Exercise instructions for a uniform cluster

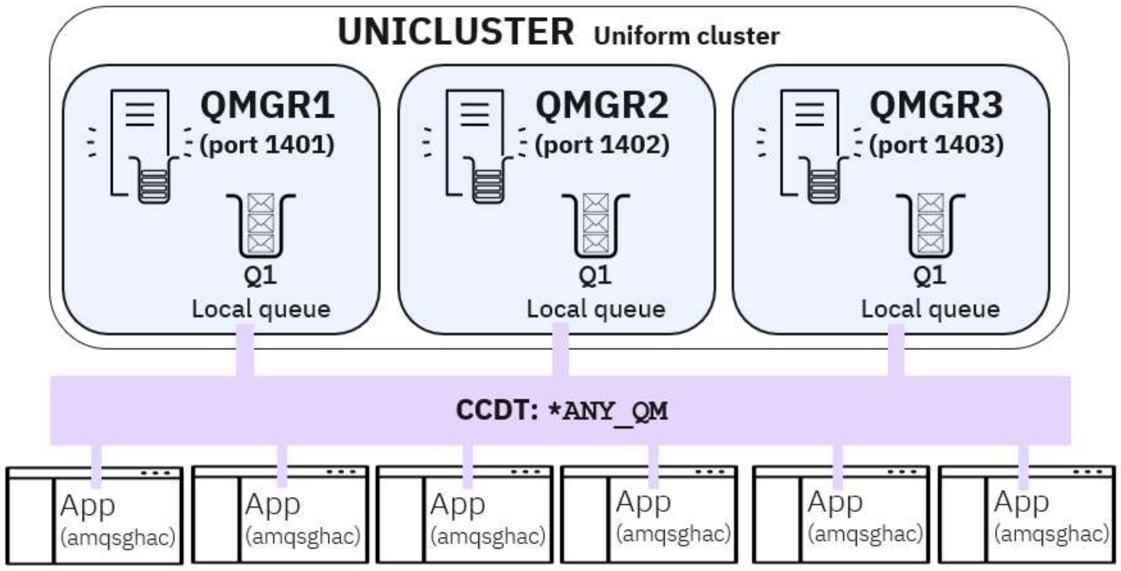
In the second half of this exercise, you create a uniform cluster named UNICLUSTER with three queue managers: QMGR1, QMGR2, and QMGR3. Each queue manager has a local cluster queue named Q1.

You work with a Client Channel Definition Table CCDT in JSON format to define a connection list. The connection list defines a single client connection that contains the connection information of all the queue managers in the uniform cluster. The simulated applications then simply need to specify that it wants to use this list, which will ensure it connects to any available queue manager in the uniform cluster.

You then use the amqsghac sample program to simulate multiple client applications connecting simultaneously to the cluster.

As the exercise progresses, you make changes to your cluster, like stopping a queue manager and adding a fourth queue manager. As you make these changes, you observe that the simulated applications never lose the ability to connect to the cluster queue. You also see that the connections from the applications are automatically load-balanced across the queue managers.

Figure 10. A diagram showing a uniform cluster with four queue managers



Section 1. Create the queue managers and the cluster

In this section, you use two text files to create the queue managers and the cluster:

- `AutoCluster.ini`: This file contains settings to be automatically applied on every queue manager restart. You use this file to override to the default qm.ini file.
- `UniCluster.mqsc`: This file contains MQSC commands to be automatically applied to the queue manager on every queue manager restart.

__ 1. Inspect the `AutoCluster.ini` file to learn more about it. Do not make any changes to the file.

__ a. In the terminal, run the following command to display the contents of the `AutoCluster.ini` file.

```
gedit /labfiles/Lab05/AutoCluster.ini &
```

__ b. Find following section at the end of the file.

```
...output omitted..
AutoCluster:
   Type=Uniform
   Repository1Name=QMGR1
   Repository1Conname=127.0.0.1(1401)
   Repository2Name=QMGR2
   Repository2Conname=127.0.0.1(1402)
   ClusterName=UNICLUSTER
```

Notice the `AutoCluster` stanza. This stanza is used when the queue manager starts to identify if the cluster is a member of an automatic cluster and to identify the full repositories of the cluster.

The stanza in this example defines a uniform cluster named UNICLUSTER. QMGR1 and QMGR2 will be full repositories in the cluster.

__ c. Close the file without making or saving any changes.

__ 2. Inspect the `UniCluster.mqsc` file. Do not make any changes to the file. Do not run any of the commands in the file.

