10.0

IBM MQ in containers

Note

Before using this information and the product it supports, read the information in “Notices” on page 335.

This edition applies to version 9 release 4 of IBM® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© Copyright International Business Machines Corporation 2007, 2026.
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

IBM MQ in containers and IBM Cloud Pak for Integration..... 5
About..... 5
Release history for IBM MQ Operator..... 5
Release history for queue manager container images..... 7
Planning..... 9
Choosing how you want to use IBM MQ in containers..... 9
Support..... 10
Licensing..... 17
Storage..... 18
High availability..... 20
Disaster recovery..... 25
Security..... 29
Scalability and performance..... 38
Networking..... 39
Preparing, installing and upgrading..... 41
Dependencies for the IBM MQ Operator..... 41
Cluster-scoped permissions required by the IBM MQ Operator..... 42
Installing and upgrading the IBM MQ Operator on Red Hat OpenShift..... 42
Installing and upgrading the IBM MQ Operator on Amazon EKS..... 66
Verifying image signatures..... 84
Preparing to use Kubernetes by creating a pull secret..... 86
Preparing to use Kubernetes by installing the IBM License Service..... 86
Preparing to use Podman or Docker by pulling the prebuilt container image..... 87
Preparing for IBM MQ by building your own container image..... 87
Deploying and configuring..... 96
Deploying and configuring queue managers using the IBM MQ Operator..... 97
Self-deploying and configuring queue managers on Kubernetes..... 200
Self-deploying queue managers on Podman..... 215
Migrating to the IBM MQ Operator on Red Hat OpenShift..... 220
Checking that required functions are available..... 221
Extracting the queue manager configuration..... 222
Optional: Extracting and acquiring the queue manager keys and certificates..... 222
Optional: Configuring LDAP..... 224
Optional: Changing the IP addresses and host names in the IBM MQ configuration..... 232
Updating the queue manager configuration for a container environment..... 233
Selecting the target HA architecture for IBM MQ running in containers..... 236
Creating the resources for the queue manager..... 236
Creating the new queue manager on Red Hat OpenShift..... 238
Verifying the new container deployment..... 242
Operating..... 243
Operating IBM MQ using the IBM MQ Operator..... 243
Viewing the status of Native HA queue managers..... 258
Native HA CRR switchover and failover..... 264
Ending Native HA queue managers..... 268
Troubleshooting..... 269
Troubleshooting unplanned restarts of IBM MQ in containers..... 269
Collecting troubleshooting information with IBM MQ container inspector..... 271
Troubleshooting problems with the IBM MQ Operator..... 274
Collecting troubleshooting information for self-deployed queue managers on Kubernetes..... 280
Reference..... 281
Licensing and API references for the IBM MQ Operator..... 281
IBM MQ container license annotations..... 314

IBM MQ container license annotations that are applied by the IBM MQ Operator....319
IBM MQ Advanced container image....322
IBM MQ Advanced for Developers container image....326
IBM MQ MFT Agent for Developers container image....327
IBM MQ CASE for air-gap install and upgrade....329
IBM MQ Operator Helm chart values....332

Notices....335
Programming interface information....336
Trademarks....336

IBM MQ in containers and IBM Cloud Pak for Integration

Containers allow you to package an IBM MQ queue manager or IBM MQ client application, with all of its dependencies, into a standardized unit for software development.

You can run IBM MQ using the IBM MQ Operator on Red Hat® OpenShift® and, from IBM MQ Operator 3.9.0, Amazon Elastic Kubernetes Service (EKS). This can be done using IBM Cloud Pak® for Integration, IBM MQ Advanced, or IBM MQ Advanced for Developers.

You can also run IBM MQ in a container you build yourself.

For more information about the IBM MQ Operator, see the following links.

About IBM MQ in containers

Introductory information to help you get started with IBM MQ in containers.

Containers are a technology to allow the packaging and isolation of code with its runtime environment, which can be run in a way which is isolated from other software on the same infrastructure. This makes it easy to move a queue manager or application between environments (such as dev, test and production). Modern container orchestrators, such Red Hat OpenShift Container Platform and Kubernetes can run many types of containers on the same machine, each isolated from each other in terms of resources, security and failures.

You can run IBM MQ queue managers or your IBM MQ applications in containers.

Related information

What are containers?

Release history for IBM MQ Operator

Notes:

- For information about security fixes, see the security bulletin for each release. Security bulletins are usually available within a few days of a release.
- For information about earlier IBM MQ Operators, see Release history for IBM MQ Operator in the IBM MQ 9.4 documentation.
- For information about future IBM MQ updates, see the overall IBM MQ Recommended Fixes and Planned Maintenance release dates page.

IBM MQ Operator 4.0.2

IBM Cloud Pak for Integration version
IBM Cloud Pak for Integration 16.2.0

Operator channel
`v4.0-sc2`

Allowed values for `.spec.version`
“10.0.0.0-r3” on page 7

Allowed values for `.spec.version` during migration

9.4.0.0-r1, 9.4.0.0-r2, 9.4.0.0-r3, 9.4.0.5-r1, 9.4.0.5-r2, 9.4.0.6-r1, 9.4.0.6-r2, 9.4.0.7-r1, 9.4.0.10-r1, 9.4.0.10-r2, 9.4.0.11-r1, 9.4.0.11-r2, 9.4.0.11-r3, 9.4.0.12-r1, 9.4.0.15-r1, 9.4.0.15-r2, 9.4.0.15-r3, 9.4.0.15-r4, 9.4.0.16-r1, 9.4.0.16-r2, 9.4.0.17-r1, 9.4.0.17-r2, 9.4.0.20-r1, 9.4.0.21-r1, 9.4.0.21-r2, 9.4.0.25-r1, 9.4.1.0-r1, 9.4.1.0-r2, 9.4.1.1-r1, 9.4.2.0-r1, 9.4.2.0-r2, 9.4.2.1-r1, 9.4.2.1-r2,

© Copyright IBM Corp. 2007, 2026

9.4.3.0-r1, 9.4.3.0-r2, 9.4.3.1-r1, 9.4.3.1-r2, 9.4.3.1-r3, 9.4.4.0-r1, 9.4.4.0-r2, 9.4.4.0-r3, 9.4.4.0-r4, 9.4.4.1-r1, 9.4.5.0-r1, 9.4.5.0-r2, 9.4.5.1-r1, 10.0.0.0-r1, “10.0.0.0-r2” on page 8

Red Hat OpenShift Container Platform versions
OpenShift Container Platform 4.12 and later. For information about the version compatibility for the IBM MQ Operator see Where to find product requirements and support information.

Kubernetes versions on Amazon EKS
1.31 and later. For information about the version compatibility for the IBM MQ Operator see Where to find product requirements and support information.

What's changed

- Security fixes (see the link at the top of this topic).

IBM MQ Operator 4.0.1

IBM Cloud Pak for Integration version
IBM Cloud Pak for Integration 16.2.0

Operator channel
`v4.0-sc2`

Allowed values for `.spec.version`
“10.0.0.0-r2” on page 8

Allowed values for `.spec.version` during migration

9.4.0.0-r1, 9.4.0.0-r2, 9.4.0.0-r3, 9.4.0.5-r1, 9.4.0.5-r2, 9.4.0.6-r1, 9.4.0.6-r2, 9.4.0.7-r1, 9.4.0.10-r1, 9.4.0.10-r2, 9.4.0.11-r1, 9.4.0.11-r2, 9.4.0.11-r3, 9.4.0.12-r1, 9.4.0.15-r1, 9.4.0.15-r2, 9.4.0.15-r3, 9.4.0.15-r4, 9.4.0.16-r1, 9.4.0.16-r2, 9.4.0.17-r1, 9.4.0.17-r2, 9.4.0.20-r1, 9.4.0.21-r1, 9.4.0.21-r2, 9.4.0.25-r1, 9.4.1.0-r1, 9.4.1.0-r2, 9.4.1.1-r1, 9.4.2.0-r1, 9.4.2.0-r2, 9.4.2.1-r1, 9.4.2.1-r2, 9.4.3.0-r1, 9.4.3.0-r2, 9.4.3.1-r1, 9.4.3.1-r2, 9.4.3.1-r3, 9.4.4.0-r1, 9.4.4.0-r2, 9.4.4.0-r3, 9.4.4.0-r4, 9.4.4.1-r1, 9.4.5.0-r1, 9.4.5.0-r2, 9.4.5.1-r1, “10.0.0.0-r1” on page 8

Red Hat OpenShift Container Platform versions
OpenShift Container Platform 4.12 and later. For information about the version compatibility for the IBM MQ Operator see Where to find product requirements and support information.

Kubernetes versions on Amazon EKS
1.30 and later. For information about the version compatibility for the IBM MQ Operator see Where to find product requirements and support information.

What's changed

- Security fixes (see the link at the top of this topic).

IBM MQ Operator 4.0.0

IBM Cloud Pak for Integration version
IBM Cloud Pak for Integration 16.2.0

Operator channel
`v4.0-sc2`

Allowed values for `.spec.version`
“10.0.0.0-r1” on page 8

Allowed values for `.spec.version` during migration

9.4.0.0-r1, 9.4.0.0-r2, 9.4.0.0-r3, 9.4.0.5-r1, 9.4.0.5-r2, 9.4.0.6-r1, 9.4.0.6-r2, 9.4.0.7-r1, 9.4.0.10-r1, 9.4.0.10-r2, 9.4.0.11-r1, 9.4.0.11-r2, 9.4.0.11-r3, 9.4.0.12-r1, 9.4.0.15-r1, 9.4.0.15-r2, 9.4.0.15-r3, 9.4.0.15-r4, 9.4.0.16-r1, 9.4.0.16-r2, 9.4.0.17-r1, 9.4.0.17-r2, 9.4.0.20-r1, 9.4.0.21-r1, 9.4.0.21-r2, 9.4.1.0-r1, 9.4.1.0-r2, 9.4.1.1-r1, 9.4.2.0-r1, 9.4.2.0-r2, 9.4.2.1-r1, 9.4.2.1-r2, 9.4.3.0-r1,

9.4.3.0-r2, 9.4.3.1-r1, 9.4.3.1-r2, 9.4.3.1-r3, 9.4.4.0-r1, 9.4.4.0-r2, 9.4.4.0-r3, 9.4.4.0-r4, 9.4.4.1-r1, 9.4.5.0-r1, 9.4.5.0-r2, 9.4.5.1-r1

Red Hat OpenShift Container Platform versions
OpenShift Container Platform 4.12 and later. For information about the version compatibility for the IBM MQ Operator see Where to find product requirements and support information.

Kubernetes versions on Amazon EKS
1.30 and later. For information about the version compatibility for the IBM MQ Operator see Where to find product requirements and support information.

What's new

- From IBM MQ Operator 4.0.0, IBM Instana® tracing is available with IBM MQ and IBM MQ Advanced licenses in addition to IBM Cloud Pak for Integration licenses. IBM Instana tracing is still not available with IBM MQ Advanced for Developers licenses. To enable IBM Instana, see "Integrating IBM MQ with IBM Instana tracing" on page 177.

What's changed

- The IBM MQ 9.3.x versions, which were previously deprecated, are now removed.

  Important: Before you upgrade the IBM MQ Operator, ensure that you do not have queue managers for any of the removed versions. After you upgrade, you will no longer be able to edit the QueueManager resource (other than to upgrade to an in-support version) because the IBM MQ Operator no longer recognizes the older versions.
- From 10.0.0.0 onwards, queue managers that use an IBM Cloud Pak for Integration license no longer use `ibm-mqadvanced-server-integration` images. Instead, these queue managers use `ibm-mqadvanced-server` images, which now provide the equivalent functionality.
- IBM MQ Operator images from 4.0.0 onwards use `ubi-micro 10` base images instead of `ubi-minimal 9`.
- Documentation updates include the removal of the *Version support for the IBM MQ Operator* topic; the information within this topic is already available in these release notes and in the IBM MQ system requirements pages.
- Security fixes (see the link at the top of this topic).

OpenShift CP4I Release history for queue manager container images

Notes:

- For information about security fixes, see the security bulletin for each release. Security bulletins are usually available within a few days of a release.
- For information about earlier queue manager container images, see Release history for IBM MQ Operator in the IBM MQ 9.4 documentation.

10.0.0.0-r3

CP4I-SC2 CD

Compatible operator version
Supported standalone, or with "IBM MQ Operator 4.0.2" on page 5 or higher.

Supported architectures
`amd64`, `s390x`, `ppc64le`

Images

- `cp.icr.io/cp/ibm-mqadvanced-server:10.0.0.0-r3` (IBM MQ Advanced)
- `icr.io/ibm-messaging/mq:10.0.0.0-r3` (IBM MQ Advanced for Developers)

What's new

- What's new in IBM MQ 10.0 for Multiplatforms - base and Advanced entitlement.

What's changed

- What's changed in IBM MQ 10.0.0.

10.0.0.0-r2

CP4I-SC2 CD

Compatible operator version
Supported standalone, or with "IBM MQ Operator 4.0.1" on page 6 or higher.

Supported architectures
amd64, s390x, ppc64le

Images

- `cp.icr.io/cp/ibm-mqadvanced-server:10.0.0.0-r2` (IBM MQ Advanced)
- `icr.io/ibm-messaging/mq:10.0.0.0-r2` (IBM MQ Advanced for Developers)

What's new

- What's new in IBM MQ 10.0 for Multiplatforms - base and Advanced entitlement.

What's changed

- What's changed in IBM MQ 10.0.0.

10.0.0.0-r1

CP4I-SC2 CD

Compatible operator version
Supported standalone, or with "IBM MQ Operator 4.0.0" on page 6 or higher.

Supported architectures
amd64, s390x, ppc64le

Images

- `cp.icr.io/cp/ibm-mqadvanced-server:10.0.0.0-r1` (IBM MQ Advanced)
- `icr.io/ibm-messaging/mq:10.0.0.0-r1` (IBM MQ Advanced for Developers)

What's new

- What's new in IBM MQ 10.0 for Multiplatforms - base and Advanced entitlement.
- Language packs for languages that are supported by IBM MQ are added. For example, setting the environment variable `LANG=de_DE.utf8` configures the container to use the German language.
- A new `MQ_METRICS_REQUIRE_QUANTUM_SAFE` environment variable is added. When set to `true`, this variable configures the metrics server to accept only quantum-safe TLS connections.

What's changed

- What's changed in IBM MQ 10.0.0.
- The DT472893 upgrade issue is fixed by refreshing the WebSphere® Liberty `ltpa.keys` file. To prevent your existing `ltpa.keys` file from being deleted when you upgrade to this IBM MQ version, set the following environment variable: `AMQ_ENABLE_DT472893_REGENERATE_LTPA_KEYS=false`.
- Probe logging is added to capture Kubernetes liveness and startup probe execution events, with summary reporting available on container shutdown.
- FIPS mode checks are updated to ensure that FIPS mode is enabled only on supported platforms (amd64 and ppc64le). FIPS support is no longer available for s390x. If you require FIPS support on s390x, you must use a previous version of IBM MQ.
- The default locale is changed from `C` to `C.utf8`. As a result, new queue managers use CCSID 1208 (UTF-8) instead of CCSID 819 (ISO 8859-1).

- An initial key for Liberty AES encryption is now generated automatically. You can also provide this key by using a Secret, by mounting the initial key at `/run/secrets/initial.key`.
- The minimum TLS version for the queue manager metrics server is updated to 1.3.
- The metrics HTTPS server now uses the Go cryptography provider for FIPS compliance, instead of OpenSSL.
- The IBM MQ Advanced for Developers image now prioritizes secrets over environment variables for application and administrator user credentials.
- Version 3.8.0 (2026.1.3) of the IBM MQ tracing user exit is now included in the `ibm-mqadvanced-server` image.
- IBM MQ images from 10.0.0.0 onwards now use `ubi-micro 10` base images instead of `ubi-minimal 9`.
- Security fixes (see the link at the top of this topic).

Planning for IBM MQ in containers

When planning for IBM MQ in containers, consider the support that IBM MQ provides for various architectural options, such as how high availability is managed, and how to secure your queue managers.

About this task

Before you plan your IBM MQ in containers architecture, you should familiarize yourself with the basic IBM MQ concepts (see IBM MQ Technical overview) as well as basic Kubernetes, Red Hat OpenShift, and Amazon EKS concepts (see OpenShift Container Platform architecture or Amazon EKS architecture).

Procedure

- “Choosing how you want to use IBM MQ in containers” on page 9.
- “Support for IBM MQ in containers” on page 10.
- “Planning for licensing IBM MQ in containers” on page 17.
- “Planning storage for the IBM MQ Operator” on page 18.
- “Planning high availability for IBM MQ in containers” on page 20.
- “Disaster recovery for IBM MQ in containers” on page 25.
- “Planning security for IBM MQ in containers” on page 29.
- “Planning networking for IBM MQ in containers” on page 39.

Choosing how you want to use IBM MQ in containers

There are multiple options for using IBM MQ in containers: you can choose to use the IBM MQ Operator, or you can deploy IBM MQ using your own deployment code.

Deploy using the IBM MQ Operator

The IBM MQ Operator automates the creation, configuration, and management of queue managers. It offers several key features:

1. The IBM MQ Operator offers an abstraction of the lower level Kubernetes resources, so that you don't need such deep Kubernetes skills. The operator adds a new `QueueManager` custom resource and watches for new queue manager definitions, turning them into the necessary low-level resources. The IBM MQ Operator continues to observe the low-level resources under its control, to prevent configuration drift if they were to be edited manually.

2. For high availability, the operator can also perform the rolling update of queue manager instances. For example, managing a rolling update across three Native HA instances, in a way which minimizes fail-overs (for example, by upgrading failed Pods first, then replica Pods, then the active Pod).

You can deploy the IBM MQ Operator on the following platforms:

- Red Hat OpenShift Container Platform
- Amazon Elastic Kubernetes Service (from IBM MQ Operator 3.9.0)

For more information, see “Support for the IBM MQ Operator” on page 11.

Deploy using your own deployment code

There are several key considerations for writing your own code to self-deploy IBM MQ in containers:

1. Which container image to use:
   a. V 10.0.0 **Prebuilt images**. From IBM MQ 9.4.1, you can use the prebuilt images that are available from the IBM Container Registry. For more information, see “Support for the IBM MQ Advanced container images” on page 12 and “Support for IBM MQ licensed container images” on page 13.
   b. **Build your own images**. This is required for IBM MQ 9.4.0 and earlier, but you can still choose to do this on later versions. This option offers flexibility to choose your own base image, to install any IBM MQ component, or alter anything else about the image that you require. It requires you to "own" the resultant container. See “Support for building your own IBM MQ container images” on page 15 for details of what is supported when building your own image and deployment code. Samples for building your own images are available. See “Preparing for IBM MQ by building your own container image” on page 87.
2. You will need to manage and support your own deployment code with this option, for example using **`helm`**, **`kustomize`**, **`terraform`** or **`ansible`**. This gives a lot of flexibility over how MQ is deployed, but requires you to have the right skills in configuring containers. IBM provides defect and usage support for the IBM-supplied MQ container image, including IBM MQ itself, but does not help users configure and debug problems with their container environment or deployment pipeline. Examples of how to deploy into Kubernetes are provided in the documentation. There is also an unsupported sample Helm chart. For more information, see “Self-deploying and configuring queue managers on Kubernetes” on page 200, and “Self-deploying and configuring queue managers using the sample Helm chart” on page 215
3. Rolling updates for highly available queue managers need to be carefully managed. For example, the standard Kubernetes rolling update will update Pods in a fixed order, regardless of which one is the "active" Pod, which can lead to multiple fail-overs. For more details, see “Considerations for performing your own rolling update of a Native HA queue manager” on page 24.

Related reference

“Support for IBM MQ in containers” on page 10
Not all IBM MQ features are available and supported in the same way in containers.

Support for IBM MQ in containers

Not all IBM MQ features are available and supported in the same way in containers.

Related concepts

IBM MQ FAQ for Long Term Support and Continuous Delivery releases

Related reference

IBM Cloud Pak for Integration Software Support Lifecycle Addendum

Related information

IBM MQ support for SELinux and AppArmor

Support for the IBM MQ Operator

The IBM MQ Operator is supported as part of IBM MQ, IBM MQ Advanced, and IBM Cloud Pak for Integration

Supported platforms

The IBM MQ Operator is supported on Red Hat OpenShift Container Platform and Amazon EKS.

- Releases of Red Hat OpenShift Container Platform are no longer supported by IBM MQ after Red Hat stops support. See Red Hat OpenShift Container Platform Life Cycle Dates for more details.
- Releases of Amazon EKS are no longer supported by IBM MQ after Amazon stops both standard and extended support. See Understand the Kubernetes version lifecycle on EKS for more details.

For more information, see the **Containers** tab of the relevant IBM MQ system requirements page.

Important: Red Hat provides support for the Red Hat packages only for containers running on Red Hat Enterprise Linux® or Red Hat Enterprise Linux CoreOS hosts. See Red Hat Container Support Policy for more information.

Supported CPU architectures

The IBM MQ Operator is supported on `amd64` and `s390x` z/Linux, and on `ppc64le` Power® Systems version 9 and above. Note that all nodes in a Red Hat OpenShift or Amazon EKS cluster must use the same CPU architecture.

Duration of support

IBM provides defect and usage support for the IBM MQ Operator and the associated queue manager images.

There are two types of release for IBM MQ in containers: Container Long Term Support (Container-LTS) and Continuous Delivery (CD). See IBM MQ release types and versioning for how to distinguish release versions, and details of support duration.

For CD releases, security fixes are pro-actively applied until the next CD release level. After the next CD release, on request IBM will assess the feasibility of making reasonable efforts to provide a fix on the previous CD release level.

See IBM MQ FAQ for Long Term Support and Continuous Delivery releases for more information on support, and Lifecycle dates for IBM MQ in containers for specific dates.

Fix availability

Periodic and interim queue manager fixes are available from IBM Container Registry. Interim fixes for the IBM MQ Operator are not available.

Unavailable features

The following features of IBM MQ Advanced are not available for queue managers operated by the IBM MQ Operator:

- Replicated Data Queue Managers (RDQM) are not available - this feature is tightly coupled with the Linux kernel, so it not supported in containers
- Linear logging for queue managers is not available
- Specifying custom command line options for **`crtmqdir`**, **`crtmqm`**, **`strmqm`** and **`endmqm`** are not available — Most options can be configured using an INI file, however some cannot be configured, such as the use of linear logging.
- Operating system (OS) users are not available
- AMQP is not installed
- MQTT is not installed

- Managed File Transfer agent is not available. However, you can use the IBM MQ Operator to provide one or more Coordination, Command, or Agent queue managers.
- Native HA In-Region Replication (IRR) is not supported
- Advanced Message Security with MCA interception using a separate key store is not available. MCA interception can still be configured to use the main queue manager key store.

You can only use the queue manager container images supplied by IBM with the MQ Operator, and cannot build your own image with custom code. The exception to this is to add a container image layer to an IBM-supplied image, in order to add configuration files or exits.

Related concepts

IBM MQ FAQ for Long Term Support and Continuous Delivery releases

Related reference

IBM Cloud Pak for Integration Software Support Lifecycle Addendum

Related information

Lifecycle dates for IBM MQ in containers

##

Support for the IBM MQ Advanced container images

You can use the prebuilt IBM MQ Advanced container image without requiring the IBM MQ Operator.

The prebuilt IBM MQ Advanced container image is made up of several logical layers:

- IBM MQ container enablement code (`runmqserver`, `chkmqhealthy`, `chkmqready` and `chkmqstarted`)
- IBM MQ Advanced itself
- Red Hat packages from the Red Hat Universal Base Image

Supported platforms

The prebuilt IBM MQ Advanced image can be deployed into container runtime environments provided by containerd, cri-o, Podman or Docker. For more details, see System Requirements for IBM MQ. IBM provides defect and usage support for the IBM-supplied MQ container enablement code and IBM MQ Advanced itself, but does not help users configure and debug problems with their container environment or deployment pipeline.

Important: Red Hat provides support for the Red Hat packages only for containers running on Red Hat Enterprise Linux or Red Hat Enterprise Linux CoreOS hosts. See Red Hat Container Support Policy for more information.

Licensing

The prebuilt IBM MQ Advanced image is eligible for container licensing if used on Kubernetes with the IBM License Service. If you are not employing Kubernetes orchestration (for example, using Podman or Docker), then you cannot take advantage of this optimized licensing model. See Container licensing FAQs for more details.

Supported CPU architectures

The prebuilt IBM MQ Advanced image is supported on `amd64` and `s390x` z/Linux, and on `ppc64le` Power Systems version 9 and above. Note that all containers which are part of the same queue manager (for example, Native HA replicas) must use the same CPU architecture.

Duration of support

IBM provides defect and usage support for the prebuilt queue manager container images.

There are two types of release for IBM MQ in containers: Container Long Term Support (Container-LTS) and Continuous Delivery (CD). See IBM MQ release types and versioning for how to distinguish release versions, and details of support duration.

For CD releases, security fixes are pro-actively applied until the next CD release level. After the next CD release, on request IBM will assess the feasibility of making reasonable efforts to provide a fix on the previous CD release level.

See IBM MQ FAQ for Long Term Support and Continuous Delivery releases for more information on support, and Lifecycle dates for IBM MQ in containers for specific dates.

Fix availability

Periodic and interim fixes are available from IBM Container Registry.

Unavailable features

The following features of IBM MQ Advanced are not available with the prebuilt container image:

- Replicated Data Queue Managers (RDQM) are not available - this feature is tightly coupled with the Linux kernel, so it not supported in containers
- Linear logging for queue managers is not available
- Specifying custom command line options for **crtmqdir**, **crtmqm**, **strmqm** and **endmqm** are not available — Most options can be configured using an INI file, however some cannot be configured, such as the use of linear logging.
- Operating system (OS) users are not available
- AMQP is not installed
- MQTT is not installed
- Managed File Transfer agent is not available. However, you can use the prebuilt container image to provide one or more Coordination, Command, or Agent queue managers.
- Native HA In-Region Replication (IRR) is not supported.

If you want to use these features, you can build your own container image.

Related concepts

IBM MQ FAQ for Long Term Support and Continuous Delivery releases

Red Hat Container Support Policy

Related information

Lifecycle dates for IBM MQ in containers

Support for IBM MQ licensed container images

You can use the prebuilt IBM MQ licensed container image without requiring the IBM MQ Operator.

The prebuilt IBM MQ container image is made up of several logical layers:

- IBM MQ container enablement code (`runmqserver`, `chkmqhealthy`, `chkmqready` and `chkmqstarted`)
- IBM MQ itself
- Red Hat packages from the Red Hat Universal Base Image

Note: The IBM MQ licensed container image uses the IBM MQ Advanced image, though features that are specific to IBM MQ Advanced cannot be used without appropriate licenses. For more information, see the “Unavailable features from IBM MQ Advanced” on page 14 section later in this topic.

Supported platforms

The prebuilt IBM MQ image can be deployed into container runtime environments that are provided by containerd, cri-o, Podman or Docker. For more details, see System Requirements for IBM MQ. IBM provides defect and usage support for the IBM-supplied IBM MQ container enablement code and IBM

MQ itself, but does not help users configure and debug problems with their container environment or deployment pipeline.

Important: Red Hat provides support for the Red Hat packages only for containers that are running on Red Hat Enterprise Linux or Red Hat Enterprise Linux CoreOS hosts. See Red Hat Container Support Policy for more information.

Licensing

The prebuilt IBM MQ image is eligible for container licensing if used on Kubernetes with the IBM License Service. If you are not employing Kubernetes orchestration (for example, using Podman or Docker), then you cannot take advantage of this optimized licensing model. See Container licensing FAQs for more details.

Supported CPU architectures

The prebuilt IBM MQ image is supported on `amd64` and `s390x` z/Linux, and on `ppc64le` Power Systems version 9 and above. Note that all containers that are part of the same queue manager (for example, Native HA replicas) must use the same CPU architecture.

Duration of support

IBM provides defect and usage support for the prebuilt queue manager container images.

There are two types of release for IBM MQ in containers: Container Long Term Support (Container-LTS) and Continuous Delivery (CD). See IBM MQ release types and versioning for how to distinguish release versions, and details of support duration.

For CD releases, security fixes are pro-actively applied until the next CD release level. After the next CD release, on request IBM will assess the feasibility of making reasonable efforts to provide a fix on the previous CD release level.

See IBM MQ FAQ for Long Term Support and Continuous Delivery releases for more information on support, and Lifecycle dates for IBM MQ in containers for specific dates.

Fix availability

Periodic and interim fixes are available from IBM Container Registry.

Unavailable features from IBM MQ Advanced

The following features of IBM MQ Advanced cannot be used with the prebuilt IBM MQ licensed container image:

- Native HA and Cross-Region Replication. To use Native HA and Cross-Region Replication features on an IBM MQ licensed queue manager, see “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.
- AMS
- Managed File Transfer support for OpenSSH type Private key for SFTP servers
- Autorecovery of damaged objects for Native HA queue managers
- Media image scheduling - replicated logging
- RDQM available on Red Hat Enterprise Linux 9
- Kafka Connect

If you want to use these features, you can deploy IBM MQ Advanced. For more information, see Support for the IBM MQ Advanced container images.

Linux Support for building your own IBM MQ container images

The IBM MQ container sample provides unsupported sample wrapper code for enabling IBM MQ in a container. This is the most flexible container solution, but it requires you to have strong skills in configuring containers, and to "own" the resultant container.

Downloads

The IBM MQ container sample is available here: https://github.com/ibm-messaging/mq-container.

Note: The container sample, and any code you add or modify, is not supported by IBM.

For use with the container sample, you download an IBM MQ non-install image.

- If you have an IBM MQ Advanced for Developers license, use the Continuous Delivery (CD) version of IBM MQ. These non-install images are available from Downloading IBM MQ 10.0 CD.
- If you have a Production license, use either the Long Term Support (LTS) or CD version of IBM MQ. These non-install images are available from Fix Central.

Supported platforms

See System Requirements for IBM MQ for detailed information on the supported container runtime environments and Linux distributions. IBM MQ can be used with container runtime environments provided by containerd, cri-o, Podman or Docker. IBM provides defect and usage support for the core IBM MQ software, but does not help you configure and debug problems with your container environment or deployment pipeline.

The container sample uses a Red Hat Universal Base Image as a base image. Red Hat provides support for the Red Hat packages only for containers running on Red Hat Enterprise Linux or Red Hat Enterprise Linux CoreOS hosts. See Red Hat Container Support Policy for more information. You can build your own container image on any Linux base image that matches a Linux distribution supported by IBM MQ, for example:

- Red Hat Universal Base Image
- Ubuntu
- SUSE Linux Enterprise Server Base Container Image

Supported CPU architectures

See System Requirements for IBM MQ.

Duration of support for IBM MQ

Support is provided for the Production versions of the IBM MQ non-install images. For LTS, support duration is 5 years from General Availability. For CD, each incremental release is supported for 1 year from General Availability. For more information, see the IBM MQ FAQ for Long Term Support and Continuous Delivery releases.

Fix availability for IBM MQ

For the Production versions of the IBM MQ non-install images, fixes are provided throughout the duration of support. You can also request fixes. If a CD update is no longer eligible for defect fixes, update to a version that is eligible, and if the defect can be recreated a fix will be provided. For information on available LTS and CD releases and latest fixes, see IBM MQ fixes by version.

Unavailable features

You cannot use Operating System (OS) users with an IBM MQ non-install image. If you want to use OS users, then you need to build your own image using the RPM installer, for which there is no sample.

See System Requirements for IBM MQ for full details of supported components. There are some samples available, but not for every component.

Related concepts

IBM MQ FAQ for Long Term Support and Continuous Delivery releases

Related tasks

“Preparing for IBM MQ by building your own container image” on page 87

Develop a self-built container. This is the most flexible container solution, but it requires you to have strong skills in configuring containers, and to "own" the resultant container.

MQ Adv. Linux Availability of the IBM MQ Advanced for Developers container image

The prebuilt IBM MQ Advanced for Developers image is available from IBM Container Registry: `icr.io/ibm-messaging/mq`. The image is a free download and is non-warranted. **The image is not supported.**

Available platforms

The prebuilt IBM MQ Advanced for Developers image can be deployed into container runtime environments provided by containerd, cri-o, Podman or Docker. For more details, see System Requirements for IBM MQ.

Available CPU architectures

The prebuilt IBM MQ Advanced for Developers image is available on `amd64` and `s390x` z/Linux, and on `ppc64le` Power Systems version 9 and above. Note that all containers which are part of the same queue manager (for example, Native HA replicas) must use the same CPU architecture.

Unavailable features

The following features of IBM MQ Advanced for Developers are not available with the prebuilt container image:

- Replicated Data Queue Managers (RDQM) are not available - this feature is tightly coupled with the Linux kernel, so it not supported in containers
- Linear logging for queue managers is not available
- Specifying custom command line options for **`crtmqdir`**, **`crtmqm`**, **`strmqm`** and **`endmqm`** are not available — Most options can be configured using an INI file, however some cannot be configured, such as the use of linear logging.
- Operating system (OS) users are not available
- AMQP is not installed
- MQTT is not installed
- Managed File Transfer agent is not available. However, you can use the prebuilt container image to provide one or more Coordination, Command, or Agent queue managers.

Related concepts

IBM MQ FAQ for Long Term Support and Continuous Delivery releases

“IBM MQ Advanced for Developers container image” on page 326

A prebuilt container image is available for IBM MQ Advanced for Developers. This image is available from the IBM Container Registry. This image is suitable for use with Docker, Podman, Kubernetes, and other container environments.

Editing resources created by the IBM MQ Operator

The IBM MQ Operator reconciles a QueueManager custom resource by creating and managing native Kubernetes resources. These resources managed **must not** be edited directly.

You can usually determine if a resource is owned by another higher-level resource, by looking at the `ownerReferences`. For example, the following metadata taken from a `StatefulSet` shows that it is owned by the `QueueManager` resource "qm1":

```
metadata:
  ownerReferences:
    - apiVersion: mq.ibm.com/v1beta1
      kind: QueueManager
      name: qm1
      uid: 60fda34c-9f7c-42d2-a293-78fec4315c62
      controller: true
      blockOwnerDeletion: true
```

Note that not all resources have this metadata.

It is the responsibility of the IBM MQ Operator to manage the underlying resources, such as `StatefulSet`, `Service` and `Route`. If you change any of these underlying resources, the IBM MQ Operator will change them back, and you may experience downtime if that change requires a rolling update.

Most of the important settings for queue managers are available on the `QueueManager` resource. However, if you find that you need full control of the underlying resources, there are some options:

- If you need to override the settings on the Pod created by the IBM MQ Operator, then you can add a Pod override template in the `.spec.template` section of the `QueueManager` YAML.
- If you deployed on Red Hat OpenShift Container Platform and need to override settings on the queue manager `Route` that is created by the IBM MQ Operator, disable the route entirely by changing the `.spec.route.enabled` setting to `false`, then create your own Route.
- Settings such as labels and annotations, and Pod settings like `security Context`, can be set on the `QueueManager` resource.
- In other cases, the IBM MQ Operator might not be appropriate for your use case if you need full control.

Planning for licensing IBM MQ in containers

Container licensing allows you to license only the available capacity of your individual IBM MQ containers, rather than requiring you to license the entire server where your containers are running. To take advantage of container licensing, the IBM License Service must be used to track license usage and determine your required entitlement.

Note that container-based licensing is available only if you are on Red Hat OpenShift Container Platform or Kubernetes, including Amazon EKS. For more information, see Container licensing FAQs for more details.

To use the IBM License Service:

- If you are using the IBM MQ Operator, see the *Deploy the License Service* step of the IBM MQ Operator installation process that is appropriate for your platform:
  - Red Hat OpenShift.
  - Amazon EKS.
- If you are self-deploying queue managers (in other words, you are not using the IBM MQ Operator), on Kubernetes, see Installing License Service on the Kubernetes cluster in the IBM Cloud Pak foundational services documentation.

If you are not using Red Hat OpenShift Container Platform or Kubernetes (including Amazon EKS), you must use the IBM License Metric Tool. This tool is required for metering and reporting license usage of IBM products that are deployed in traditional virtualization environments to take advantage of sub-capacity licensing. For more information, see IBM License Metric Tool documentation.

Related reference

"IBM MQ container license annotations" on page 314
In Kubernetes and Red Hat OpenShift Container Platform environments, the IBM License Service uses Pod annotations to track usage based on the limits defined on the container, rather than on the underlying machine. The IBM MQ Operator sets these Pod annotations for you, but when you deploy using other means, you must set them yourself.

Related information

IBM Container Licenses

Container licensing FAQs

Installing License Service

Viewing and tracking license usage

Planning storage for the IBM MQ Operator

When you plan storage, consider aspects such as storage type, capacity, encryption, and any requirements that are specific to your cloud platform.

The IBM MQ Operator can configure IBM MQ queue managers to run in one of two storage modes:

- **Ephemeral storage** is used when all state information for the container can be discarded when the container restarts. This is commonly used when environments are created for demonstration, or when developing with stand-alone queue managers.
- **Persistent storage** is the common configuration for IBM MQ and ensures that if the container is restarted, the existing configuration, logs and persistent messages are available in the restarted container.

The IBM MQ Operator provides the capability to customize the storage characteristics which can differ considerably depending on the environment, and the desired storage mode.

Ephemeral storage

IBM MQ is a stateful application and persists this state to storage for recovery in the event of a restart. If using ephemeral storage, all state information for the queue manager is lost on restart. This includes:

- All messages
- All queue manager to queue manager communication state (channel message sequence numbers)
- The MQ Cluster identity of the queue manager
- All transaction state
- All queue manager configuration
- All local diagnostic data

For this reason you need to consider if ephemeral storage is a suitable approach for a production, test or development scenario. For example, where all messages are known to be non-persistent and the queue manager is not a member of an MQ Cluster. As well as disposing of all messaging state at restart, the configuration of the queue manager is also discarded. To enable a completely ephemeral container the IBM MQ configuration must be added to the container image itself (for more information, see "Building an image with custom MQSC and INI files, using the Red Hat OpenShift CLI" on page 185). If this is not completed, then IBM MQ will need to be configured each time the container restarts.

OpenShift CP4I For example, to configure IBM MQ with ephemeral storage the storage type of the QueueManager should include the following:

```
queueManager:
  storage:
    queueManager:
      type: ephemeral
```

Persistent storage

OpenShift CP4I

IBM MQ normally runs with persistent storage to assure the queue manager retains its persistent messages and configuration after a restart. This is the default behavior. Because there are various storage providers, each supporting different capabilities, this often means that customization of the configuration is required. The example below outlines the common fields that customize the IBM MQ storage configuration in the v1beta1 API:

- **spec.queueManager.availability** controls the availability mode. If you are using SingleInstance or NativeHA, you only require ReadWriteOnce storage. For multiInstance you require a storage class that supports ReadWriteMany with the correct file locking characteristics. IBM MQ provides a support statement and a testing statement. The availability mode also influences the persistent volume layout. For more information, see "Planning high availability for IBM MQ in containers" on page 20.
- **spec.queueManager.storage** controls the individual storage settings. A queue manager can be configured to use between one and four persistent volumes.

The following example shows a snippet of a simple configuration using a single-instance queue manager:

```
spec:
  queueManager:
    storage:
      queueManager:
        type: persistent-claim
```

Storage capacity

OpenShift CP4I

When you use the IBM MQ Operator, try to ensure that you request volumes that are large enough for your ongoing needs.

V 10.0.0 One example of when you should increase volume size is if you set the **enabled** property of the Scratch or Tmp volumes to false (the default value is true). In this configuration, Scratch or Tmp data is written to the queue manager data volume. You should therefore increase the ".spec.queueManager.storage.queueManager" on page 300 **size** property to account for the expected use of the Scratch and Tmp volumes. The default value for the **size** property is 2Gi. The default values of the **sizeLimit** properties for Scratch and Tmp volumes are 100M and 2Gi respectively; account for these sizes when Scratch and Tmp data is written to the queue manager data volume.

If you need to increase the storage capacity of one or more volumes, these volumes can be expanded if your storage class supports volume expansion. Volumes can be expanded either by an online or offline procedure. An offline procedure requires QueueManager pods to be restarted, whereas an online procedure does not. To determine if your storage class supports volume expansion, and which procedure the volume expansion follows, refer to your storage provider documentation. You should consider this information when selecting a storage class. For a guide to volume expansion, see "Expanding persistent volumes" on page 196.

Encryption

OpenShift CP4I

IBM MQ does not actively encrypt data at rest. Therefore you should use passively encrypted storage, or IBM MQ Advanced Message Security, or both, to encrypt your messages. On IBM Cloud® both block and file storage are available with passive encryption at rest.

Storage considerations for cloud platforms

Some cloud platforms have specific storage requirements for deploying IBM MQ queue managers with the IBM MQ Operator.

IBM Cloud

The `ibmc-file-gold-gid` storage class on IBM Cloud requires you to set the following `supplementalGroups` value in the `spec.securityContext` field of your `QueueManager` YAML:

```
spec:
  queueManager:
    storage:
      queueManager:
         class: ibmc-file-gold-gid
  securityContext:
    supplementalGroups: [65534] # Change to 99 for clusters with RHEL7 or earlier worker nodes
```

Amazon EKS

The `aws-ebs` storage class on Amazon EKS requires you to set a `fsGroup` value of your choice in the `spec.securityContext` field of your `QueueManager` YAML. For example, the following YAML sets the `fsGroup` to `1001`:

```
spec:
  queueManager:
    storage:
      defaultClass: aws-ebs
  securityContext:
    fsGroup: 1001
```

Planning high availability for IBM MQ in containers

There are three choices for high availability with IBM MQ Operator: **Native HA queue manager** (which has an active replica and two standby replicas), **Multi-instance queue manager** (which is an active-standby pair, using a shared, networked file system), or **Single resilient queue manager** (which offers a simple approach for HA using networked storage). The latter two rely on the file system to ensure the availability of the recoverable data, however Native HA does not. Therefore, when not using Native HA, the availability of the file system is critical to queue manager availability. Where data recovery is important the file system should ensure redundancy through replication.

You should consider separately **message** and **service** availability. With IBM MQ for Multiplatforms, a message is stored on exactly one queue manager. So if that queue manager becomes unavailable, you temporarily lose access to the messages it holds. To achieve high `message` availability, you need to be able to recover a queue manager as quickly as possible. You can achieve `service` availability by having multiple instances of queues for client applications to use, for example by using an IBM MQ uniform cluster.

A queue manager can be thought of in two parts: the data stored on disk, and the running processes that allow access to the data. Any queue manager can be moved to a different Kubernetes Node, as long as it keeps the same data (provided by Kubernetes Persistent Volumes) and is still addressable across the network by client applications. In Kubernetes, a Service is used to provide a consistent network identity.

IBM MQ relies on the availability of the data on the persistent volumes. Therefore, the availability of the storage providing the persistent volumes is critical to queue manager availability, because IBM MQ cannot be more available than the storage it is using. If you want to tolerate an outage of an entire availability zone, you need to use a volume provider that replicates disk writes to another zone.

Native HA queue manager

Native HA queue managers involve an **active** and two **replica** Kubernetes Pods, which run as part of a Kubernetes StatefulSet with exactly three replicas each with their own set of Kubernetes Persistent Volumes. The IBM MQ requirements for shared file systems also apply when using a native HA queue manager (except for lease-based locking), but you do not need to use a shared file system. You can use block storage, with a suitable file system on top. For example, *xfs* or *ext4*. The recovery times for a native HA queue manager are controlled by the following factors:

1. How long the replica instances take to detect that the active instance has failed. This is configurable.
2. How long it takes for the Kubernetes Pod readiness probe to detect that the ready container has changed and redirect network traffic. This is configurable.
3. How long it takes IBM MQ clients to reconnect.

For more information, see “Native HA in containers” on page 22.

Native HA CRR (Cross-Region Replication)

You can implement a disaster recovery solution that comprises two Native HA groups located in separate regions. One Native HA group has the **live** role, while the other group has the **recovery** role. When a planned or unplanned outage occurs at the region where the live group is, user action can switch the role of the group in the remote region from recovery to live. This group then takes over the work of the original queue manager instance, accepting application connections and so on.

For more information, see “Native HA Cross-Region Replication” on page 26.

Multi-instance queue manager

Multi-instance queue managers involve an **active** and a **standby** Kubernetes Pod, which run as part of a Kubernetes Stateful Set with exactly two replicas and a set of Kubernetes Persistent Volumes. The queue manager transaction logs and data are held on two persistent volumes, using a shared file system.

Multi-instance queue managers require both the **active** and the **standby** Pods to have concurrent access to the persistent volume. To configure this, you use Kubernetes Persistent Volumes with **`access mode`** set to `ReadWriteMany`. The volumes must also meet the IBM MQ requirements for shared file systems, because IBM MQ relies on the automatic release of file locks to instigate a queue manager failover. IBM MQ produces a list of tested file systems.

The recovery times for a multi-instance queue manager are controlled by the following factors:

1. How long it takes after a failure occurs for the shared file system to release the locks originally taken by the active instance.
2. How long it takes for the standby instance to acquire the locks and then start.
3. How long it takes for the Kubernetes Pod readiness probe to detect that the ready container has changed and redirect network traffic. This is configurable.
4. How long it takes for IBM MQ clients to reconnect.

Single resilient queue manager

A single resilient queue manager is a single instance of a queue manager running in a single Kubernetes Pod, where Kubernetes monitors the queue manager and replaces the Pod as necessary.

The IBM MQ requirements for shared file systems also apply when using a single resilient queue manager (except for lease-based locking), but you do not need to use a shared file system. You can use block storage, with a suitable file system on top. For example, *xfs* or *ext4*.

The recovery times for a single resilient queue manager are controlled by the following factors:

1. How long it takes for the liveness probe to run, and how many failures it tolerates. This is configurable.
2. How long the Kubernetes Scheduler takes to re-schedule the failed Pod to a new Node.
3. How long it takes to download the container image to the new Node. If you use an **imagePullPolicy** value of `IfNotPresent`, then the image might already be available on that Node.
4. How long it takes for the new queue manager instance to start.
5. How long it takes for the Kubernetes Pod readiness probe to detect that the container is ready. This is configurable.
6. How long it takes for IBM MQ clients to reconnect.

Important:

Although the single resilient queue manager pattern offers some benefits, you need to understand whether you can reach your availability goals with the limitations around Node failures.

In Kubernetes, a failing Pod is typically recovered quickly; but the failure of an entire Node is handled differently. When using a stateful workload like IBM MQ with a Kubernetes StatefulSet, if a Kubernetes Master Node loses contact with a worker node, it cannot determine if the node has failed or if it has simply lost network connectivity. Therefore Kubernetes takes **no action** in this case until one of the following events occurs:

1. The node recovers to a state where the Kubernetes Master Node can communicate with it.
2. An administrative action is taken to explicitly delete the Pod on the Kubernetes Master Node. This does not necessarily stop the Pod from running, but just deletes it from the Kubernetes store. This administrative action must therefore be taken very carefully.

Note: Changing the details of the StatefulSet of an IBM MQ queue manager, including the number of replicas, is not supported when the queue manager is created through the IBM MQ Operator.

Related concepts

Configuring high availability

Related tasks

“Configuring high availability for queue managers using the IBM MQ Operator” on page 118

You can configure high availability for queue managers by using either the Native HA solution or the multi-instance queue manager solution.

CP4I MQ Adv. Kubernetes Native HA in containers

Native HA is a native (built-in) high availability solution for IBM MQ that is suitable for use with cloud block storage.

A Native HA configuration provides a highly available queue manager where the recoverable MQ data (for example, the messages) are replicated across multiple sets of storage, preventing loss from storage failures. The queue manager consists of multiple running instances, one is the leader, the others are ready to quickly take over in the event of a failure, maximizing access to the queue manager and its messages.

A Native HA configuration consists of three Kubernetes pods, each with an instance of the queue manager. One instance is the active queue manager, processing messages and writing to its recovery log. Whenever the recovery log is written, the active queue manager sends the data to the other two instances, known as replicas. Each replica writes to its own recovery log, acknowledges the data, and then updates its own queue data from the replicated recovery log. If the pod running the active queue manager fails, one of the replica instances of the queue manager takes over the active role and has current data to operate with.

The log type is known as a 'replicated log'. A replicated log is essentially a linear log, with automatic log management and automatic media images enabled. See Types of logging. You use the same techniques for managing the replicated log that you use for managing a linear log.

A Kubernetes Service is used to route TCP/IP client connections to the current active instance, which is identified as being the only pod which is ready for network traffic. This happens without the need for the client application to be aware of the different instances.

Three pods are used to greatly reduce the possibility of a split-brain situation arising. In a two-pod high availability system split-brain could occur when the connectivity between the two pods breaks. With no connectivity, both pods could run the queue manager at the same time, accumulating different data. When connection is restored, there would be two different versions of the data (a 'split-brain'), and manual intervention is required to decide which data set to keep, and which to discard.

Native HA uses a three pod system with quorum to avoid the split-brain situation. Pods that can communicate with at least one of the other pods form a quorum. A queue manager can only become the active instance on a pod that has quorum. The queue manager cannot become active on a pod that is not connected to at least one other pod, so there can never be two active instances at the same time:

- If a single pod fails, the queue manager on one of the other two pods can take over. If two pods fail, the queue manager cannot become the active instance on the remaining pod because the pod does not have quorum (the remaining pod cannot tell whether the other two pods have failed, or they are still running and it has lost connectivity).
- If a single pod loses connectivity, the queue manager cannot become active on this pod because the pod does not have quorum. The queue manager on one of the remaining two pods can take over, which do have quorum. If all pods lose connectivity, the queue manager is unable to become active on any of the pods, because none of the pods have quorum.

If an active pod fails, and subsequently recovers, it can rejoin the group in a replica role.

For performance and reliability, RWO (ReadWriteOnce) persistent storage is recommended for use with a Native HA configuration. RWO volumes from any storage provider are supported if they meet the following conditions:

- Obtained from a block storage provider.
- Formatted as ext4 or XFS (which ensures POSIX compliance).
- Supports dynamic volume provisioning and "volumeBindingMode: WaitForFirstConsumer".

The following providers are explicitly prohibited:

- NFS
- GlusterFS
- Other non-block providers.

Note: V 10.0.0 If you configure Native HA cross-region replication, you will require at least twice the amount of disk space than for Native HA. This is due to the requirement for a log backup directory during rebasing (see “Native HA Cross-Region Replication” on page 26)

The following figure shows a typical deployment with three instances of a queue manager deployed in three containers.

