DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift

Table of Contents

Credit to Joel Gomez for assisting with these instructions!!

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift

Prerequisites

You must have these tools installed on your workstation

- MAC:
  - READ ME MacOS
  - IBM MQ MacOS Toolkit for developers
- WINDOWS / LINUX:
  - MQ Client for Linux & Windows
  - MQ Clients are also listed on FixCentral

- OpenShift OC command line tool
- OpenSSL tool for your operating system

Assumptions

- You have a Running OpenShift Cluster (4.8 or higher)
- You have deployed Cloud Pak for Integration (V2021.2 or higher)
  - In our example below the namespace where the Cloud Pak was installed is CP4I (That is the default for Daffy Install). ***NOTE: You may need to modify the namespace for your installation***

Testing Remote Connectivity

To confirm the queue manager is configured for one-way TLS communication, We will be using the **`amqsputc`** and **`amqsgetc`** sample applications to put and get messages.

Note: *These testing tools would have been downloaded and installed as part of the prerequisites.*

Find the HOST & PORT of the OpenShift MQ Route

Note*: In this example we created this queue manager inside the CP4I namespace, so we will add the -n cp4i to ensure we are pointed at the right OpenShift namespace.*

```
oc get routes secureqm-ibm-mq-qm -n cp4i
```

Copy the value listed for HOST/PORT
Note: *You may not see a port number listed. That's ok if it's using the default 443 port.*

(Here is an example of the value we are looking for)
secureqm-ibm-mq-qm-cp4i.apps.gamma02.kvm01.ibm-cp4-dojo.net

We will need this for the next step, regardless of the platform your client is running on.

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift

Testing on the Bastion

All the commands below are run Bastion machine where you previously ran the deploy-demo.sh script. It is assumed that your running these command from the /data/daffy/cp4i/demos/secureqm/ directory

/data/daffy/cp4i/demos/secureqm/client-side-artifacts

Step 1.) Create cert database and add your cert to the database

****
********** STOP **** STOP **********
****

STEP 1 COMMANDS WERE RAN AS PART OF THE deploy-demo.sh SCRIPT. PLEASE PROCEED TO STEP 2.

Note: *The MQ Client will install a tool (runmqakm) that will be used to create a cert database. This command line tool will be in the <mq client install dir>/bin folder.*

Create the cert database on your local workstation using this command.

```
./runmqakm -keydb -create -db clientkey.kdb -pw password -type cms -stash
```

This will create a NEW file called clientkey.kdb (placed in the directory where the command was executed). This is the client key database, where we will next store the tls.crt (certificate).

Add the tls.crt file to the key store database.

```
./runmqakm -cert -add -db clientkey.kdb -label mqservercert -file tls.crt -format ascii -stashed
```

Step 2.) Modify the CCDT.JSON

There is a sample CCDT.JSON file in the /data/daffy/cp4i/demos/secureqm/client-side-artifacts directory. We will need to modify the host name value in this file.

IMPORTANT: *You will need to update the value for the HOST in this JSON. Below is what the sample CCDT.JSON file looks like.*

```
{
    "channel":
    [
        {
            "name": "SECUREQMCHL",
            "clientConnection":
            {
                "connection":
                [
                    {
                        "host": "<hostname from previous step>",
                        "port": 443
                    }
                ],
                "queueManager": "SECUREQM"
            },
            "transmissionSecurity":
            {
              "cipherSpecification": "ECDHE_RSA_AES_128_CBC_SHA256"
            },
            "type": "clientConnection"
        }
    ]
}
```

Step 3.) Linux Only - Export the environment variables

These environment variables are necessary for the MQ Client commands to work.

IMPORTANT: *You will need to fully qualify the location of these two files. Notice that we do* ***NOT*** *put the file extension on the clientkey.*

```
export MQCCDTURL='/data/daffy/cp4i/demos/secureqm/client-side-
artifacts/CCDT.JSON'
```

```
export MQSSLKEYR='/data/daffy/cp4i/demos/secureqm/clientkey'
```

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift

Step 4.) Put message on the queue

The amqsputc command is located in the following directory.

