How to connect an MQ Client to an MQ Queue Manager deployed in an OpenShift cluster using the default route?

https://www.ibm.com/support/pages/node/7145555

Date last updated: 4-April-2024

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

+++ Objective +++

You have an IBM MQ Queue Manager deployed in OpenShift and you would like to connect an MQ Client that exists on a host that is external to the OpenShift cluster.

You will need an OpenShift Route to connect an application to a MQ queue manager from outside a Red Hat OpenShift cluster. Internally, the OpenShift route uses SNI for routing requests to the appropriate queue manager pod. Thus, you must enable TLS on your MQ queue manager and client application, because SNI is only available in the TLS protocol when a TLS 1.2 or higher protocol is used.

When a queue manager is created in an OpenShift cluster using the IBM MQ Operator, there are two routes created by default. One is to connect to the MQ queue manager, and another route is used for the IBM MQ Console.

Optionally you can create additional routes to connect to the queue manager. For more information about how to create a route in OpenShift please review the following on this IBM MQ documentation page:

https://www.ibm.com/docs/en/ibm-mq/9.3?topic=cqmumo-configuring-route-connect-queue-manager-from-outside-red-hat-openshift-cluster
Title: Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster

On this tutorial we will use the default route and the MQ samples “amqsputc” and “amqpsget”.

++ Summary ++

Summary of steps for the queue manager (Qmgr)

- Create a Queue Qmgr using the MQ Operator.

- Create the Qmgr certificates outside the OpenShift cluster.
- Create the OpenShift Secrets that will contain the Qmgr's certificates.
- Create the Qmgr's objects: SVRCONN channel and queues.
- Exchange the Qmgr's trusted certificate with the MQ client.
- Import the MQ client trusted certificate in a Secret.

Summary of steps for the MQ client:

- Create the key repository.
- Create the MQ client certificate.
- Import the Qmgr's trusted certificates.
- Create CCDT file to be able to connect to the Qmgr.

++ Detailed configuration steps ++

Queue Manager Certificates

On this tutorial, we will start by creating the Qmgr's certificate which will be signed by an internal Certificate Authority (CA).
We will use the "runmqakm" command that is shipped with MQ.

https://www.ibm.com/docs/en/ibm-mq/9.3?topic=windows-runmqckm-runmqakm-options-aix-linux
runmqckm and runmqakm options on AIX, Linux, and Windows

1. Create the key repository:
```
# runmqakm -keydb -create -db qmgr.p12 -pw passw0rd -type pkcs12
```

2. Create a certificate request (CSR):
```
# runmqakm -certreq -create -db qmgr.p12 -pw passw0rd -label cloud-qmgr -dn "CN=CLOUDQMGR,O=IBM,C=US,OU=MQ Support,ST=NorthCarolina" -size 2048 -sig_alg SHA256WithRSA -file certreq-cloudqmgr.arm
```

3. Send the CSR to a Certificate Authority (CA) to be signed. The CA will send back the certificate signed and the trusted certificates used to sign the certificate, which is usually an intermediate(s) and a root certificate.

4. Received the signed certificate into the Qmgr's key repository:
```
# runmqakm -cert -receive -file signed-cloudqmgr.arm -db qmgr.p12 -pw passw0rd
```

5. Add the CA trusted certificate, for this example I just have a root certificate:
```
# runmqakm -cert -add -db qmgr.p12 -pw passw0rd -label root-ca-cert -file /home/mqm/ca-certs/rootca.arm
```

** The same command will be used to add the MQ client trusted certificate(s).

Now we need to convert the Qmgr's certificates to pem format. Why? Because we will create Kubernetes Secrets in the OpenShift cluster that will contain these certificates. Then the MQ Operator will load the certificates into the queue manager pod using the associated Secrets.

The following openssl command can be used:

```
# openssl pkcs12 -info -in qmgr.p12 -nodes
```

Example of the pem format, note that it includes the "BEGIN/END CERTIFICATE":

-----BEGIN CERTIFICATE-----
MIIDHjCCAgagAwIBAgIICSeSU5nAB00wDQYJKoZIhvcNAQELBQAwLTEMMAoGA1UEBhMDVVNBMQ
wwCgYDVQQKEwNJQk0xDzANBgNVBAMTBk1BUjFGUjAeFw0yMz
QiqND9RoJw7zki+lVzAfBgNVHSMEGDAWgBQIMJcZYJ8VQiqND9RoJw7zki+lVzAN
BgkqhkiG9w0BAQsFAAOCAQEAxuiTo5uIzXGCO+GHwqllpyNoc9jWUDpBQRnmbrEf
AmE/epeQ4Ec4acP3d0fea+gj7cTKvgIdi0fOvlOLlYEL8bYtAypPkIsWpFioBXF6
N72iFzbsYOYbxIOVPAgViYTuPrC2cY69qWqutbj1zVuGLX9CtgFDaS55RW03e9Uz
DSnu7woE8/lmEmc1y4gBIO3CM4zBh8qG2RbaOz9i4IrI1ZHkkEfEiqirGnreWUCP
HhHydtLowEHlHSz50lVryS4HUHJglvgF3txkRx8aZqsHb+UeUC1Fk7PXyu9R3vNM
36MNobMMDvdNi+tXRXdz8gjsZLHY/SJqPAO0/91DdUfTFQ==
-----END CERTIFICATE-----

MQ Client Certificates

The following commands are used to create a CA signed certificate for the MQ Client.

