MQ Admin

Moving your IBM MQ estate to containers

Callum Jackson
Published on 08/06/2018 / *Updated on 01/11/2018*

Many MQ users are trying to understand how they can adopt containers and what benefits they can deliver in an organization. This guide takes you through this journey, and explains what other clients are exploring and their challenges.

Introducing container technology, benefits and the key aspects for adoption

Software Containers

Prior to jumping into the benefits of a containerization strategy, it is important to level set on what software containers are. The concept of a software container, is similar to a cargo container. The container can include any type of cargo, which can fit on any transportation ship/lorry that supports containers. Software containers are the same, within the container any software can be installed, and run on a container supported platform. This reduces the effort associated with moving from one platform to another. For instance if you decide to initially run a container within your own data center, and then want to move into the cloud, the barriers of getting the software installed and configured is reduced.

Benefits of a container strategy

Regardless of the software technology installed and configured within a container, there are a number of common benefits:

1. **Infrastructure Optimization** – Isolation between running artefacts is often desirable, however previous virtualization technology has often imposed a large system resource cost. Containers provide better **density** within the infrastructure, when compared to virtual machines. This is due to containers having less memory overhead per instance, as each container reuses the existing host operating system. This allows us to tackle resource management issues using a new approach. For instance an existing multiple component solution, which was hosted on a single machine can be separated into multiple containers and each container can have its resources capped.
2. **Scalability & Availability** – Container orchestration technology provides the capability to scale containers horizontally, and balance the inbound traffic across the available instances.
3. **Operational Consistency** – All parts of the solution are deployed and managed in the same way, as containers, regardless of the underlying technology. This provides a

common deployment, management, operational and monitoring approach across the deployed software.

4. **Team Agility** – Often a container will be scoped to an individual team, allowing them to build their own solutions independently, react to change more rapidly, and reaching a greater team velocity. As containers are specific to a team and a particular project, they are smaller in nature, which allows for fast deployment.
5. **Fine Grained Resilience** – Embrace the disposable approach of containers, if a container stops working, kill it off, and start a new one. This reduces the cost, effort of maintenance, and potentially the overall resilience of the solution. This simplifies the process of upgrading the solution, as the container will not hold any required state, the old versions can be disposed of, and the new versioned containers started.
6. **Component Portability** – Any platform with a container engine can run containers. This lowers the barrier to moving cloud providers.

Container technology

There are many software container technologies within the market. One of the most common is Docker, although alternatives such as CRI-O do exist. Docker is currently the most established, launched in March 2013, and has a large community of supporters. These include contributors to Docker itself, support for running Docker containers, and finally, support/images for running software within Docker containers. Most major cloud providers support running Docker containers, such as IBM Cloud, Amazon Web Services and Microsoft Azure.

One common question is the difference between containers and virtual machines. Virtual Machines have provided a degree of portability for a number of years, so how are containers different? The answer is that containers reuse the existing host operating system, while a Virtual Machine includes a complete guest operating system. This allows containers to provide a lighter weight footprint, while still providing isolation between containers.

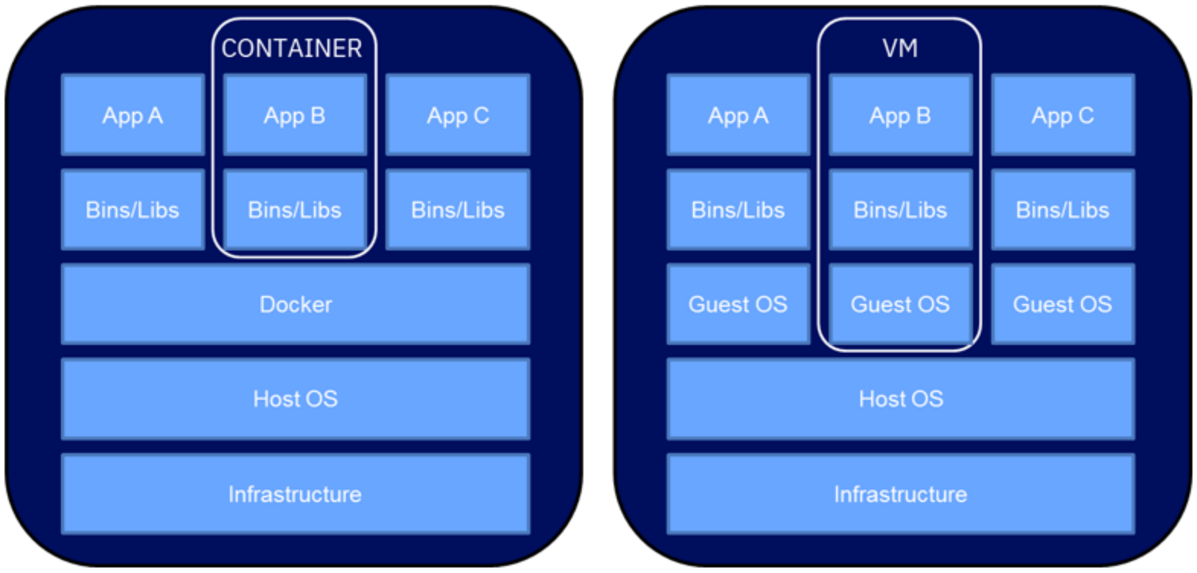

Figure 1: Comparison of Containers vs Virtual Machines

Container Orchestrator technology can be used to provide management, scalability and availability of containers across clusters of servers. For instance, a container may be duplicated for scalability and availability, and the number potentially changing based on the demand. As we are dynamically changing the number and location of containers, either due to failures or load requirements, we need load balancing, so that the service can continue to be accessed. This is provided by Container Orchestrator technology.