/data/daffy/cp4i/demos/secureqm/samp/bin

```
amqsputc EXAMPLE.QUEUE SECUREQM
```

If connection to the queue manager is successful, the following response is output:

```
target queue is EXAMPLE.QUEUE
```

Put several messages to the queue, by entering some text then pressing **Enter** each time.
To finish, press **Enter** twice.

Step 5.) Get message from the queue

The amqsgetc command is located in the following directory.

/data/daffy/cp4i/demos/secureqm/samp/bin

```
amqsgetc EXAMPLE.QUEUE SECUREQM
```

The messages you added in the previous step have been consumed and are displayed in the terminal.

After a few seconds, the command exits.

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift

Testing on a remote client

All the commands below are run on your (remote) client. It is assumed that your running these command from the directory where you downloaded the client side artifacts. In these examples this is (mydir)

Step 1.) Create cert database and add your cert to the database

Note: *The MQ Client will install a tool (runmqakm) that will be used to create a cert database. This command line tool will be in the <mq client install dir>/bin folder.*

Create the cert database on your local workstation using this command.

```
./runmqakm -keydb -create -db clientkey.kdb -pw password -type cms -stash
```

This will create a NEW file called clientkey.kdb (placed in the directory where the command was executed). This is the client key database, where we will next store the tls.crt (certificate).

Add the tls.crt file to the key store database.

```
./runmqakm -cert -add -db clientkey.kdb -label mqservercert -file tls.crt -
format ascii -stashed
```

Step 2.) Modify the CCDT.JSON

There is a sample CCDT.JSON file in the client-side-artifacts zip file you downloaded with this documentation. We will need to modify the host name value in this file.

IMPORTANT*: You will need to update the value for the HOST in this JSON. Below is what the sample CCDT.JSON file looks like.*

```
{
    "channel":
    [
        {
            "name": "SECUREQMCHL",
            "clientConnection":
            {
                "connection":
                [
                    {
                        "host": "<hostname from previous step>",
                        "port": 443
                    }
                ],
                "queueManager": "SECUREQM"
            },
            "transmissionSecurity":
            {
              "cipherSpecification": "ECDHE_RSA_AES_128_CBC_SHA256"
            },
            "type": "clientConnection"
        }
    ]
}
```

Step 3.) Linux Only - Export the environment variables

These environment variables are necessary for the MQ Client commands to work.

IMPORTANT*: You will need to fully qualify the location of these two files. Notice that we do* ***NOT*** *put the file extension on the clientkey.*

```
export MQCCDTURL='<full path to file>/CCDT.JSON'
```

```
export MQSSLKEYR='<full path to file>/clientkey'
```

Step 4.) Put message on the queue

```
amqsputc EXAMPLE.QUEUE SECUREQM
```

If connection to the queue manager is successful, the following response is output:
`target queue is EXAMPLE.QUEUE`

Put several messages to the queue, by entering some text then pressing **Enter** each time.
To finish, press **Enter** twice.

Step 5.) Get message from the queue

```
amqsgetc EXAMPLE.QUEUE SECUREQM
```

The messages you added in the previous step have been consumed and are displayed in the terminal.

After a few seconds, the command exits.

Other Related Resources

- Tutorial: https://developer.ibm.com/tutorials/mq-secure-msgs-tls/
- Joel Gomez: Tutorial https://github.ibm.com/joel-gomez/tinkering-cp4i/blob/master/MQ/OCP/MQccConfigOCPv43ExtAccess.md
- Joel Gomez: tinkering-cp4i https://github.ibm.com/joel-gomez/tinkering-cp4i
- Examples from the IBM MQ 9.2 Documentation: https://www.ibm.com/docs/en/ibm-mq/9.2?topic=manager-example-configuring-tls

FOR REFERNCE ONLY – This is what the DAFFY DEMO (deploy-demo.sh) script automates.

NOTE: *This section is just for your reference. You do not need to execute these steps if you run the script to deploy the demo.*

From your local workstation (or where you plan to run the sample testing application) run the following commands to generate the TLS keys.

