

MQ Uniform Clusters

Messaging Developer

Messaging Administrator

| Summary: Uniform Cluster

MQ Uniform Clusters and Application Rebalancing

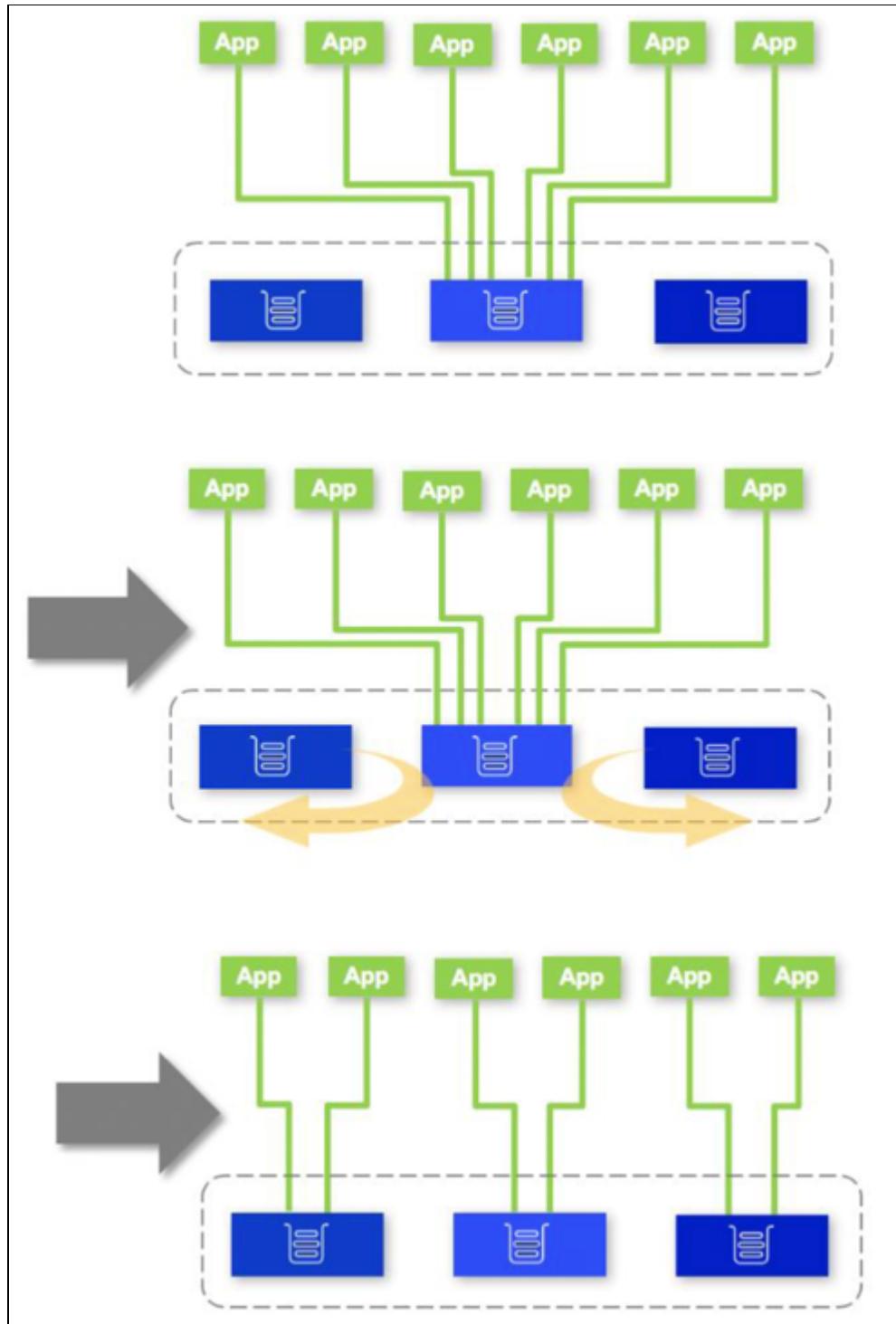
Featuring:

- Creating a Uniform Cluster
- Application Rebalancing
- Application Rebalancing & Queue Manager Outage
- Metrics
- Using CCDT Queue Manager Groups

Introduction

This lab introduces MQ Uniform Clusters and Application Rebalancing at the MQ 9.1.5 code level (and in the MQ 9.2 Long Term Support release). In this lab, you will:

- Create a Uniform Cluster quickly using configuration files, a feature introduced in MQ 9.1.4
- Run re-connectable sample applications to a queue manager, within the Uniform Cluster, to show automatic rebalancing of the apps to other queue managers in the Uniform Cluster



- Stop, then restart a queue manager, with connected apps to show automatic application rebalancing to remaining running queue managers in the Uniform Cluster
- Report resource usage metrics for the applications, introduced in MQ 9.1.5
- Connect an application to a Queue Manager Group instead of a queue manager

Further information

- IBM MQ Knowledge Center
- “Building scalable fault tolerant systems with IBM MQ 9.1.2 CD” article by David Ware, STSM, Chief Architect, IBM MQ: [David Ware article ↗](#)
- “Active/active IBM MQ with Uniform Clusters” video by David Ware: [YouTube Demo Video ↗](#)

Acknowledgements

The technical content of this document is based on materials originally provided by Lewis Weedon, Laurence Bonney, Jason Edmeades and Ian Edwards from IBM MQ Development.

The illustration of application balancing is taken from the aforementioned article by David Ware.

Lab Environment

To develop this lab guide, we used a single MQ 9.2.0.0 installation running on Windows 10.

A Windows user account – *ibmdemo* – has been created. This has been added to the Administrators and mqm groups. The password for *ibmdemo* is *passw0rd*.

Entering commands for this lab

In this lab, you will come across some long commands to enter. To avoid a lot of typing, it may be quicker to copy the commands from this document to a Notepad window, then edit and format them there, before copying and pasting them to the Windows Command Prompt.

Some points to consider:

- There may be hyphens (-) introduced in this document where text is split between lines; you should remove these
- Commands may occupy multiple lines in this document; you should concatenate these into a single line in a Notepad window
- Sometimes a command may be similar to a previous one that you have entered, and it may be quicker to use the up arrow in the Windows Command Prompt to retrieve the earlier command, and then editing it there on the command line

⚠ Important:

If a command doesn't work, check the hyphen in front of the parameters. When copying commands, occasionally the hyphen will appear as a long dash. Make sure to change it to a short dash (-).

Create Uniform Cluster using Configuration Files (New in MQ 9.1.4)

A Uniform Cluster, UNIDEMO, is to be created, comprising three queue managers: QM1, QM2 and QM3. QM1 and QM2 are to contain full repositories and QM3 is to contain a partial repository. The following are to be created for each queue manager in our cluster:

- a cluster receiver channel
- a cluster queue
- a server connection channel

Also, appropriate channel authentication needs to be set up for the server connection channel.

