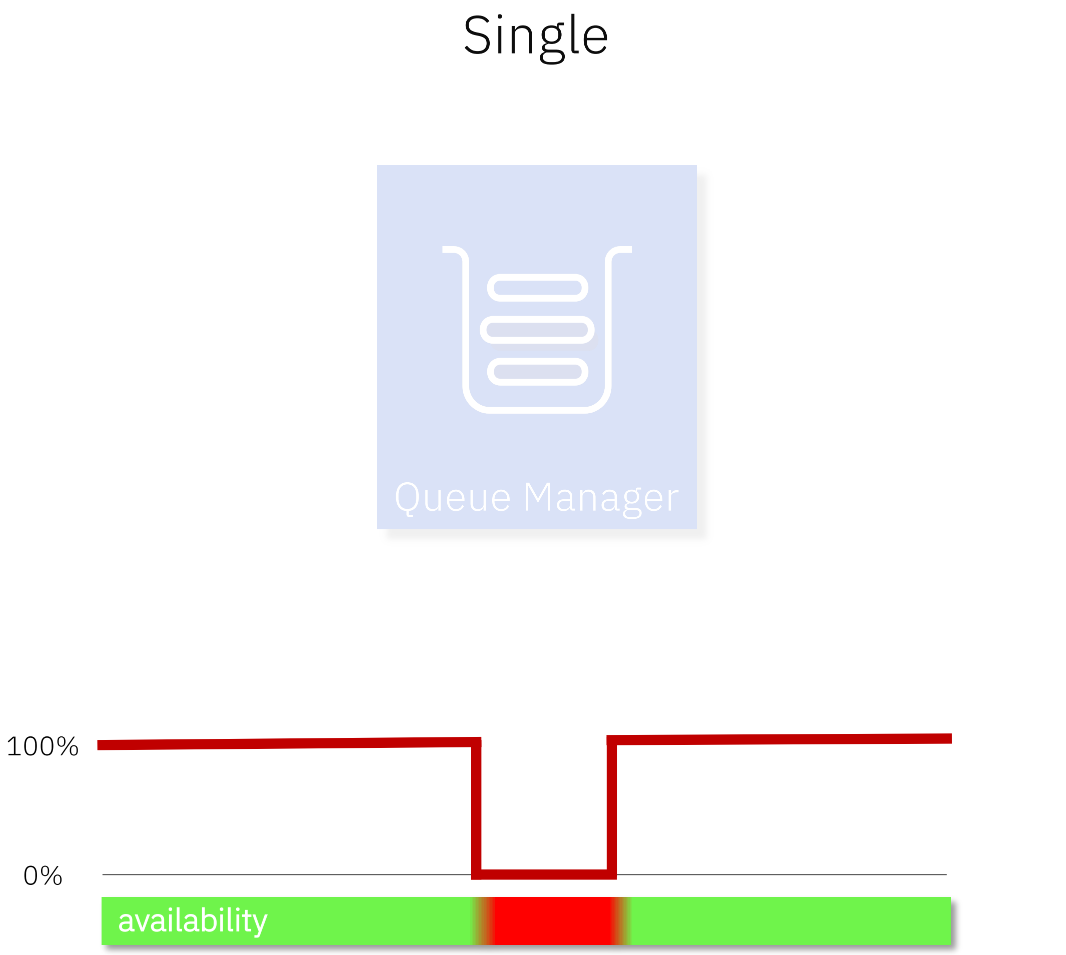
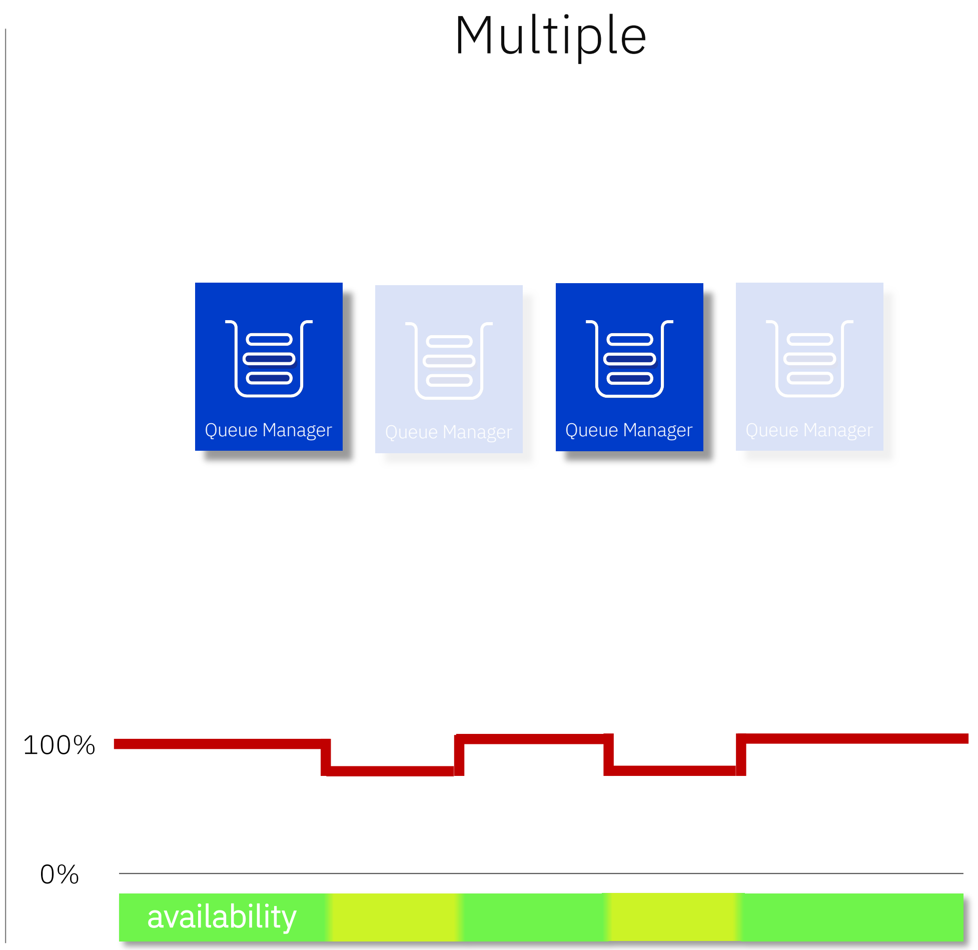
Build a truly fault tolerant and scalable IBM MQ messaging solution

Anthony Beardsmore
IBM MQ Development

Fault Toleration

More is better for availability





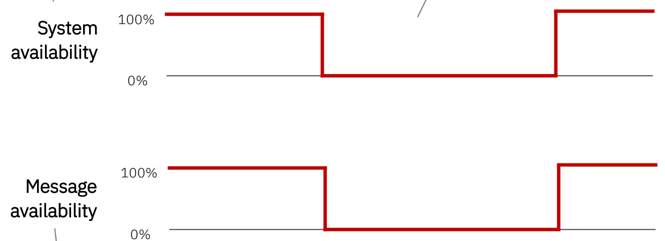
Let’s go through that availability thing one step at a time…

Single, non-HA queue manager

Don't just think about the messaging layer, even with a single queue manager it's better to have multiple instances of an application running. This builds redundancy into the application availability

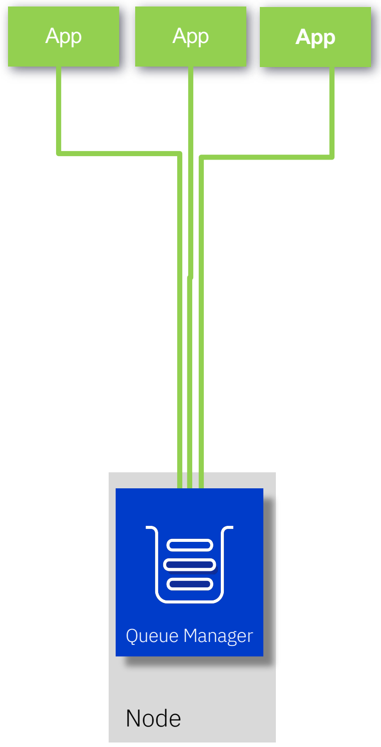
How much messaging work can proceed

While the QMgr is down **all** of the applications need to wait for it to be restarted



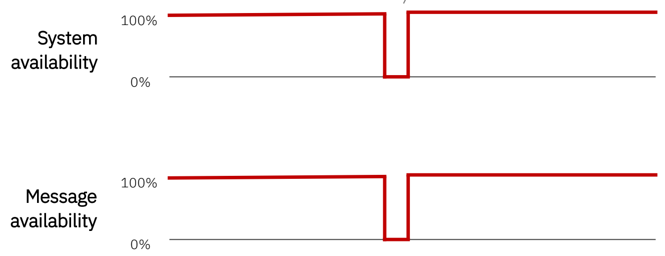
All queued messages require the QMgr to restart

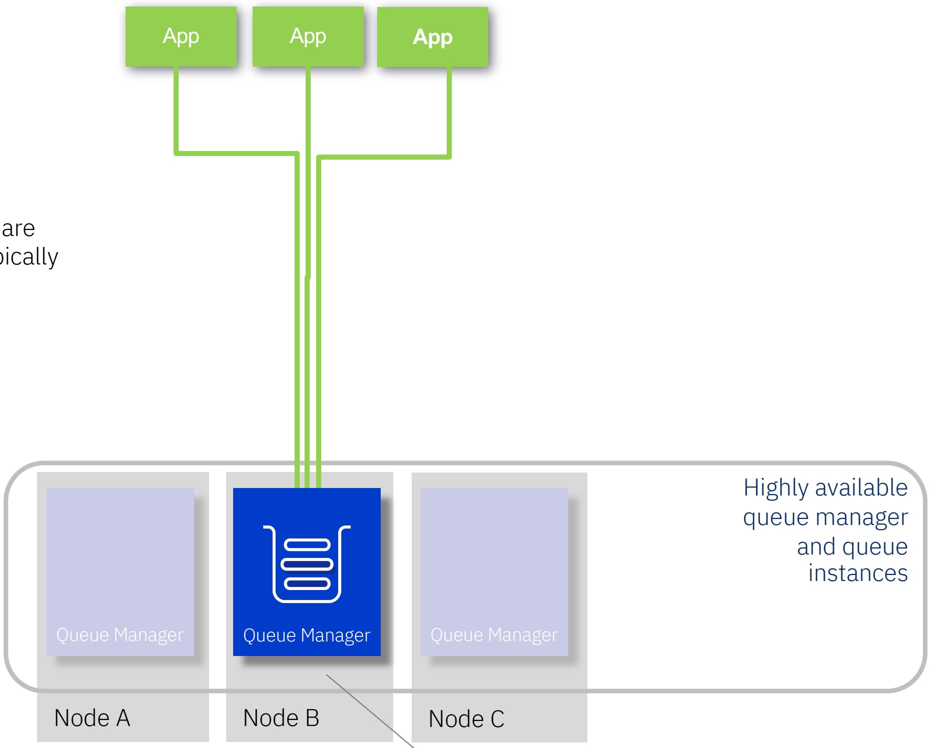
Proportion of queued message that are available