__ a. In the terminal, run the following command to display the contents of the `UniCluster.mqsc` file.

```
gedit /labfiles/Lab05/UniCluster.mqsc &
```

__ b. Find following two MQSC commands.

```
ALTER QMGR CHLAUTH(DISABLED) CONNAUTH(' ')
REFRESH SECURITY TYPE(CONNAUTH)
```

These commands disable the use of channel authentication and connection authentication. You typically would not use these commands on a real-world queue manager, but you run them now to eliminate any security errors and security configuration in this exercise.

__ c. Look at the next MQSC command in the file.

```
DEFINE CHANNEL(UNICLUSTER.+QMNAME+) CHLTYPE(CLUSRCVR) CLUSTER(+AUTOCL+)
CONNAME(+CONNAME+) SHORTRTY(120) SHORTTMR(5)
```

This command defines a cluster-receiver channel and joins the queue manager to the cluster.

Notice the `+QMNAME+` string in the command. This string is a substitute token, which is denoted by the `+` characters at the beginning and the end of the string. This token represents the name of the queue manager. Tokens like this are replaced with their expanded text when the MQSC command is run. You can see two other substitute tokens in the command: `+AUTOCL+` and `+CONNAME+`.

__ d. Go to the next command:

```
DEFINE CHANNEL(SVRCONN.CHANNEL) CHLTYPE(SVRCONN)
```

This command defines a server-connection channel named `SVRCONN.CHANNEL` to accept client connections.

__ e. Look at the last command in the file.

```
DEFINE QLOCAL(Q1) CLUSTER(UNICLUSTER) DEFPSIST(YES) DEFBIND(NOTFIXED)
```

This command defines a local queue named `Q1` that is part of the cluster named `UNICLUSTER`. Also note that the default bind type is `NOTFIXED`.

__ f. Close the file without making or saving any changes.

__ 3. Use the AutoCluster.ini and UniCluster.mqsc files to create three queue managers:

- QMGR1, with a listener on port 1401
- QMGR2, with a listener on port 1402
- QMGR3, with a listener on port 1403

__ a. Change to the `/labfiles/Lab05` directory.

```
cd /labfiles/Lab05
```

Important

For the rest of this exercise, run all commands from the `/labfiles/Lab05` directory.

End of Important section.

__ b. Run the following three commands, one at a time, to create the queue managers and the cluster.

```
crtmqm -ii AutoCluster.ini -ic UniCluster.mqsc -iv CONNAME="localhost(1401)" -p 1401 QMGR1

crtmqm -ii AutoCluster.ini -ic UniCluster.mqsc -iv CONNAME="localhost(1402)" -p 1402 QMGR2

crtmqm -ii AutoCluster.ini -ic UniCluster.mqsc -iv CONNAME="localhost(1403)" -p 1403 QMGR3
```

__ 4. Start all three queue managers.

At this point, your cluster is ready to use.

Section 2. Connecting clients

In this section, you use a CCDT file in JSON format to define a connection list. You then start multiple copies of the amqsghac sample program to simulate multiple client applications connecting simultaneously to the cluster.

__ 1. Inspect the CCDT file named `CCDT3.JSON` to learn more about its contents. Do not make any changes to the file.

__ a. Run the following command to open the file in a text editor.

```
gedit /labfiles/Lab05/CCDT3.JSON &

Example output:

{
    "channel": [{
        "name": "SVRCONN.CHANNEL",
        "clientConnection": {
          "connection": [{
            "host": "localhost",
            "port": 1401
          },{
            "host": "localhost",
            "port": 1402
          },{
            "host": "localhost",
            "port": 1403
            }],
          "queueManager": "ANY_QM"
        },
        "type": "clientConnection"
        },
        {
            "name": "SVRCONN.CHANNEL",
            "clientConnection":
            {
                "connection":
                [
                    {
                        "host": "localhost",
                        "port": 1401
                    }
                ],
                "queueManager": "QMGR1"
            },
...output omitted...
```

Notice the file contains connection information about each of the three queue managers: QMGR1, QMGR2, and QMGR3.

It also contains information about a queue manager named `ANY_QM`. The queue manager ANY_QM it not really a queue manager, it is a specification. ANY_MQ specifies three connections to represent each of the three queue managers in the uniform cluster. You can choose anything as the queue manager name, and it would work in the same way; ANY_QM was chosen for readability.

