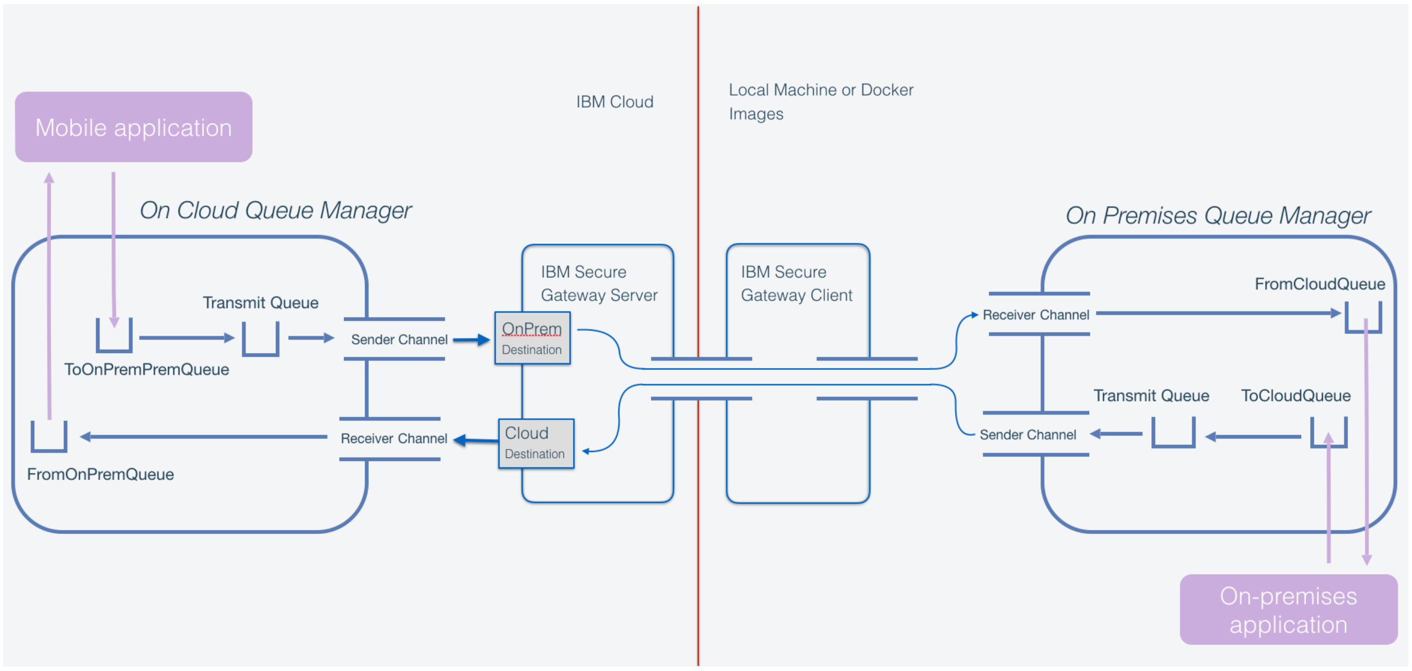
Connecting an on-premise queue manager to an IBM MQ On Cloud queue manager via the IBM Secure Gateway.



1. TABLE OF CONTENTS

2. CONNECTING TO AN ON-PREMISES QUEUE MANAGER

2.1. OVERVIEW

This document describes how to connect a cloud-based IBM MQ Queue Manager to an 'on premise' Queue Manager via the IBM Secure Gateway. The content is divided into multiple sections, they contain details of how to establish a connection via the secure gateway using MQSC commands or by using the MQ Console user interface. The other sections establish how to set up one-way TLS and how to set up Mutual authentication TLS.

Be aware when copying and pasting commands, they may not copy in the correct format e.g. incorrect single quotes may need to be re-typed

2.2. INITIAL SETUP

There are two components of this system, which you will need to have installed:

1. A cloud-hosted queue manager deployed using the MQ service in IBM Cloud. In the following pages this is referred to as “MyCloudQM”.
2. An “on-premises” queue manager that will be connected to the cloud queue manager. The queue manager in this document is called “QM1”

2.3. THE STEPS TO FOLLOW

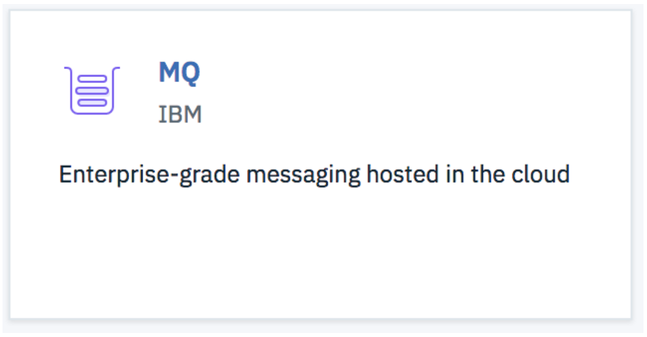
1. Create a cloud based MQOnCloud Queue Manager.
2. Create a local Docker image containing a MQ Queue Manager. (optional)
3. Create a cloud based Secure Gateway, install its client in a second Docker Image and configure the Secure Gateway to link the two Queue Managers.
4. Create channels to pass messages between the two Queue Managers.
5. Install TLS security.

2.4. CREATE AN MQ ON CLOUD QUEUE MANAGER

If you already have a queue manager, you can skip this section.

Log into your IBM Cloud dashboard.

In the IBM Cloud dashboard, select ‘Catalog’, then ‘Integrate’. You should see the tile you need in the ‘integrate’ category.



Click on the MQ tile - note the service name you are creating and click ‘Create’.

You now have a service running, but no queue manager, so click ‘Create’ in the queue managers list window. Give your queue manager a name, and a display name.

Select the size of queue manager which is appropriate for your requirements and click ‘Create’.

The creation of a queue manager sets up all the cloud infrastructure for you - this can take a few minutes. When this is finished, your queue manager will show in a list, and will have a status of ‘Running’. You must wait until that state shows before proceeding.

In order to be able to connect to the queue manager later, you will need to gather the connection information. You can do this as soon as the queue manager shows “running”.

Click on your queue manager in the list and select “Connection Information”. Download the Plain Text version and save the details in a file for later use.

