Using runmqsc and dmpmqcfg over TLS client

T.Rob Wyatt

All your applications and adjacent QMgrs connect using TLS.

Congratulations!

Is the control traffic still connecting over plaintext channels?

Uh-oh!

Let's fix that, Shall we?

Like a basic client connection but requires specification of TLS parameters.

Two styles:

1) Client Channel DefinitionTable – runmqsc

2) MQCONNX – dmpmqcfg

Wikipedia
Irony – Examples

Example #2035: IBM's dmpmqcfg utility accepts MQSC format commands to define a client channel whereas runmqsc, the native MQSC command interpreter does not and requires a CCDT file instead.

Note: This was briefly true, but only because Wikipedia lets anyone edit entries and quickly rolls back "vandalism".

Common Prereqs – KDB file

Both of the programs are C code and therefore use the KDB format keystore.

At a minimum, the KDB will require the QMgr's signer certificate or, if it uses a self-signed certificate, the public portion of the QMgr's personal certificate.

If the SVRCONN channel on the QMgr specifies SSLCAUTH(REQUIRED) then the client will require its own personal certificate. In this case the QMgr needs to validate the client's personal certificate and therefore will require the client's signer certificate or, if the client uses a self-signed certificate, the public portion of the client's personal certificate.

Whew! Got that? If not see the sessions from prior MQTC years on cert management and SSL configuration.

Common Prereqs – MQSSLKEYR

In both cases we will use the MQSSLKEYR environment variable to locate the client's KDB files stored in the same directory as the script.

The environment variable is used in part because it allows multiple client-side scripts to run simultaneously with different keystores.

If you wish to disable client-side OCSP or CRL checking, or both, you need to specify that in the mqclient.ini file so may not have the option to delete it. However, you can generate multiple mqclient.ini files and point to them using the MQCLNTCF environment variable.

In any case, make sure you know the search path for the mqclient.ini file and the presence of mqclient.ini files in that search path before trying to run these scripts.

CCDT - runmqsc

Requirements:

1. **Populated kdb file with certs and stashed password.**
   This is beyond the scope of this session. For more help, see previous years sessions on certificate management and TLS configuration.

2. **A modern runmqsc that can manipulate table files independent of a QMgr.**
   Available in MQ Advanced for Developers. Is this available in MQ Client?

CCDT - runmqsc

Approach: Build a table file on demand for each connection attempt

- **Eliminates namespace collisions on identically named channels**
- **File contents never go stale**
- **No ambiguity as to which CCDT in which location is being accessed. (Easier to troubleshoot.)**

CCDT - runmqsc

Setting up the environment

```
1: _SCRIPTDIR="$(cd "$(dirname "$0")"; pwd)"
2: cd "$_SCRIPTDIR"
3: export MQSSLKEYR="$_SCRIPTDIR/key"
4: export MQCHLLIB="$_SCRIPTDIR"
5: export MQCHLTAB="$_PROG.TAB"
```

1. Fairly reliable way to obtain script's directory
2. Make sure we are in that directory because we expect to find the KDB and channel table file there.
3. Locate KDB file set in the script's directory
4. Point CCDT path to script directory
5. Name the CCDT.
   ToDo: This should use mktmp to avoid a race condition on the file.

CCDT - runmqsc

Building the CCDT on demand

```
while read LINE
do
    # QMNAME,host1[:host2],port,SVRCONN.NAME,CIPHER|CIPHER(dup)|TLSversion
    QMGR=$(echo "$LINE" | cut -d ',' -f 1)
    HOST=$(echo "$LINE" | cut -d ',' -f 2)
    PORT=$(echo "$LINE" | cut -d ',' -f 3)
    CHL=$( echo "$LINE" | cut -d ',' -f 4)
    CIPH=$(echo "$LINE" | cut -d ',' -f 5 | cut -d '|' -f 1)

    if [[ "$QMGR" == "$REPO1" || "$QMGR" == "$REPO2" ]]; then
        echo "DEF CHANNEL($CHL) CHLTYPE(CLNTCONN) +
              CONNAME('${HOST//:/($PORT),}($PORT)') +
              SSLCIPH($CIPH) QMNAME($QMGR) REPLACE" | runmqsc -n > /dev/null 2>&1
```

Fetch these variables from whatever source you are using for QMgr connection info. In this case the same flat-file format as my session last year.