To help you do this, we have created two configuration files for you in C:\Student. The use of such files was introduced in MQ 9.1.4. This simplifies creation of a uniform cluster and ensures the queue managers in it are created in a consistent way.

These will be used when we create the queue managers in the cluster:

- uniclass.ini, which describes how the uniform cluster will look once it is set up. This will be added automatically to the qm.ini file for each queue manager during queue manager startup.
- uniclass.mqsc, which will be applied automatically to each queue manager at startup. Note that if you modify this file and restart a queue manager that uses it, then the definitions in the updated file will be applied to the queue manager as it restarts.

Review Cluster configuration files

1. Log on to the provided system as user **ibmdemo**, password **passw0rd**. This user is a member of the Administrators and mqm groups.
2. Examine the file **uniclass.ini** in C:\Student. This shows there will be two full repositories stored by queue managers QM1 and QM2. Connection names for these repositories are also given. The cluster name, **UNIDEMO**, is specified and it is to be a Uniform Cluster.

```
AutoCluster:  
Repository2Conname=127.0.0.1(3001)  
Repository2Name=QM1  
Repository1Conname=127.0.0.1(3002)  
Repository1Name=QM2  
ClusterName=UNIDEMO  
Type=Uniform
```

3. Examine the file **uniclus.mqsc** in C:\Student, numbered below for clarity:

- a. define channel('+AUTOCL+_+QMNAME+') chltype(clusrcvr) trptype(tcp)
conname('+CONNAME+') cluster('+AUTOCL+') shorttmr(5) shortrty(12000) replace
- b. define QL(QL) cluster(UNIDEMO) defbind(notfixed) clwluseq(any) maxdepth(99999999) replace
- c. define channel(TO_UNIDEMO) chltype(svrconn) trptype(tcp) replace
- d. SET CHLAUTH(TO_UNIDEMO) TYPE(ADDRESSMAP) ADDRESS(*) USERSRC(CHANNEL)
- e. SET CHLAUTH(TO_UNIDEMO) TYPE(BLOCKUSER) USERLIST('nobody') DESCRIPTOR('Allow privileged users on this channel')
- f. ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS)
CHKCLNT(OPTIONAL)
- g. REFRESH SECURITY

Line a – defines, for each queue manager, a cluster receiver channel, which in our case will be called UNIDEMO_QMn, where n=1, 2 or 3.

Line b – defines a local queue, which is shared in the cluster:

- To allow the messages to be spread across all queue managers in the cluster, the DEFBIND (Default bind type) parameter is set as shown
- Setting the CLWUSEQ (Cluster Workload Use Queue) parameter as shown will allow other queue managers in the Uniform Cluster to get messages from that queue
- Uniform Cluster automatic application rebalancing is not dependent on the queues used by applications being cluster queues. We're using cluster queues in this lab to demonstrate how messages put on to a single queue manager are shared among all the queue managers in the cluster.

Line c – defines a server connection channel for each queue manager.

Line d – allows connections through the server connection channel.

Line e – allows only privileged users to connect through this channel.

Line f – the queue manager uses the local operating system to authenticate any provided user ID and password. If these are provided by an application, they must be a valid pair, but it is not mandatory to provide them.

Line g – refreshes security settings.

Create queue managers and the Uniform Cluster

We shall now create the queue managers – and our cluster - using these configuration files.

1. Open a Windows Command Prompt session and change directory to C:\Student.

```
cd C:\Student
```

2. Before proceeding, shutdown and delete any queue manager with the name **QM1**, **QM2** or **QM3** as we shall be using these names for queue managers in this lab:

```
endmqm -w <queue manager>
```

```
dltmqm <queue manager>
```

3. Now create the queue manager **QM1**:

```
crtmqm -p 3001 -ii .\uniclus.ini -ic .\uniclus.mqsc -iv CONNAME=127.0.0.1(300  
1) QM1
```

Note:

- this queue manager will listen on port 3001
- we are providing automatic configuration of qm.ini attributes and also automatic MQSC commands to be run at startup, via the uniclus.ini and uniclus.mqsc files respectively
- the connection name parameter, comprising the loopback IP address of the host and the listener port of the queue manager, will be used by the cluster receiver channel definition in the uniclus.mqsc file

```
c:\Student>crtmqm -p 3001 -ii .\uniclus.ini -ic .\uniclus.mqsc -iv CONNAME=127.0.0.1(3001) QM1
IBM MQ queue manager created.
Directory 'C:\ProgramData\IBM\MQ\qmgrs\QM1' created.
The queue manager is associated with installation 'Installation1'.
Creating or replacing default objects for queue manager 'QM1'.
Default objects statistics : 87 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.

c:\Student>
```

4. Start this queue manager.

```
strmqm QM1
```

5. Similarly create queue manager QM2. This will also contain a full repository.

```
crtmqm -p 3002 -ii .\uniclus.ini -ic .\uniclus.mqsc -iv CONNAME=127.0.0.1(300
2) QM2
```

6. Now create queue manager QM3. As this is not specified in the uniclus.ini file, it will have a partial repository.

```
crtmqm -p 3003 -ii .\uniclus.ini -ic .\uniclus.mqsc -iv CONNAME=127.0.0.1(3003) QM3
```

