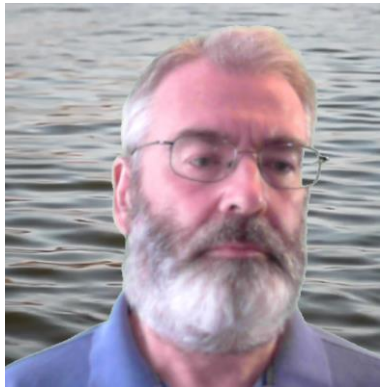
Proving it is not MQ's fault



Mark Taylor

marke_taylor@uk.ibm.com

IBM Hursley

Lyn Elkins

elkinsc@us.ibm.com

IBM Washington Systems Center

Proving it's not MQ's fault

- How many times do we get told "MQ is broken"
- And then we have to prove that the cause is elsewhere
- Use these tools and tips to reduce the Mean Time To Innocence

Why does MQ so often get initial blame

- Because of where it sits
- In between so many interacting components it is often a common layer
- MQ administrators can be the only people able to interpret language between different platforms
- And often they have the skills!

Three problem classes

- My view is there are essentially three core problems to investigate
- "Cannot connect to MQ"
- "MQ has lost my message"
- "MQ is running slowly" (aka "MQ is using too much resource")
  - This is probably where we'll spend most time

Getting the right problem description

- Exactly what is going wrong
- Versions of everything
- What changed recently
- What are the symptoms
- What have you already tried
- Can it be reproduced - testcase

Get this final error on npm install
gyp ERR! node -v v12.19.0
gyp ERR! node-gyp -v v5.1.0
gyp ERR! not ok

(After asking for more info a few more lines of output were supplied and)

"This should be enough for you"

(It wasn't.)

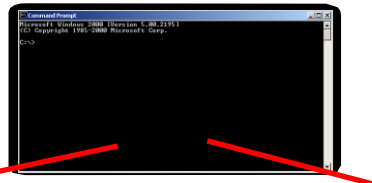
hi - Please kindly check the links provided to the documentation, one of them does not work.

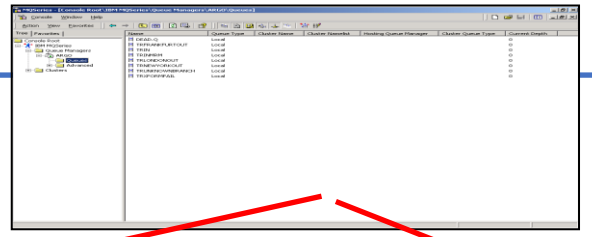
TLS isn't working

And you expect help?

- http://www.catb.org/~esr/faqs/smart-questions.html

MQ Administration





MQSC

OS

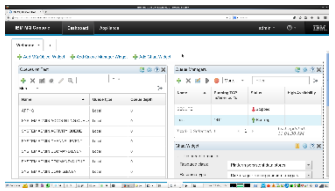
PCF

OS configuration

ini files

Queue Manager

REST



Events (async)

Log files

Status (polled)

These are all documented and supported interfaces – it has encouraged a management ecosystem over many years

MQ Monitoring Products

- Most of what we will talk about here is "tooling agnostic"

- A huge number of MQ monitoring products can look at status information
  - This session has command-line/MQSC output to show what MQ produces
  - You can then work out how that is accessed from your monitor product

- A later session looks at some specific mostly-free integrations with tools commonly used in cloud or open-source-heavy environments

Where do I look for diagnostics

- First things first …
- Application logs (assuming developer has bothered creating them)
  - MQRC values often a good indicator (and MQRC_STR is available for better logs)
- MQ Client logs
  - /var/mqm/errors
  - Redist client typically puts logs under $HOME/IBM/MQ/data
- Queue Manager/System logs
  - /var/mqm/errors, /var/mqm/qmgrs/<QM>/errors, JES logs
- FDC
  - /var/mqm/errors
- Windows: %MQ_DATA_PATH% is usual root

What is important in an FDC file

```
IBM MQ First Failure Symptom Report
=========================================

Date/Time          :- Thu February 04 2021 14:46:57 GMT
UTC Time           :- 1612450017.992878
UTC Time Offset    :- 0 (GMT)
Host Name          :- test.example.com
Operating System   :- Linux 5.8.13-200.fc32.x86_64
OS Details         :- Fedora 32 (Workstation Edition)
PIDS               :- 5724H7251
LVLS               :- 9.2.1.0
Product Long Name  :- IBM MQ for Linux (x86-64 platform)
Vendor             :- IBM
O/S Registered     :- 1 (libmqmcs_r.so) Installed
Data Path          :- /var/mqm
Installation Path  :- /opt/mqm
Installation Name  :- Installation1    (1)
License Type       :- Production
Probe Id           :- ZX000065
Application Name   :- MQM
Component          :- zxcExecutionController
Source filename    :- /Localdev/metaylor/mf/MQ9XXDFCT/lib/zu/amqzxma0.c
Line Number        :- 1345
Build Date         :- Jan  6 2021
Build Level        :- p000
Build Type         :- IKAP - (Production)
Effective UserID   :- 979 (mqm)
Real UserID        :- 6505 (met)
Program Name       :- amqzxma0
Arguments          :- -m QM1 -u met
Addressing mode    :- 64-bit
LANG               :- en_GB.utf8
Process            :- 3518101
Process(Thread)    :- 3518101
Thread             :- 1    ECMain
QueueManager       :- QM1
SubpoolName        :- QM1
UserApp            :- FALSE
```

```
Major Errorcode    :- xecL_W_PERFORMANCE_BOTTLENECK
Minor Errorcode    :- OK
Probe Type         :- INCORROUT
Probe Severity     :- 3
Probe Description  :- AMQ6125E: An internal IBM MQ error has occurred.
FDCSequenceNumber  :- 0

MQM Function Stack
amqzxma0
zxcExecutionController
xcsFFST

MQM Trace History
} xcsWaitThreadEvent rc=xecL_W_TIMEOUT
{ zapInquireStatus
} zapInquireStatus rc=OK
```

- Probe ID is key indicator of where error occurred: use it to search for known APARs
- Maj/Min Errorcodes may be followed by Arith*x*/Comment*x* fields with additional info such as *errno*

JES Logs – the first place to look

- Some examples:
  - CSQP004E – Page set I/O error
  - CSQJ109E – Out of space in BSDS
  - CSQX438E – Unable to reallocate messages in a cluster transmission queue to another target

- A JES log ‘scraper’ setting alerts is only as good as its database!

Can a trace help?

- If you need to open a PMR/Case then L2 will often want a trace
- Though some types of problem do go away when tracing enabled
  - Might slow things down just enough to avoid issues

- While intended for IBM Service use, trace can be useful to admins
  - If you have a basic idea of the queue manager and process structure
  - You can sometimes see things like OS errors show up, or additional debug statements

Favourite answers

- Lyn: "It depends"

- Mark: "What happens when you try it"

Cannot connect

- What is connected

Common Problems

- Wrong queue manager name
  - MQRC 2059 (Q_MGR_NOT_AVAILABLE) or 2058 (Q_MGR_NAME_ERROR)
  - MQ is case sensitive: "QMGR1" is not the same as "qmgr1"
  - Attempting to connect to a local queue manager instead of as client
- TLS config
  - Often MQRC 2538 (HOST_NOT_AVAILABLE)
  - Commonly because client does not have signing information for qmgr certificate
  - But lots of other reasons too – usually more details in error logs for both ends
- Authentication/authorisation
  - Look in queue manager error logs and events
  - Default configuration rejects admin users on SVRCONN without authentication

DISPLAY CONN

```
DISPLAY CONN(*) TYPE(HANDLE) ALL
```

```
AMQ8276: Display Connection details.
   CONN(577C425321295301)
   EXTCONN(414D514347415445574159312020202020)
   TYPE(HANDLE)

   OBJNAME(WLMMDB.REQUEST)                  OBJTYPE(QUEUE)
   ASTATE(NONE)                             HSTATE(INACTIVE)
   OPENOPTS(MQOO_OUTPUT,MQOO_FAIL_IF_QUIESCING)
   READA(NO)

   OBJNAME(SENDINGAPP.REPLY)                OBJTYPE(QUEUE)
   ASTATE(ACTIVE)                           HSTATE(ACTIVE)
   OPENOPTS(MQOO_INPUT_SHARED,MQOO_INQUIRE,MQOO_SAVE_ALL_CONTEXT,MQOO_FAIL_IF_QUIESCING)
   READA(NO)
```

Use CONN to match TYPE(CONN) and TYPE(HANDLE) records

TYPE(HANDLE) records let you find applications by the objects they access.
See all open handles for an app in one place, unlike DIS QSTATUS records

```
DISPLAY CONN(*) ALL
```

```
AMQ8276: Display Connection details.
   CONN(577C425321295301)
   EXTCONN(414D514347415445574159312020202020)
   TYPE(CONN)
   PID(9740)                                TID(185)
   APPLDESC(WebSphere MQ Channel)           APPLTAG(jms/GATEWAY1_CF)
   APPLTYPE(SYSTEM)                         ASTATE(NONE)
   CHANNEL(WAS.CLIENTS)                     CONNAME(127.0.0.1)
   CONNOPTS(MQCNO_SHARED_BINDING)           USERID(met)
   UOWLOG( )                                UOWSTDA(2020-04-08)
   UOWSTTI(13.24.00)                        UOWLOGDA( )
   UOWLOGTI( )                              URTYPE(XA)
   EXTURID(XA_FORMATID[DSAW]
XA_GTRID[00000145414B8AB40000000104DF48FC000102030405060708090
XA_BQUAL[00000145414B8AB40000000104DF48FC000102030405060708090
   QMURID(0.7940075)                        UOWSTATE(ACTIVE)
```

Channel name + IP help identify client apps.

