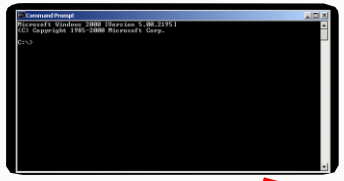
Open Source Monitoring for IBM MQ

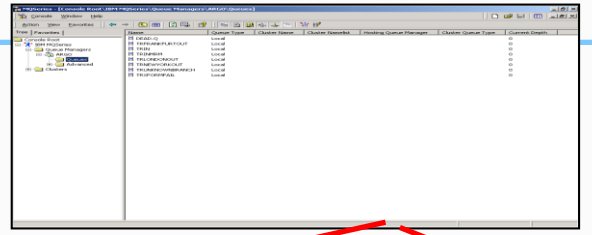
Mark Taylor

marke_taylor@uk.ibm.com

IBM Hursley

MQ Administration



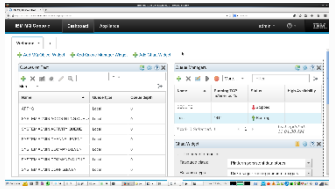


MQSC

OS

PCF

REST



OS configuration

Queue Manager

ini files

Events (async)

Status (polled)

Log files

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

IBM MQ - MQSC

Old Example (1995): AIX smit panels

- Command line interface
- V8 enhanced runmqsc
  - Make it world-executable
  - Enable direct client-connection
- MQSC intended for human consumption
  - Parsable by eye, less easy in programs
  - For example, **DESCR('This is 'a' description with quote & paren(')**
  - No guaranteed ordering in runmqsc, two-column output
- Despite awkwardness, basis for many script-based admin tools
  - echo "DISPLAY Q(X) IPPROCS" | runmqsc QM1
- Same commands – different front-end (CSQUTIL) – for z/OS

IBM MQ - PCF

- A "self-describing" MQ message used for administrative operations
- Your programs can send commands and get responses using PCF
  - Equivalent to "DISPLAY QSTATUS" or "ALTER CHANNEL"
- MQ emits events in PCF format
  - "Queue is getting full"
- PCF intended for programs – usually C or Java
  - Can tell exactly what the parameter is for, its length and value
  - But cannot easily be scripted
- Approximately one-one mapping between MQSC commands and PCF
- Remember that PCF invented before formats like JSON or XML
  - And there are many MQ apps that are built on PCF

An event message

```
****   Message length - 300 of 300 bytes ***

00000000:  0000 0007 0000 0024 0000 0003 0000 0063 '.......$.......c'
00000010:  0000 0001 0000 0001 0000 0000 0000 096C '...............l'
00000020:  0000 0002 0000 0014 0000 0010 0000 1F41 '...............A'
00000030:  0000 0004 0000 0004 0000 0020 0000 0BE5 '........... ...å'
00000040:  0000 0333 0000 000C 6D65 7461 796C 6F72 '...3....metaylor'
00000050:  2020 2020 0000 0003 0000 0010 0000 03F3 '    ...........ó'
00000060:  0000 0001 0000 0004 0000 0044 0000 0BE7 '...........D...ç'
00000070:  0000 0333 0000 0030 5638 3030 335F 4120 '...3...0V8003_A '
00000080:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000090:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
000000A0:  2020 2020 2020 2020 0000 0003 0000 0010 '        ........'
000000B0:  0000 03FD 0000 005A 0000 0014 0000 0010 '...ý...Z........'
000000C0:  0000 1F42 0000 0004 0000 0004 0000 0018 '...B............'
000000D0:  0000 0BFB 0000 0000 0000 0001 5800 0000 '...û........X...'
000000E0:  0000 0003 0000 0010 0000 03F8 0000 0001 '...........ø....'
000000F0:  0000 0006 0000 0024 0000 0BF9 0000 0000 '.......$...ù....'
00000100:  0000 0001 0000 0008 6D65 7461 796C 6F72 '........metaylor'
00000110:  0000 0000 0000 0005 0000 0018 0000 045C '...............\'
00000120:  0000 0002 0000 000B 0000 0009           '............    '
```

An event message

```
****   Message length - 300 of 300 bytes ***

00000000:  0000 0007 0000 0024 0000 0003 0000 0063 '.......$.......c'
00000010:  0000 0001 0000 0001 0000 0000 0000 096C '...............l'
00000020:  0000 0002 0000 0014 0000 0010 0000 1F41 '...............A'
00000030:  0000 0004 0000 0004 0000 0020 0000 0BE5 '........... ...å'
00000040:  0000 0333 0000 000C 6D65 7461 796C 6F72 '...3....metaylor'
00000050:  2020 2020 0000 0003 0000 0010 0000 03F3 '    ...........ó'
00000060:  0000 0001 0000 0004 0000 0044 0000 0BE7 '...........D...ç'
00000070:  0000 0333 0000 0030 5638 3030 335F 4120 '...3...0V8003_A '
00000080:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000090:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
000000A0:  2020 2020 2020 2020 0000 0003 0000 0010 '        ........'
000000B0:  0000 03FD 0000 005A 0000 0014 0000 0010 '...ý...Z........'
000000C0:  0000 1F42 0000 0004                     '...B............'
000000D0:                                          '...û........X...'
000000E0:  0000 0003 0000 0010 0000 03F8 0000 0001 '...........ø....'
000000F0:  0000 0006 0000 0024 0000 0BF9 0000 0000 '.......$...ù....'
00000100:  0000 0001 0000 0008 6D65 7461 796C 6F72 '........metaylor'
00000110:  0000 0000 0000 0005 0000 0018 0000 045C '...............\'
00000120:  0000 0002 0000 000B 0000 0009           '............    '
```

		TYPE (cfst)	LEN (24)
PARM (MQCA…)	CCSID (0)	LEN (1)	DATA

Event formatting C sample in V8.0.0.4

- No sample previously shipped to format all "standard" events
  - Authorisation, queue full, service interval, command/config etc
  - Other product samples **are** available for acct/stats, activity reports
  - Several SupportPacs but product only has out-of-date source code in the KC
- The **amqsevt** program formats events into readable English-ish text
  - Option to stay with full MQI constant name instead of making it look nice
  - Uses MQCB to read from multiple event queues. No polling required
  - Can connect as client to any remote queue manager including z/OS
  - Source code included
- Includes C header file to help convert MQI numbers to strings
  - Similar to Java MQConstants.lookup() capability for all sets of constants

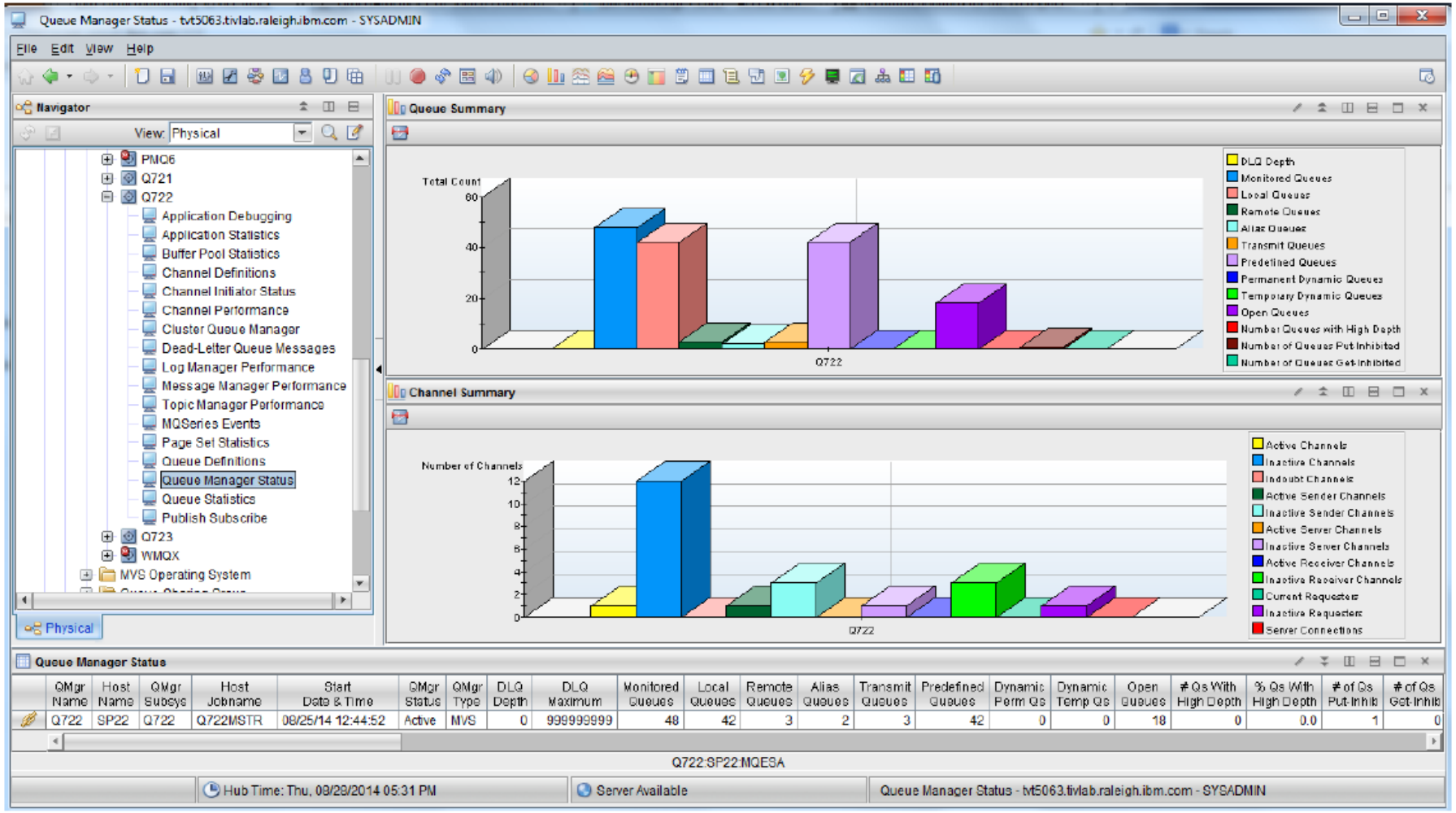
```
printf("Error is %s\n",MQRC_STR(2035));
```