Single HA queue manager

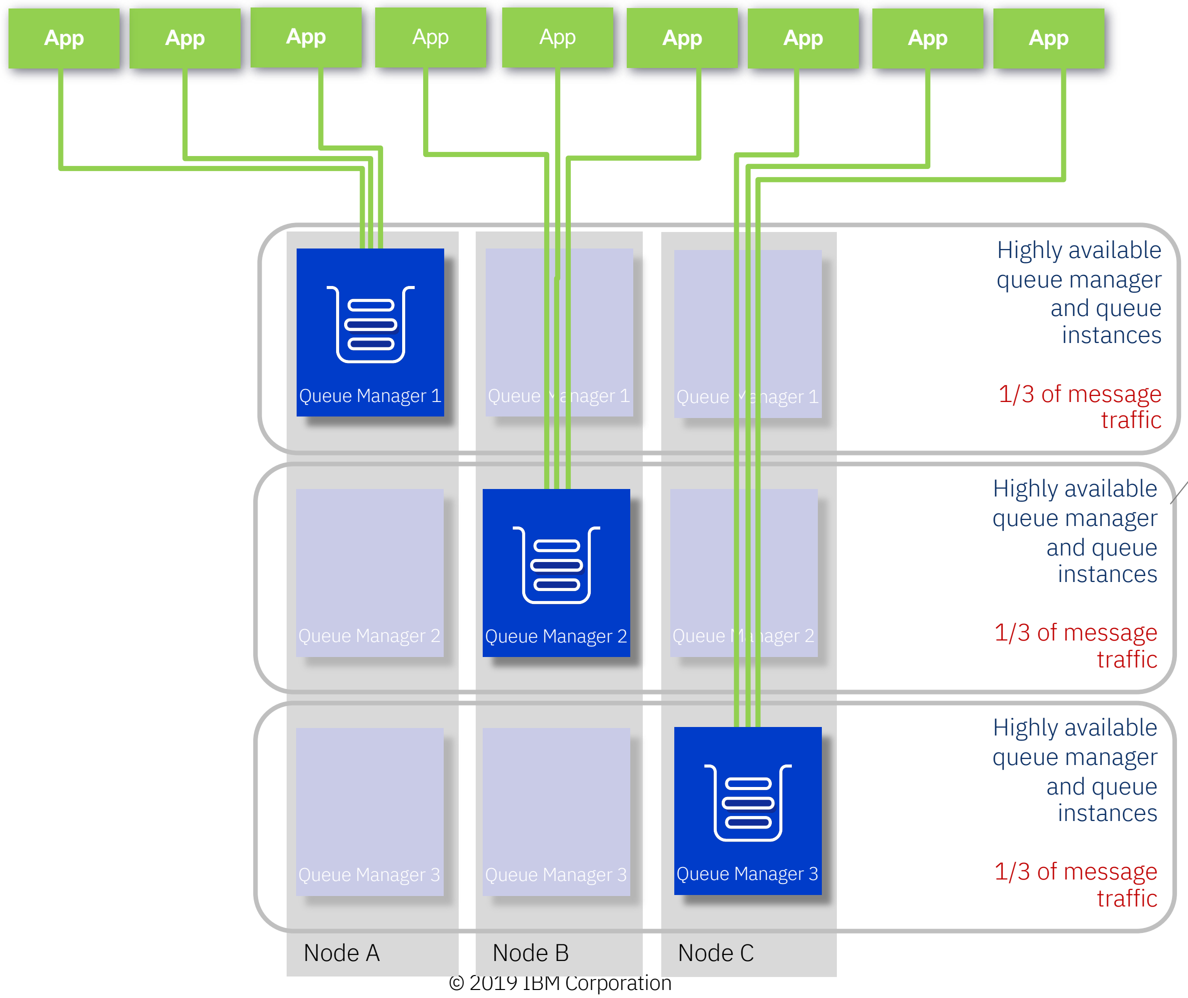
HA queue managers are restarted quickly (typically a ‘few’ seconds)





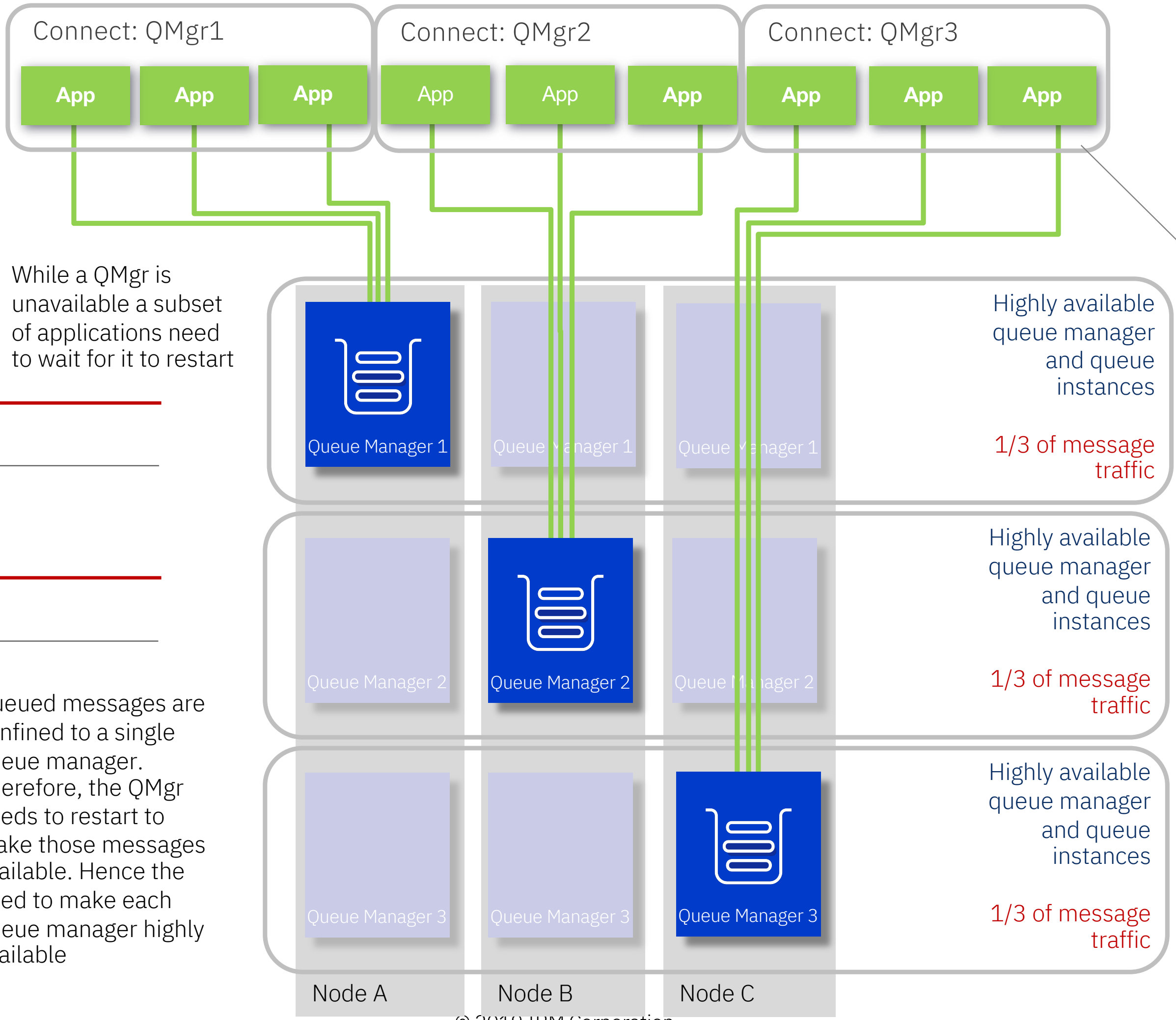
Here we have one active instance of the queue manager with two replica, standby, instances ready to take over.
(This happens to be the MQ RDQM HA model, other solutions like multi-instance queue managers are subtly different (only one standby) but essentially the same)

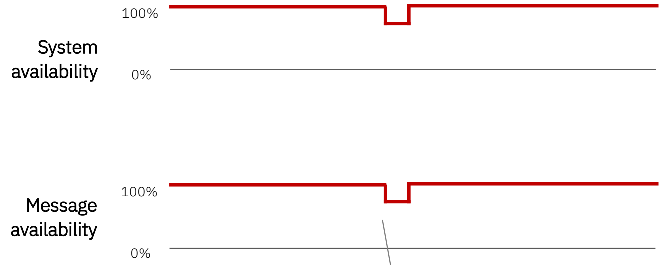
Multiple HA queue managers



To further increase the availability you need to remove the single point of failure that is a queue manager.
For this, create multiple queue managers and stripe the messaging workload across them by defining the "same" queue on all of them.
Each message is only queued on a single queue manager but the multiple queue managers mean any one outage is confined to a subset of the workload

Multiple HA queue managers – **ordered consumption**



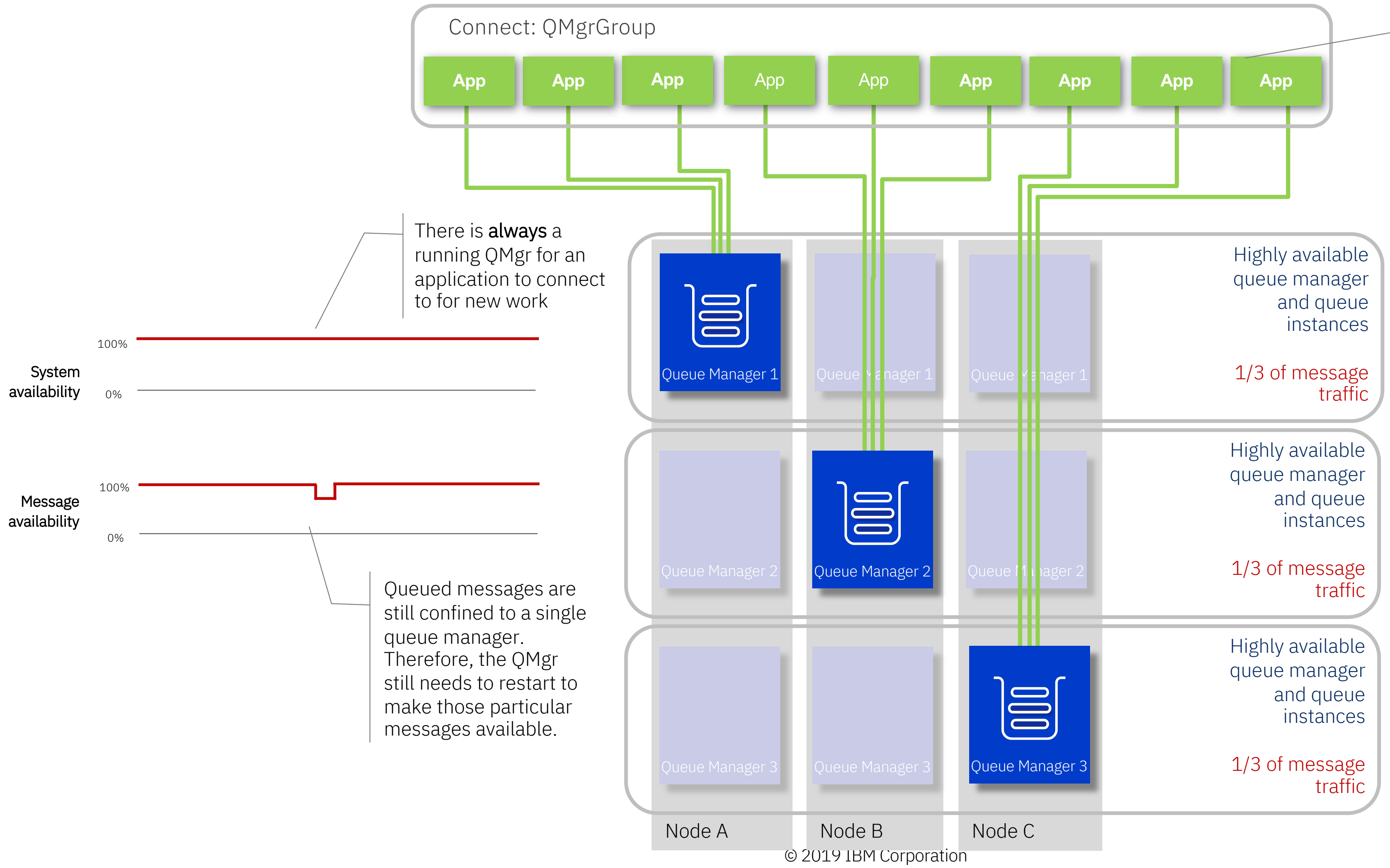


While a QMgr is unavailable a subset of applications need to wait for it to restart

Queued messages are confined to a single queue manager. Therefore, the QMgr needs to restart to make those messages available. Hence the need to make each queue manager highly available

Messages that require ordering need to go to the same queue manager. One approach to achieve this is for application instances to connect to a specific queue manager based on their ordered stream of messages. This ensures messages are processed in order per-group

Multiple HA queue managers – **unordered consumption**



There is **always** a running QMgr for an application to connect to for new work

Queued messages are still confined to a single queue manager. Therefore, the QMgr still needs to restart to make those particular messages available.