JMS clients can supply an application name in the CF. MQ V9.1.2 and later, all bindings and client applications can provide a custom application name

Long running UOW information.
XID can be tied up with app server txn timeout

DISPLAY CHSTATUS

```
DISPLAY CHSTATUS(*) ALL
```

```
AMQ8417: Display Channel Status details.
   CHANNEL(WAS.CLIENTS)                    CHLTYPE(SVRCONN)
   BUFSRCVD(17)                            BUFSSENT(13)
   BYTSRCVD(2296)                          BYTSSENT(2456)
   CHSTADA(2014-04-08)                     CHSTATI(15.26.59)
   COMPHDR(NONE,NONE)                      COMPMSG(NONE,NONE)
   COMPRATE(0,0)                           COMPTIME(0,0)
   CONNAME(127.0.0.1)                      CURRENT
   EXITTIME(0,0)                           HBINT(5)
   JOBNAME(0000260C000000B9)               LOCLADDR( )
   LSTMSGDA(2019-04-08)                    LSTMSGTI(15.26.59)
   MCASTAT(RUNNING)                        MCAUSER(bbroad)
   MONCHL(OFF)                             MSGS(6)
   RAPPLTAG(jar)                           SSLCERTI(CN=ExampleCA,O=Example)
   SSLKEYDA( )                             SSLKEYTI( )
   SSLPEER(SERIALNUMBER=53:43:FD:D6,CN=ExampleApp1,O=Example)
   SSLRKEYS(0)                             STATUS(RUNNING)
   STOPREQ(NO)                             SUBSTATE(RECEIVE)
   CURSHCNV(1)                             MAXSHCNV(1)
   RVERSION(09020100)                      RPRODUCT(MQJM)
```

Check suitable heartbeats are negotiated

Is there work going on this channel?

See SSLPEER information not in DIS CONN

What kind of application is connected

MQ has lost my message

Investigating "lost" messages

- Was the message ever accepted by MQ?
  - Applications have been known to ignore the return code from MQPUT
- Can look at the current application behaviour
  - Does the app really send persistent messages?
- Did a queue fill up?
- Network routing – messages may go somewhere other than intended
  - DLQ perhaps
- Might be able to find what happened to message via recovery logs

What are the apps doing?

- Distributed have application activity trace
  - Can give information about all the MQI operations an application does in the order that they do it
- Useful for:
  - Application audit trails
  - Message duplication (though also see Stream Queues as new option)
  - Detailed problem determination
  - Enforcing application coding standards
- Should be selectively enabled to reduce performance impact
- MQ for z/OS doesn't have any direct equivalent to this

Summary report from amqsact

- By default amqsact runs in summary mode, showing you each MQI call and its MQRC and MQCC. Use –v to show full detail report.

```
MonitoringType: MQI Activity Trace
QueueManager: 'V71'
Host Name: 'localhost'
CommandLevel: 710
ApplicationName: 'MQ Client for Java'
ApplicationPid: 18612354
UserId: 'mquser'
ConnName: '9.20.95.106'
Channel Type: MQCHT_SVRCONN
Platform: MQPL_UNIX
-----------------------------------------------------------------------
Time     Operation  CompCode MQRC  HObj (ObjName)

10:04:09 MQXF_INQ   MQCC_OK  0000  2
10:04:09 MQXF_CLOSE MQCC_OK  0000  2
10:04:09 MQXF_OPEN  MQCC_OK  0000  4 ()
10:04:09 MQXF_INQ   MQCC_OK  0000  4
10:04:09 MQXF_CLOSE MQCC_OK  0000  4
10:04:09 MQXF_OPEN  MQCC_OK  0000  4 (SYSTEM.DEFAULT.LOCAL.QUEUE)
10:04:09 MQXF_INQ   MQCC_OK  0000  4
```

Detailed report from amqsact

```
ALTER QMGR ACTVTRC(ON)          <-  should be tuned via mqat.ini
amqsact -m GATEWAY1 -v
```

```
MQI Operation: 6
  Operation Id: MQXF_PUT
  ApplicationTid: 12451
  OperationDate: '2014-04-09'
  OperationTime: '01:39:48'
  High Res Time: 1397003988665548
  Completion Code: MQCC_OK
  Reason Code: 0
  Hobj: 18225032
  Put Options: 139330
  Msg length: 460
  Recs_present: 0
  Known_dest_count: 1
  Unknown_dest_count: 0
  Invalid_dest_count: 0
  Object_type: MQOT_Q
  Object_name: 'SENDINGAPP.REPLY'
  Object_Q_mgr_name: 'GATEWAY1'
  Resolved_Q_Name: 'SENDINGAPP.REPLY'
  Resolved_Q_mgr: 'GATEWAY1'
  Resolved_local_Q_name: 'SENDINGAPP.REPLY'
  Resolved_local_Q_mgr: 'GATEWAY1'
  Resolved_type: MQOT_Q
  Report Options: 0
  Msg_type: MQMT_DATAGRAM
  Expiry: -1
  Format_name: 'MQHRF2'
  Priority: 4
  Persistence: 0
  Msg_id:
  00000000:  414D 5120 4741 5445 5741 5931 2020 2020  'AMQ GATEWAY1    '
  00000010:  0207 4453 2007 2603                      '..DS .&.        '
  Correl_id:
  00000000:  414D 5120 4741 5445 5741 5931 2020 2020  'AMQ GATEWAY1    '
  00000010:  0207 4453 2007 2203                      '..DS ".         '
  Reply_to_Q : '                                                  ^D'
  Reply_to_Q_Mgr: '                                                 ^C'
  Coded_char_set_id: 1208
  Encoding: 273
  Put_date: '20140409'
  Put_time: '00394866'
```

Enabling trace

Make sure you ask amqsact to show everything the QM has generated (-v)

Check the options used for coding standards

Check queue name resolution, to find out why messages are going to the wrong place

Track individual messages and request/reply scenarios with Msg_id and Correl_id

Application activity – system topics

- Application activity trace enabled through subscriptions rather than queue manager configuration

- Subscribe to topic
  - e.g. $SYS/MQ/INFO/QMGR/QMGR1/ActivityTrace/ApplName/amqsput
  - Filter by application name, channel or connection id

- When a subscription is created, PCF messages start to flow to the subscriber’s queue. When subscription is deleted, messages stop.

- Much easier to get just the data you want

Tracing apps on z/OS

- There is an MQ API trace on MQ for z/OS
  - It is a type of GTF trace, occasionally used in development and test environments.
  - Can include before and after values of the parameters being passed on MQ API calls
  - https://www.ibm.com/support/pages/mq-gtf-api-trace-entry-fundamentals

- MQ Accounting Data
  - The Class (3) accounting data is also known as the task related data.
  - Detailed information about each task, but not about each call (unless the application only does one thing)
- If using CICS, use CEDF or CEDX
  - When testing CICS transactions can stop before and after MQ calls to examine the input and results

Did a queue fill?