If it is a full repository, we want to process it. In that case, make the CCDT.

The substitution on the hostname builds a proper MIQM CONNAME.

CCDT - runmqsc

Finally! Running the commands

```
echo "DIS CLUSQMGR(*) CLUSTER($CLUSTER) ALL" | runmqsc -e -c $QMGR
echo "DIS QCLUSTER(*) CLUSTER($CLUSTER) ALL" | runmqsc -e -c $QMGR
```

The -e suppresses echo of the commands.
The -c says to use Client Mode.

The previously set variables locate the KDB file set and the newly minted CCDT file.

CCDT - runmqsc

Adverse interactions

We used environment variables to locate the keystore and channel table file.

The mqclient.ini file can contain the location of the KDB in the SSL stanza.

Make sure that you use one and only one of these methods and that you know which one it is!

MQ Knowledge Center: *Environment variables that were supported in releases of IBM WebSphere® MQ earlier than Version 7.0 continue to be supported, and where such an environment variable matches an equivalent value in the client configuration file, the environment variable overrides the client configuration file value*.

CCDT - runmqsc

Practical uses

The companion session *Automated cluster health monitoring* uses this technique to fetch cluster information from the full repositories and reconcile the clustered object entries.

If you have a central admin/monitoring tool that uses MQ Client connections, you can build MQSC client scripts using its keys and channel definitions. In this case the tool used was Avada's IR-360 but anything that can serve up basic QMgr connection details would work.

Session: *Automated cluster health monitoring*

Monday	**Aloeswood**	**15:45**
Wednesday	**Aloeswood**	**08:30**

MQCONNX – dmpmqcfg

Setting up the environment

```
1:  _SCRIPTDIR="$(cd "$(dirname "$0")"; pwd)"
2:  cd "$_SCRIPTDIR"
3:  export MQSSLKEYR="$ SCRIPTDIR/key"
```

1. Fairly reliable way to obtain script's directory
2. Go to that directory because we expect to find the KDB there.
3. Locate KDB file set in the script's directory

MQCONNX – dmpmqcfg

Running the backup

```
while read LINE
do
    # Parse the lines from the Connections Report. Lines are in format:
    # QMNAME,host1[:host2],port,SVRCONN.NAME,CIPHER|CIPHER(dup)|TLSversion
    QMGR=$(echo "$LINE" | cut -d ',' -f 1)
    HOST=$(echo "$LINE" | cut -d ',' -f 2)
    PORT=$(echo "$LINE" | cut -d ',' -f 3)
    CHL=$( echo "$LINE" | cut -d ',' -f 4)
    CIPH=$(echo "$LINE" | cut -d ',' -f 5 | cut -d '|' -f 1)

    dmpmqcfg -m $QMGR -a -o 1line -c  \
    "DEFINE CHANNEL(${CHL}) CHLTYPE(CLNTCONN) CONNAME('${HOST//:/($PORT),}($PORT)') SSLCIPH($CIPH)" \
    > $_BKUP/$_DATE.$QMGR.mqsc 2>&1
```

Fetch these variables from whatever source you are using for QMgr connection info. In this case the script fetches the *Connection Report* from Avada's IR-360 to obtain the QMgr connection parameters.

Use the variables to build a proper CLNTCONN definition for dmpmqcfg.

The substitution on the hostname builds a proper MIQM CONNAME.

MQCONNX – dmpmqcfg

Adverse interactions

We used environment variables to locate the keystore and channel table file.

The mqclient.ini file can contain the location of the KDB in the SSL stanza.

Make sure that you use one and only one of these methods and that you know which one it is!

MQ Knowledge Center: *Environment variables that were supported in releases of IBM WebSphere® MQ earlier than Version 7.0 continue to be supported, and where such an environment variable matches an equivalent value in the client configuration file, the environment variable overrides the client configuration file value*.