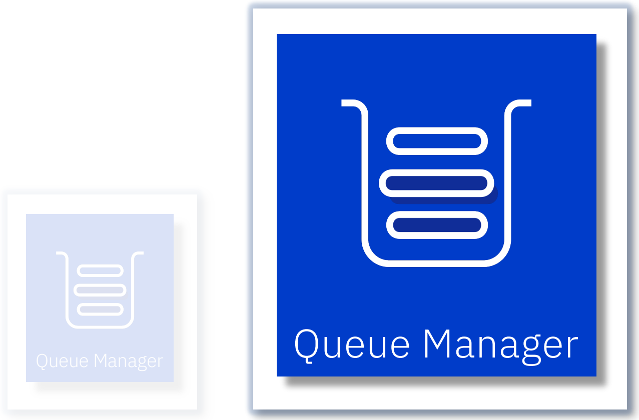
Application instances can connect to any queue manager as the order that they are queued across the multiple instances is not a concern.
Connecting across a group can be achieved with a CCDT queue manager group

It’s not just availability that will benefit…

Scaling

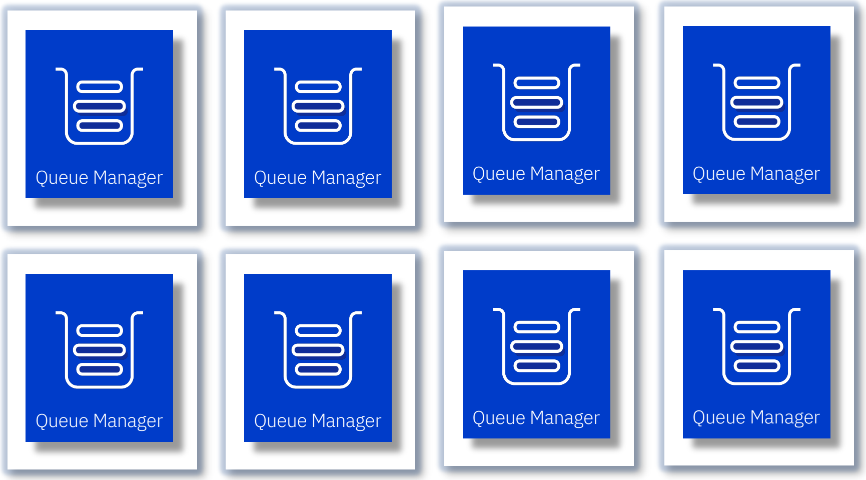
More is better for scalability

Single



x4

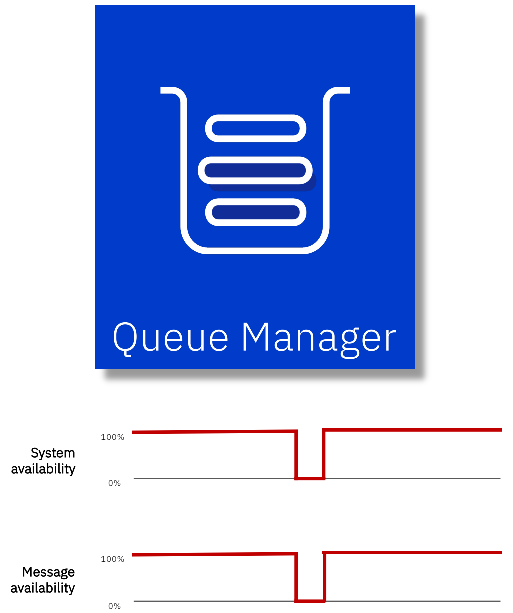
Multiple



xn

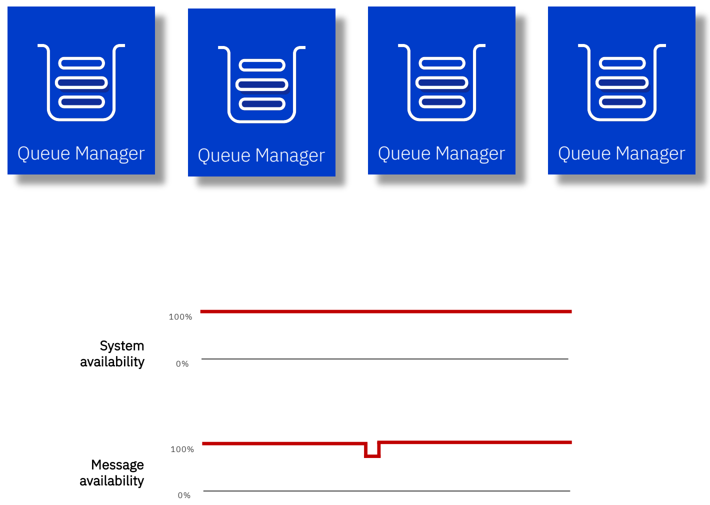
So it’s a no brainer?

Single

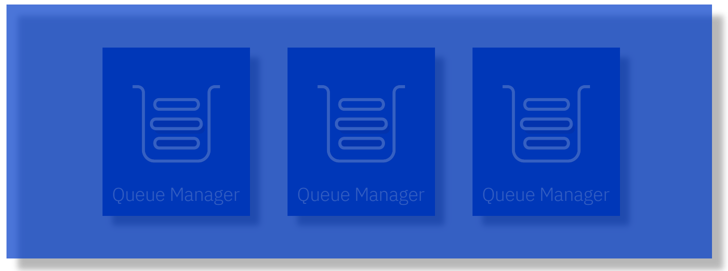


- ○ Simple
- ○ Invisible to applications
- ○ Limited by maximum system size
- ○ Liable to hit internal limits
- ○ Not all aspects scale linearly
- ○ Restart times can grow
- ○ Every outage is high impact

Multiple



- ○ Unlimited by system size
- ○ All aspects scale linearly
- ○ More suited to cloud scaling
- ○ Reduced restart times
- ○ Enables rolling upgrades
- ○ Tolerate partial failures
- ○ Visible to applications – limitations apply
- ○ Potentially more complicated



Try to stop thinking about each individual queue manager and start thinking about them as a ~~cluster~~

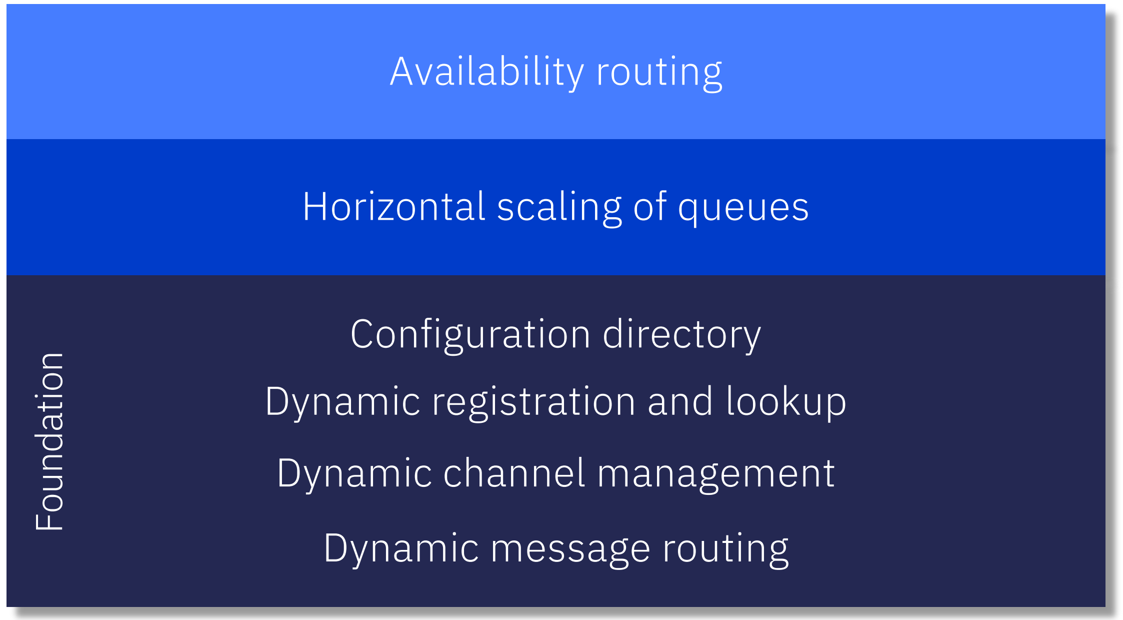
uniform cluster…

The fundamentals on MQ Clusters

(skip this if you know it)

MQ clustering

What MQ Clusters provide :


Availability routing
Horizontal scaling of queues
Foundation
Configuration directory
Dynamic registration and lookup
Dynamic channel management
Dynamic message routing

Horizontal scaling with MQ Clustering

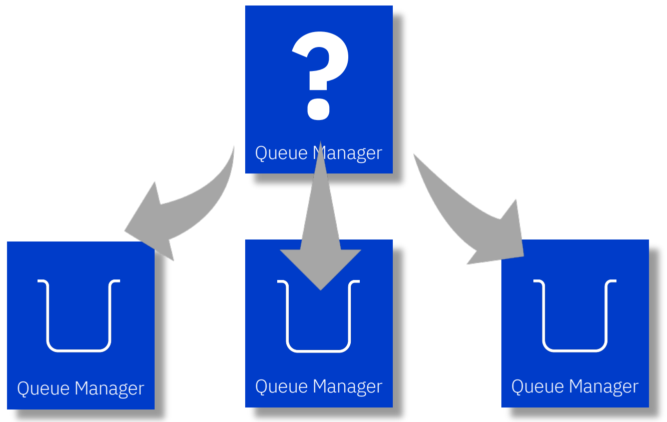
Horizontal scaling with MQ Clustering

Earlier we showed how to scale applications directly across multiple queue managers, with an MQ Cluster you can do that with queue manager-to-queue manager message traffic.

A queue manager will typically route messages based on the name of the target queue

In an MQ Cluster it is possible for multiple queue managers to independently define the same named queue

Any queue manager that needs to route messages to that queue now has a choice…



Channel workload balancing

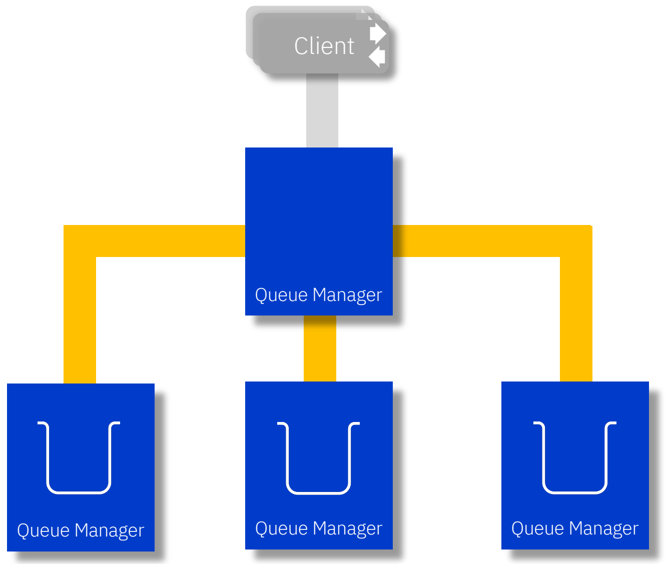
- Cluster workload balancing applies when there are **multiple cluster queues of the same name**

- Cluster workload balancing will be applied in **one of three ways**:
  - When the putting application opens the queue **- bind on open**
  - When a message group is started **- bind on group**
  - When a message is put to the queue **- bind not fixed**

- When workload balancing is applied:
  - The source queue manager builds a list of all potential targets based on the queue name
  - **Eliminates** the impossible options
  - **Prioritises** the remainder
  - If more than one come out equal, **workload balancing** ensues ...