An event message decoded

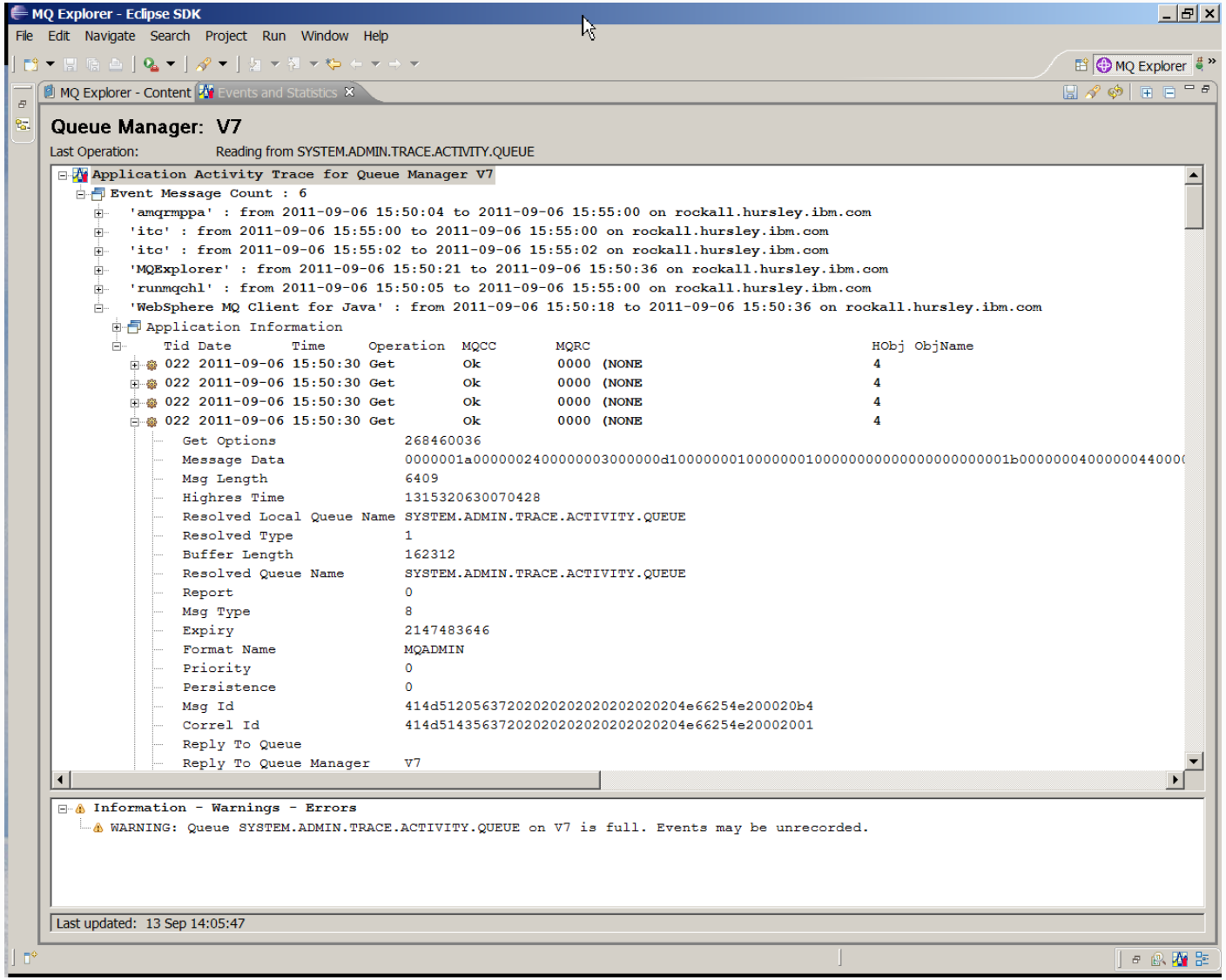
```
Event Type                    : Command Event
Reason                        : Command MQSC
Event created                 : 2015/06/03 13:28:20.51 GMT
Correlation ID : 414D5120563830303 35F412020202020556F00F120001E05
  COMMAND CONTEXT
    Event User Id             : metaylor
    Event Origin              : Console
    Event Queue Mgr           : V8003_A
    Command                   : Set Auth Rec
  COMMAND DATA
    Auth Profile Name         : X
    Object Type               : Queue
    Principal Entity Names: metaylor
    Auth Add Auths            : Output
                              : Input
```

Third-party solutions

- Many vendor products – this screenshot from ITCAM/Omegamon

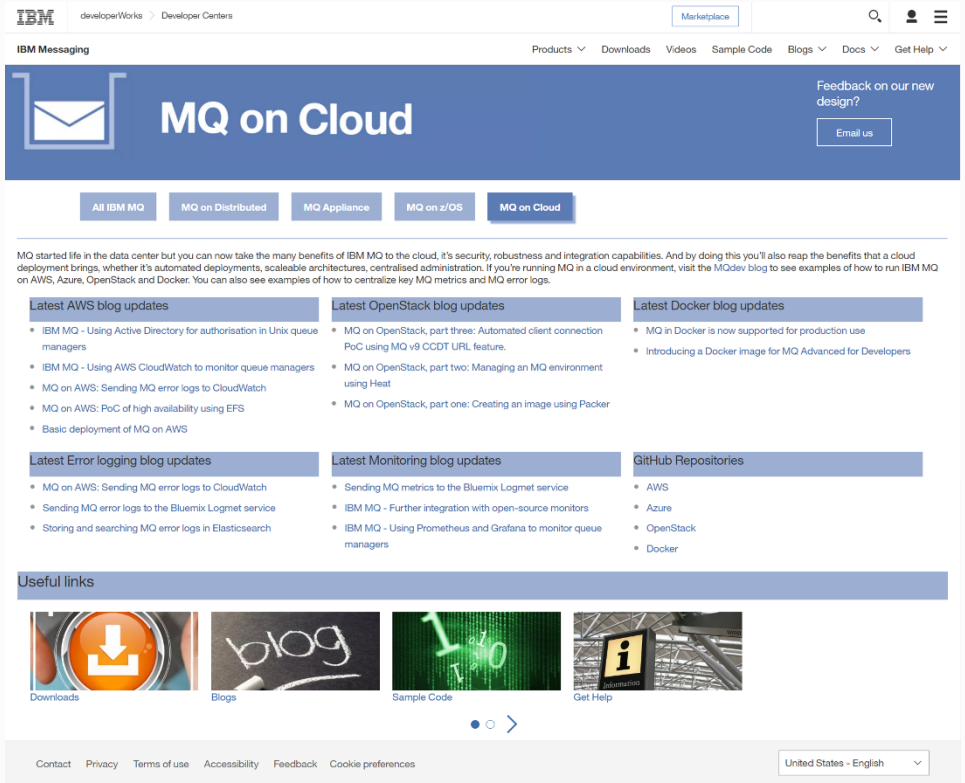


Application Activity inside MQ Explorer using MS0P



Many people now using different tools

- Because they are using those tools for other products
- And because MQ is being used in more environments
- Therefore MQ has to be able to be integrated with them
- You are unlikely to buy Tivoli if other parts of your infrastructure are being monitored via Grafana



https://developer.ibm.com/messaging/mq-on-cloud/

Decided to demonstrate MQ monitoring integration

- Using the V9 resource statistics data
- Feeding a variety of monitoring tools
- And doing it in public – Github, blog articles etc
  - See **github.com/ibm-messaging/mq-golang**
  - Video at **youtube.com/watch?v=Pi_jHCiqTgU**
- Other integration aspects – availability, security, deployment – also demonstrated

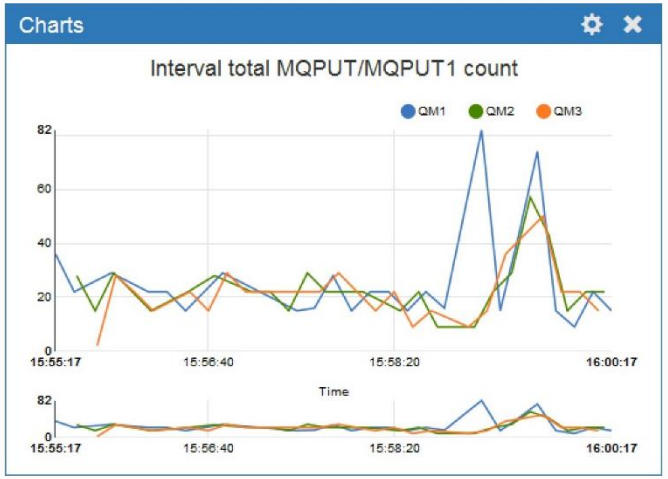
System Monitoring with V9

- More statistics available via a pub/sub model
- Includes CPU and Disk usage
  - As well as many MQ statistics
  - Not full replacement for accounting/statistics events but many key values
- 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
  - See amqsrua sample program
- Distributed platforms only
- User applications can generate their own monitoring data in this style
  - The MQ Bridge to Salesforce contributes statistics

System Monitoring Example

```
$ 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
```

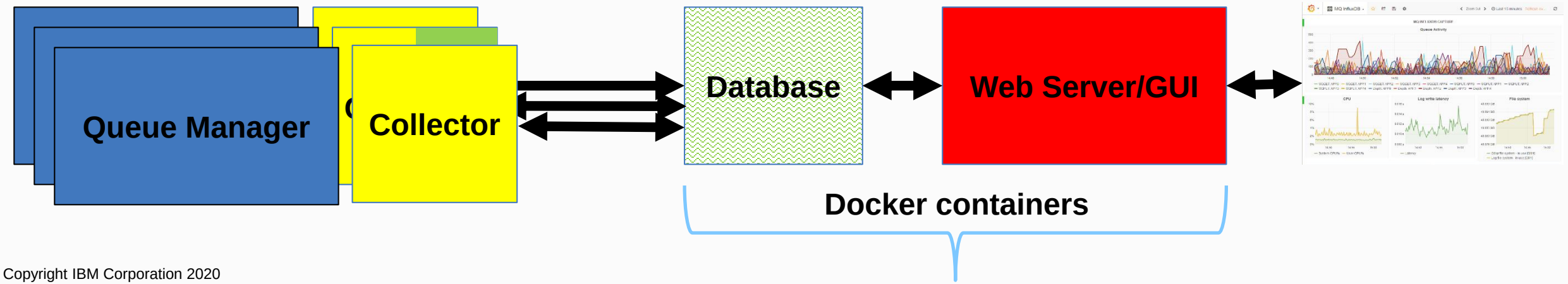
This capability underpins the charting in the MQ Console UI



Monitoring Architecture

- Architecture is split – database and user interface
  - The database is usually a "time-series" DB, not traditional SQL
  - Designed and optimised for {timestamp, metric, value} storage and queries

- These databases include Prometheus, InfluxDB, OpenTSDB
- Collection architecture may have intermediate layers – collectd



Started with Prometheus

- Seemed to be one of the most popular
- Which does have its own limited GUI
- Model is "pull" – calls a collector program at intervals via http
  - Most other DBs are "push" where collector sends to DB at interval
- Standard API for getting data to Prometheus is in Go
  - And we had no Go API for MQ …

The Go API for MQ

- So first off, I had to create a new language binding
  - Based on full MQI rather than a "simplified" version
  - But not all function implemented
  - Trying to make it look natural to Go programmers

```
if err == nil {
  putmqmd := ibmmq.NewMQMD()
  pmo     := ibmmq.NewMQPMO()
  pmo.Options = ibmmq.MQPMO_SYNCPOINT | ibmmq.MQPMO_NEW_MSG_ID
  putmqmd.Format = "MQSTR"
  msgData := "Hello from Go"
  buffer := []byte(msgData)
  err = qObject.Put(putmqmd, pmo, buffer)
  if err != nil {
    fmt.Println(err)
  }
}
```

Collector configurations

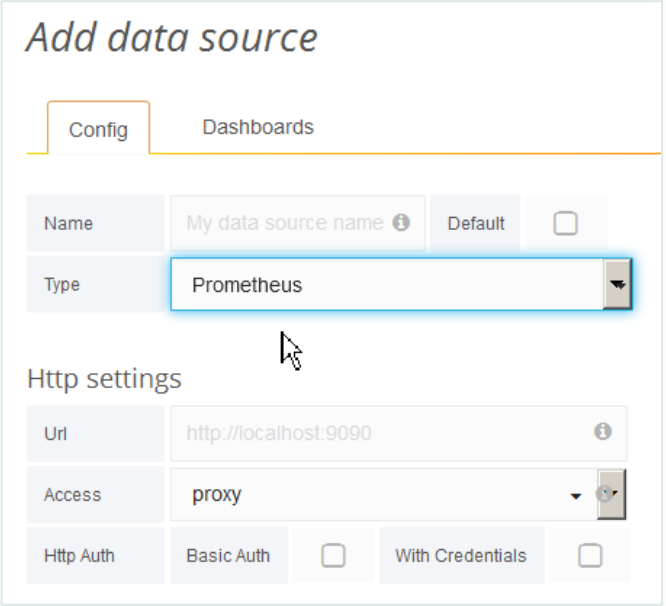
- Collector subscribes to all data for qmgr (cpu, disk etc) and nominated queues
  - Command line parameters name the queues with wildcards
- Started via MQ Service definition and shell script
- Can connect as client to remote queue managers including MQ appliance
  - Any system that supports the resource statistics
  - One collector instance per queue manager