__ b. Close the file without making or saving any changes.

__ 2. Copy the named `CCDT3.JSON` and name the copy `CCDT.JSON`.

```
cp CCDT3.JSON CCDT.JSON
```

__ 3. Before you test your cluster with the amqsghac sample program, find the number of client applications that have opened the Q1 queue for input (`IPPROCS`). Find this number for each of the queue managers. Run the following command as a single block.

```
echo "DIS QL(Q1)" | runmqsc QMGR1 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR2 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR3 | grep IPPROCS

Example output:

   INITQ( )                                IPPROCS(0)
   INITQ( )                                IPPROCS(0)
   INITQ( )                                IPPROCS(0)
```

In this example, the Q1 queue has zero connections on QMGR1, QMGR2, and QMGR3 (`IPPROCS(0)`). These numbers will increase after you start connecting clients in the next steps.

__ 4. Use the rClient.sh script to start 40 instances of the amqsghac sample program to test your cluster.

__ a. Open new terminal. You use this this new terminal window to run the sample client programs.

__ b. In the new terminal, change to the `/labfiles/Lab05` directory.

```
cd /labfiles/Lab05
```

__ c. Set the value of the `MQCHLTAB` environment variable to `CCDT.JSON`.

```
export MQCHLTAB=CCDT.JSON
```

__ d. Set the value of the `MQCHLLIB` environment variable to `/labfiles/Lab05`.

```
export MQCHLLIB=/labfiles/Lab05
```

__ e. Confirm that the MQCHLTAB and MQCHLLIB environment variables are set correctly. Run the following command and verify that your output matches the example.

```
env | grep MQCHL*

Example output:

MQCHLTAB=CCDT.JSON
MQCHLLIB=/labfiles/Lab05
```

__ f. Run the following command to start the rClient.sh script. After you run the command, the script takes control of the terminal.

```
./rClient.sh 40
```

__ g. Leave the script running for the rest of the exercise. Minimize the terminal window so you do not accidently interact with the script and cause it to end early.

Attention

The rClient.sh script was copied to your lab environment to make this exercise easier. This script is not included with IBM MQ and it is not supported by IBM.

The rClient.sh script starts multiple instances of the amqsghac sample program with the command:

```
$MQ_INSTALLATION_PATH/samp/bin/amqsghac Q1 *ANY_QM &
```

End of Attention section.

__ 5. Check the number of client applications that have opened the Q1 queue for input (IPPROCS) again. Find this number for each of the queue managers.

   __ a. Return to your original terminal.

   __ b. Run the following command as a single block.
   You can also use the up arrow on your keyboard to find this command in your history.

```
echo "DIS QL(Q1)" | runmqsc QMGR1 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR2 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR3 | grep IPPROCS

Example output:

INITQ( )                                IPPROCS(14)
INITQ( )                                IPPROCS(13)
INITQ( )                                IPPROCS(13)
```

The output should show that the 40 client connections are evenly distributed across all three queue managers in the cluster.

Initially, all of the connections go to a single queue manager, but after 5-10 seconds, the cluster automatically balances them over all of the queue managers. You might need to run the command more than once to see the connections settle into an even distribution.

This confirms that your uniform cluster is working as expected.

Section 3. Queue manager maintenance scenario

In this section, you work through a scenario where one of the queue managers must be stopped for maintenance. With a uniform cluster, you can remove a queue manager without disruption to your connected applications. In these steps, you stop QMGR3 and watch the client connections seamlessly rebalance across the remaining two queue managers.

__ 1. Run the following command to stop QMGR3.

```
endmqm -r QMGR3
```

The `-r` option is used to try to reconnect reconnectable clients. This option has the effect of reestablishing the connectivity of clients to other queue managers in their queue manager group.

__ 2. Again, view the IPPROCS count on all three queue managers.

```
echo "DIS QL(Q1)" | runmqsc QMGR1 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR2 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR3 | grep IPPROCS

Example output:

INITQ( )                                  IPPROCS(22)
INITQ( )                                  IPPROCS(18)
```

You now see only two lines of output because QMGR3 is stopped and cannot respond to the MQSC command. However, the 40 client connections are now balanced across QMGR1 and QMGR2.

__ 3. Start QMGR3.

```
strmqm QMGR3
```

__ 4. Confirm that the 40 connections have rebalanced across all three queue managers in the cluster.

```
echo "DIS QL(Q1)" | runmqsc QMGR1 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR2 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR3 | grep IPPROCS

Example output:

   INITQ( )                                  IPPROCS(14)
   INITQ( )                                  IPPROCS(14)
   INITQ( )                                  IPPROCS(12)
```

Section 4. Cluster scaling scenario

This section gives you practice with another scenario where uniform clusters are a good fit: scaling up to accommodate more application traffic. In these steps, you add a fourth queue manager to the cluster and observe the results.