7. Start these additional queue managers:

```
strmqm QM2
```

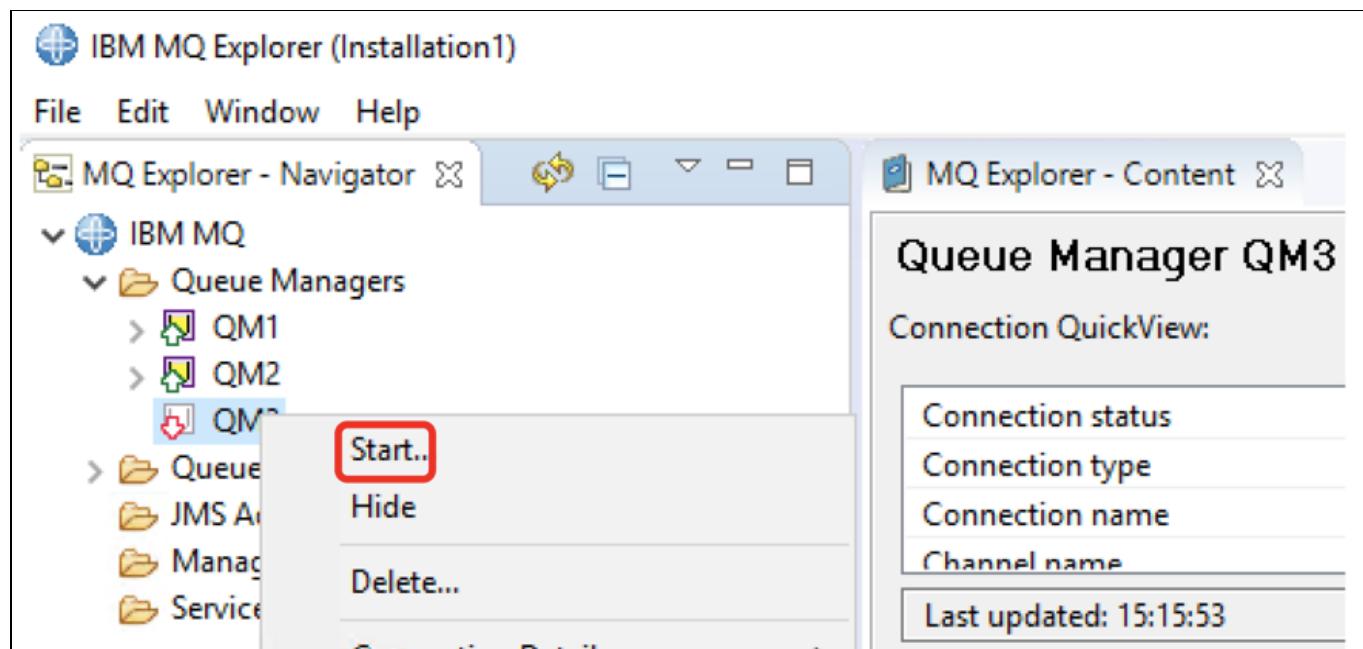
```
strmqm QM3
```

You should now have a fully functional Uniform Cluster!

Perform health-check on Uniform Cluster

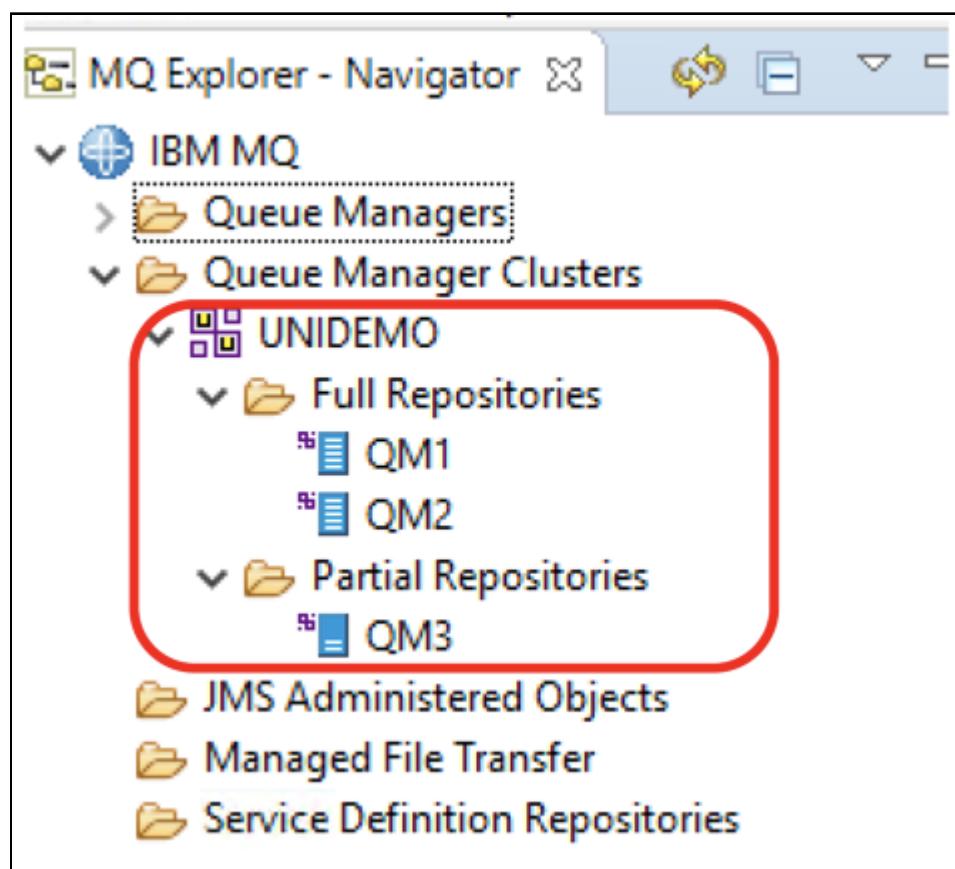
Before proceeding, we need to check the cluster is up and running.

1. Start *MQ Explorer* (you can do this from a Windows Command Prompt using the **strmqcfg** command). Check that queue managers **QM1**, **QM2**, and **QM3** are running, and if not, start them.



2. Expand **Queue Manager Clusters** all the way to confirm that queue managers **QM1** and **QM2** have

full repositories, while **QM3** has a partial repository.



3. Note that in addition to the Cluster Receiver channels specified in the uniclus.mqsc file, Cluster Sender channels to the full repository queue manager have been created automatically. In the case of QM3, which has a partial repository, there are 2 of these channels.

Tip:

Deselect **Show System Objects** in MQ Explorer to see just the list of the channels you created.

Channel name	Channel type	Overall channel status
TO_UNIDEMO	Server-connection	Inactive
UNIDEMO_QM1	Cluster-receiver	Running
UNIDEMO_QM2	Cluster-sender	Running

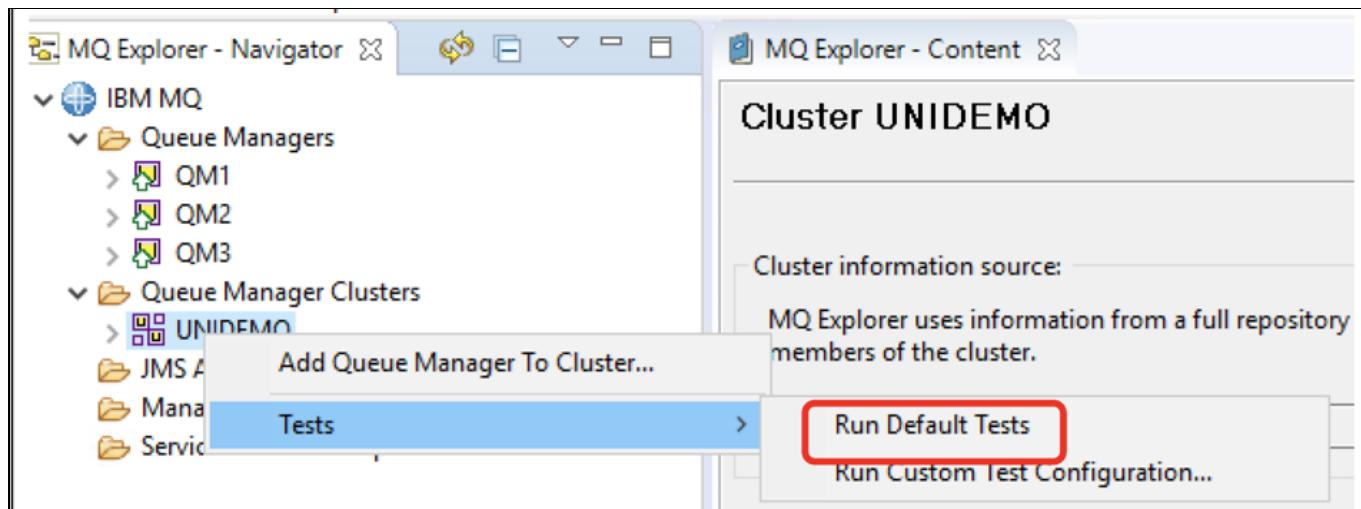
Channel name	Channel type	Overall channel status	Cor
TO_UNIDEMO	Server-connection	Inactive	
UNIDEMO_QM1	Cluster-sender	Running	127.
UNIDEMO_QM2	Cluster-sender	Running	127.
UNIDEMO_QM3	Cluster-receiver	Running	127.