Kubernetes is one of the leading container orchestrators on the market, originally created by Google, and now maintained by the Cloud Native Computing Foundation. Other orchestrators do exist such as Docker Swarm. In addition to assuring that a certain number of containers are running and available across multiple nodes at all times, Kubernetes provides management APIs to control its behaviour. When updates to a deployment are required, built-in capabilities allow you to control how these updates will be applied, for instance using rolling updates. Kubernetes also provides an extensive framework for plugins, allowing the community to extend its capabilities.

Stateful Containers

The use of containers appears to be more challenging when we consider stateful applications such as messaging providers or databases, where the data needs to be persisted. One of the benefits of containers is the **fine grained resilience**, where containers are killed off when they are no longer required, but how does this work with stateful applications? The container platform provides the ability to attach shared storage. In the case of a stateful application, any data that needs to be retained is stored in the shared storage so it is available to the new container. This is illustrated in the figure below:

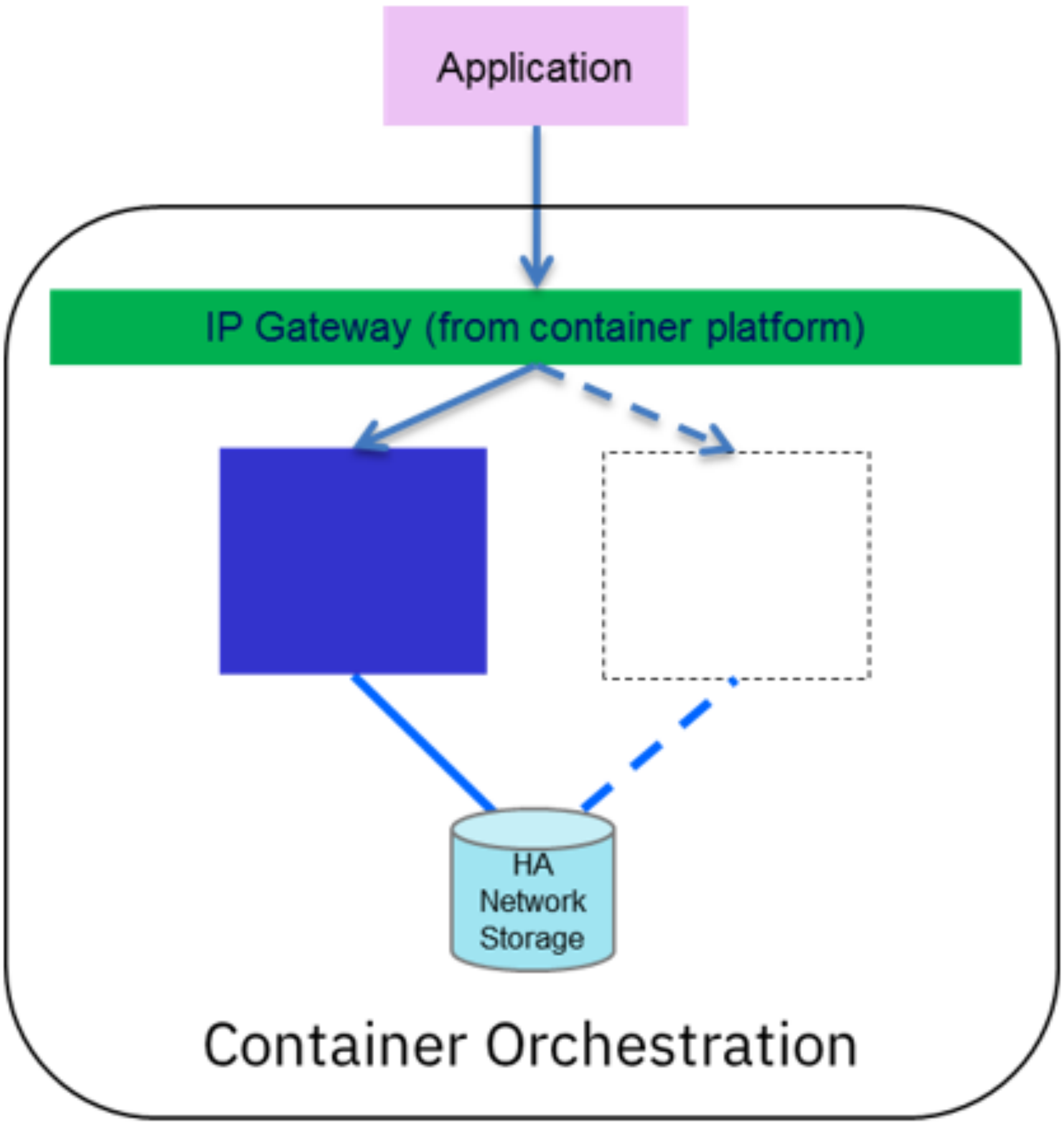

Figure 2: Stateful Container

Overview of adopting containers within a MQ estate

A key benefit of using containers is that you get a more common operational model across multiple technologies. Therefore the adoption of containers should be part of an organization wide strategy instead of focusing on an individual technology, to truly realize the benefits.