__ 1. Confirm you are in the `/labfiles/Lab05` directory.

```
pwd
Example output:
/labfiles/Lab05
```

If your current directory is not `/labfiles/Lab05`, then change to that directory before you continue.

__ 2. Run the following command to create a fourth queue manager: QMGR4.

```
crtmqm -ii AutoCluster.ini -ic UniCluster.mqsc -iv CONNAME="localhost(1404)" -p 1404 QMGR4
```

__ 3. Copy the named `CCDT4.JSON` and name the copy `CCDT.JSON`.

```
cp CCDT4.JSON CCDT.JSON
```

The `CCDT4.JSON` file is similar to the `CCDT3.JSON` file, except `CCDT4.JSON` contains additional connection information for QMRG4. You are renaming the file to `CCDT.JSON` so that the client applications can use the new connection information without the need to set a new value for the `MQCHLTAB` environment variable.

__ 4. Start the new queue manager.

```
strmqm QMGR4
```

__ 5. Check the current IPPROCS count on all four queue managers.

```
echo "DIS QL(Q1)" | runmqsc QMGR1 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR2 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR3 | grep IPPROCS; \
echo "DIS QL(Q1)" | runmqsc QMGR4 | grep IPPROCS

Example output:

   INITQ( )                                IPPROCS(10)
   INITQ( )                                IPPROCS(10)
   INITQ( )                                IPPROCS(10)
   INITQ( )                                IPPROCS(10)
```

You should see that the 40 client connections have been spread across all four queue managers. It might take 5-10 seconds for the connections to rebalance, so you might need to run the command more than once to see the expected result.

Section 5. Exercise clean up

In this section, you prepare your lab environment for the next exercise.

__ 1. Stop the rClient.sh script and all instances of the amqsghac sample program.

__ a. Return to the terminal window where the rClient.sh script is running.

__ b. Press the Enter key 2 or 3 times until you see the Linux shell prompt again.

```
Example output:

10:32:41 : EVENT : Connection Reconnected
10:32:41 : EVENT : Connection Reconnected
[localuser@rhserver Lab05]$
[localuser@rhserver Lab05]$
[localuser@rhserver Lab05]$
```

__ c. Run the follow command to stop all instances of the amqsghac sample program.

```
pkill amqsghac
```

__ d. Type `exit` to close the terminal window.

__ 2. In the remaining terminal, stop QMGR1, QMGR2, QMGR3 and QMGR4. Run the following commands, one at a time.

```
endmqm QMGR1
endmqm QMGR2
endmqm QMGR3
endmqm QMGR4
```

__ 3. Verify that the queue managers are stopped.

```
dspmq

Example output:

QMNAME(QMC1)                                              STATUS(Ended normally)
QMNAME(QMC2)                                              STATUS(Ended normally)
QMNAME(QMC3)                                              STATUS(Ended normally)
QMNAME(QMC4)                                              STATUS(Ended normally)
QMNAME(QMGR1)                                             STATUS(Ended normally)
QMNAME(QMGR2)                                             STATUS(Ended normally)
QMNAME(QMGR3)                                             STATUS(Ended normally)
QMNAME(QMGR4)                                             STATUS(Ended normally)
```

Confirm that all 8 queue managers have ended normally before you continue.

__ 4. Delete all the queue managers that you created in this exercise. Run the following commands, one at a time.

```
dltmqm QMC1
dltmqm QMC2
dltmqm QMC3
dltmqm QMC4
dltmqm QMGR1
dltmqm QMGR2
dltmqm QMGR3
dltmqm QMGR4
```

__ 5. Run the `dspmq` command again to verify that the queue mangers have been deleted.

```
dspmq
```

The output of the command should be empty.

__ 6. Type `exit` to close the terminal window.

__ 7. Close MQ Explorer.

Exercise review and wrap-up

You have completed the lab exercise. In this exercise you have implemented two types of clusters: a basic cluster and a uniform cluster.

In the first half of this exercise, you created a basic cluster of four queue managers. You then used cluster workload management techniques to control the distribution of messages across the queue managers in the cluster.

In the second half of this exercise, you created a uniform cluster. You then made changes to your cluster and observed how client application connections are automatically rebalanced.

End of exercise.