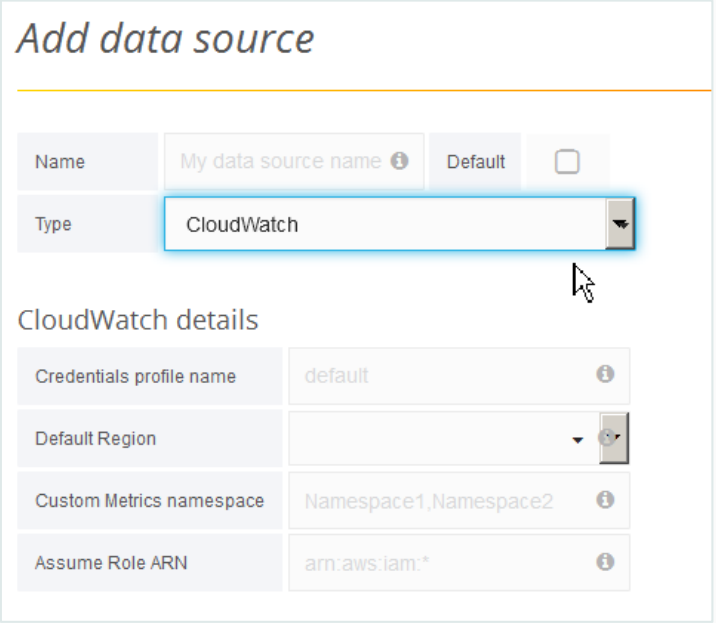
```
/usr/local/bin/mqgo/mq_prometheus -ibmmq.queueManager=QM1
    -ibmmq.monitoredQueues=APP.*,MYQ.*
    -ibmmq.httpListenPort=9157
    -log.level=error
```

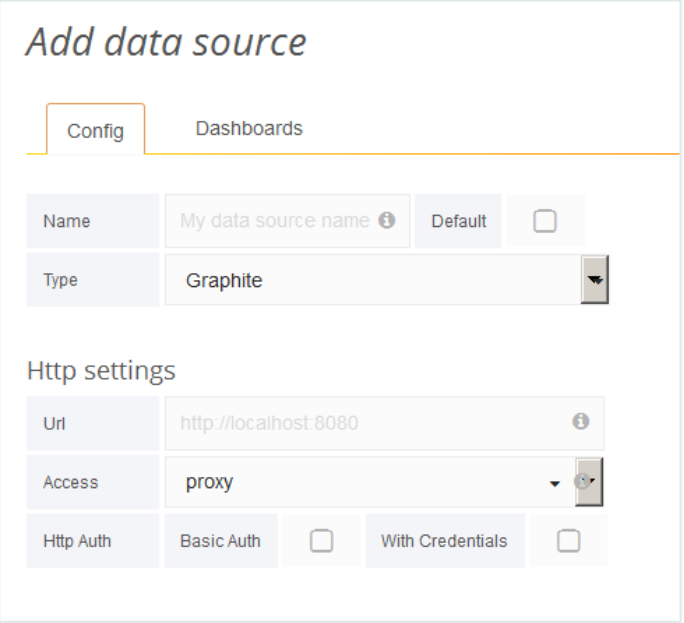
Grafana

- Although Prometheus has a GUI it is not very sophisticated

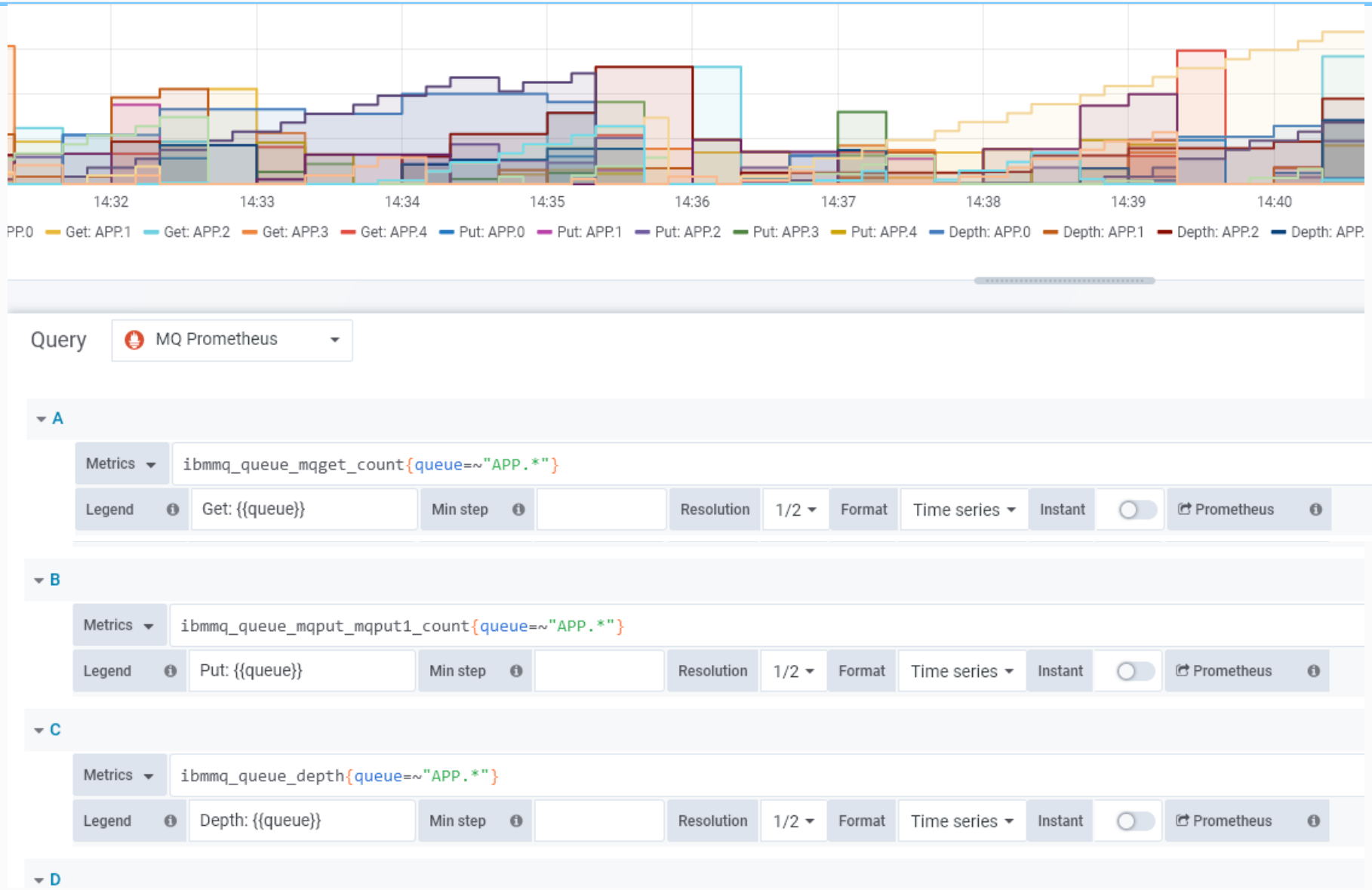
- Instead, prefer to use Grafana as visualisation tool
  - Supports many different backend databases
  - Understands the metric names, query capabilities etc of each



