Automated cluster health monitoring

T.Rob Wyatt

Your MQ Cluster is as critical as the most critical app that runs over it.

So when was the last time it had a checkup?

Nothing wrong with commercial solutions...

If you own them.

For the rest of us, here's a (relatively) simple, scripted, DIY cluster health checker.

Approach – Reconcile repository objects

- Both full repositories should contain a record of each object known to the cluster.
- In a healthy cluster, both full repositories will have the same set of objects in inventory.
- In a healthy cluster, both full repositories will show identical state for any given object.
- These assertions assume that no updates are pending in the cluster.

For purposes of this tool, an unhealthy cluster…

- May have discrepancies in the sets of objects reported by the repositories
- May have discrepancies in the state of any given object as reported by both repositories
- Or a combination of these.

Approach – Methodology

- **Collect DIS QCLUSTER and DIS CLUSQMGR output from both repositories**
- **Strip local attributes away, such as local update time of the objects**
- **Reconcile the object inventory by object name, type, and hosting QMgr.**
- **Reconcile the cluster attribute state for each object**
  - Queues: PUT, DEFBIND, MAXMSGL, etc.
  - Channels: CONNAME, SSLCIPH, etc.
  - QMgrs: Type, suspend status, etc.

Note: Local attributes such as timestamps and some operational state such as whether a channel is currently active are ignored.

Approach – Script design

- Cluster state is dynamic. Therefore a race condition exists if the cluster records are collected as they are needed. Collecting them all at once does not eliminate the race condition but it does minimize the race window.
- The script begins by collecting QCLUSTER and CLUSQMGR records for each repository and saving this data into arrays.
- Later each array is piped through utilities like grep, sed, sort, etc. to perform the analysis.
- Since client runmqsc can be a bit fragile, the collection is re-run up to 10 times or until an error-free sample is obtained.

Approach – Identifying repositories

The sample script provided fetches a flat-file database from a web server to determine the full repositories. This is the same file format as last year's presentation which fetched the same file from an Amazon S3 bucket.

An alternative is to connect to any convenient QMgr and query the full repositories it knows about, or simply pass the cluster name and repository names to the script.

The connection parameters for each of the repositories are fetched from Avada's IR-360 *Connection Inventory Report* because my last client used that tool and this was the place to get a comprehensive inventory of all QMgrs and their connection details.

An alternative is to use the MQ Explorer MQHandles.xml file, a CCDT file, a flat parameter file, or just pass the connection details to the script.

An approach

to reconciling

cluster records

Comparing repository records

- ALTDATE and ALTTIME refer to the object. These should be identical.
- The QMID is that of the QMgr hosting *this instance* of the object. If the repository has more than one entry for the same object name, they will differ by their QMID fields.
- CLUSDATE and CLUSTIME refer to the local repository's record about the object and therefore will almost never be identical. When reconciling object records, these fields must be ignored.
- Note that if CLUSDATE and CLUSTIME differ across repositories by hours or even days this indicates extremely slow propagation of cluster control data and should be investigated. That is a good topic for a different session but *not* the kind of thing this script will catch, once the propagation has completed.

Comparing CLUSQMGR records

Same as above with these additional considerations:

- **The CLUSQMGR records for the repositories will never be identical because they will be DEFTYPE of CLUSSDRB (hopefully!) on the remote repository and DEFTYPE of CLUSRCVR on the local repository.**
- **Because DEFTYPE is a critical field to check, the script logic is simpler to keep that field and instead delete the entries for the repositories.**
- **If the repositories are hosted on dedicated QMgrs that have no business objects or traffic, their CLUSQMGR records can safely be ignored.**
- **Yet another good reason to host repositories on dedicated QMgrs.**

