IBM MQ on z/OS & Distributed: Are they like oil and water?

Mark Taylor

marke_taylor@uk.ibm.com

IBM Hursley

Lyn Elkins

elkinsc@us.ibm.com

IBM Advanced Technology Group

Introduction

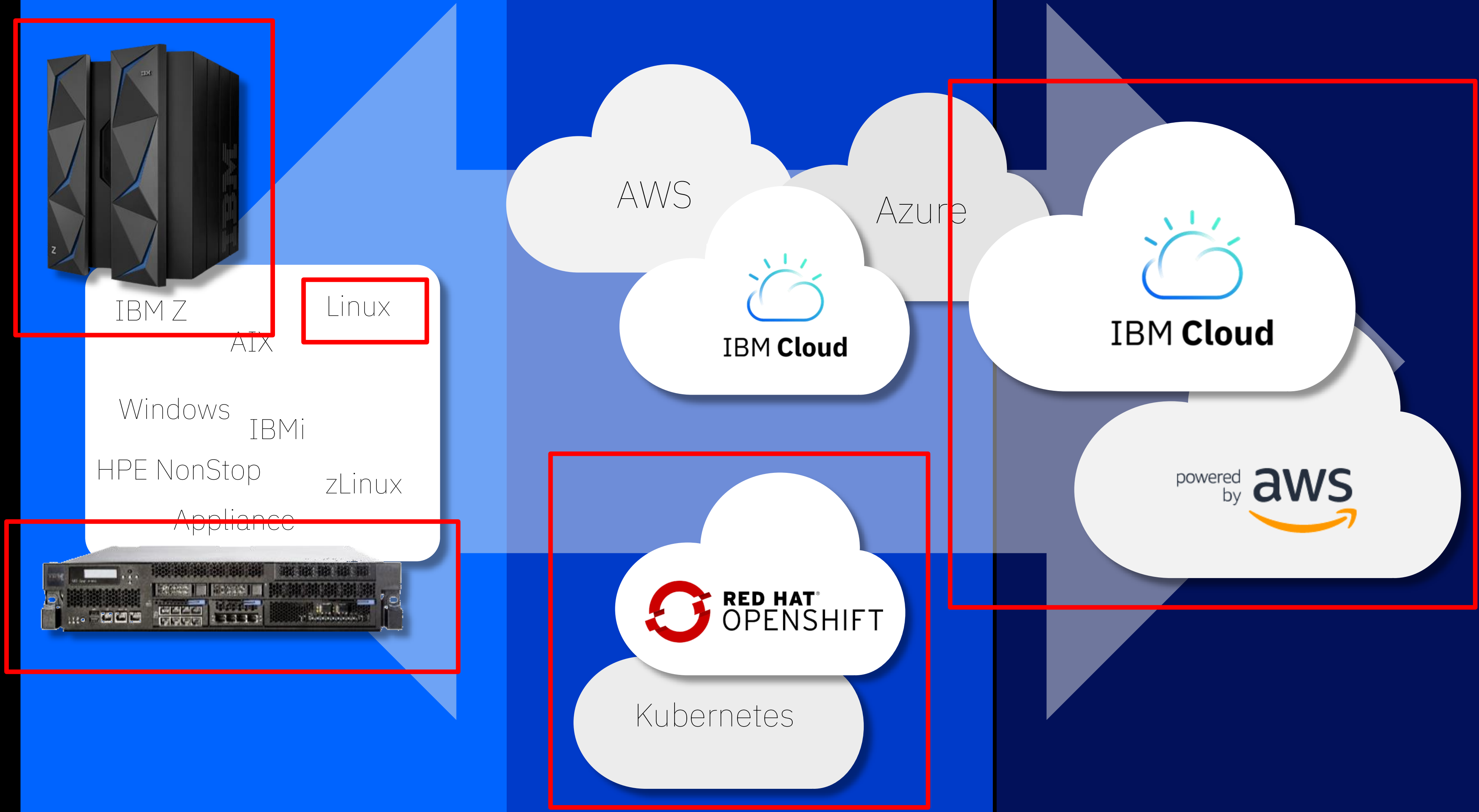
- One objective of MQ is isolating apps from needing to understand platforms
  - There is a common API that can be expressed in many languages
- Another objective is to have (reasonably) common operational model
  - Much of admin is the same on all platforms
- But it’s not all the same
  - One dichotomy has always been whether to be natural to MQ-ness or behave like other things on the platform
  - Some features don’t make sense on some platforms
    - For example, .Net interface is only on Windows (and from 9.1.2, on Linux with .Net core)
  - Some features have not been implemented everywhere for other reasons
- So there are differences, and that is what this presentation will cover
- This is based on V9.2