1. Create key repository:
   ```
   # runmqakm -keydb -create -type cms -db client.kdb -pw passw0rd -stash
   ```

2. Create the MQ client CSR:
   ```
   # runmqakm -certreq -create -db client.kdb -stashed -label client-personal -dn "CN=MQCLIENT,O=IBM,C=US,OU=MQ Support,ST=NorthCarolina" -size 2048 -sig_alg SHA256WithRSA -file certreq-client.arm
   ```

3. Send the CSR to a Certificate Authority (CA) to be signed.

4. received the signed certificate:

```
# runmqakm -cert -receive -file signed-client.arm -db client.kdb -stashed
```

5. Add the CA trusted, on this tutorial is the same CA root that signed the Qmgr's certificate:
```
# runmqakm -cert -add -db client.kdb -stashed  -label root-ca-cert -file /home/mqm/ca-certs/rootca.arm
```

Create the Kubernetes Secrets in OpenShift Web Console

Note: The secrets can also be created in the OpenShift command line using "oc" commands. This tutorial uses the Console GUI.

1. Log in to the OpenShift Web Console
2. Workloads → Secrets → Create → Key/Value secret.
   - Secret name: unique name for the secret
   - Key: certificate name with ".crt" or ".key" extension. For example: qmgr.crt
   - Value: The file that contains the certificate or the pem format value.

Project: marianap

Key/value secrets let you inject sensitive data into your application as files or environment variables.

Secret name *

test-qmgr-secret

Unique name of the new secret.

Remove key/value

Key *

qmgr.crt

Value

Browse...

Drag and drop file with your value here or browse to upload it.


```
DSnu7woE8/lmEmc1y4gBIO3CM4zBh8qG2RbaOz9i4IrI1ZHkkEfEiqirGnreWUCP
HhHydtLowEHlHSz50lVryS4HUHJglvgF3txkRx8aZqsHb+UeUC1Fk7PXyu9R3vNM
36MNobMMDvdNi+tXRXdz8gjsZLHY/SJqPAO0/91DdUfTFQ==
-----END CERTIFICATE-----
```


Remove key/value

Key *

qmgr.key

Value

Browse...

Drag and drop file with your value here or browse to upload it.


```
gEgi6q13WVxc6/KCmh4WZ8HqtWZHwjaB/kL9ausezjVczlagzzik0mMJvaFF2vks
81reC011Ivnmtp/SEFK0DvwkQYjMu7tDFDHdXZsLPMxyHbuFjfYnehgtheJIQsqK
I39qoK+EAmEXTpwF0o+q480=
-----END PRIVATE KEY-----
```


Add key/value

Create Cancel

Create a ConfigMap to pass MQSC commands.

----

1. Log in to the OpenShift Web Console
2. Workloads → ConfigMaps → Create ConfigMap → Configure via: Form View
3. Name: unique name for the configMap
4. Data – Key: name with ".mqsc" extension
5. Value: can upload a file or type the mqsc commands.
6. Create

On this tutorial we define a local queue and a SVRCONN channel that MQ external client will use to connect.

Create ConfigMap

Config maps hold key-value pairs that can be used in pods to read application configuration.

Configure via: (•) Form view ( ) YAML view

Name *

qmgr-test-configmap

A unique name for the ConfigMap within the project

[ ] Immutable

Immutable, if set to true, ensures that data stored in the ConfigMap cannot be updated

Data

Data contains the configuration data that is in UTF-8 range

Remove key/value

Key *

qmgr.mqsc

Value

Browse...

Drag and drop file with your value here or browse to upload it.

```
DEFINE QLOCAL('TESTQ') REPLACE

DEFINE CHANNEL(CLIENTSSL) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCAUTH(OPTIONAL)
SSLCIPH('TLS_RSA_WITH_AES_256_CBC_SHA256') REPLACE
```

Add key/value

Binary Data

BinaryData contains the binary data that is not in UTF-8 range

Add key/value

Create Cancel

Alter the queue manager YAML file to add the Secrets.