Questions & Answers

Tutorials and more

People kept asking where to find the slides, videos. Here ya go…

- **YouTube tutorials: https://www.youtube.com/tdotrob**
- **Twitter**
  - @deepqueue (MQ & security)
  - @tdotrob (MQ & security + politics, humor, autism)
- **LinkedIn: https://www.linkedin.com/in/tdotrob/**
- **Blogging on general IT, security, malvertising. How to hire me: https://ioptconsulting.com**
- **MQ web site and blog: https://t-rob.net (Slides are uploaded here)**

All my web sites are linked together in the nav bar. Go to Ask-An-Aspie for autism content, or The Odd is Silent for everything that's not autism or IT.

```
#!/bin/ksh
cd "$(cd "$(dirname "$0")"; pwd)"
#===============================================================================
# ~mqm/scripts/ir360-qmgrBackup.ksh
#
# Script to backup QMgrs known to IR-360
#
# 20180312 T.Rob - New script
# 20180618 T.Rob - Modified to use IR-360 Connection Report as source
#===============================================================================
_PROG=${0##*/}
_DATE=`date "+%y%m%d"`

_BKUP=~mqm/IR360/QMgrBkup
_SCRIPTDIR="$(cd "$(dirname "$0")"; pwd)"
cd "$_SCRIPTDIR"
export MQSSLKEYR="$_SCRIPTDIR/key"



# For testing, save backups in user's home directory instead
[[ $(whoami) != "mqm" ]] && _BKUP=~/QMgrBkup

# Make sure backup directory exists
mkdir -p $_BKUP >/dev/null 2>&1


# Fetch connections and iterate.
curl --cacert CA.pem -s -H "Accept:application/json" -u sa:sa https://ir360dev:9443/Infrared360/reports/ConnectionInventoryWmq-Report.csv | grep -v -e ',,,,,' -e ^Name | sort | sed 's/"\([^"]\+\),\([^"]\+\)"/\1:\2/g' | {
    while read LINE
    do
        # Parse the lines from the Connections Report. Lines are in format:
        # QMNAME,host1[:host2],port,SVRCONN.NAME,CIPHER|CIPHER(dup)|TLSversion
        QMGR=$(echo "$LINE" | cut -d ',' -f 1)
        HOST=$(echo "$LINE" | cut -d ',' -f 2)
        PORT=$(echo "$LINE" | cut -d ',' -f 3)
        CHL=$( echo "$LINE" | cut -d ',' -f 4)
        CIPH=$(echo "$LINE" | cut -d ',' -f 5 | cut -d '|' -f 1)

        dmpmqcfg -m $QMGR -a -o 1line -c  \
        "DEFINE CHANNEL(${CHL}) CHLTYPE(CLNTCONN) CONNAME('${HOST//:/($PORT),}($PORT)') SSLCIPH($CIPH)" \
        > $_BKUP/$_DATE.$QMGR.mqsc 2>&1

        # Get rid of backups older than 30 days
        find $_BKUP/*.$QMGR.mqsc -mtime +31 -exec rm {} \;
    done
}

exit
```

```
#!/bin/ksh
#===============================================================================
# [path to script goes here]
#
# Cluster health report
#
# 20180319 T.Rob - New script
#===============================================================================
_PROG=${0##*/}
_VERS="v1.2"                  # Please update the version number when updating the program!
_DATE=`date "+%Y%m%d"`
_RPTDIR=~mqm/IR360/Infrared360SA/webapps/Infrared360/exports

_SCRIPTDIR="$(cd "$(dirname "$0")"; pwd)"
cd "$_SCRIPTDIR"
export MQSSLKEYR="$_SCRIPTDIR/key"
export MQCHLLIB="$_SCRIPTDIR"
export MQCHLTAB="$_PROG.TAB"


# ---------------------------------------
# Comment out line below when not testing
_TESTING=1
_VERBOSE=0
# ---------------------------------------


# Establish associative arrays for cluster data
typeset -A CLUSQMGR
typeset -A QCLUSTER

# Exceptions found? Yes if not NULL.
_ERRFLAG=