- As well as return codes given to an application, you should enable Events
- Remember that QDEPTHHI/LO are percentages of MAXDEPTH

```
ALTER QL(Q1) QDPHIEV(ENABLED) QDPLOEV(ENABLED) QDEPMAXEV(ENABLED)+
             QDEPTHHI(80) +
             QDEPTHLO(20)

ALTER QMGR PERFMEV(ENABLED)
```

Looking at events

- The amqsevt sample is one way to look at events:

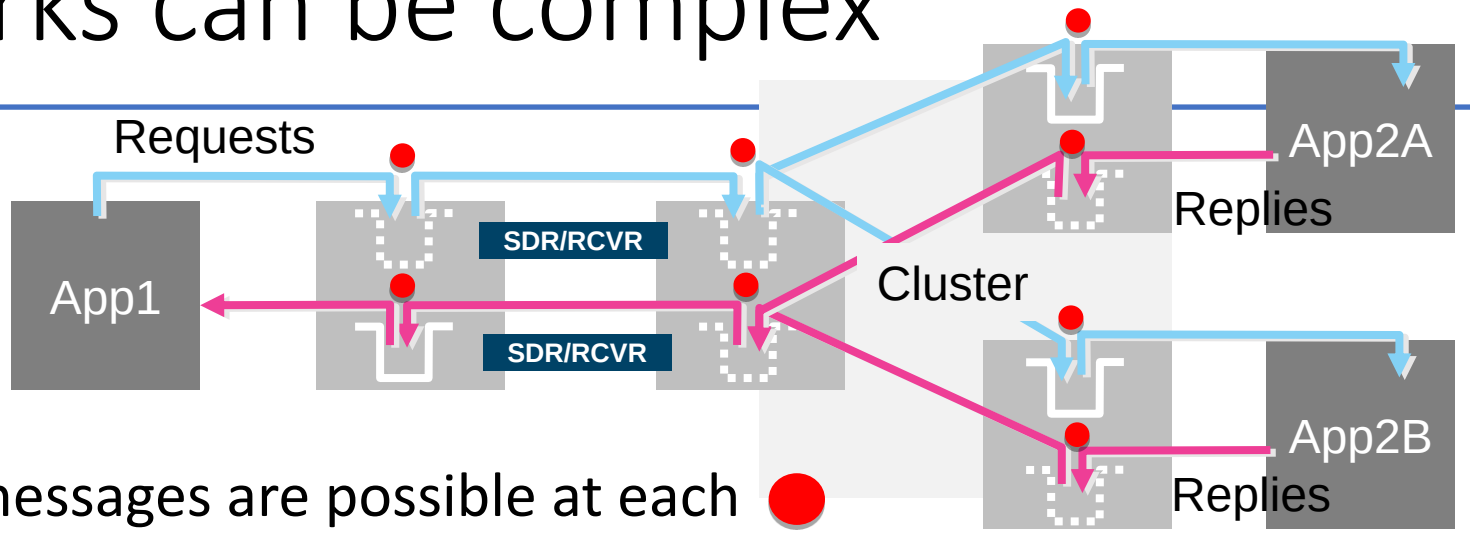

```
{"eventSource" : {"objectName":"SYSTEM.ADMIN.PERFM.EVENT","objectType":"Queue" },
 "eventType" : {"name" : "Perfm Event", "value" : 45},
 "eventReason" : {"name" : "Queue Depth High","value" : 2224},
 "eventCreation" : {"timeStamp"  : "2021-02-17T10:34:38Z","epoch": 1613558078},
 "eventData" : {"queueMgrName" : "QM1","baseObjectName" : "Q1",
   "timeSinceReset" : 22,"highQueueDepth" : 8,
   "msgEnqCount" : 8,"msgDeqCount" : 2}
}

{"eventSource" :{"objectName":"SYSTEM.ADMIN.PERFM.EVENT","objectType" : "Queue" },
 "eventType" : {"name" : "Perfm Event","value" : 45},
 "eventReason" : {"name" : "Queue Full","value" : 2053},
 "eventCreation" : {"timeStamp": "2021-02-17T10:34:39Z","epoch": 1613558079},
 "eventData" : {"queueMgrName" : "QM1","baseObjectName" : "Q1",
   "timeSinceReset" : 1,"highQueueDepth" : 10,
   "msgEnqCount" : 2,"msgDeqCount" : 0}
}
```

Events can give lots more …

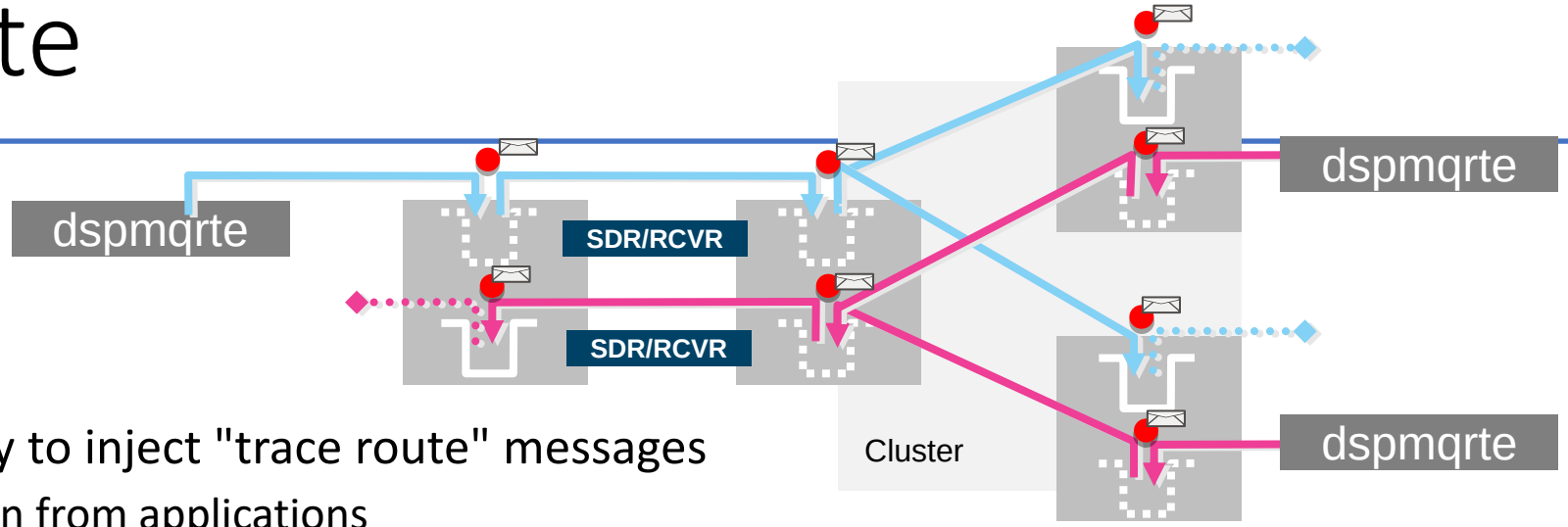
- Command/Config events
  - An audit trail of admin changes
  - I used this in conjunction with App Activity Events to determine how a monitoring tool worked – could see "RESET QSTATS" and how TDQs were used
- Authorisation/Authentication failures on Distributed platforms
  - Has more detail than entries in AMQERRxx.LOG
- But Service Interval events not always helpful
- You can reconfigure the queue manager to publish these via TOPIC ALIAS
  - Allowing multiple consumers of events

MQ networks can be complex



- Stuck / mis-sent messages are possible at each
  - MQOPENs of the wrong queue / queue manager by apps
  - Full queues
  - Stopped channels
  - Stopped apps
  - Incorrectly configured QREMOTE/QALIAS routing objects
  - Cluster membership problems
- The standard problem diagnosis approach
  - Methodically checking channels/queues/DLQs at each point
- Is there anything to speed up this process?

Trace route



- MQ has the ability to inject "trace route" messages
  - (Can be) hidden from applications
  - Generates reports as they pass through
- Tools are available to trace routes using these reports
  - dspmqrte – command line tool supplied with the product on Distributed, but can connect as client to z/OS
  - SupportPac MS0P - Explorer plugin extensions
- Lets you see the path messages could have taken
  - Test connectivity through the MQ network
  - Test cluster workload balancing
- Can quickly jump you close to the problem
  - The point your trace message veers off in the wrong direction
  - The point the trail goes cold

