in the following tutorial (which is "Part 1").
You should be able to successfully run the MQ sample amqsputc, which is C-based, with the CCDT in JSON:
  https://www.ibm.com/support/pages/node/6955535
  Using a CCDT in JSON with SSL/TLS with the IBM MQ sample amqsputc

You must install Python and PyMQI and do basic verification tasks.
  https://www.ibm.com/support/pages/node/6856737
  How to install Python IBM MQ module PyMQI in Linux and Windows

+ Configuration:

The configuration for this tutorial is exactly as the one described in:
  https://www.ibm.com/support/pages/node/6955535
  Using a CCDT in JSON with SSL/TLS with the IBM MQ sample amqsputc

Please consult the above tutorial to see the contents of the CCDT JSON files.

+ PyMQI program: simple-mqput.py

The Python program that is available at the web page in the cover pager, below the title, is called "simple-mqput.py" and it is not complex, but it is complete enough to be a usesful test for confirming the workings of the PyMQI module to server as an MQ Client application to connect to a queue manager.

There are 3 important points about this program:

1) To allow for the MQ client shared libraries to do remote connections, it is CRITICAL to setup the following environment variable:
   export MQ_CONNECT_TYPE=CLIENT

2) You must use the methods highlighted below. This will tell the MQ client shared libraries to do the proper handling of MQSERVER and the CCDT JSON files.

   queue_manager = **pymqi.QueueManager(None)**
   queue_manager.**connect_with_options**(queue_manager_name)

3) For simple situations, there is no need to add complicated code in your program for handling SSL. The SSL attributes in the CCDT in JSON can be easily specified and modified. Then, let the MQ client shared libraries handle the underlying complexity!

++ Section 1: Host-2, Linux Client, using MQSERVER

You need to set the following environment variable in order to allow remote connections:

```
$ export MQ_CONNECT_TYPE=CLIENT
```

Let's set the MQSERVER environment variable (cannot use SSL with it):

```
$ export MQSERVER='NON.SSL.SVRCONN/TCP/volterra1.fyre.ibm.com(1419)'
```

Let's unset the other connectivity environment variables:

```
$ unset MQCLNTCF
$ unset MQCHLLIB
$ unset MQCHLTAB
```

Let's not pass any arguments and let's see the usage syntax:

```
$ python3 simple-mqput.py
usage: Put message into Queue in Queue Manager [-h] -q Q -m M [-c C]
                                               [--host HOST] [--port PORT]
                                               [--msg MSG] [--verbose]
Put message into Queue in Queue Manager: error: the following arguments are required: -q,
-m
```

Let's specify the queue and the queue manager.
If a message is not specified as argument, the program uses a default.

```
$ python3 simple-mqput.py -q Q1 -m QM93TLS
2023-02-14 13:48:53,763     INFO : Put message was successful: Hello from Python!
```

Let's try again, but this time with a message and with --verbose

```
$ python3 simple-mqput.py -q Q1 -m QM93TLS --msg "From simple-mqput.ph" --verbose
2023-02-14 13:51:19,772    DEBUG : Starting: simple-mqput
2023-02-14 13:51:19,773    DEBUG : Namespace(c='SYSTEM.DEF.SVRCONN', host='localhost',
m='QM93TLS', msg='From simple-mqput.ph', port='1414', q='Q1', verbose=True)
2023-02-14 13:51:19,773    DEBUG : Queue: Q1 QMgr: QM93TLS Channel:
SYSTEM.DEF.SVRCONN
2023-02-14 13:51:19,773    DEBUG : Host: localhost Port: 1414 Host_port: localhost(1414)
2023-02-14 13:51:19,773    DEBUG : Message text: From simple-mqput.ph
2023-02-14 13:51:19,773    DEBUG : mq_connect_type, MQ_CONNECT_TYPE : CLIENT
2023-02-14 13:51:19,773    DEBUG : mq_server, MQSERVER              :
NON.SSL.SVRCONN/TCP/volterra1.fyre.ibm.com(1419)
2023-02-14 13:51:19,774    DEBUG : mq_client_config, MQCLNTCF       : None
2023-02-14 13:51:19,774    DEBUG : mq_ccdt_path, MQCHLLIB           : None
2023-02-14 13:51:19,774    DEBUG : mq_ccdt_file, MQCHLTAB           : None
2023-02-14 13:51:19,798    DEBUG : Connection successful.
2023-02-14 13:51:19,798    DEBUG : Open queue was successful.
```

```
2023-02-14 13:51:19,800    DEBUG : Put was successful.
2023-02-14 13:51:19,801    DEBUG : Closing queue was successful.
2023-02-14 13:51:19,802    DEBUG : Disconnect was successful.
2023-02-14 13:51:19,802     INFO : Put message was successful: From simple-mqput.py
2023-02-14 13:51:19,803    DEBUG : Ending: simple-mqput
```