- Balancing is based on:
  - The **channel** – not the target queue
  - **Channel traffic** to **all queues** is taken into account
  - **Weightings** can be applied to the channel

- ... this is used to send the messages to the chosen target



Channel workload balancing

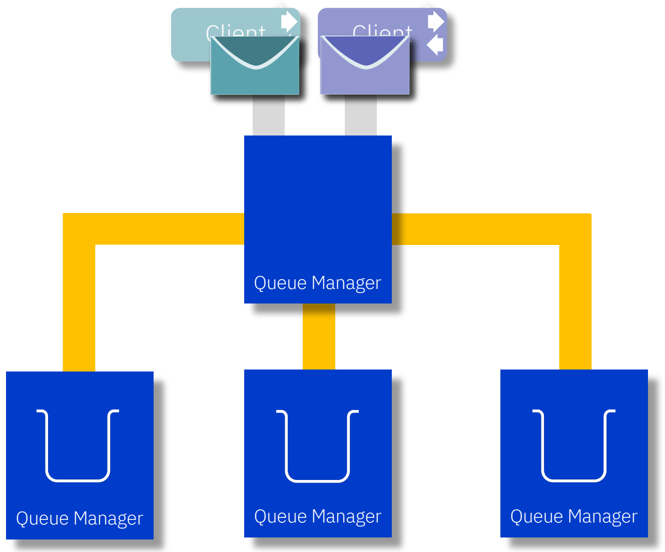
- Cluster workload balancing applies when there are **multiple cluster queues of the same name**

- Cluster workload balancing will be applied in **one of three ways**:
  - When the putting application opens the queue **- bind on open**
  - When a message group is started **- bind on group**
  - When a message is put to the queue **- bind not fixed**

- When workload balancing is applied:
  - The source queue manager builds a list of all potential targets based on the queue name
  - **Eliminates** the impossible options
  - **Prioritises** the remainder
  - If more than one come out equal, **workload balancing** ensues …

- Balancing is based on:
  - The **channel** – not the target queue
  - **Channel traffic** to **all queues** is taken into account
  - **Weightings** can be applied to the channel

- … this is used to send the messages to the chosen target



Channel workload balancing

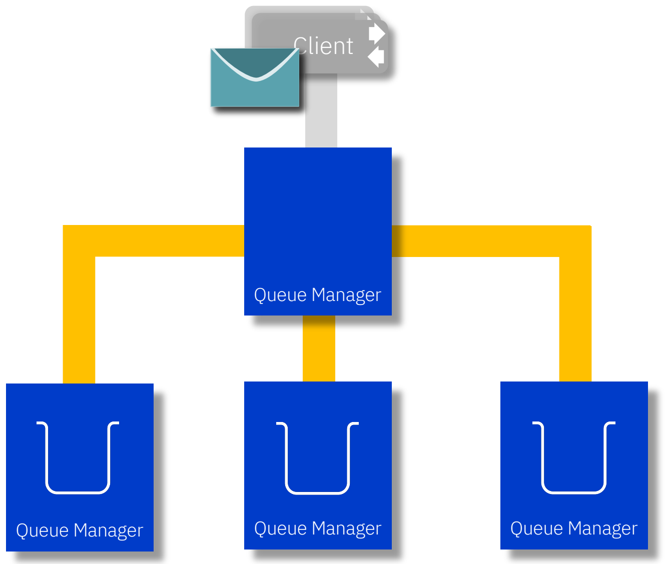
- Cluster workload balancing applies when there are **multiple cluster queues of the same name**

- Cluster workload balancing will be applied in **one of three ways**:
  - When the putting application opens the queue **- bind on open**
  - When a message group is started **- bind on group**
  - When a message is put to the queue **- bind not fixed**

- When workload balancing is applied:
  - The source queue manager builds a list of all potential targets based on the queue name
  - **Eliminates** the impossible options
  - **Prioritises** the remainder
  - If more than one come out equal, **workload balancing** ensues ...

- Balancing is based on:
  - The **channel** – not the target queue
  - **Channel traffic** to **all queues** is taken into account
  - **Weightings** can be applied to the channel

- ... this is used to send the messages to the chosen target



Channel workload balancing

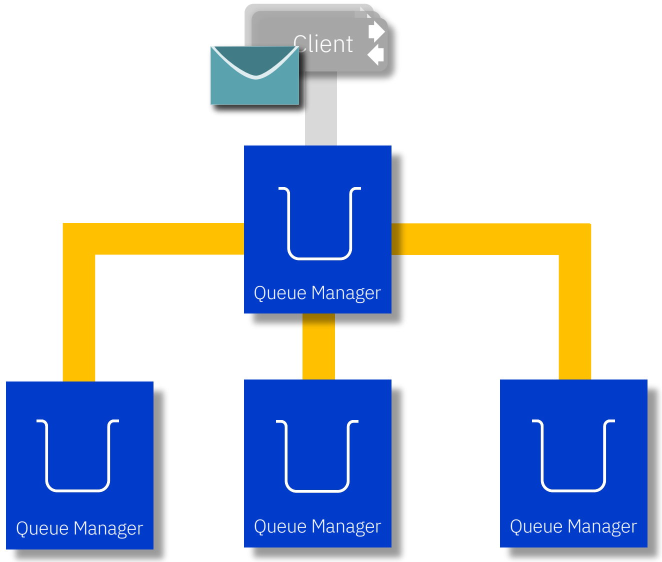
- Cluster workload balancing applies when there are **multiple cluster queues of the same name**

- Cluster workload balancing will be applied in **one of three ways**:
  - When the putting application opens the queue **- bind on open**
  - When a message group is started **- bind on group**
  - When a message is put to the queue **- bind not fixed**

- When workload balancing is applied:
  - The source queue manager builds a list of all potential targets based on the queue name
  - **Eliminates** the impossible options
  - **Prioritises** the remainder
  - If more than one come out equal, **workload balancing** ensues …

- Balancing is based on:
  - The **channel** – not the target queue
  - **Channel traffic** to **all queues** is taken into account
  - **Weightings** can be applied to the channel

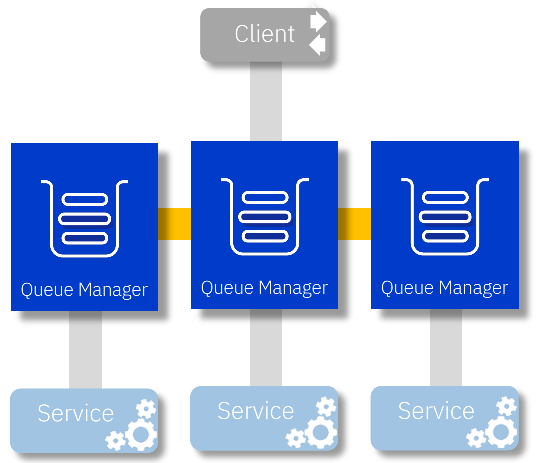
- … this is used to send the messages to the chosen target



Tip: By default, a matching queue on the same queue manager that the application is connected to will be prioritized over all others for speed. To overcome that, look at *CLWLUSEQ*

Horizontal scaling – *do I really need MQ Clustering?*

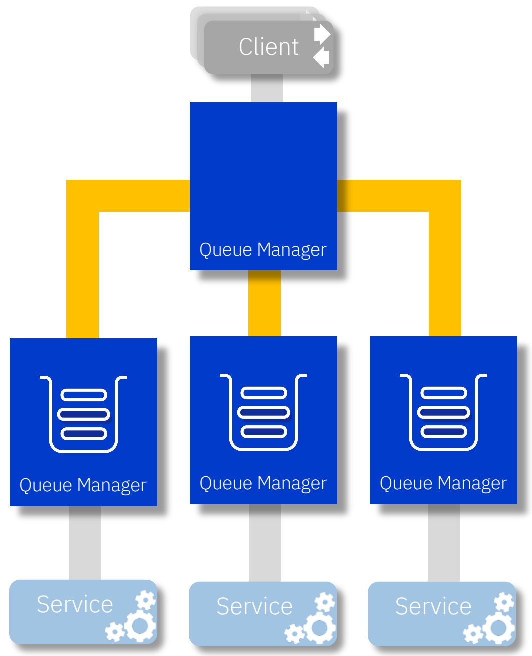
Single producing application



Q. Is clustering required?

A. Definitely

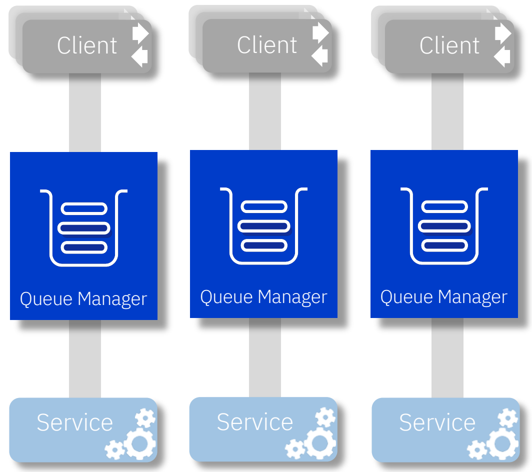
Gateway routing



Q. Is clustering required?

A. Definitely

Scaled out applications



Q. Is clustering required?

A. Maybe, maybe not …

Back to the *uniform cluster*

Building scalable, fault tolerant, solutions

Many of you have built your own continuously available and horizontally scalable solutions over the years

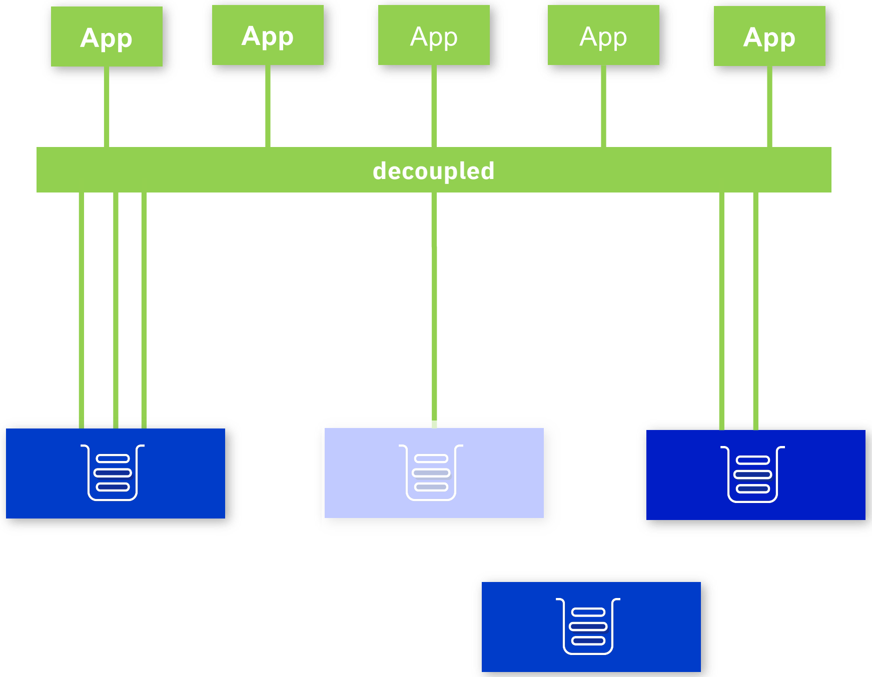
Let's call this the ***"uniform cluster"*** pattern

MQ has provided you many of the building blocks -

- Client auto-reconnect
- CCDT queue manager groups

But you're left to solve some of the problems, particularly with long running applications -

- Efficiently distributing your applications
- Ensuring all messages are processed
- Maintaining availability during maintenance
- Handling growth and contraction of scale



MQ 9.1.2 started to make that easier

IBM MQ 9.1.2 CD

For the distributed platforms, declare a set of matching queue managers to be following the uniform cluster pattern