# Set report directory to CWD for testing.
[[ $(whoami) != "mqm" ]] && _RPTDIR=.

# Cluster name must be passed
CLUSTER=$(echo "${1}" | tr -d -c "[a-zA-Z0-9_.]")
[[ $# -eq 0 ]] &&  print "$_PROG: FATAL! Please pass a cluster name.\n" && exit

# Set up temp file to catch console log output
_CONSOLE="$(mktemp -p $_RPTDIR --suffix=.txt $_DATE.$_PROG.XXXXXXXXXX)"
trap "rm -f $_CONSOLE" EXIT

# Set up report file name using current date and cluster name
_FILE=${_DATE}-$_PROG-$CLUSTER.html

# Delete file if it exists, in case noclobber option is set
rm -f $_RPTDIR/$_FILE
```

```
51
52
53
54  # ================================================================================
55  # Use cluster name passed to fetch repository names then check for success
56  # This uses a config file served from IR-360.  Last year we used an S3 bucket.
57  # Alternatively, pass the repository names or connect to a convenient QMGR
58  # and query the clusters and repositories it knows about.
59  #
60  # NOTE! This script assumes exactly two repositories for any cluster!  Any
61  # fewer and it breaks. More than two the results will be inconclusive.
62  # ================================================================================
63  REPO1=$(curl --cacert CA.pem -s -u sa:sa https://ir360dev:9443/Infrared360/cfg/Clusters.ini | grep $CLUSTER | grep -i -e ':PRI:' | sed 's/:.*//')
64  REPO2=$(curl --cacert CA.pem -s -u sa:sa https://ir360dev:9443/Infrared360/cfg/Clusters.ini | grep $CLUSTER | grep -i -e ':SEC:' | sed 's/:.*//')
65
66  [[ ${#REPO1} -eq 0 || ${#REPO2} -eq 0  ]] &&  print "$_PROG: FATAL! Fetch of repository names for cluster $CLUSTER failed.\nValues returned were
    REPO1='$REPO1' and REPO2='$REPO2'\n" && exit
67
68
69
70  # ================================================================================
71  # Validation complete. Notify user and proceed.
72  # ================================================================================
73  printf "$_PROG - Begin cluster reconciliation report for $CLUSTER at $(date)\nPrimary=$REPO1, Secondary=$REPO2\n\n" | tee -a $_CONSOLE
74  print "<!DOCTYPE html>
75  <html>
76  <head>
77  <meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\" />
78  <title>$CLUSTER Cluster Health</title>
79  <style type=\"text/css\">
80    h1 {font-family: Arial, Helvetica, sans-serif;font-weight: normal;text-align: center;vertical-align: middle; color: #000}
81    h2 {font-family: Arial, Helvetica, sans-serif;font-weight: normal;text-align: center;vertical-align: middle; color: #000}
82    td {vertical-align: top;font-family: Arial, Helvetica, sans-serif;}
83    caption {font-family: Arial, Helvetica, sans-serif;color: #006;}
84    th {font-family: Arial, Helvetica, sans-serif;font-weight: bold;}
85  </style>
86  </head>
87
88  <body>
89  <h1 align="center">$CLUSTER Cluster Health Report</h1>
90  <h2 align="center">Repositories $REPO1 and $REPO2<br>Report run at $(date)</h2>" > $_RPTDIR/$_FILE
91
92  # ================================================================================
93  # Gather CLUSQMGR and QCLUSTER records
94  # ================================================================================
95  echo IR-360 returned $(curl --cacert CA.pem -s -H "Accept:application/json" -u sa:sa