Run IBM MQ in any location or cloud, exactly as you need it

On-premise, software and the MQ Appliance

Run MQ yourself in public or private clouds

Let IBM host MQ for you with its managed SaaS MQ service in public clouds,
IBM Cloud and AWS



Code Streams

- There are essentially two implementations of MQ from the Hursley lab
  - z/OS
  - Distributed (Windows, Unix, Linux, i)
  - There are some further subspecies variants like VSE or NSS
- Within Distributed implementation, there are some platform unique features
  - Newer environments demand some unique features
  - Most platform-unique code abstracts OS facilities like locking or NLS or threads
- Internal architecture (eg tasks, threads) very different
  - But we won't discuss much of that. Understanding externals is more important
- In the early days, some code was written for one and then “ported”
  - In particular, the channel code
  - Meant double-fixing, and re-porting for each release
- Some code is now truly common
  - Just one copy of the source part shared between both
  - New features generally use common code where feasible

Sections

- Setting up
- Application Programming
- Administration

How this presentation works

- Lyn will talk about z/OS in this color

- Mark will talk about Distributed in this colour

Setting Up

Getting started

- Lots of differences in initial installation and setup

- Getting the code on the box is part of the job
  - MQ uses native installation techniques for all platforms
  - Needs a suitably-authorised person to do that installation
    - SMPE for z/OS, installp for AIX, rpm for Linux etc
  - Firmware on appliance, container images for cloud

- But other differences primarily due to
  - Security
  - Storage

- Share philosophy of needing no more features than is found on the system
  - So no prereq software for core capabilities of MQ
  - Product ships components that are needed such as gskit or Liberty
  - But can exploit things that we know are there
    - For example, on z/OS we use the system-provided TLS or RRS
  - Some extended capabilities may have additional prereqs
    - Shared Queues need DB2

Security

- On Distributed, MQ implements its own authorisation mechanism
  - There is no generally-accepted standard interface on these systems
- And mostly relies on the existence of certain userids
  - There are differences even between individual platforms
  - Certified container runs under random id!

- On z/OS, MQ exploits the common authorization interface, SAF
  - And so the z/OS security administrator has to be involved
  - Define the profiles etc.

- Will look more at security later on

Storage (Distributed)

- On Distributed, MQ uses directories such as /var/mqm/qmgrs and /var/mqm/logs
  - The system administrator will probably allocate filesystems and mount them
  - These days, likely to have separate SAN administrator
- Each queue has its own file within the filesystem
  - To store the message data
  - Each queue can now hold 255TB (!)
  - From 9.1.5, maximum queue file size is configurable
- Queues do not interfere with each other's storage requirements
  - Subject to max size of filesystem
- Logs can be LINEAR or CIRCULAR
  - Choice made when qmgr is created (though you can switch later)
  - With linear logging, you then used to need a job to remove old log files (can now be automatic)
  - MQ does not directly implement dual-logging; relies on RAID/replicated filesystems
  - Appliance is only circular

Storage (z/OS)

- Private Queues are handled via pagesets and bufferpools
- Multiple queues may use the same pageset and bufferpool
  - Can lead to storage contention
  - V8 increased number of bufferpools to match number of pagesets
- No direct equivalent of circular logging but constraints can be applied to achieve a similar effect
  - Semi-circular?
  - Active logs are ‘almost like circular’, with offloading to archive logs
  - Log shunting
- Logs are managed via the BSDS (Bootstrap Dataset)
- MQ understands and implements Dual Logging
- Tool provided to format and extract messages from log

Shared Queues

- A z/OS-unique feature
  - Multiple queue managers can see the same physical queue
  - Continuous processing of messages from a queue even when one LPAR fails

- Relies on the Coupling Facility hardware
  - And relies on DB2 Data Sharing
  - Messages reside in the CF or in auxiliary storage

- Results in several unique possibilities
  - Inter-qmgr communication without standard channels
  - Dynamic selection of which qmgr to connect to

- Effects appear in many places
  - For example, single MQSC command can be issued to multiple queue managers giving multiple responses

Application Programming

with the MQI

MQI and other APIs

- The MQI is the core application programming interface
- Other APIs are built on top including JMS, XMS
  - Even the REST Messaging API
- The other Object Oriented APIs are either stabilised or deprecated
  - including Base Java classes, C++
- JMS/XMS mostly hide platform differences but do not give access to full capabilities of MQI
  - Though one unique feature is "delayed delivery"
- Understanding how application uses the APIs can be key for performance and PD