Your text should look similar to this:

```
Platform: IBM MQ on Cloud

Queue manager name: MyCloudQM
Hostname: mycloudqm-c699.qm.us-preprod.mqcloud.ibm.com
Listener port: 31835

Application channel name: CLOUD.APP.SVRCONN
Administration channel name: CLOUD.ADMIN.SVRCONN

Deployment location: bmx-us-south

MQ Web Console login: https://web-mycloudqm-c699.qm.us-
preprod.mqcloud.ibm.com/ibmmq/console
```

If you do not already have one, you will also need a username and password to administer via the web console. You can create a user as follows.

Above the list of queue managers there are two tabs "user permissions" and "application permissions". Click the "user permissions" tab and create a new user. In the email address field give your own email address and click the button to “Generate MQ username”.

The name which is generated is the username which will be required to access the administration console later. The password will be the API key associated with your account - which you can reset when you first log into the console if you have forgotten it.

Follow the same procedure in the application permissions tab to create an application. Note that this tab asks for an API key name and will create an API key for you. This is not an account-wide API key - it applies only to this application, so every application has a different password. Download this, you can use it to post messages directly to the MQ On IBM Cloud queue manager.

You will use the hostname and the port to connect the secure gateway later. You will use the administration channel to help configure TLS later. Note that you are also given the address of the web console. You can use this in a browser, or alternatively you can click on the 'Administer' button next to the Connection Information button you used earlier.

Open the details panel of your queue manager and select "Adminstration" from the menu. Your login details will be displayed, and you have to opportunity to reset your API key if you do not know it.

Finally select the "Launch MQ Console" button - enter the credentials and log in.

2.5. CREATE A LOCAL QUEUE MANAGER

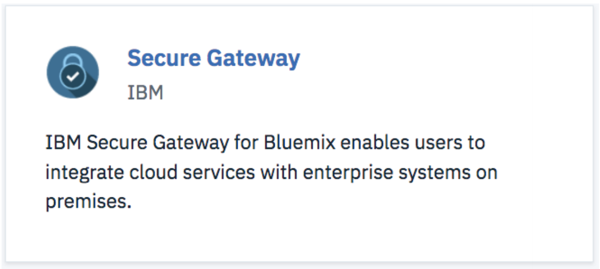
This document assumes that you already have a local queue manager, and can access it with administrative tools such as mqsc and the MQ Console.

Instructions for how to run MQ in a docker container as well as the docker image can be found here:

https://github.com/ibm-messaging/mq-container/blob/master/docs/usage.md

2.6. CREATE THE CLOUD-BASED SECURE GATEWAY

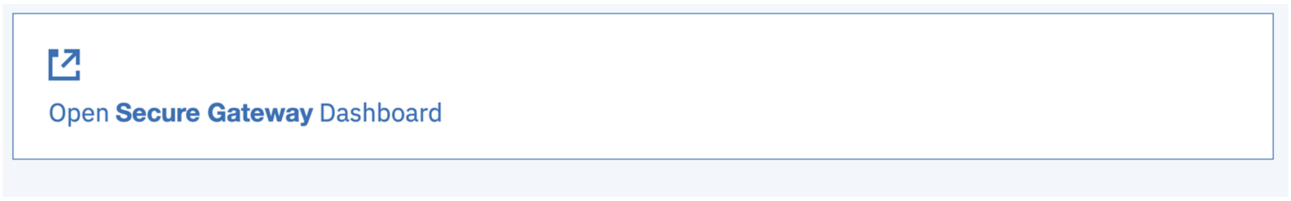
In a browser, go back you your IBM cloud catalog, and from the 'integrate' group select the 'IBM Secure gateway'. Click 'Create'. The number of destinations and clients is zero, as you have not configured any yet.



IBM Secure Gateway for Bluemix enables users to integrate cloud services with enterprise systems on premises.

You will be creating a two-way traffic of messages which may be initiated by either end of the flow, so in the IBM Secure Gateway you will need two destinations configured, and one client downloaded to the local environment.

First the gateway needs to be created. Once your service instance has been created, click the following button which will take you to the secure gateway dashboard:



On the dashboard. There is a large button that has a + symbol and says “Add Gateway”. Click this and name the gateway something memorable. Leave the other fields set to their defaults and click “add gateway”.

Click on “destinations” then select the big “+” under destinations. You will be asked to choose whether the destination is in the cloud or on-premise. The first destination will be “on premise”. This is for delivering messages from MQ on Cloud to the on-premise queue manager. Select the on-premise radio-button and click ‘Next’.

The host and port of your destination are those of your local queue manager. (In my case a docker image IP address, and port 1414).

By running

```
docker ps
```

You can get the id of your queue manager container. The command:

```
docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}' container_name_or_id
```

Enables you to get the IP address of the specified container

Click Next.

You will not be using TLS, so select “TCP” for the protocol, and do not upload certificates. Click Next.

Select “none” when asked what authentication your destination enforces. Click Next.

If you want to add IP table rules to permit traffic only from your MQ on Cloud, you can do that here. Finally give your destination a name and create it.

Follow the same procedure for a second destination, but this time make it a Cloud destination. The resource hostname and port come from the hostname and listener port fields in the connection information file you downloaded earlier, and the client port is 1414. The rest of the process is the same as the above.

The created destinations appear as grey panels, showing their names. There are icons in those panels, and the ‘wheel’ icon will show properties. The properties of the Cloud To ON-Prem destination will show a host and port which the Secure Gateway has generated to allow your MQ On Cloud queue manager to connect.

Now click the title ‘Clients(0), and the big “+” to connect a client.

When asked how you would like to connect this gateway, click on the Docker image.

Open a new command prompt on your local machine and paste the first command from the secure gateway window into the command prompt and press enter.

You should see the client start up and connect back to the server:

```
[2018-01-23 09:04:13.830] [INFO] (Client ID 1) No password provided. The UI will
not require a password for access
[2018-01-23 09:04:13.840] [WARN] (Client ID 1) UI Server started. The UI is not
currently password protected
[2018-01-23 09:04:13.841] [INFO] (Client ID 1) Visit localhost:9003/dashboard to
view the UI.
cli> [2018-01-23 09:04:14.087] [INFO] (Client ID 12) Setting log level to INFO
[2018-01-23 09:04:16.753] [INFO] (Client ID 12) The Secure Gateway tunnel is
connected
[2018-01-23 09:04:16.931] [INFO] (Client ID MyVZM7A0Ph6_rWZ) Your Client ID is
MyVZM7A0Ph6_rWZ
MyVZM7A0Ph6_rWZ> [2018-01-23 09:04:16.943] [INFO] (Client ID MyVZM7A0Ph6_rWZ)
Creating TCP server for reverse connections. Listening on port 1414
```

Note that your command prompt is now inside a running client instance, and you have the client commands available to you. Try 'help'.