The adoption of containers in the context of MQ can be separated into three aspects:

1. **Foundational**: Many organizations have a traditional software on-premise deployment, and are keen to cloud enable. For some organizations this may mean moving to a public cloud provider, while others will take a more gradual approach containerizing their on-premise deployment. Once running within containers, this will facilitate **component portability**, as containers can be copied to any platform that supports containers. The infrastructure team will streamline their operations

across software products, as all software can be handled in a standard approach, providing **operational consistency**. As the footprint of new containers is low, this increases the number of running containers a machine can handle, compared to virtual machines, providing the **infrastructure optimization**. Finally the container orchestration technology will assure the containers are available, which provides a level of high availability and **fine grained resilience**.

2. **Decentralized**: New development architectures are being embraced within organizations, which facilitate greater **team agility**, by decoupling dependencies between teams. To succeed, this needs to include resources such as IBM MQ. Within the organization there is normally an IBM MQ Administration team that owns and maintains the messaging platform, however individual development teams are eager for more control and access. To provide the additional flexibility, individual IBM MQ instances can be provided to the development teams, so they can be modified independently of the wider IBM MQ estate. Although the development teams will have additional access, the IBM MQ instance will normally be integrated into the wider messaging backbone, and may be administered by the central MQ team.
3. **Optimized**: Container orchestration technology provides the ability to provide **scalability and continuous availability**. Although IBM MQ is infrequently the bottleneck, it is reassuring to know that IBM MQ can utilize the same container scaling principles.

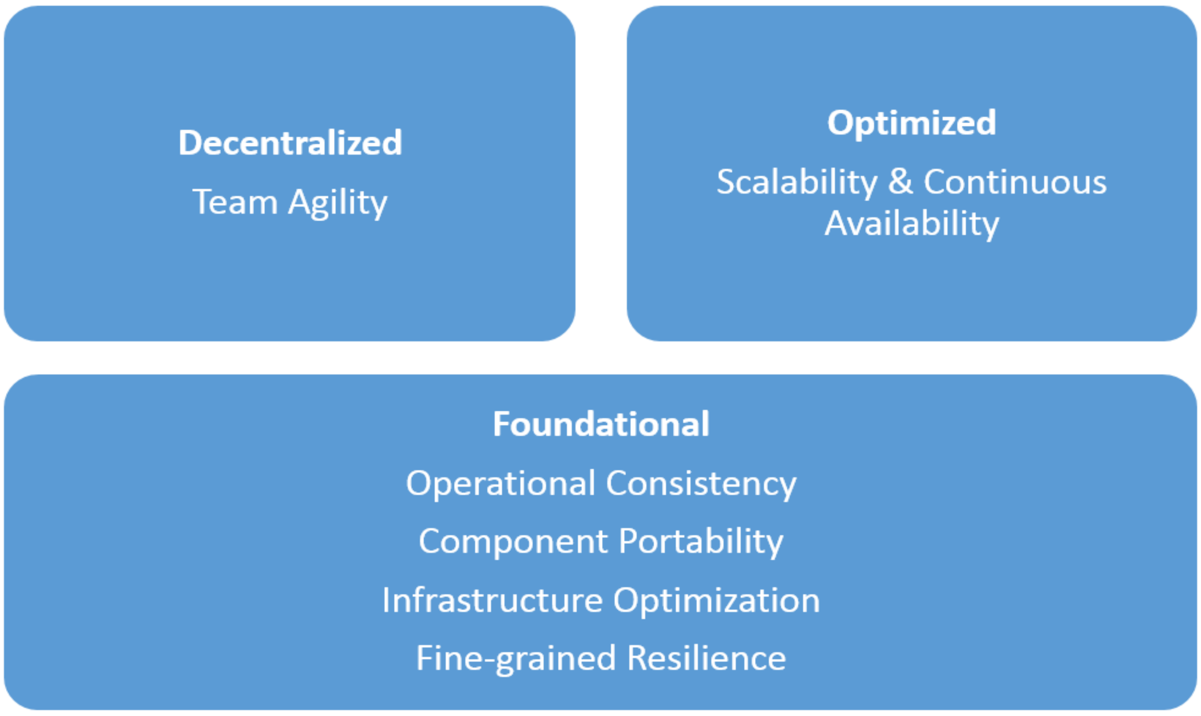

Figure 3: Aspects of a MQ Container Journey

When an organization is planning to adopt containers it is critical to understand what benefits they want to realise. Some organizations may pause after the **Foundational** aspects, while others will want to continue immediately with the **Decentralized** and **Optimized** aspects, to realise these benefits. The additional aspects can be completed together or separately, depending on the objectives of the organization. Each of these aspects will be considered further to understand how and what changes will occur within an IBM MQ estate.

Aspects: Foundational

Let’s consider the Foundational aspect. To discuss the changes we will start with a high level representation of a typical client’s IBM MQ estate as shown below:

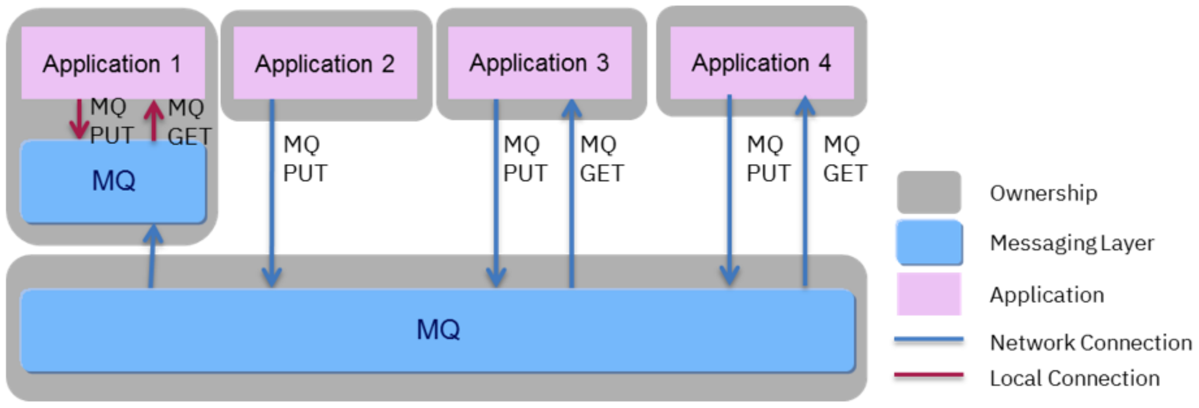

Figure 4: Existing IBM MQ estate

An existing IBM MQ estate may include MQ instances (one or more Queue Managers), logically associated with application teams (as shown with application 1), and also central MQ instances controlled by a dedicated central MQ team. Interestingly, although the application teams (application 1) have a MQ instances, the administration of these instances may be completed by the same administrators who control the central MQ (due to skills). This could be due to the particular requirements that the application team had for MQ, requirement for additional isolation, performance, security, need for global transactions or a drive to decouple for agility. Regardless many organizations will have a starting position where they have multiple instances of MQ. In addition all or some of these instances may or may not, be included in a MQ Cluster to assist with scalability, availability and routing.

The existing IBM MQ estate may also include different types of connectivity between the applications and IBM MQ. IBM MQ supports two options for connecting to a queue manager:

- **Local Connections**: applications running on the same system can connect using inter-process communication. This is called a server binding.
- **Network Connections**: applications running on remote or the same system can communicate using a network connection, using MQ channels support. This is called a client binding.

As shown in the diagram it is common for a MQ estate to have different mechanisms being used. Often for simplicity (and sometime for transactional and performance reasons) the server binding option was used.

New Containerized Architecture

The high level target architecture is shown below:

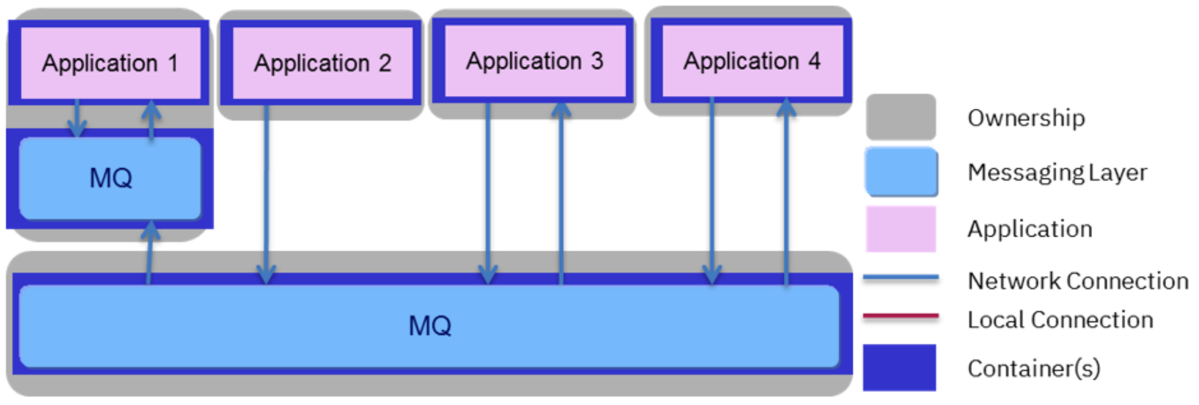

Figure 5: Containerized IBM MQ estate

There are three changes that are important to highlight:

- **MQ containerization**: the IBM MQ installations have been placed into containers, but the original high level topology remains the same. The diagram deliberately does not show the detailed configuration of IBM MQ from an availability and scalability view point, as we will discuss the considerations here in more detail later.
- **Application containerization**: the application logic has also been containerized. This is deliberate and an important factor to be considered. If you are wanting to realise operational consistency across software, then the scope of your containerization project will need to be wider than simply MQ. If it is not then you will lose the benefit of operational consistency.
- **Separation of MQ and Application logic**: as illustrated above, Application 1 and the associated MQ are deployed into two separate containers. These two containers can be deployed separately, or logically linked together at the time of deployment into a Kubernetes Pod. A pod provides the ability to group together containers that were previously relatively tightly coupled and run on the same machine. Within a pod you share several resources, such as the IP address and port space, which can simplify the move towards containers. Regardless of the deployment mechanism a network connection between the application and MQ container is encourage, to allow flexibility. This removes the ability to use local connections (Server Bindings) when communicating with IBM MQ.

As the new architecture is adopted, and the applications and MQ are moved into separate containers, there are several aspects to consider:

- **Verify the performance characteristics** – As you are moving to a containization strategy you will need to re-validate the performance characteristics of the entire solution, and the MQ part is no different.
- **Evaluate the security requirements for network connectivity** – As a network connection is being established, this may change the network security requirements

for the communication, and also the security checks the MQ completes on accepting a connection.

- **Verify MQ is not acting as a global transactional coordinator** – If MQ is acting as the transactional coordinator, then server bindings would either need to be maintained, or a new approach taken. It is important to highlight that you can still use MQ in a transactional manner, and as part of a global transaction, but MQ would not be the transactional coordinator.
- **Separate lifecycles between the application and MQ** – Previously people may have assumed MQ and the Application were available if one of the two were available (for instance if they were residing on the same machine), moving forward this already risky assumption becomes less appropriate.