General

- Default codepages and encoding differ by platform
- Always use the header files for your platform
  - Don't be tempted to cross-compile
- Maximum lengths of fields may vary
- Language support may vary
  - Assembler only on z/OS
  - Go, Node.JS on Distributed
  - And some APIs: MQAI only on Distributed
- MQI return codes may be different
  - Often because underlying storage mechanisms have different error conditions
  - For example, Coupling Facility errors on shared queues
- z/OS does not – in general - have MQ clients
  - Some parameters to some verbs only apply in client environments
  - For example, the MQCD passed during MQCONNX

API - Connections

- MQCONN/MQCONNX
  - Verbs not required for CICS transactions
    - ➢ MQHC_DEF_HCONN can be used for subsequent verbs in applications
  - ConnTag is available to control serialization
    - ➢ An application (especially an MCA) can tell if another instance of itself is already running
    - ➢ On either the same local qmgr or any other in the QSG
  - Group connection to QSG

  - Lots of client-only options for connection
    - ➢ MQCD can be specified
    - ➢ Reconnect options
  - MQCNO_SHARED options for multi-threaded applications
    - ➢ Controls whether an hConn can be (serially) used by other threads in the same process
  - Fastpath binding
  - Control of accounting
    - ➢ When accounting information is being collected, some apps may request exclusion
  - Application name can be explicitly set

API - Disconnections

- MQDISC
  - Always recommended

  - Rollback when application abends
    - Although definition of "abend" is not clear in every case
    - CICS and IMS do make it clear!
    - A JVM has been known to return OK to the operating system even when the user's code has caused a fatal exception

  - Rollback when not used and application ends

  - Internally, normal rollback (ie not on restart) can be handled entirely from in-memory state
  - Internally, rollback requires reading the log files

API - Objects

- MQOPEN
  - Default dynamic queue names begin with CSQ.* or AMQ.*
  - Distributed can open multiple queues simultaneously via Distribution List
    - ~~"Noone uses this~~"
    - Publish/Subscribe preferred cross-platform model
- MQCLOSE
  - No platform differences in practice
- MQSET
  - Follows the same rules as MQSC attributes for platforms
- MQINQ
  - Follows the same rules as MQSC attributes for platforms

API - Messages

- MQPUT/MQPUT1
  - Messages can be automatically segmented
    - ➢ But Message groups are cross-platform
  - Distributed supports "Reference messages" which can avoid putting large amounts of data on a queue
- MQGET
  - z/OS has "get with signal" to asyncronously notify app when messages appear
    - ➢ MQCB is now preferred cross-platform model
  - z/OS has MARK_SKIP_BACKOUT for simpler processing of poison messages
    - ➢ Bad messages can be moved to an application-specific DLQ while backing out other resource changes
  - Distributed can get portions of messages via segmentation
  - Waiting Getter – z/OS only NP messages; Dist can work with P messages (but both require out of sync)
    - ➢ Internal optimisation (no direct application control) but can lead to unexpectedly skewed workload
- MQSUB
  - No platform differences
- MQSUBRQ
  - No platform differences

API – Flow control

- MQCB
  - Definition of the callback function in MQCBD varies by environment
  - eg C function pointer, CICS program name

- MQCTL
  - Not in IMS adapter
  - In CICS, cannot use MQOP_START – use MQOP_START_WAIT
  - On z/OS, apps must be authorized to use USS to use MQOP_START

- MQSTAT
  - Client applications only
  - But usable regardless of server platform

API - Properties

- MQDLTMP
- MQBUFMH
- MQCRTMH
- MQDLTMH
- MQMHBUF
- MQSETMP
- MQINQMP

- No platform differences

API - Transactions

- MQBEGIN
  - Only available on Distributed local bindings (not client)
  - z/OS always has a transaction manager available
- MQCMIT
  - On all platforms when not running under external TM
- MQBACK
  - On all platforms when not running under external TM

- Default for MQ transactional behaviour **is different**
  - MQI on Distributed assumes NO_SYNCPOINT
  - MQI on z/OS assumes SYNCPOINT
  - Always specify syncpoint options on MQI calls
- Environments for two-phase transactions differ
  - On z/OS, RRS CICS and IMS are all available for transaction management
  - On Distributed, XA is available as the standard interface (JTA for JEE)
    - ➢ And MQ can act as a transaction manager

Administration

Object Definitions

- Attributes and ini files
  - Some items are queue manager attributes on one platform but not other
  - z/OS has lots related to its storage