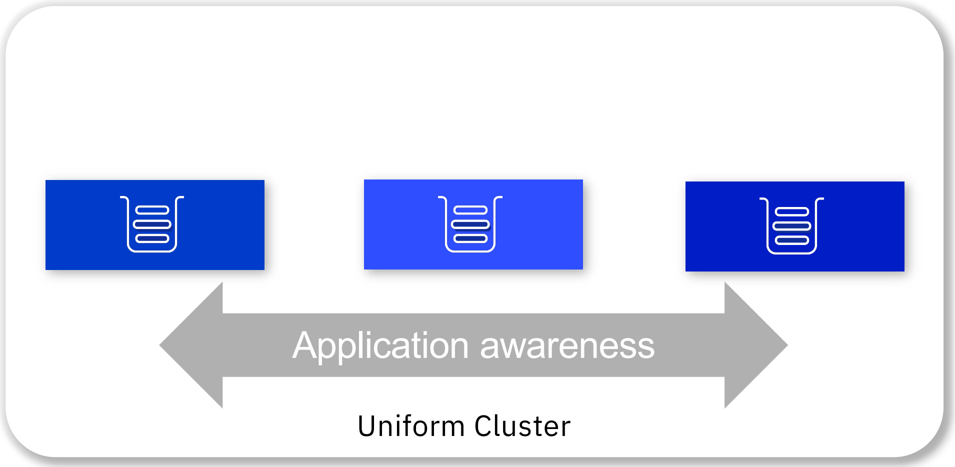
- All members of an MQ Cluster
- Matching queues are defined on every queue manager
- Applications can connect as clients to every queue manager

MQ will automatically share application connectivity knowledge between queue managers

The group will use this knowledge to automatically keep matching application instances balanced across the queue managers

- Matching applications are based on application name (new abilities to programmatically define this)

MQ 9.1.2 is started to roll out the client support for this



https://developer.ibm.com/messaging/2019/03/21/building-scalable-fault-tolerant-ibm-mq-systems/

Automatic Application balancing

IBM MQ 9.1.2 CD

Application instances can initially connect to any member of the group

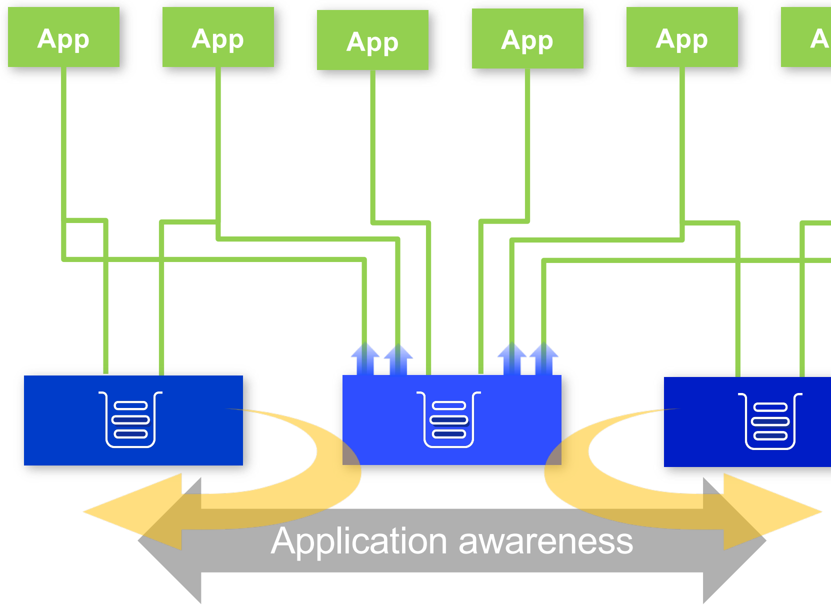
We recommend you use a queue manager group and CCDT to remove any SPoF

Every member of the uniform cluster will detect an imbalance and request other queue managers to donate their applications

Hosting queue managers will instigate a client auto-reconnect with instructions of where to reconnect to

Applications that have enabled auto-reconnect will automatically move their connection to the indicated queue manager

Client support has been increased over subsequent CD releases. 9.1.2 CD started with support for C-based applications, 9.1.3 CD added JMS …



https://developer.ibm.com/messaging/2019/03/21/building-scalable-fault-tolerant-ibm-mq-systems/

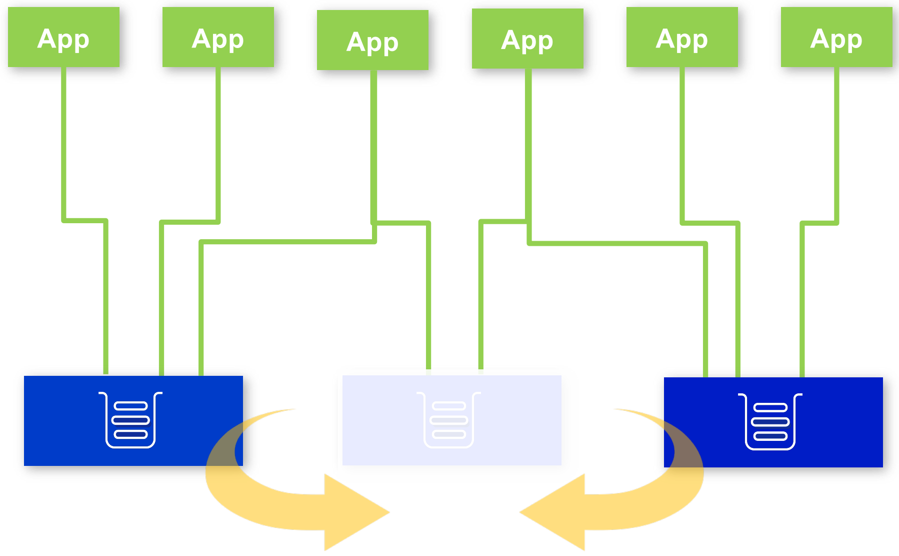
Automatic Application balancing

IBM MQ 9.1.2 CD

Automatically handle rebalancing following planned and unplanned queue manager outages

Existing client auto-reconnect and CCDT queue manager groups will enable initial re-connection on failure

Uniform Cluster rebalancing will enable automatic rebalancing on recovery



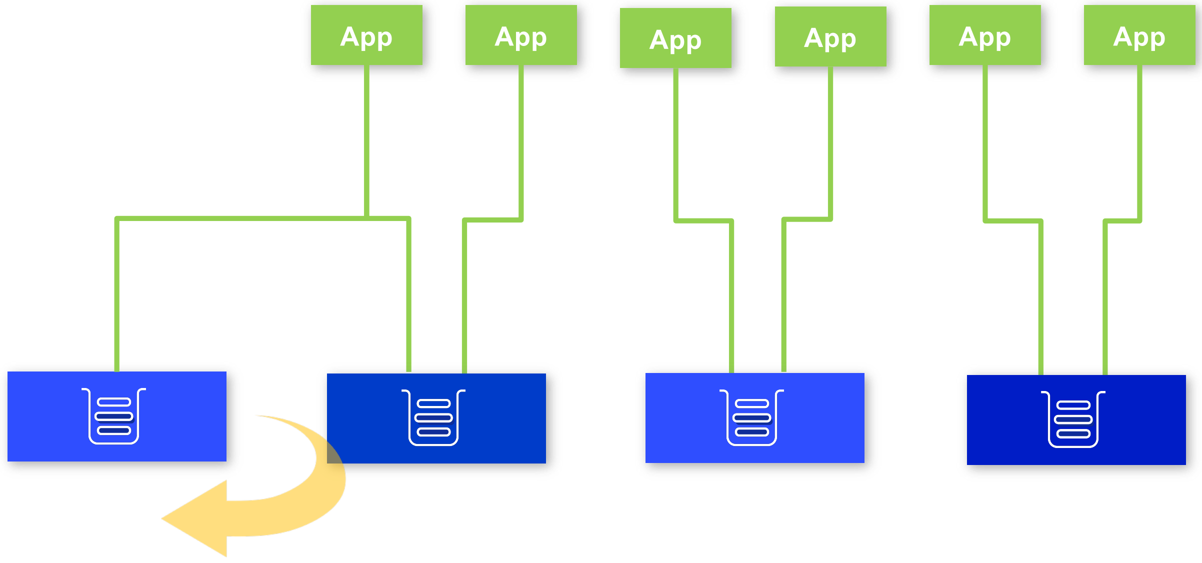
https://developer.ibm.com/messaging/2019/03/21/building-scalable-fault-tolerant-ibm-mq-systems/

Automatic Application balancing

IBM MQ 9.1.2 CD

Even to horizontally scale out a queue manager deployment

- Simply add a new queue manager to the uniform cluster
- The new queue manager will detect an imbalance of applications and request its fair share



https://developer.ibm.com/messaging/2019/03/21/building-scalable-fault-tolerant-ibm-mq-systems/

Uniform Cluster features

IBM MQ 9.1.2 CD

As well as the automatic rebalancing of the C library based clients, MQ 9.1.2 CD introduced a number of new or improved features for the distributed platforms that tie together to make all this possible

This means you need both the queue managers *and* the clients to be the latest MQ version

Watch this space…

Creation of a Uniform Cluster
- A simple qm.ini tuning parameter for now

```
TuningParameters:
   UniformClusterName=CLUSTER1
```

The ability to identify applications by name, to define grouping of related applications for balancing
- Extends the existing JMS capability to all languages

```
$ export MQAPPLNAME=MY.SAMPLE.APP
```

Auto reconnectable applications
- Only applications that connect with the auto-reconnect option are eligible for rebalancing

```
Channels:
   DefRecon=YES
```

Text based CCDTs to make it easier to configure this behaviour
- And to allow duplicate channel names

```
...
```

https://developer.ibm.com/messaging/2019/03/21/walkthrough-auto-application-rebalancing-using-the-uniform-cluster-pattern/

Balancing by application name

IBM MQ 9.1.2 CD

Automatic application balancing is based on the application name alone

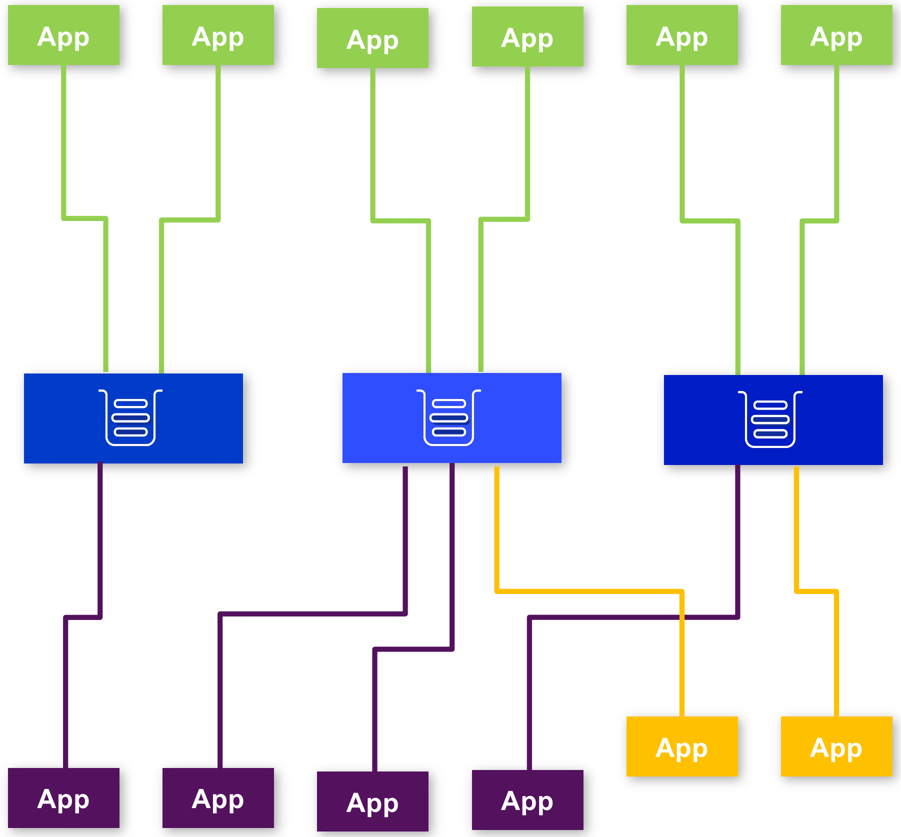
Different groups of application instances with different application names are balanced independently