Deep dive into the IBM MQ design

Within a container environment clients often start with a ***single resilient queue manager*** as the foundational building block for deployments. The ***single resilient queue manager*** topology is built on the Stateful Container pattern mentioned previously and illustrated below:

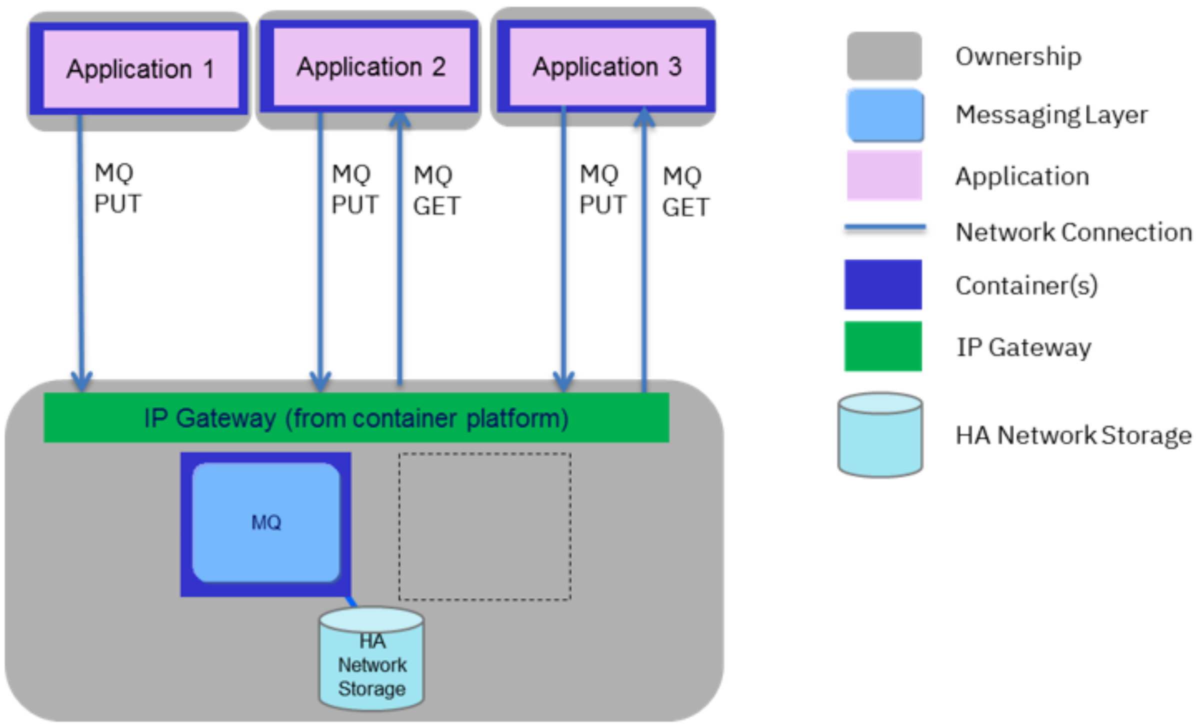

Figure 6: Single resilient queue manager

This pattern has the advantage that a degree of high availability is provided automatically by the container orchestration platform. The MQ container will be automatically restarted in the case of a failure, or if the container is detected to be unhealthy. The container orchestration platform will provide an IP Gateway to allow the routing of requests to the location of the active container. The Queue Manager data and logs will be stored on shared storage, so any running container can attach. Later we will discuss how this pattern can be used for scalability and additional availability scenarios. Depending on your requirements for high availability, your choice of container orchestration platform and its corresponding configuration, the above approach may or may not be adequate for your needs.

Alternative approaches are available and a brief summary are discussed below:

- **Multi-instance queue manager**: this has an active and standby MQ Queue Manager, where the standby instance is ready to take over in the case of a failure. The failover logic is provided by the product, but the routing to the active instance is the responsibility of the client or an external IP Gateway. In addition you need to manage and appropriately license the active and standby instances. The failover time of the multi-instance queue manager can be lower than the ***single resilient queue manager,*** especially when we consider an entire node failure.
- **Replicated data queue manager**: unfortunately it is generally not suitable for use with containers, due to the use of Linux kernel modules. In most cases, users running containers will not have sufficient access to the host server to manage kernel modules.

Aspect: Decentralized

One driver for a containerization strategy, is to move towards a decentralized and modern architecture, where application logic moves away from a monolithic application, and is separated into application components (perhaps using microservices). This allows the development of the components to be decoupled, and provides greater agility. It also introduces a new challenge and a new requirement for **decoupled communication** between components. For synchronous communication this would be HTTP, while with asynchronous this would be Messaging.