---

1. Log in to the OpenShift Web Console.
2. Operators → Install Operators → IBM MQ → Queue Manager (tab) → Queue manager name → YAML (tab)
3. Under "spec", add a "pki" section for defining the keys and certificates to load at Qmgr pod creation time.
   For keys: spec.pki.keys.secret
   For trusted certificates: spec.pki.trust.secret

Reference to ".spec.pki" API documentation:
https://www.ibm.com/docs/en/ibm-mq/9.3?topic=mqibmcomv1beta1-api-reference-

queuemanager#ctr_api_v1beta1_QueueManager__.spec.pki__title__1

NOTE: If you create the secrets before creating the queue manager, the secrets and configMaps can be passed at the time of creating the Qmgr to avoid dealing with YAML indentation issues.

Alter the Qmgr YAML file to add the ConfigMap
----

1. Log in to the OpenShift Web Console.
2. Operators → Install Operators → IBM MQ → Queue Manager (tab) → Queue manager name → YAML (tab)
3. Under “spec.queueManager” add a “mqsc” attribute with the configMap information: For example: spec.queueManager.mqsc.configMap

Reference to the “.spec.queueManager.mqsc” API documentation:
https://www.ibm.com/docs/en/ibm-mq/9.3?topic=mqibmcomv1beta1-api-reference-queuemanager#ctr_api_v1beta1_QueueManager__.spec.queueManager.mqsc__title__1

Example of the Qmgr YAML file after adding reference to the Secrets and ConfigMap:

```
98   spec:
99 >   license: ...
103    pki:
104      keys:
105        - name: qmgrpersonal
106          secret:
107            items:
108              - personal.crt
109              - personal.key
110            secretName: mq-test-personal-cert
111      trust:
112        - name: root
113          secret:
114            items:
115              - root.crt
116            secretName: mq-test-ca-certs
117    queueManager:
118      availability:
119        type: SingleInstance
120      mqsc:
121        - configMap:
122            items:
123              - mq2024.mqsc
124            name: mqtest-2024-config
125      name: QMGR2024
```

Once the Qmgr YMAL file modifications are complete, click on “Save” then “Reload”.

Verify the certificates from the Qmgr pod terminal.

From the OpenShift console go to Workloads → Pods → Qmgr pod → Terminal (tab)

+ List the certificates using the following command:

```
# runmqakm -cert -list -db /run/runmqserver/tls/key.kdb -stashed
```

<u>Example output:</u>

```
Certificates found
* default, - personal, ! trusted, # secret key
!       CN=RootSigner,OU=CertificateAuthority,O=MMCA,ST=NorthCarolina,C=US
!       CN=cs-ca-certificate
-       qmgrpersonal
```

+ View the certificate details by running:

```
# runmqakm -cert -details -label qmgrpersonal -db /run/runmqserver/tls/key.kdb -stashed
```

+ Validate the certificates:

```
# runmqakm -cert -validate -db /run/runmqserver/tls/key.kdb -stashed
```

Example output:

```
CN=RootSigner,OU=CertificateAuthority,O=MMCA,ST=NorthCarolina,C=US : OK
CN=cs-ca-certificate : OK
qmgrpersonal : OK
```

** The same command can be used to validate the MQ client certificates.

+ Verify the SVRCONN channel attributes SSLCIPH, SSLPEER and SSLCAUTH.
SSLCIPH – determine the cipherSpec to use for encryption.
SSLPEER – used if you are going to allow specific certificates based on Distinguished Name.
SSLCAUTH – determines one-way or two-way certificate authentication.

At this point we are done with the certificate configuration.

Obtain the Qmgr route.

To connect an MQ Client or Qmgr that is external to the Qmgr deployed in OpenShift cluster, we need to know the OpenShift route, and this will be use in the CONNAME channel attribute.