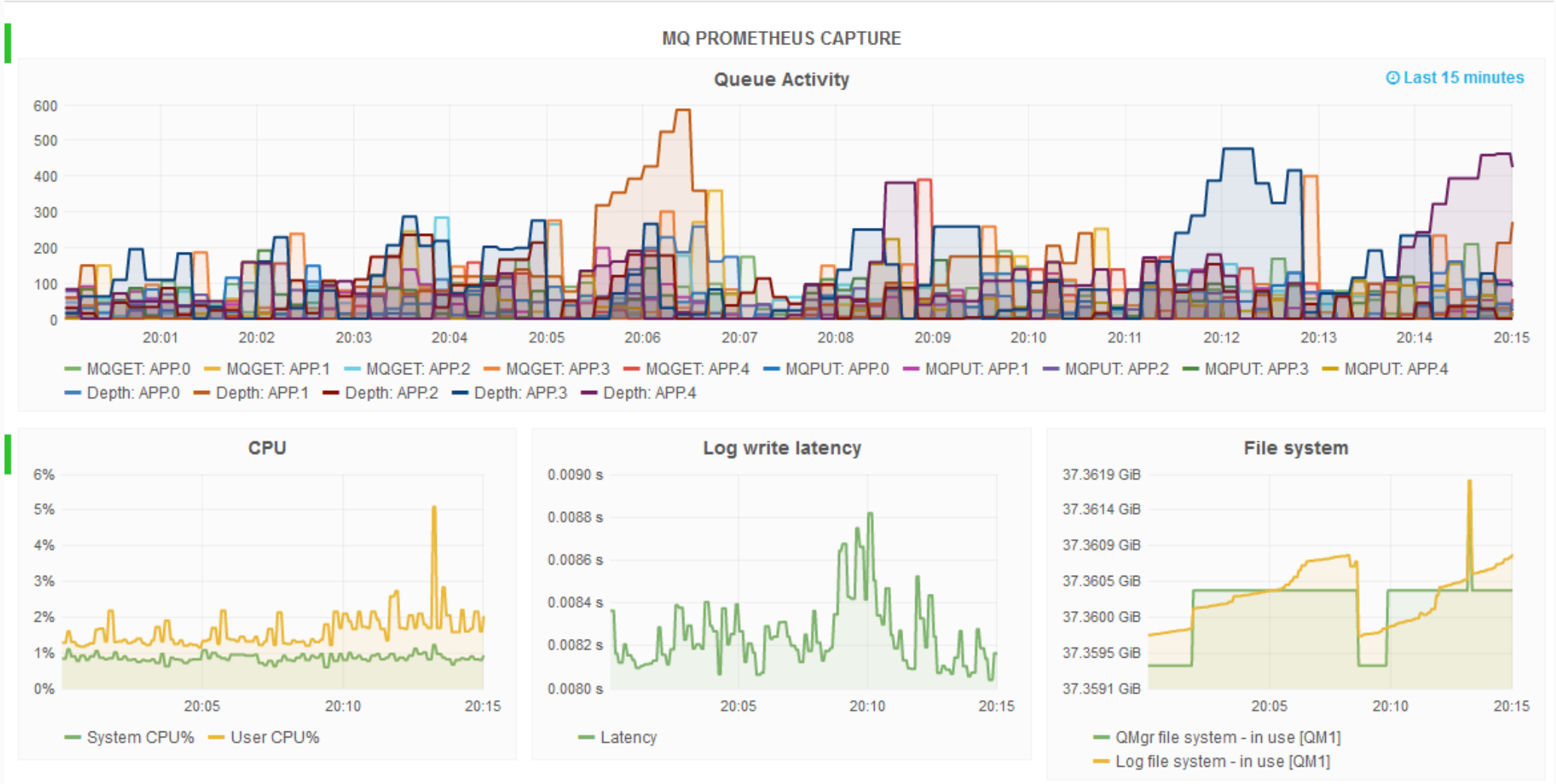




Accessing queue stats from Prometheus in Grafana



Grafana dashboard

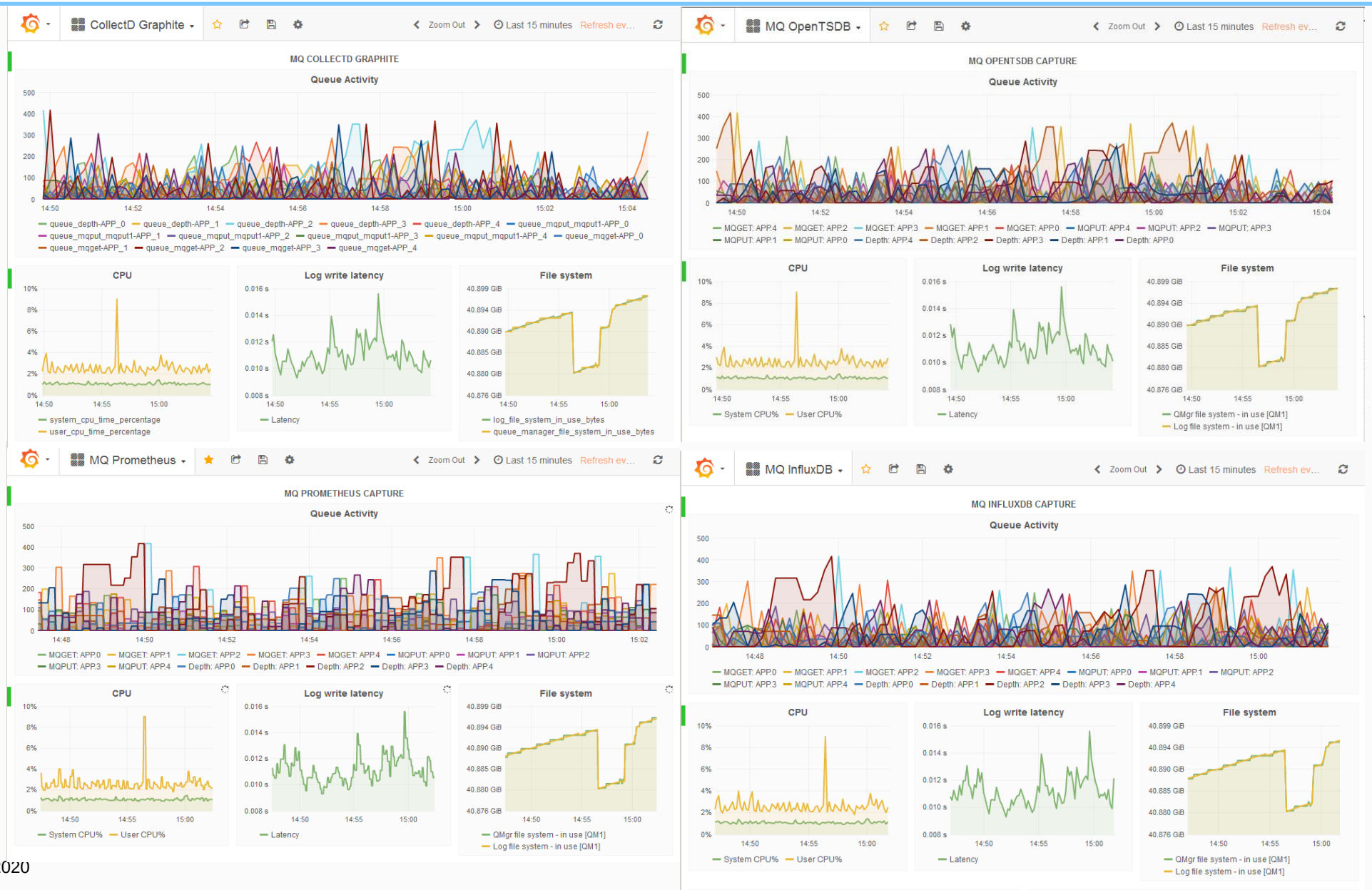


Then added more variants

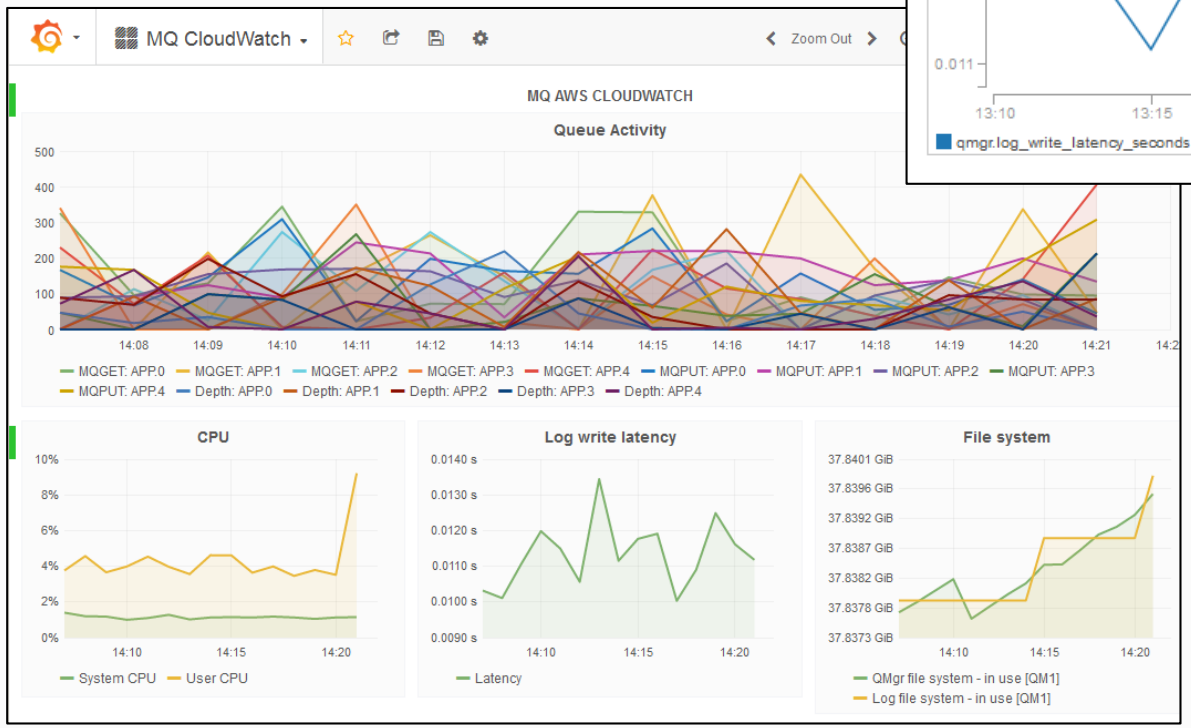
- Rapidly added support for Influx, OpenTSDB
  - Different collectors with slightly different parameters
- Graphite is another database, but fed via collectd
- Also added an AWS collector for CloudWatch
- Generic JSON formatting

```
{ "collectionTime" : {
      "timeStamp" : "2016-11-07-T15:00:55Z"
      "epoch" : 1478527255   },
   "points" : [
      { "queueManager" : "QM1", "ramTotalBytes" : 15515735206 },
      { "queueManager" : "QM1", "userCpuTimePercentage" : 1.33 }
   ]
}
```

Four equivalent Grafana dashboards



AWS Cloudwatch



Requirement: Monitor channels

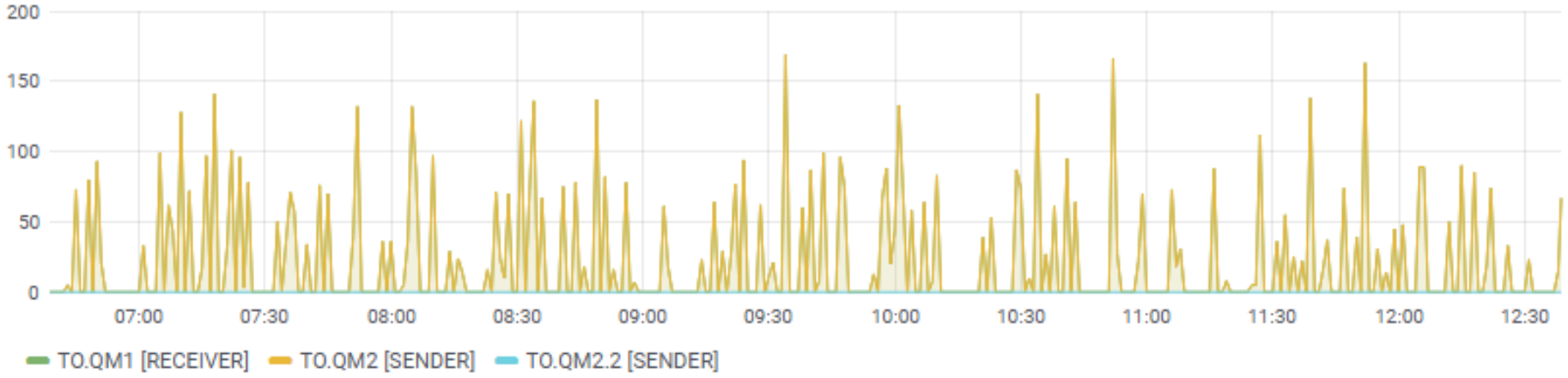
Monitoring - channels

Channels

Channel Status

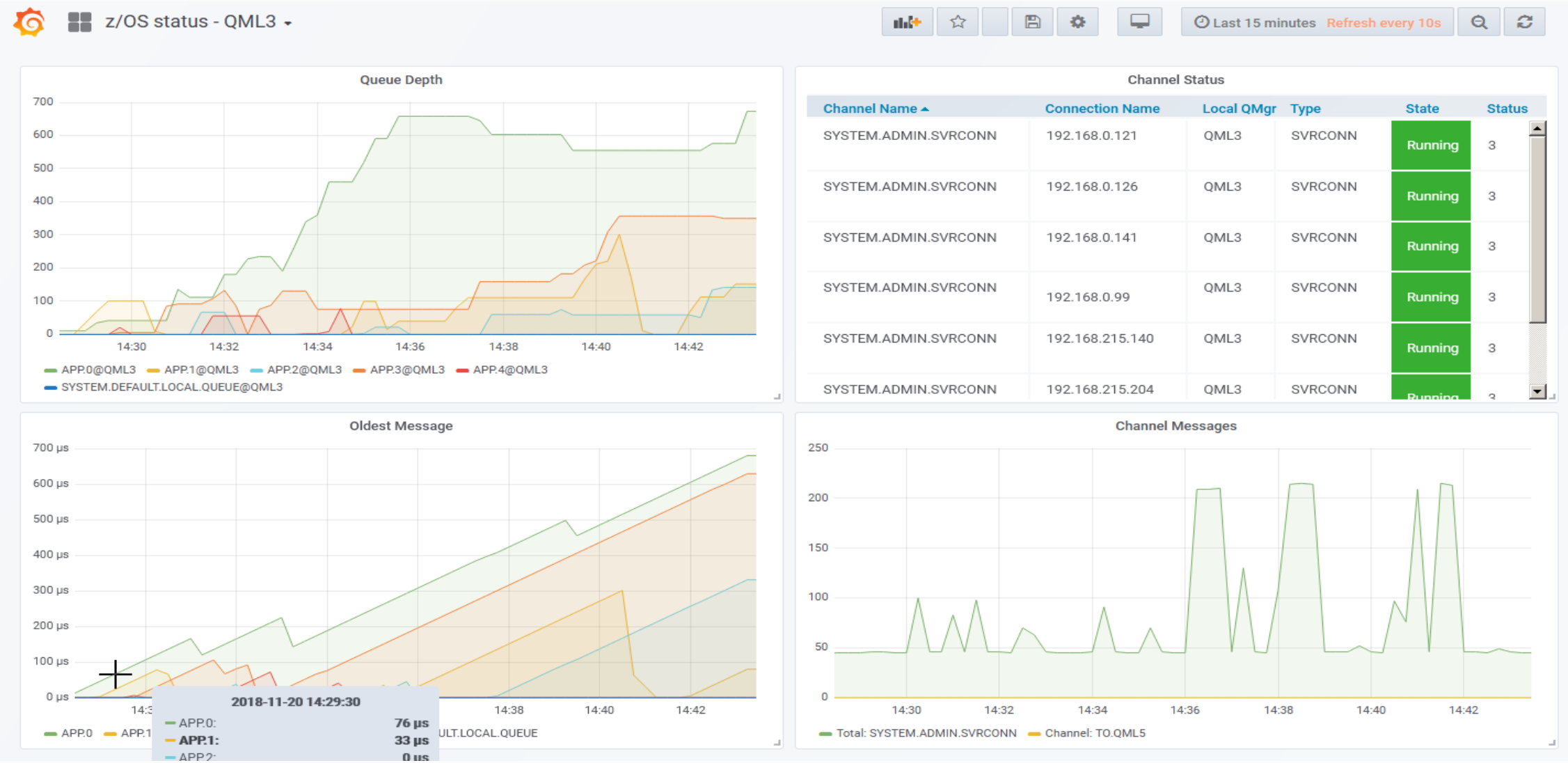
Channel Name	Connection Name	Local QMgr	Remote QMgr	Type	State	Status
TO.QM1	127.0.0.1	QM1	QM2	RECEIVER	Running	3
TO.QM2	127.0.0.1(1415)	QM1	QM2	SENDER	Running	3
TO.QM2.2	localhost(1415)	QM1		SENDER	Transition	5