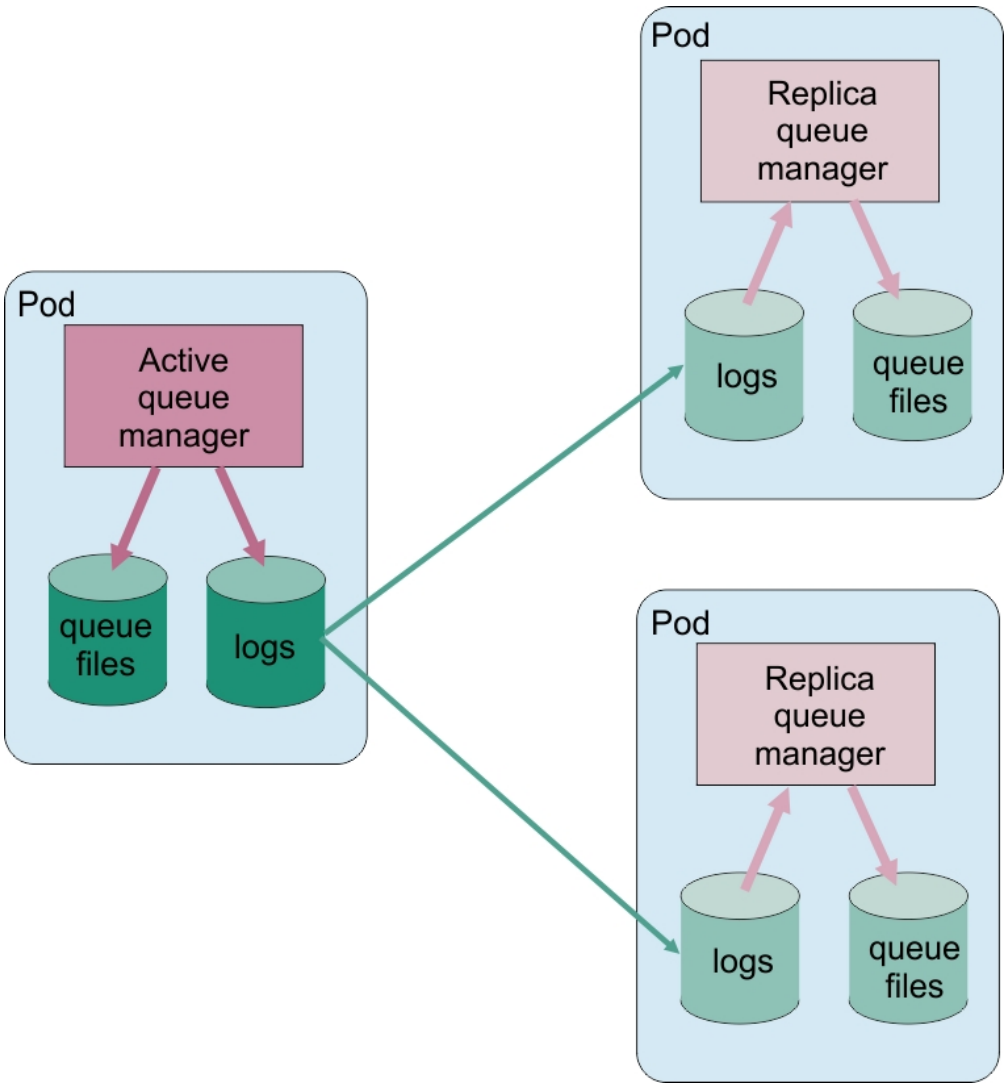


Figure 1. Example of Native HA configuration

MQ Adv. Kubernetes Considerations for performing your own rolling update of a Native HA queue manager

Any update to the IBM MQ version or Pod specification for a Native HA queue manager, will require you to perform a rolling update of the queue manager instances. The IBM MQ Operator handles this for you automatically, but if you are building your own deployment code, then there are some important considerations.

Note: The sample Helm chart includes a shell script to perform a rolling update, but the script is **not** suitable for production use, as it does not address the considerations in this topic.

Kubernetes In Kubernetes, `StatefulSet` resources are used to manage ordered start-up and rolling updates. Part of the start-up procedure is to start each Pod individually, wait for it to become ready, and then move onto the next Pod. This won't work for Native HA, as all Pods need to be started so that they can run a leader election. Therefore the `.spec.podManagementPolicy` field on the `StatefulSet` needs to be set to `Parallel`. This also means that all Pods will be updated in parallel too, which is particularly undesirable. For this reason, the `StatefulSet` should also use the `OnDelete` update strategy.

Inability to use the `StatefulSet` rolling update code drives a need for custom rolling update code, which should consider the following:

- General rolling update procedure
- Minimizing down time by updating Pods in the best order
- Handling changes in cluster state
- Handling errors
- Handling timing problems

General rolling update procedure

The rolling update code should wait for each instance to show a status of `REPLICA` from `dspmq`. This means that the instance has performed some level of start up (for example, the container is started, and MQ processes are running), but it has not necessarily managed to talk to the other instances yet. For example: Pod A gets restarted, and as soon as it's in `REPLICA` state, Pod B gets restarted. Once Pod B starts with the new configuration, it should be able to talk to Pod A, and can form quorum, and either A or B will become the new active instance.

As part of this, it is useful to have a delay after each Pod has reached the `REPLICA` state, to allow for it to connect to its peers and establish quorum.

Minimizing down time by updating Pods in the best order

The rolling update code should delete Pods one at a time, starting with Pods which are in a known error state, followed by any Pods that have not successfully started. The active queue manager Pod should generally be updated last.

It is also important to pause the deletion of Pods if the last update resulted in a Pod going into a known error state. This prevents the roll-out of a broken update across all Pods. For example, this can happen if the Pod is updated to use a new container image which isn't accessible (or contains a typo).

Handling changes in cluster state

The rolling update code needs to react appropriately to real-time changes in cluster state. For example, one of the queue manager's Pods might be evicted due to a Node reboot or due to Node pressure. It's possible that an evicted Pod might not be immediately re-scheduled if the cluster is busy. In this case, the rolling update code would need to wait appropriately before restarting any other Pods.

Handling errors

The rolling update code needs to be robust to failures when calling the Kubernetes API and other unexpected cluster behaviour.

In addition, the rolling update code itself needs to be tolerant to being restarted. A rolling update can be long-running, and the code may need to be restarted.

Handling timing problems

The rolling update code needs to check the update revisions of the Pod, so that it can ensure the Pod has restarted. This avoids timing issues where a Pod may indicate that it is "Started", but it has in-fact not yet terminated.

Related concepts

“Choosing how you want to use IBM MQ in containers” on page 9
There are multiple options for using IBM MQ in containers: you can choose to use the IBM MQ Operator, or you can deploy IBM MQ using your own deployment code.

OpenShift CP4I Kubernetes Disaster recovery for IBM MQ in containers

You need to consider what kind of disaster you are preparing for. In cloud environments, the use of availability zones provides a certain level of toleration for disasters, and are much easier to use.

If you have an odd number of data centers (for quorum) and a low latency network link, then you could potentially run a single Red Hat OpenShift Container Platform or Kubernetes cluster with multiple availability zones, each in a separate physical location. This topic discusses considerations for disaster recovery where these criteria cannot be met: that is to say either an even number of data centers, or a high latency network link.

For disaster recovery, you need to consider the following:

- Replication of IBM MQ data (held in one or more `PersistentVolume` resources) to the disaster recovery location
- Re-creating the queue manager using the replicated data
- The queue manager network ID that is visible to IBM MQ client applications and other queue managers. This ID could be a DNS entry, for example.

Persistent data needs to be replicated, either synchronously or asynchronously, to the disaster recovery site. This is usually specific to the storage provider, but can also be done using a `VolumeSnapshot`. See CSI volume snapshots for more information on volume snapshots.

When recovering from a disaster, you will need to re-create the queue manager instance on the new Kubernetes cluster, using the replicated data. If you are using the IBM MQ Operator, you will need the `QueueManager` YAML, as well as the YAML for other supporting resources like a `ConfigMap` or `Secret`.

Related information

ha_for_ctr.dita

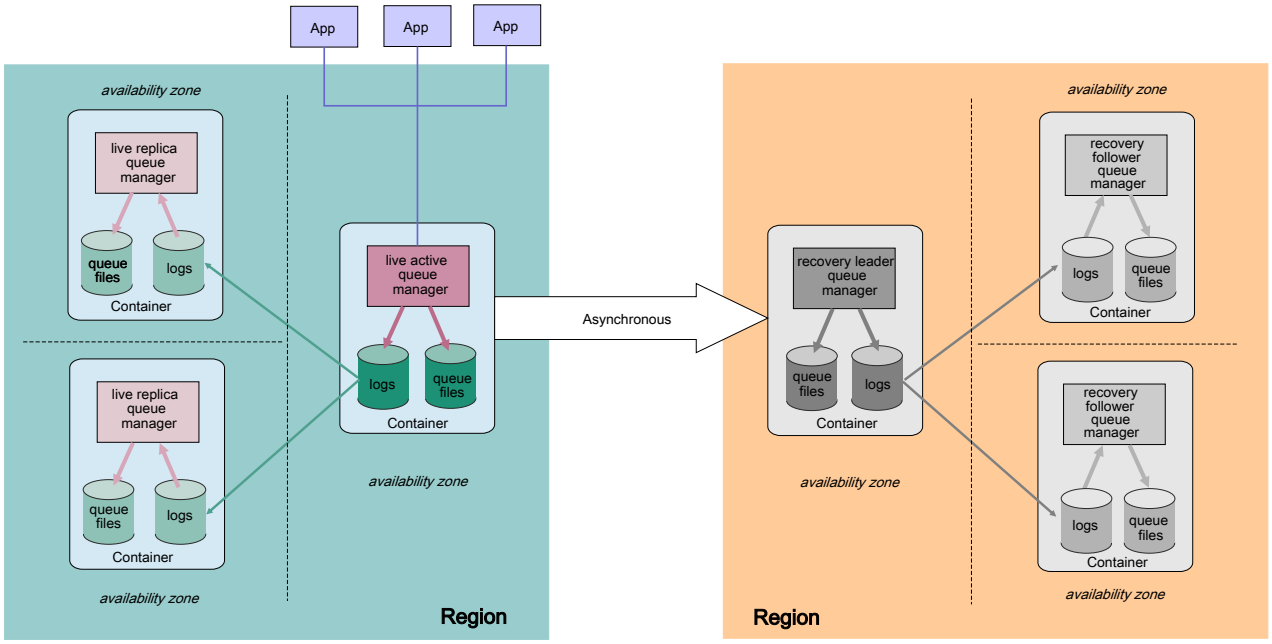
V 10.0.0 Native HA Cross-Region Replication

You can implement a Native HA Cross-Region Replication (CRR) configuration based on the Native HA solution. The CRR configuration provides disaster recovery.

Note: Native HA In-Region Replication (IRR) is not currently available in containers.

See “Native HA in containers” on page 22 for an overview of the Native HA solution.

Native HA uses a group of three queue manager instances to maintain a replicated copy of log data. The group automatically elects an active instance by comparing and choosing the best copy of data. CRR extends log replication to include a potentially geographically distant group (region) that is also highly available. When there is a planned or unplanned outage at the main site, the group at the recovery site can take over the work. You initially define which role a group plays, and then switch the role of a group manually as required to complete a planned switchover, or when an unplanned outage occurs. The log data is replicated to the recovery group asynchronously, so there might be some data loss when a switchover following an unplanned outage occurs.



Requested roles and Effective roles

A Requested role is the GroupRole configuration setting for a Native HA instance that is specified in the `NativeHALocalInstance` stanza of its `qm.ini`. The setting requests that an instance co-operate with the other instances in its group to fulfill a role. Once a majority of instances share the same Requested role, an election can take place to choose a leader of the group. On winning an election, the Requested role is compared with the Last role the instance used to derive an Effective role. See NativeHAInstance stanza of the qm.ini file and NativeHALocalInstance stanza of the qm.ini file.

Live role

A Live group is a regular Native HA queue manager consisting of three instances. One instance is elected as the active one and this instance accepts new work from applications. An existing Native HA queue manager can be migrated to a Native HA CRR configuration.

If a Requested role is not explicitly specified in the `qm.ini`, the default is considered to be Live. An elected leader of a Live group is also known as the active queue manager, this instance can accept application and tooling connections to perform new messaging work and manage queue manager objects.

Recovery role

A Recovery group also consists of three instances and one of these instances is elected as leader. The leader acts only as a proxy, accepting work sent from a paired Live group. No application connections are permitted to the Recovery group, although some control commands are allowed so that operations such as ending the queue manager instance can be completed.

The `NativeHARecoveryGroup` stanza in the `qm.ini` configuration file of the Live group identifies the network address for the Recovery group.

Pending role transitions

When performing a planned switchover where the current Live group becomes the new Recovery group (and the current Recovery group becomes the new Live group) there needs to be co-ordination between the groups. This co-ordination ensures that the logs are identical and achieves an RPO (recovery point objective) of zero.

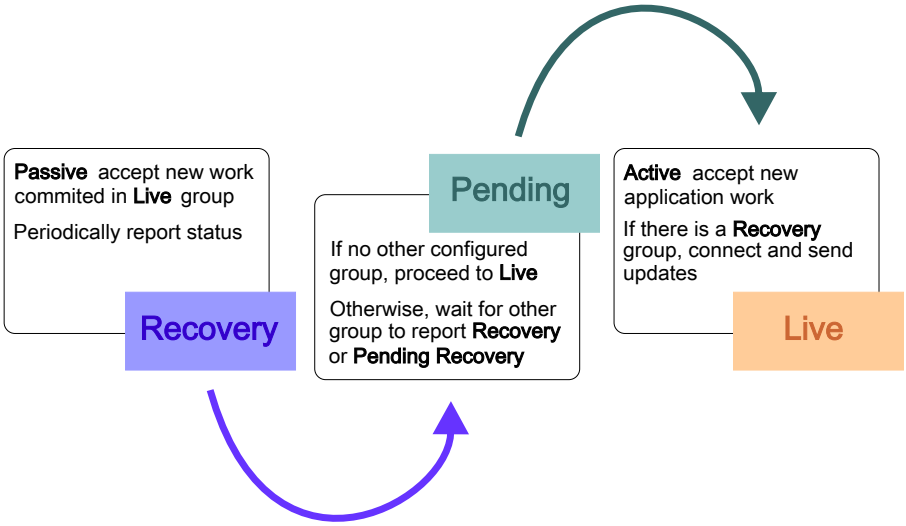
Where two groups are coordinating the switch between Recovery and Live, an effective Pending role is temporarily adopted by both groups.

Recovery role to Live role (Pending Live)

A group that has previously operated in a Recovery role that intends on changing to a Live role makes the decision on when to use the role based on whether there is another group configured.

If there is no other configured group, it transitions to the Live role immediately.

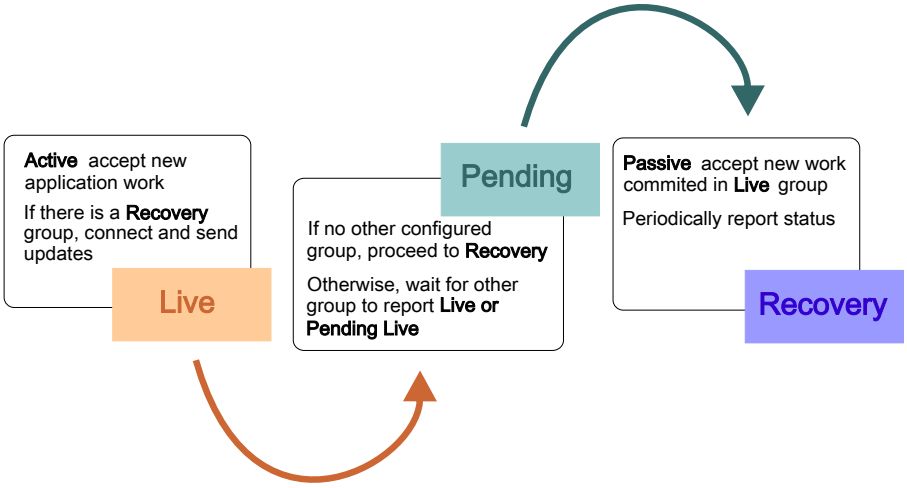
If another group is configured and enabled, the transition to a Live role is blocked until the other group adopts a Recovery or Pending Recovery role, and until both groups are in-sync.



Live role to Recovery role (Pending Recovery)

A group that had previously operated in a Live role that intends on changing to a Recovery role will make the decision on when to use the Recovery role, based on whether there is another group configured.

If another group is configured and enabled, the transition to a Recovery role is blocked until the other group adopts a Live or Pending Live role and until both groups are in-sync.



Replication and rebasing

While the two groups in a Native HA CRR configuration are connected, log updates are passed to the leader of the recovery group and passed on to the other recovery instances. This is known as **replication**.

If the network connection between the two groups is lost, it might happen that the recovery group could not catch up by using replicated logs (for example, because the live group has reused log extents that would be needed for the catch up). In this case, the recovery leader performs a **rebase**. A rebase is the process of discarding the received log data and rebuilding the queue manager from a complete set of log

data sent from the live group. Once the recovery leader has completed the rebase, the logs are replicated to the recovery instances.

If a problem is encountered during a rebase operation, then it is possible that the queue manager could be unable to start on any of the members of the recovery group. To avoid this possibility, a backup of the log is taken before the rebase operation commences. The queue manager can then be restarted using the backup log data if the rebase fails. The backup log is deleted after the rebase has succeeded.

Because of the requirement for a backup log, a Native HA CRR configuration requires at least twice the storage space of a Native HA configuration.

Planning security for IBM MQ in containers

Security considerations when planning your IBM MQ in containers configuration.

Procedure

- "User authentication and authorization for IBM MQ in containers" on page 29
  - "Security constraints on the use of operating system users in containers" on page 30
- "Considerations for restricting network traffic to IBM MQ in containers" on page 30

User authentication and authorization for IBM MQ in containers

IBM MQ in containers can be configured to authenticate users through LDAP, Mutual TLS, or a custom MQ plugin.

Note that the IBM MQ Operator does not allow the use of operating system users and groups within the container image. For more information, see "Security constraints on the use of operating system users in containers" on page 30.

LDAP

For information about configuring IBM MQ to use an LDAP user repository, see Connection authentication: User repositories and LDAP authorization.

Mutual TLS

If you configure incoming connections to a queue manager to require a TLS certificate (mutual TLS), you can map the distinguished name of the certificate to a user name. You need to do two things:

- Configure a channel authentication record to create the mapping to a user name, using SSLPEER. For more information, see Mapping an SSL or TLS Distinguished Name to an MCAUSER user ID.
- Configure the queue manager to allow you to define authority records for a user name that is not known to the system. For more information, see Service stanza of the qm.ini file.

JSON Web Tokens

For information about configuring IBM MQ to use JSON Web Tokens (JWT), see Working with authentication tokens.

Custom MQ plugin

This is an advanced technique, and requires a lot more work. For more information, see Using a custom authorization service.

Related tasks

"Example: Configuring a queue manager with mutual TLS authentication" on page 109

This example deploys a queue manager by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Security constraints on the use of operating system users in containers

Using operating system users in containers is not recommended, and is prohibited with the IBM MQ Operator.

In a multi-tenant containerized environment, security constraints are typically put in place to prevent potential security issues, for example:

- **Preventing use of the "root" user inside a container**
- **Forcing the use of a random UID**. For example, in Red Hat OpenShift Container Platform the default `SecurityContextConstraints` (called `restricted`) uses a randomized user ID for each container.
- **Preventing the use of privilege escalation**. IBM MQ on Linux uses privilege escalation to check the passwords of users — it uses a "setuid" program so as to become the "root" user to do this.

To ensure compliance with these security measures, the IBM MQ Operator does not allow the use of IDs that are defined on the operating system libraries inside a container. There is no `mqm` user ID or group defined in the container.

Considerations for restricting network traffic to IBM MQ in containers

You can define network policies to restrict traffic to pods in your cluster in OpenShift Container Platform and Kubernetes, including in Amazon EKS.. This topic describes some considerations for how network policies can apply to IBM MQ.

For network ingress to a queue manager, there are several ports to consider:

- Port 1414 for queue manager traffic
- Port 9414 for native HA
- Port 9157 for metrics
- Port 9443 for the web console and REST APIs

Network egress is more complex. Examples of network egress which you might want to consider:

- DNS — if you have channels or other configuration which use DNS names
- Other queue managers
- Online Certificate Status Protocol (OCSP) and Certificate Revocation Lists (CRLs) - determined by your certificate provider.
- Authentication providers:
  - LDAP
  - Open ID Connect or other configured login provider for the IBM MQ web server. This includes the IBM Cloud Pak Keycloak.
- Tracing providers:
  - IBM Instana

  Note: For earlier IBM MQ versions, the IBM Cloud Pak for Integration Operations Dashboard was also available as a tracing provider. However, the Operations Dashboard was removed at IBM MQ 9.3.3 CD and IBM MQ 9.4.0 LTS.

Example ingress NetworkPolicy

The following is an example network policy to control ingress for a queue manager called "myqm", for deployments on Red Hat OpenShift or Amazon EKS.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: myqm
```

```
spec:
  podSelector:
    matchLabels:
      app.kubernetes.io/instance: myqm
      app.kubernetes.io/name: ibm-mq
  ingress:
    # Allow access to queue manager listener from anywhere
    - ports:
        - protocol: TCP
          port: 1414
    # Allow access to Native HA port from other instances of the same queue manager
    - from:
        - podSelector:
            matchLabels:
              app.kubernetes.io/instance: myqm
              app.kubernetes.io/name: ibm-mq
      ports:
        - protocol: TCP
          port: 9414
```

Red Hat OpenShift users only: You can add additional configuration to the previous example network policy:

```
spec:
  ingress:
    # Allow access to metrics from monitoring project
    - from:
        - namespaceSelector:
            matchLabels:
              network.openshift.io/policy-group: monitoring
      ports:
        - protocol: TCP
          port: 9157
    # Allow access to web server via Route
    - from:
        - namespaceSelector:
            matchLabels:
              network.openshift.io/policy-group: ingress
      ports:
        - protocol: TCP
          port: 9443
```

FIPS compliance for IBM MQ in containers

At start up, IBM MQ in containers detects whether the operating system on which the container is starting is FIPS compliant, and (if so) configures FIPS support automatically. Requirements and limitations are noted here.

Federal Information Processing Standards

The US government produces technical advice on IT systems and security, including data encryption. The National Institute for Standards and Technology (NIST) is a government body concerned with IT systems and security. NIST produces recommendations and standards, including the Federal Information Processing Standards (FIPS).

A significant FIPS standard is FIPS 140-2 , which requires the use of strong cryptographic algorithms. FIPS 140-2 also specifies requirements for hashing algorithms to be used to protect packets against modification in transit.

IBM MQ provides FIPS 140-2 support if it has been configured to do so.

Notes:

- ALW V 10.0.0 V 10.0.0 On AIX®, Linux, and Windows, IBM MQ provides FIPS 140-3 compliance through the GSKit 9 IBM Crypto for C (ICC) cryptographic module. The NIST certification associated with the FIPS 140-3 module can be viewed at https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755.
- V 10.0.0 V 10.0.0 FIPS support is currently not available for Linux s390x. Customers on this platform who require FIPS support should remain at a previous version of IBM MQ. Support for FIPS 140-3 will be enabled in

a future FixPack once the IBM Crypto for C (ICC) cryptographic module has received its certification on this platform.

- V 10.0.0 The FIPS 140-3 cryptographic module within IBM Semeru Runtime was approved by NIST in August 2024. IBM MQ 10.0.0 adds support for the handling of IBM MQ classes for JMS and IBM MQ classes for Java client connections using TLS for FIPS 140-3 in Java 8 and IBM Semeru Runtime 11+. The NIST certification associated with the FIPS 140-3 module can be viewed at https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755.
- For IBM MQ in Containers, the IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance for IBM MQ in Containers is currently pending.

  V 10.0.0 If FIPS is enabled, IBM MQ in Container control processes use a FIPS 140-3 Certified OpenSSL Module. Details of the NIST certification can be viewed at: https://access.redhat.com/compliance/fips. IBM MQ queue managers running in container images have the same FIPS certification level as the base image platform version of IBM MQ.

Requirements

For requirements related to cluster setup and other considerations, see FIPS Wall: Current IBM approach to FIPS compliance.

IBM MQ in containers can run in FIPS 140-2 compliance mode. During start up, IBM MQ in containers detects whether the host operating system on which the container is starting is FIPS compliant. If the host operating system is FIPS compliant, and private keys and certificates have been supplied, the IBM MQ container configures the queue manager, the IBM MQ web server, and data transfer between the nodes in a Native High Availability deployment, to run in FIPS compliance mode.

When using IBM MQ Operator to deploy queue managers on the Red Hat OpenShift Container Platform, the operator creates a route with a termination type of **Passthrough**. This means that the traffic is sent straight to the destination without the router providing TLS termination. The IBM MQ queue manager and IBM MQ web server are the destinations in this case, and they already provide FIPS compliant secure communication.

To ensure FIPS compliance for deployments of the IBM MQ Operator in containers on other platforms, you must configure your own network ingress with the relevant termination type.

Key requirements:

1. A private key and certificates, provided in a secret to the queue manager and web server, that allow external clients to connect securely to the queue manager and web server.
2. A private key and certificates for data transfer between different nodes in a Native High Availability configuration.

Limitations

For a FIPS compliant deployment of IBM MQ in containers, consider the following:

- IBM MQ in containers provides an endpoint for collection of metrics through HTTPS. You can serve metrics through HTTPS or HTTP. To be FIPS compliant, you can choose from the following options:
  - You can serve metrics through HTTPS. For more information, see “Monitoring when using the IBM MQ Operator” on page 244.
  - You can turn off the metrics endpoint to make the rest of IBM MQ compliant if you do not want to use HTTPS for metrics.
- IBM MQ in containers allows custom image overrides. That is, you can build custom images using the IBM MQ container image as the base image. FIPS compliance might not apply for such customized images.
- For message tracking using IBM Instana, the communication between IBM MQ and IBM Instana is HTTP or HTTPS, with no FIPS compliance.

- IBM MQ Operator access to IBM identity and access management (IAM)/Zen services is not FIPS compliant.

How FIPS compliance is detected and FIPS support is configured automatically

If the operating system on which the container is starting is FIPS compliant, FIPS support is configured automatically.

Notes:

- ALW V 10.0.0 V 10.0.0 On AIX, Linux, and Windows, IBM MQ provides FIPS 140-3 compliance through the GSKit 9 IBM Crypto for C (ICC) cryptographic module. The NIST certification associated with the FIPS 140-3 module can be viewed at https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755.
- V 10.0.0 V 10.0.0 FIPS support is currently not available for Linux s390x. Customers on this platform who require FIPS support should remain at a previous version of IBM MQ. Support for FIPS 140-3 will be enabled in a future FixPack once the IBM Crypto for C (ICC) cryptographic module has received its certification on this platform.
- V 10.0.0 The FIPS 140-3 cryptographic module within IBM Semeru Runtime was approved by NIST in August 2024. IBM MQ 10.0.0 adds support for the handling of IBM MQ classes for JMS and IBM MQ classes for Java client connections using TLS for FIPS 140-3 in Java 8 and IBM Semeru Runtime 11+. The NIST certification associated with the FIPS 140-3 module can be viewed at https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755.
- For IBM MQ in Containers, the IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance for IBM MQ in Containers is currently pending.

  V 10.0.0 If FIPS is enabled, IBM MQ in Container control processes use a FIPS 140-3 Certified OpenSSL Module. Details of the NIST certification can be viewed at: https://access.redhat.com/compliance/fips. IBM MQ queue managers running in container images have the same FIPS certification level as the base image platform version of IBM MQ.

During start up, IBM MQ in containers detects whether the operating system on which the container is starting is FIPS compliant. If so, then the following actions are taken automatically:

Important: Linux On Linux s390x, the **SSLFIPS** attribute is always set to NO. If you require FIPS support on this platform, you should remain at a previous version of IBM MQ. Support for FIPS 140-3 will be enabled in a future FixPack once the IBM Crypto for C (ICC) cryptographic module has received its certification on this platform.

Queue manager

If the host operating system is FIPS compliant, and the private key and certificates are supplied, the queue manager attribute **SSLFIPS** is set to YES. Otherwise, the **SSLFIPS** attribute is set to NO.

IBM MQ web server

The IBM MQ web server provides an HTTP/HTTPS interface for administering IBM MQ. If the host operating system is FIPS compliant, the JVM options are updated to make the web server use FIPS compliant cryptography. To be able to use FIPS, the private key and certificates must be supplied during container start.

Native HA

Security of the data replicated between nodes is controlled by the **NativeHALocalInstance** stanza of the `qm.ini` file. For example:

```
NativeHALocalInstance:
    KeyRepository=/run/runmqserver/ha/tls/key.kdb
    CertificateLabel=NHAQM
    CipherSpec=ECDHE_RSA_AES_256_GCM_SHA384
```

If FIPS is enabled, the **SSLFipsRequired** attribute is added to the stanza, with the value set to Yes:

```
NativeHALocalInstance:
    KeyRepository=/run/runmqserver/ha/tls/key.kdb
    CertificateLabel=NHAQM
```

```
    CipherSpec=ECDHE_RSA_AES_256_GCM_SHA384
    SSLFipsRequired=Yes
```

If the container is running in an environment without FIPS support, then the queue manager, IBM MQ web server, and Native HA components do not have their FIPS support automatically enabled. To explicitly enable FIPS support in the IBM MQ components for these environments, set the `MQ_ENABLE_FIPS` environment variable to true in the queue manager YAML. The following YAML snippet describes the usage of the `MQ_ENABLE_FIPS` environment variable:

Important: Linux On Linux s390x, all components will not run in FIPS mode, regardless of the value of the *MQ_ENABLE_FIPS* environment variable. If you require FIPS support on this platform, you should remain at a previous version of IBM MQ. Support for FIPS 140-3 will be enabled in a future FixPack once the IBM Crypto for C (ICC) cryptographic module has received its certification on this platform.

```
template:
    pod:
      containers:
        - env:
            - name: MQ_ENABLE_FIPS
              value: "true"
          name: qmgr
```

Overriding automatic FIPS mode for IBM MQ in containers

Use environment variable *MQ_ENABLE_FIPS* to explicitly enable or disable FIPS mode for the IBM MQ components in the container.

Before you begin

Notes:

- ALW V 10.0.0 V 10.0.0 On AIX, Linux, and Windows, IBM MQ provides FIPS 140-3 compliance through the GSKit 9 IBM Crypto for C (ICC) cryptographic module. The NIST certification associated with the FIPS 140-3 module can be viewed at https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755.
- V 10.0.0 V 10.0.0 FIPS support is currently not available for Linux s390x. Customers on this platform who require FIPS support should remain at a previous version of IBM MQ. Support for FIPS 140-3 will be enabled in a future FixPack once the IBM Crypto for C (ICC) cryptographic module has received its certification on this platform.
- V 10.0.0 The FIPS 140-3 cryptographic module within IBM Semeru Runtime was approved by NIST in August 2024. IBM MQ 10.0.0 adds support for the handling of IBM MQ classes for JMS and IBM MQ classes for Java client connections using TLS for FIPS 140-3 in Java 8 and IBM Semeru Runtime 11+. The NIST certification associated with the FIPS 140-3 module can be viewed at https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4755.
- For IBM MQ in Containers, the IBM MQ Operator 3.2.0 and queue manager container image 9.4.0.0 onwards are based on UBI 9. FIPS 140-3 compliance for IBM MQ in Containers is currently pending.

  V 10.0.0 If FIPS is enabled, IBM MQ in Container control processes use a FIPS 140-3 Certified OpenSSL Module. Details of the NIST certification can be viewed at: https://access.redhat.com/compliance/fips. IBM MQ queue managers running in container images have the same FIPS certification level as the base image platform version of IBM MQ.

About this task

MQ_ENABLE_FIPS supports three values.

Important: Linux On Linux s390x, all components will not run in FIPS mode, regardless of the value of the *MQ_ENABLE_FIPS* environment variable. If you require FIPS support on this platform, you should remain at a previous version of IBM MQ. Support for FIPS 140-3 will be enabled in a future FixPack once the IBM Crypto for C (ICC) cryptographic module has received its certification on this platform.

auto
This is the default value.

If the host operating system is FIPS enabled then all components (queue manager, IBM MQ web server and Native HA) run in FIPS mode.

If the host operating system is not FIPS enabled, then all components do not run in FIPS mode.

true
This value turns on FIPS for selected components in the container.

The queue manager attribute **SSLFIPS** is set to YES even if IBM MQ in containers is running on a host operating system that is not FIPS compliant. That is, if the IBM MQ queue manager, web server and Native HA are FIPS compliant, but the operating system of the container is not.

false
This value turns off FIPS compliance.

The queue manager attribute **SSLFIPS** is set to NO, even if IBM MQ in containers is running on a FIPS compliant host machine. However, IBM MQ still secures connections if the private key and certificates are supplied.

JVM options are not updated for the IBM MQ web server. However, the IBM MQ web server still runs an HTTPS endpoint if the private key and certificates are supplied.

Data replication in Native HA does not use FIPS cryptography.

Example

Here is sample queue manager YAML code that describes enabling TLS and FIPS for the queue manager component.

Note: This example uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.

- Example queue manager YAML code for Red Hat OpenShift:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  namespace: ibm-mq-fips
  name: exampleqm
spec:
  license:
    accept: true
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    storage:
      queueManager:
        type: ephemeral
  template:
    pod:
      containers:
        - env:
            - name: MQ_ENABLE_FIPS
              value: "true"
          name: qmgr
  version: 10.0.0.0-r3
  web:
    enabled: false
  pki:
    keys:
      - name: ibmmqtls
        secret:
          secretName: ibm-mq-tls-secret
          items:
          - tls.key
          - tls.crt
```

- Example queue manager YAML code for Amazon EKS:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  namespace: ibm-mq-fips
  name: exampleqm
spec:
  license:
    accept: true
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    storage:
      queueManager:
        type: ephemeral
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  template:
    pod:
      containers:
        - env:
            - name: MQ_ENABLE_FIPS
              value: "true"
          name: qmgr
  version: 10.0.0.0-r3
  web:
    enabled: false
    route:
      enabled: false
  pki:
    keys:
      - name: ibmmqtls
        secret:
          secretName: ibm-mq-tls-secret
          items:
          - tls.key
          - tls.crt
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

Quantum Safe IBM MQ in Containers

Information about Quantum Safe support for IBM MQ in containers and the IBM MQ Operator.

Overview

When you deploy IBM MQ in containers by using the IBM MQ Operator, several endpoints that use TLS communications can benefit from quantum safe protection. Use the information in this topic to configure quantum safe TLS for these container-specific endpoints.

For information about securing the queue manager itself and messages, see Quantum Safe IBM MQ.

Prerequisites

The following versions are required to use quantum safe features in containers:

- IBM MQ 10.0.0.0 or later
- IBM MQ Operator 4.0.0 or later

Quantum Safe endpoints in IBM MQ containers

The following endpoints in IBM MQ container deployments support quantum safe TLS:

- The IBM MQ container metrics server
- The IBM MQ Operator webhook

The following quantum safe key exchange algorithms are accepted:

- X25519MLKEM768
- SecP256r1MLKEM768
- SecP384r1MLKEM1024

The following endpoints do not currently support quantum safe TLS:

- IBM MQ web console
- IBM Instana agent connections

Configuring quantum safe TLS for the metrics server

The IBM MQ metrics server supports, but does not by default require, quantum safe TLS key exchange algorithms when it serves metrics over HTTPS. Quantum safe key exchanges are enabled by default, allowing clients to negotiate their use, but the metrics server accepts connections from clients that do not support quantum safe algorithms unless explicitly configured otherwise.

Platform requirements

To use quantum safe TLS with the metrics server on Red Hat OpenShift, you must have Red Hat OpenShift Container Platform 4.21 or later. This version includes Prometheus 3.7.3, which supports quantum safe TLS.

Configuration

To use quantum safe TLS, you must serve metrics through HTTPS. For more information about configuring HTTPS for metrics, see “Monitoring when using the IBM MQ Operator” on page 244.

Require quantum safe TLS for metrics

When HTTPS is enabled, the metrics server supports quantum safe key exchanges but falls back to traditional key exchanges if the client does not support quantum safe algorithms. To enforce quantum safe TLS and reject connections that do not use quantum safe key exchanges, set the `MQ_METRICS_REQUIRE_QUANTUM_SAFE` environment variable to `true`.

Example: Configuring required quantum safe TLS for metrics

The following example shows how to configure a queue manager to require quantum safe TLS for the metrics server (change the `license.accept` field to `true` if you accept the license, which in this example is for IBM MQ Advanced):

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: qm-quantum-safe-metrics
spec:
  version: 10.0.0.0-r3
  license:
    accept: true
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    env:
    - name: MQ_METRICS_REQUIRE_QUANTUM_SAFE
      value: "true"
```

When `MQ_METRICS_REQUIRE_QUANTUM_SAFE` is set to `true`, the metrics server rejects any TLS connections that do not support quantum safe key exchanges.

Verifying quantum safe configuration

When you configure the metrics server to require quantum safe key exchanges, the following message is logged:

```
2026-05-20T11:38:57.644Z Metrics server configured to require quantum-safe key exchange
```

You can verify this message by examining the queue manager pod logs.

Configuring quantum safe TLS for the IBM MQ Operator webhook

The IBM MQ Operator webhook supports quantum safe TLS key exchange algorithms when it communicates with the Kubernetes API server. Quantum safe key exchanges are enabled by default for the operator webhook.

Platform requirements

On Red Hat OpenShift, the API server supports quantum safe TLS from OpenShift Container Platform 4.20 or later.

Configuration

No additional configuration is required to enable quantum safe support for the operator webhook. The webhook automatically negotiates quantum safe key exchanges with the API server when both support them.

Related reference

"Licensing and API references for the IBM MQ Operator" on page 281

IBM MQ provides a Kubernetes Operator, which you can deploy on Red Hat OpenShift Container Platform and, from IBM MQ Operator 3.9.0, Amazon EKS. The `v1beta1` API can be used to create and manage queue manager resources.

Planning scalability and performance for IBM MQ in containers

In most cases, scaling and performance of IBM MQ in containers is the same as IBM MQ for Multiplatforms. However there are a few additional limits that can be imposed by the container platform.

About this task

When planning scalability and performance for IBM MQ in containers, consider the following options:

Procedure

- **Limit the number of threads and processes.**

  IBM MQ uses threads to manage concurrency. In Linux, threads are implemented as processes, so you can encounter limits imposed by the container platform or operating system, on the maximum number of processes. For users deploying on Red Hat OpenShift Container Platform, there is a default limit of 4096 processes per container. While this is adequate for the vast majority of scenarios, there might be cases where this can impact the number of client connections for a queue manager.

  For users deploying the IBM MQ Operator on Amazon EKS, or deploying IBM MQ queue managers on Kubernetes, the process limit in Kubernetes can be configured by a cluster administrator by using the kubelet configuration setting **`podPidsLimit`**. See Process ID limits and reservations in the Kubernetes documentation. In Red Hat OpenShift Container Platform, you can also create a **`ContainerRuntimeConfig`** custom resource to edit CRI-O parameters.

  In your IBM MQ configuration, you can also set the maximum number of client connections for a queue manager. See Server-connection channel limits for applying limits to an individual server-connection channel, and the MAXCHANNELS INI attribute for applying limits to the whole queue manager.

  In addition to IBM MQ and pod process limits, Kubernetes platforms have a **`threads-max`** limit for each worker node. If a node is reaching its threads limit, it cannot create new processes and sets the node **`PIDPressure`** condition to `True` (see Conditions in the Kubernetes documentation). The **`threads-max`** limit is determined by the node machine size and the underlying operating system configuration. For example, an Amazon EC2 t3.medium machine has 4 GiB of RAM, which typically results in a **`threads-max`** value of approximately 30,000. When you select the machine type for your Kubernetes node, ensure that it has sufficient RAM to handle the number of threads required for all the queue managers that will run on that node. This setting is particularly important for deployments that have a high number of client connections or server-connection channels.

- **Limit the number of volumes.**

In cloud and container systems, network-attached storage volumes are commonly used. There are limits to the number of volumes that can be attached to Linux Nodes. For example, AWS EC2 limits to no more than 30 volumes per VM. Red Hat OpenShift Container Platform has a similar limit, as do Microsoft Azure and Google Cloud Platform.

A Native HA queue manager requires one volume for each of the three instances, and enforces instances to be spread across Nodes. However, you can configure the queue manager to use three volumes per instance (queue manager data, recovery logs and persisted data).

- **Use IBM MQ scaling techniques.**

  Instead of a small number of large queue managers, it can be beneficial to use IBM MQ scaling techniques such as IBM MQ uniform clusters to run multiple queue managers with the same configuration. This has the added benefit that the impact of a single container restarting (for example, as part of container platform maintenance) is lessened.

Planning networking for IBM MQ in containers

Networking considerations when planning your IBM MQ in containers configuration.

Procedure

- “Planning networking for IBM MQ in containers” on page 39
- “Internet Protocol version support for MQ in containers” on page 40

Considerations for connecting to a queue manager from outside of your Kubernetes cluster

In Kubernetes and Red Hat OpenShift Container Platform, the queue manager Pod can move around between Nodes. Kubernetes uses a software-defined overlay network, to keep consistent network addressability within the Kubernetes cluster. For example, a “ClusterIP” Service provides a fixed IP address within the Kubernetes cluster, which will always route to the Pod, no matter which Node it is scheduled on. If you want to connect to a queue manager from outside the Kubernetes cluster, then you need something which sits on both the network which is internal to the Kubernetes cluster, and the “external” network, which can then route traffic to the right place. There are several possibilities for routing technology, including IBM MQ Internet Pass-Thru, a `NodePort` Service, a cloud-based load balancer, or a software load balancer such as HAProxy. Red Hat OpenShift Container Platform provides an SNI-based router using HAProxy by default, but other types of routing are available.

Note: You must maintain a one-to-one mapping in connections between queue managers. For example, if QM1 outside your Kubernetes cluster is configured to talk to QM2 inside the Kubernetes cluster, it must always talk to QM2 (you can’t load balance to QM3 as well). This is because state is held at both ends of the connection.

The following are examples of some of the different options for routing network traffic.

SNI-based router

SNI-based routers listen on a single port and then use the TLS Server Name Indicator (SNI) header to determine how to route traffic. As long as you use the router's IP address, or a DNS entry which resolves to it, then the router can receive the connection. In the case of an unencrypted TCP/IP connection, the router does not know what the contents of the packets mean (the IBM MQ protocol), so doesn't know how to route the traffic. With a TLS connection, it can see the TLS SNI header and use that to route traffic. IBM MQ has two options for what to put in the SNI header:

- `OutboundSNI=CHANNEL` puts the IBM MQ channel name in the SNI header. When using an SNI-based router, using `OutboundSNI=CHANNEL` means that you would need a unique MQ channel name for an entire Kubernetes cluster, which is often undesirable. For example if the router sees “MY.SVRCONN” in the SNI header, you have to decide which queue manager that connection is for in the router configuration (for example, the “Route” resource in Red Hat OpenShift Container Platform). This option

also allows you to use a different certificate on each IBM MQ channel, because the certificate to use can be determined before decrypting the connection.

- `OutboundSNI=HOSTNAME` puts the hostname the client is using to connect in the SNI header. Multiple host names could be routed to the same router using DNS wildcard entries. For example, the DNS server might be configured to route `*.apps.mydomain.com` to your router, and the SNI header would include the full address, such as `qm1.apps.mydomain.com`. The router can then map this SNI header to the intended queue manager. This unfortunately means that you can't use multiple certificates for different channels, as there's no way to know which channel you're intending to use, until IBM MQ has performed the TLS handshake and decrypted the data.

Cloud-based router or `LoadBalancer` service

You can also deploy a cloud-based router (for example, an AWS Network Load Balancer) per queue manager. In this case, each router would gets its own IP address and be individually addressable. The disadvantage here is often increased cloud costs.

This sort of router can often be managed in Kubernetes by using a `LoadBalancer` service.

IBM MQ Internet Pass-Thru

IBM MQ Internet Pass-Thru is a routing component designed for running at the edge of an enterprise network. If you want to run this in Kubernetes, you will need to deploy it yourself.

`NodePort` service

A `NodePort` service issues a unique port number to every queue manager. e.g. connecting on port 32000 is for QM1, and 32001 is for QM2. Every Node listens on this address, and can then forward the traffic onto the right place. This is quite a simple approach, but is often discouraged for several reasons:

- You can't easily use `NetworkPolicy` with `NodePort` services. Using `NodePort` services is flagged by some security scanners as being insecure.
- You need to run Kubernetes Nodes on the external network, and thus manage Pod placement carefully.
- You are using non-standard port numbers, and the port numbers for a particular use may potentially change. Port 1414 is the standard port for MQ traffic, and port 443 for HTTPS — but with a `NodePort`, a network administrator can't tell what's going on.
- Port allocation can be problematic. In you can automatically assign a port number dynamically, or you can specify it manually. Specifying it manually can result in increased predictability, but can cause clashes.
- The high port numbers used by `NodePort` services can create problems on some networks where egress might only be allowed on certain known ports.

Internet Protocol version support for MQ in containers

IBM MQ in Containers provides IPv4, IPv6 and dual-stack (IPv4, IPv6) support.

When you deploy on a single-stack IPv4 or IPv6 environment, no additional configuration is required unless you deploy on a single-stack IPv6 system on Amazon EKS (see the following section).

When you deploy on a dual-stack environment, if you are using the IBM MQ Operator, you can configure the `IPFamilies` and `IPFamilyPolicy` settings for the queue manager services via the `QueueManager spec.service`.

Amazon EKS single-stack IPv6 systems

If you are using single-stack IPv6 systems on Amazon EKS, modify the following Kubernetes resources when you configure connections from outside your cluster:

`Ingress`
Add the following values to the `metadata.annotations`:

- `alb.ingress.kubernetes.io/ip-address-type: "dualstack"`
- `alb.ingress.kubernetes.io/target-type: "ip"`

For more information, see the *Create an Ingress* step of "Configuring Ingress to connect to a queue manager IBM MQ Console and IBM MQ Operator REST API from outside an Amazon EKS cluster" on page 173.

`Service` (of type `LoadBalancer`)

Add the following values to the `metadata.annotations`:

- `service.beta.kubernetes.io/aws-load-balancer-ip-address-type: "dualstack"`
- `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"`

For more information, see the *Create a LoadBalancer Service* step of "Configuring a LoadBalancer to connect to a queue manager for Amazon EKS deployments" on page 171.

For more information about using single-stack IPv6 systems on Amazon EKS, see The journey to IPv6 on Amazon EKS: Implementation patterns (Part 2) in the AWS blog.

Preparing, installing and upgrading your environment for IBM MQ in containers

You perform a range of tasks to prepare your environment for IBM MQ

About this task

If you are using the IBM MQ Operator, prepare your cluster by installing the operator. For more information, see the topic that is relevant to your platform:

- Red Hat OpenShift Container Platform: "Installing and upgrading the IBM MQ Operator on Red Hat OpenShift" on page 42
- Amazon EKS: "Installing and upgrading the IBM MQ Operator on Amazon EKS" on page 66

Otherwise, you prepare your container environment by building your own container images. See "Preparing for IBM MQ by building your own container image" on page 87

OpenShift CP4I Dependencies for the IBM MQ Operator

No other operators are installed automatically when you install IBM MQ Operator.

The IBM Licensing Operator needs to be installed separately to track license usage.

- For Red Hat OpenShift users, see Deploying License Service in the IBM Cloud Pak for Integration documentation.
- For Amazon EKS users, see Installing License Service on the Kubernetes cluster in the IBM Cloud Pak foundational services documentation.

OpenShift CP4I **Red Hat OpenShift users only:** When you create a `QueueManager` using an IBM Cloud Pak for Integration license, you can choose whether or not you want to use single sign-on with the IBM Cloud Pak for Integration instance of Keycloak. Use of Keycloak is enabled by default with an IBM Cloud Pak for Integration license, but if it is not installed the `QueueManager` will enter a "Blocked" state until the correct dependencies are installed. See "Installing the IBM MQ Operator for use with IBM Cloud Pak for Integration on Red Hat OpenShift" on page 50 for more details on the dependencies.

Cluster-scoped permissions required by the IBM MQ Operator

The IBM MQ Operator always requires cluster-scoped permissions to read storage class and cluster version information.

The IBM MQ Operator can be installed in two different scopes:

1. **Cluster scope** means that the operator will be assigned permissions using `ClusterRole` and `ClusterRoleBinding` resources. These will enable the operator to act across all namespaces.
2. **Namespace scope** means that the operator will be assigned permissions using `Role` and `RoleBinding`, which will only give permissions in the given namespace. In addition, there will be a `ClusterRole` and `ClusterRoleBinding` for the following read-only permissions:

- **Red Hat OpenShift only:** Permission to read the cluster version. This allows the Operator to feed back any issues with the cluster environment.
  - API Groups: **`config.openshift.io`**
  - Resources: **`clusterversions`**
  - Verbs: **`get, list, watch`**
- **All deployments:** Permission to read storage classes on the cluster. This allows the Operator to attempt to feed back any issues with selected storage classes in containers.
  - API Groups: **`storage.k8s.io`**
  - Resources: **`storageclasses`**
  - Verbs: **`get, list`**

Installing and upgrading the IBM MQ Operator on Red Hat OpenShift

You perform a range of tasks to install, uninstall, and upgrade the IBM MQ Operator on the Red Hat OpenShift Container Platform.

Before you begin

Note: For information about installing and upgrading on Amazon EKS, see “Installing and upgrading the IBM MQ Operator on Amazon EKS” on page 66.

About this task

To get started with installing and upgrading the IBM MQ Operator on Red Hat OpenShift Container Platform, see the following topics.

Procedure

- “Installing the IBM MQ Operator on Red Hat OpenShift” on page 43
- “Installing the IBM MQ Operator for use with IBM Cloud Pak for Integration on Red Hat OpenShift” on page 50
- “Upgrading the IBM MQ Operator and queue managers on Red Hat OpenShift” on page 54
- “Uninstalling the IBM MQ Operator on Red Hat OpenShift” on page 64

Installing the IBM MQ Operator on Red Hat OpenShift

The IBM MQ Operator can be installed onto Red Hat OpenShift using the OpenShift console or command line interface (CLI).

Before you begin

Note: To install the IBM MQ Operator from Amazon EKS, see "Installing the IBM MQ Operator on Amazon EKS" on page 66.

Important:

- This topic is for installing the IBM MQ Operator onto Red Hat OpenShift for standalone use **only**. If you intend to use the IBM Cloud Pak for Integration, or Keycloak SSO for one or more queue managers, see "Installing the IBM MQ Operator for use with IBM Cloud Pak for Integration on Red Hat OpenShift" on page 50.
- Review the guidance on structuring your deployment before you install the IBM MQ Operator.

To ensure that your installation goes as smoothly as possible, make sure that you understand all of the prerequisites and requirements before you start your installation. See "Planning for IBM MQ in containers" on page 9.

About this task

The following steps represent the typical task flow for installing your IBM MQ Operator on Red Hat OpenShift:

1. Install Red Hat OpenShift Container Platform.
2. Configure storage.
3. Mirror images (air-gap only).
4. Add the IBM MQ Operator catalog.
5. Install the IBM MQ Operator.
6. Create the entitlement key secret (online installs only).
7. Deploy the License Service.
8. Deploy a queue manager.

Procedure

1. Install Red Hat OpenShift Container Platform.

   For detailed steps to install OpenShift, see Installing Red Hat software 4.6 or later.

   Important: Ensure that you install a supported version of OpenShift Container Platform. For example, to use IBM MQ Operator 4.0 or later, you must install OpenShift Container Platform 4.12 or later. For more information, see IBM Cloud Pak and Red Hat OpenShift Container Platform compatibility.

   For any steps that use the Red Hat OpenShift Container Platform CLI, you must be logged in to your OpenShift cluster with `oc login`. To install the CLI, see Getting started with the OpenShift CLI.

   After you install OpenShift, you can verify and gain access to your container software by using the IBM entitlement key that you create in Create the entitlement key secret.
2. Configure storage.

   You must define storage classes in Red Hat OpenShift Container Platform and set your storage configuration to satisfy your sizing requirements.

   Important: IBM MQ single-instance and Native HA queue managers can use RWO access mode, while multi-instance queue managers require RWX as described in "Planning storage for the IBM MQ Operator" on page 18. IBM MQ multi-instance queue managers require particular file system

characteristics, which can be verified using the instructions for Testing a shared file system for IBM MQ.

A list of known compliant and non-compliant file systems, and notes on other limits or restrictions, can be found in the Testing statement for IBM MQ file systems.

Recommended storage providers can be found on the CP4I Storage considerations page.

3. Mirror images (air-gap only).

   If your cluster is in a restricted (air-gapped) network environment, you must mirror the IBM MQ images.

   Important:

   The IBM MQ CASE contains only a single queue manager version. The image mirroring process runs considerably faster, but might need to be run multiple times: once for each version of the queue manager that is being installed.

   For more information, see "IBM MQ CASE for air-gap install and upgrade" on page 329.

   You must look up which IBM MQ CASE versions contain the operator and queue manager versions you require, and repeat the air-gap mirror process for each. See the IBM MQ CASE Lookup Tables for the version mappings.

   For example, when following the instructions linked below to mirror images, you install the latest operator and single corresponding queue manager version by using the following values:

   ```
   export OPERATOR_PACKAGE_NAME=ibm-mq
   export OPERATOR_VERSION=4.0.2
   ```

   For information about the contents of the IBM MQ CASE, which is used for air-gap installs and upgrades, see "IBM MQ CASE for air-gap install and upgrade" on page 329.

   To create mirror images, see Mirroring images for an air-gapped cluster.

4. Add the IBM MQ Operator catalog source.

   Add the catalog source that makes the operators available to your cluster. See "Adding the IBM MQ Operator catalog source" on page 45.

5. Install the IBM MQ Operator.

   Choose one of the following two options (use the console, or use the CLI):

   - Option 1: Install the IBM MQ Operator using the OpenShift console.
   - Option 2: Install the IBM MQ Operator using the OpenShift CLI.

6. Create the entitlement key secret (online installs only).

   The IBM MQ Operator deploys queue manager images that are pulled from a container registry that performs a license entitlement check. This check requires an entitlement key that is stored in a `docker-registry` pull secret. If you do not yet have an entitlement key in the namespace in which you will install queue managers, follow these instructions to get an entitlement key and create a pull secret.

   Note: The entitlement key is not required if only IBM MQ Advanced for Developers (Non-Warranted) queue managers are going to be deployed.

   You can create the entitlement key secret using either the OpenShift console or the CLI. The following example uses the CLI:

   a. Get the entitlement key that is assigned to your IBM ID. Log in to MyIBM Container Software Library with the IBM ID and password that are associated with the entitled software.

   b. In the **Entitlement keys** section, select **Copy key** to copy the entitlement key to the clipboard.

c. From the OpenShift CLI, run the following command to create an image pull secret called `ibm-entitlement-key`.