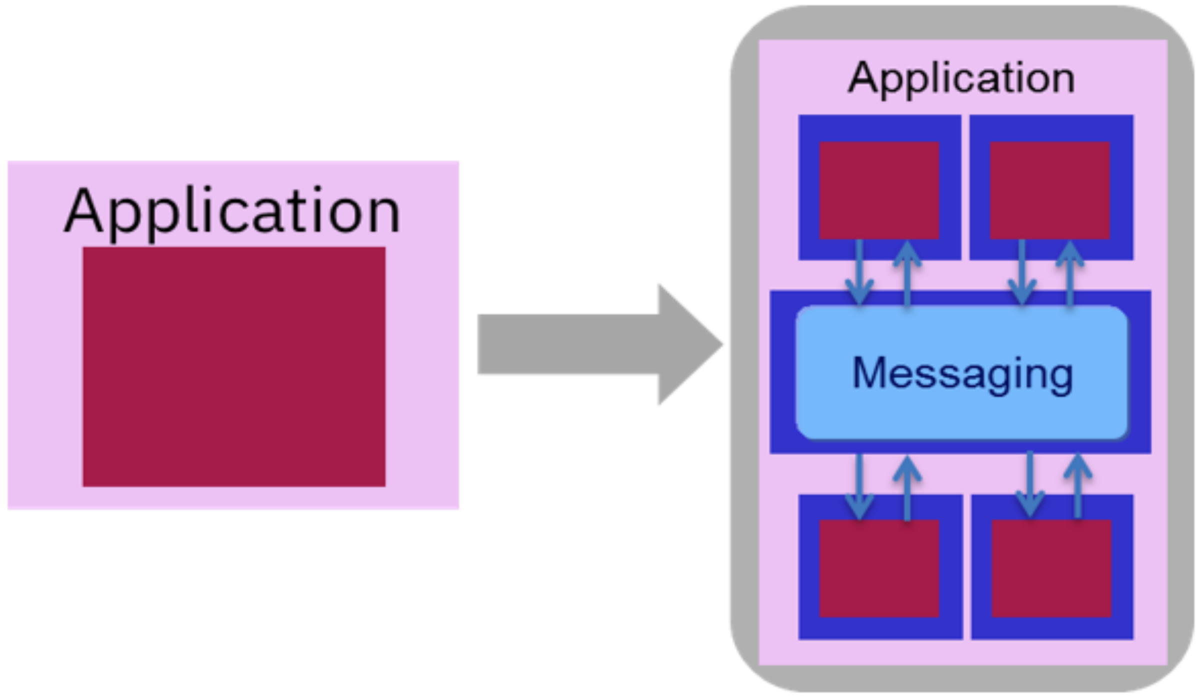

Figure 7: Application Modernization

Applying this principle to our architecture, we start to introduce additional IBM MQ instances within the application boundary, as illustrated below:

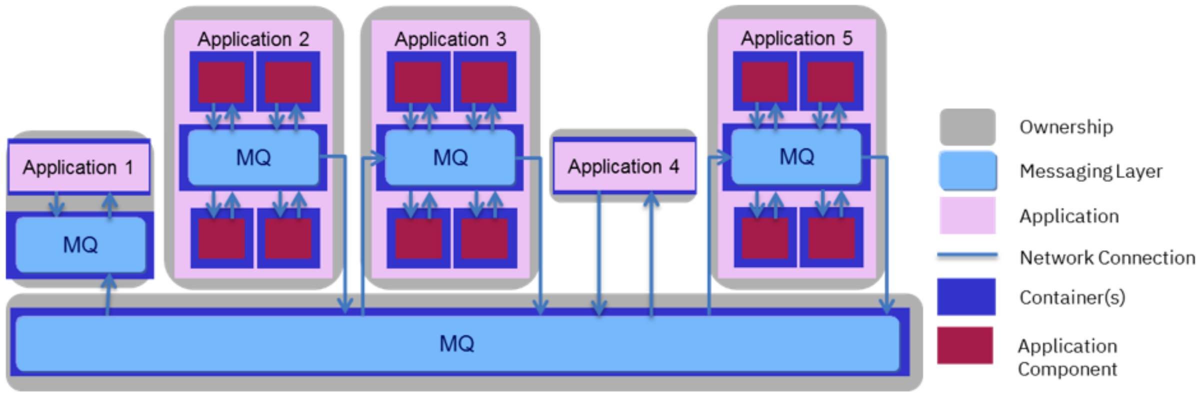

Figure 8: Decentralized IBM MQ Containerized Architecture

Taking a closer look, we can see that a number of the applications have been modernized, moving away from a monolithic application into a modern multi-component architecture (applications 2 & 3). While other applications (applications 1 & 4) have remained as before, this is likely to be the case in many organizations, where application modernization only occurs after a cost benefit analysis. The new application (application 5) shows that we expect new applications to be built using the modern architecture approach.

Many experienced IBM MQ clients may see similarities between this decentralized pattern and those of a traditional architecture, where the Application and MQ are co-located on the same machine. Some clients have raised a number of concerns regarding this traditional architecture as their deployments mature:

- **Licensing**: with the traditional architecture you licensed the whole machine hosting the application and MQ products. For instance if the application logic requires a large number of cores, the same number of cores need to be licensed for MQ. Within a container architecture this is overcome, as the MQ and application logic is hosted within different containers, and will be able to be licenced separately. For further information regarding the license model please consult https://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=an&subtype=ca&appname=gpateam&supplier=897&letternum=ENUS218-069.
- **Managing the estate**: the traditional architecture proliferated the IBM MQ instances across the organization, onto various machines and platforms. When administrators of the estate attempted to monitor or roll-out updates this involved significant effort due to the locations and interdependencies. Moving forward container platforms such as IBM Cloud Private provide a single platform for this administration, and therefore reduces the effort and time associated with this management.

As discussed previously, if the decentralized aspect is important to your organization will depend on the benefits that you want to realise from a containerization strategy. Each organization will need to develop a customized adoption journey.

Aspect: Optimized

The **optimized** aspect delivers two benefits, **availability** and **scalability**:

Continuous Availability: In messaging when required we separate out the availability to PUT (send) a message, from GETting (retrieving) a message. Many use cases want to assure an application is able to offload work (PUT a message), and therefore has a higher level of availability compared to the ability to GET (retrieve) the message. We call this higher level of availability for offloading work, **continuous availability**.