- Some unique object types
  - z/OS has STGCLASS and CFSTRUCT
  - Distributed has SERVICES and COMMINFO

- Startup
  - CSQZPARM is assembled/linked and other inputs run during startup
    - ➢ Reset configuration, define default objects etc
  - On Distributed, standard objects are created by qmgr creation and updated during migration
    - ➢ Now supports automatic MQSC and ini file operations during qmgr startup

Object Attributes – Queue Manager

- Apart from shared queue and storage-related attributes …
  - Various events differ as shown in other charts
  - Some z/OS attributes are available in qm.ini for Distributed
    - In particular MaxChannel, MaxActiveChannel
- z/OS only
  - EXPIRYINT
  - CHIDISPS, CHIADAPS
  - ACTCHL, MAXCHL
  - ADOPTCHK, ADOPTMCA
  - DNSGROUP, DNSWLM
  - GROUPUR, IGQ, IGQAUT, IGQUSER
  - LSTRTMR,LU62ARM, LU62CHL, OPORTMIN, OPORTMAX, RCVTMIN, RCVTTYPE, TCPCHL, TCPNAME, TCPKEEP, TCPSTACK
  - SCYCASE
  - SSLTASKS
  - TRAXSTR, TRAXTBL
- Distributed only
  - ACCTCONO, ACCTINT, ACCTMQI, ACCTQ
  - ACTIVREC
  - CCSID
  - CERTVPOL
  - CHAD
  - SCHINIT, SCMDSERV
  - IMGINTVL, IMGLOGLN, IMGRCOVO, IMGRCOVQ, IMGSCHED

Object Attributes – Queues and Channels

- Apart from shared queue and storage-related attributes …
- Queue
  - DEFTYPE(SHAREDYN) z/OS
  - HARDENBO only effective on z/OS
  - INDXTYPE z/OS
  - DISTL
  - NPMCLASS
  - SCOPE Distributed only, but obsolete
  - STATQ
  - MAXFSIZE
  - TriggerData for transmission queues can be blank on Distributed, names channel on z/OS
- Channel
  - CONNAME 48 characters on z/OS, 264 elsewhere
  - Format of exit names and exit data is platform-specific
  - PUTAUT ONLYMCA/ALTMCA
  - KAINT only effective on z/OS
  - Default protocol for channels still LU62
    - ➢ Distributed don't need to use TRPTYPE on many MQSC commands

Queue Manager operations

- Message Expiry
  - z/OS has explicit config for timing of task to remove expired messages
  - Distributed has a similar task but no documented configuration
- Security Cache Scavenger
  - z/OS has parameters to control authority cache lifetime
  - No equivalent on Distributed; use REFRESH SECURITY explicit command
- Storage Scavengers
  - z/OS has tasks to release bufferpool and pageset storage
  - Distributed will release queue file storage at intervals
- Queue Indexing
  - z/OS has explicit indexes on queues to assist with retrieval patterns
  - Distributed has hashing to perform similar role but no documented configuration

Intercommunication and Clusters

- Channels are essentially the same
  - Still claims to support some obsolete protocols

- Clustering is essentially the same across all platforms
  - Including capability for multiple cluster transmission queues

- Uniform Clusters from V9.1 (automatic client rebalancing)

- MQTT, AMQP channels

Security (1)

- Authentication
  - Distributed supports userid validation in operating system and explicit LDAP
  - z/OS supports userid validation in operating system

- SSL/TLS configuration
  - Distributed (mostly) qmgr uses gskit toolkit
  - z/OS qmgr uses System SSL
  - Can lead to discrepancies in crypto algorithms supported
    - ➢ As example, there was lag between all platforms supporting TLS13
  - Different versions support different algorithms
  - And there may also be client-related discrepancies
    - ➢ Java programs rely on JSSE implementations – CipherSpec/Suite names vary by JRE
    - ➢ .Net programs usually use Microsoft inbuilt implementations
    - ➢ NSS uses OpenSSL
  - There is a good common overlap between all of these, but not identical sets
  - Current versions significantly reduce the options on all platforms after vulnerabilities & deprecations

Security (2) – Access Control

- z/OS
  - Uses system-provided interface for authorization
    - ➢ SAF is common API to RACF, Top Secret, ACF2
  - Has to work with the 4 permissions available in SAF
  - No distinction between PUT and GET
    - ➢ Often alias queues are used to isolate permissions
  - Granular control of "impersonation" (setting context, alt-user)
  - One operation may result in several authorization queries