```
oc create secret docker-registry ibm-entitlement-key \
--docker-server=cp.icr.io \
--docker-username=cp \
--docker-password=entitlement_key \
--docker-email=user_email
\--namespace=namespace
```

Where *entitlement_key* is the entitlement key that you copied in step b, *user_email* is the IBM ID associated with the entitled software, and *namespace* is the namespace that you installed your IBM MQ Operator into.

7. Deploy the License Service.

   This is required for monitoring license usage of queue managers. Follow the instructions in Deploying License Service.

8. Deploy a queue manager.

   For instructions on deploying an example "quick start" queue manager, see “Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97.

Related tasks

“Uninstalling the IBM MQ Operator on Red Hat OpenShift” on page 64
You can use the Red Hat OpenShift console or CLI to uninstall the IBM MQ Operator from Red Hat OpenShift.

OpenShift *Adding the IBM MQ Operator catalog source*

Add the IBM MQ Operator catalog source to your OpenShift cluster to make the IBM MQ Operator available for installation. This task is also required if you are applying catalog source fix packs before completing an upgrade.

About this task

An Operator Catalog is an index of operators available to extend the API of a Red Hat OpenShift Container Platform cluster to enable IBM software products.

The following catalog sources are available:

Option 1: Specific catalog source for the IBM MQ Operator.

By using a specific IBM MQ Operator catalog source, you gain full control of software versioning on a cluster and when upgrades happen. A new IBM MQ Operator version becomes available in an OpenShift cluster **only** after you update the catalog source. This process effectively gives you manual control of upgrades, so you do not need to use the `Manual` option for the **`Update approval`** setting for operators. The **Manual** option forces all possible upgrades to be done at the same time and can block upgrades, so use the **Automatic** option only. For more information, see the "Restricting automatic updates with an approval strategy" section of Installing the operators by using the Red Hat OpenShift console.

Choose this option if you are completing an upgrade and need to add the IBM MQ Operator catalog source of a newer version.

To use this option, skip to Option 1: Add specific catalog sources for the IBM MQ Operator.

Option 2: IBM Operator Catalog.

With this option, new operator versions become available and are applied **without** any intervention from you. So use this option **only** for online installations where you want **automatic** upgrades of the IBM MQ Operator, and where deterministic installations are not needed.

Note: This option can be useful for proof-of-concept environments but it is **not suitable for production environments**.

To use this option, skip to Option 2: Add the IBM Operator Catalog.

Procedure

- **Option 1: Add specific catalog sources for the IBM MQ Operator.**

  This task assumes that you have completed the first 3 steps of “Installing the IBM MQ Operator on Red Hat OpenShift” on page 43.

  This task must be performed by a cluster administrator and must be performed using the CLI.

  a) Upgrade only: If you are applying catalog source fix packs before an upgrade, complete the following steps:

  - Confirm that your operators are running properly.
  - If there are any pending IBM MQ Operator updates that require manual approval, approve those before starting this procedure. For more information, see "Restricting automatic updates with an approval strategy" in Installing the operators by using the Red Hat OpenShift console.

  b) If you have not already installed it, or if it needs updating, download the IBM Catalog Management plug-in (version 1.6.0 or later) from GitHub.

  This plug-in allows you to run **oc ibm-pak** commands against the cluster.

  c) Log into your cluster by using the **oc login** command and your user credentials:

```
oc login openshift_url -u username -p password -n namespace
```

  d) Export the following environment variables for the IBM MQ Operator:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=CHOSEN_OPERATOR_VERSION
export ARCH=ARCHITECTURE
```

  where

  - *ARCHITECTURE* is the architecture of the system on which you are deploying the IBM MQ Operator, and has a value of `amd64`, `ppc64le`, or `s390x`.
  - *CHOSEN_OPERATOR_VERSION* is your required version of the IBM MQ Operator. The current CD version is 4.0.2. The current SC2 version is 4.0.2.

  Important: If you are moving from the IBM Operator Catalog to the specific catalog source for the IBM MQ Operator, set `OPERATOR_VERSION` to the version of your deployment of the IBM MQ Operator.

  e) Download the files for the IBM MQ Operator.

  Note: If you are completing an **air-gapped** installation, you should already have the files that you need after completing the "Mirror images" step of *Installing the IBM MQ Operator on Red Hat OpenShift*, in which case you can skip to step “8” on page 46 "Apply the IBM MQ Operator catalog source to the cluster".

```
oc ibm-pak get ${OPERATOR_PACKAGE_NAME} --version ${OPERATOR_VERSION}
```

  f) Generate the catalog source required for the IBM MQ Operator:

```
oc ibm-pak generate mirror-manifests ${OPERATOR_PACKAGE_NAME} icr.io --version $
{OPERATOR_VERSION}
```

  g) Optional: Generate the catalog sources and save them on another directory.

  a. Get the catalog source:

```
cat ~/.ibm-pak/data/mirror/${OPERATOR_PACKAGE_NAME}/${OPERATOR_VERSION}/catalog-
sources.yaml
```

  b. (Optional) Navigate to the directory in your file browser to copy these artifacts into files that you can keep for re-use or for pipelines.

  h) Apply the IBM MQ Operator catalog source to the cluster.

```
oc apply -f ~/.ibm-pak/data/mirror/${OPERATOR_PACKAGE_NAME}/${OPERATOR_VERSION}/catalog-
sources.yaml
```

i) Confirm that the IBM MQ Operator catalog source was created in the `openshift-marketplace` namespace:

```
oc get catalogsource -n openshift-marketplace
```

Example output:

```
oc get catalogsource -n openshift-marketplace

NAME                           DISPLAY                TYPE   PUBLISHER   AGE
ibmmq-operator-catalogsource   ibm-mq-3.1.3           grpc   IBM         23h
```

You are now ready to complete Step 5 of Installing the IBM MQ Operator on Red Hat OpenShift.

- **Option 2: Add the IBM Operator Catalog.**

Important: Use the IBM Operator Catalog **only** for online installations where you want **automatic** upgrades of the IBM MQ Operator, and where deterministic installations are not needed. This option can be useful for proof-of-concept environments but it is **not suitable for production environments**.

The IBM Operator Catalog is an index of operators available to extend the API of a Red Hat OpenShift Container Platform cluster to enable IBM software products. Adding the catalog sources to your OpenShift cluster adds the IBM operators to the list of operators you can install.

This task assumes that you have completed the first 3 steps of “Installing the IBM MQ Operator on Red Hat OpenShift” on page 43.

This task can be performed by using the CLI or by using the OpenShift web console.

Using the CLI

1. Copy the following resource definition for IBM operators into a local file on your computer:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: ibm-operator-catalog
  namespace: openshift-marketplace
spec:
  displayName: IBM Operator Catalog
  publisher: IBM
  sourceType: grpc
  image: icr.io/cpopen/ibm-operator-catalog:latest
  updateStrategy:
    registryPoll:
      interval: 45m
```

2. Run the following command. Replace *filename*.yaml with the name of the file you created in the previous step:

```
oc apply -f filename.yaml
```

Using the OpenShift web console

1. Log into the OpenShift web console with your OpenShift cluster administrator credentials.
2. In the banner, click the plus ("+") icon to open the **Import YAML** dialog box.

   Note: You do not need to select a value for **Project**. The YAML code in the next step already includes the correct value for `metadata:namespace`, which ensures that the catalog source is installed in the correct project (namespace).
3. Paste the following resource definition into the dialog box:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: ibm-operator-catalog
  namespace: openshift-marketplace
```

```
spec:
  displayName: IBM Operator Catalog
  image: 'icr.io/cpopen/ibm-operator-catalog:latest'
  publisher: IBM
  sourceType: grpc
  updateStrategy:
    registryPoll:
      interval: 45m
```

4. Click **Create**.

You are now ready to complete Step 5 of Installing the IBM MQ Operator on Red Hat OpenShift.

OpenShift CP4I *Installing the IBM MQ Operator using the OpenShift console*

The IBM MQ Operator can be installed onto Red Hat OpenShift using the OperatorHub.

Before you begin

This task assumes that you have completed Steps 1-4 of "Installing the IBM MQ Operator on Red Hat OpenShift" on page 43.

Procedure

1. Log in to your Red Hat OpenShift cluster console.
2. From the navigation pane, click **Operators** > **OperatorHub**.

   The OperatorHub page is displayed.
3. In the **All Items** field, enter "IBM MQ".

   The IBM MQ catalog entry is displayed.
4. Select **IBM MQ**.

   The IBM MQ window is displayed.
5. Click **Install**.

   The Install Operator page is displayed.
6. Enter the following values:
   a) Set **Channel** to your chosen version.

      See "Release history for IBM MQ Operator" on page 5 to determine which operator channel to choose (the IBM MQ version for each operator release is specified by the "Allowed value for `.spec.version`").

   b) Set **Installation Mode** to either "a specific namespace on the cluster" (which you can create in the next step), or the cluster-wide scope.

      Choosing the cluster-wide scope is recommended, because installing different versions of an Operator in different namespaces can lead to problems. Operators are designed to be extensions of the control plane.

   c) Optional: If you chose "a specific namespace on the cluster", set the **Namespace** to the project (namespace) value that you want to install the operator into.

      Note: When using the console to install the operator, you can either use an existing namespace, the default namespace that is provided by the operator, or create a new namespace. If you want to create a new namespace you can create it from this form, as follows: From the navigation pane, click **Home** > **Projects**, select **Create Project**, specify the **Name** of the project (the namespace) that you want to create, then click **Create**.

   d) Set **Approval Strategy** to Automatic.
7. Click **Install** and wait for your operator to install.

   You are provided with a confirmation when the installation is complete.

To verify the installation, navigate to **Operators** > **Installed Operators**, and select your project from the **Projects** drop-down list. The status of the operator changes to Succeeded when the installation is complete.

What to do next

You are now ready to Create the entitlement key secret (step 6 of "Installing the IBM MQ Operator on Red Hat OpenShift" on page 43).

Installing the IBM MQ Operator using the Red Hat OpenShift CLI

The IBM MQ Operator can be installed onto Red Hat OpenShift using the command line interface (CLI).

Before you begin

This task assumes that you have completed Steps 1-4 of "Installing the IBM MQ Operator on Red Hat OpenShift" on page 43.

Procedure

1. Log into the Red Hat OpenShift command line interface (CLI) using **`oc login`**.
2. Optional: Create a namespace to use for the IBM MQ Operator.

   The IBM MQ Operator can be installed scoped to a single namespace or all namespaces. This step is only needed if you want to install into a particular namespace that does not already exist.

   To create a new namespace in the CLI, run the following command:

   ```
   oc create namespace namespace_name
   ```

   Where *namespace_name* is the name of the namespace that you want to create.
3. View the list of operators available to the cluster from the OperatorHub:

   ```
   oc get packagemanifests -n openshift-marketplace
   ```
4. Inspect the IBM MQ Operator to verify its supported **`InstallModes`** and available **`Channels`**.

   ```
   oc describe packagemanifests ibm-mq -n openshift-marketplace
   ```
5. Optional: Create an **`OperatorGroup`**.

   An **`OperatorGroup`** is an OLM resource that selects target namespaces in which to generate required RBAC access for all operators in the same namespace as the **`OperatorGroup`**.

   The namespace to which you subscribe the operator must have an **`OperatorGroup`** that matches the operator's **`InstallMode`**, either the `AllNamespaces` or `SingleNamespace` mode.

   If the operator you want to install uses `AllNamespaces` mode, then the `openshift-operators` namespace already has an appropriate **`OperatorGroup`** in place, and you can skip this step.

   If the Operator uses the `SingleNamespace` mode and you do not already have an appropriate **`OperatorGroup`** in place, create one by running the following command:

   ```
   cat << EOF | oc apply -f -
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: operatorgroup_name
     namespace: namespace_name
   spec:
     targetNamespaces:
     - namespace_name
   EOF
   ```
6. See "Release history for IBM MQ Operator" on page 5 to determine which operator channel to choose (the IBM MQ version for each operator release is specified by the "Allowed value for `.spec.version`").

7. Install the operator.

   Use the following command, changing the following values:

   - Change *ibm_mq_operator_channel* to match the channel for the version of the IBM MQ Operator you want to install.
   - Change *namespace_name* to **openshift-operators** if you are using "AllNamespaces" mode, or to the namespace you want to deploy the IBM MQ Operator to if you are using "SingleNamespace" mode.
   - Change *catalog_source_name* to the name of the catalog source being used to install the IBM MQ Operator, as added in the previous install step "Adding the IBM MQ Operator catalog source" on page 45.

   ```
   cat << EOF | oc apply -f -
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: ibm-mq
     namespace: namespace_name
   spec:
     channel: ibm_mq_operator_channel
     installPlanApproval: Automatic
     name: ibm-mq
     source: catalog_source_name
     sourceNamespace: openshift-marketplace
   EOF
   ```

8. After a few minutes, the operator is installed. Run the following command to verify that all of the components are in the Succeeded state:

   ```
   oc get csv -n namespace_name | grep ibm-mq
   ```

   Where *namespace_name* is **openshift-operators** if you are using "AllNamespaces" mode, or the project (namespace) name if you are using "SingleNamespace" mode.

What to do next

You are now ready to Create the entitlement key secret (step 6 of "Installing the IBM MQ Operator on Red Hat OpenShift" on page 43).

OpenShift CP4I Installing the IBM MQ Operator for use with IBM Cloud Pak for Integration on Red Hat OpenShift

For use with the IBM Cloud Pak for Integration, the IBM MQ Operator can be installed onto Red Hat OpenShift through the OpenShift console or command line interface (CLI).

Before you begin

Note: To install the IBM MQ Operator on Amazon EKS, see "Installing the IBM MQ Operator on Amazon EKS" on page 66.

Important:

- This topic is for installing the IBM MQ Operator for use with IBM Cloud Pak for Integration, or if you intend to deploy at least one of your queue managers using a IBM Cloud Pak for Integration license **only**. For instructions on installing the IBM MQ Operator for standalone use, see "Installing the IBM MQ Operator on Red Hat OpenShift" on page 43.
- Review the guidance on structuring your deployment before you install the IBM MQ Operator.

To ensure that your installation goes as smoothly as possible, make sure that you understand all of the prerequisites and requirements before you start your installation. See "Planning for IBM MQ in containers" on page 9.

About this task

The following steps represent the typical task flow for installing your IBM MQ Operator onto Red Hat OpenShift:

1. Install Red Hat OpenShift Container Platform.
2. Configure storage.
3. Mirror images (air-gap only).
4. Add the IBM MQ Operator catalog and prepare your cluster.
5. Install the IBM MQ Operator.
6. Create the entitlement key secret (online installs only).
7. Optional: Install IBM Cloud Pak for Integration and its dependencies.
8. Deploy the License Service.
9. Deploy a queue manager.

Procedure

1. Install Red Hat OpenShift Container Platform.

   For detailed steps to install OpenShift, see Installing Red Hat software 4.6 or later.

   Important: Ensure that you install a supported version of OpenShift Container Platform. For example, to use IBM MQ Operator 4.0 or later, you must install OpenShift Container Platform 4.12 or later. For more information, see IBM Cloud Pak and Red Hat OpenShift Container Platform compatibility.

   For any steps that use the Red Hat OpenShift Container Platform CLI, you must be logged in to your OpenShift cluster with `oc login`. To install the CLI, see Getting started with the OpenShift CLI.

   After you install OpenShift, you can verify and gain access to your container software by using the IBM entitlement key that you create in Create the entitlement key secret.

2. Configure storage.

   You must define storage classes in Red Hat OpenShift Container Platform and set your storage configuration to satisfy your sizing requirements.

   Important: IBM MQ single-instance and Native HA queue managers can use RWO access mode, while multi-instance queue managers require RWX as described in “Planning storage for the IBM MQ Operator” on page 18. IBM MQ multi-instance queue managers require particular file system characteristics, which can be verified using the instructions for Testing a shared file system for IBM MQ.

   A list of known compliant and non-compliant file systems, and notes on other limits or restrictions, can be found in the Testing statement for IBM MQ file systems.

   Recommended storage providers can be found in Storage considerations in the IBM Cloud Pak for Integration documentation.

3. Mirror images (air-gap only).

   If your cluster is in a restricted (air-gapped) network environment, you must mirror the IBM MQ images. Depending on your configuration, you might also need to mirror some additional components. Read the following information, then mirror the images as required.

   - You must mirror the IBM MQ images.

     Important:

     The IBM MQ CASE contains only a single queue manager version. The image mirroring process runs considerably faster, but might need to be run multiple times: once for each version of the queue manager that is being installed.

     For more information, see “IBM MQ CASE for air-gap install and upgrade” on page 329.

You must look up which IBM MQ CASE versions contain the operator and queue manager versions you require, and repeat the air-gap mirror process for each. See the IBM MQ CASE Lookup Tables for the version mappings.

For example, when following the instructions linked below to mirror images, you install the latest operator and single corresponding queue manager version by using the following values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=4.0.2
```

For information about the contents of the IBM MQ CASE, which is used for air-gap installs and upgrades, see "IBM MQ CASE for air-gap install and upgrade" on page 329.

- You must also mirror some additional required components if you intend to deploy at least one queue manager where **all** of the following statements are true:
  - You are using a IBM Cloud Pak for Integration license.
  - The IBM MQ Console is enabled.
  - You are using the IBM Cloud Pak for Integration Keycloak service for IBM MQ Console single sign-on (SSO) authentication and authorization (the default).

  If all of the previous statements are true, then SSO is provided by Keycloak. Therefore, as well as for the IBM MQ Operator, you must also repeat the steps for each of these additional required components:
  - IBM Cloud Pak for Integration
  - Keycloak (Red Hat OpenShift operator)

To create mirror images, see Mirroring images for an air-gapped cluster.

4. Add the IBM MQ Operator catalog source.

   Add the catalog source that makes the IBM MQ Operator available to your cluster using the following values:

   ```
   export OPERATOR_PACKAGE_NAME=ibm-mq
   export OPERATOR_VERSION=CHOSEN_OPERATOR_VERSION
   export ARCH=ARCHITECTURE
   ```

   where

   - *ARCHITECTURE* is the architecture of the system on which you are deploying the IBM MQ Operator, and has a value of `amd64`, `ppc64le`, or `s390x`.
   - *CHOSEN_OPERATOR_VERSION* is your required version of the IBM MQ Operator. The current CD version is 4.0.2. The current SC2 version is 4.0.2.

   There are some additional required components when you deploy at least one queue manager where **all** of the following statements are true:

   - You are using a IBM Cloud Pak for Integration license.
   - The IBM MQ Console is enabled.
   - You are using the IBM Cloud Pak for Integration Keycloak service for IBM MQ Console single sign-on (SSO) authentication and authorization (the default).

   If all of the previous statements are true, then SSO is provided by Keycloak. Therefore, as well as for the IBM MQ Operator catalog source, you must also repeat the steps for each of these additional required components:

   - IBM Cloud Pak for Integration
   - Keycloak (Red Hat OpenShift operator)

   Follow the steps for your required catalog sources in Adding catalog sources to a cluster.

5. Install the IBM MQ Operator.

   Choose one of the following two options (use the console, or use the CLI):

- Option 1: Install the IBM MQ Operator using the OpenShift console.
- Option 2: Install the IBM MQ Operator using the OpenShift CLI.

6. Create the entitlement key secret (online installs only).

   The IBM MQ Operator deploys queue manager images that are pulled from a container registry that performs a license entitlement check. This check requires an entitlement key that is stored in a `docker-registry` pull secret. If you do not yet have an entitlement key in the namespace in which you will install queue managers, follow these instructions to get an entitlement key and create a pull secret.

   Note: The entitlement key is not required if only IBM MQ Advanced for Developers (Non-Warranted) queue managers are going to be deployed.

   You can create the entitlement key secret using either the OpenShift console or the CLI. The following example uses the CLI:

   a. Get the entitlement key that is assigned to your IBM ID. Log in to MyIBM Container Software Library with the IBM ID and password that are associated with the entitled software.

   b. In the **Entitlement keys** section, select **Copy key** to copy the entitlement key to the clipboard.

   c. From the OpenShift CLI, run the following command to create an image pull secret called `ibm-entitlement-key`.

   ```
   oc create secret docker-registry ibm-entitlement-key \
   --docker-server=cp.icr.io \
   --docker-username=cp \
   --docker-password=entitlement_key \
   --docker-email=user_email
   \--namespace=namespace
   ```

   Where *entitlement_key* is the entitlement key that you copied in step b, *user_email>* is the IBM ID associated with the entitled software, and *namespace* is the namespace that you installed your IBM MQ Operator into.

7. Optional: Install IBM Cloud Pak for Integration and its dependencies.

   There are some additional required components when you deploy at least one queue manager where **all** of the following statements are true:

   - You are using a IBM Cloud Pak for Integration license.
   - The IBM MQ Console is enabled.
   - You are using the IBM Cloud Pak for Integration Keycloak service for IBM MQ Console single sign-on (SSO) authentication and authorization (the default).

   If all the previous statements are true, then SSO is provided by Keycloak and you must complete the following additional steps:

   - Install the IBM Cloud Pak for Integration Operator.
   - Optional: Deploy the Platform UI.

     a. Create the `ibm-common-services` namespace. When logged into your OpenShift cluster through the CLI, run the following command:

     ```
     oc new-project ibm-common-services
     ```

     b. Deploy the Platform UI.

8. Deploy the License Service.

   This is required for monitoring license usage of queue managers. Follow the instructions in Deploying License Service.

9. Deploy a queue manager.

   For instructions on deploying an example "quick start" queue manager, see “Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97.

Related tasks

"Uninstalling the IBM MQ Operator on Red Hat OpenShift" on page 64
You can use the Red Hat OpenShift console or CLI to uninstall the IBM MQ Operator from Red Hat OpenShift.

OpenShift CP4I Upgrading the IBM MQ Operator and queue managers on Red Hat OpenShift

There are different upgrade processes for users of the IBM MQ Operator on Red Hat OpenShift, depending on whether you use IBM MQ licenses, or IBM Cloud Pak for Integration (CP4I) licenses. Complete the upgrade step for your deployment type.

Before you begin

Note: These instructions are for deployments of the IBM MQ Operator on Red Hat OpenShift. For deployments on Amazon EKS, see "Upgrading the IBM MQ Operator and queue managers on Amazon EKS" on page 77.

About this task

To upgrade your IBM MQ Operator and queue managers, complete one of the following steps:

Procedure

- Option 1: **Upgrade deployments to the latest version on your current Operator channel.**

  To upgrade deployments of the IBM MQ Operator on Red Hat OpenShift to the latest version on your current Operator channel, see "Upgrading to an IBM MQ Operator channel latest security release on Red Hat OpenShift" on page 55.
- Option 2: **Upgrade the IBM MQ Operator for IBM MQ licenses.**

  To upgrade deployments of the IBM MQ Operator on Red Hat OpenShift where **only** IBM MQ licenses are used, see "Upgrading the IBM MQ Operator on Red Hat OpenShift" on page 54.
- Option 3: **Upgrade the IBM MQ Operator for CP4I users.**

  Upgrade deployments of the IBM MQ Operator on Red Hat OpenShift for users of the IBM Cloud Pak for Integration. This includes if you have deployed at least one of your queue managers under a CP4I license. See "Upgrading the IBM MQ Operator for IBM Cloud Pak for Integration users on Red Hat OpenShift" on page 60.

OpenShift *Upgrading the IBM MQ Operator on Red Hat OpenShift*

Upgrade deployments of the IBM MQ Operator on Red Hat OpenShift where **only** IBM MQ licenses are used.

Before you begin

Important: This task is for users of the IBM MQ Operator on Red Hat OpenShift where **only** IBM MQ licenses are used.

- If you are a IBM Cloud Pak for Integration user, or you have deployed at least one of your queue managers using a CP4I license on Red Hat OpenShift, see "Upgrading the IBM MQ Operator for IBM Cloud Pak for Integration users on Red Hat OpenShift" on page 60.
- If you deployed the IBM MQ Operator on Amazon EKS, see "Upgrading the IBM MQ Operator and queue managers on Amazon EKS" on page 77.

About this task

Complete whichever of the following steps matches the upgrade that you need.

Note:

- Version 4.0.2 of the IBM MQ Operator is the latest CD release.
- Version 4.0.2 of the IBM MQ Operator is the latest SC2 release.

Procedure

- Option 1: “Upgrading to an IBM MQ Operator channel latest security release on Red Hat OpenShift” on page 55
- Option 2: “Upgrading to the 4.0.x CD or SC2 channel of IBM MQ Operator on Red Hat OpenShift” on page 57

Upgrading to an IBM MQ Operator channel latest security release on Red Hat OpenShift

Upgrading the IBM MQ Operator allows you to upgrade your queue managers.

Before you begin

Important: This task is for upgrading deployments of the IBM MQ Operator on Red Hat OpenShift to the latest Security Release on the deployment's Channel.

- If you are upgrading a deployment of the IBM MQ Operator on Red Hat OpenShift to an alternative Channel, see the alternative upgrade paths described in Upgrading the IBM MQ Operator and queue managers on Red Hat® OpenShift®.
- If you deployed the IBM MQ Operator on Amazon EKS, see “Upgrading the IBM MQ Operator and queue managers on Amazon EKS” on page 77.

About this task

There are different procedures, depending on the catalog source used to deploy the IBM MQ Operator that is being upgraded.

Option 1: Specific catalog source for the IBM MQ Operator

A new IBM MQ Operator version becomes available in an OpenShift cluster **only** after you update the catalog source. This process effectively gives you manual control of upgrades, so you do not need to use the **Manual** option for the **`Update approval`** setting for operators. The **Manual** option forces all possible upgrades to be done at the same time and can block upgrades, so use the **Automatic** option only.

To use this option, skip to Upgrade with the specific catalog source for the IBM MQ Operator.

Option 2: IBM Operator Catalog

With this option, new operator versions become available and are applied **without** any intervention from you. So, use this option **only** for online installations where you want **automatic** upgrades of the IBM MQ Operator, and where deterministic installations are not needed. This option can be useful for proof-of-concept environments but it is **not suitable for production environments**.

To use this option, skip to Upgrade with the IBM Operator Catalog.

To move from using the IBM Operator Catalog to using the specific catalog source for the IBM MQ Operator, which give you greater control over upgrades, see “Moving to the specific catalog source for the IBM MQ Operator” on page 58.

Procedure

- **Upgrade with the specific catalog source for the IBM MQ Operator**

  a) Mirror images (air-gap only).

If your cluster is in a restricted (air-gapped) network environment, you must mirror the IBM MQ images.

Important:

The IBM MQ CASE contains only a single queue manager version. The image mirroring process runs considerably faster, but might need to be run multiple times: once for each version of the queue manager that is being installed.

For more information, see “IBM MQ CASE for air-gap install and upgrade” on page 329.

You must look up which IBM MQ CASE versions contain the operator and queue manager versions you require, and repeat the air-gap mirror process for each. See the IBM MQ CASE Lookup Tables for the version mappings.

For example, when following the instructions linked below to mirror images, you install the latest operator and single corresponding queue manager version by using the following values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=4.0.2
```

For information about the contents of the IBM MQ CASE, which is used for air-gap installs and upgrades, see “IBM MQ CASE for air-gap install and upgrade” on page 329.

To create mirror images, see Mirroring images for an air-gapped cluster.

b) Apply the latest Catalog source.

Follow the instructions under "Add specific catalog sources for the IBM MQ Operator" in Adding the IBM MQ Operator catalog source.

c) Upgrade the IBM MQ Operator.

If the **Update approval** status for the IBM MQ Operator is set to **Automatic**, your operator upgrades. Otherwise, manually upgrade the IBM MQ Operator:

a. From the navigation pane, click **Operators** > **Installed Operators**.

All installed Operators in the specified project are displayed.

b. Select the **IBM MQ Operator**

c. Navigate to the **Subscription** tab

d. Click **Upgrade available**

e. Click **Preview InstallPlan**

f. Click **Approve** to complete the upgrade

The operator upgrades to the new version.

d) Upgrade any IBM MQ queue managers.

Follow the instructions in Upgrade IBM MQ queue managers.

- **Upgrade with the IBM Operator Catalog**

a) Upgrade the IBM MQ Operator to a newer version.

If you have automatic upgrades set, then upon the release of a new Security Release your IBM MQ Operator completes an upgrade. Otherwise, manually approve your IBM MQ Operator upgrade:

– If there is an upgrade available, the **`Upgrade Status`** might be "`Upgrade available`".

– In this case, there might be an available control that you can use to approve the **InstallPlan** that upgrades the IBM MQ Operator.

b) Upgrade any IBM MQ queue managers

Follow the instructions in Upgrade IBM MQ queue managers.

- **Upgrade IBM MQ queue managers**.

You should upgrade any IBM MQ queue managers to a newer version after upgrading the IBM MQ Operator. The following table describes the latest version of the IBM MQ queue manager for each active Operator channel. Using the relevant version, follow the procedure in "Upgrading an IBM MQ queue manager using Red Hat OpenShift" on page 63.

Operator channel	Latest IBM MQ queue manager
v4.0 (SC2)	10.0.0.0-r3
v4.0 (CD)	10.0.0.0-r3

Upgrading to the 4.0.x CD or SC2 channel of IBM MQ Operator on Red Hat OpenShift

Upgrade either a CD or SC2 deployment of the IBM MQ Operator where **only** IBM MQ licenses are used.

Before you begin

IBM MQ Operator 4.0.2 is a CD and SC2 release.

Important: Ensure that your system meets the following criteria:

- Your system is running on a supported version of Red Hat OpenShift.
  - To verify the compatible versions for each IBM MQ Operator channel, see the **Containers** tab of the system requirements page for the relevant version of IBM MQ. To upgrade the platform, see Upgrading Red Hat OpenShift in the IBM Cloud Pak for Integration documentation.
  - If your system is running on Amazon EKS, see "Upgrading the IBM MQ Operator and queue managers on Amazon EKS" on page 77 instead.
- You are using an IBM MQ license. If you are using IBM Cloud Pak for Integration (CP4I) or you deployed at least one of your queue managers by using a CP4I license on Red Hat OpenShift, see "Upgrading the IBM MQ Operator for IBM Cloud Pak for Integration users on Red Hat OpenShift" on page 60 instead.
- Your system is on the latest version of one of the following IBM MQ Operator channels:
  - 3.9.x (CD). The latest version is 3.9.2.
  - 3.2.x (SC2). The latest version is 3.2.27.

  If your system is not on one of these versions, update it as follows then return here to continue with the steps in this task:
  - If your operator is on one of the valid channels, follow the steps in "Upgrading to an IBM MQ Operator channel latest security release on Red Hat OpenShift" on page 55 to upgrade to the latest version of that channel.
  - If your operator is not on one of the valid channels, follow the steps in Upgrading the IBM MQ Operator on Red Hat OpenShift in the IBM MQ 9.4.x documentation to upgrade to one of the latest versions mentioned previously.

Procedure

1. If your cluster is in a restricted (air-gapped) network environment, mirror the IBM MQ images.

   Note: The IBM MQ CASE, which is used for air-gap installations and upgrades, contains one queue manager version. This structure speeds up the image mirroring process but requires you to run the process multiple times if you want to install more than one version of the queue manager.

   a) Look up the IBM MQ CASE versions that contain the operator and queue manager versions that you require, in IBM MQ CASE lookup lables.

   b) Mirror the images by following the instructions in the documentation for the version of IBM Cloud Pak for Integration that is relevant to your channel.

      You can omit the section titled "Configure the OpenShift cluster" because the connection to the image registry should exist from previous installations or upgrades:

- **3.9.x channel:** Mirroring images for an air-gapped cluster (IBM Cloud Pak for Integration 16.2.0)
- **3.2.x channel:** Mirroring images for an air-gapped cluster (IBM Cloud Pak for Integration 16.1.0)

For example, install the latest operator and single corresponding queue manager version by using the following values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=<LATEST_OPERATOR_IN_CHANNEL>
```

where <LATEST_OPERATOR_IN_CHANNEL> is the version that you determined from the IBM CASE lookup tables.

c) If you require multiple operator and queue manager versions, repeat these steps for each version.

2. Upgrade your IBM MQ Operator to 4.0.2 by following the instructions in "Upgrading the IBM MQ Operator using Red Hat OpenShift" on page 61.
3. Upgrade the instances.

   To receive the latest features and security fixes, upgrade the IBM MQ Operand (Queue Manager Container image) to version 10.0.0.0-r3, which is a CD and SC2 release. See "Upgrading an IBM MQ queue manager using Red Hat OpenShift" on page 63.

OpenShift *Moving to the specific catalog source for the IBM MQ Operator*

If you have an installation of the IBM MQ Operator from a previous release, and you are using the IBM Operator Catalog, applying the specific catalog source is the most effective way to fully control software versioning on a cluster.

Before you begin

Important: This task must be performed by a cluster administrator. See OpenShift Roles and permissions.

The following steps are completed using the CLI.

About this task

The IBM Operator Catalog is an index of operators available to extend the API of a Red Hat OpenShift Container Platform cluster to enable IBM software products.

This procedure moves an installation of the IBM MQ Operator from the IBM Operator Catalog so that you can use the specific catalog source for the IBM MQ Operator.

Procedure

1. Add the IBM MQ Operator Catalog.

   Follow the instructions under "Add specific catalog sources for the IBM MQ Operator" in Adding the IBM MQ Operator catalog source.
2. Confirm that the IBM MQ Operator catalog source was created in the `openshift-marketplace` namespace.

   Run the following command:

   ```
   oc get catalogsource -n openshift-marketplace
   ```

   Example output:

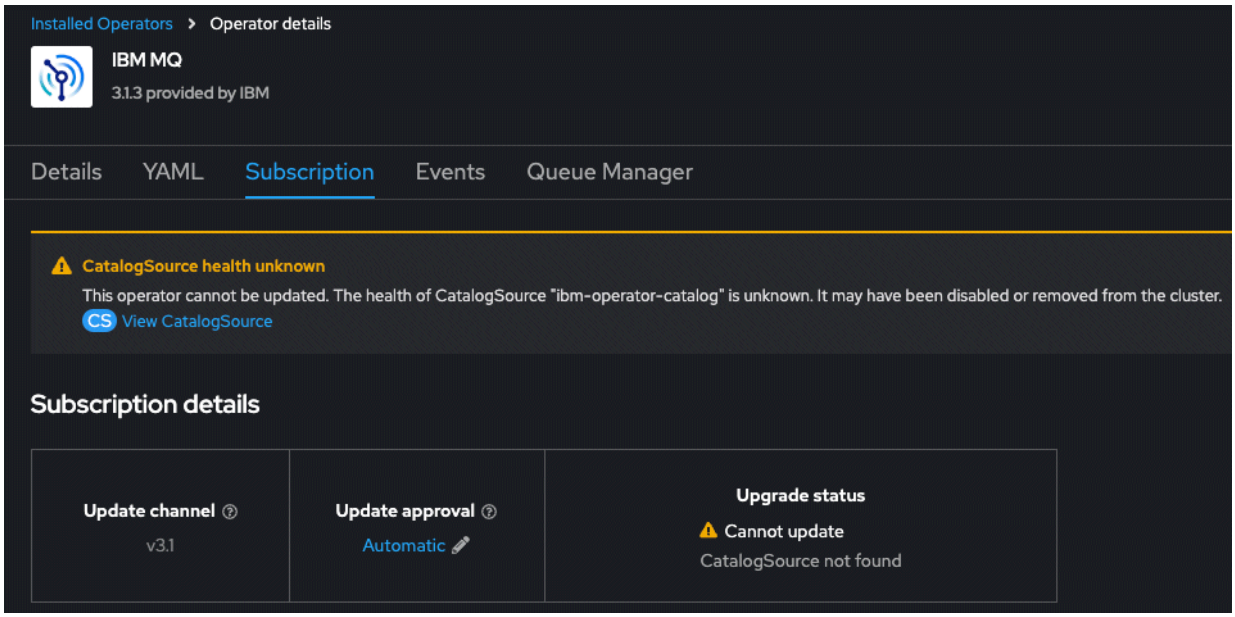
   ```
   oc get catalogsource -n openshift-marketplace

   NAME                          DISPLAY               TYPE   PUBLISHER   AGE
   ibm-operator-catalog          IBM Operator Catalog  grpc   IBM         23h
   ibmmq-operator-catalogsource  ibm-mq-3.1.3          grpc   IBM         23h
   ```
3. Optional: Delete the IBM Operator catalog source.

Warning: You should only complete this step if you are certain there are no other Operators using the IBM Operator Catalog.

Run the following command:

```
oc delete catalogsource ibm-operator-catalog -n openshift-marketplace
```

The IBM MQ Operator status changes to `CatalogSource not found`. This is expected.



4. Change the subscription of the IBM MQ Operator to point to the new specific IBM MQ Operator catalog source.
   a) Edit the subscription.

      Run the following command, replacing *OPERATOR-NAMESPACE* with either `openshift-operators` for cluster-wide installations of the IBM MQ Operator, or the specific namespace the IBM MQ Operator is deployed in:

      ```
      oc edit subscription ibm-mq -n OPERATOR-NAMESPACE
      ```

   b) Change the `spec.source` value from `ibm-operator-catalog` to the name of the catalog source created in step "1" on page 58.

      For example:

      ```
      spec:
        channel: v3.1
        installPlanApproval: Automatic
        name: ibm-mq
        source: ibm-operator-catalog # CHANGE --> ibmmq-operator-catalogsource
        sourceNamespace: openshift-marketplace
      ```

   c) Save the changes.

      The IBM MQ Operator installation now points at the IBM MQ Operator catalog source. If you deleted the IBM Operator Catalog, the status reverts from "CatalogSource not found" to "Succeeded".

Results

Your installation of the IBM MQ Operator now points to the specific catalog source for the IBM MQ Operator. This gives you full control over upgrades to the operator.

Upgrading the IBM MQ Operator for IBM Cloud Pak for Integration users on Red Hat OpenShift

Upgrade deployments of the IBM MQ Operator on Red Hat OpenShift where an IBM Cloud Pak for Integration license is used.

Before you begin

Important: Ensure that your system meets the following criteria:

- You are using IBM Cloud Pak for Integration (CP4I) or you deployed at least one of your queue managers by using a CP4I license. If you deployed on Red Hat OpenShift using an IBM MQ license, see "Upgrading the IBM MQ Operator on Red Hat OpenShift" on page 54 instead.
- Your system is running on a supported version of Red Hat OpenShift.
  - To verify the compatible versions for each IBM MQ Operator channel, see the **Containers** tab of the system requirements page for the relevant version of IBM MQ. To upgrade the platform, see Upgrading Red Hat OpenShift in the IBM Cloud Pak for Integration documentation.
  - If your system is running on Amazon EKS, see "Upgrading the IBM MQ Operator and queue managers on Amazon EKS" on page 77 instead.
- Your system is on the latest version of one of the following IBM MQ Operator channels:
  - 3.9.x (CD). The latest version is 3.9.2.
  - 3.2.x (SC2). The latest version is 3.2.27.

  If your system is not on one of these versions, update it as follows then return here to continue with the steps in this task:
  - If your operator is on one of the valid channels, follow the steps in "Upgrading to an IBM MQ Operator channel latest security release on Red Hat OpenShift" on page 55 to upgrade to the latest version of that channel.
  - If your operator is not on one of the valid channels, follow the steps in Upgrading the IBM MQ Operator on Red Hat OpenShift in the IBM MQ 9.4.x documentation to upgrade to one of the latest versions mentioned previously.

Procedure

1. If your cluster is in a restricted (air-gapped) network environment, mirror the IBM MQ images.

   Note: The IBM MQ CASE, which is used for air-gap installations and upgrades, contains one queue manager version. This structure speeds up the image mirroring process but requires you to run the process multiple times if you want to install more than one version of the queue manager.

   a) Look up the IBM MQ CASE versions that contain the operator and queue manager versions that you require, in IBM MQ CASE lookup lables.

   b) Mirror the images by following the instructions in the documentation for the version of IBM Cloud Pak for Integration that is relevant to your channel.

   You can omit the section titled "Configure the OpenShift cluster" because the connection to the image registry should exist from previous installations or upgrades:

   - **3.9.x channel:** Mirroring images for an air-gapped cluster (IBM Cloud Pak for Integration 16.2.0)
   - **3.2.x channel:** Mirroring images for an air-gapped cluster (IBM Cloud Pak for Integration 16.1.0)

   For example, install the latest operator and single corresponding queue manager version by using the following values:

   ```
   export OPERATOR_PACKAGE_NAME=ibm-mq
   export OPERATOR_VERSION=<LATEST_OPERATOR_IN_CHANNEL>
   ```

where <LATEST_OPERATOR_IN_CHANNEL> is the version that you determined from the IBM CASE lookup tables.

c) If you require multiple operator and queue manager versions, repeat these steps for each version.

2. Upgrade your installation to the latest IBM MQ Operator version, 4.0.2.

   Follow the steps in Upgrading from version 16.1.0 or 16.1.3.

   Note: Your version of IBM Cloud Pak for Integration should be as follows:

   - If your IBM MQ Operator version is 3.2.27, you should be on IBM Cloud Pak for Integration version 16.1.0.
   - If your IBM MQ Operator version is 3.9.2, you should be on IBM Cloud Pak for Integration version 16.1.3.

OpenShift CP4I *Upgrading the IBM MQ Operator using Red Hat OpenShift*

The IBM MQ Operator can be upgraded from the Red Hat OpenShift console or command line.

Before you begin

Note: To upgrade deployments of the IBM MQ Operator on Amazon EKS, see "Upgrading the IBM MQ Operator and queue managers on Amazon EKS" on page 77.

Note:

- The latest CD version of the IBM MQ Operator is 4.0.2.
- The latest SC2 version of the IBM MQ Operator is 4.0.2.

For the latest IBM MQ Operator release notes, see Release history for IBM MQ Operator.

Procedure

- **Option 1 : Upgrade the IBM MQ Operator using the Red Hat OpenShift console.**

  a) Log into your Red Hat OpenShift cluster console.

  b) Mirror images (air-gap only).

  If your cluster is in a restricted (air-gapped) network environment, you must mirror the IBM MQ images using the following values:

  ```
  export OPERATOR_PACKAGE_NAME=ibm-mq
  export OPERATOR_VERSION=4.0.2
  ```

  For information about the contents of the IBM MQ CASE, which is used for air-gap installs and upgrades, see "IBM MQ CASE for air-gap install and upgrade" on page 329.

  To create mirror images, see Mirroring images for an air-gapped cluster.

  c) Apply latest Catalog source.

  If you are using the specific catalog source for the IBM MQ Operator, rather than the `ibm-operator-catalog`, you must apply the catalog source for the new IBM MQ version.

  To move from using the IBM Operator Catalog to using the specific catalog source for the IBM MQ Operator, and gain greater control over upgrades, refer to the steps in "Moving to the specific catalog source for the IBM MQ Operator" on page 58 before returning to complete the next step.

  If you are using the IBM Operator catalog (some online installs only), proceed to the next step.

  Otherwise, follow the instructions in "Adding the IBM MQ Operator catalog source" on page 45 then proceed to the next step.

  d) Upgrade the IBM MQ Operator. New major/minor IBM MQ Operator versions are delivered through new Subscription Channels. To upgrade your Operator to a new major/minor version, you need to update the selected channel in your IBM MQ Operator Subscription.

a. From the navigation pane, click **Operators** > **Installed Operators**.

   All installed Operators in the specified project are displayed.

b. Select the **IBM MQ Operator**.

c. Navigate to the **Subscription** tab.

d. Click the **Channel**.

   The **Change Subscription Update Channel** window is displayed.

e. Select the desired channel, then click **Save**.

   The operator upgrades to the latest version that is available in the new channel. See "Release history for IBM MQ Operator" on page 5 to determine what this version will be.

- **Option 2 : Upgrade the IBM MQ Operator using the Red Hat OpenShift command line.**

  a) Log into your cluster using **`oc login`**.

  b) Mirror images (air-gap only).

  If your cluster is in a restricted (air-gapped) network environment, you must mirror the IBM MQ images using the following values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=4.0.2
```

  For information about the contents of the IBM MQ CASE, which is used for air-gap installs and upgrades, see "IBM MQ CASE for air-gap install and upgrade" on page 329.

  To create mirror images, see Mirroring images for an air-gapped cluster.

  c) Apply latest Catalog source.

  If you are using the specific catalog source for the IBM MQ Operator, rather than the `ibm-operator-catalog`, you must apply the catalog source for the new IBM MQ version.

  To move from using the IBM Operator Catalog to using the specific catalog source for the IBM MQ Operator, and gain greater control over upgrades, refer to the steps in "Moving to the specific catalog source for the IBM MQ Operator" on page 58 before returning to complete the next step.

  If you are using the IBM Operator catalog (some online installs only), proceed to the next step.

  Otherwise, follow the instructions in "Adding the IBM MQ Operator catalog source" on page 45 then proceed to the next step.

  d) Upgrade the IBM MQ Operator. New major or minor IBM MQ Operator versions are delivered through new subscription channels. To upgrade your Operator to a new major or minor version, complete the following steps to update the selected channel in your IBM MQ Operator Subscription.

  a. Ensure the required IBM MQ Operator Upgrade Channel is available.

```
oc get packagemanifest ibm-mq -o=jsonpath='{.status.channels[*].name}'
```

  b. Patch the `Subscription` to move to the desired update channel (where *vX.Y* is the desired update channel identified in the previous step.

```
oc patch subscription ibm-mq --patch '{"spec":{"channel":"vX.Y"}}' --type=merge
```

OpenShift CP4I *Upgrading an IBM MQ queue manager using Red Hat OpenShift*

An IBM MQ queue manager, deployed using the IBM MQ Operator, can be upgraded in Red Hat OpenShift by using any of the following interfaces: Operator Hub; Command Line; IBM Cloud Pak for Integration Platform UI.

Before you begin

Note: To upgrade an IBM MQ queue manager deployed by using the IBM MQ Operator on Amazon EKS, see "Upgrading IBM MQ queue managers on Amazon EKS" on page 82.

As part of the process to upgrade the IBM MQ queue managers, you might have been sent to this topic from the IBM Cloud Pak for Integration documentation.

Note:

- The latest CD version of the IBM MQ queue manager is 10.0.0.0-r3.
- The latest SC2 version of the IBM MQ queue manager is 10.0.0.0-r3.

For the latest IBM MQ queue manager release notes, see "Release history for queue manager container images" on page 7.

Ensure that your IBM MQ Operator is at the required version. To upgrade your IBM MQ Operator, see "Upgrading the IBM MQ Operator on Red Hat OpenShift" on page 54.

Before you can upgrade the queue manager in an air-gap environment, you must mirror the latest IBM Cloud Pak for Integration images through the air-gap specific steps that are completed during an upgrade of the IBM MQ Operator. For an example of these steps, see "Upgrading to the 4.0.x CD or SC2 channel of IBM MQ Operator on Red Hat OpenShift" on page 57.

About this task

This task offers three options for upgrading an IBM MQ queue manager that is deployed on OpenShift:

1. Upgrading an IBM MQ queue manager using the Red Hat OpenShift console.
2. Upgrading an IBM MQ queue manager using the Red Hat OpenShift CLI
3. Upgrading an IBM MQ queue manager in Red Hat OpenShift using the Platform UI

Procedure

- **Option 1**: Upgrading an IBM MQ queue manager using the Red Hat OpenShift console.
  a) Log in to your Red Hat OpenShift cluster web console.
  b) From the navigation pane, click **Operators** > **Installed Operators**.
  All installed Operators in the specified project are displayed.
  c) Select the **IBM MQ Operator**.
  The **IBM MQ Operator** window is displayed.
  d) Navigate to the **Queue Manager** tab.
  The **Queue Manager Details** window is displayed.
  e) Select the queue manager that you want to upgrade.
  f) Navigate to the YAML tab.
  g) Update the following fields, where necessary, to match the desired IBM MQ queue manager version upgrade.
  – spec.version
  – spec.license.licence
  See "Release history for queue manager container images" on page 7 for a mapping of IBM MQ Operator versions and IBM MQ queue manager container images.

h) Save the updated queue manager YAML.

- **Option 2**: Upgrading an IBM MQ queue manager using the Red Hat OpenShift CLI
  a) Log in to the Red Hat OpenShift command line interface (CLI) using `oc login`.
  b) Edit the **`QueueManager`** resource to update the following fields, where necessary, to match the desired IBM MQ queue manager version upgrade.
     – spec.version
     – spec.license.licence

     See "Release history for queue manager container images" on page 7 for a mapping of IBM MQ Operator versions and IBM MQ queue manager container images.