Example dspmqrte output

```
AMQ8653I: DSPMQRTE command started with options '-m QM1 -q LOOP.INPUT -v outline'.
AMQ8659I: DSPMQRTE command successfully put a message on queue 'QM2', queue manager 'QM1'.
AMQ8674I: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------
Activity:
 ApplName: dspmqrte
 Operation:
  OperationType: Put
  QMgrName: QM1
  QName: LOOP.INPUT
  ResolvedQName: QM2
  RemoteQName: LOOP.RETURN
  RemoteQMgrName: QM2
--------------------------------------------------------------------------------
Activity:
 ApplName: runmqchl
 Operation:
  OperationType: Get
  QMgrName: QM1
  QName: QM2
  ResolvedQName: QM2
 Operation:
  OperationType: Send
  QMgrName: QM1
  RemoteQMgrName: QM2
  ChannelName: TO.QM2
  ChannelType: Sender
  XmitQName: QM2
--------------------------------------------------------------------------------
Activity:
 ApplName: amqrmppa
 Operation:
  OperationType: Receive
  QMgrName: QM2
  RemoteQMgrName: QM1
  ChannelName: TO.QM2
  ChannelType: Receiver
 Operation:
  OperationType: Put
  QMgrName: QM2
  QName: LOOP.RETURN
  ResolvedQName: QM1
  RemoteQName: LOOP.OUTPUT
  RemoteQMgrName: QM1
--------------------------------------------------------------------------------
Activity:
 ApplName: runmqchl
 Operation:
  OperationType: Get
  QMgrName: QM2
  QName: QM1
  ResolvedQName: QM1
 Operation:
  OperationType: Send
  QMgrName: QM2
  RemoteQMgrName: QM1
  ChannelName: TO.QM1
  ChannelType: Sender
  XmitQName: QM1
--------------------------------------------------------------------------------
Activity:
 ApplName: amqrmppa
 Operation:
  OperationType: Receive
  QMgrName: QM1
  RemoteQMgrName: QM2
  ChannelName: TO.QM1
  ChannelType: Receiver
 Operation:
  OperationType: Discard
  QMgrName: QM1
  QName: LOOP.OUTPUT
  Feedback: NotDelivered

--------------------------------------------------------------------------------
AMQ8652I: DSPMQRTE command has finished.
```

Put to QREMOTE

Sender channel

Receiver channel puts to QREMOTE

Sender channel

Receiver channel

Discard at QLOCAL destination

MQ recovery logs

- For persistent messages regardless of whether in a transaction or not
  - MQ logs each operation performed (Distributed has a valid exception to this but we'll ignore here)
- Can we use this to look back in time to 2am and see what happened?
  - Recover the original payload if the app lost the message
  - See what happened inside long-running units of work
- MQ on z/OS provides a utility: CSQ1LOGP
  - Allows you to look at both active and archive logs even if queue manager is running
  - Can extract info on messages put/got in committed, rolled-back and in-flight UOW
  - Can also be used to get information on alterations to object definitions
- MQ on Distributed documents how you can get information
  - If you use the text formatting tool provided with MQ (dmpmqlog)
  - If the logging is linear so the historical data is available in the tool
  - If you follow the right steps to extract data from running qmgrs
  - If you do the work to follow through the logs

CSQ1LOGP - running

```
//CSQ1LOGP JOB NOTIFY=&SYSUID
//PRTLOG   EXEC PGM=CSQ1LOGP
//STEPLIB  DD DISP=SHR,DSN=ANTZ.MQ.V000.COM.OUT.SCSQANLE
//         DD DISP=SHR,DSN=ANTZ.MQ.V000.COM.OUT.SCSQLOAD
//ACTIVE1  DD DSN=VICY.MQ21.LOGCOPY1.DS02,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSSUMRY DD SYSOUT=*
//CSQCMT   DD SYSOUT=*
//SYSIN DD *
  PAGESET(4)
  EXTRACT(YES)
  RBASTART(000000004C99D19B)
/*
```

- ACTIVE1 DD card: name of active log to search
- CSQCMT DD card: message data from committed units of work go here
  CSQBACK, CSQBOTH, CSQINFLT and CSQOBJS are also available
- PAGESET(4): only look at data in pageset 4
- EXTRACT(YES): extract message data to specified datasets
- RBASTART(): start from this RBA to the end of the log
  RBAEND and LRSN variants are also available

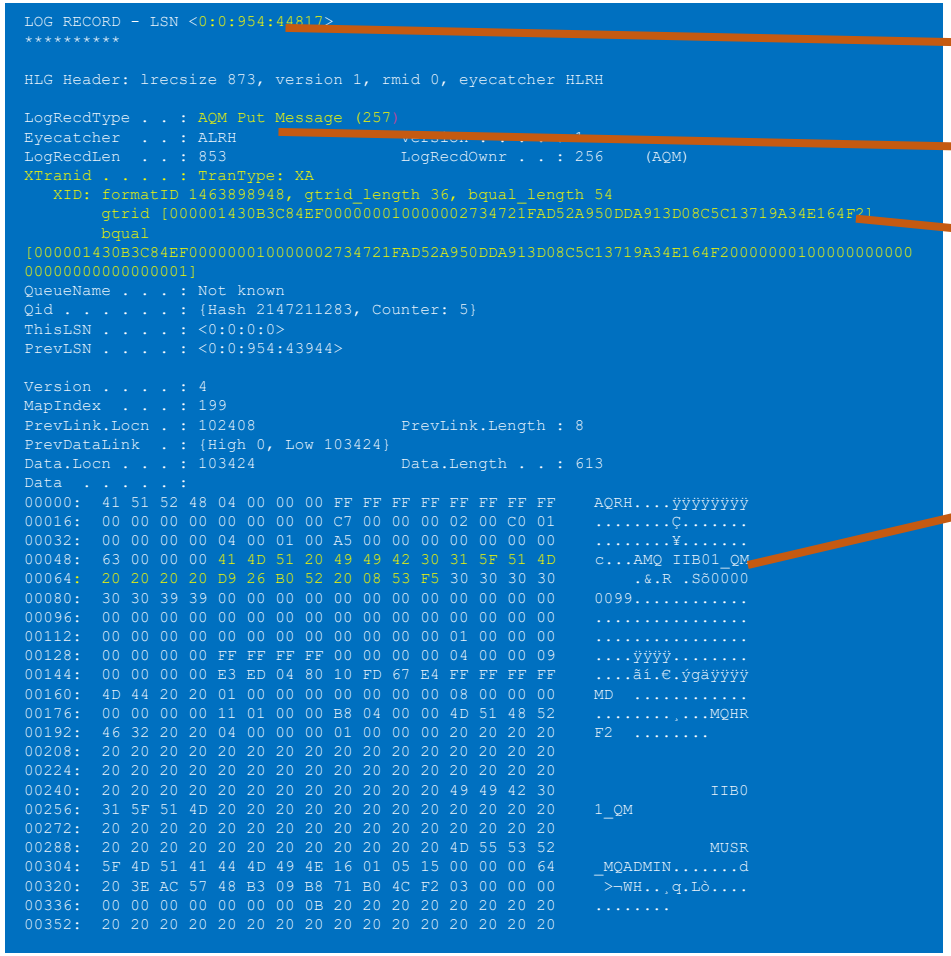
CSQ1LOGP - output

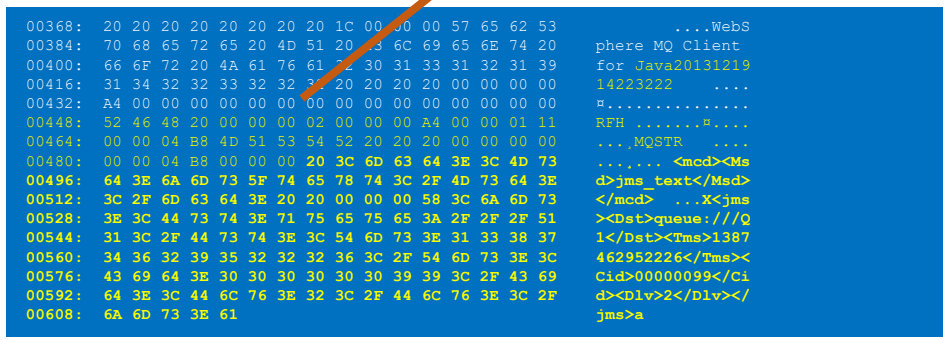
- Output is one line for each MQPUT, MQGET and each UR related operation
- MQPUT example shown below. Payload is in **bold**

```
2019.108  7:31:42.730       0N m          < w  i 8XX** Ö \MQSX00  N m   l          CHIN
MQ21CHINBUR    MQ21AQUEUE                                                          y
MQPUT   CN    < x   AMD                                          CSQ MQ21         N m
MQ21                                          MQSX00        MQ21CHIN1BEC3AE0  Ö \
LoadQueueLoopJMSWIthDifferen20190418073142 73   This is a message
```

- While this output format can be useful it is likely that you will want to run tooling against it to further refine the data.
- We provide sample C code (CSQ4LOGS) which parses output from CSQ1LOGP
  - Display summary information about each unit of work (no message data)
  - Replay messages to a target queue, possibly a different queue manager, where it can be browsed
- Obvious extension would dump just message data
  - This is left as an exercise for the user…

dmpmqlog readable, but tedious





Ordered unique IDs for each record (LSN)

A set of documented record types

Transaction information with XIDs, or re-used MQ transaction IDs

MQMD header data at discoverable offsets in the hex of a message Put

The message payload itself

Check out dmpmqlog scraper tool

- Takes the tedium out of analysing the output from dmpmqlog
- Download http://www.ibm.com/support/docview.wss?uid=swg21660642

```
java -jar dmpmqlog.scraper-20151201.jar -b little-endian -i dmpmqlog.txt -o .
```

- Generates file per message PUT in the supplied data
- Summary file

- A bit old but should still work

MQ is too slow

- MQ is using too much resource

Overview

- Start by looking if messages are flowing at all
  - Queue status
  - Channel status

- Longer-term tracking – very different models on z/OS and Distributed
  - Statistics
  - Accounting information

Real-time/online monitoring – queues

- Set detail level for queue manager. Override for individual queues

```
!MQ21 ALTER QMGR MONQ(MEDIUM)
!MQ21 ALTER QLOCAL(QUEUE1) MONQ(HIGH)
!MQ21 ALTER QLOCAL(QUEUE2) MONQ(OFF)
```

NB: Off by default, so consider switching it on at the queue manager level!