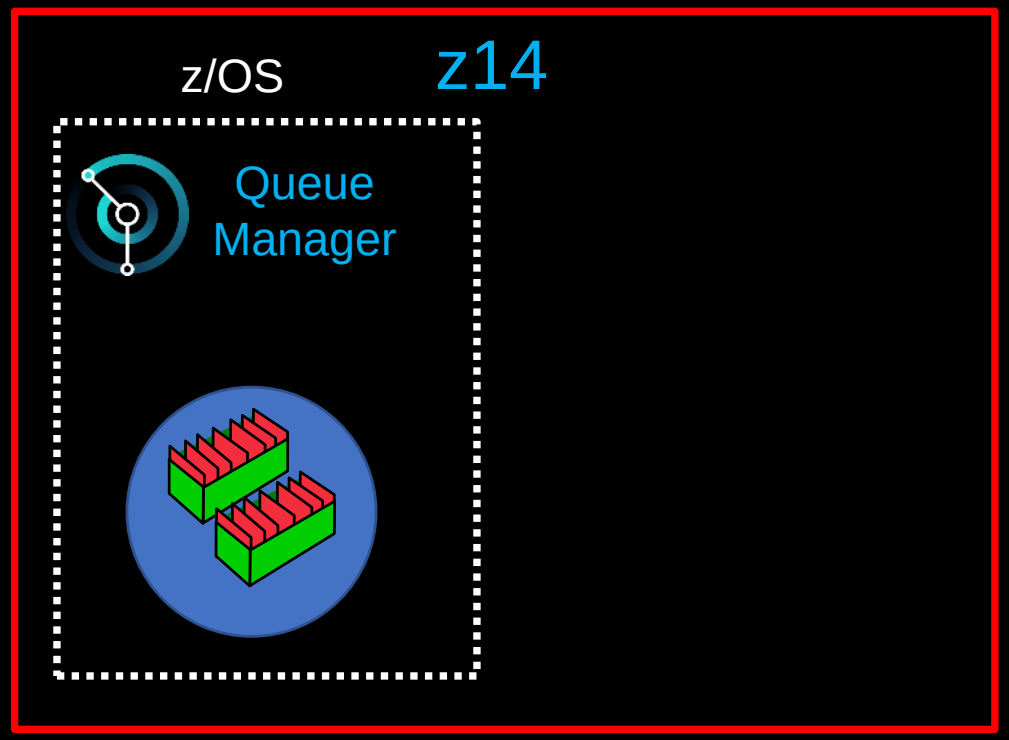
Channel Messages

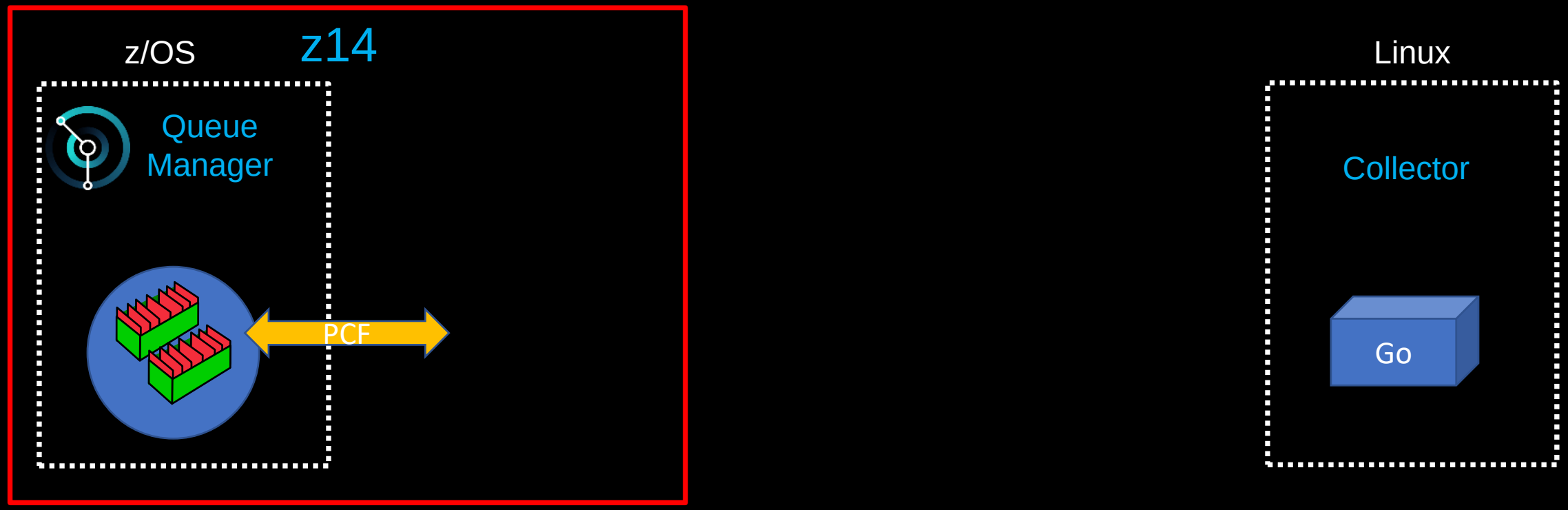


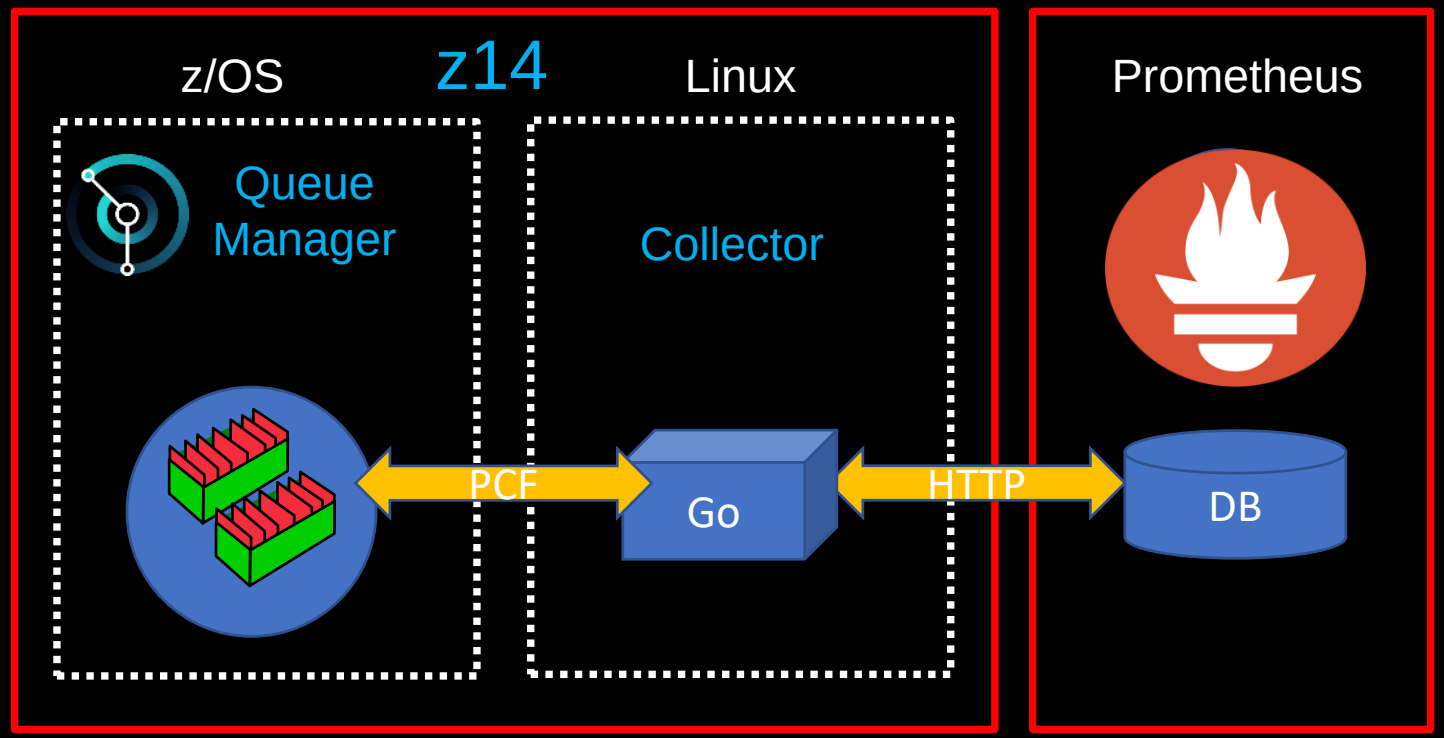
Requirement: Monitor z/OS

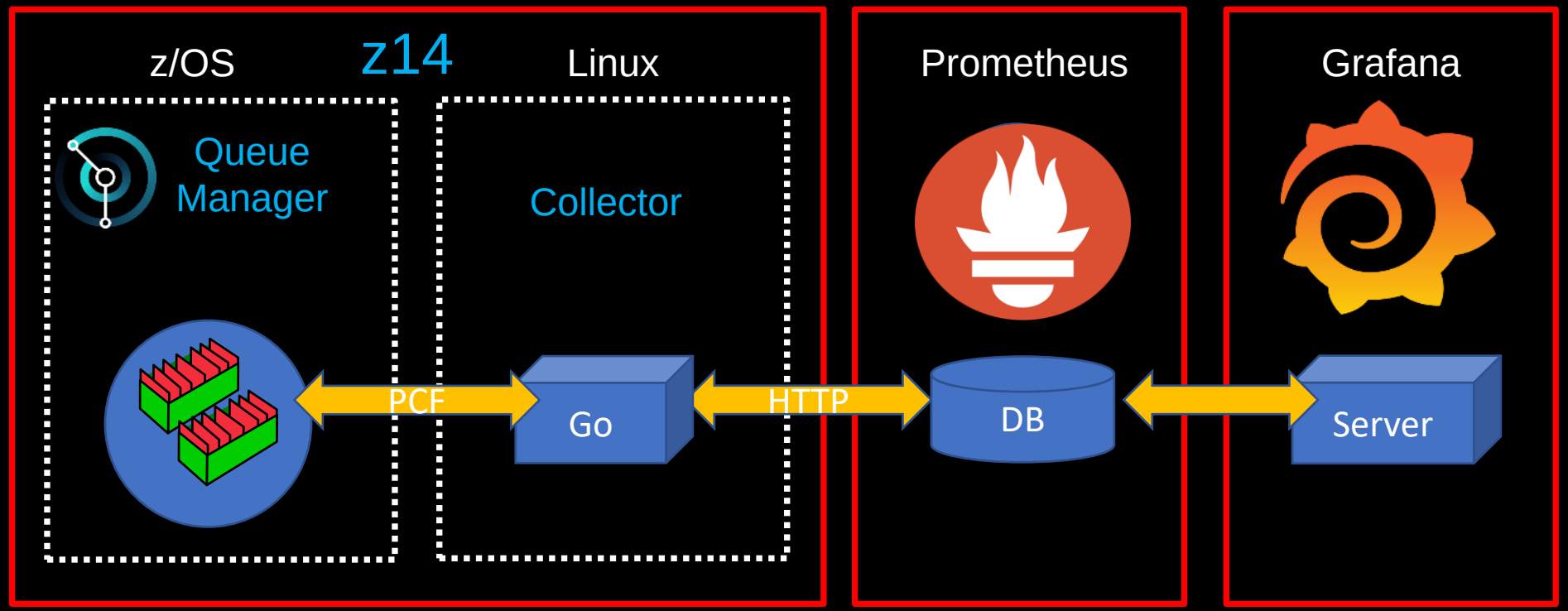
Monitoring for z/OS

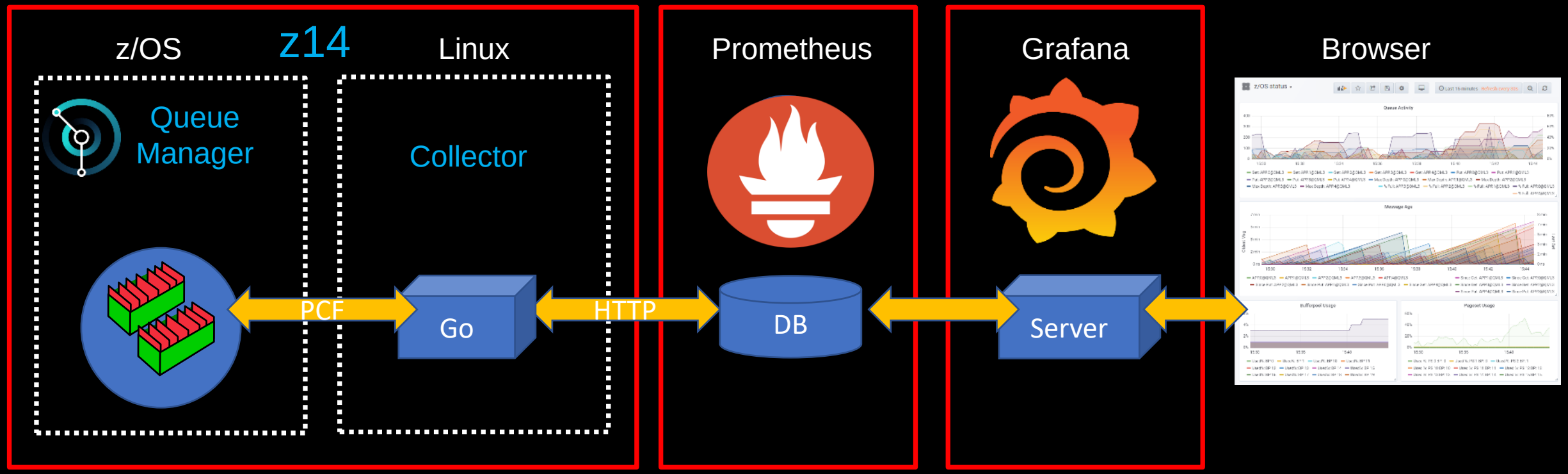



z/OS
z14
Queue
Manager


z/OS
z14
Queue
Manager
PCF
Linux
Collector
Go


z/OS
z14
Linux
Prometheus
Queue Manager
Collector
PCF
Go
HTTP
DB

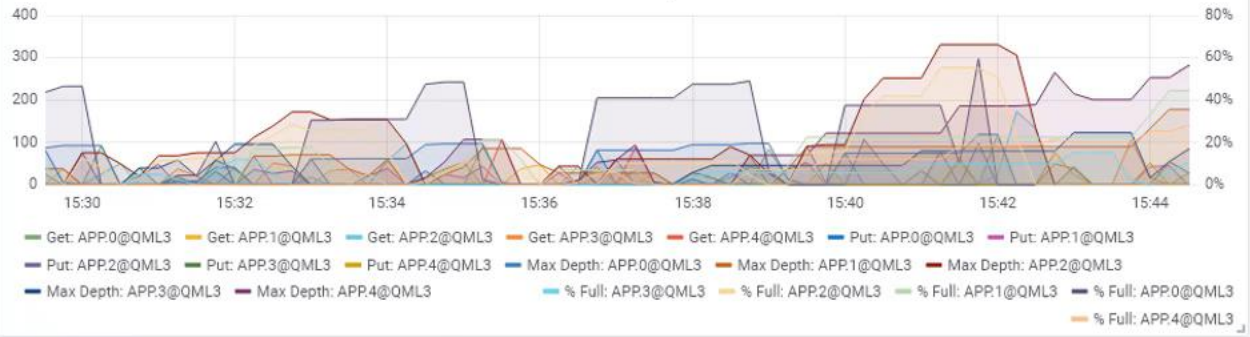

z/OS
z14
Linux
Prometheus
Grafana
Queue Manager
Collector
PCF
Go
HTTP
DB
Server


z/OS
z14
Linux
Queue Manager
Collector
PCF
Go
HTTP
Prometheus
DB
Grafana
Server
Browser

z/OS status

Last 15 minutes Refresh every 30s

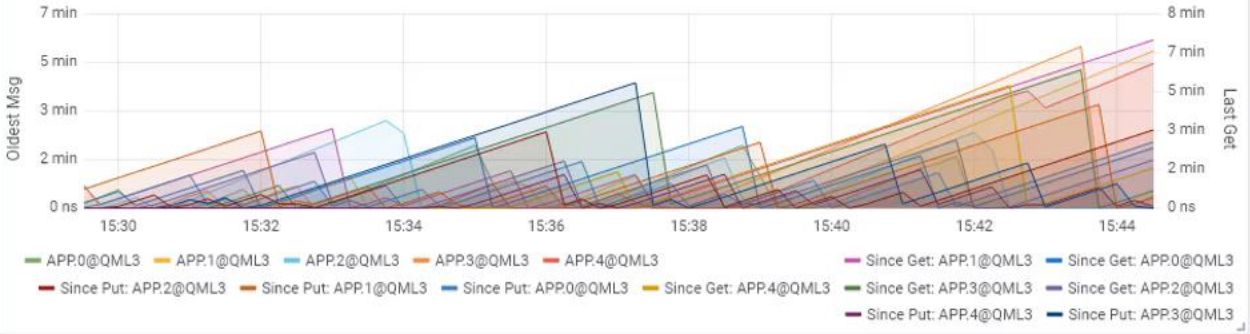
Queue Activity



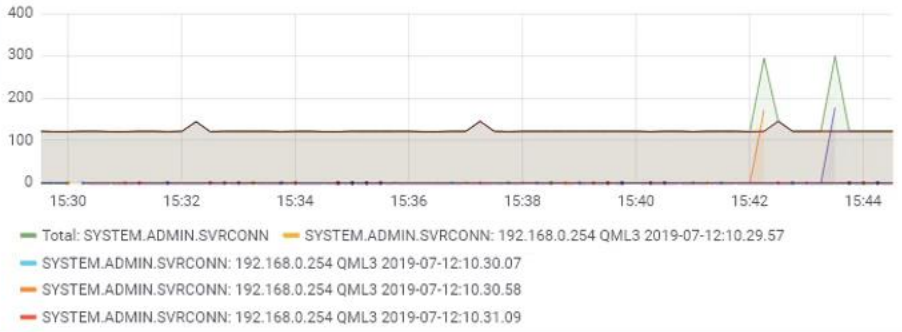
Channel Status

Channel Name	Connection Name	Local QMgr	Type	State	Status
SYSTEM.ADMIN.SVRCONN	192.168.0.254	QML3	SVRCONN	Running	3
SYSTEM.ADMIN.SVRCONN	192.168.17.254	QML3	SVRCONN	Running	3
TO.QML5	192.168.17.252	QML3	CLUSSDR	Transition	5

Message Age



Channel Messages