     Use the following command:

     ```
     oc edit queuemanager my_qmgr
     ```

     where *my_qmgr* is the name of the QueueManager resource that you want to upgrade.
- **Option 3**: Upgrading an IBM MQ queue manager in Red Hat OpenShift using the Platform UI.

  Follow the relevant instructions in Managing versions and upgrades by using the Platform UI in the IBM Cloud Pak for Integration documentation.

OpenShift CP4I Uninstalling the IBM MQ Operator on Red Hat OpenShift

You can use the Red Hat OpenShift console or CLI to uninstall the IBM MQ Operator from Red Hat OpenShift.

Before you begin

Note: To uninstall the IBM MQ Operator from Amazon EKS, see "Uninstalling the IBM MQ Operator on Amazon EKS" on page 83.

Procedure

- Option 1: Uninstall the IBM MQ Operator with the OpenShift console.

  Note: If the IBM MQ Operator is installed across all projects/namespaces on the cluster, repeat steps 2-6 of the following procedure for each project where you want to delete queue managers.

  a) Log in to the Red Hat OpenShift Container Platform web console with your Red Hat OpenShift Container Platform cluster admin credentials.
  b) Change **Project** to the namespace from which you want to uninstall the IBM MQ Operator. Select the namespace from the **Project** dropdown list.
  c) In the navigation pane, click **Operators** > **Installed Operators**.
  d) Click the **IBM MQ** operator.
  e) Click the **Queue Managers** tab, to view the queue managers managed by this IBM MQ Operator.
  f) Delete one or more queue managers.

     Note that, although these queue managers continue to run, they might not function as expected without an IBM MQ Operator.
  g) Optional: If appropriate, repeat steps 2-6 for each project where you want to delete queue managers.
  h) Return to **Operators** > **Installed Operators**.
  i) Next to the **IBM MQ** operator, click the three dots menu and select **Uninstall Operator**.
- Option 2: Uninstall the IBM MQ Operator with the OpenShift CLI
  a) Log in to your Red Hat OpenShift cluster using `oc login`.

b) If the IBM MQ Operator is installed in a single namespace, complete the following substeps:

a. Ensure you are in the project containing the IBM MQ Operator to be uninstalled:

```
oc project project_name
```

b. View the queue managers installed in the project:

```
oc get qmgr
```

c. Delete one or more queue managers:

```
oc delete qmgr qmgr_name
```

Note that, although these queue managers continue to run, they might not function as expected without an IBM MQ Operator.

d. View the **ClusterServiceVersion** instances:

```
oc get csv
```

e. Delete the IBM MQ **ClusterServiceVersion**:

```
oc delete csv ibm_mq_csv_name
```

f. View the subscriptions:

```
oc get subscription
```

g. Delete all subscriptions:

```
oc delete subscription ibm_mq_subscription_name
```

c) If the IBM MQ Operator is installed and available to all namespaces on the cluster, complete the following substeps:

a. View all the installed queue managers:

```
oc get qmgr -A
```

b. Delete one or more queue managers:

```
oc delete qmgr qmgr_name -n namespace_name
```

Note that, although these queue managers continue to run, they might not function as expected without an IBM MQ Operator.

c. View the **ClusterServiceVersion** instances:

```
oc get csv -A
```

d. Delete the IBM MQ **ClusterServiceVersion** from the cluster:

```
oc delete csv ibm_mq_csv_name -n openshift-operators
```

e. View the subscriptions:

```
oc get subscription -n openshift-operators
```

f. Delete the subscriptions:

```
oc delete subscription ibm_mq_subscription_name -n openshift-operators
```

Installing and upgrading the IBM MQ Operator on Amazon EKS

You perform a range of tasks to install, uninstall, and upgrade the IBM MQ Operator on Amazon EKS.

Before you begin

Note: For information about installing and upgrading on Red Hat OpenShift, see “Installing and upgrading the IBM MQ Operator on Red Hat OpenShift” on page 42.

About this task

The IBM MQ Operator is available on Amazon EKS. To get started with installing and upgrading the IBM MQ Operator on Amazon EKS, see the following topics.

Procedure

- “Installing the IBM MQ Operator on Amazon EKS” on page 66
- “Upgrading the IBM MQ Operator and queue managers on Amazon EKS” on page 77
- “Uninstalling the IBM MQ Operator on Amazon EKS” on page 83

Installing the IBM MQ Operator on Amazon EKS

You can install the IBM MQ Operator onto Amazon EKS by using a provided Helm chart.

Before you begin

Important: These instructions are for online installations of the IBM MQ Operator on Amazon EKS **only**. For installations in an air-gap environment on Amazon EKS, see “Installing the IBM MQ Operator on Amazon EKS in an air-gap environment” on page 68. For installations on Red Hat OpenShift, see “Installing the IBM MQ Operator on Red Hat OpenShift” on page 43.

To ensure that your installation goes as smoothly as possible, make sure that you understand all of the prerequisites and requirements before you start your installation. See “Planning for IBM MQ in containers” on page 9.

The IBM MQ Operator must be installed by a cluster administrator.

Login to your Amazon EKS cluster then ensure the following command line tools are installed:

- `kubectl`: the Kubernetes CLI tool
- Helm v3 or v4: a Kubernetes application management tool

If you intend to install IBM MQ queue managers that use persistent storage, ensure that a Storage Class is defined in your cluster.

About this task

For information about the versions of Amazon EKS that the IBM MQ Operator is supported on, see the **Containers** tab of the relevant IBM MQ system requirements page.

The provided Helm chart deploys an IBM MQ Operator Deployment into a namespace. The QueueManager CRD is deployed from this chart only if a version of it does not already exist in the cluster.

The IBM MQ Operator deploys by default with 1 CPU core and 1 GB memory.

Each IBM MQ queue manager defaults to 1 CPU core and 1 GB memory but can be run with fewer resources with lower performance

Procedure

1. Provide access to the Helm chart, by using one of the following options:

- **Option 1:** Use the IBM Helm chart repository
  - If you already added the IBM Helm chart to your local repository, run the following command to update it:

    ```
    helm repo update ibm-helm
    ```

  - If you did not add the IBM Helm chart to your local repository, run the following command:

    ```
    helm repo add ibm-helm https://raw.githubusercontent.com/IBM/charts/master/repo/ibm-helm
    ```

- **Option 2:** Download and extract the IBM MQ Operator Helm chart package.
  a. Go to the GitHub repo for IBM Helm charts, ibm-helm.
  b. Locate the Helm chart archive that has a name beginning with `ibm-mq-operator`. For example, `ibm-mq-operator-4.0.2.tgz`.
  c. Download the chart to your local machine.
  d. Extract the IBM MQ Operator Helm chart `.tgz` file then enter the directory. Run the following command, replacing `<IBM_MQ_OPERATOR_HELM_CHART>.tgz` with the file that you downloaded:

    ```
    tar -xvzf <IBM_MQ_OPERATOR_HELM_CHART>.tgz
    cd ibm-mq-operator/
    ```

2. Optional: If you want to override any of the default values for the IBM MQ Operator configuration properties, create an overrides file.

   The properties and their default values are described in "IBM MQ Operator Helm chart values" on page 332. The default values are set in the `values.yaml` file in the Helm chart package. To override the default values, create a new file called `overrides.yaml`, then add to that file the properties that you want to override, along with their new values. You will specify the `overrides.yaml` file when you install the IBM MQ Operator.

   For example, you could use the overrides file to set the install mode for the IBM MQ Operator to `AllNamespaces`:

   ```
   operator:
       installMode: "AllNamespaces"
   ```

   (The default `installMode` value is `OwnNamespace`, which means that the IBM MQ Operator operates at a single namespace scope. The `installMode AllNamespaces` means that the IBM MQ Operator operates at a cluster scope.)

3. Install the IBM MQ Operator.

   The command to use varies depending on how you accessed the Helm chart in step "1" on page 66. In the following commands, replace <NAMESPACE> with your desired namespace and omit the `--values overrides.yaml` parameter if you did not create an `overrides.yaml` file in the previous step:

   - Option 1: If you are using the IBM Helm chart repository:

     ```
     helm install ibm-mq-operator ibm-helm/ibm-mq-operator --namespace <NAMESPACE> --values overrides.yaml
     ```

   - Option 2: If you downloaded the IBM MQ Operator Helm chart:

     ```
     helm install ibm-mq-operator --namespace <NAMESPACE> --values overrides.yaml .
     ```

   Note: If you prefer not to use an override file, you can instead override default values directly from these commands, by replacing the `--values overrides.yaml` parameter with the `--set <FIELD>=<VALUE>` parameter.

4. Confirm the successful installation of the IBM MQ Operator:

```
helm list
kubectl get pods -n <NAMESPACE>
```

5. Deploy the IBM License Service.

   The IBM License Service is required for monitoring license usage of queue managers. Follow the procedure in “Preparing to use Kubernetes by installing the IBM License Service” on page 86.

6. Deploy a queue manager.

   For instructions for deploying an example "quick start" queue manager, see “Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS” on page 100.

Installing the IBM MQ Operator on Amazon EKS in an air-gap environment

If your Amazon EKS cluster is not directly accessible on the internet, or you wish to use your own image registry, you can mirror images to a registry in your network-restricted environment by using a bastion host, or a portable compute or storage device.

Procedure

1. Mirror the images by using one of the following approaches:
   - “Mirroring images with a bastion host on Amazon EKS” on page 71
   - “Mirroring images with a portable compute or storage device on on Amazon EKS” on page 73
2. Install the IBM MQ Operator in Amazon EKS.
   a) Log into the Amazon EKS cluster.
   b) If required, create a new namespace.
   c) Create a `docker-registry` type secret that contains credentials for the private registry, in the namespace in which you are installing the operator:

   ```
   kubectl create secret docker-registry <SECRET NAME> -n <NAMESPACE> --
   docker-server=<PRIVATE REGISTRY HOSTNAME> --docker-username=<USERNAME> --docker-
   password=<PASSWORD>
   ```

   Replace the parameter values as follows:

   - <SECRET NAME>: the name of the Kubernetes secret. Later steps use the example name `private-registry-credentials`.
   - <NAMESPACE>: the namespace in which you will deploy the IBM MQ Operator
   - <PRIVATE REGISTRY HOSTNAME>: the private registry hostname. For example, `my-private-registry.com`
   - <USERNAME>: the username to access your private registry
   - <PASSWORD>: the password to access your private registry

   d) Prepare the IBM MQ Operator Helm chart.

   When you mirrored images, you downloaded the IBM MQ CASE. The command that you used included the `--install-method helm` parameter, which results in the IBM MQ Helm chart being included in the CASE. The Helm chart is in a `.tgz` file at the following location:

   ```
   ~/.ibm-pak/data/cases/ibm-mq/$OPERATOR_VERSION/charts/ibm-mq-operator-
   $OPERATOR_VERSION.tgz
   ```

   Note: If the Helm chart was not downloaded with the CASE, you can download the `ibm-mq-operator` chart from https://github.com/IBM/charts/tree/master/repo/ibm-helm.

   Extract the `.tgz` file then enter the directory. For example, run the following command, changing <PATH> to the directory in which to extract the `.tgz` file (this directory must already exist):

   ```
   tar -xvzf ~/.ibm-pak/data/cases/ibm-mq/$OPERATOR_VERSION/charts/ibm-mq-operator-
   $OPERATOR_VERSION.tgz -C <PATH>
   cd <PATH>/ibm-mq-operator
   ```

e) Create an `overrides.yaml` file to override some of the values in the IBM MQ Operator Helm chart.

When you install the IBM MQ Operator in an air-gap environment, you must override some of the values in the IBM MQ Operator Helm chart. Create a file called `overrides.yaml`, then add the values to override to the file. (An alternative approach is to specify the values in the the `--set` parameter of the install command but that approach is not documented here.)

For a list of values in the IBM MQ Operator Helm chart that you can override, see "IBM MQ Operator Helm chart values" on page 332.

Important: `operator.imagePullSecrets` is used only to provide credentials for pulling the IBM MQ Operator image. Later, when you install queue managers, each QueueManager YAML must reference a secret (under `spec.imagePullSecrets`) that contains the private registry credentials.

For an air-gap installation, you must override at least the following values:

- `privateRegistry` (REQUIRED): set to $TARGET_REGISTRY (the private registry hostname), with optional port and optional repository prefix. You set $TARGET_REGISTRY when you mirrored the image in step "1" on page 68.
- `operator.imagePullSecrets`: a Kubernetes secret that contains credentials for the private registry. This value is used only for pulling the IBM MQ Operator image. This value is required if the private registry requires credentials.

If you need greater control, such as overriding the repository paths for each image, see the `operator.deployment.repository` and `operand.repositories` values in "IBM MQ Operator Helm chart values" on page 332.

For example, if you want to override only the `privateRegistry` and `operator.imagePullSecrets` values, and your private registry requires credentials that are in a secret called `private-registry-credentials`, you can use the following command to create the override file:

```
cat << EOF > overrides.yaml
privateRegistry: $TARGET_REGISTRY
operator:
    imagePullSecrets:
    - private-registry-credentials
EOF
```

f) Install the IBM MQ Operator Helm chart.

Run the following command, changing <NAMESPACE> to the namespace in which you are installing the operator. This command also specifies the override file that you created earlier.

```
helm install ibm-mq-operator --namespace <NAMESPACE> --values overrides.yaml .
```

g) Confirm the successful installation of the IBM MQ Operator.

Run the following command, changing the namespace as before:

```
helm list
kubectl get pods -n <NAMESPACE>
```

h) Deploy the IBM License Service.

The IBM License Service is required for monitoring the license usage of queue managers. For instructions for air-gap installations, see Installing IBM License Service using ibm-pak plug-in.

3. Deploy an IBM MQ queue manager.

You can now use the IBM MQ Operator to deploy an IBM MQ queue manager.

The following steps include commands for deploying an example queue manager. For more information about deploying queue managers in Amazon EKS see "Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS" on page 100.

Important: For an air-gap installation, the custom resource YAML for each queue manager must include `spec.imagePullSecrets` if the private registry that contains the images requires credentials. This field must reference a Kubernetes secret that contains the private registry credentials. For example:

```
spec:
    imagePullSecrets:
        - name: <KUBERNETES_SECRET_NAME>
```

If you are installing the queue manager in the same namespace as the IBM MQ Operator, you can reuse the same secret. Alternatively, repeat the steps above from step "2.c" on page 68 to create a `docker-registry` type secret in the namespace where you want to install the queue manager.

a) Create a YAML file called `mq-quickstart-airgap.yaml`, for the queue manager.

The following example YAML assumes the Kubernetes secret that contains the private registry credentials is called `private-registry-credentials`.

Important: If you accept the license agreement, change `accept: false` to `accept: true`. See "mq.ibm.com/v1beta1: Current license versions" on page 281 for details on the license.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
    name: quickstart
    annotations:
        com.ibm.mq/write-defaults-spec: 'false'
spec:
    version: 10.0.0.0-r3
    imagePullSecrets:
        - name: "private-registry-credentials"
    license:
        accept: false
        license: L-CXDB-XTZ9J8
    queueManager:
        name: "QUICKSTART"
        storage:
            queueManager:
                type: ephemeral
        route:
            enabled: false
        metrics:
            serviceMonitor:
                enabled: false
    web:
        enabled: false
        route:
            enabled: false
```

Note: `spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

b) Deploy the queue manager.

Run the following command, changing <NAMESPACE> to the namespace in which you are deploying the queue manager:

```
kubectl apply -f mq-quickstart-airgap.yaml -n <NAMESPACE>
```

c) Verify the queue manager status.

Run the following command, changing <NAMESPACE> as before:

```
kubectl get queuemanager quickstart -n <NAMESPACE>
```

Mirroring images with a bastion host on Amazon EKS

If your Amazon EKS cluster is not directly accessible on the internet, or you wish to use your own image registry, you can mirror images to a registry in your network-restricted environment by using a bastion host.

Before you begin

You must be able to connect your bastion host to the internet and to the restricted network environment (with access to the Amazon EKS cluster and the private registry) at the same time. Your host must be on a Linux x86_64 or macOS platform with any operating system that the following CLI tools support.

The following tools must exist on the bastion host:

- `kubectl`: the Kubernetes CLI tool
- `ibm-pak`: the IBM Catalog Management Plug-in for IBM Cloud Paks. Follow these instructions to download the plugin and make it available to `kubectl`.
  1. Download the IBM Catalog Management plug-in **version 1.18.0 or later** from GitHub.
  2. Extract the file then make it available as a `kubectl` plugin. For example, on macOS:

     ```
     tar -xvf oc-ibm_pak-darwin-amd64.tar.gz
     cp oc-ibm_pak-darwin-amd64 /usr/local/bin/kubectl-ibm_pak
     ```

     This plug-in allows you to run `kubectl ibm-pak` commands.
  3. To confirm that `ibm-pak` is installed, run the following command:

     ```
     kubectl ibm-pak --help
     ```
- `helm` v3 or v4: a Kubernetes application management tool
- `skopeo`: a command line utility that performs various operations on container images and image repositories

Note: Where possible, the air-gap process for Amazon EKS is the same as the process for Red Hat OpenShift Container Platform. Therefore, additional information is available in Mirroring images with a bastion host (a Red Hat OpenShift Container Platform task) in the IBM Cloud Pak for Integration documentation.

About this task

A bastion host is a device that has access to both the public internet and the network-restricted environment where a private registry and Amazon EKS clusters reside. Using the bastion host, you can mirror your images directly to the private registry. The process of mirroring images pulls the image from the IBM Container Registry then pushes it to your private registry.

Note: These instructions are for bastion host air-gapped deployments of the IBM MQ Operator on Amazon EKS. If your deployment is different, see the relevant task:

- If you are completing a portable device air-gapped deployment on Amazon EKS, see “Mirroring images with a portable compute or storage device on on Amazon EKS” on page 73.
- If you are completing an online deployment on Amazon EKS, see “Installing the IBM MQ Operator on Amazon EKS” on page 66 or “Upgrading the IBM MQ Operator on Amazon EKS” on page 77.
- If you are deploying on the OpenShift Container Platform, see “Installing and upgrading the IBM MQ Operator on Red Hat OpenShift” on page 42.

Procedure

1. Look up the IBM MQ CASE version that contains the operator and queue manager version that you require.

If you intend to install, or upgrade to, more than one queue manager version, you must repeat the air-gap image mirror process for each version. For version mappings, see the "IBM MQ CASE Lookup Tables" on page 330.

2. Set environment variables on your host. To install or upgrade to the latest operator and queue manager, set the following environment variables:

```
export CASE_NAME=ibm-mq
export OPERATOR_VERSION=4.0.2
export TARGET_REGISTRY="<YOUR PRIVATE REGISTRY LOCATION>"
```

   Set <YOUR PRIVATE REGISTRY LOCATION> to the image registry hostname (with optional port) where you will store IBM MQ images. For example, `my-private-registry.com` or `my-private-registry.com:5000`. If you intend to use an additional, common repository prefix for all IBM MQ mirrored images, specify the prefix too. For example, `my-private-registry.com:5000/my-prefix`. If you use this example to mirror an IBM MQ Operator image, the image is available at `my-private-registry.com:5000/my-prefix/cpopen/ibm-mq-operator`.

3. Connect your host to the internet.

   The host does not yet need to be connected to the network-restricted environment.

4. Download the CASE to your host by running the following command:

```
kubectl ibm-pak get $CASE_NAME --version $OPERATOR_VERSION --install-method helm
```

   The CASE now exists in `~/.ibm-pak/data/cases/$CASE_NAME/$OPERATOR_VERSION`.

5. Generate the mirror manifests by running the following command:

```
kubectl ibm-pak generate mirror-manifests $CASE_NAME --version $OPERATOR_VERSION
$TARGET_REGISTRY --install-method helm
```

   The command generates files in `~/.ibm-pak/data/mirror/$CASE_NAME/$OPERATOR_VERSION`. The `images-mapping.txt` file provides the mapping for each image, in the following format: <public location>=<private registry location>

6. Authenticate with the `cp.icr.io` registry.

   For example:

```
skopeo login cp.icr.io
```

   When you are prompted, enter the following details:

   - Username: `cp`
   - Password: <IBM ENTITLEMENT KEY>

   To get your entitlement key, go to the Container software library then click **Copy** for any key in the list.

7. Connect your host to the network-restricted environment.
8. Authenticate with the private registry.

   For example:

```
skopeo login <PRIVATE REGISTRY>
```

   When you are prompted, enter the following details:

   - Username: <PRIVATE REGISTRY USERNAME>
   - Password: <PRIVATE REGISTRY PASSWORD>

9. Mirror the images to your private registry.

Copy the images from the IBM Container Registry to the private registry. For example, the following command parses the `images-mapping.txt` file then copies each image to the private registry by using skopeo:

```
cat ~/.ibm-pak/data/mirror/$CASE_NAME/$OPERATOR_VERSION/images-mapping.txt | awk -F'='
'{ print "skopeo copy --preserve-digests --all docker://"$1" docker://"$2 }' | xargs -I {}
sh -c 'echo {}; {}'
```

Notes:

- If the private registry uses a self signed certificate, add the `--tls-verify=false` option to the command.
- If you are running the command on macOS and you encounter the error `xargs: command line cannot be assembled, too long`, add `-S1024` to the `xargs` part of the command.

The images now exist in your private registry.

What to do next

Continue with one of the following procedures:

- Install the IBM MQ Operator on Amazon EKS by following the instructions in "Installing the IBM MQ Operator on Amazon EKS in an air-gap environment" on page 68
- Upgrade the IBM MQ Operator on Amazon EKS by following the instructions in "Upgrading the IBM MQ Operator on Amazon EKS in an air-gap environment" on page 80.

Mirroring images with a portable compute or storage device on on Amazon EKS

If your Amazon EKS cluster is not directly accessible on the internet, or you wish to use your own image registry, you can mirror images to a registry in your network-restricted environment by using a portable compute or storage device, such as a laptop or USB storage device.

Before you begin

You must be able to connect your device to the internet, and later to the restricted network environment (with access to the Amazon EKS cluster and the private registry). You should use a Linux x86_64 or macOS platform with any operating system that the following CLI tools support.

The following tools must exist on the device:

- `kubectl`: the Kubernetes CLI tool
- `ibm-pak`: the IBM Catalog Management Plug-in for IBM Cloud Paks. Follow these instructions to download the plugin and make it available to `kubectl`.

  1. Download the IBM Catalog Management plug-in **version 1.18.0 or later** from GitHub.
  2. Extract the file then make it available as a `kubectl` plugin. For example, on macOS:

     ```
     tar -xvf oc-ibm_pak-darwin-amd64.tar.gz
     cp oc-ibm_pak-darwin-amd64 /usr/local/bin/kubectl-ibm_pak
     ```

     This plug-in allows you to run `kubectl ibm-pak` commands.
  3. To confirm that `ibm-pak` is installed, run the following command:

     ```
     kubectl ibm-pak --help
     ```

- `helm` v3 or v4: a Kubernetes application management tool
- `skopeo`: a command line utility that performs various operations on container images and image repositories

Note: Where possible, the air-gap process for Amazon EKS is the same as the process for Red Hat OpenShift Container Platform. Therefore, additional information is available in Mirroring images with a bastion host (a Red Hat OpenShift Container Platform task) in the IBM Cloud Pak for Integration documentation.

About this task

You can download images from the IBM Container Registry to a local file system on the device. You can then bring the device into the network-restricted environment, or transfer the files to a device within the network-restricted environment, to mirror the images to the private registry.

Note: These instructions are for portable device air-gapped deployments of the IBM MQ Operator on Amazon EKS.

- If you are completing a bastion host air-gapped deployment on Amazon EKS, see "Mirroring images with a bastion host on Amazon EKS" on page 71.
- If you are completing an online deployment on Amazon EKS, see "Installing the IBM MQ Operator on Amazon EKS" on page 66 or "Upgrading the IBM MQ Operator on Amazon EKS" on page 77.
- If you are deploying on the OpenShift Container Platform, see "Installing and upgrading the IBM MQ Operator on Red Hat OpenShift" on page 42.

Procedure

1. Download images from the IBM Container Registry.
   a) Look up the IBM MQ CASE version that contains the operator and queue manager version that you require.

      If you intend to install, or upgrade to, more than one queue manager version, you must repeat the air-gap image mirror process for each version. For version mappings, see the "IBM MQ CASE Lookup Tables" on page 330.

   b) To install, or upgrade to, the latest operator and Queue Manager, set the following environment variables on your device.

      ```
      export CASE_NAME=ibm-mq
      export OPERATOR_VERSION=4.0.2
      export TARGET_REGISTRY="<YOUR PRIVATE REGISTRY LOCATION>"
      export IMAGE_PATH=<YOUR LOCAL IMAGE PATH>
      ```

      Set the following values:

      - `<YOUR PRIVATE REGISTRY LOCATION>`: the image registry hostname (with optional port) where you will store IBM MQ images. For example, `my-private-registry.com` or `my-private-registry.com:5000`. If you intend to use an additional, common repository prefix for all IBM MQ mirrored images, specify the prefix too. For example, `my-private-registry.com:5000/my-prefix`. If you use this example to mirror an IBM MQ Operator image, the image is available at `my-private-registry.com:5000/my-prefix/cpopen/ibm-mq-operator`.
      - `<YOUR LOCAL IMAGE PATH>`: the local directory to store the images. For example, a directory in the current working directory called `images`.

   c) Connect your device to the internet. The device does not yet need to be connected to the network-restricted environment.

   d) Download the CASE to your device by running the following command:

      ```
      kubectl ibm-pak get $CASE_NAME --version $OPERATOR_VERSION --install-method helm
      ```

      The CASE now exists in `~/.ibm-pak/data/cases/$CASE_NAME/$OPERATOR_VERSION`

   e) Generate the mirror manifests by running the following command:

      ```
      kubectl ibm-pak generate mirror-manifests $CASE_NAME file://$IMAGE_PATH --version
      $OPERATOR_VERSION --final-registry $TARGET_REGISTRY --install-method helm
      ```

      The command generates files in `~/.ibm-pak/data/mirror/$CASE_NAME/$OPERATOR_VERSION`. The following files provide mapping in these formats:

- `images-mapping-to-filesystem.txt`: maps from `<IBM Container Registry>=<file system location>`
- `images-mapping-from-filesystem.txt`: maps from `<file system location>=<private registry location>`

f) Authenticate with the `cp.icr.io` registry.

For example:

```
skopeo login cp.icr.io
```

When you are prompted, enter the following details:

- Username: `cp`
- Password: `<IBM ENTITLEMENT KEY>`

To get your entitlement key, go to the Container software library then click **Copy** for any key in the list.

g) Copy the images to the file system.

Process the `images-mapping-to-filesystem.txt` file and copy the images to the file system. For example:

i) Generate a script that contains commands to copy the images from the IBM Container Registry to the file system:

```
mkdir -p $IMAGE_PATH/cp $IMAGE_PATH/cpopen $IMAGE_PATH/ibm-messaging

cat ~/.ibm-pak/data/mirror/$CASE_NAME/$OPERATOR_VERSION/images-mapping-to-
filesystem.txt | awk -F'=file://' '{ print "skopeo copy --preserve-digests --all
docker://"$1" dir:"$2 }' > copy_images_to_filesystem.sh
```

ii) Inspect the `copy_images_to_filesystem.sh` script to validate the commands to be run.

iii) Run the commands to copy the images to the file system:

```
chmod +x ./copy_images_to_filesystem.sh

./copy_images_to_filesystem.sh
```

The images are now on the file system, in `$IMAGE_PATH`.

2. Mirror the images to the private registry.

a) If you are using a portable computer device such as a laptop, disconnect the device from the internet then connect it to the restricted network environment.

b) If you are using portable storage (such as USB storage) to transfer the files to a device within the network-restricted environment, complete the following steps to prepare the new device:

- Transfer the following directories and their contents to the device in the restricted network environment:
  - The `~/.ibm-pak` directory.
  - The image path where you stored the images on the filesystem (`$IMAGE_PATH`) .

  Important: You must create these directories with the same path as on the original device.
- Set the same environment variables on the new device as on the original device. For example:

```
export CASE_NAME=ibm-mq
export OPERATOR_VERSION=4.0.2
export TARGET_REGISTRY="<YOUR PRIVATE REGISTRY LOCATION>"
export IMAGE_PATH=<YOUR LOCAL IMAGE PATH>
```

- Connect the new device to the network-restricted environment.

c) Verify the image digests on the file system.

The IBM MQ Operator depends on image digests remaining the same, so check that there were no accidental changes. For example:

i) Process the `images-mapping-from-filesystem.txt` file to extract the expected digests:

```
cat ~/.ibm-pak/data/mirror/$CASE_NAME/$OPERATOR_VERSION/images-mapping-from-
filesystem.txt | awk -F'[:=@]' '{ sub("file://", ":"); print $4 " " $2 ":" $6 "/
manifest.json"}' > checksums.txt
```

ii) Verify the digests on the filesystem:

```
sha256sum -c checksums.txt
```

This command validates the digests for every manifest list. If all lists have the expected digest, each line ends with `: OK`.

d) Authenticate with the private registry.

For example:

```
skopeo login <PRIVATE REGISTRY>
```

When prompted, enter the following details:

- username: `<PRIVATE REGISTRY USERNAME>`
- password: `<PRIVATE REGISTRY PASSWORD>`

e) Mirror the images to your private registry.

i) Generate the commands to copy the images to the private registry, in a script called `copy_images_from_filesystem.sh`:

```
cat ~/.ibm-pak/data/mirror/$CASE_NAME/$OPERATOR_VERSION/images-mapping-from-
filesystem.txt | awk -F'[:=@]' '{ sub("file://", "dir:"); print "skopeo copy --
preserve-digests --all  " $1 ":" $2 ":" $6 " docker://" $5 ":" $6 }' >
copy_images_from_filesystem.sh
```

Note: If the private registry uses a self signed certificate, add the `--tls-verify=false` to the `skopeo copy` part of the command.

ii) Inspect the script to validate the commands to be run.

iii) Run the commands to copy the images to the private registry

```
chmod +x ./copy_images_from_filesystem.sh

./copy_images_from_filesystem.sh
```

Important: Check the output of the `skopeo copy` commands for any errors. The `--preserve-digests` parameter in the `skopeo copy` commands ensures that images are copied only if the manifests of the architecture-specific images matches those listed in the manifest list. Otherwise, an error is reported.

f) Verify image digests in the private registry.

To ensure that there were no changes when copying into the private registry, validate that the digests for the images in the private registry remain the same. For example, the following command compares the expected digest with the digest for each image in the private registry:

```
cat ~/.ibm-pak/data/mirror/$CASE_NAME/$OPERATOR_VERSION/images-mapping-from-
filesystem.txt | awk -F'[:=@]' '{ sub("file://", "dir:"); print " printf "$5":"$6" && if
[ \"sha256:"$4"\" = \"$(skopeo manifest-digest <( skopeo inspect --raw docker://" $5 ":"
$6 "))\" ]; then echo \\\": OK\\\"; else printf \\\" FAIL: Digest does not match "$4"\\n\\
\"; fi" }' | xargs -I {} bash -c '{}'
```

Note:

- If the private registry uses a self signed certificate, add the `--tls-verify=false` option to `skopeo inspect` part of the command.

- If you run the command on macOS and you encounter the error `xargs: command line cannot be assembled, too long`, add `-S1024` to the `xargs` part of the command.

This command validates the digests for every manifest list. If all lists have the expected digest, each line ends with `: OK`.

What to do next

Continue with one of the following procedures:

- Install the IBM MQ Operator on Amazon EKS by following the instructions in "Installing the IBM MQ Operator on Amazon EKS in an air-gap environment" on page 68
- Upgrade the IBM MQ Operator on Amazon EKS by following the instructions in "Upgrading the IBM MQ Operator on Amazon EKS in an air-gap environment" on page 80.

Upgrading the IBM MQ Operator and queue managers on Amazon EKS

Upgrade the IBM MQ Operator and queue managers deployed on Amazon EKS.

Before you begin

Note: These instructions are for deployments of the IBM MQ Operator on Amazon EKS. For deployments on Red Hat OpenShift, see "Upgrading the IBM MQ Operator and queue managers on Red Hat OpenShift" on page 54.

About this task

To upgrade your IBM MQ Operator and queue managers, complete one of the following steps:

Procedure

Upgrading the IBM MQ Operator on Amazon EKS

Upgrade deployments of the IBM MQ Operator on Amazon EKS.

Before you begin

Important: This task is for users of the IBM MQ Operator on Amazon EKS. If you have deployed the IBM MQ Operator on Red Hat OpenShift, see "Upgrading the IBM MQ Operator and queue managers on Red Hat OpenShift" on page 54.

About this task

Complete whichever of the following steps matches the upgrade that you need.

Note: Version 4.0.2 of the IBM MQ Operator is the latest CD release.

Procedure

1. Option 1: Upgrade deployments of the IBM MQ Operator on Amazon EKS clusters with access to the internet.

   For instructions, see "Upgrading the IBM MQ Operator on Amazon EKS for online installations" on page 78.
2. Option 2: Upgrade deployments of the IBM MQ Operator on Amazon EKS clusters that are not able to directly access the internet, or where you wish to use your own image registry.

   For instructions, see "Upgrading the IBM MQ Operator on Amazon EKS in an air-gap environment" on page 80.

Upgrading the IBM MQ Operator on Amazon EKS for online installations
Upgrading the IBM MQ Operator allows you to upgrade your queue managers.

Before you begin

Important: This topic is for upgrading deployments of the IBM MQ Operator on Amazon EKS clusters *with* access to the internet. If your Amazon EKS cluster is not directly accessible on the internet, or where you wish to use your own image registry, follow the instructions in "Upgrading the IBM MQ Operator on Amazon EKS in an air-gap environment" on page 80 instead. If you deployed the IBM MQ Operator on Red Hat OpenShift, see "Upgrading the IBM MQ Operator and queue managers on Red Hat OpenShift" on page 54 instead.

Note: The latest CD version of the IBM MQ Operator is 4.0.2. For the latest IBM MQ Operator release notes, see "Release history for IBM MQ Operator" on page 5.

Before upgrading the IBM MQ Operator, complete the following steps:

- Log into the Amazon EKS cluster.
- Ensure that the installed IBM MQ Operator, and any queue managers, are running.
- If you installed multiple IBM MQ Operators on the cluster, ensure that you are upgrading to a version that is greater than, or equal to, the latest installed IBM MQ Operator. (This step ensures that you do not downgrade the Custom Resource Definition (CRD).)
- Before updating the CRD, validate that a newer version of the CRD was not previously installed on the cluster.

Procedure

1. Provide access to the latest Helm chart by using one of the following options:
   - **Option 1:** Use the IBM Helm chart repository
     - If you already added the IBM Helm chart to your local repository, run the following command to update it:

       ```
       helm repo update ibm-helm
       ```

     - If you did not add the IBM Helm chart to your local repository, run the following command:

       ```
       helm repo add ibm-helm https://raw.githubusercontent.com/IBM/charts/master/repo/ibm-helm
       ```

   - **Option 2:** Download and extract the IBM MQ Operator Helm chart package.
     a. Go to the GitHub repo for IBM Helm charts, ibm-helm.
     b. Locate the Helm chart archive for the IBM MQ Operator that you want to upgrade to. The name begins with `ibm-mq-operator`, for example, `ibm-mq-operator-4.0.2.tgz`.
     c. Download the chart to your local machine.
     d. Extract the IBM MQ Operator Helm chart `.tgz` file then enter the directory. Run the following command, replacing *<operator_helm_chart>* with the file that you downloaded:

        ```
        tar -xvzf <operator_helm_chart>.tgz
        cd ibm-mq-operator/
        ```

2. Upgrade the CRD.

   When you upgrade the IBM MQ Operator, Helm does not deploy the new `QueueManager` CRD from the Helm chart because a `QueueManager` CRD already exists in the cluster. You must therefore manually update the CRD. Use the same option that you chose in step 1.

   - **Option 1:** If you are using the IBM Helm chart repository, complete the following steps:
     a. Extract the CRD that is associated with the IBM MQ Operator version that you want to upgrade to. Run the following command, replacing *<upgrade_version>* with the IBM MQ Operator

version that you want to upgrade to (for example, 4.0.2) and *<crd_yaml_file>* with the name of a YAML file in which to save the CRD (for example, /myfolder/qm-crd.yaml):

```
helm show crds ibm-helm/ibm-mq-operator --version <upgrade_version> > <crd_yaml_file>
```

b. Check the contents of the *<crd_yaml_file>* file to ensure that the CRD was saved to it correctly.

c. Upgrade the CRD by running the following command. The `--force-conflicts` parameter is required only the first time that you upgrade the CRD.

```
kubectl apply --server-side --force-conflicts -f <crd_yaml_file>
```

- **Option 2:** If you downloaded the IBM MQ Operator Helm chart, apply the `QueueManager` CRD from the new chart by running the following command. The `--force-conflicts` parameter is required only the first time that you upgrade the CRD.

```
kubectl apply --server-side --force-conflicts -f crds/queuemanagers.yaml
```

3. Upgrade the IBM MQ Operator.

Important: If you specified any non-default Helm chart values when you previously installed or upgraded the IBM MQ Operator, you must specify the same values again otherwise both the queue managers and the IBM MQ Operator are likely to experience downtime. Default values are documented in "IBM MQ Operator Helm chart values" on page 332. You might have previously set values by using one of the following methods when you installed or upgraded the IBM MQ Operator:

- Specifying `--set <field>=<value>` parameters on the `helm install` or `helm upgrade` command
- Specifying the `--values <filename>.yaml` parameter of the `helm install` or `helm upgrade` command
- Modifying the original `values.yaml` (this method is not recommended)

a) View the IBM MQ Operator values that you used previously:

```
helm get values ibm-mq-operator
```

b) If you have the `overrides.yaml` file that you used previously, verify that the values in the file match the values that you obtained in the previous step.

c) If you don't have the `overrides.yaml` file that you used previously, re-create it by extracting the current values from the deployed operator. Run the following command, replacing *<namespace>* with the namespace in which the IBM MQ Operator is installed:

```
helm get values ibm-mq-operator -n <namespace> -o yaml > overrides.yaml
```

Review the `overrides.yaml` file to ensure it is as required.

d) Upgrade the IBM MQ Operator Helm chart, specifying the overrides file.

Use the same option that you chose in step 1.

Important: The `--reset-values` parameter ensures that Helm first uses the values from the chart that you are upgrading to, then applies the overrides file. The `helm upgrade` command has an alternative flag: `--reset-then-reuse-values`. The flag's documented behavior is *reset the values to the ones built into the chart, apply the last release's values and merge in any overrides from the command line via --set and -f.*. This flag might reduce the likelihood of accidentally omitting previously specified values. However, when you use the `--reset-then-reuse-values` parameter, the final state after the upgrade depends on the initial state, so can vary for each upgrade, or for each environment.

- **Option 1:** If you are using the IBM Helm chart repository, run the following command, replacing *<namespace>* with the namespace in which the IBM MQ Operator is installed and

<upgrade_version> with the version of the IBM MQ Operator that you want to upgrade to (for example, 4.0.2):

```
helm upgrade ibm-mq-operator ibm-helm/ibm-mq-operator --version <upgrade_version> --
namespace <namespace> --reset-values --values overrides.yaml
```

- **Option 2:** If you downloaded the IBM MQ Operator Helm chart, run the following command, replacing *<namespace>* with the namespace in which the IBM MQ Operator is installed:

```
helm upgrade ibm-mq-operator . --namespace <namespace> --reset-values --values
overrides.yaml
```

Note: If you prefer not to use an override file, you can instead override default values directly from the `helm upgrade` command, by replacing the `--values overrides.yaml` parameter with `--set <field>=<value>` parameters.

a) Confirm the success of the upgrade.

Run the following command, replacing *<namespace>* with the namespace in which the IBM MQ Operator is installed:

```
helm list
kubectl get pods -n <namespace>
```

4. Upgrade IBM MQ queue managers.

After upgrading the IBM MQ Operator, upgrade any IBM MQ queue managers to a newer version by following the procedure in "Upgrading IBM MQ queue managers on Amazon EKS" on page 82.

Upgrading the IBM MQ Operator on Amazon EKS in an air-gap environment

If your Amazon EKS cluster is not directly accessible on the internet, or you wish to use your own image registry, you can mirror images to a registry in your network-restricted environment by using a bastion host, or a portable compute or storage device.

Before you begin

Important: This topic is for upgrading deployments of the IBM MQ Operator on Amazon EKS clusters *without* access to the internet, or where you wish to use your own image registry. If your Amazon EKS cluster has access the internet, and you do not wish to use your own image registry, see "Upgrading the IBM MQ Operator on Amazon EKS for online installations" on page 78 instead. If you deployed the IBM MQ Operator on Red Hat OpenShift, see "Upgrading the IBM MQ Operator and queue managers on Red Hat OpenShift" on page 54 instead.

Note: The latest CD version of the IBM MQ Operator is 4.0.2. For the latest IBM MQ Operator release notes, see "Release history for IBM MQ Operator" on page 5.

Before upgrading the IBM MQ Operator, complete the following steps.

- Ensure that the installed IBM MQ Operator, and any queue managers, are running.
- If you installed multiple IBM MQ Operators on the cluster, ensure that you are upgrading to a version that is greater than, or equal to, the latest installed IBM MQ Operator. (This step ensures that you do not downgrade the Custom Resource Definition (CRD).)

About this task

This documentation assumes that you will mirror images for the upgrade to the same image registry that is used by the current installed IBM MQ Operator and queue manager images.

Procedure

1. Mirror the images by using one of the following approaches:
   - Mirroring images with a bastion host on Amazon EKS
   - Mirroring images with a portable compute or storage device on Amazon EKS

2. Prepare the IBM MQ Helm Chart.

   When you mirrored images, you downloaded the IBM MQ CASE. The command that you used included the `--install-method helm` parameter, which results in the IBM MQ Helm chart being included in the CASE. The Helm chart is in a `.tgz` file at the following location:

   ```
   ~/.ibm-pak/data/cases/ibm-mq/$OPERATOR_VERSION/charts/ibm-mq-operator-$OPERATOR_VERSION.tgz
   ```

   Note: If the Helm chart was not downloaded with the CASE, you can download the `ibm-mq-operator` chart from https://github.com/IBM/charts/tree/master/repo/ibm-helm.

   Extract the `.tgz` file then enter the directory. For example, run the following command, changing <PATH> to the directory in which to extract the `.tgz` file (this directory must already exist):

   ```
   tar -xvzf ~/.ibm-pak/data/cases/ibm-mq/$OPERATOR_VERSION/charts/ibm-mq-operator-
   $OPERATOR_VERSION.tgz -C <PATH>
   cd <PATH>/ibm-mq-operator
   ```

3. Upgrade the CRD.

   When you upgrade the IBM MQ Operator, Helm does not deploy the new `QueueManager` CRD from the Helm chart because a `QueueManager` CRD already exists in the cluster. You must therefore manually update the CRD.

   Apply the new `QueueManager` CRD from the new chart by running the following command. You must specify the `--force-conflicts` parameter only the first time that you upgrade the CRD.:

   ```
   kubectl apply --server-side --force-conflicts -f crds/queuemanagers.yaml
   ```

4. Upgrade the IBM MQ Operator

   a) Log into the Amazon EKS cluster.

   b) You should already have a `docker-registry` type secret that contains credentials for the private registry, in the namespace in which the operator is installed. Retrieve the secret by running the following command, where <NAMESPACE> is the namespace in which you will deploy the IBM MQ Operator.

   ```
   kubectl get secrets -n <NAMESPACE>
   ```

   c) View the Helm chart values that you used when you previously installed or upgraded the IBM MQ Operator.

   Important: When you install the IBM MQ Operator in an air-gap environment, you must override some of the values in the IBM MQ Operator Helm chart. When you upgrade the IBM MQ Operator, you must apply the same values that you used previously otherwise both the queue managers and the IBM MQ Operator are likely to experience downtime. Default values are documented in "IBM MQ Operator Helm chart values" on page 332. You might have previously set values by using one of the following methods when you installed or upgraded the IBM MQ Operator:

   - Specifying the `--set <FIELD>=<VALUE>` parameter of the `helm install` or `helm upgrade` command
   - Specifying the `--values <FILENAME>.yaml` parameter of the `helm install` or `helm upgrade` command
   - Modifying the original `values.yaml` (this method is not recommended)

   To find the values that you used previously, run the following command:

   ```
   helm get values ibm-mq-operator
   ```

   d) If you have the `overrides.yaml` file that you used previously, verify that the values in the file match the values that you obtained in the previous step.

   e) If you don't have the `overrides.yaml` file that you used previously, re-create it by extracting the current values from the deployed operator. Run the following command, replacing *<namespace>* with the namespace in which the IBM MQ Operator is installed:

```
helm get values ibm-mq-operator -n <namespace> -o yaml > overrides.yaml
```

Review the `overrides.yaml` file to ensure it is as required.

f) Upgrade the IBM MQ Operator Helm chart, specifying the overrides file.

Run the following command, replacing <NAMESPACE> with the namespace in which the IBM MQ Operator is installed:

```
helm upgrade ibm-mq-operator . --namespace <NAMESPACE> --reset-values --values
overrides.yaml
```

Important: The `--reset-values` parameter ensures that Helm first uses the values from the chart that you are upgrading to, then applies the overrides file. The `helm upgrade` command has an alternative flag: `--reset-then-reuse-values`. The flag's documented behavior is *reset the values to the ones built into the chart, apply the last release's values and merge in any overrides from the command line via --set and -f.*. This flag might reduce the likelihood of accidentally omitting previously specified values. However, when you use the `--reset-then-reuse-values` parameter, the final state after the upgrade depends on the initial state, so can vary for each upgrade, or for each environment.

Note: If you prefer not to use an override file, you can instead override default values directly from the `helm upgrade` command, by replacing the `--values overrides.yaml` parameter with the `--set <FIELD>=<VALUE>` parameter.

g) Confirm the success of the upgrade.

Run the following command, replacing *<namespace>* with the namespace in which the IBM MQ Operator is installed:

```
helm list
kubectl get pods -n <namespace>
```

5. Upgrade IBM MQ queue managers.

After upgrading the IBM MQ Operator, upgrade any IBM MQ queue managers to a newer version by following the procedure in "Upgrading IBM MQ queue managers on Amazon EKS" on page 82.

Upgrading IBM MQ queue managers on Amazon EKS

You can upgrade an IBM MQ queue manager that was deployed by using the IBM MQ Operator in Amazon EKS, by using the command line.

Before you begin

Note: The latest CD version of the IBM MQ queue manager is 10.0.0.0-r3. For the latest IBM MQ queue manager release notes, see "Release history for queue manager container images" on page 7.

Ensure that your IBM MQ Operator is at the required version. To upgrade your IBM MQ Operator, see "Upgrading the IBM MQ Operator on Amazon EKS" on page 77.

Important: For air-gap deployments, you must mirror the images of the IBM MQ Operator version that released alongside the queue manager version you want to upgrade to. You must do this before you can upgrade the queue manager. If you are upgrading your queue managers to the latest version that released alongside of the IBM MQ Operator that you have deployed, you should have already mirrored the queue manager images during the IBM MQ Operator upgrade. You can look up which IBM MQ CASE version contains the operator and queue manager version that you require in the IBM MQ CASE Lookup Tables.

Procedure

1. Log in to your cluster's command line interface.
2. Edit the version and license fields in the `QueueManager` resource.

Important: In an air-gap environment, the QueueManager custom resource YAML for each queue manager must also include `spec.imagePullSecrets` if the private registry that contains the images

requires credentials. This field must reference a Kubernetes secret that contains the private registry credentials. For example:

```
spec:
    imagePullSecrets:
        - name: <KUBERNETES_SECRET_NAME>
```

This code will already exist for a queue manager that is being upgraded.

a) Run the following command to open the `QueueManager` resource for editing, replacing <QUEUEMANAGER_NAME> with the `metadata.name` of the queue manager that you are upgrading and <NAMESPACE> with the namespace that the queue manager is installed in:

```
kubectl edit queuemanager <QUEUEMANAGER_NAME> -n <NAMESPACE>
```

b) In the resource, update the following fields (where necessary) to match your desired IBM MQ queue manager version upgrade.

- `spec.version`
- `spec.license.license`

For more information on valid licenses for different IBM MQ versions, see "Licensing and API references for the IBM MQ Operator" on page 281.

The following table lists the latest version of the IBM MQ queue manager for each active Operator channel.

Operator channel	Latest IBM MQ queue manager
v4.0 (CD)	10.0.0.0-r3

You can look up which IBM MQ CASE version contains the operator and queue manager version that you require in the "IBM MQ CASE Lookup Tables" on page 330.

Note: If you are installing in an air-gap environment, you must upgrade only to queue manager versions where you already mirrored the corresponding images. The CASE lookup tables show which queue manager version images were mirrored when you mirrored each IBM MQ CASE.

3. Verify the queue manager status by running the following command, replacing <NAMESPACE> and <QUEUEMANAGER_NAME> as before:

```
kubectl get queuemanager <QUEUEMANAGER_NAME> -n <NAMESPACE>
```

Uninstalling the IBM MQ Operator on Amazon EKS

To uninstall the IBM MQ Operator from Amazon EKS, you uninstall the Helm chart then delete the Queue Manager Custom Resource Definitions (CRD).

Before you begin

Note: These instructions are for uninstalling the IBM MQ Operator on Amazon EKS. To uninstall the IBM MQ Operator on Red Hat OpenShift, see "Uninstalling the IBM MQ Operator on Red Hat OpenShift" on page 64.

About this task

Important: Due to limitations in Helm, the CRDs are not deleted when you use Helm to uninstall the IBM MQ Operator. To clean up the QueueManager CRD, you must manually delete it, which causes **all** IBM MQ queue managers to also be deleted.

Procedure

1. Uninstall the IBM MQ Operator Helm chart by running the following command, replacing <NAMESPACE> with the namespace in which the IBM MQ Operator is deployed:

   ```
   helm uninstall ibm-mq-operator --namespace <NAMESPACE>
   ```

2. Delete the QueueManager CRD.

   Important: Deleting the QueueManager CRD causes **all** IBM MQ queue managers to also be deleted.

   If you are sure that you want to delete all the queue managers, run the following command:

   ```
   kubectl delete crd/queuemanagers.mq.ibm.com
   ```

OpenShift CP4I Verifying image signatures

IBM MQ Operator and IBM MQ queue manager container images are digitally signed. Digital signatures provide a way for consumers of content to ensure that what they download is both authentic (it originated from the expected source) and has integrity (it is what we expect it to be).

Before you begin

The machine on which you want to do signature verification must meet the following requirements:

- These command line tools must be installed on the machine:
  - Gnu Privacy Guard v2
  - OpenSSL
  - skopeo
- The IBM Cloud Pak for Integration public keys must exist on the machine. There are three public keys, for images with the following publish dates:
  - Before 2022-07-22
  - Between 2022-07-23 and 2023-03-22
  - After 2023-03-22

  Copy the following text block exactly as shown into a text editor, then save it in a file named `cp4i-public.gpg`:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQENBF8PmIABCADhk5bSsNY7Oi3sA5uxqXjNY2vHFEIgaHhhdWc6Y61mRcowcJPl
JclMvNJltmQCFgInQ8uhuXYq2N+q1Yk+Q6PzRRUmbFtA88O43ZoW8hd7A+Ukh55b
to/tSYtwRdR4l05kX4dZsP/kpYEzU+hc3buhn3y/LvM9uNcK05t+M402cRzROx48
Vl22lSX+DsuZMj78ECREZU9uDEyAOCyGuuk94mLJkvD7QRN8IFjQXVSt7+aOb/LX
Ox5jRORg9m1aOPQOgwRCMR+A2uwdyFA6LFddC0uEzvttiNe+/yN23VoSM+uoF8wS
H1TIV6QCaqemZ1FajQyuW76cykXfUB+KnkCbABEBAAG0CGNwNGlzaWduiQE5BBMB
CAAjBQJfD5iAAhsvBwsJCAcDAgEGFQgCCQoLBBYCAwECHgECF4AACgkQaRnhuFDQ
5lvoaAgAt7nKxRv2FZ5geYyDW31n1J4LJn2BW/6KoMV9hlv9pSqPbaneMjaX0w4A
jABoaolmDjrjPmaLJY+Gh7KahCwmnJkrrHZmpUIw0rgNJFRlMiWrs61+qFFJEmst
PO33He6JxL5MbJDIKP16Zvxg1+4Vjkd+Hl0ZA0HvfFpdPXl0TMaVelqaZhmavQJx
cLq34lglHOUC3NuAa4ab5YSdCXQ5j6RQKV4M81TVBbgtm2fsvHp+wK09Ruu8s71h
i4Xq2eTopdnn1hKUfDPwXolaa/dChnWMkvXOdVjZn+nTSyYigfqoLCnr0aAnCb8O
3DzzaOf293Klj0JCwRDOGpJEiyfKmg==
=a1wb
-----END PGP PUBLIC KEY BLOCK-----
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQINBGLLaAABEAC6oFMtVrdFn2JaQ9he6J75+7mKvnouSzbqBPVVxpdyOIY93/Xi
fG+01JPkDCFA968xDxK+f9pt7fDXpY1EkWQI5K2ejxFz1NutaKERdYflFY5etXht
w90h1Yz5rBhp5eAMzHxog++MhPXOt1kYG5nb99zMuiOgFQFtussmDiFCjOaY8rzL
Bl3h0YFehACGi1U06uzZRv85ZaxpDBMA1/dkUUXPMAYC7HM/pL1lWn0IWwpdpQ9n
MPqTlWsubweB+Gb01HdDKpG9NVV95pB9oki0qN8yh26cgxkGd0td7MTMiSr+CMdS
WZL84LvscFpcrdCBIPnWL3shOvKHNkP7flj0sF4QBR+iIFE9BdKrDksHxz4wLBFb
qm0A2RTDP4RhoKHD5byRzv+aYYxiD7z+z9614ET3//kzsCa7cr122LAtHJ4QwudR
Nm7hPIg4AY2+cMZl/GqRecR+FuoGbLJunzwqIEdyZHEhwWOtXgcvSnGhWSEEvi0w
w4EmWk6UKhS4RGn7JyvO8GVcgf1o/1cFgV92nOiKHkZMFEfwgr8F3Nt17xij4os5
```

```
yk0oWkpmqojRhYwL7Ub4XwpWBwowKBhUxFAKnf15OOysu8DRpEG5a9JkSE/kRS2i
FyPhgFo5t/mocsgVkzYC+MQq2rJ5waOCGMHovqdbeKjrcMTEZVxuAXOIbwARAQAB
tAhjcDRpc2lnbokCOQQTAQgAIwUCYstoAAIbLwcLCQgHAwIBBhUIAgkKCwQWAgMB
Ah4BAheAAAoJEDw8drvUQ1QP1KcQALgBc8T4C82csYaPmZbPmb6yrZMOA6RyD4gD
MVO5iorx7EBcI//VSGOxvb8+NnY2P0xjVskV3kKnrr8IsZdZVrYL5irjKkB5HTMK
DYAOIYNrvV2x0QRBwXrvtVvP4LKMfNzISPVADbWNcV3aBZ7cZsEQOs6YJ1eM2wbl
437+8dKRCDs8Yo4D3deyxXRv/mTnvmPwLYDFWjCQMrJrkHMIrIQJMRQ66rGUDMgh
XyaAd6om/ir3WRuLVyTBVdnCqMCdEjFnzzLX6NK2MLBTyONPNRr/419DYDoumbhR
JHrew59kx9c+HAJqdk54xfOy5GJSt1opQt5oQ3gke/3bdg6/uiUTSwdM2OE46zB+
dQs3FZqlUAs7T1l616XccI0+h/tIkCQFg3i7dk2RRZ6sFhbj19lziUHnTszO+FxV
iI32w4UExM7xpfs2RalGCYMFUjq+xlMcxm1t3y6GwLxH2O92di99HRqrJPT//Qd+
vuENqqByW83CSB5oc/WHsoXIh5FAkt5PLp9bymqcoUloB8FDb4h5smszUIsgXrsq
HSK7uKsJAiRYFKdig85PK85vtEvyIPfrcx9nhbAKUfexKsQne/G3B+ebj+UDmNyA
/4fMqyR8ZTqCM4MkuYQXaIObu1bN8sCbstJuGTYvBfTI4ElXVHWk9tdlPub1HatR
2SdIxtM7
=PgzJ
-----END PGP PUBLIC KEY BLOCK-----
-----BEGIN PGP PUBLIC KEY BLOCK-----

mQINBGP3eKsBEADesGAGhDZL/IVcB48Y/68Lcp/thTLp44WDcTGQ0fVXAtNWzW+X
1iTW/sn+2XAQv1NoN1bTVsatfvFNKKW7WKSYSi4qnz942yjEAb7wRyylOtb3aMlL
zlTISHFRwhl98K6J+555vToxK2iGeuTv0CvzGB+2KF4mrB3VlIhHm8JK+QQsD9O6
IDcar+GzLkd3ZBx5zN7ivEvDcZ/qxiEeFnFIQ2jknqoZ+gsDkpYfHOQtNDR45+WN
Y3VGkUVPYdAHWbU8uJeGcfhT87pdeMx6zQ+mrkFM2eptsL7muQF66Q5PLvIDWGXs
N1eCbOAITgXdd1CbhkM11U2pgYZ7ONyAJp22dUJP+43UC3aQt3IVGhDJa2LryUVT
HfsoJ84T38lH/GQHAxjw7a/0Sq8xy0iEtA81lukiK9D50796mbs25kanxslfDBoX
5a2UQrX45Grq1h5+vR3QVx/wRmfHCyAo0sMImnHG2SVDZpVhJWChYxWwQOgIDZDE
scxa+QU3bPy0bNKJdsSBb/4rW4mCYbX5K1119VrtTr0tRHU7ItkW21uzz5oxb9eb
LsGsM9hwKbQXCaU83UAJy/V7JtJk3nJ+JihLpaUhszoMPgiuvXu+jH3uqoHhLjwc
SzfSX3aXLzsIvYcDPiKWWrYK5JxWx/AWGP8o1QCflobS4MaAH5UXHBOBCwARAQAB
tD5JbnRlcm5hdGlvbmFsIEJ1c2luZXNzIE1hY2hpbmVzIENvcnBvcmF0aW9uIDxw
c2lydEB1cy5pYm0uY29tPokCOgQTAQgAJAUCY/d4qwIbDwULCQgHAgYVCgkICwIE
FgIDAQIeAQUJAAAAAAAKCRCVFASm54U5I/ThEACDegHJz7Dl4aAz2XHC6c/C5ZrJ
7W++tckc3tX3wFg2Ux8AWUwc/3oaT63nG3J7JsF0y+FmD2Loi5grReKRIE7IAAga
aA6y/S73VOm7sApIz4hRvQ8DoWYcuu/pFMZaQKLKh3d7n0Uhlg+IxcnDWYT/xKvn
1RrCOLhokEu22zRDg+VGClmmiyQ8qZdyA8c6OqjT9U2RFwSUnNPkLsdoCU5KVT5l
1znNG5gEovgxIRBbeJuoYXBRI1UO7bH0w3poTZ8dPFzOSLTfG5hLF1sZVMfGBlCf
c94rtLUJyU5LBBe7Py2y0q89v9L1a/BwNRe1ron39PLsmXPUFqV5f796wAnMif9l
5FI1HDHpkjn3RRHjPMUQSeMOJP9HMLjhEIg9I0bDxsbqCg4XW2xfk8J0YE3GhF2p
X5BKFThsMQWAKwwCRrD2EMW3YMv7/E385m3QA3xTU4z2SFEyp/KaoTxFw+Z9Z6ta
7tzdE4NqmcEcVSrOvktIQ3xi9UlngOp7OACe10WhmJiDkAfaTOExHfRLVzvo3ryA
5pNJ6FkV+eJVljBMUzRSkyyGrnesh6c6ViBcV0rZpiE1f7T4gp9tg2YQaOMaoI6P
M9LfRKfyxSklz/aX811eRfmJRPkA9pF0DlCV6j9XJwFaSKZhxoVQVu2/pIGzQwuu
uBfw3hQI0oUbrt3dIw==
=yKDk
-----END PGP PUBLIC KEY BLOCK-----
```

Procedure

1. Import the IBM Cloud Pak for Integration public key on the machine that you prepared according to the *Before you begin* section:

   ```
   sudo gpg2 --import cp4i-public.gpg
   ```

   You need to do this step only once on each machine that you use for signature verification.

2. Calculate the fingerprints:

   ```
   fingerprint=$(sudo gpg2 --fingerprint --with-colons | grep 'cp4i\|ibm' -B1 | grep fpr | tr
   -d 'fpr:')
   ```

   This command stores the fingerprints for all the keys in an environment variable called `fingerprint`, which is needed to verify the signature. When you exit your shell session, the variable is deleted. The next time you log in to your machine, you can set the variable again by rerunning the command.

3. Create a directory for the image then use skopeo to pull the image into local storage.

   Run the following command, replacing <IMAGE_LOCATION> with the full path and tag of the image whose signature you are verifying. For example, `docker://cp.icr.io/cp/ibm-mqadvanced-server:9.4.4.0-r1-amd64`:

   ```
   mkdir images
   skopeo copy <IMAGE_LOCATION> dir:./images
   ```

This command downloads the image as a set of files then places them in the `images` directory. These files include a manifest file called `images/manifest.json` and a signature file called `images/signature-1`. You reference both these files in the next step.

4. Verify the signature.

   Run the following command, replacing <IMAGE> with the full path and tag of the image whose signature you are verifying. For example, `cp.icr.io/cp/ibm-mqadvanced-server:9.4.4.0-r1-amd64`:

   ```
   for fp in ${fingerprint}; do sudo skopeo standalone-verify ./images/manifest.json <IMAGE>
   ${fp} ./images/signature-1; done
   ```

   You get a confirmation similar to this:

   ```
   FATA[0000] Error verifying signature: Signature by 0000000000000000000000000000000000000000
   does not match expected fingerprint 1111111111111111111111111111111111111111
   FATA[0000] Error verifying signature: Signature by 0000000000000000000000000000000000000000
   does not match expected fingerprint 2222222222222222222222222222222222222222
   Signature verified, digest
   sha256:0000000000000000000000000000000000000000000000000000000000000000
   ```

   In this case, the image was not signed by the first key or the second key, which is the cause of the first two errors. However, it was verified as signed by the final key. If the image to be verified is signed by one of the other keys, the message order is different.

Kubernetes Preparing to use Kubernetes by creating a pull secret

The prebuilt IBM MQ Advanced image can be pulled from the IBM Container Registry, but the registry performs a license entitlement check. This check requires an entitlement key that is stored in a `docker-registry` pull secret.

Before you begin

Install the **`kubectl`** command line tool and log into your Kubernetes cluster.

Procedure

1. Get the entitlement key that is assigned to your IBM ID.
   a) Log in to MyIBM Container Software Library with the IBM ID and password that are associated with the entitled software.
   b) In the **Entitlement keys** section, select **Copy key** to copy the entitlement key to the clipboard.
2. Create the entitlement key secret.
   a. From the command line interface on your cluster, run the following command to create an image pull secret called `ibm-entitlement-key`.

   ```
   kubectl create secret docker-registry ibm-entitlement-key \
   --docker-server=cp.icr.io \
   --docker-username=cp \
   --docker-password=entitlement_key \
   --docker-email=user_email \
   --namespace=namespace
   ```

   Where *entitlement_key* is the entitlement key that you copied in step b, *user_email* is the IBM ID associated with the entitled software, and *namespace* is the namespace that you plan to use for queue managers.

Kubernetes Preparing to use Kubernetes by installing the IBM License Service

Container licensing allows you to license only the available capacity of your individual IBM MQ containers, rather than requiring you to license the entire server where your containers are running. To take

advantage of container licensing, the IBM License Service must be used to track license usage and determine your required entitlement.

Procedure

Install the IBM License Service by following the instructions in Installing License Service with Helm charts in the IBM Cloud Pak foundational services documentation.

Linux Preparing to use Podman or Docker by pulling the prebuilt container image

The prebuilt IBM MQ Advanced image can be pulled from the IBM Container Registry using command line tools, such as Podman or Docker.

Before you begin

Install Podman or Docker.

Note: The following example commands use Podman, but you can use either Podman or Docker with the same command parameters.

Procedure

1. Get the entitlement key that is assigned to your IBM ID.
   a) Log in to MyIBM Container Software Library with the IBM ID and password that are associated with the entitled software.
   b) In the **Entitlement keys** section, select **Copy key** to copy the entitlement key to the clipboard.
2. Log into the container registry

   For example, using Podman:

   ```
   podman login --username cp --password ... cp.icr.io
   ```

   Where "..." is replaced with the entitlement key you copied in the previous step.
3. Pull the image

   For example, using Podman:

   ```
   podman pull cp.icr.io/cp/ibm-mqadvanced-server:10.0.0.0-r3
   ```

Preparing for IBM MQ by building your own container image

Develop a self-built container. This is the most flexible container solution, but it requires you to have strong skills in configuring containers, and to "own" the resultant container.

Before you begin

Before you develop your own container, consider whether you can instead use the IBM MQ Operator. See “Choosing how you want to use IBM MQ in containers” on page 9

About this task

Procedure

- “General considerations when building your own queue manager image” on page 88
- “Building a sample IBM MQ queue manager container image” on page 88
- “Running local binding applications in separate containers” on page 91
- Review the IBM MQ sample Helm chart.

General considerations when building your own queue manager image

There are several requirements to consider when running an IBM MQ queue manager in a container. The sample container image provides a way to handle these requirements, but if you want to use your own image, you need to consider how these requirements are handled.

Process supervision

When you run a container, you are essentially running a single process (PID 1 inside the container), which can later spawn child processes.

If the main process ends, the container runtime stops the container. An IBM MQ queue manager requires multiple processes to be running in the background.

For this reason, you need to make sure that your main process stays active as long as the queue manager is running. It is good practice to check that the queue manager is active from this process, for example, by performing administrative queries.

Populating `/var/mqm`

Containers must be configured with `/var/mqm` as a volume.

When you do this, the directory of the volume is empty when the container first starts. This directory is usually populated at installation time, but installation and runtime are separate environments when using a container.

To solve this, when your container starts, you can use the **`crtmqdir`** command to populate `/var/mqm` when it runs for the first time.

Container security

In order to minimize the runtime security requirements, the samples container images are installed using the IBM MQ unzippable install. This ensures that no `setuid` bits are set, and that the container doesn't need to use privilege escalation. Some container systems define which user IDs you are able to use, and the unzippable install does not make any assumptions about available operating system users.

Building a sample IBM MQ queue manager container image

Use this information to build a sample container image for running an IBM MQ queue manager in a container.

About this task

Firstly, you build a base image containing an Red Hat Universal Base Image file system and a clean installation of IBM MQ.

Secondly, you build another container image layer on top of the base, which adds some IBM MQ configuration to allow basic user ID and password security.

Finally, you run a container using this image as its file system, with the contents of `/var/mqm` provided by a container-specific volume on the host file system.

Procedure

- For information on how to build a sample container image for running an IBM MQ queue manager in a container, see the following subtopics:
  - “Building a sample base IBM MQ queue manager image” on page 89
  - “Building a sample configured IBM MQ queue manager image” on page 89

Building a sample base IBM MQ queue manager image

In order to use IBM MQ in your own container image, you need initially to build a base image with a clean IBM MQ installation. The following steps show you how to build a sample base image, using sample code hosted on GitHub.

Procedure

- Use the make files supplied in the mq-container GitHub repository to build your production container image.

  Follow the instructions in Building a container image on GitHub.
- Optional: If you plan to configure secure access using the Red Hat OpenShift Container Platform "restricted" Security Context Constraint (SCC), use one of the IBM MQ non-install images.

  Links to download these images are available in the Containers section of IBM MQ downloads.

Results

You now have a base container image with IBM MQ installed.

You are now ready to build a sample configured IBM MQ queue manager image.

Building a sample configured IBM MQ queue manager image

After you have built your generic base IBM MQ container image, you need to apply your own configuration to allow secure access. To do this, you create your own container image layer, using the generic image as a parent.

Before you begin

This task assumes that, when you built your sample base IBM MQ queue manager image, you used the "No-Install" IBM MQ package. Otherwise you cannot configure secure access using the Red Hat OpenShift Container Platform "restricted" Security Context Constraint (SCC). The "restricted" SCC, which is used by default, uses random user IDs, and prevents privilege escalation by changing to a different user. The IBM MQ traditional RPM-based installer relies on an `mqm` user and group, and also uses `setuid` bits on executable programs. In the current version of IBM MQ, when you use the "No-Install" IBM MQ package, there is no `mqm` user any more, nor an `mqm` group.

Procedure

1. Create a new directory, and add a file called `config.mqsc`, with the following contents:

```
DEFINE QLOCAL(EXAMPLE.QUEUE.1) REPLACE
```

   Note that the preceding example uses simple user ID and password authentication. However, you can apply any security configuration that your enterprise requires.

2. Create a file called `Dockerfile`, with the following contents:

```
FROM mq
COPY config.mqsc /etc/mqm/
```

3. Build your custom container image using the following command:

```
docker build -t mymq .
```

   where "`.`" is the directory containing the two files you have just created.

   Docker then creates a temporary container using that image, and runs the remaining commands.

   Note: On Red Hat Enterprise Linux (RHEL), you use the command **`docker`** (RHEL V7) or **`podman`** (RHEL V7 or RHEL V8). On Linux, you will need to run **`docker`** commands with **`sudo`** at the beginning of the command, to gain extra privileges.

4. Run your new customized image to create a new container, with the disk image you have just created.

   Your new image layer did not specify any particular command to run, so that has been inherited from the parent image. The entry point of the parent (the code is available on GitHub):

   - Creates a queue manager
   - Starts the queue manager
   - Creates a default listener
   - Then runs any MQSC commands from `/etc/mqm/config.mqsc`.

   Issue the following commands to run your new customized image:

```
docker run \
  --env LICENSE=accept \
  --env MQ_QMGR_NAME=QM1 \
  --volume /var/example:/var/mqm \
  --publish 1414:1414 \
  --detach \
  mymq
```

   where the:

   First env parameter
   Passes an environment variable into the container, which acknowledges your acceptance of the license for IBM IBM WebSphere MQ. You can also set the LICENSE variable to view to view the license.

   See IBM MQ license information for further details on IBM MQ licenses.

   Second env parameter
   Sets the queue manager name that you are using.

   Volume parameter
   Tells the container that whatever MQ writes to `/var/mqm` should actually be written to `/var/example` on the host.

   This option means that you can easily delete the container later, and still keep any persistent data. This option also makes it easier to view log files.

   Publish parameter
   Maps ports on the host system to ports in the container. The container runs by default with its own internal IP address, which means that you need to specifically map any ports that you want to expose.

   In this example, that means mapping port 1414 on the host to port 1414 in the container.

   Detach parameter
   Runs the container in the background.

Results

You have built a configured container image and can view running containers using the **`docker ps`** command. You can view the IBM MQ processes running in your container using the **`docker top`** command.

Attention:

You can view the logs of a container using the **`docker logs ${CONTAINER_ID}`** command.

What to do next

- If your container is not shown when you use the **`docker ps`** command the container might have failed. You can see failed containers by using the **`docker ps -a`** command.
- When you use the **`docker ps -a`** command, the container ID is displayed. This ID was also printed when you issued the **`docker run`** command.

- You can view the logs of a container by using the **`docker logs ${CONTAINER_ID}`** command.

Running local binding applications in separate containers

With process namespace sharing between containers, you can run applications that require a local binding connection to IBM MQ in separate containers from the IBM MQ queue manager.

About this task

You must adhere to the following restrictions:

- You must share the containers PID namespace using the `--pid` argument.
- You must share the containers IPC namespace using the `--ipc` argument.
- You must either:
  1. Share the containers UTS namespace with the host using the `--uts` argument, or
  2. Ensure the containers have the same hostname using the `-h` or `--hostname` argument.
- You must mount the IBM MQ data directory in a volume that is available to the all containers under the `/var/mqm` directory.

The following example uses the sample IBM MQ container image. You can find details of this image on Github.

Procedure

1. Create a temporary directory to act as your volume, by issuing the following command:

```
mkdir /tmp/dockerVolume
```

2. Create a queue manager (`QM1`) in a container, with the name `sharedNamespace`, by issuing the following command:

```
docker run -d -e LICENSE=accept -e MQ_QMGR_NAME=QM1 --volume /tmp/dockerVol:/mnt/mqm
--uts host --name sharedNamespace ibmcom/mq
```

3. Start a second container called `secondaryContainer`, based off `ibmcom/mq`, but do not create a queue manager, by issuing the following command:

```
docker run --entrypoint /bin/bash --volumes-from sharedNamespace --pid
container:sharedNamespace --ipc container:sharedNamespace --uts host --name
secondaryContainer -it --detach ibmcom/mq
```

4. Run the **dspmq** command on the second container, to see the status for both queue managers, by issuing the following command:

```
docker exec secondaryContainer dspmq
```

5. Run the following command to process MQSC commands against the queue manager running on the other container:

```
docker exec -it secondaryContainer runmqsc QM1
```

Results

You now have local applications running in separate containers, and you can now successfully run commands like **dspmq**, **amqsput**, **amqsget**, and **runmqsc** as local bindings to the `QM1` queue manager from the secondary container.

If you do not see the result you expected, see “Troubleshooting your namespace applications” on page 92 for more information.

Troubleshooting your namespace applications

When using shared namespaces, you must ensure that you share all namespaces (IPC, PID and UTS/hostname) and mounted volumes, otherwise your applications will not work.

See "Running local binding applications in separate containers" on page 91 for a list of restrictions you must follow.

If your application does not meet all the restrictions listed, you could encounter problems where the container starts, but the functionality you expect does not work.

The following list outlines some common causes, and the behavior you are likely see if you have forgotten to meet one of the restrictions.

- If you forget to share either the namespace (UTS/PID/IPC), or the hostname of the containers, and you mount the volume, then your container will be able to see the queue manager but not interact with the queue manager.
  - For **dspmq** commands, you see the following:

```
docker exec container dspmq

QMNAME(QM1)                     STATUS(Status not available)
```

  - For **runmqsc** commands, or other commands that try to connect to the queue manager, you are likely to receive an AMQ8146 error message:

```
docker exec -it container runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2026.
Starting MQSC for queue manager QM1.
AMQ8146: IBM MQ queue manager not available
```

- If you share all the required namespaces but you do not mount a shared volume to the `/var/mqm` directory, and you have a valid IBM MQ data path, then your commands also receive AMQ8146 error messages.

  However, **dspmq** is not able to see your queue manager at all, and instead returns a blank response:

```
docker exec container dspmq
```

- If you share all the required namespaces but you do not mount a shared volume to the `/var/mqm` directory, and you do not have a valid IBM MQ data path (or no IBM MQ data path), then you see various errors as the data path is a key component of an IBM MQ installation. Without the data path, IBM MQ cannot operate.

  If you run any of the following commands, and you see responses similar to those shown in these examples, you should verify that you have mounted the directory or created an IBM MQ data directory:

```
docker exec container dspmq
'No such file or directory' from /var/mqm/mqs.ini
AMQ6090: IBM MQ was unable to display an error message FFFFFFFF.
AMQffff

docker exec container dspmqver
AMQ7047: An unexpected error was encountered by a command. Reason code is 0.

docker exec container mqrc
<file path>/mqrc.c[1152]
lpiObtainQMDetails --> 545261715

docker exec container crtmqm QM1
AMQ8101: IBM MQ error (893) has occurred.

docker exec container strmqm QM1
AMQ6239: Permission denied attempting to access filesystem location '/var/mqm'.
AMQ7002: An error occurred manipulating a file.

docker exec container endmqm QM1
AMQ8101: IBM MQ error (893) has occurred.
```

```
docker exec container dltmqm QM1
AMQ7002: An error occurred manipulating a file.

docker exec container strmqweb
<file path>/mqrc.c[1152]
lpiObtainQMDetails --> 545261715
```

MQ Adv. Kubernetes Creating the Native HA group if creating your own containers

You must create, configure, and start three queue managers to create the Native HA group.

About this task

The recommended method for creating a Native HA solution is by using the IBM MQ operator (see Native HA). Alternatively, if you create your own containers, you can follow these instructions.

To create a Native HA group, you create three queue managers on three nodes with their log type set to `log replication`. You then edit the `qm.ini` file for each queue manager to add the connection details for each of the three nodes so that they can replicate log data to each other.

You must then start all three queue managers so that they can check that all three instances can communicate with one another, and determine which of them will be the active instance and which will be the replicas.

Note: You can only create a Native HA group in your own containers in this way if you are running Kubernetes or Red Hat OpenShift.

Procedure

1. On each of the three nodes, create a queue manager, specifying a log type of log replica, and supplying a unique name for each log instance. Each queue manager has the same name:

   ```
   crtmqm -lr instance_name qmname
   ```

   For example:

   ```
   node 1> crtmqm -lr qm1_inst1 qm1

   node 2> crtmqm -lr qm1_inst2 qm1

   node 3> crtmqm -lr qm1_inst3 qm1
   ```

2. On successful creation of each queue manager, an additional stanza named `NativeHALocalInstance` is added to the queue manager configuration file, `qm.ini`. A `Name` attribute is added to the stanza specifying the supplied instance name.

   You can add optionally add the following attributes to the `NativeHALocalInstance` stanza in the `qm.ini` file:

   `KeyRepository`
   The location of the key repository that holds the digital certificate to use for protection of log replication traffic. The location is given in stem format, that is, it includes the full path and file name without an extension. If the `KeyRepository` stanza attribute is omitted, log replication data is exchanged between instances in plain text.

   `CertificateLabel`
   The certificate label identifying the digital certificate to use for protection of log replication traffic. If `KeyRepository` is provided but `CertificateLabel` is omitted, a default value of `ibmwebspheremq`*`queue_manager`* is used.

CipherSpec
The MQ CipherSpec to use to protect log replication traffic. If this stanza attribute is provided, `KeyRepository` must also be provided. If `KeyRepository` is provided but `CipherSpec` is omitted, a default value of ANY is used.

LocalAddress
The local network interface address that accepts log replication traffic. If this stanza attribute is provided it identifies the local network interface and/or port using the format "[addr][(port)]". The network address can be specified as a hostname, IPv4 dotted decimal, or IPv6 hexadecimal format. If this attribute is omitted, the queue manager attempts to bind to all network interfaces, it uses the port specified in the `ReplicationAddress` in the `NativeHAInstances` stanza matching the local instance name.

HeartbeatInterval
The heartbeat interval defines how often in milliseconds an active instance of a Native HA queue manager sends a network heartbeat. The valid range of the heartbeat interval value is 500 (0.5 seconds) to 60000 (1 minute), a value outside of this range causes the queue manager to fail to start. If this attribute is omitted, a default value of 5000 (5 seconds) is used. Each instance must use the same heartbeat interval.

HeartbeatTimeout
The heartbeat timeout defines how long a replica instance of a Native HA queue manager waits before it decides that the active instance is unresponsive. The valid range of the heartbeat interval timeout value is 500 (0.5 seconds) to 120000 (2 minutes). The value of the heartbeat timeout must be greater than or equal to the heartbeat interval.

An invalid value causes the queue manager to fail to start. If this attribute is omitted a replica waits for 2 x `HeartbeatInterval` before starting the process to elect a new active instance. Each instance must use the same heartbeat timeout.

RetryInterval
The retry interval defines how often in milliseconds a Native HA queue manager should retry a failed replication link. The valid range of the retry interval is 500 (0.5 seconds) to 120000 (2 minutes). If this attribute is omitted a replica waits for 2 x `HeartbeatInterval` before retrying a failed replication link.

3. Edit the `qm.ini` file for each queue manager and add connection details. You add three `NativeHAInstance` stanzas, one for each queue manager instance in the Native HA group (including the local instance). Add the following attributes:

   Name
   Specify the instance name that you used when you created the queue manager instance.

   ReplicationAddress
   Specify the hostname, IPv4 dotted decimal or IPv6 hexadecimal format address of the instance. You can specify the address as a hostname, IPv4 dotted decimal, or IPv6 hexadecimal format address. The replication address must be resolvable and routable from each instance in the group. The port number to use for the log replication must be specified in brackets, for example:

   ```
   ReplicationAddress=host1.example.com(4444)
   ```

   Note: The `NativeHAInstance` stanzas are identical on every instance and could be provided by using automatic configuration (**`crtmqm -ii`**).

4. Start each of the three instances:

   ```
   strmqm QMgrName
   ```

   When the instances are started they communicate to check that all three instances are running, then decide which of the three is the active instance, while the other two instances continue to run as replicas.

Example

The following example shows the section of a `qm.ini` file specifying the required Native HA details for one of the three instances:

```
NativeHALocalInstance:
  LocalName=node-1

NativeHAInstance:
  Name=node-1
  ReplicationAddress=host1.example.com(4444)
NativeHAInstance:
  Name=node-2
  ReplicationAddress=host2.example.com(4444)
NativeHAInstance:
  Name=node-3
  ReplicationAddress=host3.example.com(4444)
```

Creating a Native HA CRR configuration if creating your own containers

You must create, configure, and start three queue managers to create the Native HA group.

About this task

The recommended method for creating a Native HA CRR solution is by using the IBM MQ operator (see “Configuring Native HA CRR using the IBM MQ Operator” on page 123). Alternatively, if you create your own containers, you can follow these instructions.

To create a Native HA CRR configuration, you must create two Native HA groups for a queue manager, one on the 'live' site and one on the 'recovery' site. You then edit the `qm.ini` file for the queue manager to set up the cross-region replication between them.

Note: You can only create a Native HA CRR configuration in your own containers in this way if you are running Kubernetes or Red Hat OpenShift.

Procedure

1. On your Live site, create a Native HA group, see “Creating the Native HA group if creating your own containers” on page 93.
2. On your Recovery site, create a second Native HA group, see “Creating the Native HA group if creating your own containers” on page 93.
3. Specify that the two groups on the live and recovery sites work together in a Native HA CRR configuration by making the following alterations to the qm.ini file for the queue manager.

   On the recovery group, specify the following values in the `NativeHALocalInstance` stanza:

   `GroupName`
   Specify a name for the recovery group, for example, "NHACRR_recovery".

   `GroupRole`
   Specify the current role of the group. In this case set to "Recovery".

   Specify the following values in the `NativeHARecoveryGroup` stanza:

   `GroupName`
   The name of the group on the live site, for example, "NHACRR_live".

   `ReplicationAddress`

   Specify a hostname and port that addresses the replication traffic port on the other group in the Native HA cross-region replication configuration. The address can be a single service address and port or a comma-separated list.

   On the live group, specify the following values in the `NativeHALocalInstance` stanza:

GroupName
Specify a name for the live group, for example, "NHACRR_live".

GroupRole
Specify the current role of the group. In this case set to "Live".

Specify the following values in the `NativeHARecoveryGroup` stanza:

GroupName
The name of the group on the recovery site, for example, "NHACRR_recovery".

ReplicationAddress

Specify a hostname and port that addresses the replication traffic port on the other group in the Native HA cross-region replication configuration. The address can be a single service address and port or a comma-separated list.

4. You must secure the connection used for replication by using TLS. If you specified security for internal replication when you created the two groups, you can use the same settings to secure the external replication. Alternatively, you can specify the following values in the `NativeHALocalInstance` stanza for the recovery group and the live group.

   GroupCipherSpec
   Overrides the Cipher Spec specified by `CipherSpec`.

   GroupCertificateLabel
   Overrides the certificate label specified by `CertificateLabel`.

   If you wish to use these values for external replication, while using plain text for internal replication, set these two values for live and recovery groups, while setting `CipherSpec` to NULL.

Example

The following example shows the section of a `qm.ini` file specifying the required Native HA CRR details for one of the three instances on the live group:

```
NativeHALocalInstance:
  ...
  GroupName=NHACRR_live
  GroupRole=live

NativeHARecoveryGroup:
  GroupName=NHACRR_recovery
  ReplicationAddress=recovery-cluster.fyre.ibm.com(4445)
```

Related information

NativeHALocalInstance stanza of the qm.ini file
NativeHARecoveryGroup stanza of the qm.ini file
NativeHAInstance stanza of the qm.ini file

Deploying and configuring queue managers in containers

You perform a range of tasks to deploy and configure IBM MQ queue managers.

About this task

To get started with deploying and configuring queue managers, see the following topics.

Procedure

- “Deploying and configuring queue managers using the IBM MQ Operator” on page 97
- “Self-deploying and configuring queue managers using the sample Helm chart” on page 215

Deploying and configuring queue managers using the IBM MQ Operator

Configuration examples; configuring HA; connecting from outside an OpenShift or Amazon EKS cluster; integrating with the CP4i dashboard; integrating with Instana tracing; building an image with custom MQSC and INI files; adding custom annotations and labels.

About this task

Procedure

- "Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift" on page 97.
- "Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS" on page 100.
- "Examples for configuring a queue manager" on page 102.
- "Configuring high availability for queue managers using the IBM MQ Operator" on page 118.
- "Configuring connections to a queue manager from outside a cluster" on page 169.
- "Configuring audit logging for IBM MQ in containers" on page 176.
- "Integrating IBM MQ with IBM Instana tracing" on page 177.
- "Configuring authenticated IBM Instana monitoring with TLS" on page 179.
- "Building an image with custom MQSC and INI files, using the Red Hat OpenShift CLI" on page 185.
- "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.
- "Adding annotations to queue managers by using the IBM MQ Operator" on page 188.
- "Adding custom annotations and labels to queue manager resources" on page 189.
- "Disabling runtime webhook checks on Red Hat OpenShift" on page 190.
- "Disabling default value updates to the queue manager specification" on page 190.
- "Running the IBM MQ container with a read-only root file system" on page 190.
- "Configuring the IBM MQ Console with a basic registry using the IBM MQ Operator" on page 193.
- "Expanding persistent volumes" on page 196.

Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift

This example deploys a "quick start" queue manager using the IBM MQ Operator on Red Hat OpenShift. The queue manager uses ephemeral (non-persistent) storage, and turns off IBM MQ security. Messages are not persisted across restarts of the queue manager. You can adjust the configuration to change many queue manager settings.

Before you begin

Note: To deploy a queue manager using the IBM MQ Operator deployed on Amazon EKS, see "Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS" on page 100.

About this task

This task offers 3 options for deploying a queue manager onto OpenShift:

1. Deploy a queue manager with the OpenShift console.
2. Deploy a queue manager with the OpenShift CLI.
3. Deploy a queue manager with the IBM Cloud Pak for Integration Platform UI.

Procedure

- **Option 1: Deploy a queue manager with the OpenShift console.**

  a) Deploy a queue manager.

  a. Log in to the OpenShift console with your Red Hat OpenShift Container Platform cluster administrator credentials.

  b. Change **Project** to the namespace where you installed the IBM MQ Operator. Select the namespace from the **Project** drop-down list.

  c. In the navigation pane, click **Operators** > **Installed Operators**.

  d. In the list on the Installed Operators panel, find and click **IBM MQ**.

  e. Click on the **Queue Manager** tab.

  f. Click on the **Create QueueManager** button. The instance creation panel is displayed, and offers two methods for configuring the resource; the **Form view** and the **YAML view**. The **Form view** is selected by default.

  b) Configure the queue manager.

  Step 2 Option 1: Configure in the **Form view**.

  The **Form view** opens a form that you can use to view or modify the resource configuration.

  a. Next to **License**, click the arrow to expand the license acceptance section.

  b. Set **License accept** to **true** if you accept the license agreement.

  c. Click the arrow to open the drop-down list, and select a license. IBM MQ is available under several different licenses. For more information on the valid licenses, see “mq.ibm.com/v1beta1: Current license versions” on page 281. You must accept the license to deploy a queue manager.

  d. Click **Create**. The list of queue managers in the current project (namespace) is now displayed. The new `QueueManager` should be in a `Pending` state.

  Step 2 Option 2: Configure in the **YAML view**.

  The **YAML view** opens an editor containing an example YAML file for a `QueueManager`. Update the values in the file by following the steps below.

  a. Change `metadata.namespace` to your project (namespace) name.

  b. Change the value of `spec.license.license` to the license string that matches your requirements. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for the license details.

  c. Change `spec.license.accept` to `true` if you accept the license agreement.

  d. Click **Create**. The list of queue managers in the current project (namespace) is now displayed. The new `QueueManager` should be in a `Pending` state.

  c) Verify queue manager creation.

  You can verify that you've created a Queue Manager by completing the following steps:

  a. Ensure that you're in the namespace that you created your IBM MQ Operator in.

  b. From the **Home** screen, click **Operators** > **Installed Operators**, then select the installed IBM MQ Operator that you created the queue manager for.

  c. Click on the **Queue Manager** tab. The creation is complete when the `QueueManager` status is `Running`.

- **Option 2: Deploy a queue manager with the OpenShift CLI.**

  a) Create a `QueueManager` YAML file

For example, to install a basic queue manager in IBM Cloud Pak for Integration, create the file "mq-quickstart.yaml" with the following contents:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: quickstart-cp4i
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-VEMB-RABZSN
    use: NonProduction
  web:
    enabled: true
  queueManager:
    name: "QUICKSTART"
    storage:
      queueManager:
        type: ephemeral
```

Important: If you accept the license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

This example also includes a web server deployed with the queue manager, with the web console enabled with Single Sign-On within IBM Cloud Pak for Integration. For Single Sign-On to work, you first need to install other IBM Cloud Pak for Integration components. See “Installing the IBM MQ Operator for use with IBM Cloud Pak for Integration on Red Hat OpenShift” on page 50.

To install a basic queue manager independently of IBM Cloud Pak for Integration, create the file "mq-quickstart.yaml" with the following contents:

Note: This example uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see “Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator” on page 187.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: quickstart
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
  web:
    enabled: true
  queueManager:
    name: "QUICKSTART"
    storage:
      queueManager:
        type: ephemeral
```

Important: If you accept the MQ license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

b) Create the `QueueManager` object.

```
oc apply -f mq-quickstart.yaml
```

c) Verify queue manager creation.

Verify that you've created a queue manager by completing the following steps:

a. Validate the deployment:

```
oc describe queuemanager Queue_Manager_Resource_Name
```

b. Check the status:

```
oc describe queuemanager quickstart
```

- **Option 3: Deploy a queue manager with the IBM Cloud Pak for Integration Platform UI.**

  Follow the instructions in Deploying an instance by using the Platform UI.

Related tasks

"Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster " on page 169
You need a Red Hat OpenShift Route to connect an application to an IBM MQ queue manager from outside a Red Hat OpenShift cluster. You must enable TLS on your IBM MQ queue manager and client application because SNI is available in the TLS protocol only when a TLS 1.2 or higher protocol is used. The Red Hat OpenShift Container Platform Router uses SNI for routing requests to the IBM MQ queue manager.

"Connecting to the IBM MQ Console" on page 244
How to connect to the IBM MQ Console of a queue manager that has been deployed with the IBM MQ Operator.

"Examples for configuring a queue manager" on page 102
A queue manager can be configured by adjusting the contents of the QueueManager custom resource.

Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS

This example deploys a "quick start" queue manager by using the IBM MQ Operator on Amazon EKS. The queue manager uses ephemeral (non-persistent) storage and turns off IBM MQ security. Messages are not persisted across restarts of the queue manager. You can adjust the configuration to change many queue manager settings.

Before you begin

Note: To deploy a queue manager by using the IBM MQ Operator on Red Hat OpenShift, see "Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift" on page 97.

Deploy the IBM MQ Operator on your Amazon EKS cluster. For more information see "Installing and upgrading the IBM MQ Operator on Amazon EKS" on page 66.

About this task

Queue managers can be installed by a namespace administrator.

Unless you have performed a cluster-scoped (`AllNamespaces`) installation of the IBM MQ Operator, deploy your queue manager in the same namespace where you installed the IBM MQ Operator.

Note: This example uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.

Procedure

1. **Online installations only:** Create an entitlement key secret.

   The IBM MQ Operator deploys queue manager images that are pulled from a container registry that performs a license entitlement check.

   Follow the instructions in "Preparing to use Kubernetes by creating a pull secret" on page 86.

   Note: The entitlement key is not required if you will deploy only IBM MQ Advanced for Developers (Non-Warranted) queue managers.
2. **Air gapped installations only:** Create a `docker-registry secret` with credentials for your private registry, for use when pulling the queue manager image.

   Note: If the IBM MQ Operator installation is namespace scoped, you can reuse the same secret that you used when you installed the Operator, in which case you can skip this step.

Run the following command:

```
kubectl create secret docker-registry mq-airgap-credentials -n <NAMESPACE> --docker-
server=<YOUR_IMAGE_REGISTRY> --docker-username=<USERNAME> --docker-password=<PASSWORD>
```

Replace the following values in the command:

- <NAMESPACE> is the namespace in which you will deploy the queue manager
- <YOUR_IMAGE_REGISTRY> is your private registry, for example `my-private-registry.com`
- <USERNAME> is your username to access your private registry
- <PASSWORD> is your password to access your private registry

3. Create a queue manager YAML file.

   Important: If you accept the license agreement, change `accept: false` to `accept: true`. See "mq.ibm.com/v1beta1: Current license versions" on page 281 for details on the license.

   Create a file called `mq-quickstart.yaml` with the following contents:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: quickstart
  annotations:
    com.ibm.mq/write-defaults-spec: 'false'
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
  queueManager:
    name: "QUICKSTART"
    storage:
      queueManager:
        type: ephemeral
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  web:
    enabled: false
    route:
      enabled: false
```

   `spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

4. **Air gapped installations only:** Add the credentials for your private registry to the queue manager YAML file:

```
spec:
  ...
  imagePullSecrets:
    - name: "mq-airgap-credentials"
```

   Note: For an air-gap installation, the custom resource YAML for each queue manager must include `spec.imagePullSecrets` if the private registry that contains the images requires credentials. This field must reference a Kubernetes secret that contains the private registry credentials. For more information, see the *Deploy an IBM MQ queue manager* step of Installing the IBM MQ Operator on Amazon EKS in an air-gap environment.

5. Create the queue manager:

```
kubectl apply -f mq-quickstart.yaml -n <NAMESPACE>
```

6. Verify the queue manager status:

```
kubectl get queuemanager quickstart -n <NAMESPACE>
```

OpenShift CP4I Examples for configuring a queue manager

A queue manager can be configured by adjusting the contents of the QueueManager custom resource.

About this task

Use the following examples to help you configure a queue manager using the QueueManager YAML file.

Procedure

- "Example: Supplying MQSC and INI files" on page 102
- "Example: Supplying additional files to a queue manager" on page 104
- "Example: Supplying additional environment variables for a queue manager" on page 106
- "Creating a self-signed PKI using OpenSSL" on page 107
- "Example: Configuring a queue manager with mutual TLS authentication" on page 109
- "Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift" on page 112
- "Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS" on page 114
- "Example: Customizing license service annotations" on page 117

OpenShift CP4I *Example: Supplying MQSC and INI files*

This example creates a Kubernetes ConfigMap that contains two MQSC files and one INI file. A queue manager is then deployed that processes these MQSC and INI files.

About this task

MQSC and INI files can be supplied when a queue manager is deployed. The MQSC and INI data must be defined in one or more Kubernetes ConfigMaps and Secrets. These must be created in the namespace (project) where you are deploying the queue manager.

Note: A Kubernetes Secret should be used when the MQSC or INI file contains sensitive data.

Example

The following example creates a Kubernetes ConfigMap that contains two MQSC files and one INI file. A queue manager is then deployed that processes these MQSC and INI files.

ConfigMap

Apply the following YAML in your cluster.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mqsc-ini-example
data:
  example1.mqsc: |
    DEFINE QLOCAL('DEV.QUEUE.1') REPLACE
    DEFINE QLOCAL('DEV.QUEUE.2') REPLACE
  example2.mqsc: |
    DEFINE QLOCAL('DEV.DEAD.LETTER.QUEUE') REPLACE
  example.ini: |
    Channels:
       MQIBindType=FASTPATH
```

QueueManager

Note: The following YAML uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.

Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See "mq.ibm.com/v1beta1: Current license versions" on page 281 for details on the license.

Example QueueManager for Red Hat OpenShift

Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: mqsc-ini-qm
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  web:
    enabled: true
  queueManager:
    name: "MQSCINI"
    mqsc:
    - configMap:
        name: mqsc-ini-example
        items:
        - example1.mqsc
        - example2.mqsc
    ini:
    - configMap:
        name: mqsc-ini-example
        items:
        - example.ini
    storage:
      queueManager:
        type: ephemeral
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: mqsc-ini-qm
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  web:
    enabled: true
    route:
      enabled: false
  queueManager:
    name: "MQSCINI"
    mqsc:
    - configMap:
        name: mqsc-ini-example
        items:
        - example1.mqsc
        - example2.mqsc
    ini:
    - configMap:
        name: mqsc-ini-example
        items:
        - example.ini
    storage:
      queueManager:
        type: ephemeral
```

```
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

Additional information:

- A queue manager can be configured to use a single Kubernetes ConfigMap or Secret (as shown in this example) or multiple ConfigMaps and Secrets.
- You can choose to use all of the MQSC and INI data from a Kubernetes ConfigMap or Secret (as shown in this example) or configure each queue manager to use only a subset of the available files.
- MQSC and INI files are processed in alphabetical order based on their key. So `example1.mqsc` is always processed before `example2.mqsc`, regardless of the order in which they appear in the queue manager configuration.
- If multiple MQSC or INI files have the same key, across multiple Kubernetes ConfigMaps or Secrets, then this set of files is processed based on the order in which the files are defined in the queue manager configuration.
- When a queue manager pod is running, any changes to the Kubernetes ConfigMap are not picked up because the IBM MQ Operator is not aware of the change. If you make changes to the ConfigMap, for example changes to the MQSC commands or to the INI files, then you must manually restart the queue managers to pick up those changes. For single instance queue managers, delete the pod to trigger the required restart. For Native HA deployments, restart the stand-by pods first by deleting them. When they are again in a running state, delete the active pod to restart it. This order of restarts ensures minimum downtime for the queue manager.

Example: Supplying additional files to a queue manager

This example creates a Kubernetes ConfigMap that contains two files. A queue manager is then deployed that mounts these files into the IBM MQ container in the specified locations.

About this task

Additional files can be supplied when a queue manager is deployed. These files must be defined in one or more Kubernetes ConfigMaps and Secrets. These must be created in the namespace (project) where you are deploying the queue manager. A Kubernetes Secret should be used when the files contain sensitive data.

Example

The following example creates a Kubernetes ConfigMap that contains two files. A queue manager is then deployed that mounts these files into the IBM MQ container in the specified locations.

ConfigMap

Apply the following YAML in your cluster.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: files-example
data:
  file1.txt: |
    Test File 1
  file2.txt: |
    Test File 2
```

QueueManager

Note: The following YAML uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.

Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See "mq.ibm.com/v1beta1: Current license versions" on page 281 for details on the license.

Example QueueManager for Red Hat OpenShift

Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: files-qm
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    files:
    - configMap:
        name: files-example
        defaultMountPath: "/etc/mqm/myfiles"
        items:
        - item: file1.txt
        - item: file2.txt
          mountPath: "/etc/mqm/otherfiles"
    storage:
      queueManager:
        type: ephemeral
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: files-qm
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    files:
    - configMap:
        name: files-example
        defaultMountPath: "/etc/mqm/myfiles"
        items:
        - item: file1.txt
        - item: file2.txt
          mountPath: "/etc/mqm/otherfiles"
    storage:
      queueManager:
        type: ephemeral
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and

`spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

File locations:

- The file `file1.txt` is mounted into location `/etc/mqm/myfiles/file1.txt`, which is the **defaultMountPath** specified for this ConfigMap.
- The file `file2.txt` is mounted into location `/etc/mqm/otherfiles/file2.txt`, which is the **mountPath** override for this specific file.

Additional information:

- When **defaultMountPath** is unset (for a ConfigMap or Secret) and the **mountPath** override is unspecified, then files are mounted by default into location `/etc/mqm/customfiles`
- If you change the contents of any of the files in the ConfigMap or Secret, then you must manually restart the queue managers to pick up those changes. For single instance queue managers, delete the pod to trigger the required restart. For Native HA deployments, restart the stand-by pods first by deleting them. When they are again in a running state, delete the active pod to restart it. This order of restarts ensures minimum downtime for the queue manager.

Example: Supplying additional environment variables for a queue manager

This example deploys a queue manager with two additional environment variables. These environment variables are set on the IBM MQ container.

About this task

Additional environment variables can be supplied when a queue manager is deployed. This allows the setting of environment variables that are not already exposed by the IBM MQ Operator.

Example

The following example deploys a queue manager with two additional environment variables. These environment variables are set on the IBM MQ container.

Note: The following YAML uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.

Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See "mq.ibm.com/v1beta1: Current license versions" on page 281 for details on the license.

Example QueueManager for Red Hat OpenShift

Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: env-qm
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    env:
    - name: TEST_ENV_1
      value: "true"
    - name: TEST_ENV_2
      value: "false"
    storage:
      queueManager:
        type: ephemeral
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: env-qm
spec:
  version: 10.0.0.0-r3
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    env:
    - name: TEST_ENV_1
      value: "true"
    - name: TEST_ENV_2
      value: "false"
    storage:
      queueManager:
        type: ephemeral
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

Additional information:

- If you specify an environment variable that would otherwise have been set by the IBM MQ Operator, this might lead to unpredictable behavior. In this scenario, a warning status condition is added to your QueueManager.

OpenShift CP4I AmazonEKS

Creating a self-signed PKI using OpenSSL

IBM MQ allows you to use mutual TLS for authentication, where both ends of a connection supply a certificate, and details in the certificate are used to establish an identity with the queue manager. This topic presents how to create an example Public Key Infrastructure (PKI) using the OpenSSL command line tool, creating two certificates which can be used in other examples.

Before you begin

Ensure that the OpenSSL command line tool is installed.

Install the IBM MQ client, and add `samp/bin` and `bin` to your *PATH*. You need the **runmqicred** command, which can be installed as part of the IBM MQ client as follows:

- Windows Linux For Windows and Linux: Install the IBM MQ redistributable client for your operating system from https://ibm.biz/mq94redistclients
- macOS For Mac: Download and set up the IBM MQ MacOS Toolkit: https://developer.ibm.com/tutorials/mq-macos-dev/

About this task

Important: The examples described here are not suitable for a production environment, and are solely intended as examples to get going quickly. Certificate management is a complex subject for advanced

users. For production, you must consider things like rotation, revocation, key length, disaster recovery and much more.

These steps have been tested using OpenSSL 3.1.4.

Procedure

1. Create a private key to use for your internal certificate authority

   ```
   openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out ca.key
   ```

   A private key for the internal certificate authority is created in a file called *ca.key*. This file should be kept safe and secret — it will be used to sign certificates for your internal certificate authority.

2. Issue a self-signed certificate for your internal certificate authority

   ```
   openssl req -x509 -new -nodes -key ca.key -sha512 -days 30 -subj "/CN=example-selfsigned-ca"
   -out ca.crt
   ```

   The `-days` specifies the number of days that the root CA certificate will be valid.

   A certificate is created in a file called *ca.crt*. This certificate contains the public information about the internal certificate authority, and is freely shareable.

3. Create a private key and certificate for a queue manager

   a) Create a private key and certificate signing request for a queue manager

   ```
   openssl req -new -nodes -out example-qm.csr -newkey rsa:4096 -keyout example-qm.key -subj
   '/CN=example-qm'
   ```

   A private key is created in a file called *example-qm.key*, and a certificate signing request is created in a file called *example-qm.csr*

   b) Sign the queue manager key with your internal certificate authority

   ```
   openssl x509 -req -in example-qm.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
   example-qm.crt -days 7 -sha512
   ```

   The `-days` specifies the number of days that the certificate will be valid.

   A signed certificate is created in a file called *example-qm.crt*

   c) Create a Kubernetes secret with the queue manager key and certificate

   ```
   kubectl create secret generic example-qm-tls --type="kubernetes.io/tls" --from-
   file=tls.key=example-qm.key --from-file=tls.crt=example-qm.crt --from-file=ca.crt
   ```

   A Kubernetes secret called *example-qm-tls* is created. This secret contains the private key for the queue manager, the public certificate, and the CA certificate.

4. Create a private key and certificate for an application

   a) Create a private key and certificate signing request for an application

   ```
   openssl req -new -nodes -out example-app1.csr -newkey rsa:4096 -keyout example-app1.key
   -subj '/CN=example-app1'
   ```

   A private key is created in a file called *example-app1.key*, and a certificate signing request is created in a file called *example-app1.csr*

   b) Sign the queue manager key with your internal certificate authority

   ```
   openssl x509 -req -in example-app1.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
   example-app1.crt -days 7 -sha512
   ```

   The `-days` specifies the number of days that the certificate will be valid.

   A signed certificate is created in a file called *example-app1.crt*

   c) Create a PKCS#12 key store with the application's key and certificate

IBM MQ uses a key database, and not individual key files. The containerized queue manager will create the key database for the queue manager from a Secret, but for clients applications, you need to manually create the key database.