4. Check that Cluster Sender and Receiver Channels for each queue manager are running and if not, start them.

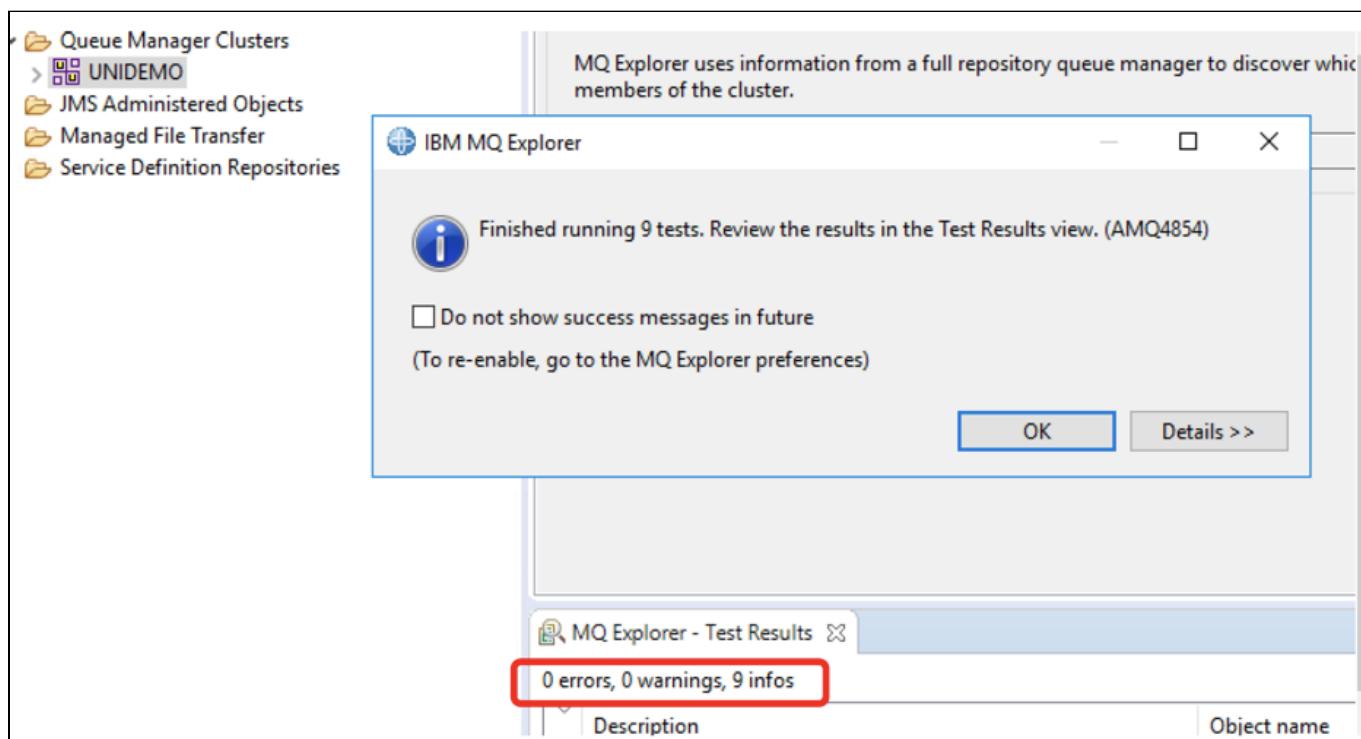
Channel name	Channel type	Overall channel status
TO_UNIDEMO	Server-connection	Inactive
UNIDEMO_QM1	Cluster-receiver	Running
UNIDEMO_QM2	Cluster-sender	Running

Note: The Server Connection Channels will be inactive – do not attempt to start these.

5. Run the default cluster tests, by clicking on **Queue Manager Clusters** -> **UNIDEMO**, then right-clicking to get the menu and clicking on **Tests** -> **Run Default Tests**.



6. When completed, click **OK** to dismiss the popup box. Check that there are no errors or warnings resulting, by looking at the **Test Results** in the lower panel.



Launch getting applications

In this section, we shall launch six instances of an application connected to the same queue manager.

The Client Channel Definition Table (CCDT) determines the channel definitions and authentication information used by client applications to connect to a queue manager. We shall be using a CCDT in JSON format. The first few lines are shown below. In the lab, the file path is: C:\Student\CCDT.json.

Note

- the channel names match those of the server connection channels on the queue managers
- the port is the listener port for the queue manager

```
{  
  "channel": [  
    {  
      "name": "TO_UNIDEMO",  
      "clientConnection": [  
        {  
          "connection": [  
            {  
              "host": "127.0.0.1",  
              "port": 3001  
            }  
          ],  
          "queueManager": "QM1"  
        },  
        {"type": "clientConnection"}  
      ]  
    }  
  ]  
}
```

1. Start a new Windows Command Prompt session and run the following commands:

a. *set MQCHLLIB=<JSON CCDT dir>* (sets the folder containing the JSON CCDT file)

In the lab environment:

```
set MQCHLLIB=C:\Student
```

b. *set MQCHLTAB=<JSON CCDT filename>* (sets the name of the JSON CCDT file)

In the lab environment:

```
set MQCHLTAB=CCDT.json
```

- c. *set MQAPPLNAME=<YOUR_GETTING_APPLICATION_NAME>* (sets the application name to a name of your choice, e.g.)