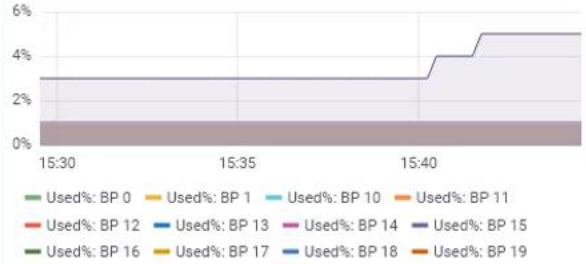


Unique Channels

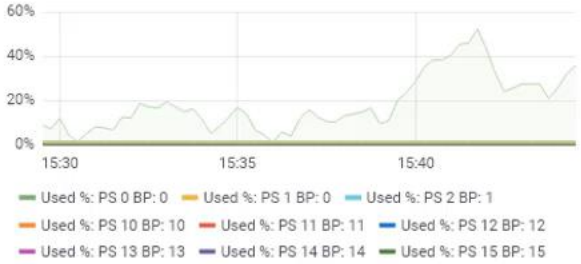
Unique Channels

46

Bufferpool Usage



Pageset Usage



BP/PS Usage

Bufferpool	Page Set	Queue Manager	PS Used	BP Used
3	4	QML3	35.6%	5.0%
0	0	QML3	1.6%	1.0%
0	1	QML3	1.0%	1.0%
2	3	QML3	0.3%	1.0%
1	2	QML3	0.2%	1.0%

No longer reliant on resource publications

- The monitors now extract information from STATUS commands
  - Queues, Channels, QMgr, Topics, Subscriptions
- And USAGE for Bufferpools and Pagesets
- Giving key information for ALL platforms including z/OS

New – June 2020

- All collectors brought to same functional level
- Can use YAML file as alternative to command line options

```yaml
prometheus:
  port: 9157
  metricsPath: "/metrics"
  namespace: ibmmq
global:
  useObjectStatus: true
  useResetQStats: false
  logLevel: INFO
  rediscoverInterval: 1h
connection:
  queueManager: QM1
  replyQueue: SYSTEM.DEFAULT.MODEL.QUEUE
objects:
  queues:
  - APP.*
  - "!SYSTEM.*"
  queueSubscriptionSelector:
  - PUT
  - GET
```