- Distributed
  - MQ-provided authorisation interface
    - ➢ Implemented in the OAM – OS or LDAP-based
  - Many permissions on objects
  - Global controls on impersonation
    - ➢ If you have authority to use alt-user, there are no constraints on which user
  - Well-known "mqm" id for full authority
    - ➢ Cloud runs slightly differently with no "mqm" id in the container but essentially the same
  - Allows user names that are not known outside of the queue manager

Security (3) – Advanced Message Security

- Protection of at-rest message data defined by policies
  - MQSC SET POLICY, command line setmqspl
  - CSQ0UTIL which then requires a refresh/restart to take effect

- Distributed MQ supports "MCA Interception" for client connections
  - AMS processing done in SVRCONN environment

- z/OS has "MCA Interception" for queue manager connections
  - AMS processing done in CHINIT

Commands

- Basic OS-level operations are different
  - Create, start, stop, delete queue manager procedures
  - Distributed has command-line interface
  - z/OS has JCL

- Issuing configuration commands like ALTER QLOCAL
  - Distributed has runmqsc shell
  - z/OS has ISPF panels for most commands
  - And the +cpf commands for runmqsc equivalence
  - MQ Explorer is product-provided common GUI
  - REST API and MQ Console for administration available on all

- OpenShift environment has MQ Operator
  - Configured with YAML files

- Common programming interface (PCF) for configuration commands
  - z/OS requires an "extended" format which may have multiple sets of responses
    - Supporting a Queue Sharing Group environment
  - Distributed supports the same format but not as the default
  - Differences are hidden in the Java PCF classes

MQSC Commands

- Some commands not available in all platforms
- RESET QSTATS is only on z/OS
  - PCF available on all platforms
- ARCHIVE LOG is only on z/OS even though Distributed also have logs
- DISPLAY SYSTEM
- MOVE QLOCAL
  - Use dmpmqmsg
- DISPLAY QMSTATUS, APSTATUS only on Distributed
- STOP CONN only on Distributed
  - All platforms have STOP CHL to kill client connections
- Some z/OS MQSC have command-line equivalents on Distributed
  - STOP QMGR == endmqm

Monitoring

- Many queue manager event messages are common
  - For example, queue full
- But not every event is on every platform
  - Authorisation, Logging, and Channel auto-definition events are Distributed only
  - IMS Bridge events are only on z/OS
- Recording queue manager and application activity is very different
  - z/OS has SMF 115 and 116 records
  - Distributed has accounting, statistics and application activity events
- Some statistics (eg Generated Messages) may have different definitions
- DIS QSTATUS(*) UNCOM
  - A number
  - Yes/No
- Distributed accounting and stats events are analogous to SMF 116
  - Resource statistics publications are closer to SMF 115
- Application Activity Trace Events

We will talk more about monitoring in the Problem Determination presentation later

Problem Determination

- On Distributed, there are several places to look for PD information
  - Diagnostic logs written to /var/mqm/errors and /var/mqm/qmgrs/<qmgr>/errors
  - Diagnostic logs can be written in JSON
  - FFST written to /var/mqm/errors for serious errors
  - Trace provided by MQ commands and written to /var/mqm/trace

We will talk more about this in the Problem Determination presentation later

- On z/OS, also numerous places to follow the clues:
  - The MSTR and CHIN JES log
    - ➢ Should always be the first place to look
  - MQ API trace (aka user parameter trace) – a GTF trace
  - SMF 115 statistical information
  - SMF 116 class(3) accounting (task related) data
  - A dump for serious problems

Backup

- MAKEDEF and dmpmqcfg are tools to backup configuration
  - From V8, have dmpmqmsg to backup messages

- On Distributed, backup of log files is done by stopping qmgr and copying /var/mqm/log directory
  - rcdmqimg takes images of queues into logs

- On z/OS, full and fuzzy backups of pagesets are supported

- CFSTRUCT backup required for QSG
  - takes image of structure into logs

High Availability and Disaster Recovery

- Shared queues on z/OS for continuous processing
- On Distributed, MQ provides multi-instance
  - Not on z/OS because ARM is provided
- On Linux x64, MQ provides RDQM
  - For HA and DR
  - Very similar to appliance-provided capability (built on same core technology)
- New: NativeHA on CloudPak for Integration
  - Replicated/replayed logs
- Cross-site DR will usually use disk replication for any platform
  - RDQM
  - zHyperwrite for best performance of z/OS

Summary

- Title asks about oil and water

- Perhaps (olive) oil and (balsamic) vinegar is better description
  - Blending together

Any questions?