NB: HIGH/MEDIUM/LOW doesn't matter for queues

- Gives live view of application responsiveness

```
!MQ21 DIS QSTATUS(QUEUE1) ALL

CSQM201I !MQ21 CSQMDRTC  DIS QSTATUS DETAILS
   QSTATUS(QUEUE1) TYPE(QUEUE)
   OPPROCS(5) IPPROCS(3) CURDEPTH(16)
   UNCOM(YES) MONQ(HIGH)
   QTIME(10101414, 10101414)
   MSGAGE(112)
   LPUTTIME(17.12.16) LPUTDATE(2019-04-08)
   LGETTIME(17.05.59) LGETDATE(2019-04-08)
   QSGDISP(QMGR)
```

Without MONQ you only get the depth and how many handles are open.

Bear in mind that CURDEPTH includes committed and uncommitted messages. Look at UNCOM!

Age in **seconds** of the oldest message on the queue

Timestamps of last PUT/GET to check for recent activity

Estimations of the time in **microseconds** that messages are waiting on the queue for processing.
First value: Calculated from recent activity
Second value: Calculated from longer term activity

NB: for shared queues these values are only this queue manager's view, apart from CURDEPTH & MSGAGE

Real-time/online monitoring – channels

```
!MQ21 ALTER QMGR MONCHL(MEDIUM) MONACLS(MEDIUM)
!MQ21 ALTER CHANNEL(CLUSTER1.QM1) CHLTYPE(CLUSRCVR) MONCHL(HIGH)
```

- Gives live view of channel throughput

```
!MQ21 DIS CHSTATUS(TO.GWY2) ALL
```

```
CSQM201I !MQ21 CSQMDRTC  DIS CHSTATUS DETAILS
   CHSTATUS(TO.GWY2) CHLDISP(PRIVATE) XMITQ(GATEWAY2) CONNAME(127.0.0.1(1522))
   CURRENT CHLTYPE(SDR) STATUS(RUNNING) SUBSTATE(RECEIVE) INDOUBT(YES)
   LSTSEQNO(11773) LSTLUWID(0107445310001B33)
   CURMSGS(50) CURSEQNO(11823) CURLUWID(D5F6358CE52A358A)
   LSTMSGTI(17.49.51) LSTMSGDA(2014-04-08)
   MSGS(1580) BYTSSENT(1192330) BYTSRCVD(1748) BATCHES(52)
   CHSTATI(17.49.03) CHSTADA(2019-04-08)
   BUFSSENT(1616) BUFSRCVD(55)
   LONGRTS(999999999) SHORTRTS(180)
   MONCHL(MEDIUM)
   XQTIME(545784,3929968)
   NETTIME(137538,29555)
   EXITTIME(0,0)
   XBATCHSZ(20,17)
   COMPTIME(0,0) COMPRATE(0,0)
   STOPREQ(NO) KAINT(360)
   QMNAME(MQ21) RQMNAME(QWY2)
   SECPROT(NONE) SSLCERTI( ) SSLCERTU(MQSX00) SSLCIPH( )
   SSLRKEYS(0) SSLKEYTI( ) SSLKEYDA( )  SSLPEER( )
   RPRODUCT(MQMV) RVERSION(09010200)
   STATCHL(OFF) LOCLADDR(127.0.0.1(53557))
   BATCHSZ(50) MAXMSGL(4194304)
   COMPHDR(NONE NONE) COMPMSG(NONE NONE) HBINT(5) NPMSPEED(FAST))
```

Last time a message was sent over the channel

Short/long term calculations of how long messages are waiting on the XMITQ for transmission

Short/long term view of the network time to send a request and receive a response. Calculated during batch completion

Short/long term calculations of how full your batches are getting, to help you tune BATCHSZ/BATCHINT

NB: Off by default, so consider switching it on at the queue manager level!

NB: HIGH/MEDIUM/LOW doesn't matter for channels on z/OS

NB: Can be expensive for large numbers of channels due to impact on BP 0

Accounting and stats overview for z/OS

- Realtime information isn't always sufficient
  - Useful to be able to spot trends, and see how system is deviating from the average
- Queue manager can periodically output monitoring data to SMF
  - Some tools now able to capture & process SMF immediately it is output
- SMF 115: Statistics - High level queue manager wide information
  - Memory usage, log information, locking information, etc
- SMF 116: Accounting - More detailed
  - Per task information: log usage, CF usage, CPU time, some MQI information
  - Per queue information: info on MQI usage, open/close time, more log info
  - Per channel information: similar to output from DIS CHSTATUS
  - Can generate a lot of data on busy queue managers
- All information is in binary format. Assembler and C mappings provided

An SMF record dumped

```
********** TOP OF DATA **********
 message manager statistics data
--Q-M-S-T---H-E-X---P-R-I-N-T----
Address  = 2072AC08
00000000 : D40F0048 D8D4E3E2 000024FE 00002402 <M...QMST.........>
00000010 : 0000EB1A 0000B480 00000000 00000C48 <................>
00000020 : 00000000 00000000 00000000 00000000 <................>
00000030 : 00000000 00000000 00000000 00000438 <................>
00000040 : 00000000 00000000                   <........         >
--Q-M-S-T---F-O-R-M-A-T-T-E-D---
qmstid   = d40f
qmstll   = 0072
qmsteyec = QMST
qmstopen = 00009470
qmstclos = 00009218
qmstget  = 00060186
qmstput  = 00046208
```

One odd thing: these values are decimal although the leading '0' & width make it look like they are formatted as hex

Accounting and stats overview

- To collect:
  - Enable SMF to collect 115/116 data
  - Enable queue manager to generate it from startup
    - Either via CSQ6SYSP
    - Or MQ's START TRACE command
- Then dump out records from SMF
- Tooling is available to work with records
  - CSQ4SMFD: basic sample C program
  - MP1B: a SupportPac, see later slides
  - MQSMFCSV: convert SMF records into csv. Can then be imported into spreadsheets and databases
    https://github.com/ibm-messaging/mq-smf-csv
  - IBM OMEGAMON for messaging
  - Others…

```
CSQY103I !MQ21 SMFACCT= YES (F0000000),
                    SMFSTAT=YES (F0000000), STATIME=1
```

```
!MQ21 START TRACE(ACCT) CLASS(*)
```

```
!MQ21 ALTER QMGR  ACCTQ(ON)
!MQ21 ALTER QLOCAL(QUEUE1)  ACCTQ(QMGR)

!MQ21 ALTER CHANNEL(TO.GWY2) STATCHL(HIGH)
        CHLTYPE(SDR)
```

A gotcha: for chinit stats and accounting you have to issue:
START TRACE(STAT) CLASS(4)
START TRACE(ACCTG) CLASS(4)

MP1B

- Unsupported SupportPac: https://www-01.ibm.com/support/docview.wss?uid=swg24005907
- Kept reasonably up to date by MQ development team
- Uses:
  - Summary of transactions, jobs and channels
  - Summary of queues being used by application
  - List of “high use” queues
  - …
- Can detect potential issues (based on experience) and output them as a summary
- Can create csv files for import into spreadsheets allowing for graphs of activity over time to be generated
- Lots of options to customize the level of output

- Much more useful than CSQ4SMFD
- Used by lots of customers

MP1B example

- QSUML report gives a summary of activity against local queues
- Useful for understanding your expected workload against a set of queues. You can then spot anomalies

Date	Time	Qmgr	Queue	Count	PS	BP	Put MB	Get MB	!	ValidPut	ValidGet	getpsn	MaxQDep	TotalGets
12/04/2019	14:00:00	MQ21	AQUEUE	1222	4	4	20	20	!	122200	122127	0	174	122127
12/04/2019	15:00:00	MQ21	AQUEUE	1684	4	4	27	27	!	168313	168300	0	174	168300