Additional info

- GitHub source: https://github.com/ibm-messaging/mq-metric-samples

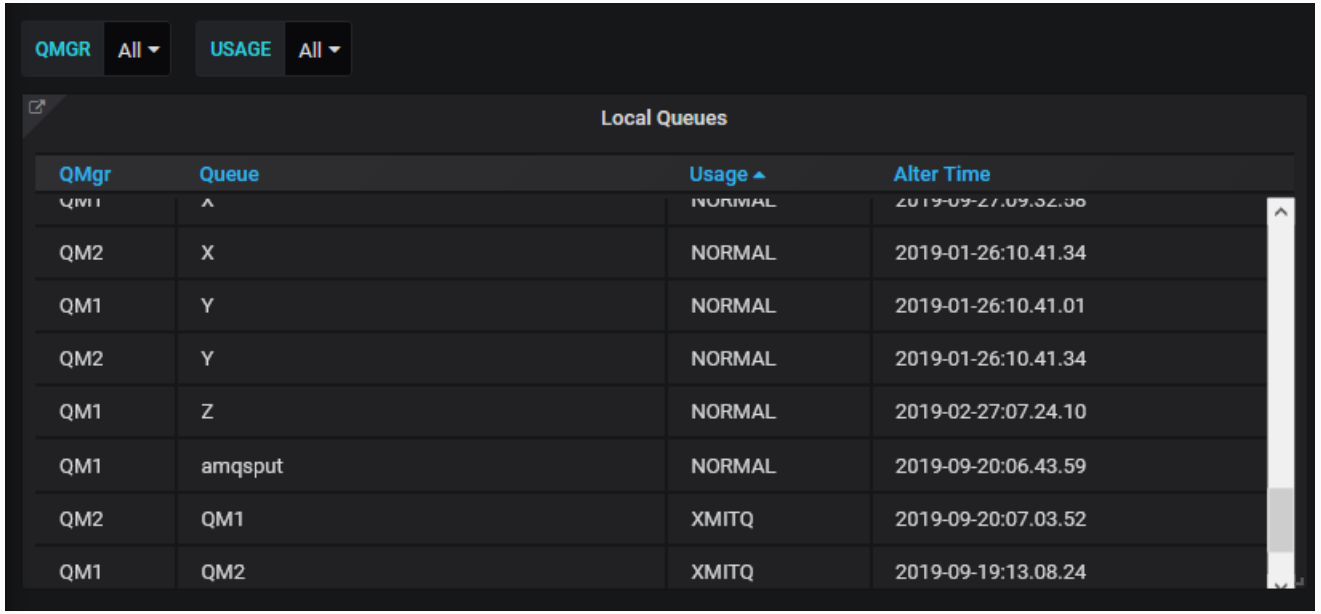
- z/OS monitoring video (2019): https://youtu.be/dCTTZWh5NDw
- Original features video (2016): https://youtu.be/Pi_jHCiqTgU

- RFEs that (if implemented) would help get more data from z/OS:
  - 135074: Queue Statistics
  - 134864: Publish SMF

Connecting metrics to configuration

- Grafana can also display information from some SQL databases
- It is possible to populate tables with MQ configuration information
  - For example, which queues are associated with particular bufferpools and pagesets
- The MQ REST API gets responses in JSON to be directly inserted
  - Databases are able to do SELECT on JSON-formatted columns

See https://marketaylor.synology.me/?p=532



QMgr	Queue	Usage	Alter Time
QM1	X	NORMAL	2019-09-27:09:32:58
QM2	X	NORMAL	2019-01-26:10.41.34
QM1	Y	NORMAL	2019-01-26:10.41.01
QM2	Y	NORMAL	2019-01-26:10.41.34
QM1	Z	NORMAL	2019-02-27:07.24.10
QM1	amqsput	NORMAL	2019-09-20:06.43.59
QM2	QM1	XMITQ	2019-09-20:07.03.52
QM1	QM2	XMITQ	2019-09-19:13.08.24

QMGR All USAGE XMITQ

Local Queues

QMgr	Queue	Usage	Alter Time
QM2	QM1	XMITQ	2019-09-20:07.03.52
QM1	QM2	XMITQ	2019-09-19:13.08.24
QM1	QM2.RETRY	XMITQ	2019-09-30:14.49.41
QM1	QM2.STOPPED	XMITQ	2019-06-11:09.53.22

Query MySQL

A

```
SELECT doc->>"$.queueManager" as 'QMgr',
       doc->>"$.a.queue" as 'Queue',
       doc->>"$.a.usage" as 'Usage',
       concat(doc->>"$.a.altdate",":",doc->>"$.a.alttime") as "Alter Time"
FROM MQCFG.QLOCAL
WHERE doc->>"$.a.queue" NOT LIKE 'SYSTEM.%' AND (
  CASE WHEN $USAGE = 'ALL' THEN
       doc->>"$.a.usage" LIKE '%'
  ELSE
       doc->>"$.a.usage" = $USAGE
```

Format as Table Show Help Generated SQL

Adding resource statistics to your own applications

- Article showing how to publish similar statistics from your own applications
  - And therefore have monitors such as these showing status
  - Even if your apps are connecting to a z/OS queue manager

- Based on the MQ Salesforce Bridge code
  - Shows how to construct the PCF metadata describing your resources

- See https://marketaylor.synology.me/?p=380

What are differences? Which is best?

- Differences are generally in
  - The names and formats of metrics ("ibmmq_queue_mqget")
  - Naming for individual resources such as the queue name
  - Query capabilities to select and display chosen metrics
    - Can you use wildcards on object names
  - Creating labels on graphs
    - Can it be automatic based on the query?
  - Alerting capabilities
- The best is going to be whatever you are already using!
  - But I found the Prometheus/Grafana combination to be flexible and usable
- No easy way to report as string (eg "STARTED", "STOPPED" status)
  - Have to do a mapping via an integer or label

Squashing Metrics

Type	State	Status
RECEIVER	Running	3
SENDER	Running	3
SENDER	Transition	5

```
ibmmq_channel_status_squash+0
```

Legend format: legend format

```
ibmmq_channel_status+0
```

Legend format: legend format

Value #A
channel
qmgr
Value #B
Value #C
type

Options

Apply to columns named	Value #A
Column Header	State
Render value as link	☐

Type

Type	String
Sanitize HTML	☐
Preserve Formatting	☐

Value Mappings

Type: Value to text

0	Stopped
1	Transition
2	Running

Thresholds

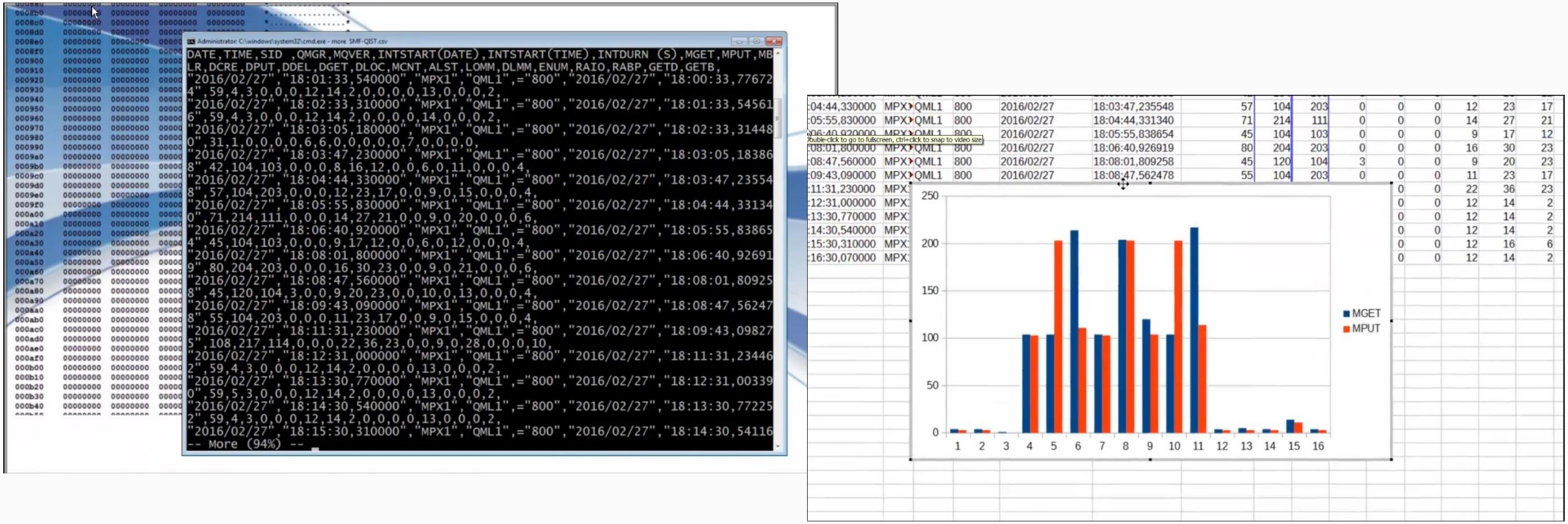
Thresholds	1,2
Color Mode	Cell
Colors	Invert

Latest Go/Prometheus status

- All repositories under https://github.com/ibm-messaging
- mq-golang has the core MQI and PCF packages
  - Some sample code to demonstrate use of most functions
  - Assumes you already know the MQI principles from another language
- mq-metric-samples has Prometheus, Cloudwatch etc monitor programs
  - Along with a "vendor" tree
- Both have scripts to compile programs in Docker containers for ease of build

Similar resource data available on z/OS but via SMF

- mqsmfcsv … open source tool to format MQ z/OS SMF records for easy import to spreadsheets and databases
  - http://github.com/ibm-messaging/mq-smf-csv



Example queries

- What was my largest message size retrieved for this queue?
  - SELECT MAX(Get_Max_Msg_Size) from MQSMF.WQ where (Base_Name= ‘LYNS.TEST.QUEUE');
  - Result was 11,189 (application people insisted it was 3,800)
- How many MQPUTs and MQPUT1s were completed?
  - SELECT SUM (Put_Count), SUM (Put1_Count) from MQSMF.WQ where (Base_Name = ‘LYNS.TEST.QUEUE');
  - Results:

Properties | SQL Results | Data Project Explorer

	1	2
1	0	3121

And we can now do it in JSON

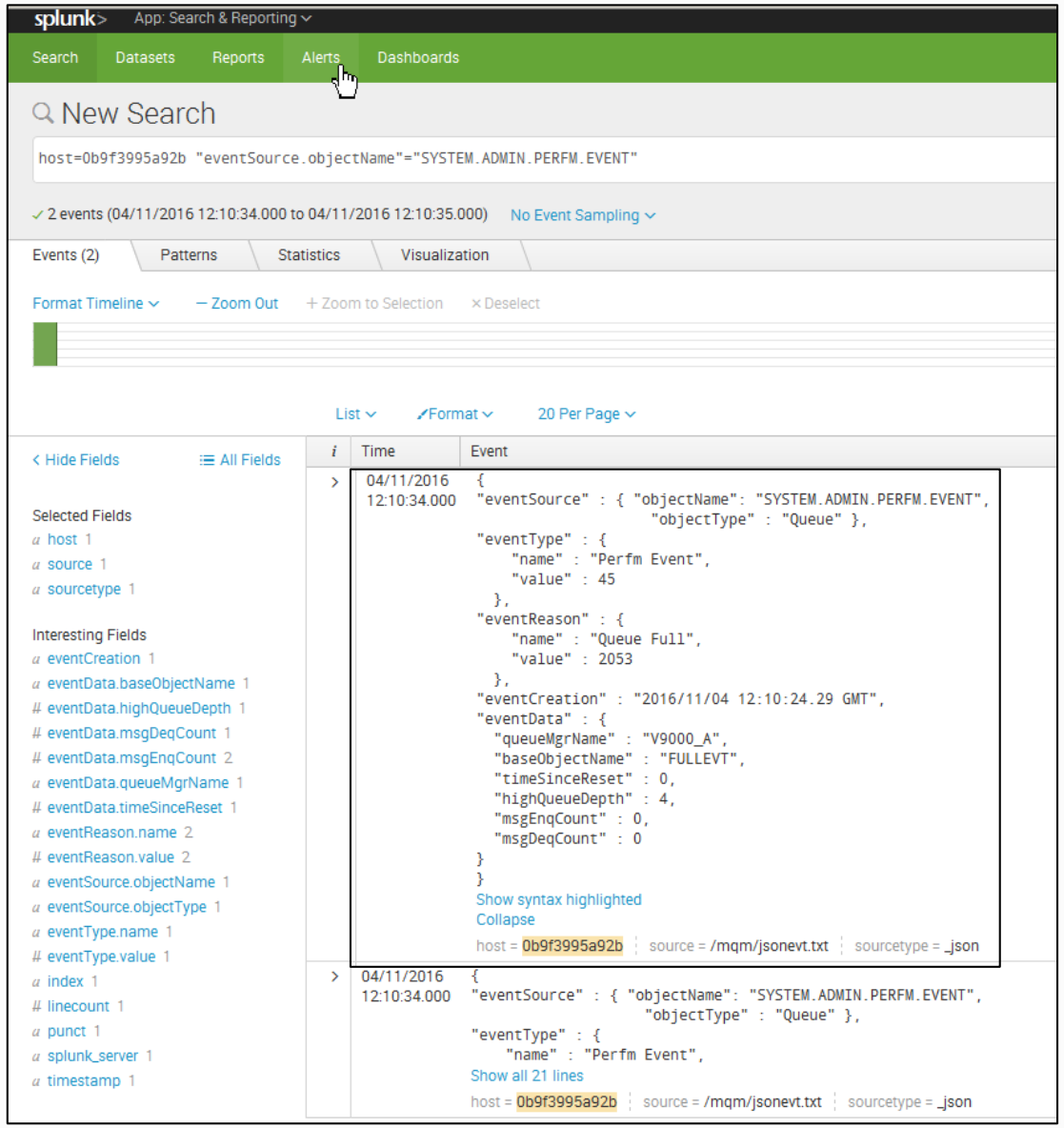
- mqsmfcsv -i <input file> -f json


```
{
"recordType" : 116,
"recordSubType" : 0,
"structure" : "QMAC",
"date" : "2015/11/23",
"time" : "11:00:00.020000",
"lpar" : "H019",
"qmgr" : "MQPC",
"mqVersion" : "800",
"authorisationId" : "IMS     ",
"correlId" : "F0F2F3F6C2C3F1E4C4D6C340",
"connectionName" : "PRDC    ",
"operatorId" : "PLN1231 ",
"applicationType" : "IMS MPP/BMP",
"accountingToken" :
"0000000000000000000000000000000000000000",
"networkId" :
"D7D9C4C340404040044E0A0800000001",
...
```