```
openssl pkcs12 -export -in "example-app1.crt" -name "example-app1" -certfile "ca.crt"
-inkey "example-app1.key" -out "example-app1.p12" -passout pass:PASSWORD
```

Where *PASSWORD* is a password of your own choosing.

A key store is created in a file called *example-app1.p12*. The application's key and certificate is stored inside, with a "label" or "friendly name" of "example-app1", as well as the CA certificate.

d) If you are using an arm64 Apple Mac, then you need to configure an additional file combining the application and CA certificates.
For example:

```
cat example-app1.crt ca.crt > example-app1-chain.crt
```

Related tasks

"Example: Configuring a queue manager with mutual TLS authentication" on page 109
This example deploys a queue manager by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

"Example: Configuring Native HA using the IBM MQ Operator" on page 119
This example deploys a queue manager using the native high availability feature by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

"Configuring a multi-instance queue manager using the IBM MQ Operator" on page 165
This example deploys a multi-instance queue manager by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Example: Configuring a queue manager with mutual TLS authentication

This example deploys a queue manager by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Before you begin

To complete this example, you must first have completed the following prerequisites:

- Create a project/namespace for this example.
- On the command line, log into your cluster then switch to the namespace that you created.
- Ensure the IBM MQ Operator is installed and available in the namespace that you created.

About this task

This example provides a custom resource YAML defining a queue manager to be deployed. This example also details the additional steps required to deploy the queue manager with TLS enabled.

Procedure

1. Create a pair of certificates as described in "Creating a self-signed PKI using OpenSSL" on page 107.
2. Create a config map containing MQSC commands and an INI file

   Create a Kubernetes ConfigMap containing the MQSC commands to create a new queue and a SVRCONN channel, and to add a channel authentication record that allows access to the channel.

Ensure that you are in the namespace you created earlier (see Before you begin), then enter the following YAML by using the command line, or optionally by using the Red Hat OpenShift web console if you are using a Red Hat OpenShift cluster.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-tls-configmap
data:
  example-tls.mqsc: |
    DEFINE CHANNEL('MTLS.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS13_OR_HIGHER') REPLACE
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
ACTION(REPLACE)
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=example-app1') USERSRC(MAP)
MCAUSER('app1') ACTION(REPLACE)
    SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)
    DEFINE QLOCAL('EXAMPLE.QUEUE') REPLACE
    SET AUTHREC PROFILE('EXAMPLE.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(BROWSE,PUT,GET,INQ)
  example-tls.ini: |
    Service:
        Name=AuthorizationService
        EntryPoints=14
        SecurityPolicy=UserExternal
```

The MQSC defines a channel called *MTLS.SVRCONN* and a queue called *EXAMPLE.QUEUE*. The channel is configured to allow access only to clients which present a certificate with a "common name" of *example-app1*. This is the common name used in one of the certificates created in Step “1” on page 109. Connections on this channel with this common name are mapped to a user ID of *app1*, which is authorized to connect to the queue manager, and to access the example queue. The INI file enables a security policy which means that the *app1* user ID does not need to exist in an external user registry — it exists only as a name in this configuration.

3. Deploy the queue manager

   Create a new queue manager using custom resource YAML. Ensure you are in the namespace you created before you began this task.

   Note that the `example-qm-tls` Secret was created in Step “1” on page 109, and the `example-tls-configmap` ConfigMap was created in Step “2” on page 109.

   Note: The following YAML uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see “Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator” on page 187.

   Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

   Example QueueManager for Red Hat OpenShift

   Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    mqsc:
    - configMap:
        name: example-tls-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-tls-configmap
```

```
        items:
        - example-tls.ini
    storage:
      queueManager:
        type: ephemeral
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    mqsc:
    - configMap:
        name: example-tls-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-tls-configmap
        items:
        - example-tls.ini
    storage:
      queueManager:
        type: ephemeral
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

4. Confirm that the queue manager is running

   The queue manager is now being deployed. Confirm it is in `Running` state before proceeding. For example:

   - For deployments on the Red Hat OpenShift Container Platform:

     ```
     oc get qmgr exampleqm
     ```

- For deployments on Amazon EKS:

```
kubectl get qmgr exampleqm
```

5. Test the connection to the queue manager

   To confirm the queue manager is configured for mutual TLS communication, follow the steps in the appropriate link for your platform:

   - **Red Hat OpenShift**: "Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift" on page 112
   - **Amazon EKS**: "Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS" on page 114

Results

Congratulations, you have successfully deployed a queue manager with TLS enabled, and which uses details provided in the TLS certificate to authenticate with the queue manager and provide an identity.

OpenShift CP4I Linux *Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift*

After you have created a queue manager using the IBM MQ Operator on a Red Hat OpenShift cluster, you can test that it is working by connecting to it and putting and getting a message. This task takes you through how to connect using the IBM MQ sample programs, by running them on a machine outside the cluster, such as your laptop.

Before you begin

Note: For deployments on Amazon EKS, see "Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS" on page 114.

To complete this example, you must first have completed the following prerequisites:

- Install the IBM MQ client. You need the **amqsputc** and **amqsgetc** commands, which can be installed as part of the IBM MQ client as follows:
  - Windows Linux For Windows and Linux: Install the IBM MQ redistributable client for your operating system from https://ibm.biz/mq94redistclients
  - macOS For Mac: Download and set up the IBM MQ MacOS Toolkit: https://developer.ibm.com/tutorials/mq-macos-dev/
- Ensure you have the necessary key and certificate files downloaded to a directory on your machine, and that you know the key store password. For example, these files are created in "Creating a self-signed PKI using OpenSSL" on page 107:
  - `example-app1.p12`
  - `example-app1-chain.crt` (only if you're using an arm64 Apple Mac)
- Deploy a queue manager configured with TLS to the OCP cluster, for example by following the steps in "Example: Configuring a queue manager with mutual TLS authentication" on page 109

About this task

This example uses the IBM MQ sample programs running on a machine outside the Red Hat OpenShift cluster, such as your laptop, to connect to a QueueManager that is configured with TLS and to put and get messages.

Procedure

1. Confirm that the queue manager is running

Confirm that the queue manager is in Running state before proceeding. For example:

```
oc get qmgr EXAMPLEQM
```

2. Find the queue manager hostname

   Use the following command to find the queue manager fully-qualified hostname for the queue manager from outside the OCP cluster, using the route which is created automatically: exampleqm-ibm-mq-qm:

   ```
   oc get route exampleqm-ibm-mq-qm --template="{{.spec.host}}"
   ```

3. Create an IBM MQ Client Channel Definition Table (CCDT)

   Create a file called ccdt.json with the following contents:

   ```
   {
       "channel":
       [
           {
               "name": "MTLS.SVRCONN",
               "clientConnection":
               {
                   "connection":
                   [
                       {
                           "host": "HOSTNAME FROM PREVIOUS STEP",
                           "port": 443
                       }
                   ],
                   "queueManager": "EXAMPLEQM"
               },
               "transmissionSecurity":
               {
                 "cipherSpecification": "ANY_TLS13",
                 "certificateLabel": "example-app1"
               },
               "type": "clientConnection"
           }
      ]
   }
   ```

   The connection uses port 443, because that's the port the Red Hat OpenShift Container Platform router is listening on. The traffic will be forwarded to the queue manager on port 1414.

   If you have used a different channel name, adjust that too. The mutual TLS examples use a channel named MTLS.SVRCONN

   For more details, see Configuring a JSON format CCDT

4. Create an client INI file to configure the connection details

   Create a file called mqclient.ini in the current directory. This file will be read by **amqsputc** and **amqsgetc**.

   ```
   Channels:
     ChannelDefinitionDirectory=.
     ChannelDefinitionFile=ccdt.json
   SSL:
     OutboundSNI=HOSTNAME
     SSLKeyRepository=example-app1.p12
     SSLKeyRepositoryPassword=PASSWORD USED WHEN CREATING THE p12 FILE
   ```

   Set the value of SSLKeyRepositoryPassword to the password you chose when creating the PKCS#12 file. There are other ways to set the key store password, including using an encrypted password. For more information see Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows

   Note that the Red Hat OpenShift Container Platform Router uses SNI for routing requests to the IBM MQ queue manager. The OutboundSNI=HOSTNAME attribute ensures that the IBM MQ client includes the necessary information for the router to work with the default route configured by the IBM MQ

Operator. For more information, see “Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster ” on page 169.

5. If you are using an arm64 Apple Mac, you need to configure an additional environment variable.

```
export MQSSLTRUSTSTORE=example-app1-chain.crt
```

This file contains the full certificate chain, including the application and CA certificates.

6. Put messages to the queue

Run the following command:

```
/opt/mqm/samp/bin/amqsputc EXAMPLE.QUEUE EXAMPLEQM
```

If connection to the queue manager is successful, the following response is output:

```
target queue is EXAMPLE.QUEUE
```

Put several messages to the queue, by entering some text then pressing **Enter** each time.

To finish, press **Enter** twice.

7. Retrieve the messages from the queue

Run the following command:

```
/opt/mqm/samp/bin/amqsgetc EXAMPLE.QUEUE EXAMPLEQM
```

The messages you added in the previous step were consumed, and are output. After a few seconds, the command exits.

Results

You tested the connection to a queue manager with TLS enabled, and showed that you can securely put and get messages to the queue manager from a client.

Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS

After you have created a queue manager using the IBM MQ Operator on Amazon EKS, you can test that it is working by creating and configuring a `LoadBalancer`, then connecting to the queue manager and putting and getting a message. This task takes you through how to connect using the IBM MQ sample programs, by running them on a machine outside the cluster, such as your laptop.

Before you begin

Note: For deployments on Red Hat OpenShift, see “Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift” on page 112.

To complete this example, you must first have completed the following prerequisites:

- Install the AWS Load Balancer Controller onto your Amazon EKS cluster.
- Install the IBM MQ client to validate that your `LoadBalancer` is working correctly. For this, you need the need the **`amqsputc`** and **`amqsgetc`** commands, which can be installed as part of the IBM MQ client as follows:
  - Windows Linux For Windows and Linux: Install the IBM MQ redistributable client for your operating system from https://ibm.biz/mq94redistclients
  - macOS For Mac: Download and set up the IBM MQ MacOS Toolkit: https://developer.ibm.com/tutorials/mq-macos-dev/
- Ensure you have the necessary key and certificate files downloaded to a directory on your machine, and that you know the key store password. For example, these files are created in “Creating a self-signed PKI using OpenSSL” on page 107:

  - `example-app1.p12`
  - `example-app1-chain.crt` (only if you're using an arm64 Apple Mac)
- Deploy a queue manager configured with TLS in the same namespace as the IBM MQ Operator, for example by following the steps in "Example: Configuring a queue manager with mutual TLS authentication" on page 109
- To avoid having to specify the namespace in all commands, set your namespace to the one in which you deployed the queue manager. For example, by running the following command:

```
kubectl config set-context --current --namespace=QUEUE_MANAGER_NAMESPACE
```

About this task

This example uses the IBM MQ sample programs running on a machine outside the Kubernetes cluster such as your laptop, to connect to a QueueManager configured with TLS and to put and get messages.

This task does not cover configuring access to a queue manager's IBM MQ Console, REST APIs, or metrics from outside a cluster. You configure such access by using a Kubernetes `Ingress`, as described in "Configuring Ingress to connect to a queue manager IBM MQ Console and IBM MQ Operator REST API from outside an Amazon EKS cluster" on page 173.

Procedure

1. Confirm that the queue manager is running

   Confirm that the queue manager is in `Running` state before proceeding. For example:

   ```
   kubectl get qmgr EXAMPLEQM
   ```

2. Configure a `LoadBalancer` for your queue manager.

   To enable connections to your deployed queue manager, follow the steps in "Configuring a LoadBalancer to connect to a queue manager for Amazon EKS deployments" on page 171 for that queue manager.

3. Retrieve the `LoadBalancer` hostname.

   ```
   kubectl get service <LOAD_BALANCER_NAME> -o
   jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
   ```

4. Create an IBM MQ Client Channel Definition Table (CCDT).

   Create a file called `ccdt.json` in the current directory, with the following contents, setting the value of `host` to the hostname that you retrieved in the previous step:

   ```
   {
       "channel":
       [
           {
               "name": "MTLS.SVRCONN",
               "clientConnection":
               {
                   "connection":
                   [
                       {
                           "host": "HOSTNAME FROM PREVIOUS STEP",
                           "port": 1414
                       }
                   ],
                   "queueManager": "EXAMPLEQM"
               },
               "transmissionSecurity":
               {
                 "cipherSpecification": "ANY_TLS13",
                 "certificateLabel": "example-app1"
               },
               "type": "clientConnection"
           }
   ```

```
   ]
}
```

The `LoadBalancer Service` exposes the queue manager on port 1414.

If you used a different channel name, adjust that too. The mutual TLS examples use a channel named `MTLS.SVRCONN`.

For more details, see Configuring a JSON format CCDT

5. Create an client INI file to configure the connection details

   Create a file called `mqclient.ini` in the current directory. This file will be read by **amqsputc** and **amqsgetc**. Set the value of `SSLKeyRepositoryPassword` to the password that you chose when creating the PKCS#12 file.

```
Channels:
  ChannelDefinitionDirectory=.
  ChannelDefinitionFile=ccdt.json
SSL:
  OutboundSNI=HOSTNAME
  SSLKeyRepository=example-app1.p12
  SSLKeyRepositoryPassword=PASSWORD USED WHEN CREATING THE p12 FILE
```

   There are other ways to set the key store password, including using an encrypted password. For more information see Supplying the key repository password for an IBM MQ MQI client on AIX, Linux, and Windows

6. If you are using an arm64 Apple Mac, you need to configure an additional environment variable.

```
export MQSSLTRUSTSTORE=example-app1-chain.crt
```

   This file contains the full certificate chain, including the application and CA certificates.

7. Put messages to the queue.

   Run the following command to put a message:

```
/opt/mqm/samp/bin/amqsputc EXAMPLE.QUEUE EXAMPLEQM
```

   If connection to the queue manager is successful, the following response is output:

```
target queue is EXAMPLE.QUEUE
```

   Put several more messages to the queue, by entering some text then pressing **Enter** each time.

   To finish, press **Enter** twice.

8. Retrieve the messages from the queue by running the following command:

```
/opt/mqm/samp/bin/amqsgetc EXAMPLE.QUEUE EXAMPLEQM
```

   The messages that you added in the previous step were consumed, and are output. After a few seconds, the command exits.

Results

You tested the connection to a queue manager with TLS enabled, and showed that you can securely put and get messages to the queue manager from a client while passing the traffic through the `LoadBalancer Service`.

Example: Customizing license service annotations

The IBM MQ Operator automatically adds IBM License Service annotations to the deployed resources. These are monitored by the IBM License Service, and reports are generated that correspond to the required entitlement.

About this task

The annotations added by the IBM MQ Operator are those expected in standard situations, and are based on the license values selected during deployment of a queue manager.

Example

If **License** is set to `L-RJON-BZFQU2 (IBM Cloud Pak for Integration 2021.2.1)`, and **Use** is set to `NonProduction`, then the following annotations are applied:

- cloudpakId: c8b82d189e7545f0892db9ef2731b90d
- cloudpakName: IBM Cloud Pak for Integration
- productChargedContainers: qmgr
- productCloudpakRatio: '4:1'
- productID: 21dfe9a0f00f444f888756d835334909
- productName: IBM MQ Advanced for Non-Production Environment
- productMetric: VIRTUAL_PROCESSOR_CORE
- productVersion: 9.2.3.0

Within the IBM Cloud Pak for Integration, deployments of IBM App Connect Enterprise include a restricted entitlement for IBM MQ. In these situations, these annotations need to be overridden to assure that the IBM License Service captures the correct usage. To do this, use the approach described in “Adding custom annotations and labels to queue manager resources” on page 189.

For example, if IBM MQ is deployed under IBM App Connect Enterprise entitlement, use the approach shown in the following code fragment:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: mq4ace
  namespace: cp4i
spec:
  annotations:
    productMetric: FREE
```

There are two other common reasons why license annotations might require modification:

1. IBM MQ Advanced is included in the entitlement of another IBM product.
   - In this situation, use the approach previously described for IBM App Connect Enterprise.
2. IBM MQ is deployed under an IBM Cloud Pak for Integration license.
   - If you have an IBM Cloud Pak for Integration license, you can decide to deploy a queue manager either under the IBM MQ or IBM MQ Advanced ratio. If you deploy under an IBM MQ ratio, you must ensure that you do not use any advanced capabilities such as Native HA or Advanced Message Security.
   - In this situation, use the following annotations for production use:

     ```
     apiVersion: mq.ibm.com/v1beta1
     kind: QueueManager
     metadata:
       name: mq4ace
       namespace: cp4i
     spec:
       annotations:
         productID: c661609261d5471fb4ff8970a36bccea
     ```

```
    productCloudpakRatio: '4:1'
    productName: IBM MQ
    productMetric: VIRTUAL_PROCESSOR_CORE
```

- Use the following annotations for non-production use:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: mq4ace
  namespace: cp4i
spec:
  annotations:
    productID: 151bec68564a4a47a14e6fa99266deff
    productCloudpakRatio: '8:1'
    productName: IBM MQ Advanced for Non-Production Environment
    productMetric: VIRTUAL_PROCESSOR_CORE
```

OpenShift MQ Adv. Amazon EKS Configuring high availability for queue managers using the IBM MQ Operator

You can configure high availability for queue managers by using either the Native HA solution or the multi-instance queue manager solution.

About this task

Follow the procedure for your chosen high availability solution.

Procedure

- “Configuring Native HA using the IBM MQ Operator” on page 118.
- “Configuring Native HA CRR using the IBM MQ Operator” on page 123.
- “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.
- “Configuring a multi-instance queue manager using the IBM MQ Operator” on page 165.

OpenShift MQ Adv. Amazon EKS *Configuring Native HA using the IBM MQ Operator*

Native HA is configured using the `QueueManager` API, and advanced options are available using an INI file.

Native HA is configured using the `.spec.queueManager.availability` of the `QueueManager` API

Note: The following YAML uses an IBM MQ Advanced license. IBM MQ licensed queue managers can also be configured to use Native HA and Cross-Region Replication features by adding additional licenses to a queue manager. For more information, see “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.

Example QueueManager for Red Hat OpenShift

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: nativeha-example
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    availability:
      type: NativeHA
  version: 10.0.0.0-r3
```

Example QueueManager for Amazon EKS

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
```

```
metadata:
  name: nativeha-example
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    availability:
      type: NativeHA
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  version: 10.0.0.0-r3
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because they are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

The `.spec.queueManager.availability.type` field must be set to `NativeHA`.

Under `.spec.queueManager.availability`, you can also configure a TLS secret and ciphers to use between queue manager instances when replicating. This is strongly recommended, and a step-by-step guide is available in "Example: Configuring Native HA using the IBM MQ Operator" on page 119.

Related tasks

"Example: Configuring Native HA using the IBM MQ Operator" on page 119
This example deploys a queue manager using the native high availability feature by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Example: Configuring Native HA using the IBM MQ Operator

This example deploys a queue manager using the native high availability feature by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Before you begin

To complete this example, you must first have completed the following prerequisites:

- Create a project/namespace for this example on your cluster.
- On the command line, log into your cluster, then switch to the above namespace.
- Ensure the IBM MQ Operator is installed and available in the above namespace.

About this task

This example provides a custom resource YAML defining a queue manager to be deployed. It also details the additional steps required to deploy the queue manager with TLS enabled.

Procedure

1. Create a pair of certificates as described in "Creating a self-signed PKI using OpenSSL" on page 107.
2. Create a config map containing MQSC commands and an INI file

   Create a Kubernetes ConfigMap containing the MQSC commands to create a new queue and a SVRCONN channel, and to add a channel authentication record that allows access to the channel.

Ensure you are in the namespace you created earlier (see Before you begin), then enter the following YAML in the OCP web console, or using the command line.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-nativeha-configmap
data:
  example-tls.mqsc: |
    DEFINE CHANNEL('MTLS.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS13_OR_HIGHER') REPLACE
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
ACTION(REPLACE)
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=example-app1') USERSRC(MAP)
MCAUSER('app1') ACTION(REPLACE)
    SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)
    DEFINE QLOCAL('EXAMPLE.QUEUE') REPLACE
    SET AUTHREC PROFILE('EXAMPLE.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(BROWSE,PUT,GET,INQ)
  example-tls.ini: |
    Service:
        Name=AuthorizationService
        EntryPoints=14
        SecurityPolicy=UserExternal
```

The MQSC defines a channel called `MTLS.SVRCONN` and a queue called `EXAMPLE.QUEUE`. The channel is configured to allow access only to clients which present a certificate with a "common name" of `example-app1`. This is the common name used in one of the certificates created in Step “1” on page 119. Connections on this channel with this common name are mapped to a user ID of `app1`, which is authorized to connect to the queue manager, and to access the example queue. The INI file enables a security policy which means that the `app1` user ID does not need to exist in an external user registry — it exists only as a name in this configuration.

3. Deploy the queue manager

   Create a new queue manager by using the following custom resource YAML. Ensure you are in the namespace that you created before you begin this task.

   The availability type is set to `NativeHA` and persistent storage is selected. The default storage class configured in your Kubernetes cluster will be used. If you do not have a storage class configured as default, or you want to use a different storage class, add `defaultClass: storage_class_name` under `spec.queueManager.storage`.

   The three pods in a Native HA queue manager replicate data over the network. This link is not encrypted by default, but this example uses the queue manager's certificate for encrypting traffic. You can specify a different certificate for additional security. The Native HA TLS Secret must be a Kubernetes TLS Secret, which has a particular structure (for example, the private key must be called `tls.key`).

   Note that the `example-qm-tls` Secret was created in Step “1” on page 119, and the `example-nativeha-configmap` ConfigMap was created in Step “2” on page 119

   Note:

   The following YAML uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.

   Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

   Example QueueManager for Red Hat OpenShift

   Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

   ```
   apiVersion: mq.ibm.com/v1beta1
   kind: QueueManager
   metadata:
     name: exampleqm
   ```

```
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: example-qm-tls
    mqsc:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.ini
    storage:
      queueManager:
        type: persistent-claim
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: example-qm-tls
    mqsc:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.ini
    storage:
      queueManager:
        type: persistent-claim
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
```

```
            - tls.crt
            - ca.crt
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

4. Confirm that the queue manager is running

   The queue manager is now being deployed. Confirm it is in `Running` state before proceeding. For example:

   - For deployments on the Red Hat OpenShift Container Platform:

     ```
     oc get qmgr exampleqm
     ```

   - For deployments on Amazon EKS:

     ```
     kubectl get qmgr exampleqm
     ```

5. Test the connection to the queue manager

   To confirm the queue manager is configured for mutual TLS communication, follow the steps in the appropriate link for your platform:

   - **Red Hat OpenShift**: "Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift" on page 112
   - **Amazon EKS**: "Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS" on page 114

     Important: Amazon EKS users must complete the steps to "Configuring a LoadBalancer to connect to a queue manager for Amazon EKS deployments" on page 171 before continuing on this page.

6. Force the active pod to fail

   To validate the automatic recovery of the queue manager, simulate a pod failure:

   a) View the active and standby pods

      Run the following command:

      - For deployments on the Red Hat OpenShift Container Platform:

        ```
        oc get pods --selector app.kubernetes.io/instance=exampleqm
        ```

      - For deployments on Amazon EKS:

        ```
        kubectl get pods --selector app.kubernetes.io/instance=exampleqm
        ```

      Note that, in the **READY** field, the active pod returns the value `1/1`, whereas the standby pod returns the value `0/1`.

   b) Delete the active pod

      Run the following command, specifying the full name of the active pod:

      - For deployments on the Red Hat OpenShift Container Platform:

        ```
        oc delete pod exampleqm-ibm-mq-<VALUE>
        ```

      - For deployments on Amazon EKS:

        ```
        kubectl delete pod exampleqm-ibm-mq-<VALUE>
        ```

   c) View the pod status again (see step "6.a" on page 122).

   d) View the queue manager status

Run the following command, specifying the full name of the <ACTIVE_POD>. The active pod should be in ready state (this is the pod with the READY value of 1/1 shown by the pod status command in the previous step):

- For deployments on the Red Hat OpenShift Container Platform:

```
oc exec -t <ACTIVE_POD> -- dspmq -o nativeha -x -m EXAMPLEQM
```

- For deployments on Amazon EKS:

```
kubectl exec -t <ACTIVE_POD> -- dspmq -o nativeha -x -m EXAMPLEQM
```

You should see the status shows that the active instance has changed, for example:

```
QMNAME(EXAMPLEQM) ROLE(Active) INSTANCE(inst1) INSYNC(Yes) QUORUM(3/3)
INSTANCE(inst1) ROLE(Active) REPLADDR(9.20.123.45) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATE(2022-01-12)       ALTTIME(12.03.44)
INSTANCE(inst2) ROLE(Replica) REPLADDR(9.20.123.46) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATE(2022-01-12)       ALTTIME(12.03.44)
INSTANCE(inst3) ROLE(Replica) REPLADDR(9.20.123.47) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATE(2022-01-12)       ALTTIME(12.03.44)
```

e) Test the connection to the queue manager again

Note: You do not need to repeat the steps to configure the connection, only those to put and receive messages to or from the queue

To confirm the queue manager has recovered, follow the steps in the appropriate link for your platform:

- **Red Hat OpenShift**: "Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift" on page 112
- **Amazon EKS**: "Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS" on page 114

Results

You deployed a queue manager with native high availability and mutual TLS authentication, and verified that it automatically recovers when the active Pod fails.

Configuring Native HA CRR using the IBM MQ Operator

Native HA Cross-Region Replication (CRR) is configured using the `QueueManager` API, and advanced options are available using an INI file.

Note: The following YAML uses an IBM MQ Advanced license. IBM MQ licensed queue managers can also be configured to use Native HA and Cross-Region Replication features by adding additional licenses to a queue manager. For more information see "Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator" on page 124.

Native HA CRR is configured by using `.spec.queueManager.availability` in the `QueueManager` API. The `.spec.queueManager.availability.type` field must be set to `NativeHA`.
The `.spec.queueManager.availability.nativeHAGroups.local.role` field is set to `Live` or `Recovery` to specify the role of the local group.

You specify the details of the remote group in the configuration in the `.spec.queueManager.availability.nativeHAGroups.remotes` section. On Red Hat OpenShift, you specify the hostname and port of the Red Hat OpenShift route that is used by the remote group where data will be replicated to. On Amazon EKS, you specify the hostname and port of the LoadBalancer (which you must configure) that is used by the remote group where data will be replicated to. The live group must include at least the `remotes` section to be able to replicate to the recovery group. The recovery group must include the `remotes` section to be able to replicate back to the other site if it ever becomes live. (See "Native HA CRR switchover and failover" on page 264.)

Under `.spec.queueManager.availability`, you can also configure a TLS secret and ciphers to use between queue manager instances when replicating. This is strongly recommended, and a step-by-step guide is available in "Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator" on page 125.

You must configure a TLS secret and ciphers to use between the live and recovery groups when replicating. You can use the same secret that you specified for replication between instances, or you can specify a different secret.

Related tasks

"Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator" on page 125
This example deploys a queue manager using a simple Native HA Cross-Region Replication (CRR) configuration using the IBM MQ Operator.

Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator

IBM MQ licensed queue managers can be configured to use Native HA Cross-Region Replication (CRR) features by adding an additional license to a queue manager.

Before you begin

Note: These instructions apply only if you are using the IBM MQ Operator. If you are self-deploying (in other words, without using the IBM MQ Operator) queue managers on Kubernetes, see "Configuring Native HA and Cross-Region Replication on IBM MQ licensed self-deployed queue managers in Kubernetes" on page 213 instead.

The IBM License Service is required to use Native HA CRR features on IBM MQ licensed queue managers. See the *Deploy the License Service* step of the IBM MQ Operator installation process that is appropriate for your platform:

- Red Hat OpenShift
- Amazon EKS.

Important: This task involves updating the queue manager YAML, which triggers a pod restart.

About this task

This task varies depending on the version of your IBM MQ Operator. IBM MQ Operator versions from 3.7.0 can do the required configuration for you. For earlier versions of the operator, you have to add additional licenses to the IBM MQ queue manager yourself.

Procedure

- Add the following information when you deploy a queue manager:
  - Specify an IBM MQ license in the `spec.license.license` field.
  - Specify an IBM MQ Native HA and Cross-Region Replication license in the `spec.license.additionalLicenses` field.

  For more information about available licenses, see "Licensing and API references for the IBM MQ Operator" on page 281.

  The IBM MQ Operator creates an `IBMLicensingDefinion` CR for each queue manager that is configured to use Native HA CRR features with an IBM MQ license. This CR enables tracking of the additional license by the IBM License Service.

  The following example shows how to configure a queue manager with an IBM MQ license and an IBM MQ Native HA and Cross-Region Replication additional license, for production use:

```
...
spec:
  license:
```

```
    accept: false
    license: L-MKDD-7KHY2Q
    additionalLicenses:
      - L-KFCR-53UGK9
    use: Production
...
```

Important: If you are upgrading a queue manager that was deployed with an IBM MQ Operator prior to version 3.7.0, and want to use the `spec.license.additionalLicenses` field, your upgrade is blocked until you remove the `spec.annotations.additionalLicense` field that you previously specified in the queue manager's YAML.

Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator

This example deploys a queue manager using a simple Native HA Cross-Region Replication (CRR) configuration using the IBM MQ Operator.

Before you begin

To complete this example, you must first complete the following steps:

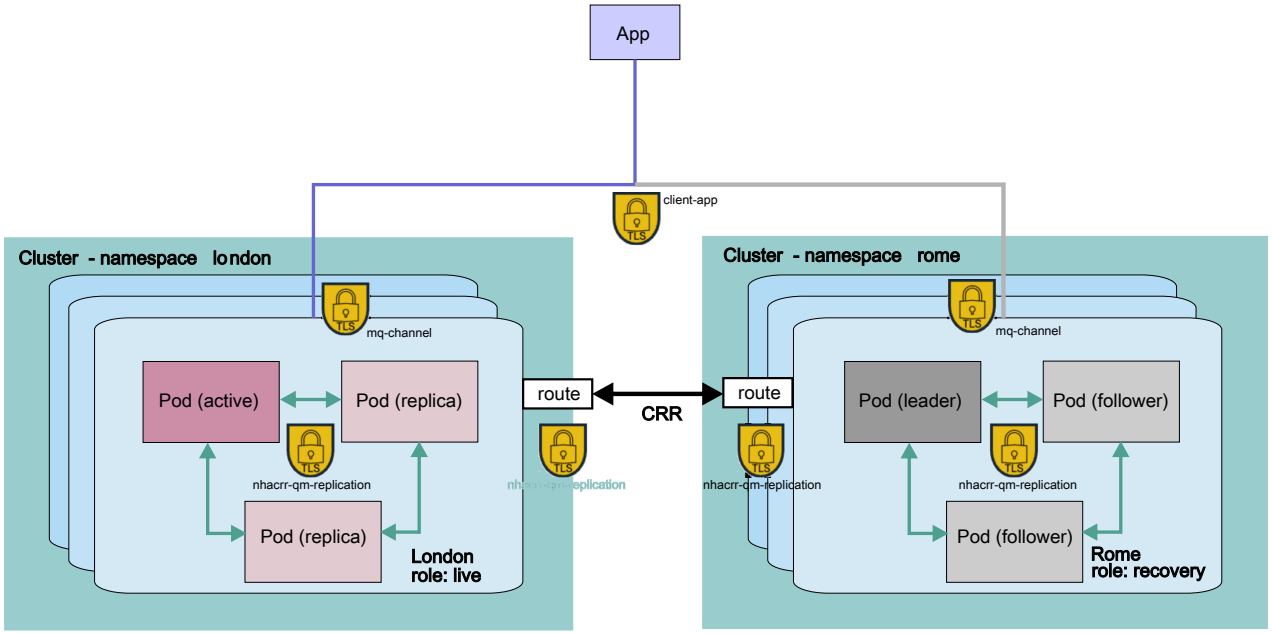
- On the command line, log into the clusters that you are using.
- Create two projects/namespaces for this example. The example will refer to the London and Rome sites. These refer to either two namespaces for London and Rome in a single cluster, or a namespace in each cluster for each site.
- Ensure the IBM MQ Operator is installed in the clusters and available in the namespaces.
- **Amazon EKS deployments only**: Install the AWS Load Balancer Controller onto your Amazon EKS cluster.
- Ensure that the OpenSSL command line tool is installed.
- Complete the “Preparing to use Kubernetes by creating a pull secret” on page 86 task.
- Read “Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97 or “Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS” on page 100 so that you are familiar with using the IBM MQ operator.

About this task

The Native HA CRR configuration defines two Native HA groups, which together form a highly available queue manager. These groups are usually deployed in two clusters in different locations, however this example allows you to deploy both groups to the same cluster if desired. See Native HA Cross-Region Replication for an overview of Native HA CRR.

For simplicity, this example uses a single certificate to secure all the queue manager replication traffic, this is named nhacrr-qm-replication.

To understand how certificates are used in Native HA CRR, and how to use different certificates to secure group and instance traffic, see “Example: using multiple TLS certificates with Native HA CRR” on page 147.



Important: These examples are intended to get you started, they are not suitable for a production environment. Certificate management is a complex subject, for advanced users. For production, you must consider things like rotation, revocation, key length, disaster recovery, and much more.

This example provides the custom resource YAML to define two Native HA groups providing cross-region replication.

Procedure

1. Create the certificates to use for the queue manager replication traffic.

   Use the following commands to create a private key and use it to issue a self-signed certificate for your internal certificate authority. Then create a private key and certificate for the queue manager, signed with your internal certificate authority:

   ```
   openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out nhacrr-ca.key
   openssl req -x509 -new -nodes -key "nhacrr-ca.key" -sha512 -days 30 -subj "/CN=nhacrr-
   selfsigned-ca" -out nhacrr-ca.crt
   openssl req -new -nodes -out nhacrr-qm-replication.csr -newkey rsa:4096 -keyout nhacrr-qm-
   replication.key -subj "/CN=nhacrr-qm-replication"
   openssl x509 -req -in nhacrr-qm-replication.csr -CA nhacrr-ca.crt -CAkey nhacrr-ca.key
   -CAcreateserial -out nhacrr-qm-replication.crt -days 7 -sha512
   ```

   For a more information about these commands, see “Creating a self-signed PKI using OpenSSL” on page 107.

2. Create two Kubernetes secrets containing the queue manager key and certificates. Create a secret named nhacrr-qm-replication on both the Rome and London sites, to store the key and certificate for the queue manager instances.

   Use the following command to create the secrets. Change to each namespace (Rome then London) before running the command.

   ```
   kubectl create secret generic nhacrr-qm-replication --type="kubernetes.io/tls" --from-
   file=tls.key=nhacrr-qm-replication.key --from-file=tls.crt=nhacrr-qm-replication.crt --from-
   file=nhacrr-ca.crt
   ```

3. Create a pair of certificates for the queue manager traffic as described in “Creating a self-signed PKI using OpenSSL” on page 107. Create the secret on both the Rome and London sites (these are the certificates named mq-channel and client-app in the diagram).

4. Create a config map containing MQSC commands and an INI file for Rome and London. Create a ConfigMap resource containing the MQSC commands to create a new persistent message queue and a SVRCONN channel, and to add a channel authentication record that allows access to the channel.

   On the Rome site, create the following ConfigMap then apply it by using the `oc apply` command on Red Hat OpenShift, or the `kubectl apply` command on Amazon EKS. On Red Hat OpenShift, you can alternatively enter the YAML in the OCP web console by selecting **ConfigMaps** then **Create ConfigMap**. Repeat this step for the London site.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: nhacrr-configmap
data:
  nhacrr.mqsc: |
    DEFINE CHANNEL('MTLS.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS13_OR_HIGHER') REPLACE
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
ACTION(REPLACE)
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=example-app1') USERSRC(MAP)
MCAUSER('app1') ACTION(REPLACE)
    SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)
    DEFINE QLOCAL('EXAMPLE.QUEUE') DEFPSIST(YES) REPLACE
    SET AUTHREC PROFILE('EXAMPLE.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(BROWSE,PUT,GET,INQ)
  nhacrr.ini: |
    Service:
        Name=AuthorizationService
        EntryPoints=14
        SecurityPolicy=UserExternal
```

   The MQSC defines a channel named MTLS.SVRCONN and a queue named EXAMPLE.QUEUE. The channel is configured to allow access only to clients which present a certificate with a CN of example-app1 like the one created in Step 3.

5. Deploy the Native HA Recovery group. The Recovery group is deployed to the Rome site. Ensure that you are in the Rome namespace, then create the QueueManager.

   Note: You must use the same queue manager name in both live and recovery groups. By default, if you do not specify `.spec.queuemanager.name`, it defaults to `.metadata.name`. You can achieve matching names by using one of the following methods:

   - Name your queue manager resource the same in each cluster (that is, `.metadata.name` matches) and do not override the name
   - Name your queue manager individually in `.metadata.name` and override the name with `.spec.queuemanager.name` so that names match in both sites.

   In this example, the queue manager name is set to `exampleqm` by using `.metadata.name`.

   Note: This example uses an IBM MQ Advanced license. IBM MQ licensed queue managers can also be configured to use Native HA and Cross-Region Replication features by adding additional licenses to a queue manager. For more information, see “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.

   Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

   Example QueueManager for Red Hat OpenShift

   Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
  labels:
    role: Recovery
spec:
  license:
    accept: false
```

```
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: nhacrr-qm-replication
      nativeHAGroups:
        local:
          name: "rome"
          role: "Recovery"
    mqsc:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.mqsc
    ini:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.ini
    storage:
      queueManager:
        type: persistent-claim
  version:  10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
  labels:
    role: Recovery
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: nhacrr-qm-replication
      nativeHAGroups:
        local:
          name: "rome"
          role: "Recovery"
          route:
            enabled: false
    mqsc:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.mqsc
    ini:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.ini
    storage:
      queueManager:
        type: persistent-claim
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
```

```
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.availability.nativeHAGroups.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

6. Verify that the Recovery group has deployed correctly:

   a) Issue the relevant command to display the queue manager resource then wait for the queue manager to be running.

   - On Red Hat OpenShift:

     ```
     oc get qmgr exampleqm
     ```

   - On Amazon EKS:

     ```
     kubectl get qmgr exampleqm
     ```

     You should see the following message:

     ```
     NAME             PHASE
     exampleqm   Running
     ```

   b) When the queue manager is running, check the pod states by using the relevant command.

   - On Red Hat OpenShift:

     ```
     oc get pods
     ```

   - On Amazon EKS:

     ```
     kubectl get pods
     ```

     One pod should be in ready state:

     ```
     NAME                 READY   STATUS    RESTARTS   AGE
     exampleqm-ibm-mq-0   1/1     Running   0          2m45s
     exampleqm-ibm-mq-1   0/1     Running   0          2m45s
     exampleqm-ibm-mq-2   0/1     Running   0          2m45s
     ```

   c) Check the status of the recovery group by using the relevant command, where <LEADER-POD> is the name of the ready 1/1 pod (`exampleqm-ibm-mq-0` in the example).

   - On Red Hat OpenShift:

     ```
     oc exec -t <LEADER-POD> -- dspmq -o nativeha -g
     ```

   - On Amazon EKS:

     ```
     kubectl exec -t <LEADER_POD> -- dspmq -o nativeha -g
     ```

You should see the following output:

```
QMNAME(EXAMPLEQM)                                         ROLE(Leader)
INSTANCE(exampleqm-ibm-mq-0) INSYNC(yes)   QUORUM(1/3) GRPLSN() GRPNAME(rome)
GRPROLE(Recovery)
 GRPNAME(rome) GRPROLE(Recovery) GRPADDR(Unknown) GRPVER(9.4.2.0) GRSTATUS(Waiting
for connection) RCOVLSN()   RCOVTIME() BACKLOG(Unknown) INSYNC(Unknown) SYNCTIME()
ALTDATE(2025-04-25) ALTTIME(13.48.15)
```

Key parts of this status are as follows:

- `Role(Leader)`
- `QUORUM(1/3)` - this remains 1/3 until the Live and Recovery groups connect
- `GRSTATUS(Waiting for connection)` - this updates when the Live group connects

7. **Amazon EKS deployments only**: Configure a `LoadBalancer` to allow the Live group to communicate with the Recovery group.
   a) Ensure you are in the Rome namespace, then create the `LoadBalancer` `Service` to facilitate a TLS connection to a queue manager from outside your cluster with the following configuration.

```
apiVersion: v1
kind: Service
metadata:
  name: crr-rome
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
spec:
  type: LoadBalancer
  ports:
    - port: 9415
      targetPort: 9415
      name: nha-crr
  selector:
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/instance: exampleqm
```

   Important: If you are using an **IPv6 single-stack** system, make the following changes to the `metadata.annotations` section:

   - Change the value of `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type` from `"instance"` to `"ip"`
   - Add the following line:

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: "dualstack"
```

   b) Verify that the `Service` of type `LoadBalancer` was created:

```
kubectl get service <LOAD_BALANCER_NAME>
```

   c) Check that your `LoadBalancer` is ready.

   The Amazon Load Balancer Controller takes several minutes to provision and register its targets. Run the following command to show the status of the `LoadBalancer`. When the `LoadBalancer` is ready, the status is `active`:

```
aws elbv2 describe-load-balancers --query 'LoadBalancers[*].
[LoadBalancerName,State.Code]' --output text
```

8. Deploy the Native HA Live group. The Live group is deployed to the London site:

   Ensure that you are in the correct namespace, then create the QueueManager.

   Note:

   This example uses an IBM MQ Advanced license. IBM MQ licensed queue managers can also be configured to use Native HA and Cross-Region Replication features by adding additional licenses to a

queue manager. For more information, see “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.

Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

Example QueueManager for Red Hat OpenShift

Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
  labels:
    role: Live
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: nhacrr-qm-replication
      nativeHAGroups:
        local:
          name: "london"
          role: "Live"
        remotes:
          - name: "rome"
            addresses:
              - "<rome group hostname>:443"
    mqsc:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.mqsc
    ini:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.ini
    storage:
      queueManager:
        type: persistent-claim
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Update the address in the YAML that specifies the address that the live group uses to connects to the recovery group (`"<rome group hostname>:443"` in the example). Replace it with the host attribute of the `exampleqm-ibm-mq-nhacrr` route on the Rome site. To retrieve this attribute, ensure you are on the Rome site and use the following command to retrieve the `nhacrr` route hostname:

```
oc get route exampleqm-ibm-mq-nhacrr --template="{{.spec.host}}"
```

Note: There are two routes created when the CRR groups are deployed, one ending `qm` and one ending `nhacrr`. The `qm` route is the one used to send messages to the queue manager, the `nhacrr` route is used for group-to-group communication. Always add the host addresses of the `nhacrr` route as the remote address for the group in the YAML.

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
  labels:
    role: Live
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: nhacrr-qm-replication
      nativeHAGroups:
        local:
          name: "london"
          role: "Live"
          route:
            enabled: false
        remotes:
          - name: "rome"
            addresses:
              - "<rome group hostname>:9415"
    mqsc:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.mqsc
    ini:
    - configMap:
        name: nhacrr-configmap
        items:
        - nhacrr.ini
    storage:
      queueManager:
        type: persistent-claim
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
  web:
    enabled: false
    route:
      enabled: false
```

Update the address in the YAML that specifies the address that the live group uses to connects to the recovery group ("<rome group hostname>:9415" in the example). Replace it with the host attribute of the LoadBalancer on the Rome site (which you created in the previous step). To retrieve this attribute, ensure that you are on the Rome site then use the following command to retrieve the nhacrr LoadBalancer hostname:

```
kubectl get service <LOAD_BALANCER_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

spec.queueManager.route.enabled,
spec.queueManager.availability.nativeHAGroups.route.enabled,
spec.queueManager.metrics.serviceMonitor.enabled, and

`spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

9. Verify the Live group and the Cross Region Replication. Repeat the actions in step 6, but this time for the London group.

   a) Issue the relevant command to display the queue manager resource then wait for the queue manager to be running.

   - On Red Hat OpenShift:

     ```
     oc get qmgr exampleqm
     ```

   - On Amazon EKS:

     ```
     kubectl get qmgr exampleqm
     ```

   You should see the following message:

   ```
   NAME            PHASE
   exampleqm   Running
   ```

   b) When the queue manager is running, check the pod states by using the relevant command.

   - On Red Hat OpenShift:

     ```
     oc get pods
     ```

   - On Amazon EKS:

     ```
     kubectl get pods
     ```

   One pod should be in ready state:

   ```
   NAME                READY   STATUS    RESTARTS   AGE
   exampleqm-ibm-mq-0   1/1     Running   0          2m45s
   exampleqm-ibm-mq-1   0/1     Running   0          2m45s
   exampleqm-ibm-mq-2   0/1     Running   0          2m45s
   ```

   c) Check the status of the recovery group by using the following command, where <ACTIVE-POD> is the name of the ready 1/1 pod (`exampleqm-ibm-mq-0` in the example).

   - On Red Hat OpenShift:

     ```
     oc exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
     ```

   - On Amazon EKS:

     ```
     kubectl exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
     ```

   ```
   oc exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
   ```

   You should see the following output:

   ```
   QMNAME(EXAMPLEQM)                                         ROLE(Active)
   INSTANCE(exampleqm-ibm-mq-0) INSYNC(yes)   QUORUM(3/3)       GRPLSN(<0:0:87:58322>)
   GRPNAME(london) GRPROLE(Live)
    GRPNAME(london) GRPROLE(Live) GRPADDR(Unknown) GRPVER(9.4.2.0)
   GRSTATUS(Normal) RCOVLSN(<0:0:87:58322>)   RCOVTIME      (2025-05-16T14:27:09.051799Z)
   INITLSN(<0:0:9:4696>) INITTIME(2025-05-12T16:58:13.582274Z) LIVETIME
   (2025-05-12T16:58:16.      319844Z) ALTDATE(2025-05-16) ALTTIME(14.27.10)
    GRPNAME(rome) GRPROLE(Recovery) GRPADDR(exampleqm-ibm-mq-nhacrr-
   <namespace>.apps.<cluster>) GRPVER(9.4.2.0) CONNGRP  (yes)       GRSTATUS(Normal)
   RCOVLSN(<0:0:87:58322>) RCOVTIME(2025-05-16T14:27:09.051799Z) BACKLOG(0) INSYNC(yes)
   SYNCTIME      (2025-05-16T14:28:38.084565Z) ALTDATE(2025-05-16) ALTTIME(14.28.31)
   ```

   Key parts of this status are as follows:

   - `Role(Active)`

- `QUORUM(3/3)`: the Live group has full quorum
- `GRSTATUS(Normal)`

d) Check the status of the Recovery group again by running the relevant command on the Rome site, where <LEADER_POD> is the name of the `READY 1/1` pod.

- On Red Hat OpenShift:

```
oc exec -t <LEADER_POD> -- dspmq -o nativeha -g
```

- On Amazon EKS:

```
kubectl exec -t <LEADER_POD> -- dspmq -o nativeha -g
```

You should see the following output:

```
QMNAME(EXAMPLEQM)                                         ROLE(Leader)
INSTANCE(exampleqm-ibm-mq-0) INSYNC(yes)   QUORUM(3/3)       GRPLSN(<0:0:87:58322>)
GRPNAME(rome) GRPROLE(Recovery)
 GRPNAME(rome) GRPROLE(Recovery) GRPADDR(exampleqm-ibm-mq-nhacrr-
<namespace>.apps.<cluster>) GRPVER(9.4.3.0)   GRSTATUS      (Normal)
RCOVLSN(<0:0:87:58322>) RCOVTIME(2025-05-16T14:27:09.051799Z) BACKLOG(0) INSYNC(yes)
SYNCTIME(2025-05-16T14:30:40.      134627Z) ALTDATE(2025-05-16) ALTTIME(14.27.17)
 GRPNAME(london) GRPROLE(Live) GRPADDR(Unknown) GRPVER(9.4.3.0) CONNGRP(yes)
GRSTATUS(Normal) RCOVLSN  (<0:0:87:58322>)       RCOVTIME(2025-05-16T14:27:09.051799Z)
INITLSN(<0:0:9:4696>) INITTIME(2025-05-12T16:58:13.  582274Z) LIVETIME
(2025-05-12T16:58:16.319844Z) ALTDATE(2025-05-16) ALTTIME(14.30.41)
```

Note that the `QUORUM(3/3)` and `GRSTATUS(Normal)` fields have updated.

10. Test the connection to the queue manager.

    To confirm the queue manager is configured and available, follow the steps in the appropriate link for your platform:

    - **Red Hat OpenShift**: "Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift" on page 112. Ensure that the queue manager hostname, which is determined in step 2 of the linked page, is the hostname of the route in the London site.
    - **Amazon EKS**: "Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS" on page 114.

11. **Amazon EKS deployments only**: Configure a `LoadBalancer` to allow the Recovery group to communicate with the Live group.

    a) Ensure you are in the London namespace, then create the `LoadBalancer Service` to facilitate a TLS connection to a queue manager from outside your cluster with the following configuration.