Number of times message data wasn't in buffer pool

MP1B Example

- The TASK report digs further into accounting (116) data

```
16305 MV41,MQ21,2019/04/12,15:19:05,VRM:913,
16305 MQ21 MOVER Jobname:MQ21CHIN Userid:MQSX00
16305 Channel SYSTEM.DEF.SVRCONN   9.174.27.35
16305 Start time Apr 12 15:18:39 2019 Started in a different time interval
16305 Interval   Apr 12 15:18:39 2019 - Apr 12 15:18:41 2019 : 1.549716 seconds
16305 Other reqs : Total ET        0.000024 Seconds
16305 Other reqs : Total CPU       0.000024 Seconds
16305 Commit count                 101
16305 Commit avg elapsed time      628 uS
16305 Backout count                1
16305 Backout avg elapsed time     15 uS
16305 Backout avg CPU time         15 uS
16305 MQCTL count                  301
16305 MQCTL avg elapsed time       0 uS
16305 MQCTL avg CPU time           0 uS
16305 Total suspend time           0.062778 Seconds
16305 Open name                    AQUEUE
16305 Queue type:QLocal            AQUEUE
16305 Page set ID                  4      AQUEUE
16305 Buffer pool                  4      AQUEUE
16305 Get count                    100    AQUEUE
16305 Get avg elapsed time         42 uS  AQUEUE
16305 Get avg CPU time             41 uS  AQUEUE
16305 Get valid destructive        100    AQUEUE
16305 Put count                    100    AQUEUE
16305 Put avg elapsed time         519 uS   AQUEUE
16305 Put avg CPU time             75 uS    AQUEUE
 ...
```

Lots of backouts is normally a bad sign

Start of per queue information

Discrepancy between these two is often a sign of messages not being in buffer pool so being got from pageset

Discrepancy between these two for persistent messages is likely to be because of logging. For non-persistent messages it could be because the buffer pool is full

Performance considerations

- Capturing SMF data has a cost: CPU and storage of the records

- SMF 115 (queue manager stats): **negligible** -> always have it on
- SMF 116 (queue manager accounting class 1): **0.5%** increase in transaction cost in our workloads
  - High level API count, and CPU usage per thread
- SMF 116 (queue manager accounting, class 3 & 4): **3.3%** increase in transaction cost in our workloads
  - Per thread and queue usage information
- SMF 115 + 116 (chinit): **1-2%** increase in transaction cost in our sample workloads

- This cost needs to be compared to the benefit of having the data
- Consider sizing your environment to assume that MQ SMF data of all types will be needed at some point, or always running with it on
- Many instances of PMRs opened for performance problems but not enough headroom to enable accounting trace, which is how we diagnose performance problems!
- Plus when investigating performance issues it is always useful to have a baseline
- See capacity and planning guide: MP16 for more information
  - https://ibm-messaging.github.io/mqperf/mp16.pdf

Accounting, statistics, and system topics

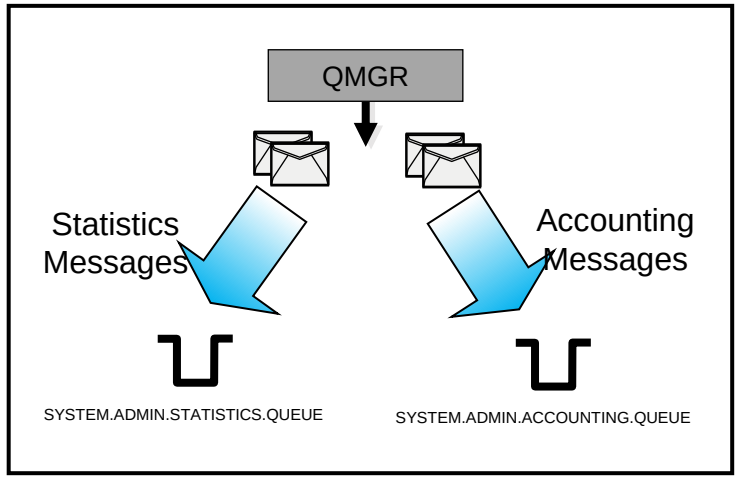
- Distributed platforms have various mechanisms to show how a queue manager is used

- Some are interactive commands as seen in the previous slides…
  - DISPLAY QMSTATUS()
  - DISPLAY QSTATUS()
  - DISPLAY CHSTATUS()
  - DISPLAY TPSTATUS()
  - DISPLAY CONN()
- Others are streams of data to consume…
  - Accounting & statistics messages
  - System topics

Better suited to monitoring over a period of time

Accounting, statistics, and system topics

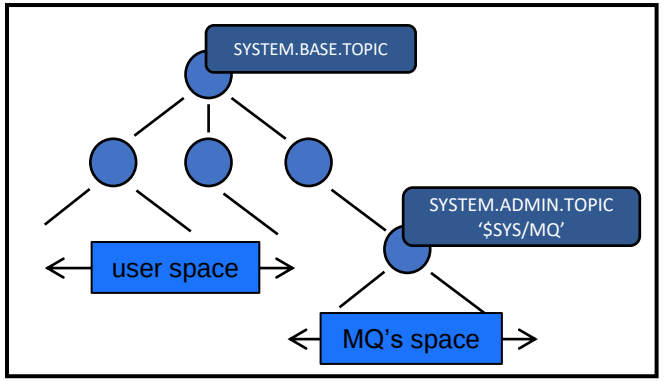
Acct
& Stats



- Enabled by turning on certain types of event message on the queue manager
- Turn on and off for specific queues and channels
- Event messages build up on system queues for you to consume

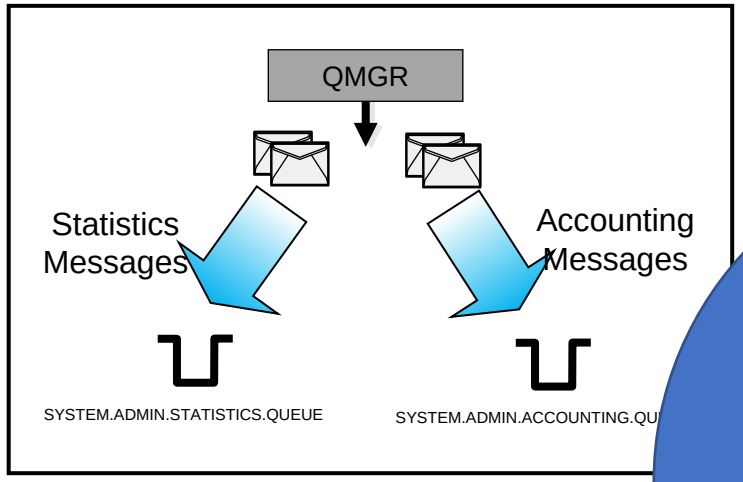
System Topics

- No need to turn on – just subscribe to the correct topic
- Better suited to multiple consumers who all want the same events
- Provides more than just MQ data (system CPU, disk IO stats)
- More granular authorisation



Accounting, statistics, and system topics

Acct
& Stats



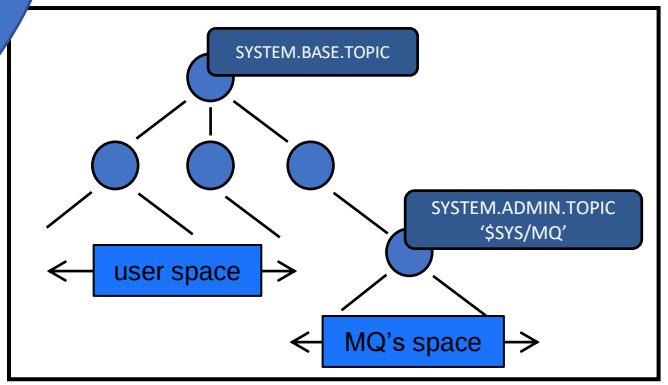
- Enabled by turning on certain types of event message on the queue manager