Let's open a terminal with the host that has the queue manager and let's get the messages. Notice that it shows the 2 messages that we just put!

```
mqm@volterra1.fyre.ibm.com: /home/mqm
$ amqsget Q1 QM93TLS
Sample AMQSGET0 start
message <Hello from Python!>
message <From simple-mqput.py>
```

++ Section 2: Host-2, Linux Client, using CCDT in JSON without SSL

Let's proceed to use a CCDT in JSON that invokes a channel that does not use SSL.

Location: /var/mqm/ssl
File name: ccdt-QM93TLS-volterra1-nonssl.json

The contents will be:
+ begin contents (ignore this line)

```
{
  "channel": [
    {
      "name": "NON.SSL.SVRCONN",
      "clientConnection": {
        "connection": [
          {
            "host": "volterra1.fyre.ibm.com",
            "port": 1419
          }
        ],
        "queueManager": "QM93TLS"
      },
      "type": "clientConnection"
    }
  ]
}
```

+ end contents (ignore this line)

Now you MUST unset the environment variables MQSERVER and MQCLNTCF.

```
$ unset MQSERVER
$ unset MQCLNTCF
```

Proceed to set the following 2 environment variables that provide the full path of the CCDT file:

```
$ export MQCHLLIB=/var/mqm/ssl
$ export MQCHLTAB=ccdt-QM93TLS-volterra1-nonssl.json
```

Notice that you need to have the environment variable that points to the key repository:

```
$ export MQSSLKEYR=/var/mqm/ssl/clientkey
```

OK, let's confirm that we have the proper environment variables:

```
$ set | grep MQ
MQCHLLIB=/var/mqm/ssl
MQCHLTAB=ccdt-QM93TLS-volterra1-nonssl.json
MQSSLKEYR=/var/mqm/ssl/clientkey
MQ_CONNECT_TYPE=CLIENT
```

Now, let's run the program again, with verbose mode and with a message.

```
$ python3 simple-mqput.py -q Q1 -m QM93TLS --msg "simple-mqput.py, CCDT JSON non
SSL" --verbose
2023-02-14 13:58:44,704   DEBUG : Starting: simple-mqput
2023-02-14 13:58:44,704   DEBUG : Namespace(c='SYSTEM.DEF.SVRCONN', host='localhost',
m='QM93TLS', msg='simple-mqput.py, CCDT JSON non SSL', port='1414', q='Q1', verbose=True)
2023-02-14 13:58:44,704   DEBUG : Queue: Q1 QMgr: QM93TLS Channel:
SYSTEM.DEF.SVRCONN
2023-02-14 13:58:44,705   DEBUG : Host: localhost Port: 1414 Host_port: localhost(1414)
2023-02-14 13:58:44,705   DEBUG : Message text: simple-mqput.py, CCDT JSON non SSL
2023-02-14 13:58:44,705   DEBUG : mq_connect_type, MQ_CONNECT_TYPE : CLIENT
2023-02-14 13:58:44,705   DEBUG : mq_server, MQSERVER           : None
2023-02-14 13:58:44,705   DEBUG : mq_client_config, MQCLNTCF     : None
2023-02-14 13:58:44,705   DEBUG : mq_ccdt_path, MQCHLLIB        : /var/mqm/ssl
2023-02-14 13:58:44,705   DEBUG : mq_ccdt_file, MQCHLTAB        : ccdt-QM93TLS-
volterra1-nonssl.json
2023-02-14 13:58:44,729   DEBUG : Connection successful.
2023-02-14 13:58:44,729   DEBUG : Open queue was successful.
2023-02-14 13:58:44,733   DEBUG : Put was successful.
2023-02-14 13:58:44,734   DEBUG : Closing queue was successful.
2023-02-14 13:58:44,735   DEBUG : Disconnect was successful.
2023-02-14 13:58:44,735    INFO : Put message was successful: simple-mqput.py, CCDT
JSON non SSL
2023-02-14 13:58:44,736   DEBUG : Ending: simple-mqput
```