In the lab environment:

```
set MQAPPLNAME=MY.GETTER.APP
```

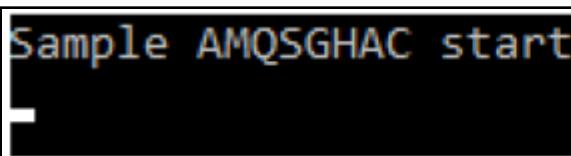
```
C:\Users\ibmdemo.DESKTOP-6DSOOH2>set MQCHLLIB=C:\Student  
C:\Users\ibmdemo.DESKTOP-6DSOOH2>set MQCHLTAB=CCDT.json  
C:\Users\ibmdemo.DESKTOP-6DSOOH2>set MQAPPLNAME=MY.GETTER.APP
```

2. Now run the following command to start the sample application to read messages from queue **QL** on **QM1**. Do this 6 times.

```
start amqsghac QL QM1
```

```
C:\Users\ibmdemo.DESKTOP-6DSOOH2>start amqsghac QL QM1  
C:\Users\ibmdemo.DESKTOP-6DSOOH2>start amqsghac QL QM1
```

This will open six new sessions, each like this:

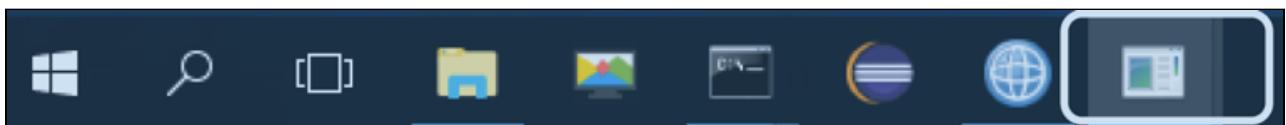


Note:

- As these sessions are created, they will appear above the session you are working on. Use this icon for the Windows Command Prompt session in the task bar to help find it again later:



- To locate these new amqsghac sessions later, use this icon in the task bar:



- There is a new MQSC command, *DISPLAY APSTATUS*, which we shall now use to display the status of an application across all queue managers in a cluster.

In a new Windows Command Prompt session, run the following as shown below:

DISPLAY APSTATUS(<YOUR_GETTING_APPLICATION_NAME>) TYPE(QMGR)

```
runmqsc QM1
```

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
```

```
end
```

At first, all instances of the application will be running on QM1 and none on the other two queue managers. However, by the time you run this command, the instances will probably be shared across all queue managers as shown below.

Note: COUNT is the number of instances of the specified application name currently running on this queue manager, while MOVCOUNT is the number of instances of the specified application name running on the queue manager which could be moved to another queue manager if required.

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
  1 : DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-01-14)
  LMSGTIME(12.47.16)               QMNAME(QM1)
  QMID(QM1_2020-01-14_10.35.05)    TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-01-14)
  LMSGTIME(12.46.54)               QMNAME(QM3)
  QMID(QM3_2020-01-14_10.35.50)    TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-01-14)
  LMSGTIME(12.46.47)               QMNAME(QM2)
  QMID(QM2_2020-01-14_10.35.36)    TYPE(QMGR)
```

4. Some of the application instances will show reconnection events as the workload is rebalanced to queue managers QM2 and QM3.

```
Sample AMQSGHAC start  
17:58:25 : EVENT : Connection Reconnecting (Reason: 2601, Qmgr: 'QM2  
: 174ms)  
17:58:25 : EVENT : Connection Reconnected
```

5. High level application summary information is also available via the MQSC command *DISPLAY APSTATUS* with *TYPE(APPL)*, instead of *TYPE(QMGR)*.

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(APPL)
```

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(APPL)  
2 : DISPLAY APSTATUS(MY.GETTER.APP) TYPE(APPL)  
AMQ8932I: Display application status details.  
APLNAME(MY.GETTER.APP) CLUSTER(UNIDEMO)  
COUNT(6) MOVCOUNT(6)  
BALANCED(YES) TYPE(APPL)
```

Note that COUNT reports the total number of application instances running in the cluster and that the applications are balanced across the cluster.

Launch putting application

We shall launch another sample application which will put messages to each queue manager in the cluster. The running samples should then pick up these messages and display them. In this lab, we are using one putting application to send messages to all getting applications using cluster workload balancing. You could set up the same scenario with one or more putting applications per queue manager and application rebalancing would work in the same way that you've seen for getting applications.

1. If you still have the Windows Command Prompt session where you issued the six “start amqsg hac” commands, you can use this and skip to step 3.
2. Open a new Windows Command Prompt session and run the following commands:
 - a. set *MQCHLLIB=<JSON CCDT dir>* (sets the folder of the JSON CCDT file)
In the lab environment:

```
set MQCHLLIB=C:\Student
```

- b. set *MQCHLTAB=<JSON CCDT filename>* (sets the name of the JSON CCDT file)

In the lab environment:

```
set MQCHLTAB=CCDT.json
```

- c. set *MQAPPLNAME=<YOUR_PUTTING_APPLICATION_NAME>* (sets the application name to a different name of your choice)

```
set MQAPPLNAME=MY.PUTTER.APP
```

3. We shall be using the sample *amqsputc* in this scenario. There are a few extra input parameters that are required for this sample to run in the way we want:

- The target queue the sample will run on (QL)
- The queue manager the target queue belongs to (QM1)
- A file containing messages that we aim to spread across all the queue managers in our cluster. (This has been provided for you in the classroom environment: *C:\Student\messages.txt*)

Enter the following command:

amqsputc QL QM1 <<PATH of messages file>

In the lab environment:

```
amqsputc QL QM1 < C:\Student\messages.txt
```

```
C:\Users\ibmdemo.DESKTOP-6DSOOH2>amqsputc QL QM1 < C:\Student\messages.txt
Sample AMQSPUT0 start
target queue is QL
Sample AMQSPUT0 end
```

```
C:\Users\ibmdemo.DESKTOP-6DSOOH2>
```

4. Looking at your six *amqsghac* windows, you should now see the generated messages split across the getting application sessions that are running. Each window will contain a subset, like this:

```
message <<Message 4>>
message <<Message 8>>
message <<Message 12>>
message <<Message 16>>
message <<Message 20>>
message <<Message 24>>
```

Note: The messages may not be evenly distributed across the getting applications instances.

Queue Manager maintenance

In this scenario, imagine a queue manager needs to be stopped for maintenance purposes. We shall demonstrate how doing this will cause the applications running on that queue manager to run instead on the remaining active queue managers in the Uniform Cluster. Once the maintenance is complete, the queue manager will be re-enabled.

1. When a queue manager is ended, the applications on that queue manager are usually lost. However, if the optional parameter **-r** is used, the applications will attempt to reconnect to a different queue manager.

Stop **QM3** by running this command from the Windows command line:

```
endmqm -r QM3
```

2. The applications connected to QM3 will reconnect back to QM1. After a while, an application imbalance will be detected, and affected applications will be reconnected to the other available queue managers.