Next the on-prem gateway needs to be given permissions. In the gateway client terminal enter the following command:

acl allow <IP/Hostname of Local queue manager>:<Port of local queue manager>
The above values are the same ones that were specified within the on-premise destination. Once you have entered this command. If you had a red hand showing on your on-premise destination it should now have disappeared.

A useful command to type in the client is start with is 'l DEBUG' - which switches on debug trace.

Now you have a client running, and that command prompt is fully occupied until you use the command "quit", so leave that command prompt now, and open a new one.

In the new window type:

```
docker ps
```

You should see the process corresponding to the secure gateway client.

You will need the docker internal IP address of the client to configure the queue manager. As you have just typed 'docker ps', you have the ID of the gateway image on the screen.

Type:

docker inspect -f '{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}}'
Gateway_container_name_or_id
This will return the IP address of the gateway client.

2.7. CONFIGURING QUEUE MANAGERS USING MQOC COMMANDS

Section 2.9 covers configuring the queue managers using the web console. This section covers using MQOC commands and could be considered more exact as the commands simply need to be copied and pasted with occasional values substituted in where shown. Any places where substitution is required is shown within < > parenthesis. Ensure you remove these parenthesis when you substitute in the correct variable.

The runmqsc terminal needs to be run from within the docker container. The command, "exec" can be used to create a bash terminal within the container.

WHEN COPYING AND PASTING THESE COMMANDS THE SINGLE QUOTES DO NOT ALWAYS COPY CORRECTLY
First, the ID of the container needs to be known.
Run:
```
docker ps
```
This will show the unique ID of your queue manager container.

Run:
```
docker exec -it <YOUR CONTAINER ID> /bin/bash
```
This will start a bash terminal within your docker container.

Run:
```
runmqsc
```
This will enable us to administer the queue manger via this bash terminal

Create a sender channel. This will allow us to send messages to the gateway. Run the following command:
```
define channel(ON_PREM.ON_CLOUD) chltype(sdr) conname(‘<IP_ADDRESS_OF_GATEWAY_CLIENT
>(1414)’) xmitq(LOCALXMITQ) trptype(tcp)
```

Create a receiver channel, this channel will receive messages, run:
```
define channel(ON_CLOUD.ON_PREM) chltype(RCVR) trptype(TCP)
```
This will create a new receiver channel for you.

We now have a sender channel and receiver channel, however they are unable to communicate until we specify their channel authentication records. Run the following two commands:
```
SET CHLAUTH(ON_PREM.ON_CLOUD) TYPE (ADDRESSMAP) ADDRESS(‘*’) USERSRC(CHANNEL)
SET CHLAUTH(ON_CLOUD.ON_PREM) TYPE (ADDRESSMAP) ADDRESS(‘*’) USERSRC(CHANNEL)
```

Next we need to create the transmission queue. We specified “LOCALXMITQ” as the transmission channel when we created our sender channel. Run this command to create it:
```
define Qlocal (LOCALXMITQ) Usage(XMITQ)
```

Next we need to create a remote queue, set the remote queue manager name to be the same as the name as your IBM Cloud Queue Manager . Run:
```
define QRemote(TOCLOUDQ) XMITQ(LOCALXMITQ) RNAME(FROMPREMQ) RQMNAME(<YOUR CLOUD QM
NAME>)
```

Create a local queue to receive messages by running:
```
define qlocal(FROMCLOUDQ)
```

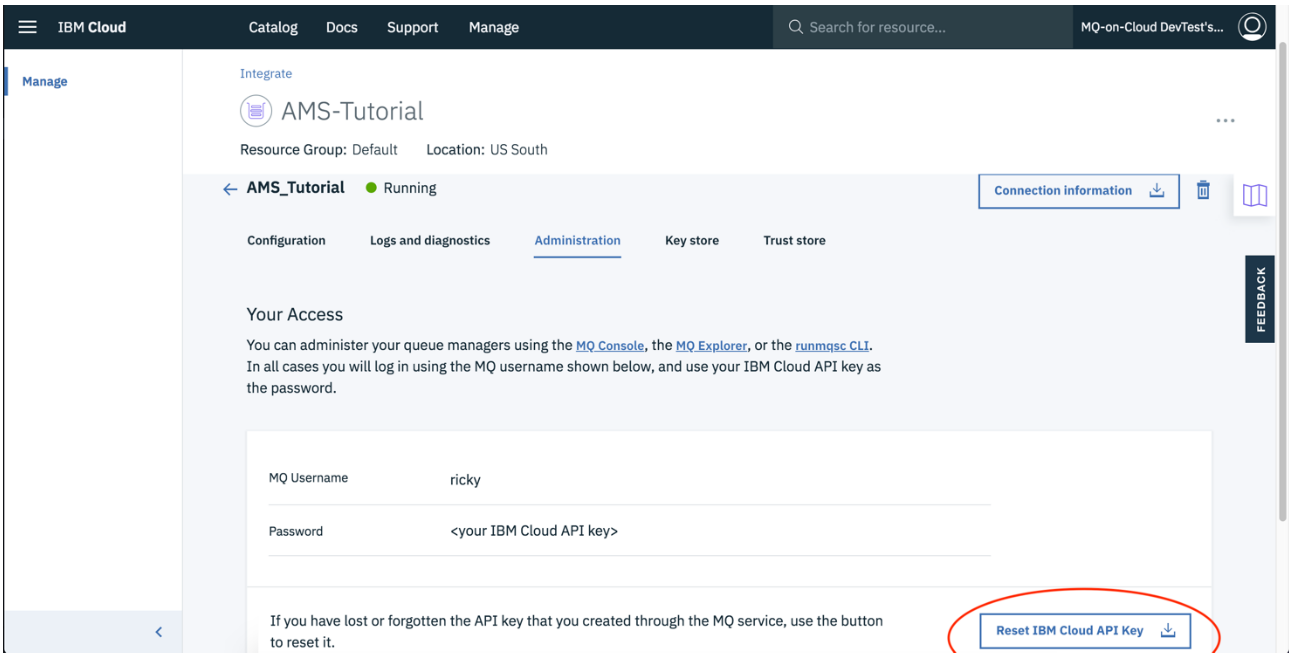
Run:
```
exit
```
This will exit the runmqsc terminal

Run:
```
exit
```
This will exit the docker terminal

The local queue manager is now fully configured

Now, it is time to configure the remote queue manager

In the following steps you will require a platform API key file. This file can be downloaded from the administration panel of your queue manager on the IBM cloud console. By clicking on “Reset IBM Cloud API key as shown below”



Resource Group: Default   Location: US South

Your Access

You can administer your queue managers using the MQ Console, the MQ Explorer, or the runmqsc CLI.
In all cases you will log in using the MQ username shown below, and use your IBM Cloud API key as the password.