By default the application name is the executable name

This has been customisable with Java and JMS applications for a while

MQ 9.1.2 CD clients have extended this to other programming languages
For example C, .NET, XMS, …

Application name can be set either programmatically or as an environment override



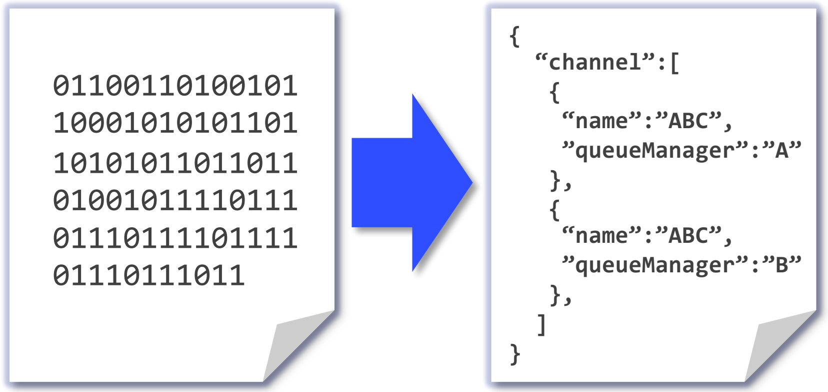
https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.dev.doc/q132920_.htm

Building scalable and available solutions

IBM MQ 9.1.2 CD

JSON CCDT

- Build your own JSON format CCDTs
- Supports multiple channels of the same name on different queue managers to simplify the building of uniform clusters
- Available with all 9.1.2 clients
  - C, JMS, .NET, Node.js, Golang clients



```
01100110100101
10001010101101
10101011011011
01001011110111
01110111101111
01110111011
```

```
{
  "channel":[
   {
    "name":"ABC",
    "queueManager":"A"
   },
   {
    "name":"ABC",
    "queueManager":"B"
   },
  ]
}
```

IBM MQ 9.1.2 CD

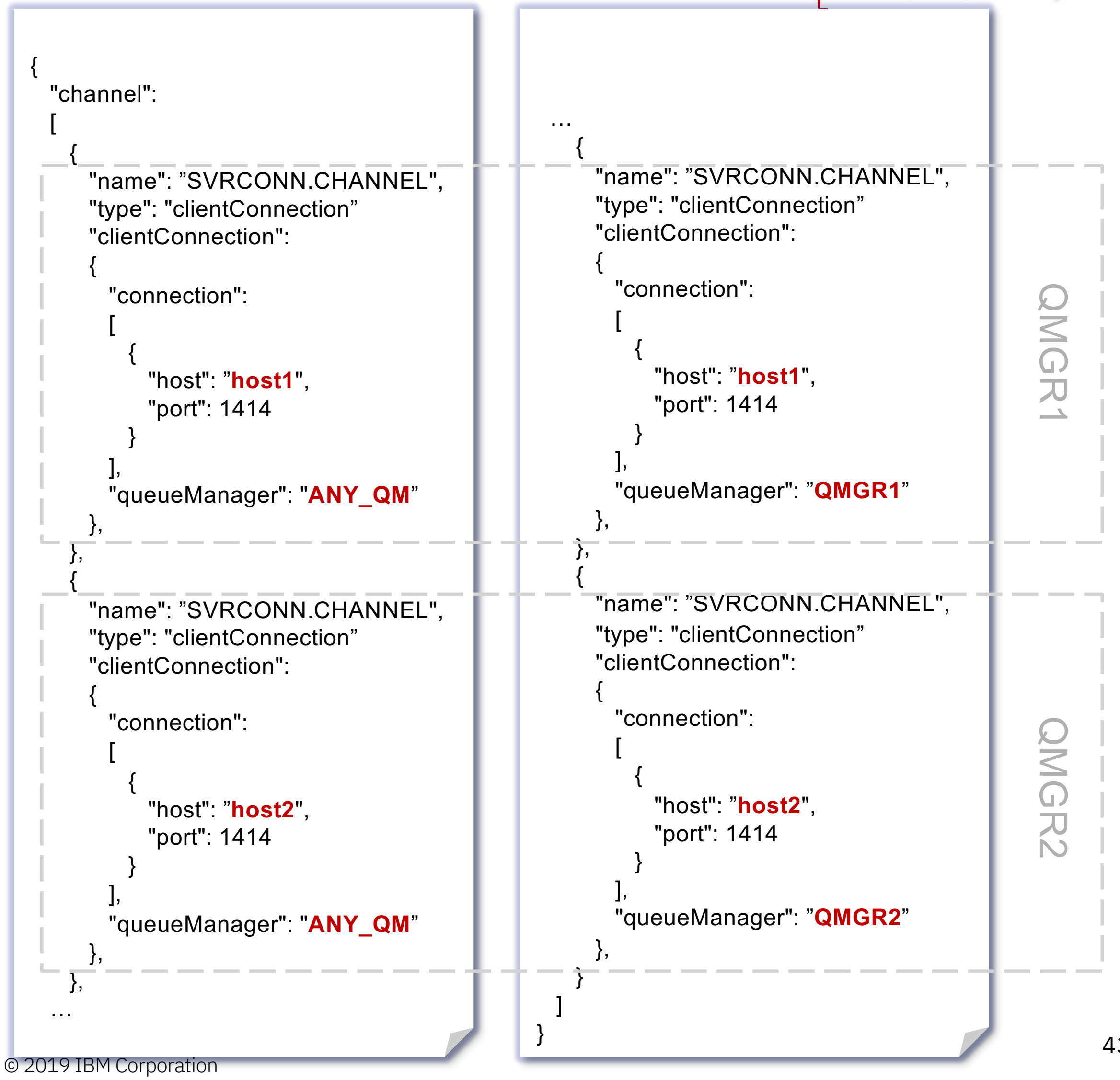
Configuring the CCDT for application balancing in a Uniform Cluster

To correctly setup a CCDT for application rebalancing it needs to contain **two** entries per queue manager:

- An entry under the name of a ***queue manager group***
- And entry under the queue *manager's **real name***

(These previously would need to be different channels, but with the JSON CCDT this is unnecessary)

The application connects using the queue manager group as the queue manager name (prefixed with an '*')


QMGR1
QMGR2

```
{
  "channel":
  [
    {
      "name": ”SVRCONN.CHANNEL",
      "type": "clientConnection”
      "clientConnection":
      {
        "connection":
        [
          {
            "host": ”host1",
            "port": 1414
          }
        ],
        "queueManager": "ANY_QM”
      },
    },
    {
      "name": ”SVRCONN.CHANNEL",
      "type": "clientConnection”
      "clientConnection":
      {
        "connection":
        [
          {
            "host": ”host2",
            "port": 1414
          }
        ],
        "queueManager": "ANY_QM”
      },
    },
  …
```

```
  …
    {
      "name": ”SVRCONN.CHANNEL",
      "type": "clientConnection”
      "clientConnection":
      {
        "connection":
        [
          {
            "host": ”host1",
            "port": 1414
          }
        ],
        "queueManager": ”QMGR1”
      },
    },
    {
      "name": ”SVRCONN.CHANNEL",
      "type": "clientConnection”
      "clientConnection":
      {
        "connection":
        [
          {
            "host": ”host2",
            "port": 1414
          }
        ],
        "queueManager": ”QMGR2”
      },
    }
  ]
}
```

IBM MQ 9.1.3 CD
Distributed

View application status

Now that MQ is taking a more application centric view, a new command has been added to Distributed runmqsc to aid the understanding of how applications are balanced across a Uniform Cluster

From any member of the Uniform Cluster, displays applications by name and highlights **Application Instances** that are not evenly balanced

```
DISPLAY APSTATUS(*) TYPE(APPL)

AMQ8932I: Display application status details.
   APPLNAME(AMQSPHAC)                       CLUSTER(UNIDEMO)
   COUNT(8)                                 MOVCOUNT(8)
   BALANCED(YES)
AMQ8932I: Display application status details.
   APPLNAME(AMQSPUTC)                       CLUSTER( )
   COUNT(2)                                 MOVCOUNT(0)
   BALANCED(NOTAPPLIC)


DISPLAY APSTATUS(*) TYPE(QMGR)

AMQ8932I: Display application status details.
   APPLNAME(AMQSPHAC)                       ACTIVE(YES)
   COUNT(3)                                 MOVCOUNT(3)
   BALSTATE(OK)                             LMSGDATE(2019-05-08)
   LMSGTIME(14:05:36)                       QMNAME(UNID001)
   QMID(UNID001_2019-05-08_13.59.31)
AMQ8932I: Display application status details.
   APPLNAME(AMQSPHAC)                       ACTIVE(YES)
   COUNT(3)                                 MOVCOUNT(3)
   BALSTATE(OK)                             LMSGDATE(2019-05-08)
   LMSGTIME(14:04:50)                       QMNAME(UNID002)
   QMID(UNID002_2019-05-08_13.59.35)
AMQ8932I: Display application status details.
   APPLNAME(AMQSPHAC)                       ACTIVE(YES)
   COUNT(2)                                 MOVCOUNT(2)
   BALSTATE(OK)                             LMSGDATE(2019-05-08)
   LMSGTIME(14:04:44)                       QMNAME(UNID003)
   QMID(UNID003_2019-05-08_13.59.40)
AMQ8932I: Display application status details.
   APPLNAME(AMQSPUTC)                       ACTIVE(YES)
   COUNT(2)                                 MOVCOUNT(0)
   BALSTATE(NOTAPPLIC)                      LMSGDATE(2019-05-08)
   LMSGTIME(14:05:36)                       QMNAME(UNID001)
   QMID(UNID001_2019-05-08_13.59.31)
```

Can I decouple any application?

Does this work for all applications? – *no*

This pattern of loosely coupled applications only works for certain applications styles.

Good

Applications that can tolerate being moved from one queue manager to another without realising and can run with multiple instances

- Datagram producers and consumers
- Responders to requests, e.g. MDBs
- No message ordering

Bad

Applications that create persistent state across multiple messaging operations, or require a single instance to be running

- Requestors waiting for specific replies
- Dependant on message ordering
- Global transactions …

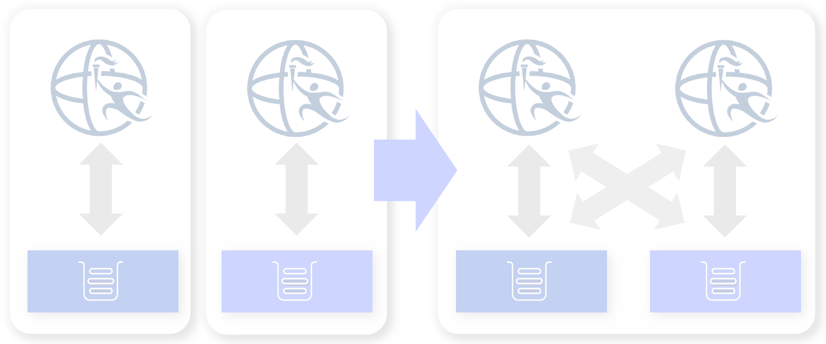
A new hope for transactions

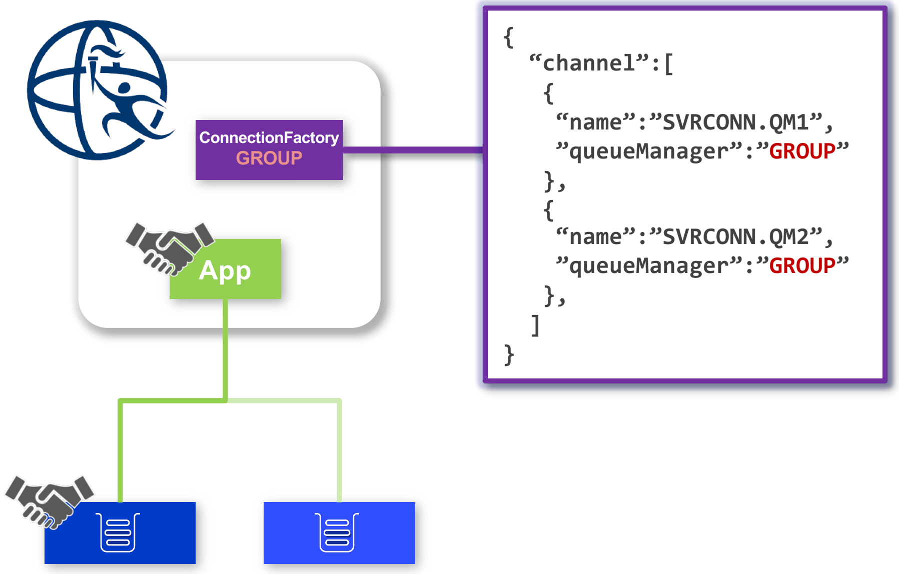
IBM MQ 9.1.2 CD

Global transactions require a single resource manager to be named when connecting. For MQ a resource manager is a queue manager.