Step 1.) Create a NEW directory on your client machine.

```
mkdir -p mydir
```

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift

```
Cd mydir
```

Step 2.) Create a self-signed private key and public certificate.

```
openssl req -newkey rsa:2048 -nodes -keyout tls.key -subj "/CN=localhost" -
x509 -days 3650 -out tls.crt
```

Step 3.) Create a client key database.

```
runmqakm -keydb -create -db clientkey.kdb -pw password -type cms -stash
```

Step 4.) Add the generated public key to the client key database.

```
runmqakm -cert -add -db clientkey.kdb -label mqservercert -file tls.crt -
format ascii -stashed
```

Step 5.) Login to the cluster and source the CP4I project

(Example Login Command – Use your login command)

```
oc login https://api.gamma02.kvm01.ibm-cp4-dojo.net:6443 -u ocpadmin -p
SUINGUkmVsxSzqiTRh --insecure-skip-tls-verify
```

```
oc project cp4i
```

Step 6.) Create a NEW Secret with the newly create TLS certificates and Key

You will need to run this next command from the directory (mydir in our example) where the tls.cert is located.

(Note: Create this secret inside the namespace of your Cloud Pak MQ deployment. In our example we are creating this inside the cp4i namespace.)

```
oc create secret tls example-tls-secret --key="tls.key" --cert="tls.crt"
```

Step 7.) Create a NEW CONFIGMAP with the newly created secret

Create a config map with some MQSC commands that will be executed when we create the queue manager. These MQSC commands will tell the queue manager to create the following resources once the queue manager is up and running. You will see that this config map is referenced in the YAML for the creation of the QM.

You can either use the OpenShift Admin console to deploy this YAML or you can use the OC apply command. Both will work just fine, but make sure you place this in the right namespace.

Note: In our example we will create this secret inside the CP4I namespace.

```
apiVersion: v1
```

```
kind: ConfigMap
metadata:
  name: example-tls-configmap
data:
  tls.mqsc: |
    DEFINE QLOCAL('EXAMPLE.QUEUE') REPLACE
    DEFINE CHANNEL(SECUREQMCHL) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCAUTH(OPTIONAL) SSLCIPH('ANY_TLS12_OR_HIGHER')
    SET CHLAUTH(SECUREQMCHL) TYPE(BLOCKUSER) USERLIST('nobody') ACTION(ADD)
```

Step 8.) Create a NEW ROUTE in OpenShift

You can either use the OpenShift Admin console to deploy this YAML or you can use the OC apply command. Both will work just fine, but make sure you place this in the right namespace.

Note: In our example the channel name we are using does not contain any characters that need to be converted for SNI. Just note, that you will need to encode the channel name to SNI mapping if you used any characters such as a .(dot) or – (dash). Here is some documentation on how to do that if needed. <ADD LINK>

Note: In our example we will create this secret inside the CP4I namespace.

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: example-tls-route
spec:
  host: secureqmchl.chl.mq.ibm.com
  to:
    kind: Service
    name: secureqm-ibm-mq
  port:
    targetPort: 1414
  tls:
    termination: passthrough
```

Step 9.) Create a NEW Queue Manager in OpenShift

You can either use the OpenShift Admin console to deploy this YAML or you can use the OC apply command. Both will work just fine, but make sure you place this in the right namespace.

Note: In our example we will create this secret inside the CP4I namespace.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: secureqm
spec:
  license:
    accept: true
    license: L-RJON-BZFQU2
    use: NonProduction
  queueManager:
    name: SECUREQM
    mqsc:
    - configMap:
        name: example-tls-configmap
        items:
        - tls.mqsc
    storage:
      queueManager:
        type: ephemeral
  template:
    pod:
      containers:
        - env:
            - name: MQSNOAUT
              value: 'yes'
          name: qmgr
  version: 9.2.3.0-r1
  web:
    enabled: true
  pki:
    keys:
      - name: example
        secret:
          secretName: example-tls-secret
          items:
          - tls.key
          - tls.crt
```

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client
Show how to put & get messages using TLS security – from an application outside of OpenShift

Step 10.) Verify the Queue Manager is up and running.