MQ Username	ricky
Password	<your IBM Cloud API key>

If you have lost or forgotten the API key that you created through the MQ service, use the button to reset it.

Run:
```
export MQSERVER="CLOUD.ADMIN.SVRCONN/TCP/<CLOUD_HOSTNAME>(<CLOUD_PORT>)"
```
This will point the console at the on-cloud queue manager
The hostname and port are specified in the connection info file downloaded earlier.

You will require the runmqsc client for the following step. If you do not have it, it is available for download from the following link:
https://developer.ibm.com/messaging/mq-downloads/

Run:
```
runmqsc -c -u <MQ USERNAME> -w60 <QUEUE_MANAGER_NAME>
```
This will start a window to administer your MQ on Cloud queue manager, the MQ Username is found in the platform_apikey file.

The queue manager name is the name of your queue manager which can be found in the connection_info file.

After entering this command you will be prompted for a password. The password is the Platform_apikey stored in the Platform API_KEY file.

Create a receiver channel, this channel will receive messages, run:
```
define channel(ON_PREM.ON_CLOUD) chltype(RCVR) trptype(TCP)
```
This will create a new receiver channel for you. Notice that the name is the same as the sender channel on the local queue manager.

Create a sender channel. This will allow us to send messages to the gateway. Run the following command:
```
define channel(ON_CLOUD.ON_PREM) chltype(sdr)
conname(‘<GATEWAY_DEST_HOSTNAME>(<GATEWAY_DEST PORT>)’) xmitq(CLOUDXMITQ) trptype(tcp)
```
The connection details are those published by the OnPremise destination in the secure gateway. Go back to the secure gateway panel and click the ‘wheel’ properties icon in the bottom right of the grey panel

representing the destination, and copy the “Cloud host:port”. Note that IBM MQ requires the hostname and port to be in the form host.name(port) - with braces instead of the usual colon.
We now have a sender channel and receiver channel, however they are unable to communicate until we

specify their channel authentication records. Run the following two commands:
```
SET CHLAUTH(ON_PREM.ON_CLOUD) TYPE (ADDRESSMAP) ADDRESS(‘*’) USERSRC(CHANNEL)
SET CHLAUTH(ON_CLOUD.ON_PREM) TYPE (ADDRESSMAP) ADDRESS(‘*’) USERSRC(CHANNEL)
```

Next we need to create the transmission queue. We specified “LOCALXMITQ” as the transmission channel when we created our sender channel. Run this command to create it:
```
define qlocal (CLOUDXMITQ) Usage(XMITQ)
```

Create a local queue to receive messages by running:
```
define qlocal(FROMPREMQ)
```

Next we need to create a remote queue, set the remote queue manager name to be the same as the name as your local queue Manager . Run:
```
define QRemote(TOPREMQ) XMITQ(CLOUDXMITQ) RNAME(FROMCLOUDQ) RQMNAME(<YOUR LOCAL QM NAME>)
```

Your On Cloud queue manager is fully configured.

DO NOT EXIT THE RUNMQSC TERMINAL as it is needed in the next step

2.8. TESTING YOUR CONNECTION

In the runmqsc window for your on cloud queue manager, enter the following command:
```
START CHANNEL(ON_CLOUD.ON_PREM)
```
This starts the sender channel on the on cloud queue manager.

Enter:
```
exit
```
To exit the runmqsc terminal

Enter the terminal of your on-premise queue manager with the following command:
```
docker exec -it <YOUR CONTAINER ID> /bin/bash
```

Open a runmqsc terminal by entering:
```
runmqsc
```

Start the sender channel with:
```
START CHANNEL(ON_PREM.ON_CLOUD)
```

Enter: exit to exit the runmqsc terminal, but ensure the docker terminal remains open.

Now refresh any and all MQ on Cloud browser windows you currently have open so that they pick up the changes made. Then enter the MQ on Cloud web console.

In your docker terminal enter the following command:
```
./opt/mqm/samp/bin/amqsput TOCLOUDQ <YOUR LOCAL QUEUE MANAGER NAME>
```
This will open an application to send messages. Type any message then press ENTER twice

Refresh your MQ on Cloud Web Console. You should see the FROMPREMQ now has a depth higher than 0, depending on how many messages you entered.

2.9. USING THE WEB CONSOLE TO CONFIGURE QUEUE MANAGERS

If you completed section 2.7 and 2.8 you can skip this section

In order to complete this section, you must have MQ Consoles open on both the local and cloud based queue managers.

In the MQ Console window belonging to the local queue manager, click “+” in the “channels” panel.

Enter a memorable channel name (I called mine “OnPremToCloud’).

The channel type is Sender - the local queue manager is going to initiate the send.

The ‘conn name’ is the IP address of the gateway - with port 1414 - in the following format:

```
172.17.0.3(1414)
```

The transmission queue is the queue which the channel will use to send messages - you have not created that yet, so just give it a memorable name (mine was ‘ToCloud’).

Now create another channel, called “CloudToOnPrem, of type “Receiver”. The other parameters are all the same as above - there is no IP address or port - this is for incoming messages.

These two channels must be authorised to communicate.

If you do not already have a panel headed “Channel Authentication Records” then click on the ‘Add Widget’ icon at the top of the page, and choose the “Channel Authentication Records” panel.

Add a new authentication record for each channel, by clicking the plus icon (+) in the new panel.

Select ‘allow’ and from the identity list choose “Address”, and click “Next”.

The channel profile is the name of the channel (OnPremToCloud), and the address is “*” (without the inverted commas). Now click “Create”.

Do the same for the cloud inbound channel (CloudToOnPrem)

Next you will create the queues:

Add a new local queue called “ToCloud”. When it is created, click on the ‘properties’ icon , and set the usage to “Transmission”. This will be the transmission queue for outbound messages to the cloud.

Add a new remote queue called “ToCloudQueue”.

In the properties panel - the remote queue name is “FromOnPremQueue”. The remote queue manager is the name of your IBM MQ On Cloud queue manager. The transmission queue is “ToCloud”

Create a new local queue called “FromCloudQueue”.

These are all the artefacts you need locally, next you must add the corresponding queues and channels in the MQ On Cloud queue manager.

Go back to the web console belonging to the MQ On Cloud queue manager.