    https://ir360dev:9443/Infrared360/reports/ConnectionInventoryWmq-Report.csv | grep -v -e ',,,,,' -e ^Name | sed 's/"\([^"]\+\),\([^"]\+\)"/\1:\2/g' | wc
    -l) QMgrs.  | tee -a $_CONSOLE
96  _FOUND=0
97  curl --cacert CA.pem -s -H "Accept:application/json" -u sa:sa https://ir360dev:9443/Infrared360/reports/ConnectionInventoryWmq-Report.csv | grep -v -e
```

```
    ',,,,,' -e ^Name | sed 's/"\([^"]\+\),\([^"]\+\)"/\1:\2/g' | {
 98     while read LINE
 99     do
100         # QMNAME,host1[:host2],port,SVRCONN.NAME,CIPHER|CIPHER(dup)|TLSversion
101         QMGR=$(echo "$LINE" | cut -d ',' -f 1)
102         HOST=$(echo "$LINE" | cut -d ',' -f 2)
103         PORT=$(echo "$LINE" | cut -d ',' -f 3)
104         CHL=$( echo "$LINE" | cut -d ',' -f 4)
105         CIPH=$(echo "$LINE" | cut -d ',' -f 5 | cut -d '|' -f 1)
106
107         if [[ "$QMGR" == "$REPO1" || "$QMGR" == "$REPO2" ]]; then
108             echo "DEF CHANNEL($CHL) CHLTYPE(CLNTCONN) CONNAME('${HOST//:/($PORT),}($PORT)') SSLCIPH($CIPH) QMNAME($QMGR) REPLACE" | runmqsc -n >
                /dev/null 2>&1
109
110             # Fault tolerance: Iterate mqsc queries until success or 10 attempts
111             _ATTEMPTS=0
112             while [[ $_ATTEMPTS -lt 10 ]]
113             do
114                 printf "." | tee -a $_CONSOLE
115                 ((_ATTEMPTS=_ATTEMPTS+1 ))
116
117                 # Gather cluster info
118                 CLUSQMGR[$QMGR]="$(echo "DIS CLUSQMGR(*) CLUSTER($CLUSTER) ALL" | runmqsc -e -w 10 -c $QMGR | sed 's/ \+/ /g' | perl -ne 'chomp; print
                    "\n" unless /^ /; print;' )"
119                 QCLUSTER[$QMGR]="$(echo "DIS QCLUSTER(*) CLUSTER($CLUSTER) ALL" | runmqsc -e -w 10 -c $QMGR | sed 's/ \+/ /g' | perl -ne 'chomp; print
                    "\n" unless /^ /; print;' )"
120
121                 # Check for fatal errors
122                 if [[ $(print "${CLUSQMGR[$QMGR]}\n${QCLUSTER[$QMGR]}" | grep -e AMQ9508 | wc -l) -ne 0 ]]; then
123                     printf "\nFATAL: Unrecoverable error. Aborting run.\n"  | tee -a $_CONSOLE
124                     printf "${CLUSQMGR[$QMGR]}${QCLUSTER[$QMGR]}" | grep '^AMQ' | sort | uniq | tee -a $_CONSOLE
125                     printf "\n\n"  | tee -a $_CONSOLE
126                     exit
127                 fi
128
129
130                 # Check for errors indicating timeout. Break from loop if no errors.
131                 [[ $(print "${CLUSQMGR[$QMGR]}\n${QCLUSTER[$QMGR]}" | grep -e AMQ8416 -e AMQ8101 | wc -l) -eq 0 ]] && [[ $(echo "${CLUSQMGR[$QMGR]}" |
                    grep AMQ8441 | grep CLUSSDR | wc -l) -ne 0 ]] && [[ $(echo "${QCLUSTER[$QMGR]}" | grep AMQ8409 | wc -l) -ne 0 ]] && break
132                 [[ $(echo "${CLUSQMGR[$QMGR]}" | grep AMQ8441 | wc -l) -eq 0 ]] && printf "\nFound unhandled CLUSQMGR exception:\n" && print
                    "${CLUSQMGR[$QMGR]}" | grep ^AMQ | sort | uniq
133                 [[ $(echo "${QCLUSTER[$QMGR]}" | grep AMQ8409 | wc -l) -eq 0 ]] && printf "\nFound unhandled QCLUSTER exception:\n" && print
                    "${CLUSQMGR[$QMGR]}" | grep ^AMQ | sort | uniq
134             done
135
136             printf "\nRepository $QMGR returned:\n%5d clusqmgr records\n%5d qcluster records\nAfter $_ATTEMPTS attempts.\n\n"  $(echo
                "${CLUSQMGR[$QMGR]}" | grep AMQ8441 | grep CLUSSDR | wc -l) $(echo "${QCLUSTER[$QMGR]}" | grep AMQ8409 | wc -l)  | tee -a $_CONSOLE