```
apiVersion: v1
kind: Service
metadata:
  name: crr-london
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
spec:
  type: LoadBalancer
  ports:
    - port: 9415
      targetPort: 9415
      name: nha-crr
  selector:
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/instance: exampleqm
```

Important: If you are using an **IPv6 single-stack** system, make the following changes to the `metadata.annotations` section:

- Change the value of `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type` from `"instance"` to `"ip"`
- Add the following line:

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: "dualstack"
```

b) Verify that the `Service` of type `LoadBalancer` was created:

```
kubectl get service <LOAD_BALANCER_NAME>
```

c) Check that your `LoadBalancer` is ready.

The Amazon Load Balancer Controller takes several minutes to provision and register its targets. Run the following command to show the status of the `LoadBalancer`. When the `LoadBalancer` is ready, the status is `active`:

```
aws elbv2 describe-load-balancers --query 'LoadBalancers[*].
[LoadBalancerName,State.Code]' --output text
```

12. Add a remote address to the Native HA Recovery group.

When Rome becomes the live group, it needs to know London's remote address so it can connect to London as the recovery group. To add this, update the `spec.queueManager.availability.nativeHAGroups` for Rome site with the following addition, below the local stanza:

On Red Hat OpenShift

```
          remotes:
            - name: "london"
              addresses:
              - "<london group hostname>:443"
```

Replace the address with the host attribute of the `exampleqm-ibm-mq-nhacrr` route on the London site. Ensure you are on the London site then use the following command to retrieve the nhacrr route's hostname:

```
oc get route exampleqm-ibm-mq-nhacrr --template="{{.spec.host}}"
```

After the update is made, wait for the pods to restart, be running and a pod to be the `READY 1/1` pod.

On Amazon EKS

```
          remotes:
            - name: "london"
              addresses:
              - "<london group hostname>:9415"
```

Replace the address with the host attribute of the `LoadBalancer` on the London site. Ensure you are on the London site then use the following command to retrieve the nhacrr `LoadBalancer` hostname:

```
kubectl get service <LOAD_BALANCER_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

After the update is made, wait for the pods to restart, be running and a pod to be the `READY 1/1` pod.

13. Add the Rome site hostname to the `ccdt.json` file.

After a switchover, where Rome becomes the live site, clients need to connect to the group in Rome. You enable this behavior by adding the hostname for Rome to the `ccdt.json` file.

On Red Hat OpenShift

a. Retrieve the `qm` hostname on the Rome site. Ensure you are on the Rome site then use the following command to retrieve the `qm` route's hostname:

```
oc get route exampleqm-ibm-mq-qm --template="{{.spec.host}}"
```

Note: There are two routes, in this case use the route for message traffic. Add the host addresses of the qm route to the `ccdt.json` file.

b. In the `ccdt.json` file, add a second hostname and port pair to the connections list so your connection list now looks like this:

```
"connection":
                [
                    {
                        "host": "<hostname from London qm route>",
                        "port": 443
                    },
                    {
                        "host": "<hostname from Rome qm route>",
                        "port": 443
                    }
                ],
```

On Amazon EKS

a. Retrieve the qm hostname on the Rome site. Ensure you are on the Rome site then use the following command to retrieve the qm `LoadBalancer` hostname:

```
kubectl get service <LOAD_BALANCER_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

b. In the `ccdt.json` file, add a second hostname and port pair to the connections list so your connection list now looks like this:

```
"connection":
                [
                    {
                        "host": "<hostname from London qm LoadBalancer>",
                        "port": 1414
                    },
                    {
                        "host": "<hostname from Rome qm LoadBalancer>",
                        "port": 1414
                    }
                ],
```

14. Add some persistent messages to the queue. The queue defined in the `mqsc` file specified in the ConfigMap had the attribute `DEFPSIST(YES)` so all messages put to the queue will be persistent and are replicated to the recovery group. Put messages to the queue using the following command:

    ```
    /opt/mqm/samp/bin/amqsputc EXAMPLE.QUEUE EXAMPLEQM
    ```

    If connection to the queue manager is successful, the following response is output:

    ```
    target queue is EXAMPLE.QUEUE
    ```

    Put several messages to the queue, by entering some text then pressing **Enter** each time. To finish, press **Enter** twice.

15. Perform a managed switchover from the Live group in London to the Recovery group in Rome to demonstrate that the Native HA - CRR setup is correct. Follow the instructions in "Complete a planned switchover on a Native HA CRR configuration" on page 264. As you update the roles of the Live and Recovery groups, you might find it useful to update `metadata.labels.role` in the YAML to reflect your change, but this is not necessary for the switchover. After you have completed the switchover, you can retrieve the messages from the new live group, in Rome.

    For further information about both failovers and switchovers, see "Native HA CRR switchover and failover" on page 264.

16. Retrieve messages sent before switchover by running the following command:

    ```
    /opt/mqm/samp/bin/amqsgetc EXAMPLE.QUEUE EXAMPLEQM
    ```

    The messages that you added in step "14" on page 136 are available, and are output. After a few seconds, the command exits.

What to do next

When the maintenance is complete, you can switch back to the original live group by repeating the process in step “15” on page 136.

You could also try a unplanned failover using the instructions at “Complete an unplanned failover on a Native HA CRR configuration” on page 266.

To learn more about how to use different certificates to secure the different parts of your CRR configuration, see “Example: using multiple TLS certificates with Native HA CRR” on page 147.

Example: Adding a recovery group to an existing Native HA configuration using the IBM MQ Operator

This example deploys a queue manager using a simple Native HA configuration using the IBM MQ Operator. The example adds a recovery group and then updates the Native HA queue manager to be a live group, extending the configuration to a Native HA Cross-Region Replication (CRR) one.

Before you begin

To complete this example, you must first complete the following steps:

- On the command line, log into the clusters that you are using.
- Create two projects/namespaces for this example. The example will refer to the London and Rome sites. These refer to either two namespaces for London and Rome in a single cluster, or a namespace in each cluster for each site.
- Ensure the IBM MQ Operator 3.5.0 or above is installed in the clusters and available in the namespaces.
- **Amazon EKS deployments only**: Install the AWS Load Balancer Controller onto your Amazon EKS cluster.
- Ensure that the OpenSSL command line tool is installed.
- Complete the “Preparing to use Kubernetes by creating a pull secret” on page 86 task.
- Read “Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97 or “Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS” on page 100 so that you are familiar with using the IBM MQ operator.

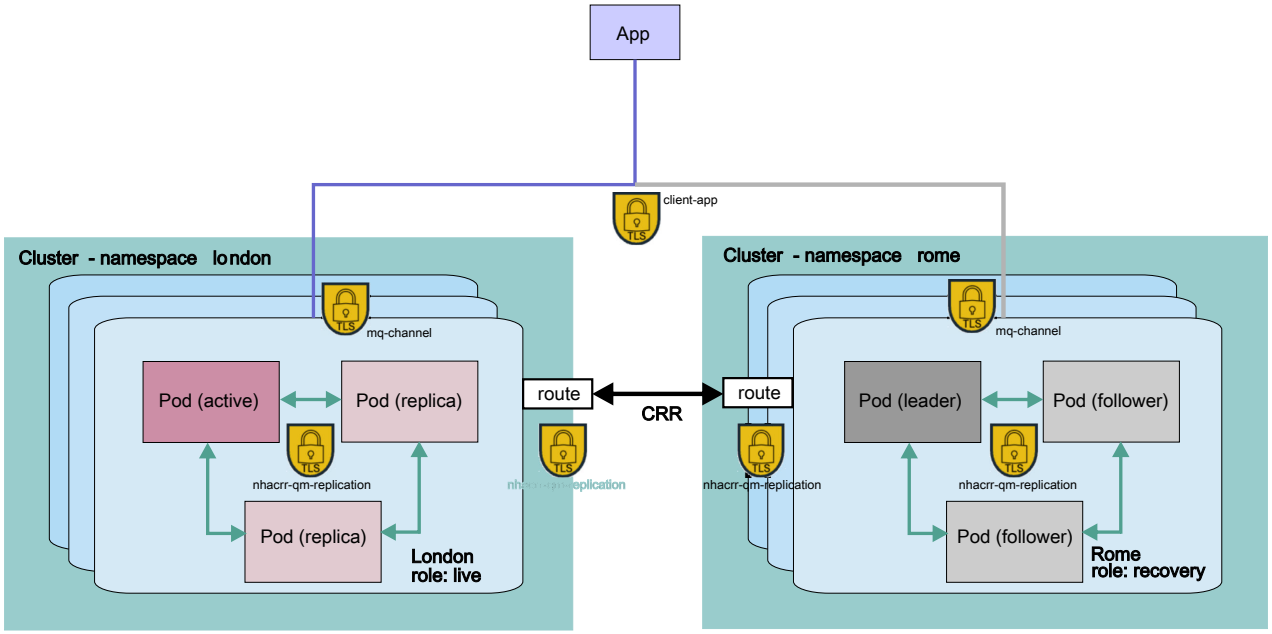
About this task

The CRR groups are usually deployed in two clusters in different locations, however this example allows you to deploy both groups to the same cluster if desired. See Native HA Cross-Region Replication for an overview of Native HA CRR.

For simplicity, this example uses a single certificate to secure all the queue manager replication traffic, this is named nhacrr-qm-replication.

To understand how certificates are used in Native HA CRR and use different certificates to secure group and instance traffic, please refer to “Example: using multiple TLS certificates with Native HA CRR” on page 147 Deploying Native HA CRR with multiple certificates.

Important: These examples are intended to get you started, they are not suitable for a production environment. Certificate management is a complex subject, for advanced users. For production, you must consider things like rotation, revocation, key length, disaster recovery, and much more.



This example provides the custom resource YAML to define two Native HA queue manager and recovery group used in the cross-region replication.

Procedure

1. Create a Native HA queue manager on the London site. Ensure you are in the London namespace on the correct cluster and create a Native HA configuration by following the instructions in “Example: Configuring Native HA using the IBM MQ Operator” on page 119. (You create the two certificates named client-app and mq-channel in the diagram when completing this step.)
2. Create the certificates to use for the queue manager replication traffic in the CRR configuration. Use the following commands to create a private key and use it to issue a self-signed certificate for your internal certificate authority. Then create a private key and certificate for the queue manager, signed with your internal certificate authority.

   ```
   openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out nhacrr-ca.key
   openssl req -x509 -new -nodes -key "nhacrr-ca.key" -sha512 -days 30 -subj "/CN=nhacrr-
   selfsigned-ca" -out nhacrr-ca.crt
   openssl req -new -nodes -out nhacrr-qm-replication.csr -newkey rsa:4096 -keyout nhacrr-qm-
   replication.key -subj "/CN=nhacrr-qm-replication"
   openssl x509 -req -in nhacrr-qm-replication.csr -CA nhacrr-ca.crt -CAkey nhacrr-ca.key
   -CAcreateserial -out nhacrr-qm-replication.crt -days 7 -sha512
   ```

   For a more information about these commands, see “Creating a self-signed PKI using OpenSSL” on page 107.
3. Create two Kubernetes secrets containing the queue manager key and certificates. Create a secret called nhacrr-qm-replication on both the Rome and London sites, to store the key and certificate for the queue manager instances. Use the command below to create the secrets, change to each namespace (Rome than London) before running the command.

   ```
   kubectl create secret generic nhacrr-qm-replication --type="kubernetes.io/tls" --from-
   file=tls.key=nhacrr-qm-replication.key --from-file=tls.crt=nhacrr-qm-replication.crt --from-
   file=nhacrr-ca.crt
   ```
4. Update the Native HA Queue Manager to use the new secret for replicated traffic. To update the certificate used for internal group traffic, and later for group to group traffic, ensure you are in the London namespace, then alter the `.spec.queuemanager.availability.tls.secretname` attribute in the QueueManager YAML to:

   ```
       tls:
         secretName: nhacrr-qm-replication
   ```

- On Red Hat OpenShift, edit the QueueManager YAML in the OCP web console, or from the command line by running the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
oc edit qmgr <QUEUEMANAGER_NAME>
```

- On Amazon EKS, run the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
kubectl edit qmgr <QUEUEMANAGER_NAME>
```

5. Create a secret in the Rome site, containing the certificate and key generated previously in step 1 of "Example: Configuring Native HA using the IBM MQ Operator" on page 119. Create the secret by logging on to the cluster, change to the Rome namespace and run the following command in the directory where the certificate file was created:

```
   kubectl create secret generic example-qm-tls --type="kubernetes.io/tls" --from-
file=tls.key=example-qm.key --from-file=tls.crt=example-qm.crt --from-file=ca.crt
```

6. Create a config map containing MQSC commands and an INI file in Rome. Duplicate the config map create on the London site in Rome.

   On the Rome site, create the following ConfigMap then apply it by using the `oc apply` command on Red Hat OpenShift, or the `kubectl apply` command on Amazon EKS. On Red Hat OpenShift, you can alternatively enter the YAML in the OCP web console by selecting **ConfigMaps** then **Create ConfigMap**.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-nativeha-configmap
data:
  example-tls.mqsc: |
    DEFINE CHANNEL('MTLS.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS13_OR_HIGHER') REPLACE
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
ACTION(REPLACE)
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=example-app1') USERSRC(MAP)
MCAUSER('app1') ACTION(REPLACE)
    SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)
    DEFINE QLOCAL('EXAMPLE.QUEUE') REPLACE
    SET AUTHREC PROFILE('EXAMPLE.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(BROWSE,PUT,GET,INQ)
  example-tls.ini: |
    Service:
        Name=AuthorizationService
        EntryPoints=14
        SecurityPolicy=UserExternal
```

   The MQSC defines a channel called MTLS.SVRCONN and a queue called EXAMPLE.QUEUE. The channel is configured to allow access only to clients which present a certificate with a CN of example-app1 like the one stored in the example-qm-tls created in Step 4.

7. Deploy the Native HA Recovery group. The Recovery group is deployed to the Rome site. Ensure that you are in the Rome namespace, then create the QueueManager.

   Note: You must use the same queue manager name in both live and recovery groups. By default, if you do not specify `.spec.queuemanager.name`, it defaults to `.metadata.name`. You can achieve matching names by using one of the following methods:

   - Name your queue manager resource the same in each cluster (that is, `.metadata.name` matches) and do not override the name.
   - Name your queue manager individually in `.metadata.name` and override the name with `.spec.queuemanager.name` so that names match in both sites.

   In this example, the queue manager name is set to `exampleqm` by using `.metadata.name`.

   Note: This example uses an IBM MQ Advanced license. IBM MQ licensed queue managers can also be configured to use Native HA and Cross-Region Replication features by adding additional licenses

to a queue manager. For more information, see “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.

Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

Example QueueManager for Red Hat OpenShift

Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
  labels:
    role: Recovery
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: nhacrr-qm-replication
      nativeHAGroups:
        local:
          name: "rome"
          role: "Recovery"
    mqsc:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.ini
    storage:
      queueManager:
        type: persistent-claim
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
  labels:
    role: Recovery
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: NativeHA
      tls:
        secretName: nhacrr-qm-replication
      nativeHAGroups:
```

```
        local:
          name: "rome"
          role: "Recovery"
          route:
            enabled: false
    mqsc:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-nativeha-configmap
        items:
        - example-tls.ini
    storage:
      queueManager:
        type: persistent-claim
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.availability.nativeHAGroups.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

8. Verify that the Recovery group has deployed correctly (run these commands on the Rome site).
   a) Issue the relevant command to display the queue manager resource then wait for the queue manager to be running.
      - On Red Hat OpenShift:

        ```
        oc get qmgr exampleqm
        ```

      - On Amazon EKS:

        ```
        kubectl get qmgr exampleqm
        ```

      You should see the following message:

      ```
      NAME            PHASE
      exampleqm   Running
      ```

   b) When the queue manager is running, check the pod states by using the relevant command.
      - On Red Hat OpenShift:

        ```
        oc get pods
        ```

      - On Amazon EKS:

        ```
        kubectl get pods
        ```

One pod should be in ready state:

```
NAME                  READY   STATUS    RESTARTS   AGE
exampleqm-ibm-mq-0   1/1     Running   0          2m45s
exampleqm-ibm-mq-1   0/1     Running   0          2m45s
exampleqm-ibm-mq-2   0/1     Running   0          2m45s
```

c) Check the status of the recovery group by using the relevant command, where <LEADER-POD> is the name of the READY 1/1 pod (exampleqm-ibm-mq-0 in the example).

- On Red Hat OpenShift:

```
oc exec -t <LEADER-POD> -- dspmq -o nativeha -g
```

- On Amazon EKS:

```
kubectl exec -t <LEADER-POD> -- dspmq -o nativeha -g
```

You should see the following output:

```
QMNAME(EXAMPLEQM)                                         ROLE(Leader)
INSTANCE(exampleqm-ibm-mq-0) INSYNC        (yes) QUORUM(1/3) GRPLSN() GRPNAME(rome)
GRPROLE(Recovery)
 GRPNAME(rome) GRPROLE(Recovery) GRPADDR(Unknown) GRPVER(10.0.0.0) GRSTATUS(Waiting for
connection) RCOVLSN()         RCOVTIME() BACKLOG(Unknown) INSYNC(Unknown) SYNCTIME()
ALTDATE(2025-04-25) ALTTIME(13.48.15)
```

Key parts of this status are as follows:

- Role(Leader)
- QUORUM(1/3): this remains 1/3 until the Live and Recovery groups connect
- GRSTATUS(Waiting for connection): this updates when the Live group connects

9. **Amazon EKS deployments only**: Configure a LoadBalancer to allow the Live group to communicate with the Recovery group.

a) Ensure you are in the Rome namespace, then create the LoadBalancer Service to facilitate a TLS connection to a queue manager from outside your cluster with the following configuration.

```
apiVersion: v1
kind: Service
metadata:
  name: crr-rome
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
spec:
  type: LoadBalancer
  ports:
    - port: 9415
      targetPort: 9415
      name: nha-crr
  selector:
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/instance: exampleqm
```

Important: If you are using an **IPv6 single-stack** system, make the following changes to the metadata.annotations section:

- Change the value of service.beta.kubernetes.io/aws-load-balancer-nlb-target-type from "instance" to "ip"
- Add the following line:

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: "dualstack"
```

b) Verify that the Service of type LoadBalancer was created:

```
kubectl get service <LOAD_BALANCER_NAME>
```

c) Check that your LoadBalancer is ready.

The Amazon Load Balancer Controller takes several minutes to provision and register its targets. Run the following command to show the status of the LoadBalancer. When the LoadBalancer is ready, the status is active:

```
aws elbv2 describe-load-balancers --query 'LoadBalancers[*].
[LoadBalancerName,State.Code]' --output text
```

10. Update the Native HA Queue Manager to use the Live group.

a) Retrieve the recovery group address on the Rome site:

On Red Hat OpenShift

```
oc get route exampleqm-ibm-mq-nhacrr --template="{{.spec.host}}"
```

Note: There are two routes created when the CRR groups are deployed, one ending qm and one ending nhacrr. The qm route is the one used to send messages to the queue manager, the nhacrr one is used for group to group communication. Ensure you add the host addresses of the nhacrr route as the remote address for the group in the YAML below.

On Amazon EKS

Run the following command to retrieve the hostname of the LoadBalancer created in the previous step:

```
kubectl get service <LOAD_BALANCER_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

b) Open the London QueueManager YAML for editing.

- On Red Hat OpenShift, edit the QueueManager YAML in the OCP web console, or from the command line by running the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
oc edit qmgr <QUEUEMANAGER_NAME>
```

- On Amazon EKS, run the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
kubectl edit qmgr <QUEUEMANAGER_NAME>
```

c) Add the hostname that you retrieved earlier to the London QueueManager YAML.

The hostname is the address that the live group uses to connect to the recovery group. Add the following code directly under the type: NativeHA attribute, replacing <rome group hostname> with the hostname that you retrieved earlier:

On Red Hat OpenShift

```
      nativeHAGroups:
        local:
          name: "london"
          role: "Live"
        remotes:
          - name: "rome"
            addresses:
              - "<rome group hostname>:443"
```

On Amazon EKS

```
      nativeHAGroups:
        local:
          name: "london"
          role: "Live"
          route:
            enabled: false
        remotes:
          - name: "rome"
```

```
            addresses:
              - "<rome group hostname>:9415"
```

`nativeHAGroups.route.enabled` is set to `false` because this is a Red Hat OpenShift specific feature that is enabled by default. This feature must be disabled explicitly on Amazon EKS.

d) Add a Live role label to the top of the metadata stanzas, then save the YAML.

```
metadata:
  labels:
    role: Live
```

11. Verify the Live group and the Cross Region Replication.

a) On the London site, issue the relevant command to display the queue manager resource then wait for the queue manager to be running. You should see the following message:

- On Red Hat OpenShift:

```
oc get qmgr exampleqm
```

- On Amazon EKS:

```
kubectl get qmgr exampleqm
```

You should see the following message:

```
NAME            PHASE
exampleqm   Running
```

b) When the queue manager is running, check the pod states by using the relevant command.

- On Red Hat OpenShift:

```
oc get pods
```

- On Amazon EKS:

```
kubectl get pods
```

One pod should be in ready state:

```
NAME                READY   STATUS    RESTARTS   AGE
exampleqm-ibm-mq-0   1/1     Running   0          2m45s
exampleqm-ibm-mq-1   0/1     Running   0          2m45s
exampleqm-ibm-mq-2   0/1     Running   0          2m45s
```

c) Check the status of the live group by issuing the following command, where <ACTIVE-POD> is the name of the ready 1/1 pod (`exampleqm-ibm-mq-0` in the example).

- On Red Hat OpenShift:

```
oc exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
```

- On Amazon EKS:

```
kubectl exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
```

```
oc exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
```

You should see the following output:

```
QMNAME(EXAMPLEQM)                                         ROLE(Active)
INSTANCE(exampleqm-ibm-mq-0) INSYNC        (yes) QUORUM(3/3) GRPLSN(<0:0:13:21327>)
GRPNAME(london) GRPROLE(Live)
GRPNAME(london) GRPROLE(Live) GRPADDR(Unknown) GRPVER(10.0.0.0)
GRSTATUS(Normal) RCOVLSN(<0:0:13:21327>)         RCOVTIME(2025-06-04T16:22:54.414645Z)
INITLSN(<0:0:9:5720>) INITTIME(2025-06-04T16:12:09.629132Z) LIVETIME
```

```
(2025-06-04T16:12:11.825065Z) ALTDATE(2025-06-04) ALTTIME(16.23.01)
GRPNAME(rome) GRPROLE(Recovery) GRPADDR(exampleqm-ibm-mq-nhacrr-
<namespace>.apps.<cluster>) GRPVER(10.0.0.0)         CONNGRP(yes) GRSTATUS(Normal)
RCOVLSN(<0:0:13:21327>) RCOVTIME(2025-06-04T16:22:54.414645Z) BACKLOG(0) INSYNC
(yes) SYNCTIME(2025-06-04T16:29:17.206996Z) ALTDATE(2025-06-04) ALTTIME(16.29.12)
```

Key parts of this status are as follows:

- `Role(Active)`
- `QUORUM(3/3)` - the live group has full quorum
- `GRSTATUS(Normal)`

d) Run the same command at the Rome site and observe that `QUORUM(3/3)` and `GRSTATUS(Normal)` were also updated.

You should see the following output:

```
QMNAME(EXAMPLEQM)                                         ROLE(Leader)
INSTANCE(exampleqm-ibm-mq-0)INSYNC        (yes) QUORUM(3/3) GRPLSN(<0:0:87:58322>)
GRPNAME(rome) GRPROLE(Recovery)
 GRPNAME(rome) GRPROLE(Recovery) GRPADDR(exampleqm-ibm-mq-nhacrr-
rome.apps.destructive-1.cp.fyre.ibmcom)         GRPVER(9.4.3.0) GRSTATUS(Normal)
RCOVLSN(<0:0:87:58322>) RCOVTIME(2025-05-16T14:27:09051799Z) BACKLOG(0)
INSYNC(yes) SYNCTIME(2025-05-16T14:30:40.134627Z) ALTDATE(2025-05-16)ALTTIME(14.27.17)
 GRPNAME(london) GRPROLE(Live) GRPADDR(Unknown) GRPVER(10.0.0.0) CONNGRP(yes)
GRSTATUS(Normal)RCOVLSN        (<0:0:87:58322>) RCOVTIME(2025-05-16T14:27:09.051799Z)
INITLSN(<0:0:9:4696>) INITTIM(2025-05-12T16:58:13.         582274Z)
LIVETIME(2025-05-12T16:58:16.319844Z) ALTDATE(2025-05-16)ALTTIME(14.30.41)
```

12. **Amazon EKS deployments only**: Configure a `LoadBalancer` to allow the Recovery group to communicate with the Live group.

a) Ensure you are in the London namespace, then create the `LoadBalancer` `Service` to facilitate a TLS connection to a queue manager from outside your cluster with the following configuration.

```
apiVersion: v1
kind: Service
metadata:
  name: crr-london
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
spec:
  type: LoadBalancer
  ports:
    - port: 9415
      targetPort: 9415
      name: nha-crr
  selector:
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/instance: exampleqm
```

Important: If you are using an **IPv6 single-stack** system, make the following changes to the `metadata.annotations` section:

- Change the value of `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type` from `"instance"` to `"ip"`
- Add the following line:

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: "dualstack"
```

b) Verify that the `Service` of type `LoadBalancer` was created:

```
kubectl get service <LOAD_BALANCER_NAME>
```

c) Check that your `LoadBalancer` is ready.

The Amazon Load Balancer Controller takes several minutes to provision and register its targets. Run the following command to show the status of the `LoadBalancer`. When the `LoadBalancer` is ready, the status is `active`:

```
aws elbv2 describe-load-balancers --query 'LoadBalancers[*].
[LoadBalancerName,State.Code]' --output text
```

13. Add a remote address to the Native HA Recovery group. When Rome becomes the live group it needs to know London's remote address so it can connect to London as the recovery group.

    a) Retrieve the live group address on the London site.

    On Red Hat OpenShift

    Run the following command:

    ```
    oc get route exampleqm-ibm-mq-nhacrr --template="{{.spec.host}}"
    ```

    Note: There are two routes created when the CRR groups are deployed, one ending `qm` and one ending `nhacrr`. The `qm` route is the one used to send messages to the queue manager, the `nhacrr` route is used for group-to-group communication. Always add the host addresses of the `nhacrr` route as the remote address for the group in the YAML.

    On Amazon EKS

    Run the following command to retrieve the hostname of the `LoadBalancer` created in the previous step:

    ```
    kubectl get service <LOAD_BALANCER_NAME> -o
    jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
    ```

    b) Add the following code to the `spec.queueManager.availability.nativeHAGroups` YAML section for the Rome site, below the local stanza.

    On Red Hat OpenShift

    ```
            remotes:
              - name: "london"
                addresses:
                  - "<london group hostname>:443"
    ```

    On Amazon EKS

    ```
            remotes:
              - name: "london"
                addresses:
                  - "<london group hostname>:9415"
    ```

    After you make the update, wait for the pods to restart, be running and a pod to be the READY 1/1 pod.

14. Add the Rome site hostname to the `ccdt.json` file.

    After a switchover, where Rome becomes the live site, clients need to connect to the group in Rome. You enable this behavior by adding the hostname for Rome to the `ccdt.json` file.

    On Red Hat OpenShift

    a. Retrieve the `qm` hostname on the Rome site. Ensure you are on the Rome site then use the following command to retrieve the `qm` route's hostname:

    ```
    oc get route exampleqm-ibm-mq-qm --template="{{.spec.host}}"
    ```

    Note: There are two routes, in this case use the route for message traffic. Add the host addresses of the `qm` route to the `ccdt.json` file.

    b. In the `ccdt.json` file, add a second hostname and port pair to the connections list so your connection list now looks like this:

    ```
    "connection":
                  [
    ```

```
                    {
                        "host": "<hostname from London qm route>",
                        "port": 443
                    },
                    {
                        "host": "<hostname from Rome qm route>",
                        "port": 443
                    }
                ],
```

On Amazon EKS

a. Retrieve the qm hostname on the Rome site. Ensure you are on the Rome site then use the following command to retrieve the qm `LoadBalancer` hostname:

```
kubectl get service <LOAD_BALANCER_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

b. In the `ccdt.json` file, add a second hostname and port pair to the connections list so your connection list now looks like this:

```
"connection":
                [
                    {
                        "host": "<hostname from London qm LoadBalancer>",
                        "port": 1414
                    },
                    {
                        "host": "<hostname from Rome qm LoadBalancer>",
                        "port": 1414
                    }
                ],
```

15. Perform a managed switchover from London to Rome to demonstrate that the Native HA - CRR setup is correct, next perform a managed switchover between the Live group in London to Recovery group in Rome. Follow the instructions in “Complete a planned switchover on a Native HA CRR configuration” on page 264. As you update the roles of the Live and Recovery groups, you might find it useful to update `metadata.labels.role` in the YAML to reflect the change, but this is not necessary to switchover. After you have completed the switchover, you can retrieve the messages from the new live group, in Rome.

    For further information on about both failovers and switchovers, see “Native HA CRR switchover and failover” on page 264.

What to do next

When the maintenance is complete, you can switch back to the original Live group by repeating the process in step “15” on page 147.

You could also try a unplanned failover using the instructions at “Complete an unplanned failover on a Native HA CRR configuration” on page 266.

Finally, to learn more about how to use different certificates to secure the different parts of your CRR configuration, see Configuring TLS for Native HA CRR.

Example: using multiple TLS certificates with Native HA CRR

This example shows how to use TLS and certificates in a Native HA CRR configuration to secure different types of traffic.

Before you begin

Before starting this example, you must first complete “Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator” on page 125 (this example assumes that the deployment exists and the London group is the live group).

Then complete the following steps:

- Ensure that you can log into the cluster or clusters that you are using, and select the relevant project/namespace. (The example described in "Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator" on page 125 created two projects/namespaces either in two namespaces for London and Rome in a single cluster, or a namespace in each cluster for each site. This example also refers to these as the London and Rome sites.)
- Ensure the IBM MQ Operator 3.5.0 or above is installed in the clusters and available in the namespaces.
- Ensure that the OpenSSL command line tool is installed.
- Complete the "Preparing to use Kubernetes by creating a pull secret" on page 86 task.
- Read "Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift" on page 97 or "Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS" on page 100 so that you are familiar with using the IBM MQ operator.

About this task

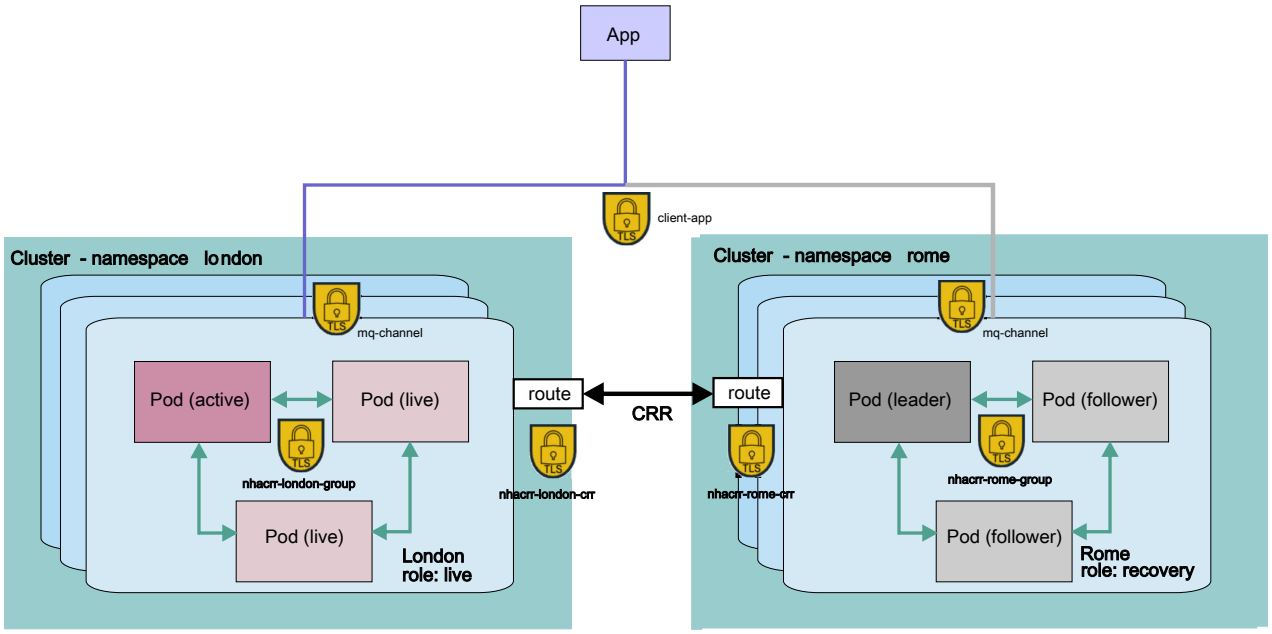
This example starts with an overview of how TLS and certificates are used in a Native HA CRR configuration, then uses simple steps to demonstrate how `nhacrr-london-group`, `nhacrr-rome-group`, `nhacrr-london-crr` and `nhacrr-rome-crr` traffic is secured.

The steps in the procedure section below, walk you through altering the "Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator" on page 125 deployment to use different certificates for each type of traffic and control which CipherSpec is used.

Configuring TLS for Native HA CRR

The following diagram shows where in the Native HA CRR configuration certificates are used. The certificates are used as follows:

1. To secure communication between the queue manager and client applications - `client-app` and `mq-channel`
2. To secure queue manager replication traffic - `nhacrr-london-group`, `nhacrr-rome-group`, `nhacrr-london-crr` and `nhacrr-rome-crr`



Secure communication between the queue manager and the applications

The two Native HA groups, shown in the diagram, together form a highly available queue manager, which client applications can connect to via the live group. When a switchover occurs, applications are disconnected and must reconnect to the new live group. (Adding the connection details of both groups to an application connection list facilitates the reconnection.)

If the applications connecting via a route are outside of, or in a different cluster to, one or both of the groups, they need to communicate via TLS using private keys and certificates. In the diagram, the client's certificate and key is labelled `client-app` and the queue manager's are labelled `mq-channel`. The `mq-channel` certificate and private key must be added as two secrets, one for each group.

"Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator" on page 125 walks you through creating these certificates, keys, and secrets. The following steps focus on the certificates for queue manager replication traffic and do not change the `mq-channel` or `client-app` certificates.

Secure queue manager replication traffic

Traffic is replicated both within the groups and between them. The network traffic between the two groups must always be secured, but the traffic within a group can be plain text as the traffic inside the cluster generally is not accessible.

1. `nhacrr-london-group` and `nhacrr-rome-group` represent the certificates and private keys used to secure traffic within each group.
2. `nhacrr-london-crr` and `nhacrr-rome-crr` represent the certificates and private keys used to secure each end of the cross-group traffic.

In Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator, the same certificate and private key, stored in a `nhacrr-qm-replication` secret, in each namespace, is used to secure the `nhacrr-london-group`, `nhacrr-rome-group`, `nhacrr-london-crr` and `nhacrr-rome-crr` traffic.

Important: These examples are intended to get you started, they are not suitable for a production environment. Certificate management is a complex subject, for advanced users. For production, you must consider things like rotation, revocation, key length, disaster recovery, and much more.

This example shows you how to use different certificates for each type of traffic.

Procedure

1. Create four sets of keys and certificates to use for the queue manager replication traffic. To create these for this example, run the following commands. Each set of commands creates a private key, uses it to issue a self-signed certificate for an internal certificate authority, then creates a private key and certificate signed with the internal certificate authority. This exercise is repeated to create certificates for `nhacrr-london-group`, `nhacrr-rome-group`, `nhacrr-london-crr` and `nhacrr-rome-crr` traffic.

   For a more detailed explanation of the commands, see "Creating a self-signed PKI using OpenSSL" on page 107.

   Note: This example creates separate CAs for each certificate chain, this is to allow the example to demonstrate setting up trust between the groups if needed. Certificates in the same organization often use a common CA, when this is the case, the trust stanzas in Queue Manager YAML are not needed as the CA is already trusted.

   ```
   openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out nhacrr-london-group-
   ca.key
   openssl req -x509 -new -nodes -key "nhacrr-london-group-ca.key" -sha512 -days 30 -subj "/
   CN=nhacrr-london-group-selfsigned-ca" -out nhacrr-london-group-ca.crt
   openssl req -new -nodes -out nhacrr-london-group.csr -newkey rsa:4096 -keyout nhacrr-london-
   group.key -subj "/CN=nhacrr-london-group"
   openssl x509 -req -in nhacrr-london-group.csr -CA nhacrr-london-group-ca.crt -CAkey nhacrr-
   london-group-ca.key -CAcreateserial -out nhacrr-london-group.crt -days 7 -sha512
   ```

   ```
   openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out nhacrr-rome-group-ca.key
   openssl req -x509 -new -nodes -key "nhacrr-rome-group-ca.key" -sha512 -days 30 -subj "/
   CN=nhacrr-rome-group-selfsigned-ca" -out nhacrr-rome-group-ca.crt
   openssl req -new -nodes -out nhacrr-rome-group.csr -newkey rsa:4096 -keyout nhacrr-rome-
   group.key -subj "/CN=nhacrr-rome-group"
   openssl x509 -req -in nhacrr-rome-group.csr -CA nhacrr-rome-group-ca.crt -CAkey nhacrr-rome-
   group-ca.key -CAcreateserial -out nhacrr-rome-group.crt -days 7 -sha512
   ```

```
openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out nhacrr-london-crr-ca.key
openssl req -x509 -new -nodes -key "nhacrr-london-crr-ca.key" -sha512 -days 30 -subj "/
CN=nhacrr-london-crr-selfsigned-ca" -out nhacrr-london-crr-ca.crt
openssl req -new -nodes -out nhacrr-london-crr.csr -newkey rsa:4096 -keyout nhacrr-london-
crr.key -subj "/CN=nhacrr-london-crr"
openssl x509 -req -in nhacrr-london-crr.csr -CA nhacrr-london-crr-ca.crt -CAkey nhacrr-
london-crr-ca.key -CAcreateserial -out nhacrr-london-crr.crt -days 7 -sha512
```

```
openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out nhacrr-rome-crr-ca.key
openssl req -x509 -new -nodes -key "nhacrr-rome-crr-ca.key" -sha512 -days 30 -subj "/
CN=nhacrr-rome-crr-selfsigned-ca" -out nhacrr-rome-crr-ca.crt
openssl req -new -nodes -out nhacrr-rome-crr.csr -newkey rsa:4096 -keyout nhacrr-rome-
crr.key -subj "/CN=nhacrr-rome-crr"
openssl x509 -req -in nhacrr-rome-crr.csr -CA nhacrr-rome-crr-ca.crt -CAkey nhacrr-rome-crr-
ca.key -CAcreateserial -out nhacrr-rome-crr.crt -days 7 -sha512
```

2. Create Kubernetes secrets containing the keys and certificates for Rome. Use the following commands to create two TLS secrets on the Rome site:

   a. One certificate to secure the internal Rome group traffic.

   b. One certificate to secure the cross-group traffic from the Rome group.

```
kubectl create secret generic nhacrr-rome-group --type="kubernetes.io/tls" --
from-file=tls.key=nhacrr-rome-group.key --from-file=tls.crt=nhacrr-rome-group.crt --from-
file=nhacrr-rome-group-ca.crt
kubectl create secret generic nhacrr-rome-crr --type="kubernetes.io/tls" --from-
file=tls.key=nhacrr-rome-crr.key --from-file=tls.crt=nhacrr-rome-crr.crt --from-file=nhacrr-
rome-crr-ca.crt
```

   Now use the following commands to create two secrets containing the Rome certificates for the London group to trust:

```
kubectl create secret generic nhacrr-london-group-trust --from-file=tls.crt=nhacrr-london-
group.crt --from-file=ca.crt=nhacrr-london-group-ca.crt --type=opaque
kubectl create secret generic nhacrr-london-crr-trust --from-file=tls.crt=nhacrr-london-
crr.crt  --from-file=ca.crt=nhacrr-london-crr-ca.crt --type=opaque
```

3. Create Kubernetes secrets containing the keys and certificates for London. Use the following commands to create two TLS secrets on the London site:

   a. One certificate to secure the internal London group traffic.

   b. One certificate to secure the cross-group traffic from the London group.

```
kubectl create secret generic nhacrr-london-group --type="kubernetes.io/tls" --from-
file=tls.key=nhacrr-london-group.key --from-file=tls.crt=nhacrr-london-group.crt --from-
file=nhacrr-london-group-ca.crt
kubectl create secret generic nhacrr-london-crr --type="kubernetes.io/tls" --
from-file=tls.key=nhacrr-london-crr.key --from-file=tls.crt=nhacrr-london-crr.crt --from-
file=nhacrr-london-crr-ca.crt
```

   Now use the following commands to create two secrets containing the London certificates for the Rome group to trust:

```
kubectl create secret generic nhacrr-rome-group-trust --from-file=tls.crt=nhacrr-rome-
group.crt --from-file=ca.crt=nhacrr-rome-group-ca.crt --type=opaque
kubectl create secret generic nhacrr-rome-crr-trust --from-file=tls.crt=nhacrr-rome-
crr.crt  --from-file=ca.crt=nhacrr-rome-crr-ca.crt --type=opaque
```

4. Secure traffic in the Rome and London sites with different certificates. The deployment created in “Example: Deploying a simple Native HA CRR configuration using the IBM MQ Operator” on page 125 secured all replication traffic with the `nhacrr-qm-replication` secret. This step changes the configuration to secure the traffic within each group with the `nhacrr-rome-group` and `nhacrr-london-group` secrets respectively.

   The `.spec.queuemanager.availability.tls.secretName` attribute specifies the secret to be used to secure traffic within the site's group. When no other TLS attributes are defined, the secret is also used to secure traffic between groups.

   a) On the Rome site, open the Rome QueueManager YAML for editing.

- On Red Hat OpenShift, edit the QueueManager YAML in the Red Hat OpenShift web console, or from the command line by running the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
oc edit qmgr <QUEUEMANAGER_NAME>
```

- On Amazon EKS, run the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
kubectl edit qmgr <QUEUEMANAGER_NAME>
```

b) Alter the `.spec.queuemanager.availability.tls.secretName` attribute in the QueueManager YAML:

```
      tls:
        secretName: nhacrr-rome-group
```

c) Add a `.spec.queueManager.availability.NativeHAGroups.remotes.trust` stanza to the YAML and add the `nhacrr-london-group-trust` secret as the one to trust:

```
...
      nativeHAGroups:
        local:
          name: "rome"
          role: "Recovery"
        remotes:
          - name: "london"
            addresses:
              - <london group hostname>:443 # Note: On Amazon EKS deployments, the port
number is 9415
            trust:
              - secret:
                  items:
                    - tls.crt
                    - ca.crt
                  secretName: nhacrr-london-group-trust
...
```

The `nhacrr-london-group-trust` secret contains the certificate and CA stored in the `nhacrr-london-group` secret in the London site. The `nhacrr-london-group` secret is used to secure the traffic from the London group, so the Rome group uses this secret to confirm the certificate supplied can be trusted.

d) Complete the equivalent steps on the London site. Change the `.spec.queuemanager.availability.tls.secretName` attribute to the following value:

```
      tls:
        secretName: nhacrr-london-group
```

e) Add the `nhacrr-rome-group-trust` secret to a `.spec.queueManager.availability.NativeHAGroups.remotes.trust` stanza:

```
...
      nativeHAGroups:
        local:
          name: "london"
          role: "Live"
        remotes:
          - name: "rome"
            addresses:
              - "<rome group hostname>:443" # Note: On Amazon EKS deployments, the port
number is 9415
            trust:
              - secret:
                  items:
                    - tls.crt
                    - ca.crt
                  secretName: nhacrr-rome-group-trust
...
```

5. Verify the Cross Region Replication with the new certificates.

a) At the London site, issue the relevant command to display the queue manager resource then wait for the queue manager to be running.

- On Red Hat OpenShift:

```
oc get qmgr exampleqm
```

- On Amazon EKS:

```
kubectl get qmgr exampleqm
```

You should see the following message:

```
NAME            PHASE
exampleqm   Running
```

b) When the queue manager is running, check the pod states by using the relevant command.

- On Red Hat OpenShift:

```
oc get pods
```

- On Amazon EKS:

```
kubectl get pods
```

One pod should be in ready state (this is the active pod):

```
NAME                 READY   STATUS    RESTARTS   AGE
exampleqm-ibm-mq-0   1/1     Running   0          2m45s
exampleqm-ibm-mq-1   0/1     Running   0          2m45s
exampleqm-ibm-mq-2   0/1     Running   0          2m45s
```

c) Check the status of the live group by using the relevant command, where <ACTIVE-POD> is the name of the READY 1/1 pod (exampleqm-ibm-mq-0 in the example).

- On Red Hat OpenShift:

```
oc exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
```

- On Amazon EKS:

```
kubectl exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
```

You should see the following output:

```
QMNAME(EXAMPLEQM)                                         ROLE(Active)
INSTANCE(exampleqm-ibm-mq-0) INSYNC        (yes) QUORUM(3/3) GRPLSN(<0:0:87:58322>)
GRPNAME(london) GRPROLE(Live)
 GRPNAME(london) GRPROLE(Live) GRPADDR(Unknown) GRPVER(9.4.3.0)
GRSTATUS(Normal) RCOVLSN(<0:0:87:58322>)         RCOVTIME(2025-05-16T14:27:09.051799Z)
INITLSN(<0:0:9:4696>) INITTIME(2025-05-12T16:58:13.582274Z) LIVETIME
(2025-05-12T16:58:16.319844Z) ALTDATE(2025-05-16) ALTTIME(14.27.10)
 GRPNAME(rome) GRPROLE(Recovery) GRPADDR(exampleqm-ibm-mq-nhacrr-
<namespace>.apps.<cluster>) GRPVER(9.4.3.0)         CONNGRP(yes) GRSTATUS(Normal)
RCOVLSN(<0:0:87:58322>) RCOVTIME(2025-05-16T14:27:09.051799Z) BACKLOG(0) INSYNC
(yes) SYNCTIME(2025-05-16T14:28:38.084565Z) ALTDATE(2025-05-16) ALTTIME(14.28.31)
```

Key parts of this status are as follows:

- Role(Leader)
- QUORUM(3/3): the group has full quorum
- GRSTATUS(Normal)

6. Secure the traffic between groups with separate certificates. You can use separate certificates for the within group and for cross-group traffic. In this step, the certificates in the nhacrr-rome-crr and nhacrr-london-crr secrets are used to secure the traffic between groups, while leaving the nhacrr-rome-group and nhacrr-london-group secrets securing the traffic in each group. To

achieve this, two changes are required on each site, one to specify the secret for traffic between groups and one to change the trusts to match the new certificates being used.

Note: If the secrets used for the `.spec.queuemanager.availability.NativeHAGroups.local.tls.secretName` and the `.spec.queuemanager.availability.tls.secretName` attributes contain the same certificates, there is a clash and the queue manager instances are not able to start. If you want to use a single certificate, set the `.spec.queuemanager.availability.tls.secretName` only.

a) On the Rome site, open the Rome QueueManager YAML for editing.

- On Red Hat OpenShift, edit the QueueManager YAML in the Red Hat OpenShift web console, or from the command line by running the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
oc edit qmgr <QUEUEMANAGER_NAME>
```

- On Amazon EKS, run the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

```
kubectl edit qmgr <QUEUEMANAGER_NAME>
```

b) Add the `.spec.queuemanager.availability.NativeHAGroups.local.tls.key.secretName` attribute in the QueueManager YAML and set it to `nhacrr-rome-crr`. Then change the `.spec.queueManager.availability.NativeHAGroups.remotes.trust.secret.secretName` to `nhacrr-london-crr-trust`:

```
...
      nativeHAGroups:
        local:
          name: "rome"
          role: "Recovery"
          tls:
            key:
              secretName: nhacrr-rome-crr
        remotes:
          - name: "london"
            addresses:
              - "<london group hostname>:443" # Note: On Amazon EKS deployments, the
port number is 9415
            trust:
              - secret:
                  items:
                    - tls.crt
                    - ca.crt
                  secretName: nhacrr-london-crr-trust
...
```

c) Complete the equivalent steps on the London site. Add the `.spec.queuemanager.availability.NativeHAGroups.local.tls.key.secretName` attribute in the London QueueManager YAML and set it to `nhacrr-london-crr`. Then change the `.spec.queueManager.availability.NativeHAGroups.remotes.trust.secret.secretName` to `nhacrr-rome-crr-trust`.

```
...
      nativeHAGroups:
        local:
          name: "london"
          role: "Live"
          tls:
            key:
              secretName: nhacrr-london-crr
        remotes:
          - name: "london"
            addresses:
              - "<london group hostname>:443" # Note: On Amazon EKS deployments, the
port number is 9415
            trust:
              - secret:
                  items:
```

```
                    - tls.crt
                    - ca.crt
                  secretName: nhacrr-rome-crr-trust
...
```

7. Verify the Cross Region Replication with the new certificates.
   a) At the London site, issue the relevant command to display the queue manager resource then wait for the queue manager to be running.
      - On Red Hat OpenShift:

        ```
        oc get qmgr exampleqm
        ```

      - On Amazon EKS:

        ```
        kubectl get qmgr exampleqm
        ```

      You should see the following message:

      ```
      NAME            PHASE
      exampleqm   Running
      ```

   b) When the queue manager is running, check the pod states by using the relevant command.
      - On Red Hat OpenShift:

        ```
        oc get pods
        ```

      - On Amazon EKS:

        ```
        kubectl get pods
        ```

   c) Check the status of the live group by using the relevant command, where <ACTIVE-POD> is the name of the READY 1/1 pod that was returned by the previous command.
      - On Red Hat OpenShift:

        ```
        oc exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
        ```

      - On Amazon EKS:

        ```
        kubectl exec -t <ACTIVE-POD> -- dspmq -o nativeha -g
        ```

      You should see the following output:

      ```
      QMNAME(EXAMPLEQM)                                         ROLE(Active)
      INSTANCE(exampleqm-ibm-mq-0) INSYNC        (yes) QUORUM(3/3) GRPLSN(<0:0:87:58322>)
      GRPNAME(london) GRPROLE(Live)
       GRPNAME(london) GRPROLE(Live) GRPADDR(Unknown) GRPVER(9.4.3.0)
      GRSTATUS(Normal) RCOVLSN(<0:0:87:58322>)         RCOVTIME(2025-05-16T14:27:09.051799Z)
      INITLSN(<0:0:9:4696>) INITTIME(2025-05-12T16:58:13.582274Z) LIVETIME
      (2025-05-12T16:58:16.319844Z) ALTDATE(2025-05-16) ALTTIME(14.27.10)
       GRPNAME(rome) GRPROLE(Recovery) GRPADDR(exampleqm-ibm-mq-nhacrr-
      <namespace>.apps.<cluster>) GRPVER(9.4.3.0)         CONNGRP(yes) GRSTATUS(Normal)
      RCOVLSN(<0:0:87:58322>) RCOVTIME(2025-05-16T14:27:09.051799Z) BACKLOG(0) INSYNC
      (yes) SYNCTIME(2025-05-16T14:28:38.084565Z) ALTDATE(2025-05-16) ALTTIME(14.28.31)
      ```

      Key parts of this status are as follows:

      - `Role(Active)`
      - `QUORUM(3/3)`: the group has full quorum
      - `GRSTATUS(Normal)`

   d) Issue the relevant command to check that the correct certificates are being used for the different types of traffic.
      - On Red Hat OpenShift: `oc logs <ACTIVE_POD>`
      - On Amazon EKS: `kubectl logs <ACTIVE_POD>`

For cross-group traffic, you should see the following message in the log:

```
AMQ3307I: The Native HA secure connection between the local group and 'rome' is using
certificate DN 'CN=nhacrr-london-crr'
```

For internal group traffic, you should see the following message, where <REPLICA_POD> is name of either of the pods with READY 0/1 status:

```
AMQ3306I: The Native HA secure connection between the local instance and '<REPLICA_POD>'
accepted remote certificate DN 'CN=nhacrr-london-group'
```

8. Add specific CipherSpecs for replicated traffic.

   If no specific CipherSpecs are defined for the replicated traffic then ANY_TLS13_OR_HIGHER is used so the protocols and encryption are negotiated between the instances. For further information about the CipherSpecs that IBM MQ supports, see Enabling CipherSpecs.

   a) Open the QueueManager YAML for editing (edit the YAML for both the Rome and London sites, according to the following steps).