Create a channel called “OnPremToCloud” of type ‘receiver’ - to match the ‘sender’ on the local side. (ToCloudQueue)

Create a channel called “CloudToOnPrem” of type ‘sender’ - to match the receiver at the On-Prem side. The transmission queue is “ToClient”. The connection details are those published by the OnPremise destination in the secure gateway. Go back to the secure gateway panel and click the ‘wheel’ properties icon in the bottom right of the grey panel representing the destination, and copy the “Cloud host:port”. Note that IBM MQ requires the hostname and port to be in the form host.name(port) - with braces instead of the usual colon.

Add the same two authentication records as you did for your local queue manager channels.

Create your ToClient local transmission queue.

Add a new local queue to your on-cloud queue manager called “FromOnPremQueue” to receive messages from the local queue manager.

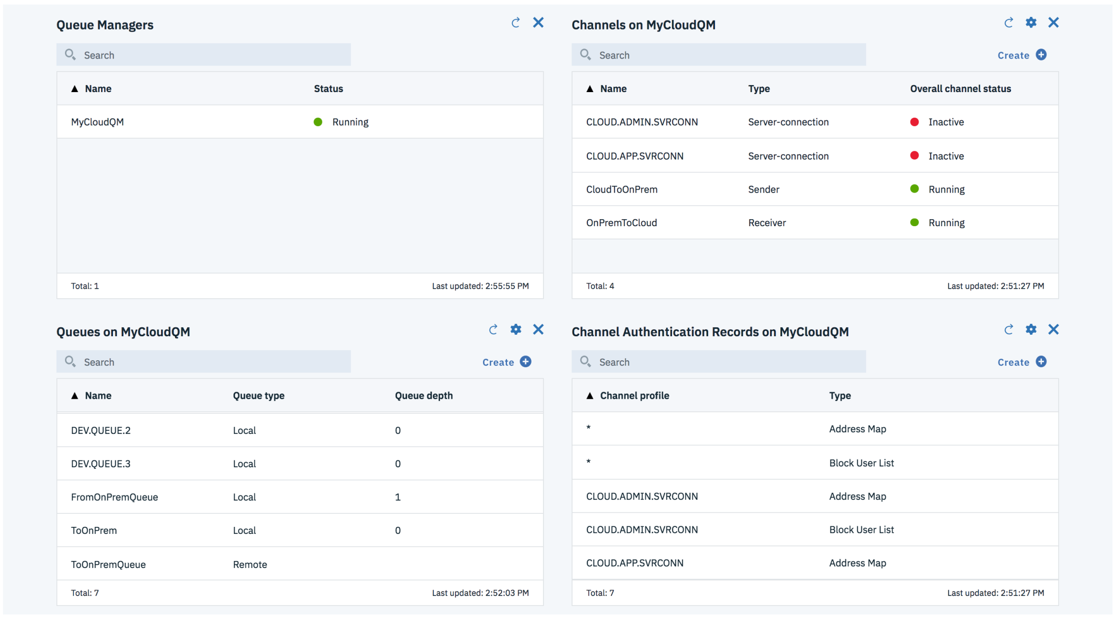
Add a new remote queue to your on-cloud queue manager called “ToOnPremQueue” to send messages.Edit its properties, and set the Remote Queue to “FromCloudQueue”. Set the remote queue manager to the name of your on-premises queue manager. The transmission queue should be ToClient.

You should now have all the pieces in place to run the system without TLS security.

In each console, select the sender channel, and click the ‘start’. You should see them bind, then start.

With these two channels started, you should be able to put a message on the ToCloudQueue in the local queue manager, and see it appear in the cloud queue manager FromOnPremQueue. You can put a message on the queue with using the web console (using the 'envelope' icon in the queue panel)
(Note you have to refresh the panel in the remote console to see the change).

You should also be able to put a message on the ToOnPremQueue in the cloud queue manager, and see it appear in the



Queue Managers

Search

▲ Name	Status
MyCloudQM	● Running

Total: 1 Last updated: 2:55:55 PM

Channels on MyCloudQM

Search Create

▲ Name	Type	Overall channel status
CLOUD.ADMIN.SVRCONN	Server-connection	● Inactive
CLOUD.APP.SVRCONN	Server-connection	● Inactive
CloudToOnPrem	Sender	● Running
OnPremToCloud	Receiver	● Running

Total: 4 Last updated: 2:51:27 PM

Queues on MyCloudQM

Search Create

▲ Name	Queue type	Queue depth
DEV.QUEUE.2	Local	0
DEV.QUEUE.3	Local	0
FromOnPremQueue	Local	1
ToOnPrem	Local	0
ToOnPremQueue	Remote	

Total: 7 Last updated: 2:52:03 PM

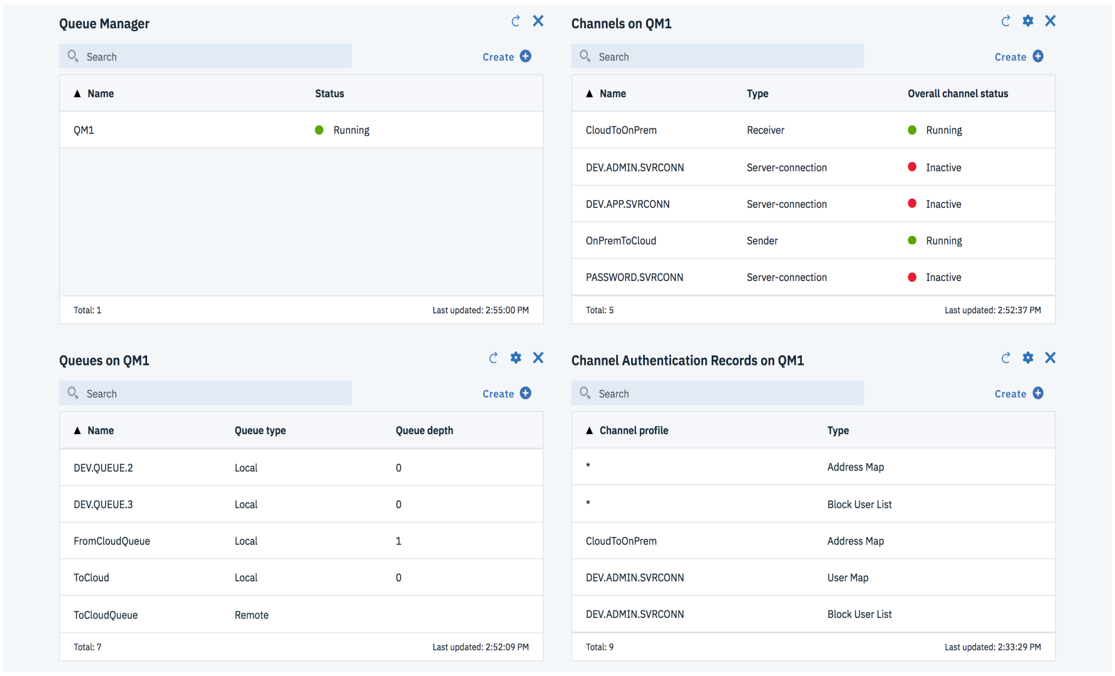
Channel Authentication Records on MyCloudQM