137             if ! [[ $_ATTEMPTS -lt 10 || $(echo "${CLUSQMGR[$QMGR]}" | grep AMQ8118 | grep CLUSSDR | wc -l) -gt 0 ]]; then
138                 printf "\nFATAL: Unable to obtain clean run from $QMGR query after $_ATTEMPTS attempts. Aborting run.\n"  | tee -a $_CONSOLE
139                 echo "${CLUSQMGR[$QMGR]}${QCLUSTER[$QMGR]}" | grep -v '^AMQ8409' | grep -v '^AMQ8441'  | grep -v '^AMQ8450' | grep -v '^$'  | tee -a
```

```
                $_CONSOLE
                printf "\n\n"  | tee -a $_CONSOLE
                exit
            fi
            ((_FOUND=_FOUND+1 )) # So we can check later that all repositories were processed in case IR-360 connection for one is missing.
        else
            [[ $_TESTING -eq 1 ]] && [[ $_VERBOSE -eq 1 ]] && printf "Skipping QMgr $QMGR\n" | tee -a $_CONSOLE
        fi
    done
}



# ================================================================================
# Reconcile CLUSQMGR records
# ================================================================================
printf "Begin reconciliation of CLUSQMGR records.\n"  | tee -a $_CONSOLE

RECS1=$(echo "${CLUSQMGR[$REPO1]}" | grep AMQ8441 | grep -v -e $REPO1 -e $REPO2 | sed "s/.* QMID(\([^)]*\)).*/$REPO1 \1/g")
RECS2=$(echo "${CLUSQMGR[$REPO2]}" | grep AMQ8441 | grep -v -e $REPO1 -e $REPO2 | sed "s/.* QMID(\([^)]*\)).*/$REPO2 \1/g")

if [[ $_TESTING -eq 1 && $_VERBOSE -eq 1 ]]; then
    echo -n
    echo "Repo1 = ${CLUSQMGR[$REPO1]}"
    echo "Repo2 = ${CLUSQMGR[$REPO2]}"

#   RECS1=$(echo "$RECS1" | head -n -2) # Force discrepancies
#   RECS2=$(echo "$RECS2" | tail -n -4) # Force discrepancies
fi

printf "Found %d cluster member records for $REPO1 and found %d cluster member records for $REPO2\n" $(echo "$RECS1" | wc -l) $(echo "$RECS2" | wc -l)  |
tee -a $_CONSOLE
# Did we find two repositories?
if [[ $_FOUND -lt 2 ]]; then
    printf "\nFATAL: Found less than two repositories. Aborting run.\n"  | tee -a $_CONSOLE
    exit
fi

print "Now checking for uniqueness.\n"


# Print report table header
cat << TBLHDR >> $_RPTDIR/$_FILE
<table id="clusqmgr" width="1080" border="5" align="center" cellpadding="3" cellspacing="1">
  <tr><td colspan="2" align="center" scope="row"><h1>CLUSGMGR Recon Exceptions</h1></td></tr>
  <tr><td colspan="2" align="right" style="font-size: 60%; font-style: italic">
  Jump to: <a href="#top">Top</a>   <a href="#clusqmgr">CLUSQMGR</a>   <a href="#qcluster">QCLUSTER</a>   <a
  href="#console">Console</a>
  </td></tr>
  <tr>
```

```
187       <th scope="col">Repository</th>
188       <th scope="col">Queue Manager</th>
189     </tr>
190   TBLHDR
191
192
193   # Use sort & uniq to compare list of QMID fields at each repository.
194   if [[ $(printf "$RECS1\n$RECS2\n" | sort -k 2 | uniq -u -f 1 | wc -l) -eq 0 ]]; then
195       # QMgr member populations in cluster match across repositories
196       printf "Both repositories reporting the same cluster members.\n"  | tee -a $_CONSOLE
197       printf "  <tr><td colspan="2" align="center" scope="row">Both repositories reporting the same cluster members.</td></tr>\n" >> $_RPTDIR/$_FILE
198   else
199       # QMgr member populations in cluster do NOT match across repositories