To see this happening, re-run the MQSC command DISPLAY APSTATUS on any active queue manager in the cluster. After a minute or two you should see all application instances now running on QM1 and QM2:

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
 4 : DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(3)                         MOVCOUNT(3)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(16.15.36)               QMNAME(QM2)
  QMID(QM2_2020-07-29_14.22.41)   TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(3)                         MOVCOUNT(3)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(16.15.28)               QMNAME(QM1)
  QMID(QM1_2020-07-29_14.20.51)   TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(NO)
  COUNT(0)                          MOVCOUNT(0)
  BALSTATE(NOTAPPLIC)             LMSGDATE(2020-07-29)
  LMSGTIME(16.12.17)               QMNAME(QM3)
  QMID(QM3_2020-07-29_14.23.05)   TYPE(QMGR)
```

3. Once you are happy that the applications have balanced out equally across the three other queue managers, re-start the stopped queue manager using the command:

```
strmqm QM3
```

```
C:\Users\ibmdemo.DESKTOP-6DSOOH2>strmqm QM3
Failed to access automatic configuration INI definitions so using previous values
Failed to access automatic configuration MQSC definitions so using previous values
IBM MQ queue manager 'QM3' starting.
The queue manager is associated with installation 'Installation1'.
7 log records accessed on queue manager 'QM3' during the log replay phase.
Log replay for queue manager 'QM3' complete.
Transaction manager state recovered for queue manager 'QM3'.
IBM MQ queue manager 'QM3' started using V9.2.0.0.
```

Note: Because this queue manager is not mentioned in the unclust.ini file, we see the messages above.

4. As this queue manager has started and has no applications connected, it will request some from the other queue managers in the cluster.

Re-run the MQSC command **DISPLAY APSTATUS** to see the applications being rebalanced once more.

Metrics (new in 9.1.5)

The *amqsrua* sample application provides a way to consume MQ monitoring publications and display performance data published by queue managers. This data can include information about the CPU, memory, and disk usage. MQ v9.1.5 adds the ability to allow you to monitor usage statistics for each application you specify by adding the STATAPP class to the *amqsrua* command. You can use this information to help you understand how your applications are being moved between queue managers and to identify any anomalies.

The data is published every 10 seconds and is reported while the command runs.

Statistics available are:

- Instance count: number of instances of the specified application name currently running on this queue manager. See also COUNT from MQSC APSTATUS that we saw earlier.
- Movable instance count: number of instances of the specified application name running on this queue manager which could be moved to another queue manager if required. See also MOVCOUNT from MQSC APSTATUS that we saw earlier.
- Instance shortfall count: how far short of the mean instance count for the uniform cluster that this queue manager's instance count is. This will be 0 if queue manager is not part of a uniform cluster.
- Instances started: number of new instances of the specified application name that have started on this queue manager in the last monitoring period (these may have previously moved from other queue managers or be completely new instances).
- Initiated outbound Instance moves: number of movable instances of the specified application that have been requested to move to another queue manager in the last monitoring period. This will be 0

if the queue manager is not part of a uniform cluster.

- Completed outbound instance moves: number of instances of the specified application that have ended following a request to move to another queue manager. This number includes those that are actioning the requested move, or that are ending for any other reason after being requested to move (note that it does not mean that the instances have successfully started on another queue manager). This will be 0 if the queue manager is not part of a uniform cluster.
 - Instances ended during reconnect: number of instances of the specified application that have ended while in the middle of reconnecting to this queue manager (whether as a result of a move request from another queue manager, or as part of an HA fail over).
 - Instances ended: number of instances of the specified application that have ended in the last monitoring period. This includes instances that have moved, and those that have failed during reconnection processing.
1. In a new Windows Command Prompt session, run the **amqsrua** command as follows, i.e. with a class of STATAPP, a type of INSTANCE, and object of your getting application name.

```
amqsrua -m QM1 -c STATAPP -t INSTANCE -o MY.GETTER.APP
```

(Note: You can omit the class, type and object parameters and enter them when prompted instead.)

Initial stats are displayed and then updated every 10 seconds to show activity in the previous interval. You should see an Instance Count and Movable Instance Count of 2 as shown below. You may see different numbers for the other stats in the first interval, but these should be 0 in subsequent intervals.

```
C:\Users\ibmdemo.DESKTOP-6D5OOH2>amqsrua -m QM1 -c STATAPP -t INSTANCE -o MY.GETTER.APP
Publication received PutDate:20200729 PutTime:20223137 Interval:26 minutes,4.217 seconds
MY.GETTER.APP                                         Instance count 2
MY.GETTER.APP                                         Movable instance count 2
MY.GETTER.APP                                         Instance shortfall count 0
MY.GETTER.APP                                         Instances started 9
MY.GETTER.APP                                         Initiated outbound instance moves 6
MY.GETTER.APP                                         Completed outbound instance moves 6
MY.GETTER.APP                                         Instances ended during reconnect 0
MY.GETTER.APP                                         Instances ended 7

Publication received PutDate:20200729 PutTime:20224138 Interval:10.014 seconds
MY.GETTER.APP                                         Instance count 2
MY.GETTER.APP                                         Movable instance count 2
MY.GETTER.APP                                         Instance shortfall count 0
MY.GETTER.APP                                         Instances started 0
MY.GETTER.APP                                         Initiated outbound instance moves 0
MY.GETTER.APP                                         Completed outbound instance moves 0
MY.GETTER.APP                                         Instances ended during reconnect 0
MY.GETTER.APP                                         Instances ended 0
```

Refer to the description of these stats at the start of this section. Keep this command running.

2. In another Window, stop **QM3** once more by running this command from the Windows command line:

```
endmqm -r QM3
```

3. Refer back to the window with the running **amqsrua** session. When the next update is shown, the following should have changed:

Instance Count & Movable Instance Count

There are now 3 instances of the application running on this queue manager;

Initiated & Completed Outbound Instance Moves, Instances ended

Temporarily equal to 1 during the first interval as an instance is moved from QM3 to this queue manager.

```

Publication received PutDate:20200428 PutTime:12131290 Interval:10.012 seconds
MY.GETTER.APP                                         Instance count 3
MY.GETTER.APP                                         Movable instance count 3
MY.GETTER.APP                                         Instance shortfall count 0
MY.GETTER.APP                                         Instances started 0
MY.GETTER.APP                                         Initiated outbound instance moves 1
MY.GETTER.APP                                         Completed outbound instance moves 1
MY.GETTER.APP                                         Instances ended during reconnect 0
MY.GETTER.APP                                         Instances ended 1

Publication received PutDate:20200428 PutTime:12132290 Interval:10.001 seconds
MY.GETTER.APP                                         Instance count 3
MY.GETTER.APP                                         Movable instance count 3
MY.GETTER.APP                                         Instance shortfall count 0
MY.GETTER.APP                                         Instances started 0
MY.GETTER.APP                                         Initiated outbound instance moves 0
MY.GETTER.APP                                         Completed outbound instance moves 0
MY.GETTER.APP                                         Instances ended during reconnect 0
MY.GETTER.APP                                         Instances ended 0

```

4. In the other Window, restart QM3.