Processing other MQ events

- Already shown amqsevt as shipped in MQ V8
- It now also supports JSON output option
  - Included from V9.1
- Can be used to feed JSON consumers such as splunk

MQ events in splunk



Using JSON event formatter with Activity Events

- Use the event formatter to output in JSON and then filter it further
  - Run via "service" if local qmgr
  - Could also use subscribe variant to obtain trace

```
amqsevt -m QM1 -q SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE -o json | jq -r -f jqFilt
```

- And then get one-line CSV output of key fields

```
"amqsput","2018-07-11","08:16:48","Connx",0,"N/A"
"amqsput","2018-07-11","08:16:48","Open",0,"QL1"
"amqsput","2018-07-11","08:16:48","Put",0,"QL1",48,"414D512056393030305F4120202020205B2B77"
"amqsput","2018-07-11","08:16:48","Close",0,"QL1"
"amqsput","2018-07-11","08:16:48","Disc",0,"N/A"
"amqsget","2018-07-11","08:16:48","Connx",0,"N/A"
"amqsget","2018-07-11","08:16:48","Open",0,"QL1"
"amqsget","2018-07-11","08:16:48","Get",0,"QL1",38,"414D512056393030305F4120202020205B2B77"
"amqsget","2018-07-11","08:16:48","Get",2033,"QL1",250059,0
"amqsget","2018-07-11","08:16:48","Close",0,"QL1"
"amqsget","2018-07-11","08:16:48","Disc",0,"N/A"
```

https://marketaylor.synology.me/?p=373

A jq filter

```
select(.eventData.activityTrace != null) | .eventData.applName as $applName |
      (.eventData.activityTrace[] |
         [
           $applName, .operationDate, .operationTime, .operationId,.reasonCode.value,
           if (.objectName | length) > 0
           then
              .objectName
           else
              "N/A"
           end,
           if .operationId == "Get" or .operationId == "Put" or .operationId == "Put1"
           then
              .qmgrOpDuration, .msgId
           else
              empty
           end
         ]
      ) |
    @csv
```

Accounting records


```
{"eventSource":{"objectName":"SYSTEM.ADMIN.ACCOUNTING.QUEUE","objectType":"Queue"},
"eventType":{"name":"Accounting MQI","value":167}, "eventReason":{"name":"None","value":0},
"eventCreation":{"timeStamp":"2018-07-25T20:46:19Z","epoch":1532551579},
"eventData":{"queueMgrName":"V9000_A","startDate":"2018-0725",
"startTime":"21.46.16","endDate":"2018-07-25","endTime":"21.46.19","commandLevel":905,
"connectionId":"414D514356393030305F4120202020205B2B779523E91D95","sequenceNumber":0,
"applName":"amqsevt","processId":35389620,"threadId":1,"userIdentifier":"metaylor","connDate":
"2018-07-25", "connTime":"21.46.16","discDate":"2018-07-25", "discTime":"21.46.19",
"opens":[0,3,0,0,0,0,0,0,0,0,0,0,0],
"opensFailed":[0,0,0,0,0,0,0,0,0,0,0,0,0],
"closes":[0,3,0,0,0,0,0,0,0,0,0,0,0],
"closesFailed":[0,0,0,0,0,0,0,0,0,0,0,0,0],
"puts":[24,0],
"putsFailed":0,
"put1s":[1,0],
"put1sFailed":0,
"putBytes":[2547972,0],
…
```

Multiple consumers for MQ events

- Traditional MQ events (queue full etc) are put to a specific named queue
- Makes it difficult to have multiple consumers for same event queue
  - Many monitors can be configured to "browse" but who does "get" and when?
- The MQ event queues can be redefined as topic aliases
- Monitor programs can then get independently from their own dedicated queues
  - I might then run Omegamon AND the JSON variant of amqsevt to different consoles

```
DELETE QLOCAL(SYSTEM.ADMIN.CHANNEL.EVENT)     PURGE
DELETE QLOCAL(SYSTEM.ADMIN.PERFM.EVENT)       PURGE

DEFINE QALIAS(SYSTEM.ADMIN.CHANNEL.EVENT)TARGET(SYSTEM.ADMIN.EVENT) TARGTYPE(TOPIC)
DEFINE QALIAS(SYSTEM.ADMIN.PERFM.EVENT)  TARGET(SYSTEM.ADMIN.EVENT) TARGTYPE(TOPIC)

DEFINE TOPIC(SYSTEM.ADMIN.EVENT) TOPICSTR('SYSTEM/ADMIN/EVENT')
DEFINE QLOCAL(SYSTEM.ADMIN.SUBSCRIBED.EVENT)
DEFINE SUB(SYSTEM.ADMIN.EVENT) TOPICOBJ(SYSTEM.ADMIN.EVENT) +
  DEST(SYSTEM.ADMIN.SUBSCRIBED.EVENT)
```

MQ REST Administration

- Enabling further management options
  - Easy access from any language
  - Scriptable via curl
- Many MQSC commands have REST equivalent
  - Others supported via generic command
  - V9.1.3 adds true JSON-formatted generic commands
- Can manage older qmgrs via proxy qmgr

```
C:\Program Files\IBM\Latest902\bin>curl -k "https://localh
{"queue": [{
  "name": "Q.LOCAL",
  "status": {
    "currentDepth": 0,
    "lastGet": "",
    "lastPut": "",
    "mediaRecoveryLogExtent": "",
    "monitoringRate": "off",
    "oldestMessageAge": -1,
    "onQueueTime": {
      "longSamplePeriod": -1,
      "shortSamplePeriod": -1
    },
    "openInputCount": 0,
    "openOutputCount": 0,
    "uncommittedMessages": 0
  },
  "type": "local"
}]}
```

From V9.0.5 "What's New and Changed"

Version 9.0.5 introduces various improvements to the management and output of error logs. The main changes are that you can:

- Log diagnostic messages, using additional file services and syslog on UNIX platforms, as well as AMQERR01.LOG.
- Use JSON for the description of the messages, as well as the existing format; see JSON format diagnostic messages.
- Reformat a log into another language or style; see mqrc.

For more information, see Diagnostic message services, and QMErrorLog service.

https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.0.0/com.ibm.mq.pro.doc/q130630_.htm#q130630___errlog

JSON Error (aka DiagnosticMessage) Logs

- In same directory as classic error files
- Files AMQERRxx.json
- Unix systems can also direct entries to syslog
  - Which has a lot of backends and routing options


```
{
"ibm_messageId": "AMQ5051I",
"arith_insert_2": 1,
"comment_insert_1": "LOGGER-IO",
"ibm_datetime": "2017-11-16T09:54:26.331Z",
"ibm_serverName": "QM1",
"type": "mq_log",
"host": "machine.somewhere.ibm.com",
"loglevel": "INFO",
"module": "amqzmut0.c:1650",
"ibm_sequence": "1510826066_332014693",
"ibm_processId": 7846,
"ibm_threadId": 4,
"ibm_version": "9.0.4.0",
"ibm_processName": "amqzmuc0",
"ibm_userName": "somebody",
"ibm_installationName": "Installation3",
"ibm_installationDir": "/opt/mqm",
"message": "AMQ5051I: The queue manager task
'LOGGER-IO' has started."
}
```

Consuming tools

- See, for example, Elasticsearch module
  - https://www.elastic.co/guide/en/beats/filebeat/master/filebeat-module-ibmmq.html#filebeat-module-ibmmq

How to configure syslog with MQ 9.1

- This example from AIX. Other Unix platforms will be similar
- In /etc/syslog.conf

```
# MQ writes to the "user" facility
user.debug /var/mqm/errors/syslog.log  rotate size 1m files 4 compress
```

- In queue manager's qm.ini

```
DiagnosticMessages:
  Name=DiagSyslog
  Service=Syslog
  Ident=mqseries
  Severities=I+
```

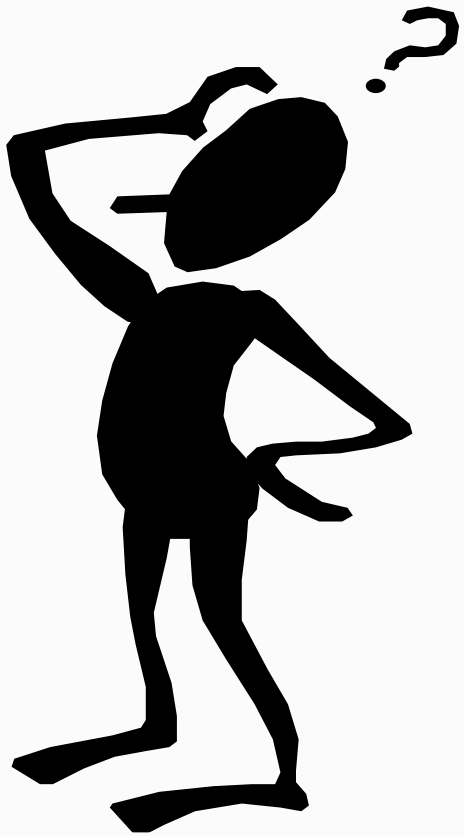
- Make sure syslog.log exists, then restart syslogd

```
Aug 14 15:58:50 example user:info mqseries: {"ibm_messageId":"AMQ9411I",
"ibm_arithInsert1":0, "ibm_arithInsert2":0,"ibm_datetime":"2018-08-14T14:58:50.250Z",
"ibm_serverName":"V9100_A", "type":"mq_log", "host":"example.hursley.ibm.com",
"loglevel":"INFO","module":"amqrrmfa.c:2108", "ibm_sequence":"1534258730_251676000",
"ibm_qmgrId":"V9100_A_2018-06-27_11.13.46", "ibm_version":"9.1.0.0", "ibm_processName":
"amqrrmfa", "ibm_userName":"metaylor", "ibm_installationDir":"/usr/mqm",
"message":"AMQ9411I: Repository manager ended normally."}
```

Summary

- MQ can be easily integrated with a variety of tools
- The pub/sub model for statistics makes it easy to add new consumers
  - Without disrupting any existing monitors
  - And makes it possible to add your own producers
- Using github for repository of code enables easy modification and sharing

  github.com/ibm-messaging
- And blog posts for documenting what we have done
- Ability to use JSON as a common format for all operations



Any questions?