Let's check the queue manager:

```
mqm@volterra1.fyre.ibm.com: /home/mqm
$ amqsget Q1 QM93TLS
Sample AMQSGET0 start
message <simple-mqput.py, CCDT JSON non SSL>
```

++ Section 3: Host-2, Linux Client, using CCDT in JSON with SSL

Finally, let's use a CCDT with JSON that uses SSL:

Location: /var/mqm/ssl
File name: ccdt-QM93TLS-volterra1-linux-ssl.json

The contents will be:
+ begin contents (ignore this line)

```
{
  "channel": [
    {
      "name": "SSL.SVRCONN.LNX",
      "clientConnection": {
        "connection": [
          {
            "host": "volterra1.fyre.ibm.com",
            "port": 1419
          }
        ],
        "queueManager": "QM93TLS"
      },
      "transmissionSecurity":
      {
        "cipherSpecification": "TLS_AES_128_GCM_SHA256",
        "certificateLabel": "ibmwebspheremqmqm",
      },
      "type": "clientConnection"
    }
  ]
}
```

+ end contents (ignore this line)

Proceed to set the following 1 environment variable that provides the file name.

```
$ export MQCHLTAB=ccdt-QM93TLS-volterra1-linux-ssl.json
```

```
python3 simple-mqput.py -q Q1 -m QM93TLS --msg "simple-mqput.py, CCDT JSON with SSL" -
-verbose
2023-02-14 14:01:48,715    DEBUG : Starting: simple-mqput
2023-02-14 14:01:48,716    DEBUG : Namespace(c='SYSTEM.DEF.SVRCONN', host='localhost',
m='QM93TLS', msg='simple-mqput.py, CCDT JSON with SSL', port='1414', q='Q1',
verbose=True)
2023-02-14 14:01:48,716    DEBUG : Queue: Q1 QMgr: QM93TLS Channel:
SYSTEM.DEF.SVRCONN
2023-02-14 14:01:48,716    DEBUG : Host: localhost Port: 1414 Host_port: localhost(1414)
2023-02-14 14:01:48,716    DEBUG : Message text: simple-mqput.py, CCDT JSON with SSL
2023-02-14 14:01:48,717    DEBUG : mq_connect_type, MQ_CONNECT_TYPE : CLIENT
2023-02-14 14:01:48,717    DEBUG : mq_server, MQSERVER          : None
2023-02-14 14:01:48,717    DEBUG : mq_client_config, MQCLNTCF      : None
```

```
2023-02-14 14:01:48,717    DEBUG : mq_ccdt_path, MQCHLLIB          : /var/mqm/ssl
2023-02-14 14:01:48,717    DEBUG : mq_ccdt_file, MQCHLTAB          : ccdt-QM93TLS-
volterra1-linux-ssl.json
2023-02-14 14:01:48,859    DEBUG : Connection successful.
2023-02-14 14:01:48,860    DEBUG : Open queue was successful.
2023-02-14 14:01:48,863    DEBUG : Put was successful.
2023-02-14 14:01:48,865    DEBUG : Closing queue was successful.
2023-02-14 14:01:48,868    DEBUG : Disconnect was successful.
2023-02-14 14:01:48,869     INFO : Put message was successful: simple-mqput.py, CCDT
JSON with SSL
2023-02-14 14:01:48,869    DEBUG : Ending: simple-mqput
```

Let's get the message from the queue manager:

```
mqm@volterra1.fyre.ibm.com: /home/mqm
$ amqsget Q1 QM93TLS
Sample AMQSGET0 start
message <simple-mqput.py, CCDT JSON with SSL>
```

+ Confirmation test:

You may think at this point... hum, how can I be sure that the CCDT with JSON with SSL is actually being used?