```
strmqm QM3
```

5. Again, refer back to the **amqsrua** session. When the next update is shown, the stats will have changed again. There are now 2 instances running on this queue manager, one having been moved (back) to QM3. As this happens, the numbers of moved and ended instances are again temporarily equal to 1.

```

Publication received PutDate:20200428 PutTime:12152294 Interval:10.007 seconds
MY.GETTER.APP                                         Instance count 3
MY.GETTER.APP                                         Movable instance count 3
MY.GETTER.APP                                         Instance shortfall count 0
MY.GETTER.APP                                         Instances started 0
MY.GETTER.APP                                         Initiated outbound instance moves 0
MY.GETTER.APP                                         Completed outbound instance moves 0
MY.GETTER.APP                                         Instances ended during reconnect 0
MY.GETTER.APP                                         Instances ended 0

Publication received PutDate:20200428 PutTime:12153294 Interval:10.002 seconds
MY.GETTER.APP                                         Instance count 2
MY.GETTER.APP                                         Movable instance count 2
MY.GETTER.APP                                         Instance shortfall count 0
MY.GETTER.APP                                         Instances started 0
MY.GETTER.APP                                         Initiated outbound instance moves 1
MY.GETTER.APP                                         Completed outbound instance moves 1
MY.GETTER.APP                                         Instances ended during reconnect 0
MY.GETTER.APP                                         Instances ended 1

```

6. Stop the **amqsrua** session when you are ready, using **ctrl-c**.

Using CCDT Queue Manager Groups

So far, we have connected our getting applications to QM1 directly, and relied on the Uniform Cluster to rebalance them across the other queue managers over a period of time. There are two disadvantages to connecting in this way:

- When the applications initially connect, they all start out connected to QM1 and there is a delay in the Uniform Cluster balancing them across the other queue managers
- If QM1 is stopped unexpectedly or for maintenance, any applications connected to it will try to reconnect to QM1 and fail. They will not attempt to connect to the other queue managers in the cluster. This will also be true if applications connected to other queue managers try to reconnect after an outage.

In this section, we shall see that by using Queue Manager Groups within our CCDT file, we can decouple application instances from a particular queue manager and take advantage of the built-in load balancing capabilities available with CCDTs.

For a fuller description of the issues highlighted here, see step 5 of the following article:

Walkthrough Uniform Cluster ↗

Stop queue manager - application refers to queue manager directly

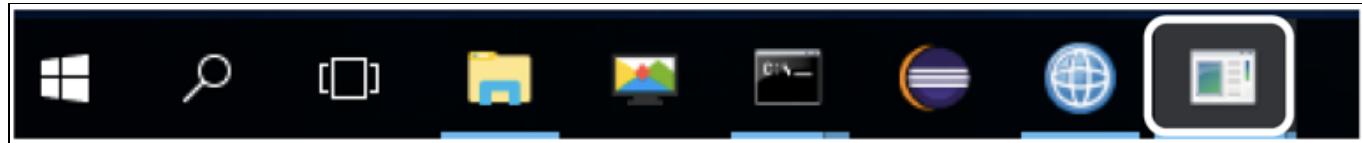
1. Stop **QM1** by entering the following from the command line.

```
endmqm -r QM1
```

2. This time, unlike the case where you stopped QM3, the getting application instances connected to QM1 will continually try to reconnect to the stopped queue manager.

```
10:19:07 : EVENT : Connection Reconnecting (Delay: 138ms)
10:19:07 : EVENT : Connection Reconnecting (Delay: 1295ms)
10:19:08 : EVENT : Connection Reconnecting (Delay: 2490ms)
10:19:11 : EVENT : Connection Reconnecting (Delay: 4498ms)
```

Note: These are the sessions you started earlier by running “start amqshac” and can be found here:



- Now run the following MQSC command on any active queue manager in the cluster.

```
DISPLAY APSTATUS(<YOUR_GETTING_APPLICATION_NAME>) TYPE(APPL)
```

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(APPL)
```

After a while (yes it will take a little while), there should be fewer than the six application instances that were originally present.

```
AMQ8932I: Display application status details.  
APPLNAME(MY.GETTER.APP) COUNT(5) BALANCED(YES) CLUSTER(UNIDEMO)  
MOVCOUNT(5) TYPE(APPL)
```

- Now restart QM1.

```
strmqm QM1
```

Stop queue manager – application refers to CCDT Queue Manager Group

- In the lab environment, an updated CCDT file has been created for you to use: C:\Student\CCDT3.json.

Browse this file. As well as containing the original set of direct references to the queue managers, it gives a queue manager group definition with a route to all queue managers using the name ANY_QM.

```
{  
  "channel": [  
    {  
      "name": "TO_UNIDEMO",  
      "clientConnection": {  
        "connection": [  
          {  
            "host": "127.0.0.1",  
            "port": 3001  
          }  
        ],  
        "queueManager": "ANY_QM"  
      },  
      ...  
    },  
    ...  
  ]  
}
```

2. In the lab environment, scroll down the file and note two new attributes.

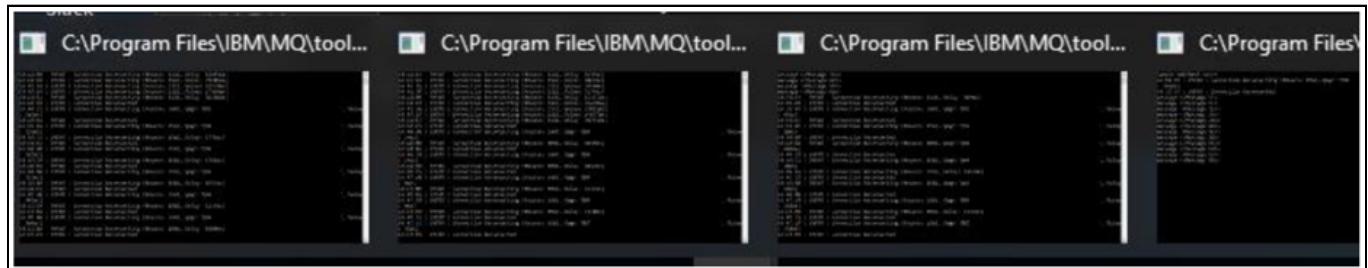
```
  ...  
  "queueManager": "ANY_QM"  
,  
  "connectionManagement": {  
    "clientWeight": 1,  
    "affinity": "none"  
  },  
  "type": "clientConnection"  
},  
...
```

These are defined under *connectionManagement*:

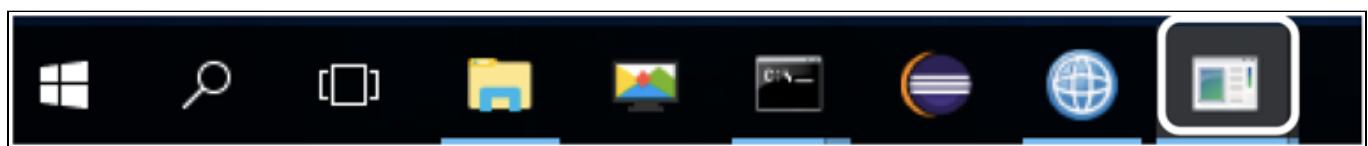
- clientWeight: a priority list for each client. The default value is zero. A client with a higher clientWeight will be picked over a client with a smaller value.