200       printf "\nALERT! Repositories report differing sets of queue managers.\n\nList of differing QMgrs follows, tagged by repository:\n\n"  | tee -a
      $_CONSOLE
201       printf "$RECS1\n$RECS2" | sort -k 2 | uniq -u -f 1 | grep -v '^$' | awk '{print "<tr><td>" $1 "</td><td>" $2 "</td></tr>";}'  >> $_RPTDIR/$_FILE
202       [[ $_TESTING -eq 1 ]] && [[ $_VERBOSE=1 ]] && printf "$RECS1\n$RECS2" | sort -k 2 | uniq -u -f 1 >> $_CONSOLE
203   fi
204   printf '\n</table>\n<p> </p>\n'  >> $_RPTDIR/$_FILE
205
206
207   # ================================================================================
208   # Reconcile QCLUSTER records
209   # ================================================================================
210   printf "\n\nBegin reconciliation of QCLUSTER records.\n" | tee -a $_CONSOLE
211
212   # Print report table header
213   cat << TBLHDR >> $_RPTDIR/$_FILE
214   <table id="qcluster" width="1080" border="5" align="center" cellpadding="3" cellspacing="1">
215     <tr><td colspan="2" align="center" scope="row"><h1>QCLUSTER Recon Exceptions</h1></td></tr>
216     <tr><td colspan="2" align="right" style="font-size: 60%; font-style: italic">
217     Jump to: <a href="#top">Top</a>   <a href="#clusqmgr">CLUSQMGR</a>   <a href="#qcluster">QCLUSTER</a>   <a
      href="#console">Console</a>
218     </td></tr>
219     <tr><th scope="col">Repository</th><th scope="col">Queue object</th></tr>
220   TBLHDR
221
222   RECS1=$(echo "${QCLUSTER[$REPO1]}" | grep AMQ8409 | sed "s/^AMQ8409: Display Queue details\. \(.*\)$/$REPO1 \1/g" | sed "s/TYPE(QCLUSTER)
      //g;s/CLUSTER($CLUSTER) //g")
223   RECS2=$(echo "${QCLUSTER[$REPO2]}" | grep AMQ8409 | sed "s/^AMQ8409: Display Queue details\. \(.*\)$/$REPO2 \1/g" | sed "s/TYPE(QCLUSTER)
      //g;s/CLUSTER($CLUSTER) //g")
224
225   if [[ -n "$_TESTING" ]]; then
226       echo -n
227   #   RECS1="$(head -n -3 <<< "$RECS1")"
228   fi
229
230   printf "Found %d records for $REPO1 and found %d records for $REPO2\nNow checking for uniqueness.\n" $(echo "$RECS1" | wc -l) $(echo "$RECS2" | wc -l)  |
      tee -a $_CONSOLE
231
```

```
# Use sort & uniq to compare list of QMID fields at each repository.
if [[ $(printf "$RECS1\n$RECS2\n" | sed 's/CLUSTIME([^)]*)//' | sed 's/CLUSDATE([^)]*)//' | sort -k 2 | uniq -u -f 1 | wc -l) -eq 0 ]]; then
    # QMgr member populations in cluster match across repositories
    printf "Both repositories reporting the same cluster queues.\n" | tee -a $_CONSOLE
    printf "  <tr><td colspan="2" align="center" scope="row">Both repositories reporting the same cluster queues.</td></tr>\n" >> $_RPTDIR/$_FILE
else
    # QMgr member populations in cluster do NOT match across repositories
    printf "\nALERT! Repositories report differing sets of queues.\n\n" | tee -a $_CONSOLE
    printf "$RECS1\n$RECS2" | sed 's/CLUSTIME([^)]*)//' | sed 's/CLUSDATE([^)]*)//' | sort -k 2 | uniq -u -f 1  | {
        while read LINE
        do
            printf "  <tr><td valign=top>${LINE%% *}</td><td>${LINE#* }</td></tr>\n"  >> $_RPTDIR/$_FILE
        done
    }
fi
printf '\n</table>\n<p> </p>\n'  >> $_RPTDIR/$_FILE


# ================================================================================
# Print report end matter
# ================================================================================
cat << RPTMTHD >> $_RPTDIR/$_FILE