This prevents the use of queue manager groups in CCDTs

However, **WebSphere Liberty 18.0.0.2** and **MQ 9.1.2 CD** support the use of CCDT queue manager groups when connecting





```
{
  "channel":[
   {
    "name":"SVRCONN.QM1",
    "queueManager":"GROUP"
   },
   {
    "name":"SVRCONN.QM2",
    "queueManager":"GROUP"
   },
  ]
}
```

Availability routing in an MQ Cluster

Clustering for availability

Is MQ Clustering a high availability solution?

Not for the message data. Each message is only available from a single queue manager

✗ **NO**

✓ **YES**

Clustering can form a part of the overall high availability of the messaging system

- Having multiple potential targets for any message can improve the availability of the solution, always providing an option to process new messages.
- A queue manager in a cluster has the ability to route new and old messages based on the **availability of the channels**, routing messages to running queue managers.
- Clustering can be used to route messages to active consuming applications.

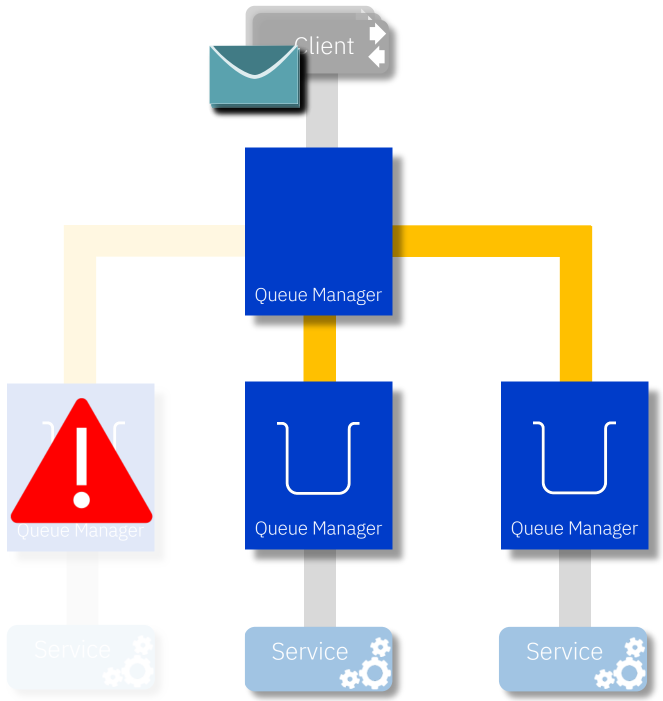
Channel availability routing

- When performing workload balancing, the availability of the channel to reach the target is a factor
- All things being equal, messages will be routed to those targets with a working channel

Routing of messages based on availability doesn't just happen when they're first put, it also occurs for **queued transmission messages** every time the channel is **retried**
So blocked messages can be re-routed, if they're not prevented…

Things that can prevent routing

- Applications targeting messages at a specific queue manage (e.g. reply message)
- Using "cluster workload rank"
- Binding messages to a target



Pros and cons of binding

Bind on open | **Bind on group**

Bind context:
Duration of an *open*
Duration of *logical group*

- All messages put within the bind context will go to same target*
- Message order can be preserved**
- Workload balancing logic is only driven at the start of the context

- Once a target has been chosen it **cannot change**
    - Whether it's available or not
    - Even if all the messages could be redirected

Bind not fixed

Bind context:
None

- Greater *availability,* a message will be redirected to an available target***

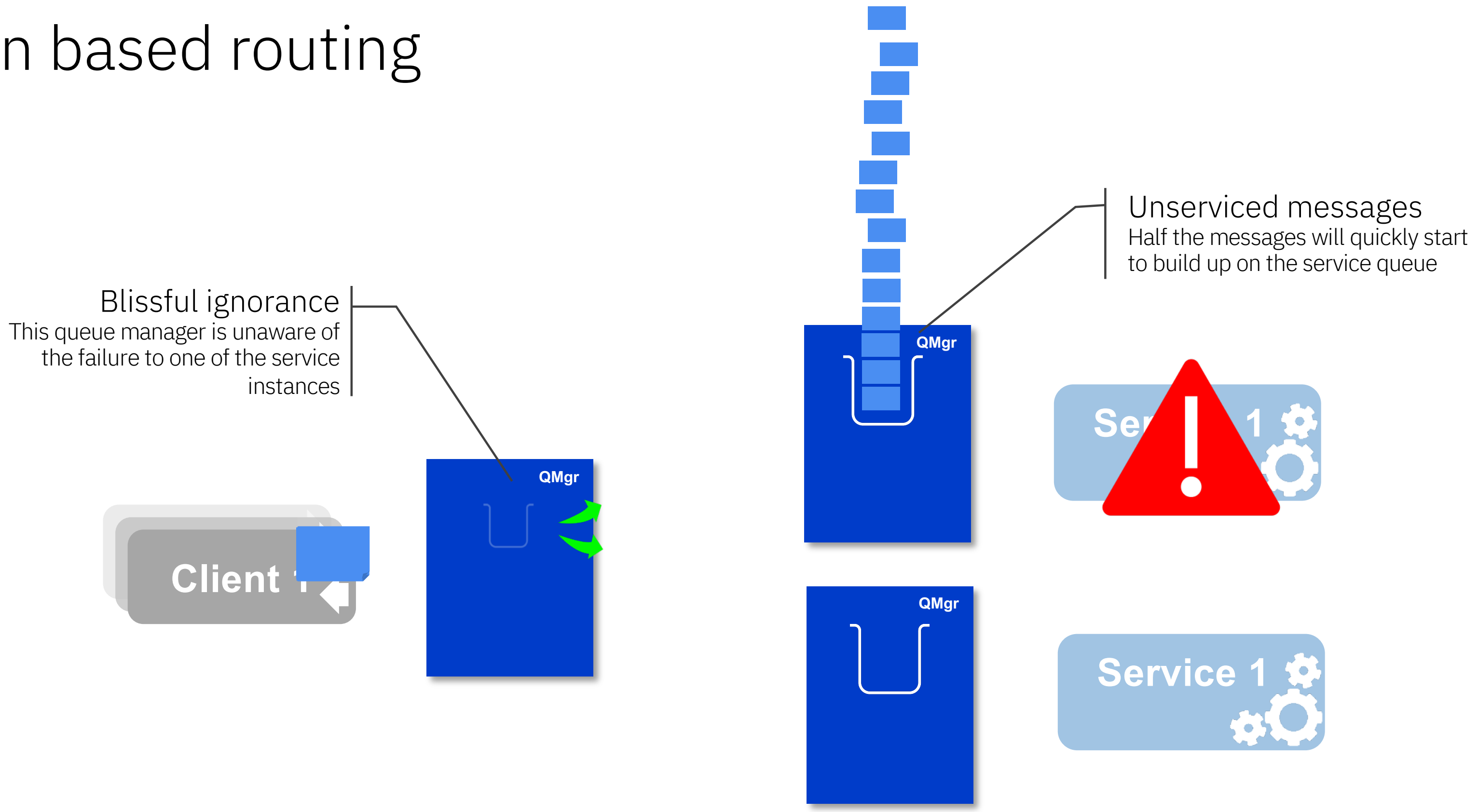
- Overhead of workload balancing logic for every message
- Message order may be affected

Bind on open is the default
It could be set on the cluster queue (don't forget aliases) or in the app

* While a route is known by the source queue manager, it won't be rebalanced, but it could be DLQd
** Other aspects may affect ordering (e.g. deadletter queueing)
*** Unless it's fixed for another reason (e.g. specifying a target queue manager)