Search Create

▲ Channel profile	Type
*	Address Map
*	Block User List
CLOUD.ADMIN.SVRCONN	Address Map
CLOUD.ADMIN.SVRCONN	Block User List
CLOUD.APP.SVRCONN	Address Map

Total: 7 Last updated: 2:51:27 PM

FromCloudQueue on the local server.



Queue Manager

Search Create

▲ Name	Status
QM1	● Running

Total: 1 Last updated: 2:55:00 PM

Channels on QM1

Search Create

▲ Name	Type	Overall channel status
CloudToOnPrem	Receiver	● Running
DEV.ADMIN.SVRCONN	Server-connection	● Inactive
DEV.APP.SVRCONN	Server-connection	● Inactive
OnPremToCloud	Sender	● Running
PASSWORD.SVRCONN	Server-connection	● Inactive

Total: 5 Last updated: 2:52:37 PM

Queues on QM1

Search Create

▲ Name	Queue type	Queue depth
DEV.QUEUE.2	Local	0
DEV.QUEUE.3	Local	0
FromCloudQueue	Local	1
ToCloud	Local	0
ToCloudQueue	Remote	

Total: 7 Last updated: 2:52:09 PM

Channel Authentication Records on QM1

Search Create

▲ Channel profile	Type
*	Address Map
*	Block User List
CloudToOnPrem	Address Map
DEV.ADMIN.SVRCONN	User Map
DEV.ADMIN.SVRCONN	Block User List

Total: 9 Last updated: 2:33:29 PM

2.10. INSTALLING STANDARD (ONE WAY) TLS SECURITY

The two queue managers can be configured to use TLS security end-to-end through the secure gateway without terminating the TLS at the gateway.

You will need a command prompt to access the MQ command line interface as the console does not yet support one of the features you need. All this configuration can be done from the local side using mqsc and the command prompt, as the cloud queue manager already has a running admin channel which the local queue manager can talk to.

The message conversations are initiated by both ends of the flow, so you will need to put the certificate from the local queue manager into the trust store of the cloud queue manager, and also insert the certificate from the cloud queue manager into the key.kdb for the local queue manager. You also need to set a cipher spec on both ends of both the channels.

The local queue manager will require the certificate from the IBM Cloud queue manager, but it will also require the whole chain of certificates up to the root. There are two ways of getting this certificate chain - both are described below.

In the MQ On Cloud console Queue Manager details page, there is a tab labelled “Key Store”. Selecting this tab shows the certificates currently in use by your queue manager. Clicking on the three dots menu in the titlebar of a certificate allows you to download the public part to a file. This action will also check whether the certificate is self-signed, and if not, it will download the corresponding key chain up to the issuer of the certificate into the same file.

Alternatively, you could open the IBM MQ Console in a browser (for example FireFox) which supports downloading certificates.

With the console open, click the padlock icon next to the URL, and select “Show Connection Details”, followed by “More Information”, “View Certificate”, “Details”,”Export...”. In the format dropdown select “X509 certificate with chain PEM”. Save this to a file for later use. It is referred to later in this document when adding certificates to the database as <your downloadedfile>.

With the second method, you might choose to click on each certificate up the chain and export each individually to separate files.

Next either create a key store , or add the certificate to the key store. In a command prompt (inside your local queue manager docker image if you have one):

```
cd /var/mqm/qmgrs/QM1/ssl
ls
```

You may already have a key.kdb file, in which case you can add the cloud certificate to that.

If you do not have a key.kdb, create one now:

```
runmqakm -keydb -create -db key.kdb -pw {choose a password} -stash
```

If you do not have a local certificate, create one now: (assuming QM1 is the name of your queue manager)

```
runmqakm -cert -create -db key.kdb -stashed -dn "CN=LOCALQM,O=IBM,C=GB" -label
ibmwebspheremqqm1
```

To export your local certificate:

```
runmqakm -cert -extract -db key.kdb -stashed -label ibmwebspheremqqm1 -file
cert.cer
```

Next add the cloud public certificate you downloaded. The label should be ibmwebspheremq<qname> - where <qname> is the name of your cloud queue manager IN LOWER CASE.

```
runmqakm -cert -add -db key.kdb -stashed -label ibmwebspheremq<your qmname>
-file  <your downloadedfile>
```

Verify that your key is present:

```
runmqakm -cert -list -db key.kdb -stashed
```

If you created a new key.kdb, make sure it is owned by mqm:

```
ls -l
chown mqm:mqm *
```

Finally, go back to the IBM Cloud console queue manager details page, and click the truststore tab. You can now upload the certificate you just exported to 'cert.cer' by selecting the 'import certificate' button. This certificate in my case is self-signing, so it has no chain. If yours is a Digicert or similar, you will need to store the chain of certificates in the trust store.

Now you have the certificates stored, you need to give each end of the channels a cipher spec. There are two ways of performing this task. The simplest way is to go back to the web consoles for each queue manager as described below, but for completeness we also describe how to perform this action using runmqsc remotely against the cloud queue manager.

The cipher specs are described here:

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.sec.doc/q014260_.htm

The cypher specifications can be set into the channels using the console. Go back to the console for the local queue manager, and select the OnPremToCloud channel - open its properties panel, and select SSL. Enter the chosen cypher spec name into the SSL cypher spec field, and save.

Do the same operation for the CloudToOnPrem channel, and the corresponding channels in the MQ On Cloud queue manager console.

Next - refresh SSL by selecting the queue manager panel in each console, and using the "..." menu to select "Refresh Security", then in the popup dialog select "SSL".