- off for specific queues and

- uild up on system queues

They provide a lot of overlapping information, but also have some differences

System Topics

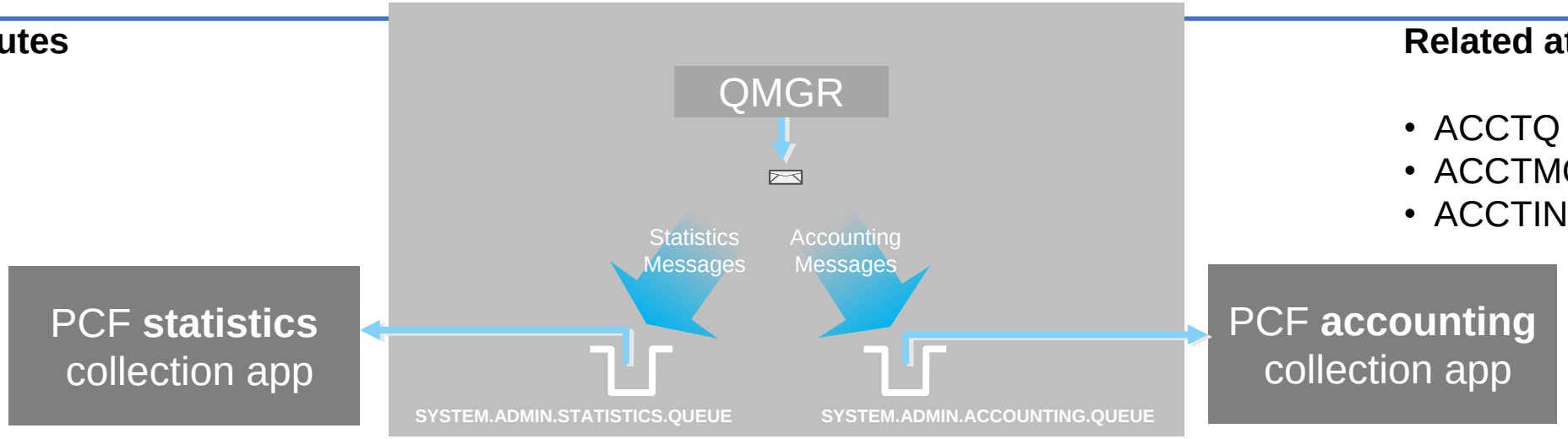


- No need to turn on – just subscribe to the co
- Better suited to multiple consumers who all want the
- Provides more than just MQ data (system CPU, disk IO stats)
- More granular authorisation

Accounting and statistics overview

Related attributes

- STATQ
- STATMQI
- STATCHL
- STATACLS
- STATINT



Related attributes

- ACCTQ
- ACCTMQI
- ACCTINT

- Monitoring data sent as a PCF message at a configured interval
- Statistics – scoped to a queue / channel / QMGR
- Accounting – scoped to an individual CONN and queue / QMGR

e.g.

MSOP Explorer Plugin
(SupportPac)

amqsmon
Specifically for accounting and statistics

amqsevt
Generic PCF event message formatter, TEXT or JSON output

Other
- ITM
- Custom app
- E.g. Java app using com.ibm.mq.headers

A statistics event message (PCF)

```
00000000:  0000 0015 0000 0024 0000 0003 0000 00A4 '.......$.......□'
00000010:  0000 0001 0000 0001 0000 0000 0000 0000 '................'
00000020:  0000 003B 0000 0004 0000 0044 0000 07DF '...;.......D...□'
00000030:  0000 0000 0000 0030 5639 3230 305F 4120 '.......0V9200_A '
00000040:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000050:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060:  2020 2020 2020 2020 0000 0004 0000 0020 '        ....... '
00000070:  0000 0A97 0000 0000 0000 000A 3230 3231 '............2021'
00000080:  2D30 322D 3138 5C06 0000 0004 0000 001C '-02-18\.........'
00000090:  0000 0A98 0000 0000 0000 0008 3039 2E31 '............09.1'
000000A0:  302E 3137 0000 0004 0000 0020 0000 0A93 '0.17....... ....'
000000B0:  0000 0000 0000 000A 3230 3231 2D30 322D '........2021-02-'
000000C0:  3138 6186 0000 0004 0000 001C 0000 0A94 '18a.............'
000000D0:  0000 0000 0000 0008 3039 2E31 302E 3239 '........09.10.29'
000000E0:  0000 0003 0000 0010 0000 001F 0000 039A '................'
```

Taking a look at statistics using amqsmon

```
ALTER QMGR STATMQI(ON)
Wait a bit, but not the default 30 minutes between stats records
RESET QMGR TYPE(STATISTICS)
amqsmon -m GATEWAY1 -t statistics -a -w 0
```

```
MonitoringType: MQIStatistics
QueueManager: 'GATEWAY1'
IntervalStartDate: '2014-04-09'
IntervalStartTime: '00.00.35'
IntervalEndDate: '2014-04-09'
IntervalEndTime: '00.01.13'
CommandLevel: 700
ConnCount: 35

PutCount: [271, 0]
PutFailCount: 0
Put1Count: [2, 0]
Put1FailCount: 0
PutBytes: [273976, 0]
GetCount: [270, 0]
GetBytes: [269468, 0]
GetFailCount: 19

DurableSubscriptionHighWater: [0, 0, 0, 0]
DurableSubscriptionLowWater: [0, 0, 0, 0]
NonDurableSubscriptionHighWater: [0, 0, 0, 0]
NonDurableSubscriptionLowWater: [0, 0, 0, 0]
PutTopicCount: [0, 0]
PutTopicFailCount: 0
Put1TopicCount: [0, 0]
Put1TopicFailCount: 0
PutTopicBytes: [0, 0]
PublishMsgCount: [0, 0]
PublishMsgBytes: [0, 0]
```

- Overall QMGR busyness
- Simple data format
  - Array shows [Persistent, NonPersistent]
- One message every X seconds
  - Use amqsmon directly
- Low/high water marks for subscriptions
  - Grouped by subscription type
- amqsmon is a sample so you can use it as a base for your own tools

Looking at accounting with amqsevt

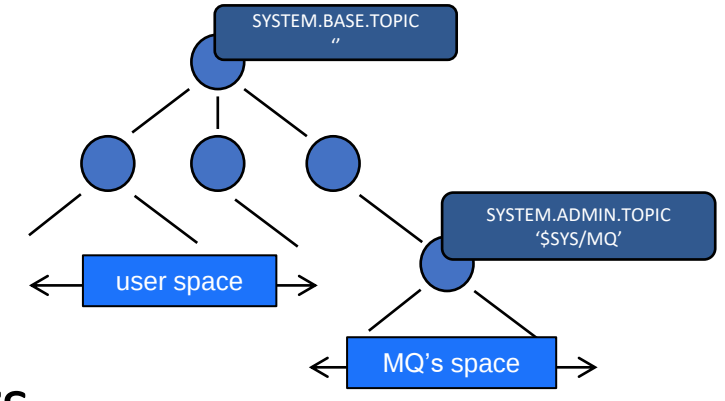
Choose JSON or TEXT format

Which type of event is this: ACCTMQI or ACCTQ?

Which application does this event relate to?

System Topics

- Distributed queue manager info is published to a range of system topics
  - $SYS/MQ/INFO/QMGR/….
- Authorised subscriptions receive stream of data based on the topic
  - Administrative subscriptions
    - E.g. For information to be continually sent to defined queues
  - Application subscriptions
    - E.g. To dynamically listen to information as required