Processing

CLUSQMGR

Records

A typical CLUSQMGR record

CLUSQMGR(BIRCH) ALTDATE(2018-09-23) ALTTIME(18.03.13) BATCHHB(0)
BATCHINT(0) BATCHLIM(5000) BATCHSZ(50) CHANNEL(FOREST.BIRCH)
CLUSDATE(2018-09-23) CLUSTER(FOREST) CLUSTIME(18.10.56)
CLWLPRTY(0) CLWLRANK(0) CLWLWGHT(50) COMPHDR(NONE)
COMPMSG(NONE) CONNAME(localhost(1415)) CONVERT(NO)
DEFTYPE(CLUSSDRB) DESCR( ) DISCINT(6000) HBINT(300) KAINT(AUTO)
LOCLADDR( ) LONGRTY(999999999) LONGTMR(1200) MAXMSGL(4194304)
MCANAME( ) MCATYPE(THREAD) MCAUSER( ) MODENAME( ) MRDATA( )
MREXIT( ) MRRTY(10) MRTMR(1000) MSGDATA( ) MSGEXIT( ) NETPRTY(0)
NPMSPEED(FAST) PASSWORD( ) PROPCTL(COMPAT) PUTAUT(DEF)
QMID(BIRCH_2018-09-19_18.06.20) QMTYPE(REPOS) RCVDATA( ) RCVEXIT( )
SCYDATA( ) SCYEXIT( ) SENDDATA( ) SENDEXIT( ) SEQWRAP(999999999)
SHORTRTY(10) SHORTTMR(60) SSLCAUTH(REQUIRED) SSLCIPH( )
SSLPEER( ) STATUS(RUNNING) SUSPEND(NO) TPNAME( ) TRPTYPE(TCP)
USEDLQ(YES) USERID( ) VERSION(09000000)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)

Prepare data for reconciliation

- Extract the CLUSQMGR repository data from the arrays.
- Exclude the repository records.
- Create a record containing the name of the repository on which this record was found and the QMID field from the record. Discard everything else.

```
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")
```

Reconcile CLUSQMGR inventory

- Sort `[repo name] QMID` records by QMID (field #2)
- Print only unique lines based on offset into record by 1 field, i.e. field #2
- Count results. Healthy=zero unique lines (every QMID appears twice)
- Otherwise, report unique lines. The `[repo name]` in the record tells the user which repository contains a record that the other does not.

```
# Use sort & uniq to compare list of QMID fields at each repository.
if [[ $(printf "$RECS1\n$RECS2\n" | 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 members.\n"  |
   tee -a $_CONSOLE
else
   # QMgr member populations in cluster do NOT match across repositories
   printf "List of differing QMgrs follows, tagged by repository:\n\n"
   printf "$RECS1\n$RECS2" | sort -k 2 | uniq -u -f 1 | grep -v '^$' |
   awk '{print "<tr><td>" $1 "</td><td>" $2 "</td></tr>";}'
fi
```

To-Do

- The current version of the report checks only whether the inventory of QMGrs across both repositories contains identical QMID values. It would be helpful to reconcile down to the individual field level as well. For example, if the two repositories have different cluster weight and rank vales for the same QMgr, the behavior of any given QMgr with respect to that channel depends on which repository provided its information for that QMgr.

- Would be nice to handle the case in which there are more than two full repositories. Although this is an edge case, it is arguably one that needs health checking even more so than does a normal cluster.

Processing

QCLUSTER

Records

A typical QCLUSTER record

echo "DIS QCLUSTER(*) CLUSTER($CLUSTER) ALL" | runmqsc $QMGR | sed 's/ \+/ /g' | perl -ne 'chomp; print "\n" unless /^ /; print;'

Captures QCLUSTER, compacts spaces and reduces to one line per object:

ASH: QUEUE(SOME.CLUSTERED.QUEUE) TYPE(QCLUSTER) ALTDATE(2018-09-23) ALTTIME(18.09.05) CLUSDATE(2018-09-23) CLUSTER(FOREST) CLUSQMGR(BIRCH) CLUSQT(QLOCAL) CLUSTIME(18.10.56) CLWLPRTY(0) CLWLRANK(0) DEFBIND(OPEN) DEFPRTY(0) DEFPSIST(NO) DEFPRESP(SYNC) DESCR( ) PUT(ENABLED) QMID(BIRCH_2018-09-19_18.06.20)