From your workstation you can now run the oc command to check the status of your queue manager.

Note: In our example we created this queue manager inside the CP4I namespace, so we will add the -n cp4i to ensure we are pointed at the right OpenShift namespace.

```
oc get qmgr secureqm -n cp4i
```

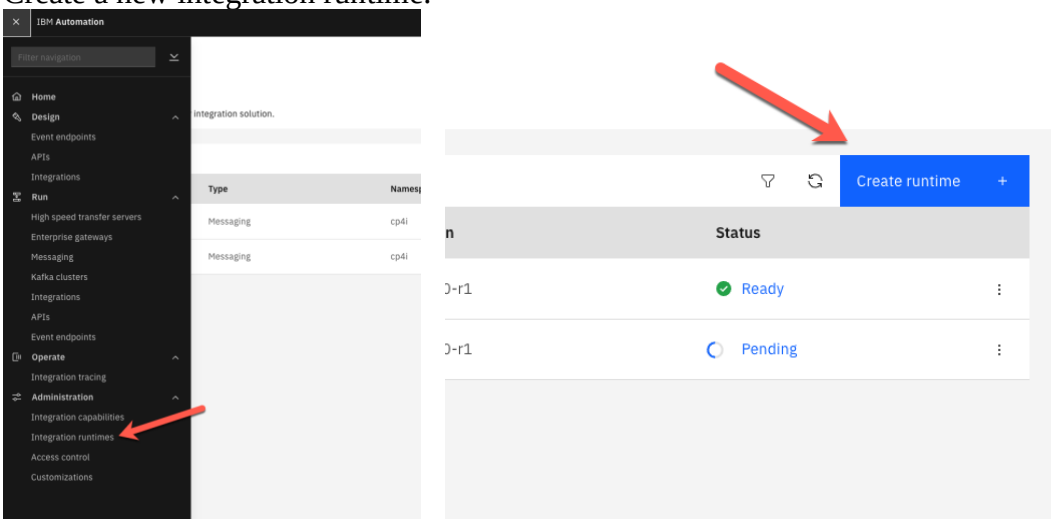
Result of that command should look like this: secureqm Running

FOR REFERNCE ONLY: Creating a Queue Manager with these configurations using the Platform Navigator

NOTE: *This is for your reference only. These are the steps you would take to deploy the Secure Queue Manager using the Platform Navigator.*

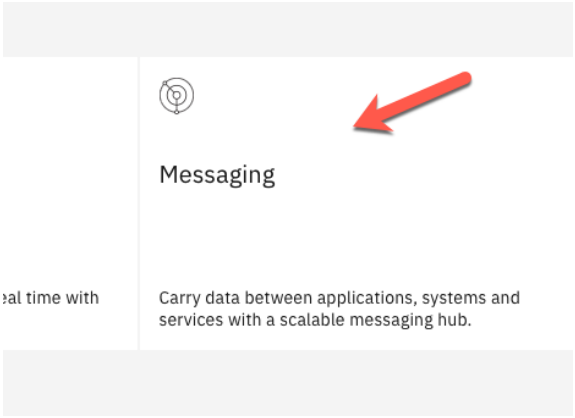
Assumptions: We are assuming you will be using the same PKI TLS crt and key files.

Create a new integration runtime.