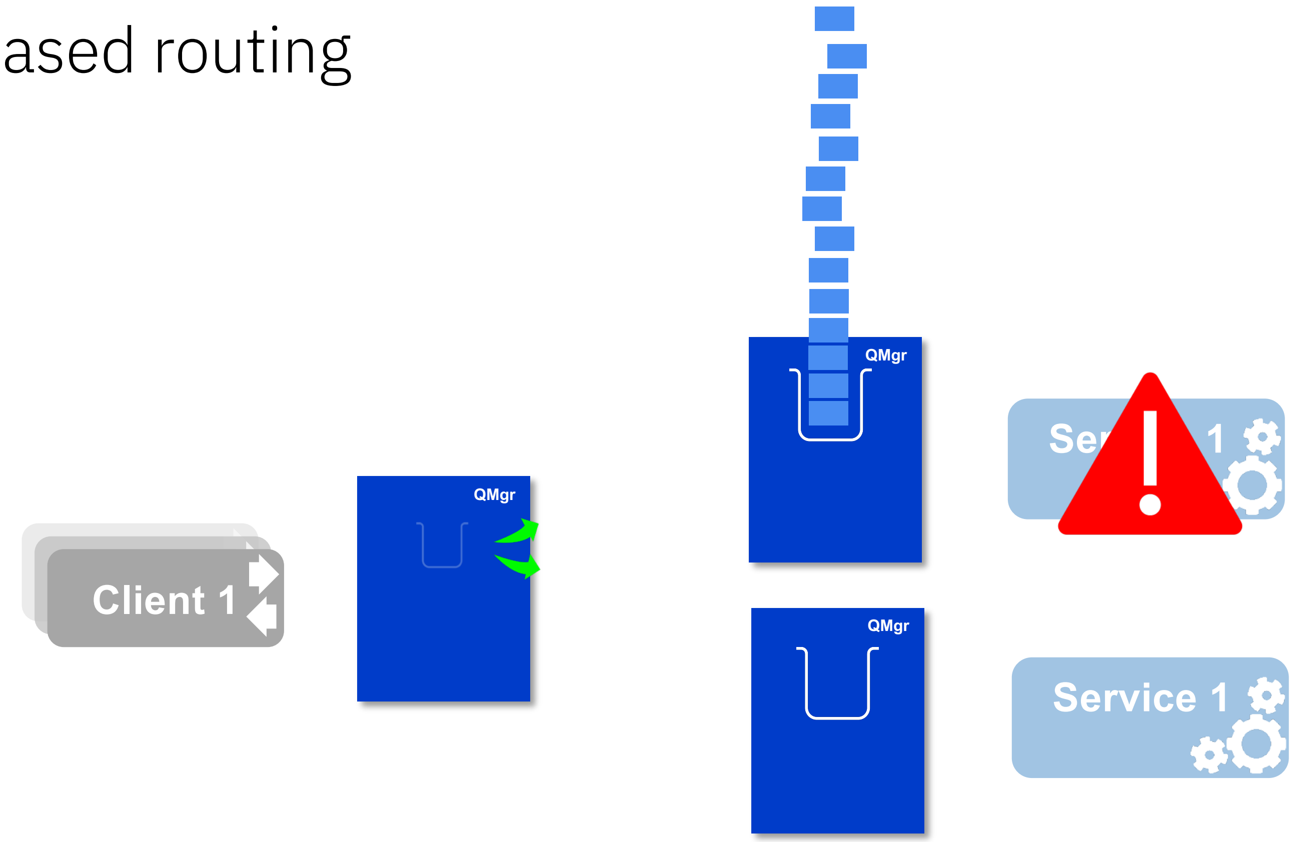
Application availability routing

Application based routing

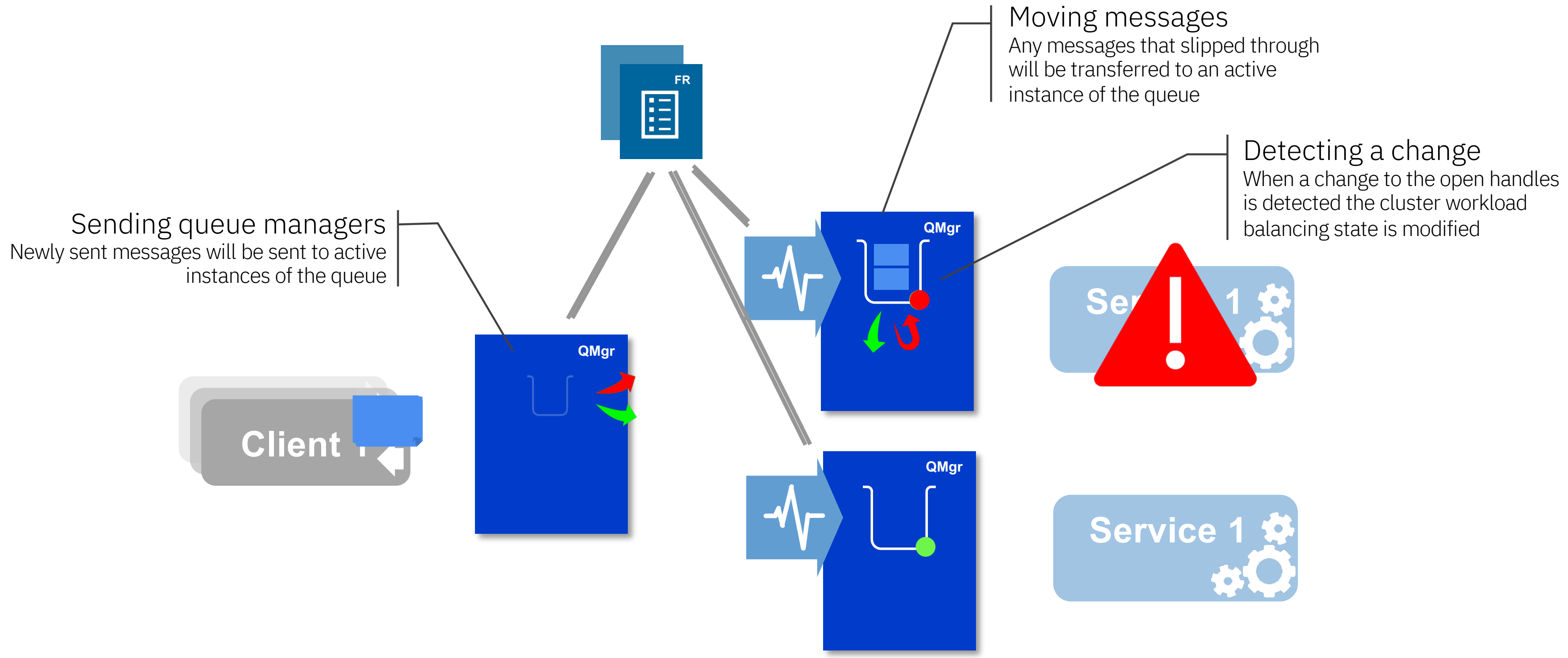

Blissful ignorance
This queue manager is unaware of the failure to one of the service instances
Unserviced messages
Half the messages will quickly start to build up on the service queue
QMgr
QMgr
QMgr
Client 1
Service 1
Service 1

- Cluster workload balancing does not take into account the availability of receiving applications
- Or a build up of messages on a queue

Application based routing



Application based routing



Moving messages
Any messages that slipped through will be transferred to an active instance of the queue
Detecting a change
When a change to the open handles is detected the cluster workload balancing state is modified
Sending queue managers
Newly sent messages will be sent to active instances of the queue
FR
QMgr
QMgr
QMgr
Client 1
Service 1
Service 1

- MQ provides a sample monitoring service tool, **amqsclm**
- It regularly checks for attached consuming applications (IPPROCS)
- And automatically adjusts the cluster queue definitions to route messages intelligently (CLWLPRTY)
- That information is automatically distributed around the cluster

Cluster Queue Monitoring Sample

- amqsclm, is provided with MQ to ensure messages are directed towards the instances of clustered queues that have consuming applications currently attached. This allows all messages to be processed effectively even when a system is asymmetrical (i.e. consumers not attached everywhere).
  - In addition it will move already queued messages from instances of the queue where no consumers are attached to instances of the queue with consumers. This removes the chance of long term marooned messages when consuming applications disconnect.
- The above allows for more versatility in the use of clustered queue topologies where applications are not under the direct control of the queue managers. It also gives a greater degree of high availability in the processing of messages.
- The tool provides a monitoring executable to run against each queue manager in the cluster hosting queues, monitoring the queues and reacting accordingly.
  - The tool is provided as source (amqsclm.c sample) to allow the user to understand the mechanics of the tool and customise where needed.

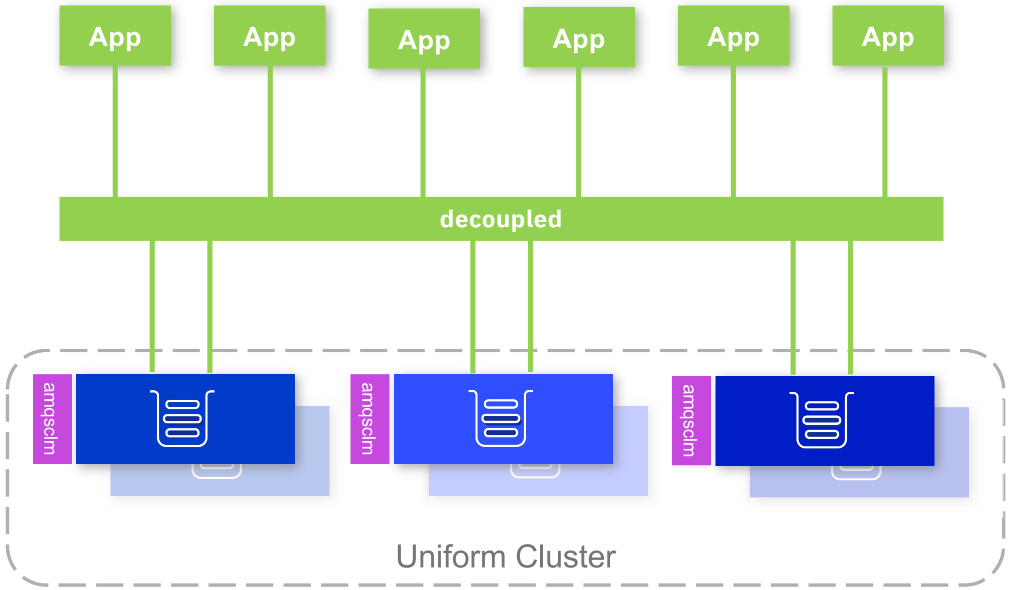
AMQSCLM Logic

- Based on the existing MQ cluster workload balancing mechanics:
  - Uses cluster priority of individual queues – all else being equal, preferring to send messages to instances of queues with the highest cluster priority (CLWLPRTY).
  - Using CLWLPRTY always allows messages to be put to a queue instance, even when no consumers are attached to any instance.
  - Changes to a queue's cluster configuration are automatically propagated to all queue managers in the cluster that are workload balancing messages to that queue.
- Single executable, set to run against each queue manager with one or more cluster queues to be monitored.
- The monitoring process polls the state of the queues on a defined interval:
  - If **no** consumers are attached:
    - CLWLPRTY of the queue is set to zero (if not already set).
    - The cluster is queried for any active (positive cluster priority) queues.
    - If they exist, any queued messages on this queue are got/put to the same queue. Cluster workload balancing will re-route the messages to the active instance(s) of the queue in the cluster.
  - If consumers **are** attached:
    - CLWLPRTY of the queue is set to one (if not already set).
- Defining the tool as a queue manager service will ensure it is started with each queue manager

Putting it all together

Bringing it all together

- Build a matching set of queue managers, in the *style* of a uniform cluster
- Make them highly available to prevent stuck messages
- Consider adding amqsclm to handle a lack of consumers
- Setup your CCDTs for decoupling applications from individual queue managers
- Look at the 9.1.2+ application rebalancing capability and see if it matches your needs
- Connect your applications

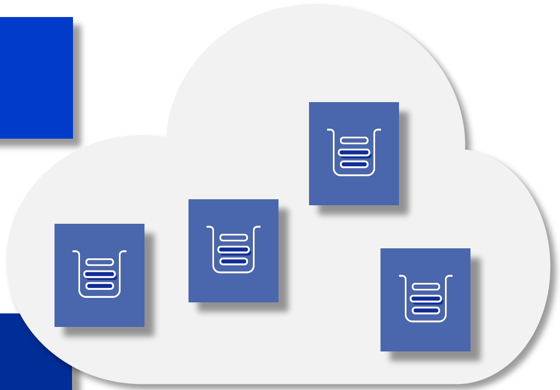


What about MQ Clusters in *the cloud?*

MQ Clusters and Clouds

“Cloud platforms provide all an MQ cluster can do”

Not quite…



Clouds often provide cluster-like capability:
Directory services and routing
Network workload balancing

Great for stateless workload balancing
Can be good for balancing unrestricted clients across multiple

But where state is involved, such as reliably sending messages from one queue manager to another without risking message loss or duplication, such routing isn’t enough

That’s still the job of an MQ cluster…

Adding and removing queue managers

Adding queue managers

+

- Have a simple clustering topology
- Separate out your full repositories and manage those separately
- Automate the joining of a new queue manager

Removing queue managers - *this is harder!*

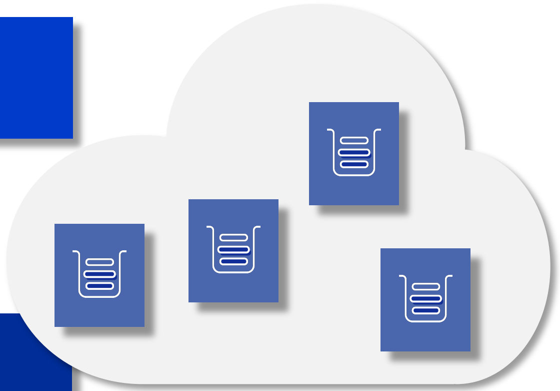
—

- You might have messages on it that you need, how are you going to remove those first?
- If you just switch off the queue manager, it'll still be known by the cluster for months!
- Is that a problem?
  - If routing is based on availability, messages will be routed to alternative queue managers
  - Messages will sit on the cluster transmission queues while the queue manager is still known
  - It makes your view of the cluster messy
- Automate the cluster removal
  - From the deleting queue manager
    - Stop and delete the cluster channels – give it a little time to work through to the FRs
  - Then from a full repository – as it'll never be coming back
    - RESET the queue manager out of the cluster

MQ Clusters and Clouds

“So will MQ clusters work in a cloud?”

Yes, but think it through first…



Some things may be different in your cloud:

- A new expectation that queue managers will be created and deleted more dynamically than before
- Queue managers will “move around” with the cloud
- Your level of control may be relaxed

Thank You

Anthony Beardsmore
abeards@uk.ibm.com