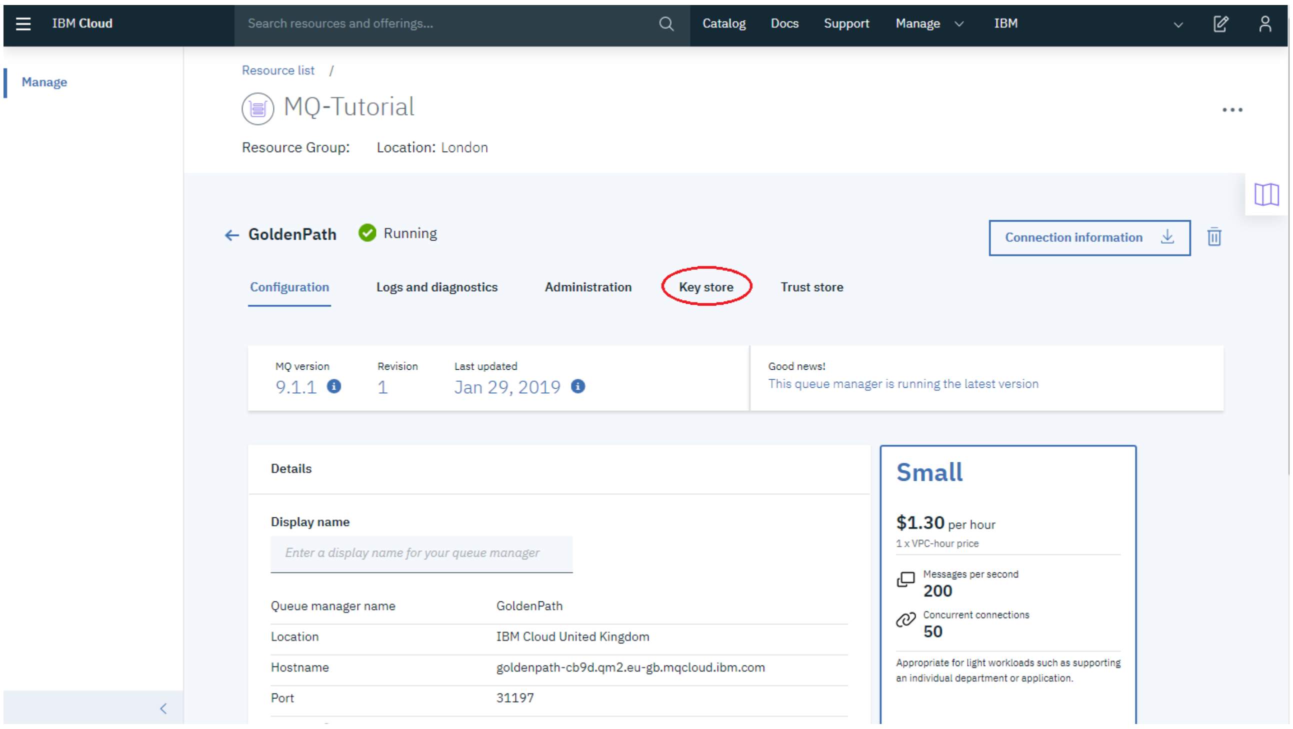
You should now be able to restart the two sender channels. They will now be configured to only start if the correct certificate exchange takes place.

2.11.INSTALLING MUTUAL (TWO WAY) TLS SECURITY

Standard TLS is commonly used when connecting to servers or websites. Using standard TLS, a client can be sure that it is connecting to the trusted server. In addition, mutual TLS ensures that the server only accepts connections from trusted clients. This is again achieved through the use of security certificates.

Each queue manager needs to have a keystore and a trust store. They also each need their own private key and the public key of the opposing trust store. IBM MQ on Cloud queue managers come with a certificate by default. So first we will ensure that the local queue manager can trust the on cloud queue manager.

First, we need the self-signed certificate from the queue manager. Navigate to your queue manager page on the IBM cloud webpage and select keystore as shown below.


IBM Cloud | Search resources and offerings... | Catalog | Docs | Support | Manage | IBM

Manage

Resource list /

MQ-Tutorial

Resource Group: Location: London

GoldenPath Running

Connection information

Configuration | Logs and diagnostics | Administration | Key store | Trust store

MQ version 9.1.1 | Revision 1 | Last updated Jan 29, 2019

Good news!
This queue manager is running the latest version

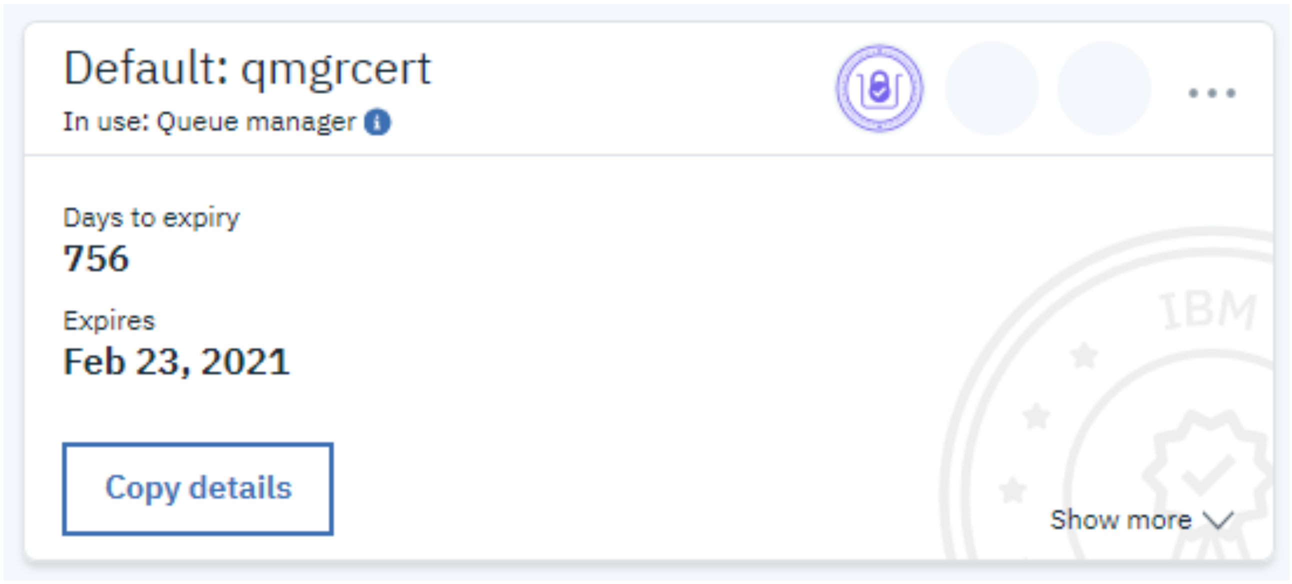
Details

Display name
Enter a display name for your queue manager

Queue manager name	GoldenPath
Location	IBM Cloud United Kingdom
Hostname	goldenpath-cb9d.qm2.eu-gb.mqcloud.ibm.com
Port	31197

Small
$1.30 per hour
1 x VPC-hour price
Messages per second 200
Concurrent connections 50
Appropriate for light workloads such as supporting an individual department or application.