<table width="1080" border="0" cellpadding="3" cellspacing="1" align="center">
  <tr><td><h2 id="console" style="text-align:left">Console output:</h2>
  <div style="font-size: 60%; font-style: italic">
  Jump to: <a href="#top">Top</a>   <a href="#clusqmgr">CLUSQMGR</a>   <a href="#qcluster">QCLUSTER</a>   <a
  href="#console">Console</a>
  </div></td></tr>
  <tr><td>
<pre>
$(<$_CONSOLE)
</pre>
</td></tr>
</table>

<table id="methodology"  width="1080" border="0" cellpadding="3" cellspacing="1" align="center">
<tr><td><h2 style="text-align:left">Methodology</h2></td></tr>
<tr><td>
    <p>Cluster health is checked by comparing the object details in both full repositories using DIS CLUSQMGR(*) and DIS QCLUSTER(*).
    All of the attributes of the objects are compared except for the cluster date and time.  The object update date and time should
    be identical across repositories, but the timestamps that represent the individual repository's receipt of the update record
    routinely varies across repositories due to things like channel start latency.</p>

    <p>In addition to removing the cluster timestamp when processing CLUSQMGR records, the channels between repositories are filtered out
    because, by definition, these can never be the identical across repositories because for every pair one will be a CLUSSDR and the
    other a CLUSRCVR.

    <p>After removing the cluster timestamps and inter-repository channels, the remaining results are passed through sort and uniq looking
    for two specific cases. The first case is that a given object record will exist in one repository but not the other.  This results
    in one instance of a fully-qualified object record in the result set.</p>
```

```
281
282       <p>The second case is that the object exists in both repositories but with different attributes.  For instance one repository might
283       show the object as PUT enabled while the other shows it as PUT disabled. This results in two instances of the fully-qualified object
284       record in the result set.</p>
285
286       <p>After sorting the records, the uniq command strips out all entries that are identical across repositories.  The remaining result set
287       represents exceptions that indicate potential cluster health problems.  These are listed in the exception report sorted first by
288       queue manager name (in the case of CLUSQMGR) or object name (in the case of QCLUSTER), and then by the name of the repository from
289       which the record was reported.</p>
290       <p> </p>
291       <p style="font-size: 60%">$_PROG $_VERS - $(date -r $_PROG)<br />
292   </td></tr>
293   </table>
294   RPTMTHD
295
296
297   printf "\n\n<!-- Report Metadata -->\n<!-- Program=$_PROG -->\n" >> $_RPTDIR/$_FILE
298   env | sort | perl -ne 'chomp;print "<!-- $_ -->\n" unless /\e/;' | grep -v LS_COLORS >> $_RPTDIR/$_FILE
299   printf "\n\n</BODY>\n"  >> $_RPTDIR/$_FILE
300   print  "\n\n$_PROG: Report processing completed at $(date) with$( [[ -z $_ERRFLAG ]] && echo -n out ) exceptions.\nReport file: $_RPTDIR/$CLUSTER.html\n"
      | tee -a $_CONSOLE
301
302
303   # If we are running as mqm then link the primary report to the daily version
304   [[ $(whoami) = "mqm" ]] && ln -f $_RPTDIR/$_FILE $_RPTDIR/$CLUSTER.html
305
306   exit
307
308
```