One easy way to test is to introduce a typo graphical error with the cipherspec in the file, which will cause a runtime error.

Instead of the valid original value of:
"certificateLabel": "ibmwebspheremqmqm",

Let's remove the last "m" in the label:
"certificateLabel": "ibmwebspheremqmq",

At runtime, the MQ client shared libraries will try to get the incorrect label from the key repository and will fail:

```
$ python3 simple-mqput.py -q Q1 -m QM93TLS --msg "simple-mqput.py, CCDT JSON with
SSL" --verbose
2023-02-14 14:05:48,046    DEBUG : Starting: simple-mqput
2023-02-14 14:05:48,046    DEBUG : Namespace(c='SYSTEM.DEF.SVRCONN', host='localhost',
m='QM93TLS', msg='simple-mqput.py, CCDT JSON with SSL', port='1414', q='Q1',
verbose=True)
2023-02-14 14:05:48,046    DEBUG : Queue: Q1 QMgr: QM93TLS Channel:
SYSTEM.DEF.SVRCONN
2023-02-14 14:05:48,046    DEBUG : Host: localhost Port: 1414 Host_port: localhost(1414)
2023-02-14 14:05:48,046    DEBUG : Message text: simple-mqput.py, CCDT JSON with SSL
2023-02-14 14:05:48,046    DEBUG : mq_connect_type, MQ_CONNECT_TYPE : CLIENT
2023-02-14 14:05:48,046    DEBUG : mq_server, MQSERVER              : None
2023-02-14 14:05:48,046    DEBUG : mq_client_config, MQCLNTCF        : None
2023-02-14 14:05:48,046    DEBUG : mq_ccdt_path, MQCHLLIB           : /var/mqm/ssl
2023-02-14 14:05:48,047    DEBUG : mq_ccdt_file, MQCHLTAB           : ccdt-QM93TLS-
volterra1-linux-ssl.json
2023-02-14 14:05:48,187    ERROR : Cannot connect to queue manager: QM93TLS - error:
MQI Error. Comp: 2, Reason 2393: FAILED: MQRC_SSL_INITIALIZATION_ERROR
Traceback (most recent call last):
  File "simple-mqput.py", line 84, in <module>
    queue_manager.connect_with_options(queue_manager_name)
  File "/usr/local/lib64/python3.6/site-packages/pymqi/__init__.py", line 1747, in
connect_with_options
    raise MQMIError(rv[1], rv[2])
pymqi.MQMIError: MQI Error. Comp: 2, Reason 2393: FAILED:
MQRC_SSL_INITIALIZATION_ERROR
MQI Error. Comp: 2, Reason 2393: FAILED: MQRC_SSL_INITIALIZATION_ERROR
```

Then we can take a look at the general error log at:
/var/mqm/errors//AMQERR01.LOG
… and we see:

```
02/14/2023 02:05:48 PM - Process(276930.1) User(mqm) Program(python3)
                    Host(suvereto1.fyre.ibm.com) Installation(Installation1)
                    VRMF(9.2.0.6)
                    Time(2023-02-14T22:05:48.185Z)
                    RemoteHost(9.46.80.212)
                    CommentInsert1(SSL.SVRCONN.LNX)
                    CommentInsert2(volterra1 (9.46.80.212)(1419))
AMQ9642E: No SSL or TLS certificate for channel 'SSL.SVRCONN.LNX'.
EXPLANATION:
The channel 'SSL.SVRCONN.LNX' did not supply a certificate to use during SSL or
TLS handshaking, but a certificate is required by the remote queue manager.
The remote host is 'volterra1 (9.46.80.212)(1419)'.
The channel did not start.
ACTION:
Ensure that the key repository of the local queue manager or MQ client contains
a certificate which is associated with the queue manager or client. If you have
configured a certificate label, check that the certificate exists.
Alternatively, if appropriate, change the remote channel definition so that its
SSLCAUTH attribute is set to OPTIONAL and it has no SSLPEER value set.
```

+++ end +++