- Unlocks system level information for MQ administrators and DevOps teams

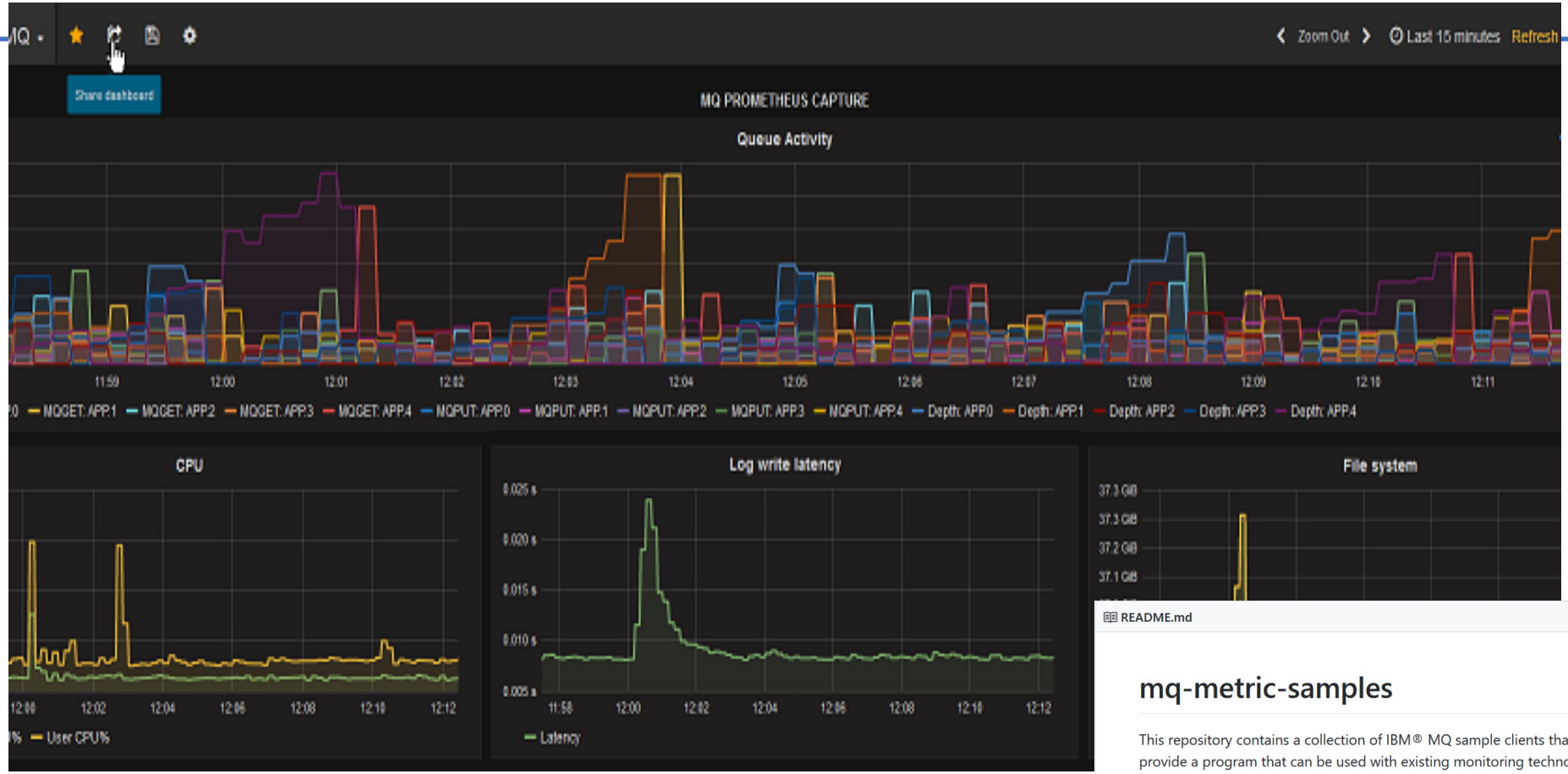
System Topics

- Familiar statistics available through subscriptions
  - Queue manager wide statistics (connects, disconnects, opens, closes, puts, gets, …)
  - Queue level statistics (opens, closes, puts, gets, …)
  - NB: statistics available from topics are not a 1-1 mapping to those available from system queues
    - E.g no channel statistics, some missing information, some new information, some merged information
  - No support for accounting data
- Extended to include CPU and disk usage. For example…
  - Queue manager CPU time, memory usage
  - Disk reads/writes, disk latency, etc.
- Subscribe to meta-topic to learn which classes of statistics are available
  - $SYS/MQ/INFO/QMGR/<qmgr>/Monitor/METADATA/CLASSES
  - Then subscribe to specific topics
    - $SYS/MQ/INFO/QMGR/<qmgr>/Monitor/class[/instance]/type]
  - See amqsrua sample program

System Topics

```
$ amqsrua -m V9000_A
CPU : Platform central processing units
DISK : Platform persistent data stores
STATMQI : API usage statistics
STATQ : API per-queue usage statistics
Enter Class selection
 ==> CPU
SystemSummary : CPU performance - platform wide
QMgrSummary : CPU performance - running queue manager
Enter Type selection
 ==> SystemSummary
Publication received PutDate:20160411 PutTime:10465573
User CPU time percentage 0.01%
System CPU time percentage 1.30%
CPU load - one minute average 8.00
CPU load - five minute average 7.50
CPU load - fifteen minute average 7.30
RAM free percentage 2.02%
RAM total bytes 8192MB
Publication received PutDate:20160411 PutTime:10466573
User CPU time percentage 0.01%
System CPU time percentage 1.30%
…
```

Used as basis for some monitors



README.md

mq-metric-samples

This repository contains a collection of IBM® MQ sample clients that utilize the IBM® MQ golang metric packages to provide a program that can be used with existing monitoring technologies such as Prometheus, AWS CloudWatch, etc.

Health Warning

This package is provided as-is with no guarantees of support or updates. There are also no guarantees of compatibility with any future versions of the package; interfaces and functions are subject to change based on any feedback.

These programs use a specific version of the `mqmetric` and `ibmmq` golang packages. Those packages are in the mq-golang repository and are also included in the `vendor` tree of this repository.

Reference – Distributed Acct/Stats

Accounting Messages Designed to show you per-application data		**Statistics Messages** Designed to show you per-object data	
ACCTMQI	**Includes** Number of all API calls, bytes put/got etc. for each application/connection to the QM, across all objects the application uses. Includes everything in the "Not included" list for ACCTQ. Where relevant, includes other object types (e.g. number of opens/closes for objects such as namelists, auth info objects, process objects etc.) **Example uses** • Discover applications that frequently backout transactions • Discover which apps use pub/sub (not included in ACCTQ events) • Charge-back to departments based on how much they use the qmgr	STATMQI	**Includes** Number of all API calls, bytes put/got etc. for the queue manager as a whole. Includes everything in the "Not included" list for STATQ. Where relevant, includes all object types (e.g. number of opens/closes for objects such as namelists, auth info objects, process objects etc.) **Example uses** Discover QMs that are very heavily or very lightly utilised Discover how often applications connect to a QM and how many are connected at any one time Discover how many messages are expired from the QM
ACCTQ	**Includes** Number of queue-related API calls, bytes put/got, queue events generated etc. for each application/connection using that queue. The max and min size of the messages used by each application. **Not included** Calls not related to messaging on a specific queue, e.g. • Topic/subscription related API calls • Inquire/set calls • Callback API calls • Commit/back API calls • Connect/disconnect activity **Example uses** • Discover which applications use QUEUE1. • See which of the applications using QUEUE1 create the most traffic. • Charge-back to departments based on how much they use a given queue	STATQ	**Includes** Summary of API activity on a specific queue, regardless of which application(s) are using it. Statistics such as • The min and max depth of the queue during the period • The average time messages spent on the queue during the period • The number of messages that expired during the period. **Not included** Calls not related to messaging on a specific queue, e.g. • Topic/subscription related API calls • Inquire/set calls • Callback API calls • Commit/back API calls Information about the size of messages put/got from the queue. **Example uses** Discover any queues that are never used Discover any queues that are being used as a database

Reference – Distributed Acct/Stats

Accounting Messages Designed to show you per-application data		**Statistics Messages** Designed to show you per-object data	
		STATCHL, STATACLS	**Includes** Per-channel information about the flow of data over each channel: • Number of messages sent across it • Number of bytes in total • The maximum, minimum and average round-trip latency • Whether batches are being filled before sending **Example uses** Identify slow network connections Understand traffic flow between QMs Identify channels that may need tuning

System Topics (e.g. $SYS/MQ/INFO/QMGR/QM1/Monitor/STATMQI/GET)			
STATMQI	**Includes** Very similar data to the MQ events generated on the SYSTEM.ADMIN.STATISTICS.QUEUE (see previous slides) **The main differences between $SYS STATMQI data and SYSTEM.ADMIN.STATISTICS.QUEUE data are:** • Each MQI call (such as GET) or pairs of related calls (such as INQ & SET) are published on their own topic. Subscribe to each MQI call you are interested in • You first subscribe to a metadata topic that describes the contents of the actual event messages • Some fields show calculated messaging rates e.g. number of PUT bytes/sec	STATQ	**Includes** Very similar data to the MQ events generated on the SYSTEM.ADMIN.STATISTICS.QUEUE (see previous slides) **The main differences between $SYS STATQ data and SYSTEM.ADMIN.STATISTICS.QEUE data are:** • Each MQI call (such as GET) or pairs of related calls (such as INQ & SET) are published on their own topic. Subscribe to each MQI call you are interested in • You first subscribe to a metadata topic that describes the contents of the actual event messages • Some additional fields, e.g. amount of lock contention on the queue

Reference – Distributed Acct/Stats

System Topics (e.g. $SYS/MQ/INFO/QMGR/QM1/Monitor/DISK/QMgrSummary)			
CPU	**Includes** • System CPU load over 1, 5 and 15 minute periods • System CPU load caused by QM and MQ applications • Free RAM (% and absolute) • RAM used by QM **Example uses** • Remotely monitor system load • Understand how much load MQ is putting on a system	DISK	**Includes** Amount of disk consumed by errors, trace & FDCs (% and absolute) Amount of disk consumed by QM (% and absolute) Log and log FS size, number of bytes written to log Log write latency Primary log currently in use **Example uses** • Remote monitor disk load • Understand how much disk MQ is consuming • Understand if MQ logs are correctly tuned

- Note 1: STATMQI and STATQ are intentionally used in relation to both statistics messages and system topic categories. This is because they offer similar information, even if they are not identical.
- Note 2: There is no equivalent of ACCTQ, ACCTMQI, or STATCHL on the $SYS system topics. These must still be generated by enabling them on the queue manager and consuming the event messages from the appropriate queue

Summary

- Lots of tools in your MQ toolbox

- Error and trace
- On-line status commands
  - DISPLAY CONN, xxSTATUS
- Off-line statistics and accounting
- Tracking
  - Trace-route
  - Application activity trace
- MQ recovery logs

Questions & Answers