BIRCH: QUEUE(SOME.CLUSTERED.QUEUE) TYPE(QCLUSTER) ALTDATE(2018-09-23) ALTTIME(18.09.05) CLUSDATE(2018-09-23) CLUSTER(FOREST) CLUSQMGR(BIRCH) CLUSQT(QLOCAL) CLUSTIME(18.09.05) CLWLPRTY(0) CLWLRANK(0) DEFBIND(OPEN) DEFPRTY(0) DEFPSIST(NO) DEFPRESP(SYNC) DESCR( ) PUT(ENABLED) QMID(BIRCH_2018-09-19_18.06.20)

Comparing Queue object records

- ALTDATE and ALTTIME refer to the object. These should be identical.
- The QMID is that of the QMgr hosting *this instance* of the object. If the repository has more than one entry for the same object name, they will differ by their QMID fields.
- CLUSDATE and CLUSTIME refer to the local repository's record about the object and therefore will almost never be identical. When reconciling object records, these fields must be ignored.
- Note that if CLUSDATE and CLUSTIME differ across repositories by hours or even days this indicates extremely slow propagation of cluster control data and should be investigated. That is a good topic for a different session but *not* the kind of thing this script will catch, once the propagation has completed.

Reconcile by field contents

```
RECS1=$(echo "${QCLUSTER[$REPO1]}" | grep AMQ8409 | sed "s/^AMQ8409: Display
Queue details\. \(.*\)$/$REPO1 \1/g")
RECS2=$(echo "${QCLUSTER[$REPO2]}" | grep AMQ8409 | sed "s/^AMQ8409: Display
Queue details\. \(.*\)$/$REPO2 \1/g")

# Use sort & uniq to compare list of QMID fields at each repository.
if [[ $(printf "$RECS1\n$RECS2\n" | sed 's/CLUSTIME([^)]*)//' | sed
's/CLUSDATE([^)]*)//' | sort -k 2 | uniq -u -f 1 | wc -l) -eq 0 ]]; then
   # QMgr member populations in cluster match across repositories
   printf "Both repositories reporting the same cluster queues.\n"
else
   # QMgr member populations in cluster do NOT match across repositories
   printf "$RECS1\n$RECS2" | sed 's/CLUSTIME([^)]*)//’ |
   sed 's/CLUSDATE([^)]*)//' | sort -k 2 | uniq -u -f 1  | {
      while read LINE
      do
         printf "<tr><td>${LINE%% *}</td><td>${LINE#* }</td></tr>\n"
      done
   }
fi
```

Field reconciliation methodology

- **Replace the AMQ8409 eye catcher with the repository name. In the event we find unique lines, it will be helpful to have the repository name that holds the record.**
- **Skipping the repository name, apply sort and uniq on all the lines.**
  - Lines that are on only one repository will be unique.
  - Lines where fields differ will be unique.
- **Count the results. A healthy cluster will have zero unique lines.**
- **If unique lines are found, report these.**

Code Walkthrough

300 lines of code →

If you can't read these, you may want to move to the front of the room.

Go ahead. I'll wait.

Summarizing

Some mqsc client error codes to look for

- **AMQ9508: Program cannot connect to the queue manager.**
  This is fatal if encountered.
- **AMQ8416: MQSC timed out waiting for a response from the command server.**
  Script retries up to 10 times.
- **AMQ8101: An unexpected reason code with hexadecimal value 457 was received from the IBM MQ queue manager during command processing.**
  Script retries up to 10 times.

Possible uses

- **Run while QMgrs are lightly loaded and post results to a web page.**
- **Push to Splunk, ELK, or other log ingention and analysis engine.**
- **Feed to your favorite monitoring tool to create tickets or page on-call.**
  - May need to add resiliency features before raising a ticket or paging the on-call.
  - In particular, to help with Command Server fragility, make sure MAXDEPTH on the runmqsc model queue is large enough to contain all possible result sets.
  - If on a clear run uniqueness is discovered for the first time, run again and report if uniqueness is consistent after 2, or even 3 iterations.

Simple and effective

Since by definition a healthy cluster will store consistent state across the repositories, uniqueness of state data indicated an unhealthy cluster.

Sort and uniq are particularly well suited for this task!

After reducing cluster records to their significant fields, eliminate those with duplicates then count and print what remains.

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
#===============================================================================
# [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
```