A certificate labeled qmgrcert can be seen, click the three dots in the corner of the certificate, then on the download public certificate button. This can be seen below


Default: qmgrcert
In use: Queue manager

Days to expiry
756

Expires
Feb 23, 2021

A public certificate called "qmgrcert.pem" will be downloaded.

Next a trust store for the local queue manager needs to be created and the public certificate added so that it knows to trust the on-cloud queue manager.

The docker container needs a copy of the public certificate of the on-cloud queue manager.

To list the docker containers run the command:

```
docker ps
```

Run
docker cp <filepath to cert>/qmgrcert.pem <YOUR CONTAINER ID>:/qmgrcert.pem

Exec back onto the queue manager docker container using the command:
docker exec -it <YOUR CONTAINER ID> /bin/bash

Copy the certificate to the ssl directory of your queue manager using the following command
cp qmgrcert.pem var/mqm/qmgrs/<YOUR LOCAL QUEUE MANAGER NAME>/ssl/qmgrcert.pem

Navigate to the directory “ssl” with the following command
cd var/mqm/qmgrs/<YOUR LOCAL QUEUE MANAGER NAME>/ssl

Run the following command to create a key database on the local queue manger
runmqakm -keydb -create -db key.kdb -pw <ANY PASSWORD> -stash

Add the on cloud certificate to the database by running the following command
runmqakm -cert -add -db key.kdb -stashed -label ibmwebspheremq<your cloud qmname>   -file <your downloadedfile>

Your certificate is now added to the trust store of the local queue manager. Now it is time to pass a local certificate into the on cloud queue manager.

create a self-signed certificate and add it to the key database using the following command:
runmqakm -cert -create -db key.kdb -stashed -dn "CN=LOCALQM,O=IBM,C=GB" -label ibmwebspheremqqm1

Extract the public part of the new certificate by using the following command:
runmqakm -cert -extract -db key.kdb -stashed -label ibmwebspheremqqm1 -file cert.cer

exit the docker container by running the command:
exit

copy the certificate file out of the container to your local machine by running:
docker cp <YOUR CONTAINER ID>:var/mqm/qmgrs/<YOUR LOCAL QM NAME>/ssl/cert.cer cert.cer

Navigate on your browser to “trust store” on your MQ on cloud page. It is located next to the keystore page we accessed earlier. Click the import certificate button and browse to the certificate just copied from the docker container and upload it to the queue manager.

Next TLS needs to be enabled and security refreshed on both queue managers.

Exec back onto the queue manager docker container using the command:
docker exec -it <YOUR CONTAINER ID> /bin/bash

Start a runmqsc terminal by executing:
runmqsc

Begin with activating TLS on the sender channel with the command:
ALTER CHANNEL('ON_PREM.ON_CLOUD') CHLTYPE(SDR) SSLCIPH(ANY_TLS12)

Activate TLS on the receiver channel with the following command:
ALTER CHANNEL('ON_CLOUD.ON_PREM') CHLTYPE(RCVR) SSLCIPH(ANY_TLS12) SSLCAUTH(REQUIRED)

Refresh the security cache with the following command:

```
REFRESH SECURITY TYPE(SSL)
```

Restart the sender channel with the commands

```
STOP CHANNEL(ON_PREM.ON_CLOUD)
START CHANNEL(ON_PREM.ON_CLOUD)
```

Exit the runmqsc window with:

```
exit
```

exit the docker container with:

```
exit
```

Run:

```
export MQSERVER="CLOUD.ADMIN.SVRCONN/TCP/<HOSTNAME>(<PORT>)"
```

This will point the console at the on cloud queue manager
The hostname and port are specified in the connection info file downloaded earlier.

Run:

```
runmqsc -c -u <MQ USERNAME> -w60 <QUEUE_MANAGER_NAME>
```

This will start a window to administer your MQ on Cloud queue manager, the MQ Username is found in the platform_apikey file.

The queue manager name is the name of your queue manager which can be found in the connection_info file.

After entering this command you will be prompted for a password. The password is the Platform_apikey stored in the Platform API_KEY file.

Activate TLS on the On Cloud sender channel with the command:

```
ALTER CHANNEL('ON_CLOUD.ON_PREM') CHLTYPE(SDR) SSLCIPH(ANY_TLS12)
```

Activate TLS on the receiver channel with the following command:

```
ALTER CHANNEL('ON_PREM.ON_CLOUD’) CHLTYPE(RCVR) SSLCIPH(ANY_TLS12) SSLCAUTH(REQUIRED)
```

Refresh the security cache with the following command:

```
REFRESH SECURITY TYPE(SSL)
```

Restart the sender channel:

```
STOP CHANNEL(ON_CLOUD.ON_PREM)
START CHANNEL(ON_CLOUD.ON_PREM)
```

Enter the following command to exit the mq console:

```
exit
```

2.12. APPENDIX: USING RUNMQSC TO ADD CIPHER SPECS

As an alternative approach to cipher specs, at the command prompt in the local queue manager docker image:

```
runmqsc {queue manager name}
alter chl('OnPremToCloud') chltype(sdr) sslciph(ANY_TLS12)
alter chl('CloudToOnPrem') chltype(rcvr) sslciph(ANY_TLS12)
```

After those you will need to refresh security - this will stop the channels if they are running.

```
REFRESH SECURITY (*) TYPE(SSL)
```

Next, you will to use runmqsc on the MQ On Cloud queue manager, so quit the local one with ‘exit’.

You need to set an environment variable to say which server you are managing:

```
export MQSERVER=‘CLOUD.ADMIN.SVRCONN/TCP/{address of your cloud
manager}({port})’
```

Run the runmqsc command again, against the cloud queue manager:

```
runmqsc -c -u {user} {cloud_qm_name}
```

The user is the username you created for yourself earlier. The -u will prompt for and password - this is the API key you use to access the cloud resource.cAdd the same cipher specs as before...

```
alter chl('CloudToOnPrem') chltype(sdr) sslciph(ANY_TLS12)
alter chl('OnPremToCloud') chltype(rcvr) sslciph(ANY_TLS12)
REFRESH SECURITY (*) TYPE(SSL)

exit
```