A **single resilient queue manager** will be automatically restarted in the event of a failure. During the restart there will be a small period of time that the queue manager is not available to receive new messages. This can range from a second or two to a few minutes under some situations (for example where millions of messages have built up on queues). This affects the overall availability of the messaging system so to achieve continuous availability we create at least two **single resilient queue managers**, which both have queues that can store messages. In this configuration it is expected that at least one queue manger is available at any one time to receive inbound messages.

For applications storing messages, these connections will be distributed across the available MQ instances. While connections for applications retrieving messages will be directly to a MQ instance. This is due to the need to connect to the MQ instance hosting the individual queue to retrieve messages. We will discuss later the options for this connection routing. This is illustrated in the figure below:

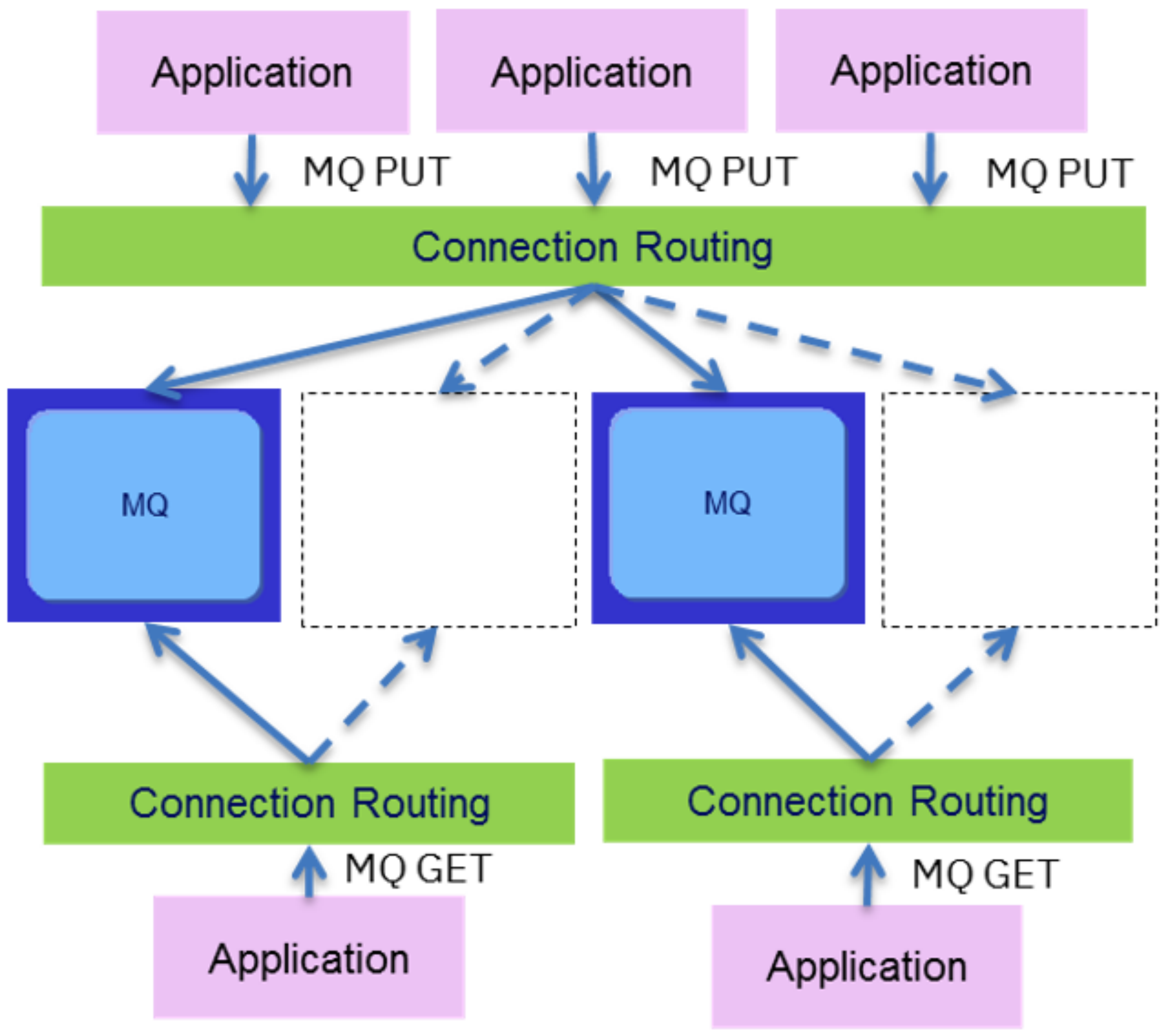

Figure 9: Multiple resilient containers

Scalability: is focused on the ability to scale the MQ instance horizontally and vertically. It is assumed that vertical scaling has been completed, and horizontal scaling is required. This is logically completed in a similar manner to the continuous availability part, as illustrated in the figure below:

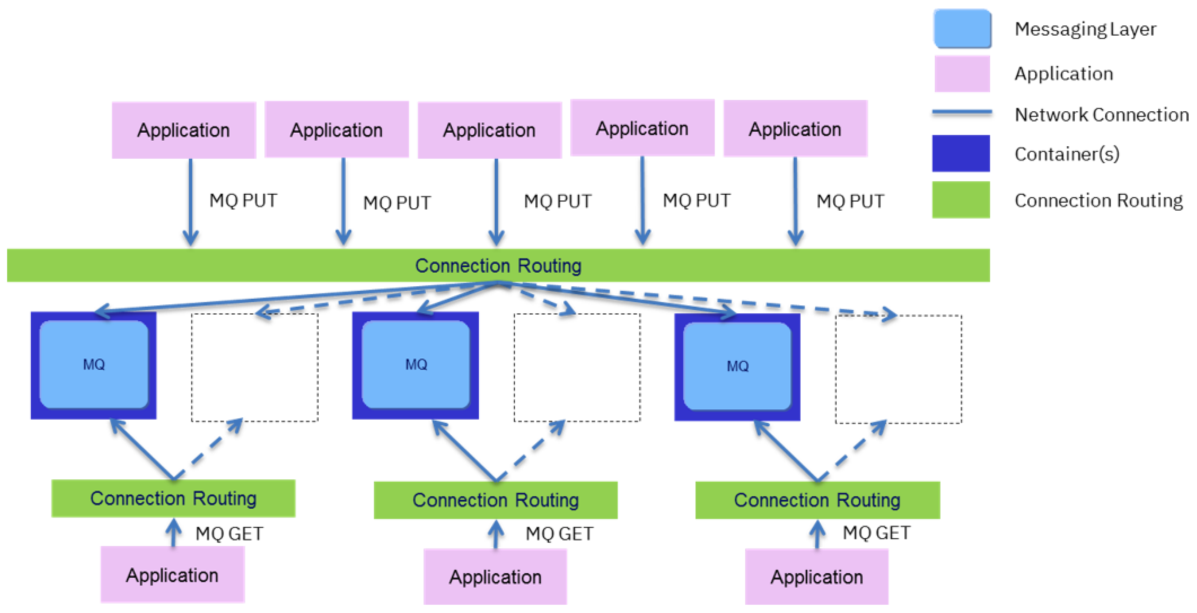

Figure 10: Scaling MQ in containers

While discussing the availability and scalability, we included the term **Connection Routing**, this provides the ability for the application to have its MQ traffic directed to one of multiple possible network destinations. We recommend for a production environment using a capability called Client Connection Definition Tables (CCDT) when providing connection routing across multiple possible MQ instances, while using the built in IP Gateway (such as a Kubernetes Service) of the container orchestrator platform when determining the active location of the container.

CCDT is a file that determines the connection information used by a client to connect to a Queue Manager. A CCDT file can contain multiple entries, for a single logical connection, allowing it to distribute traffic across a number of queue managers. The CCDT can be configured so that channels in a group are either sequentially tried (for availability), or randomly tried based on specified weightings (for workload balancing). The following figure shows the connection routing using these recommendations:

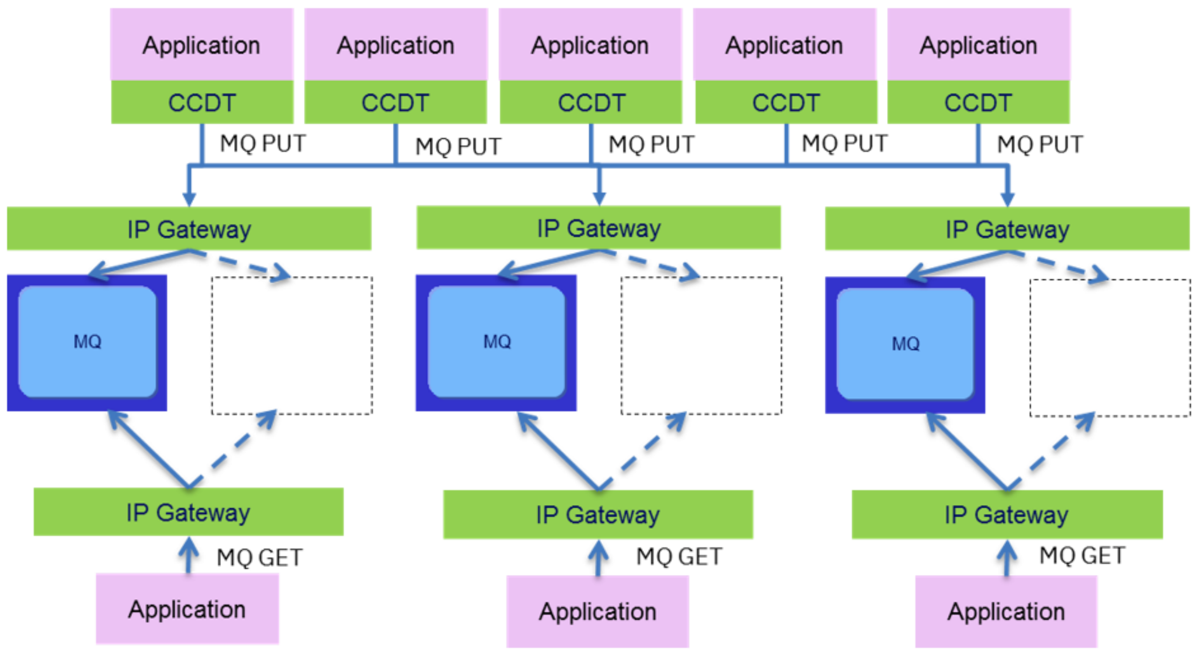

Figure 10: Routing MQ traffic within containers

The CCDT could be replaced with a container orchestration load balancing component, however this introduces a number of limitations on the applications which are documented within the ***IBM MQ as a Service Redbook*** (http://www.redbooks.ibm.com/redpapers/pdfs/redp5209.pdf) section 7.3.

Conclusion

This document discussed the high level aspects for the adoption of containers within the context of IBM MQ. Each organization will need to customize these aspects based on the benefits they want to realise, to establish their own journey. For further technical considerations please consult our technical guide here: https://developer.ibm.com/messaging/2018/05/02/availability-scalability-ibm-mq-containers/