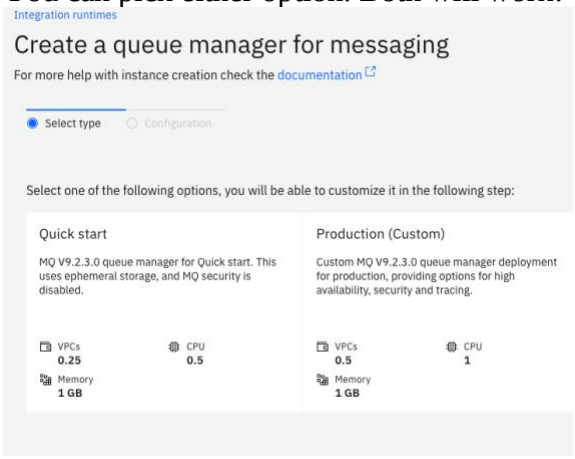


DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift



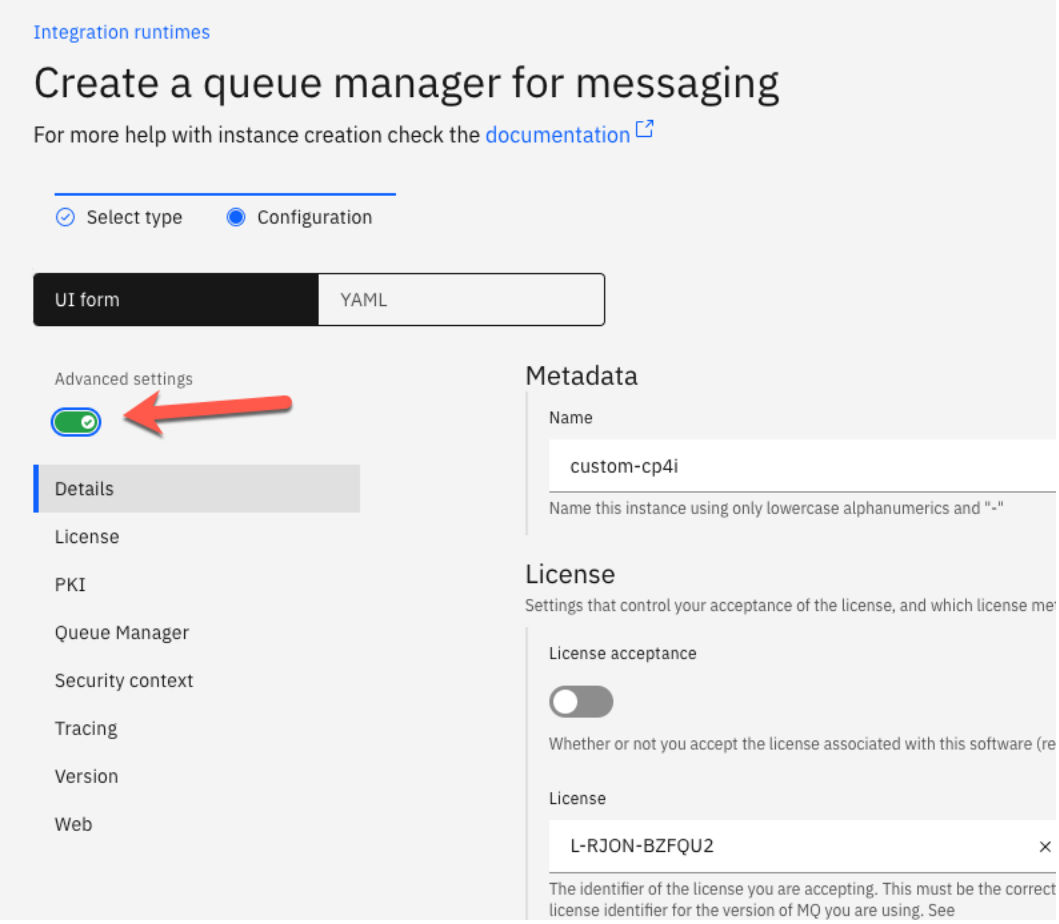
You can pick either option. Both will work!



It’s important that you pick the advanced option on this screen. This is where you will find the necessary fields to set for PKI.

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift



You will need to create a ROUTE to connect to the queue manager, just as we did in the first example, but this time we will use the OCP admin console.

Note: If you have created the ROUTE for the first example, then you MUST delete it before you can create a NEW route using the same channel name.

DAFFY DEMO – Connecting to a Secure CP4I Queue Manager using MQ Client

Show how to put & get messages using TLS security – from an application outside of OpenShift