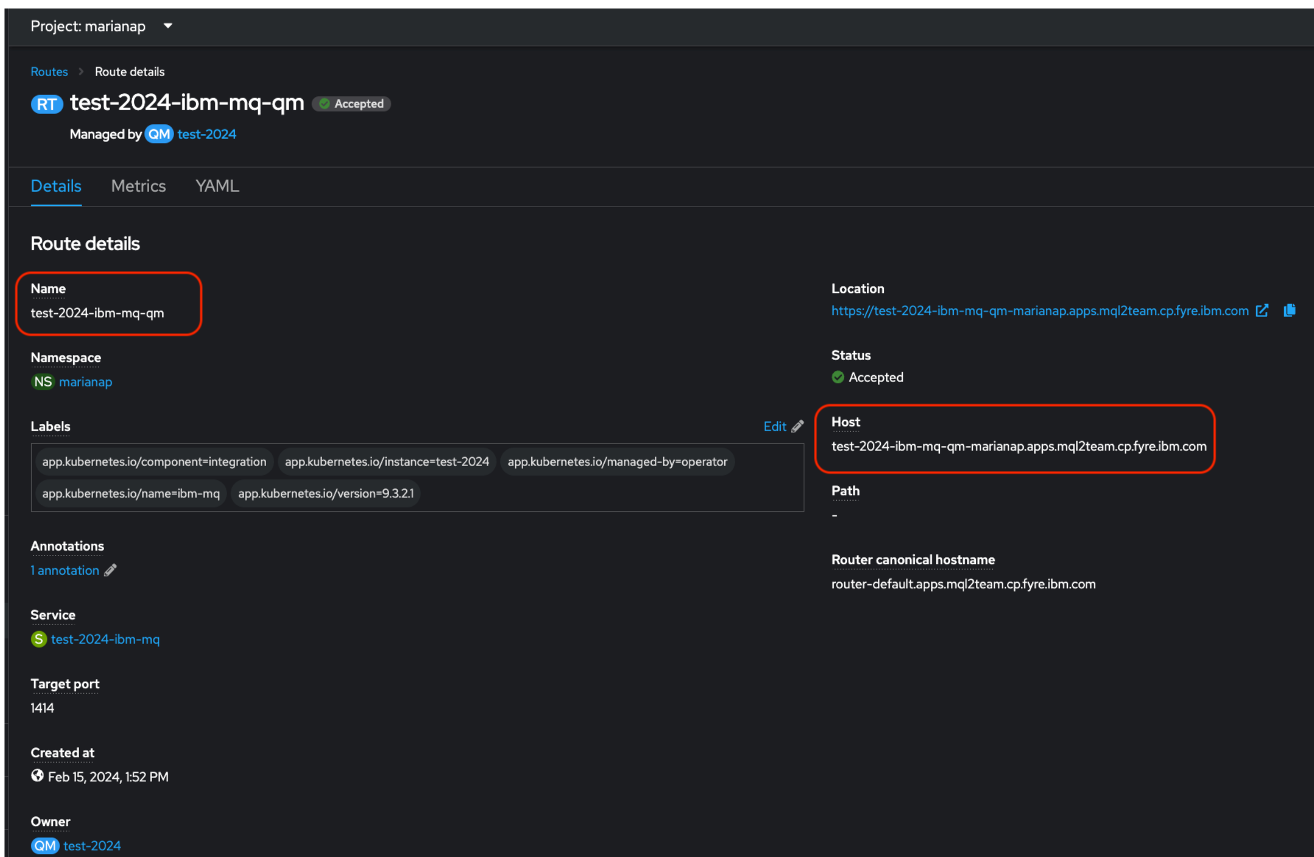
From the OpenShift console go to Networking → Routes → search for the Qmgr name. You will find two default routes, one ending on “mq-qm” and the other ending with “mq-web”.

For example:

- test-2024-ibm-mq-qm
- test-2024-ibm-mq-web

We need the one ending on “mq-qm”. This route contains a value for “Host”, this is what an external client will use to connect.

For example:
test-2024-ibm-mq-qm-marianap.apps.mql2team.cp.fyre.ibm.com



With this information we are ready to test the connection.

Configuration on the MQ Client

On this tutorial we are testing with the MQ sample programs: “amqsputc” and “amqsgetc”

1. Create a MQ client.ini file and add the following stanza:

   ```
   SSL:
     OutboundSNI=HOSTNAME
   ```

   This is very important because we are using the default route created by the MQ Operator. If you create an additional route based on a channel, then you need to set the client.inif file with the OutboundSNI equals to CHANNEL.

2. Create a CCDT file. For this tutorial we are creating a CDDT in json format, it is easy to create and human readable.

Example of ccdt.json:

```
{
   "channel":
   [
      {
         "name": "CLIENTSSL",
         "clientConnection":
         {
            "connection":
            [
               {
                  "host": "test-2024-ibm-mq-qm-marianap.apps.mql2team.cp.fyre.ibm.com",
                  "port": 443
               }
            ],
            "queueManager": "QMGR2024"
         },
         "transmissionSecurity":
         {
           "cipherSpecification": "TLS_RSA_WITH_AES_256_CBC_SHA256",
         },
         "type": "clientConnection"
      }
  ]

}
```

3. Set the MQ client environment, using the following environment variables:

```
export MQCCDTURL=file:/home/mqm/cloud/client-2024/ccdt.json
export MQSSLKEYR=/home/mqm/cloud/client-2024/client
export MQCLNTCF=/home/mqm/cloud/client-2024/mqclient.ini
```

4. Check the CCDT was configured correctly by running:

```
# runmqsc -
        dis chl(*) all
```

   You should see the CLNTCONN channel definition.

5. Test the connection:

```
# amqsputc TESTQ QMGR2024
```

```
[mqm@garpikes1 client-2024]$ amqsputc TESTQ QMGR2024
Sample AMQSPUT0 start
target queue is TESTQ
TEST FROM LINUX

Sample AMQSPUT0 end
[mqm@garpikes1 client-2024]$
```

+++ end +++