- affinity: setting the affinity to “none” will build up an ordered list of group connections to attempt to try in a random order, for any clients on a particular named host.

3. Now let's put the updated CCDT to the test. First, stop the six running getting application instances that you started earlier...



Remember, you can find them here:



4. Please note, the supplied updated CCDT file was originally created for a scenario with an additional queue manager called **QM4**. For completeness, we shall create that missing queue manager now.

In a new Windows Command Prompt session, create and start queue manager **QM4**.

```
cd C:\Student
```

```
crtmqm -p 3004 -ii .\uniclus.ini -ic .\uniclus.mqsc -iv CONNAME=127.0.0.1(300  
4) QM4
```

```
strmqm QM4
```

Like QM3, it will have a partial repository.

5. In a new Windows Command Prompt, run the following commands:

- a. `set MQCHLLIB=<JSON CCDT dir>`

(sets the folder containing the new JSON CCDT file)

In the lab environment:

```
set MQCHLLIB=C:\Student
```

- b. `set MQCHLTAB=<JSON CCDT filename>`

(sets the name of the new JSON CCDT file)

In the lab environment:

```
set MQCHLTAB=CCDT3.json
```

- c. `set MQAPPLNAME=<YOUR_GETTING_APPLICATION_NAME>`

(sets the application name to a name of your choice)

```
set MQAPPLNAME=MY.GETTER.APP
```

```
C:\Users\ibmdemo.DESKTOP-6DSOOH2>set MQCHLLIB=C:\Student  
C:\Users\ibmdemo.DESKTOP-6DSOOH2>set MQCHLTAB=CCDT3.json  
C:\Users\ibmdemo.DESKTOP-6DSOOH2>set MQAPPLNAME=MY.GETTER.APP
```

6. Now run the application instances, but this time specify the queue manager group **ANY_QM**, prefixed with * which tells the client to connect to any queue manager in the ANY_QM group.

```
start amqsghac QL *ANY_QM
```

Again, you will need to run this command 6 times.

```
C:\Users\ibmdemo.DESKTOP-6DSOOH2>start amqsghac QL *ANY_QM
```

7. The application instances will now attempt to connect to any of the queue managers defined in the queue manager group, and with the client weight and affinity options defined above, we should see each application instance connect to one of the queue managers in the Queue Manager Group and Uniform Cluster.

You can confirm this by running the MQSC command **DISPLAY APSTATUS** on any active queue manager in the cluster.

```

DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
 9 : DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(1)                         MOVCOUNT(1)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(19.38.33)               QMNAME(QM2)
  QMID(QM2_2020-07-29_14.22.41)    TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(19.38.33)               QMNAME(QM1)
  QMID(QM1_2020-07-29_14.20.51)    TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(19.38.32)               QMNAME(QM3)
  QMID(QM3_2020-07-29_14.23.05)    TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(1)                         MOVCOUNT(1)
  BALSTATE(LOW)                    LMSGDATE(2020-07-29)
  LMSGTIME(19.38.26)               QMNAME(QM4)
  QMID(QM4_2020-07-29_19.09.17)    TYPE(QMGR)

```

Note: The distribution might be different for you.

- Now end **QM1** to force the applications to be rebalanced.

```
endmqm -r QM1
```

Rather than the applications getting stuck in a reconnect loop trying to connect to QM1 as we saw using the previous version of the CCDT file, the applications now tied to the queue manager group ANY_QM will go through each of the definitions of ANY_QM and when able to successfully connect to one of the underlying queue managers, will do so. You should see this reported in a subset of the application instances:

```

Sample AMQSGHAC start
19:15:41 : EVENT : Connection Reconnecting (Reason: 2601, Qmgr:'QM1
: 90ms)
19:15:42 : EVENT : Connection Reconnected
19:19:06 : EVENT : Connection Reconnecting (Reason: 2161, Delay: 217ms)
19:19:06 : EVENT : Connection Reconnected

```

9. Run the MQSC command **DISPLAY APSTATUS** on any active queue manager in the cluster with **TYPE(APPL)**. After a while, there should once more be 6 connections in total, as there were before QM1 was shut down.

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(APPL)
  5 : DISPLAY APSTATUS(MY.GETTER.APP) TYPE(APPL)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)           CLUSTER(UNIDEMO)
  COUNT(6)                         MOVCOUNT(6)
  BALANCED(YES)                   TYPE(APPL)
```

10. Now run the MQSC command **DISPLAY APSTATUS** on any active queue manager in the cluster with **TYPE(QMGR)**. You will see that with QM1 stopped and with three queue managers running, the load has balanced again to have a **COUNT** of **2** on each of the running queue managers.

```
DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
  4 : DISPLAY APSTATUS(MY.GETTER.APP) TYPE(QMGR)
AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(19.31.22)               QMNAME(QM2)
  QMID(QM2_2020-07-29_14.22.41)    TYPE(QMGR)

AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(NO)
  COUNT(0)                          MOVCOUNT(0)
  BALSTATE(NOTAPPPLIC)            LMSGDATE(2020-07-29)
  LMSGTIME(19.18.47)              QMNAME(QM1)
  QMID(QM1_2020-07-29_14.20.51)    TYPE(QMGR)

AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(19.30.49)               QMNAME(QM3)
  QMID(QM3_2020-07-29_14.23.05)    TYPE(QMGR)

AMQ8932I: Display application status details.
  APPLNAME(MY.GETTER.APP)          ACTIVE(YES)
  COUNT(2)                         MOVCOUNT(2)
  BALSTATE(OK)                     LMSGDATE(2020-07-29)
  LMSGTIME(19.30.54)               QMNAME(QM4)
  QMID(QM4_2020-07-29_19.09.17)    TYPE(QMGR)
```

Congratulations

You have completed this Uniform Clusters and Application Rebalancing lab!

©2020 IBM. All rights reserved.

Site last generated: Jul 30, 2020