   - On Red Hat OpenShift, edit the QueueManager YAML in the Red Hat OpenShift web console, or from the command line by running the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

     ```
     oc edit qmgr <QUEUEMANAGER_NAME>
     ```

   - On Amazon EKS, run the following command, replacing <QUEUEMANAGER_NAME> with the name of the queue manager:

     ```
     kubectl edit qmgr <QUEUEMANAGER_NAME>
     ```

   b) To specify a CipherSpec for both the traffic within and across groups, the `.spec.queuemanager.availability.tls.cipherSpec` attribute is added to the QueueManager YAML.

   Add the YAML below to the QueueManager YAML in both Rome and London sites to use the TLS_AES_256_GCM_SHA384 CipherSpec for all replication traffic:

   ```
   tls:
     secretName: nhacrr-rome-group
     cipherSpec: "TLS_AES_256_GCM_SHA384"
   ```

   If you need to specify a separate CipherSpec for the traffic across groups, the `.spec.queuemanager.availability.NativeHAGroups.local.tls.cipherSpec` attribute can be added to the QueueManager YAML.

   c) Add the following YAML to the QueueManager YAML for both Rome and London sites to use the TLS_AES_128_GCM_SHA256 CipherSpec for traffic between groups and TLS_AES_256_GCM_SHA384 CipherSpec for the traffic within the groups:

   ```
   nativeHAGroups:
     local:
    ...
       tls:
         key:
           secretName: nhacrr-rome-crr
         cipherSpec: "TLS_AES_128_GCM_SHA256"
   ```

   d) After the instances restart, issue the following command on the London site:

   - On Red Hat OpenShift:

     ```
     oc logs <ACTIVE_POD>
     ```

   - On Amazon EKS:

     ```
     kubectl logs <ACTIVE_POD>
     ```

You should see the messages in the log similar to the following example. The example shows that communication between the group instances is using CipherSpec TLS_AES_256_GCM_SHA384:

```
AMQ3212I: Native HA outbound secure connection established to 'exampleqm-ibm-mq-2'.
[CommentInsert1(exampleqm-ibm-mq-2), CommentInsert2(exampleqm-ibm-mq-replica-2(9414)),
CommentInsert3(TLS_AES_256_GCM_SHA384)]
```

This example shows that the connection to the Rome group is using CipherSpec TLS_AES_128_GCM_SHA256:

```
AMQ3253I: A Native HA recovery group connection to 'rome' has been established using the
'TLS_AES_128_GCM_SHA256' CipherSpec.
```

9. Allow plain text traffic within a group. Traffic between groups must always be secured and encrypted but the traffic within a group can be unencrypted. Set the `.spec.queuemanager.availability.tls.cipherSpec` to blank in the London site to remove the encryption for the London group's internal traffic:

   ```
         tls:
           cipherSpec: ''
           secretName: nhacrr-london-group
   ```

   After the instances restart, issue the following command on the London site:

   - On Red Hat OpenShift:

     ```
     oc logs <ACTIVE_POD>
     ```

   - On Amazon EKS:

     ```
     kubectl logs <ACTIVE_POD>
     ```

   The following message is shown in the log:

   `AMQ9722W: Plain text communication is enabled.`

10. Disable security for traffic within a group. Removing the `.spec.queuemanager.availability.tls` stanza completely disables security for a group's internal traffic.

    Note: Traffic between groups must be secure so if the `.spec.queuemanager.availability.tls.secretName` is removed `.spec.queuemanager.availability.NativeHAGroups.local.tls.key.secretName` must be specified.

    Disable the security for the Rome group by deleting the `.spec.queuemanager.availability.tls` stanza from the QueueManager YAML on the Rome site:

    ```
          tls:
            secretName: nhacrr-rome-group
            cipherSpec: "TLS_AES_256_GCM_SHA384"
    ```

    After the instances restart, issue the following command on the Rome site:

    - On Red Hat OpenShift:

      ```
      oc logs <ACTIVE_POD>
      ```

    - On Amazon EKS:

      ```
      kubectl logs <ACTIVE_POD>
      ```

    The log contains messages similar to the following:

    ```
    AMQ3213I: Native HA inbound connection accepted from 'exampleqm-ibm-mq-2'.
    [CommentInsert1(exampleqm-ibm-mq-2), CommentInsert2(<ip>)]
    ```

```
AMQ3211I: Native HA outbound connection established to 'exampleqm-ibm-mq-2'.
[CommentInsert1(exampleqm-ibm-mq-2), CommentInsert2(<replica>(9414))]
```

Before the change, the log contains messages similar to the following:

```
AMQ3214I: Native HA inbound secure connection accepted from 'exampleqm-ibm-mq-0'.
[CommentInsert1(exampleqm-ibm-mq-0), CommentInsert2(<ip>), CommentInsert3(<cipherspec>)]
AMQ3212I: Native HA outbound secure connection established to 'exampleqm-
ibm-mq-0'. [CommentInsert1(exampleqm-ibm-mq-0), CommentInsert2(<replica>(9414)),
CommentInsert3(<cipherspec)
```

Viewing the status of Native HA queue managers for IBM MQ containers

For IBM MQ containers, you can view the status of the Native HA instances by running the **dspmq** command inside one of the running Pods.

About this task

You can use the **dspmq** command in one of the running Pods to view the operational status of a queue manager instance. The information returned depends on whether the instance is active or a replica. The information supplied by the active instance is definitive, information from replica instances might be out of date.

For help when a node failure occurs, see Recovering from a node failure on Native HA.

For a Native HA Cross-Region Replication (CRR) configuration, the information supplied also depends on whether the instance is part of a Live or a Recovery group. The information from the Live group is more definitive, but can still be out of date as the Recovery group sends back only periodic information.

You can perform the following actions:

- View whether the queue manager instance on the current node is active or a replica.
- View the Native HA operational status of the instance on the current node.
- View the operational status of all three instances in a Native HA configuration.
- View the status of the groups that the instance belongs to or is connected to.

The following status fields are used to report Native HA and Native HA Cross-Region Replication (CRR) configuration status of a queue manager instance. These fields always appear:

ROLE
Specifies the current role of the instance. Is one of `Active`, `Replica`, Unknown, (for a recovery group) `Leader`, or `Not Configured`.

INSTANCE
The name provided for this instance of the queue manager when it was created using the **-lr** option of the **crtmqm** command.

INSYNC
Indicates whether the instance is able to take over as the active instance if required.

QUORUM
Reports the quorum status in the form *number_of_instances_in-sync/number_of_instances_configured*.

GRPLSN
Specifies the log sequence number (LSN) that has been replicated between a quorum of instances in the HA group, in *nnnnn:nnnnn:nnnnn:nnnnn* format. Blank if the LSN is not known.

GRPNAME
Specifies the name of the HA group. Blank if a name has not been configured.

GRPROLE
Specifies the current role of the HA group and is one of:

- `Live` - A live group
- `Recovery` - A recovery group

- `Pending live` - A group waiting to become `Live`
- `Pending recovery` - A group waiting to become `Recovery`
- `Not configured` - Native HA has not been configured
- `Unknown` - The current role cannot be determined

These fields appear when extended information is requested for all instances by using the `-x` parameter:

REPLADDR
The replication address of the queue manager instance.

CONNACTV
Indicates whether the instance is connected to the active instance.

BACKLOG
Indicates the number of KB that the instance is behind the active instance.

CONNINST
Indicates whether the named instance is connected to this instance.

ALTDATE
Indicates the date when this information was last updated. Blank if it has never been updated.

ALTTIME
Indicates the time when this information was last updated. Blank if it has never been updated.

V 10.0.0 **ACKLSN**
Indicates the LSN that the instance has acknowledged as having been written to the recovery log, in *nnnnn:nnnnn:nnnnn:nnnnn* format. Blank if the LSN value is not known.

V 10.0.0 **HASTATUS**

Indicates the operational status of this instance and is one of the following values:

- `Normal` - the instance is operating normally.
- `Checking` - the instance is being checked to ensure the consistency of the recovery log.
- `Synchronizing` - the instance is being sent data to bring it in-sync.
- `Rebasing` - a new recovery log is being sent to the instance.
- `Disk full` - the instance has a full disk.
- `Disconnected` - cannot communicate with the instance.
- `Unknown` - the instance status is unknown.

V 10.0.0 **SYNCTIME**
Indicates the time when this instance was last in-sync with the active queue manager, in ISO 8601 format. Blank if the time is not known.

V 10.0.0 These fields appear when group information is requested by using the `-g` parameter. These fields are useful to view the status when the group is part of a Native HA CRR configuration:

V 10.0.0 **GRPNAME**
Specifies the name of the HA group. Blank if a name has not been configured.

V 10.0.0 **GRPROLE**
Specifies the current role of the HA group and is one of:

- `Live` - A live group
- `Recovery` - A recovery group
- `Pending live` - A group waiting to become `Live`
- `Pending recovery` - A group waiting to become `Recovery`
- `Unknown` - The current role cannot be determined
- `Not configured` - Native HA has not been configured

GRPADDR
The IP address that is used to connect to the group. This is set to Unknown if the address is not known because a connection to the group has not occurred.

GRPVER
Indicates the version of the current group leader in `V.R.M.F` format, where `V.R.M.F` is the Version, Release, Modification, and Fix Pack number. If the value is not known, this is set to `?.?.?.?`

CONNGRP
Indicates whether the group is connected to the local group and is one of:

- `yes` - the group is connected.
- no - the group is disconnected.
- `unknown` - the connection status is unknown.
- `suspended` - the group is connected, but the group cannot replicate data until a configuration incompatibility between the groups is resolved.

This field is displayed only for groups other than the local group.

INSYNC
This is only displayed for a group with role `Recovery` or `Pending live`. It indicates whether the group could become `Live` with no loss of data if a failover occurred. (Note that a `Recovery` group might not be in-sync with the `Live` group very often because the log is replicated asynchronously. Use the RCOVLSN and RCOVTIME values to indicate if data loss might occur in an unplanned failover).

SYNCTIME
This is only displayed for a group with role `Recovery` or `Pending live`. It indicates the time when this group was last in-sync with the `Live` group, in ISO 8601 format (blank if it is not known). (Note that a `Recovery` group might not be in-sync with the `Live` group very often because the log is replicated asynchronously. Use the RCOVLSN and RCOVTIME values to indicate if data loss might occur in an unplanned failover).

BACKLOG
This is only displayed for a group with role `Recovery` or `Pending live`. It indicates the number of KB that the group is behind the live group.

GRSTATUS
Indicates the operational status of this group and is one of the following values:

- `Normal` - the group is operating normally.
- `Checking` - the group is being checked to ensure the consistency of the recovery log.
- `Synchronizing` - the group is being sent data to bring it in-sync.
- `Rebasing` - a new recovery log is being sent to the instance.
- `Waiting for connection` - the group is waiting for a connection from a group with the role `Live` or `Pending Recovery`.
- `Partitioned` - two groups believe they are the Live group (that is, a 'split brain' status).
- `Unknown` - the group status is unknown.

RCOVLSN
A log sequence number (LSN) that the group could recover to, in *nnnnn:nnnnn:nnnnn:nnnnn* format. Any data written to the recovery log after this point might be lost if an unplanned failover occurs. This value can be passed into the -s option of the **dmpmqlog** command.

RCOVTIME
A time that the group could recover to, in ISO 8601 format. Any data written to the recovery log after this time might be lost if an unplanned failover occurs. This value can be passed into the -t or -u options of the **dmpmqlog** command.

INITLSN
This is only displayed for a group with role `Live` or `Pending recovery`. It indicates the log sequence number (LSN) of the last log record recovered when the Native HA group initially became live, in the format *nnnnn:nnnnn:nnnnn:nnnnn*, or blank if the group is not live.

INITTIME
This is only displayed for a group with role `Live` or `Pending recovery`. It indicates the time of the last log record recovered when the Native HA group initially became live, in ISO 8601 format. Blank, if the group is not live.

LIVETIME
This is only displayed for a group with role `Live` or `Pending recovery`. It indicates the time the group first became the live group, in ISO 8601 format. Blank, if the group is not live.

ALTDATE
Indicates the date when this information was last updated.

ALTTIME
Indicates the time when this information was last updated.

Procedure

- Find the pods that are part of your queue manager.
  - On Red Hat OpenShift:

    ```
    oc get pod --selector app.kubernetes.io/instance=nativeha-qm
    ```

  - On Amazon EKS:

    ```
    kubectl get pod --selector app.kubernetes.io/instance=nativeha-qm
    ```

- Run the `dspmq` in one of the pods
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq
    ```

    For an interactive shell, where you can run `dspmq` directly.

    ```
    oc rsh <POD>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq
    ```

    For an interactive shell, where you can run `dspmq` directly.

    ```
    kubectl exec -it <POD> -- sh
    ```

- To determine whether a queue manager instance is running as the active instance or as a replica:
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o status -m <QMGR_NAME>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o status -m <QMGR_NAME>
    ```

  An active instance of a queue manager named BOB would report the following status:

  ```
  QMNAME(BOB)                STATUS(Running)
  ```

  An instance of BOB that is a recovery group leader in a Native HA CRR configuration would report the following status:

  ```
  QMNAME(BOB)                STATUS(Recovery group leader)
  ```

  A replica instance of a queue manager named BOB would report the following status:

```
QMNAME(BOB)                 STATUS(Replica)
```

An inactive instance would report the following status:

```
QMNAME(BOB)                 STATUS(Ended Immediately)
```

- To determine Native HA operational status of the instance in the specified pod:
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o nativeha -m <QMGR_NAME>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o nativeha -m <QMGR_NAME>
    ```

  The active instance of a queue manager named BOB might report the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(Yes) QUORUM(3/3) V 10.0.0
  GRPLSN(<0:25:365:1000>) GRPNAME() GRPROLE(Live)
  ```

  A replica instance of a queue manager named BOB might report the following status:

  ```
  QMNAME(BOB)         ROLE(Replica) INSTANCE(inst2) INSYNC(Yes) QUORUM(2/3) V 10.0.0
  GRPLSN(<0:25:365:1000>) GRPNAME() GRPROLE(Live)
  ```

  An inactive instance of a queue manager named BOB might report the following status:

  ```
  QMNAME(BOB)         ROLE(Unknown) INSTANCE(inst3) INSYNC(no) QUORUM(0/3) V 10.0.0 GRPLSN()
  GRPNAME() GRPROLE(Live)
  ```

  V 10.0.0 An active instance of a queue manager named BOB belonging to the Live group in a Native HA CRR configuration might report the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(yes) QUORUM(3/3)
  GRPLSN(<0:25:365:1000>) GRPNAME(alpha) GRPROLE(Live)
  ```

  V 10.0.0 The same instance when the group was waiting to become the recovery group might report the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(yes) QUORUM(3/3)
  GRPLSN(<0:25:365:1000>) GRPNAME(alpha) GRPROLE(Pending recovery)
  ```

  V 10.0.0 A queue manager named CAROL that is not an instance of a Native HA or Native HA CRR configuration would report the following status:

  ```
  QMNAME(CAROL)       ROLE(Not configured) INSTANCE() INSYNC() QUORUM() GRPLSN() GRPNAME()
  GRPROLE(Not configured)
  ```

- To determine the Native HA operational status of all the instances in the Native HA configuration:
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o nativeha -x -m <QMGR_NAME>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o nativeha -x -m <QMGR_NAME>
    ```

  For example, if you issue this command on the node running the active instance of queue manager BOB, where all instances are in-sync, you might receive the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(yes) QUORUM(2/3)
  GRPLSN(<0:25:365:1000>) GRPROLE(Live) GRPNAME()
   INSTANCE(inst1) ROLE(Active)  REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
  BACKLOG(0)  CONNINST(yes) ACKLSN(<0:26:365:1000>)  HASTATUS(Normal)
  SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
  ```

```
 INSTANCE(inst3) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN (<0:24:365:1000>) HASTATUS(Normal)
SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
 INSTANCE(inst2) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN (<0:25:365:1000>) HASTATUS(Synchronizing)
SYNCTIME(2023-12-20T17:13:22.123321Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

In another example, if you issue this command on a node running a replica instance of queue manager BOB, which is lagging behind, you might receive the following status:

```
QMNAME(BOB)         ROLE(Replica) INSTANCE(inst2) INSYNC(no) QUORUM(2/3)
GRPLSN(<0:25:365:1000>) GRPROLE(Live) GRPNAME()
 INSTANCE(inst1) ROLE(Active)  REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN(<0:26:365:1000>)  HASTATUS(Normal)
SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
 INSTANCE(inst3) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN (<0:24:365:1000>) HASTATUS(Normal)
SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
 INSTANCE(inst2) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(no)
BACKLOG(435) CONNINST(yes) ACKLSN (<0:25:365:1000>) HASTATUS(Synchronizing)
SYNCTIME(2023-12-20T17:13:22.123321Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

If you issue this command on a node running an inactive instance of queue manager BOB, you might receive the following status:

```
QMNAME(BOB)            ROLE(Unknown) INSTANCE(inst3) INSYNC(no) QUORUM(0/3) GRPLSN()
GRPROLE() GRPNAME()
 INSTANCE(inst1) ROLE(Unknown) REPLADDR(9.20.123.45) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ACKLSN()  HASTATUS(Unknown) SYNCTIME() ALTDATE() ALTTIME()
ALTDATE() ALTTIME()
 INSTANCE(inst2) ROLE(Unknown) REPLADDR(9.20.123.46) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ACKLSN()  HASTATUS(Unknown) SYNCTIME() ALTDATE() ALTTIME()
ALTDATE() ALTTIME()
 INSTANCE(inst3) ROLE(Unknown) REPLADDR(9.20.123.47) CONNACTV(No)      INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ACKLSN()  HASTATUS(Unknown) SYNCTIME() ALTDATE() ALTTIME()
ALTDATE() ALTTIME()
```

If you issue the command when the instances are still negotiating which is active and which are replicas, you would receive the following status:

```
QMNAME(BOB)             STATUS(Negotiating)
```

- V 10.0.0

  To determine the operational status of all the Native HA groups in a Native HA CRR configuration:

  – On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o nativeha -g -m <QMGR_NAME>
    ```

  – On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o nativeha -g -m <QMGR_NAME>
    ```

  If you issue this command on a member of the Live group, named alpha, you might receive the following status for alpha, and the Recovery group, beta:

```
GRPNAME(alpha) GRPROLE(Live) GRPADDR(9.20.135.129) GRPVER(09040200)
GRSTATUS(Normal) RCOVLSN(<0:0:104:65163>) RCOVTIME(2023-12-20T17:37:27.114787Z)
INITLSN(<0:0:101:352>) INITTIME(2023-12-19T09:13:27.512294)
LIVETIME(2023-12-19T06:14:27.512294) ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

```
GRPNAME(beta) GRPROLE(Recovery) GRPADDR(9.20.135.130) GRPVER(09040300)
CONNGRP(yes) GRSTATUS(Synchronizing) RCOVLSN(<0:0:19:16723>)
RCOVTIME(2023-12-20T17:36:23.463209Z) BACKLOG(5619) INSYNC(no) SYNCTIME()
ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

Related tasks

“Example: Configuring Native HA using the IBM MQ Operator” on page 119

This example deploys a queue manager using the native high availability feature by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Related reference

dspmq (display queue managers) command

OpenShift MQ Adv. *Advanced tuning for Native HA*

Advanced settings for tuning timings and intervals. There should be no need to use these settings unless the defaults are known not to match your system's requirements.

The basic options for configuring Native HA are handled using the `QueueManager` API, which the IBM MQ Operator uses to configure the underlying queue manager INI files for you. There are some more advanced options that are only configurable using an INI file, under the NativeHALocalInstance stanza. See also "Example: Supplying MQSC and INI files" on page 102 for more information on how to configure an INI file.

`HeartbeatInterval`
: The heartbeat interval defines how often in milliseconds an active instance of a Native HA queue manager sends a network heartbeat. The valid range of the heartbeat interval value is 500 (0.5 seconds) to 60000 (1 minute), a value outside of this range causes the queue manager to fail to start. If this attribute is omitted, a default value of 5000 (5 seconds) is used. Each instance must use the same heartbeat interval.

`HeartbeatTimeout`
: The heartbeat timeout defines how long a replica instance of a Native HA queue manager waits before it decides that the active instance is unresponsive. The valid range of the heartbeat interval timeout value is 500 (0.5 seconds) to 120000 (2 minutes). The value of the heartbeat timeout must be greater than or equal to the heartbeat interval.

  An invalid value causes the queue manager to fail to start. If this attribute is omitted a replica waits for 2 x `HeartbeatInterval` before starting the process to elect a new active instance. Each instance must use the same heartbeat timeout.

`RetryInterval`
: The retry interval defines how often in milliseconds a Native HA queue manager should retry a failed replication link. The valid range of the retry interval is 500 (0.5 seconds) to 120000 (2 minutes). If this attribute is omitted a replica waits for 2 x `HeartbeatInterval` before retrying a failed replication link.

OpenShift V 10.0.0 MQ Adv. *Advanced tuning for Native HA CRR*

Advanced settings for tuning timings and intervals. There should be no need to use these settings unless the defaults are known not to match your system's requirements.

The basic options for configuring Native HA CRR are handled using the `QueueManager` API, which the IBM MQ Operator uses to configure the underlying queue manager INI files for you. There are some more advanced options that are only configurable using an INI file, under the NativeHARecoveryGroup stanza of the qm.ini file . See also "Example: Supplying MQSC and INI files" on page 102 for more information on how to configure an INI file.

`ShortRetryTimer`
: Specifies the interval to wait between attempts to connect to a Recovery group in the short retry phase. Specified in milliseconds, defaults to 2500 (2.5 seconds).

`ShortRetryCount`
: Specifies the number of attempts to make a connection to the Recovery group during the short retry phase.

`LongRetryTimer`
: Specifies the interval to wait between attempts to connect to a Recovery group in the long retry phase. The long retry phase continues indefinitely until connection is made, or until the configuration

is changed to comment out or remove the `ReplicationAddress`. Specified in milliseconds, defaults to 60000 (60 seconds).

`RetryReporting`
Set to `Phase` or `All` to specify whether each retry attempt is reported, or just the phase (long or short). If not specified, defaults to `Phase`.

`StatusInterval`
Specifies the longest time interval that the Recovery group will wait before sending a Recovery log status. Specified in milliseconds, defaults to 10000 (10 seconds).

`StatusCommitThreshold`
Specifies the maximum number of bytes that the Recovery group commits before sending a Recovery log status. Specified in bytes and defaults to 0, which means that `StatusInterval` alone is used to determine when Recovery log status is sent.

Ending Native HA queue managers

For IBM MQ in containers, you can use the **endmqm** command to end an active or a replica queue manager that is part of a Native HA group.

Before you begin

Note: This information applies only to container environments.

About this task

The procedure for stopping a queue manager that is part of a Native HA group depends on whether it is an active or a replica instance. When you end either type of instance, a check is made to ensure that ending the instance does not break the quorum of the Native HA group. If the quorum would be broken, the **endmqm** command fails.

When you issue an **endmqm** command, the other instances in the group are warned that this is happening, so that they do not report errors when the connection breaks.

If an active instance loses quorum because of the ending or disconnection of too many replica instances, the active instance waits for a configurable amount of time before ending completely. This allows a period of time to shut down processing gracefully, rather than applications just having their connections broken. This timeout value can be specified by the `QuorumConnectivityTimeout` attribute in the `NativeHALocalInstance` stanza of the `qm.ini` file. The default value is 0 seconds.

Procedure

- To end the active instance of a queue manager, issue the following command on the node where the active instance is running:

  ```
  endmqm -s QMgrName
  ```

  – Specify the `-r` option to help client applications to reconnect to another instance.
  – If this instance is not the active instance in the Native HA group then the command fails.
  – If ending this active instance would cause the group quorum to fail then the command fails. (If other instances end or become unavailable at the same time as you run this command, the quorum check might not detect this, the Native HA group ends and can only be restarted when enough instances are available.)

  When the active queue manager ends, one of the replica instances takes over the active role. You cannot specify which replica takes over, this is determined by negotiation within the group and depends on which has the most up-to-date transaction logs.
- To end a replica instance of a queue manager, issue the following command:

  ```
  endmqm -x QMgrName
  ```

– If this instance is the active instance then the command fails.
– If ending this replica instance would cause the group quorum to fail then the command fails. (If other instances end or become unavailable at the same time as you run this command, the quorum check might not detect this, the Native HA group ends and can only be restarted when enough instances are available.)

Note: You can also use the -c, -i, -p, or -w switches with the **endmqm** command on Native HA instances, regardless of which role they are in. The queue manager instance ends, ignoring the effect that has on the group quorum. Information is still shared with the other instances in the group, however. You can use these switches together with the -s for the active instance. You cannot use these switches together with the -x switch for replica instances.

Configuring a multi-instance queue manager using the IBM MQ Operator

This example deploys a multi-instance queue manager by using the IBM MQ Operator. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Before you begin

To complete this example, you must first have completed the following prerequisites:

- Create a project/namespace for this example on your cluster.
- On the command line, log into your cluster, then switch to the above namespace.
- Ensure the IBM MQ Operator is installed and available in the above namespace.

About this task

This example provides a custom resource YAML defining a queue manager to be deployed. It also details the additional steps required to deploy the queue manager with TLS enabled.

Procedure

1. Determine a suitable storage class

   Storage in a Kubernetes cluster can be accessed using multiple Persistent Volume Access modes. A multi-instance queue manager creates multiple persistent volumes: one for each queue manager, and at least one shared volume. The shared volume for a multi-instance queue manager must use a `ReadWriteMany` storage class. The default storage class in a Kubernetes cluster is typically for a `ReadWriteOnce` storage class (block storage). For example, if you are using Red Hat OpenShift Data Foundation, the storage class *ocs-storagecluster-cephfs* provides a suitable shared file system. The choice of filesystem is very important, because not all shared file systems handle file locking in the same way. See Planning file system support on Multiplatforms and Testing statement for IBM MQ multi-instance queue manager file systems.
2. Create a pair of certificates as described in “Creating a self-signed PKI using OpenSSL” on page 107.
3. Create a config map containing MQSC commands and an INI file

   Create a Kubernetes ConfigMap containing the MQSC commands to create a new queue and a SVRCONN channel, and to add a channel authentication record that allows access to the channel.

   Ensure that you are in the namespace you created earlier (see Before you begin), then enter the following YAML by using the command line, or optionally by using the Red Hat OpenShift web console if you are using a Red Hat OpenShift cluster.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-miqm-configmap
data:
  example-tls.mqsc: |
    DEFINE CHANNEL('MTLS.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS13_OR_HIGHER') REPLACE
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
```

```
ACTION(REPLACE)
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=example-app1') USERSRC(MAP)
MCAUSER('app1') ACTION(REPLACE)
    SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)
    DEFINE QLOCAL('EXAMPLE.QUEUE') REPLACE
    SET AUTHREC PROFILE('EXAMPLE.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(BROWSE,PUT,GET,INQ)
  example-tls.ini: |
    Service:
        Name=AuthorizationService
        EntryPoints=14
        SecurityPolicy=UserExternal
```

The MQSC defines a channel called `MTLS.SVRCONN` and a queue called `EXAMPLE.QUEUE`. The channel is configured to allow access only to clients which present a certificate with a "common name" of `example-app1`. This is the common name used in one of the certificates created in Step “2” on page 165. Connections on this channel with this common name are mapped to a user ID of `app1`, which is authorized to connect to the queue manager, and to access the example queue. The INI file enables a security policy which means that the `app1` user ID does not need to exist in an external user registry — it exists only as a name in this configuration.

4. Deploy the queue manager

   Create a new queue manager by using the following custom resource YAML. Ensure you are in the namespace that you created before you begin this task.

   The availability type is set to `MultiInstance`, which causes persistent storage to be selected automatically.

   Change `STORAGE_CLASS` to the storage class that you identified in Step “1” on page 165.

   Note that the `example-qm-tls` Secret was created in Step “2” on page 165, and the `example-miqm-configmap` ConfigMap was created in Step “3” on page 165

   Note: The following YAML uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see “Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator” on page 187.

   Important: If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See “mq.ibm.com/v1beta1: Current license versions” on page 281 for details on the license.

   Example QueueManager for Red Hat OpenShift

   Deploy your queue manager with the following configuration, using the command line or using the Red Hat OpenShift Container Platform web console.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: MultiInstance
    mqsc:
    - configMap:
        name: example-miqm-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-miqm-configmap
        items:
        - example-tls.ini
    storage:
      defaultClass: STORAGE_CLASS
  version: 10.0.0.0-r3
  pki:
```

```
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Example QueueManager for Amazon EKS

Deploy your queue manager with the following configuration on an Amazon EKS cluster.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
spec:
  license:
    accept: false
    license: L-CXDB-XTZ9J8
    use: Production
  queueManager:
    name: EXAMPLEQM
    availability:
      type: MultiInstance
    mqsc:
    - configMap:
        name: example-miqm-configmap
        items:
        - example-tls.mqsc
    ini:
    - configMap:
        name: example-miqm-configmap
        items:
        - example-tls.ini
    storage:
      defaultClass: STORAGE_CLASS
    route:
      enabled: false
    metrics:
      serviceMonitor:
        enabled: false
  version: 10.0.0.0-r3
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
  web:
    enabled: false
    route:
      enabled: false
```

`spec.queueManager.route.enabled`, `spec.queueManager.metrics.serviceMonitor.enabled`, and `spec.web.route.enabled` are set to `false` because these are Red Hat OpenShift specific features that are enabled by default. These features must be disabled explicitly on Amazon EKS.

5. Confirm that the queue manager is running

   The queue manager is now being deployed. Confirm it is in `Running` state before proceeding. For example:

   - For deployments on the Red Hat OpenShift Container Platform:

     ```
     oc get qmgr exampleqm
     ```

   - For deployments on Amazon EKS:

     ```
     kubectl get qmgr exampleqm
     ```

6. Test the connection to the queue manager

To confirm the queue manager is configured for mutual TLS communication, follow the steps in the appropriate link for your platform:

- **Red Hat OpenShift**: "Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift" on page 112
- **Amazon EKS**: "Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS" on page 114

  Important: Amazon EKS users must complete the steps in "Configuring a LoadBalancer to connect to a queue manager for Amazon EKS deployments" on page 171 before continuing with this task.

7. Force the active pod to fail

   To validate the automatic recovery of the queue manager, simulate a pod failure:

   a) View the active and standby pods

   Run the following command:

   - For deployments on the Red Hat OpenShift Container Platform:

     ```
     oc get pods --selector app.kubernetes.io/instance=exampleqm
     ```

   - For deployments on Amazon EKS:

     ```
     kubectl get pods --selector app.kubernetes.io/instance=exampleqm
     ```

   Note that, in the **READY** field, the active pod returns the value `1/1`, whereas the standby pod returns the value `0/1`.

   b) Delete the active pod

   Run the following command, specifying the full name of the active pod:

   - For deployments on the Red Hat OpenShift Container Platform:

     ```
     oc delete <ACTIVE_POD> exampleqm-ibm-mq-<VALUE>
     ```

   - For deployments on Amazon EKS:

     ```
     kubectl delete <ACTIVE_POD> exampleqm-ibm-mq-<VALUE>
     ```

   c) View the pod status again (see step "7.a" on page 168).

   d) View the queue manager status

   Run the following command, specifying the full name of the other pod:

   - For deployments on the Red Hat OpenShift Container Platform:

     ```
     oc exec -t <POD> -- dspmq -x
     ```

   - For deployments on Amazon EKS:

     ```
     kubectl exec -t <POD> -- dspmq -x
     ```

   You should see the status shows that the active instance has changed, for example:

   ```
   QMNAME(EXAMPLEQM)                                         STATUS(Running as standby)
       INSTANCE(exampleqm-ibm-mq-1) MODE(Active)
       INSTANCE(exampleqm-ibm-mq-0) MODE(Standby)
   ```

   e) Test the connection to the queue manager again

   Note: You do not need to repeat the steps to configure the connection, only those to put and receive messages to or from the queue

   To confirm the queue manager has recovered, follow the steps in the appropriate link for your platform:

- **Red Hat OpenShift**: “Testing a mutual TLS connection to a queue manager from your laptop for Red Hat OpenShift” on page 112
- **Amazon EKS**: “Testing a mutual TLS connection to a queue manager from a client application on your laptop for Amazon EKS” on page 114

Results

You deployed a multi-instance queue manager with mutual TLS authentication, and verified that it automatically recovers when the active Pod fails.

Configuring connections to a queue manager from outside a cluster

You can connect to a queue manager from outside the cluster that the queue manager is deployed on.

Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster

You need a Red Hat OpenShift Route to connect an application to an IBM MQ queue manager from outside a Red Hat OpenShift cluster. You must enable TLS on your IBM MQ queue manager and client application because SNI is available in the TLS protocol only when a TLS 1.2 or higher protocol is used. The Red Hat OpenShift Container Platform Router uses SNI for routing requests to the IBM MQ queue manager.

About this task

Note: To configure connections to a queue manager from outside an Amazon EKS cluster, see “Configuring a LoadBalancer to connect to a queue manager for Amazon EKS deployments” on page 171.

The required configuration of the Red Hat OpenShift Route depends on the Server Name Indication (SNI) behavior of your client application. IBM MQ supports two different SNI header settings depending on configuration and client type. An SNI Header is set to the hostname of the client's destination or alternatively set to the IBM MQ channel name. For information on how IBM MQ maps a channel name to a hostname, see How IBM MQ provides multiple certificates capability.

Whether an SNI header is set to an IBM MQ channel name or a hostname is controlled using the **OutboundSNI** attribute. Possible values are `OutboundSNI=CHANNEL` (the default value) or `OutboundSNI=HOSTNAME`. For more information, see SSL stanza of the client configuration file. Note that `CHANNEL` and `HOSTNAME` are the exact values that you use; they are not variable names that you replace with an actual channel name or host name.

Client behaviors with different `OutboundSNI` settings

If **OutboundSNI** is set to `HOSTNAME`, the following clients set a hostname SNI as long as a hostname is provided in the connection name:

- C Clients
- .NET Clients in unmanaged mode
- Java/JMS Clients

If **OutboundSNI** is set to `HOSTNAME` and an IP address is used in the connection name, the following clients send a blank SNI header:

- C Clients
- .NET Clients in unmanaged mode
- Java/JMS Clients (that cannot do a reverse DNS lookup of the hostname)

If **OutboundSNI** is set to `CHANNEL`, or not set, an IBM MQ channel name is used instead and is always sent, whether a hostname or IP address connection name is used.

The following client types do not support setting an SNI header to an IBM MQ channel name, and so always attempt to set the SNI header to a hostname regardless of the **OutboundSNI** setting:

- AMQP clients
- XR Clients

The IBM MQ managed .NET client sets SERVERNAME to the respective hostname if the **OutboundSNI** property is set to HOSTNAME, which allows an IBM MQ managed .NET client to connect to a queue manager using Red Hat OpenShift routes.

If a client application connects to a queue manager deployed in a Red Hat OpenShift cluster through IBM MQ Internet Pass-Thru (MQIPT), MQIPT can be configured to set the SNI to the hostname by using the SSLClientOutboundSNI property in the route definition.

OutboundSNI, multiple certificates, and Red Hat OpenShift routes

IBM MQ uses the SNI header to provide multiple certificates functionality. If an application is connecting to an IBM MQ channel that is configured to use a different certificate through the CERTLABL field, then the application must connect with an **OutboundSNI** setting of CHANNEL.

If your Red Hat OpenShift Route configuration requires a HOSTNAME SNI then you are unable to use the multiple certificates functionality of IBM MQ and unable to set a CERTLABL setting on any IBM MQ channel object.

If an application with an **OutboundSNI** setting of anything other than CHANNEL connects to a channel with a certificate label configured, the application is rejected with an MQRC_SSL_INITIALIZATION_ERROR, and an AMQ9673 message is printed in the queue manager error logs.

For more information on how IBM MQ provides multiple certificate functionality, see How IBM MQ provides multiple certificates capability .

Example

Client applications that set the SNI to the MQ channel require a new Red Hat OpenShift Route to be created for each channel you wish to connect to. You also have to use unique channel names across your Red Hat OpenShift Container Platform cluster, to allow routing to the correct queue manager.

It is important that MQ channel names do not end in a lower-case letter because of the way IBM MQ maps channel names to SNI headers.

To determine the required host name for each of your new Red Hat OpenShift Routes, you need to map each channel name to an SNI address. See How IBM MQ provides multiple certificates capability for more information.

You must then create a new Red Hat OpenShift Route for each channel, by applying the following `yaml` in your cluster:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: unique_name_for_the_route
  namespace: namespace_of_your_MQ_deployment
spec:
  host: SNI_address_mapping_for_the_channel
  to:
    kind: Service
    name: name_of_Kubernetes_Service_for_your_MQ_deployment (for example "queue_manager_name-
ibm-mq")
  port:
    targetPort: 1414
  tls:
    termination: passthrough
```

Configuring your client application connection details

You can determine the host name to use for your client connection by running the following command:

```
oc get route Name of hostname based Route (for example "queue_manager_name-ibm-mq-qm")>
-n namespace of your MQ deployment -o jsonpath="{.spec.host}"
```

The port for your client connection should be set to the port used by the Red Hat OpenShift Container Platform Router - normally 443.

Related concepts

“Considerations for connecting to a queue manager from outside of your Kubernetes cluster ” on page 39

Related tasks

“Connecting to the IBM MQ Console” on page 244
How to connect to the IBM MQ Console of a queue manager that has been deployed with the IBM MQ Operator.

Configuring a `LoadBalancer` to connect to a queue manager for Amazon EKS deployments

You need a `LoadBalancer` to connect to a queue manager `Service` from outside an Amazon EKS cluster.

Before you begin

Note: To configure connections to a queue manager from outside a Red Hat OpenShift cluster, see “Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster ” on page 169.

To complete this procedure, you must first complete the following prerequisites:

- Deploy a queue manager configured with TLS in the same namespace as the IBM MQ Operator. For example, by following the steps in “Example: Configuring a queue manager with mutual TLS authentication” on page 109.
- Install the AWS Load Balancer Controller onto your Amazon EKS cluster.
- Set your namespace to the one in which you deployed the queue manager. This setting avoids the need to specify the namespace in each command. For example, run the following command:

```
kubectl config set-context --current --namespace=<QUEUE_MANAGER_NAMESPACE>
```

About this task

This task describes how to set up a `LoadBalancer` to enable a mutual TLS connection from your laptop to a queue manager on an Amazon EKS cluster. A `LoadBalancer` is a type of Kubernetes `Service` that you can use to provide external access to a `Service` on a system by routing TCP traffic.

This task does not cover configuring access to a queue manager's IBM MQ Console, REST API, or metrics from outside of a cluster. You configure such access by using a Kubernetes `Ingress`, as described in “Configuring Ingress to connect to a queue manager IBM MQ Console and IBM MQ Operator REST API from outside an Amazon EKS cluster” on page 173.

Procedure

1. Confirm that the queue manager is running.

   Confirm that the queue manager's `Phase` is `Running` before proceeding. For example, run the following command:

   ```
   kubectl get qmgr exampleqm
   ```

2. Create a `LoadBalancer` `Service` to facilitate a TLS connection to a queue manager from outside your cluster.

   Create the `LoadBalancer` in the same namespace as your queue manager. Create a file (for example, `loadbalancer.yaml`) that contains the following YAML, setting the `LOAD_BALANCER_NAME` value to a unique name for the `LoadBalancer`.

   ```
   apiVersion: v1
   kind: Service
   ```

```
metadata:
  name: LOAD_BALANCER_NAME
  annotations:
    service.beta.kubernetes.io/aws-load-balancer-type: external
    service.beta.kubernetes.io/aws-load-balancer-scheme: internet-facing
    service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: instance
spec:
  type: LoadBalancer
  ports:
    - port: 1414
      targetPort: 1414
      name: qmgr
  selector:
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/instance: exampleqm
```

Important: If you are using an **IPv6 single-stack** system, make the following changes to the `metadata.annotations` section:

- Change the value of `service.beta.kubernetes.io/aws-load-balancer-nlb-target-type` from "`instance`" to "`ip`"
- Add the following line:

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: "dualstack"
```

The values within `spec.ports` identify the port information of your queue manager, such as port 1414, which is used for queue manager traffic. You can verify these values by checking the queue manager `Service` of your queue manager. For example, by running the following command, where `QUEUE_MANAGER_SERVICE` is the queue manager `Service` of your queue manager.:

```
kubectl get service <QUEUE_MANAGER_SERVICE> -o jsonpath="{.spec.ports[]}{'\n'}"
```

This `Service` has the form `<QUEUE_MANAGER_NAME>-ibm-mq`; you can find it by running the following command:

```
kubectl get service
```

The values within `spec.selector` are used to identify your queue manager by matching against Kubernetes `Labels` on a `Service`. The values in the `LoadBalancer` should match those in the queue manager `Service` of your queue manager. You can find these values by running the following command:

```
kubectl get service <QUEUE_MANAGER_SERVICE> -o jsonpath="{.metadata.labels}{'\n'}"
```

For more information about Load Balancers in Amazon Web Services, see Getting started with Network Load Balancers in the Amazon Web Services documentation.

3. Apply the `LoadBalanceer` YAML file. For example, by running the following command:

```
kubectl apply -f loadbalancer.yaml
```

4. Verify that the `Service` of type `LoadBalancer` was created:

```
kubectl get service <LOAD_BALANCER_NAME>
```

5. Check that your `LoadBalancer` is ready.

   The Amazon Load Balancer Controller takes several minutes to provision and register its targets. Run the following command to show the status of the `LoadBalancer`. When the `LoadBalancer` is ready, the status is `active`:

```
aws elbv2 describe-load-balancers --query 'LoadBalancers[*].[LoadBalancerName,State.Code]' --
output text
```

6. Retrieve the `LoadBalancer` hostname:

```
kubectl get service <LOAD_BALANCER_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

Results

You created a `LoadBalancer` `Service` to enable connections to your queue manager from outside your Amazon EKS cluster.

Configuring Ingress to connect to a queue manager IBM MQ Console and IBM MQ Operator REST API from outside an Amazon EKS cluster

You need an `Ingress` to connect to a queue manager IBM MQ Console and REST API from outside an Amazon EKS cluster.

Before you begin

Note: To configure connections to a queue manager from outside a Red Hat OpenShift cluster, see "Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster " on page 169.

To complete this example, you must first complete the following prerequisites:

- Deploy a queue manager in your Amazon EKS cluster by using the IBM MQ Operator, as described in "Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS" on page 100.
- Configure the queue manager with credentials to access the IBM MQ Console and REST API. For an example of configuring credentials to access the IBM MQ Console, see "Configuring the IBM MQ Console with a basic registry using the IBM MQ Operator" on page 193. For more information about configuring credentials for the IBM MQ Console and REST API, see IBM MQ Console and REST API security.
- Install the AWS Load Balancer Controller onto your cluster.
- Ensure that the OpenSSL command line tool is installed.
- Ensure that the `curl` command line tool is installed.
- Set your namespace to the one in which you deployed the queue manager. This setting avoids the need to specify the namespace in each command. For example, run the following command:

```
kubectl config set-context --current --namespace=<QUEUE_MANAGER_NAMESPACE>
```

About this task

An `Ingress` is a Kubernetes resource type that you can use to provide external access to a `Service` on a system by routing HTTP or HTTPS traffic. A queue manager has one `Service` for the IBM MQ Console and REST API, and another for queue manager metrics. The following instructions describe how to set up an `Ingress` to access the IBM MQ Console and REST API `Services` from outside an Amazon EKS cluster.

This task does not cover creating mutual TLS connections from a client application to a queue manager. To create such connections, use a `LoadBalancer` as described in "Configuring a LoadBalancer to connect to a queue manager for Amazon EKS deployments" on page 171.

Procedure

1. Create certificates then import them to AWS Certificate Manager (ACM).

   Note: The `CN` for the `CA` and cert must be fully qualified domain names, such as `example.com`. If you have a valid certificate in ACM that meets this requirement, you can proceed to step "2" on page 174.

   a) Create a private key to use for your internal certificate authority.

   ```
   openssl genpkey -algorithm rsa -pkeyopt rsa_keygen_bits:4096 -out ca.key
   ```

   A private key for the internal certificate authority is created in a file called `ca.key`. Keep this file safe and secret; it is used to sign certificates for your internal certificate authority.

   b) Issue a self-signed certificate for your internal certificate authority.

```
openssl req -x509 -new -nodes -key ca.key -sha512 -days 30 -subj "/CN=example-selfsigned-
ca.com" -out ca.crt
```

- CN must be a fully qualified domain name
- -days specifies the number of days that the root CA certificate will be valid

A certificate is created in a file called ca.crt. This certificate contains the public information about the internal certificate authority, and is freely shareable.

c) Create a private key and certificate signing request for a queue manager.

```
openssl req -new -nodes -out example-qm.csr -newkey rsa:4096 -keyout example-qm.key -subj
'/CN=example-qm.com'
```

- CN must be a fully qualified domain name

A private key is created in a file called example-qm.key and a certificate signing request is created in a file called example-qm.csr.

d) Sign the queue manager key with your internal certificate authority.

```
openssl x509 -req -in example-qm.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out
example-qm.crt -days 7 -sha512
```

- -days specifies the number of days that the certificate will be valid.

A signed certificate is created in a file called example-qm.crt.

a) Import the created certificates to ACM:

```
aws acm import-certificate --certificate fileb://example-qm.crt --certificate-chain
fileb://ca.crt --private-key fileb://example-qm.key
```

This command returns output that includes the imported certificate's CertificateArn. For example:

```
{
  "CertificateArn": "arn:aws:acm:us-west-2:xxxxx:certificate/xxxxxxx"
}
```

You use this value to create the Ingress in the next step.

2. Create an Ingress to facilitate your connection to the IBM MQ Console and REST API through HTTPS.

a) Create the Ingress in the same namespace where your queue manager is deployed.

Create a file (for example, ingress.yaml) that contains the following YAML, changing these values:

- INGRESS_NAME: a unique name for the Ingress
- CERTIFICATE_ARN: the Amazon Resource Name (ARN) of a certificate that is stored in AWS Certificate Manager. If you completed step “1” on page 173, the ARN is the value that was returned when you ran the final command in that step. The value is the complete string, of the form arn:aws:acm:us-west-2:<xxxxx>:certificate/<xxxxxxx>. For more information, see View AWS Certificate Manager certificate details in the Amazon Web Services documentation.
- QUEUE_MANAGER_SERVICE: the queue manager Service of your queue manager. This value is of the form <QUEUE_MANAGER_NAME>-ibm-mq. You can find the value by running the following command:

```
kubectl get service -n <QUEUE_MANAGER_NAMESPACE>
```

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: <INGRESS_NAME>
  annotations:
    alb.ingress.kubernetes.io/scheme: internet-facing
    alb.ingress.kubernetes.io/target-type: ip
```

```
    alb.ingress.kubernetes.io/healthcheck-path: /actuator/health/liveness
    alb.ingress.kubernetes.io/listen-ports: '[{"HTTPS":443}]'
    alb.ingress.kubernetes.io/certificate-arn: <CERTIFICATE_ARN>
    alb.ingress.kubernetes.io/backend-protocol: HTTPS
spec:
  ingressClassName: alb
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: <QUEUE_MANAGER_SERVICE>
                port:
                  number: 9443
```

Important: If you are using an **IPv6 single-stack** system, add the following line to the `metadata.annotations` section:

```
service.beta.kubernetes.io/aws-load-balancer-ip-address-type: "dualstack"
```

The `Ingress` exposes the queue manager's IBM MQ Console and REST API on port 9443.

b) Apply the `Ingress` YAML configuration. For example:

```
kubectl apply -f ingress.yaml
```

c) Verify that the `Ingress` was created:

```
kubectl get ingress <INGRESS_NAME> -n <QUEUE_MANAGER_NAMESPACE>
```

d) Check that your `Ingress` is ready.

The Amazon Load Balancer Controller takes several minutes to provision and register its targets. Run the following command to show the status of the `Ingress`. When the `Ingress` is ready, the status is `active`:

```
aws elbv2 describe-load-balancers --query 'LoadBalancers[*].
[LoadBalancerName,State.Code]' --output text
```

If you encounter issues with the `Ingress`, including the `Ingress` not getting an `Address`, view the Load Balancer Controller pod log. For example:

```
kubectl logs aws-load-balancer-controller-<xxxxxxxxxx>-<yyyyy> -n kube-system
```

3. Access the IBM MQ Console or REST API by using your `Ingress`.

a) Retrieve the `Ingress` `Address`:

```
kubectl get ingress -n <QUEUE_MANAGER_NAMESPACE> <INGRESS_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

b) Access the IBM MQ Console.

Use the `Address` to access the IBM MQ Console from a browser. In a browser tab, navigate to `https://<INGRESS_ADDRESS>`.

c) Access the REST API.

Use the `Address` to interact with the REST API for your queue manager, with your credentials. For example, run the following command:

```
curl -k -X GET -u <USERNAME>:<PASSWORD> https://<INGRESS_ADDRESS>/ibmmq/rest/v1/admin/
qmgr/<QM_NAME>
```

For more information about using the REST API, see Getting started with the messaging REST API.

Results

You configured an `Ingress` then used it to connect to your queue manager from outside your Amazon EKS cluster.

Configuring audit logging for IBM MQ in containers

IBM MQ in containers provides various facilities for audit logging of different events.

Depending on the audit logging requirements for your environment, you might need to configure one or more of these facilities as well as suitable mechanisms for collecting and retaining the event logs. Choose from the following options.

IBM MQ Operator validating webhook

By default, the IBM MQ Operator outputs access audit events to exposed endpoints to the pod's container log. No configuration is required.

IBM MQ events

You can configure IBM MQ to produce event messages for auditing purposes. For more information see Auditing.

MQ container metrics

The IBM MQ container exposes a Prometheus metrics endpoint (for more information, see “Monitoring when using the IBM MQ Operator” on page 244). You can enable logging of access to this endpoint by setting the `MQ_LOGGING_METRICS_AUDIT_ENABLED` environment variable to `true`. If you are using the IBM MQ Operator, you can set this variable in the `env` section of the `queueManager` configuration:

```
[...]
spec:
  queueManager:
    env:
    - name: MQ_LOGGING_METRICS_AUDIT_ENABLED
      value: "true"
[...]
```

When logging is enabled, log files are generated in the `/var/mqm/errors` directory. You can persist data that is in this directory by mounting external storage to `/mnt/mqm` (see container image reference). If you are using the IBM MQ Operator, you can instead map external storage by using the “.spec.queueManager.storage.persistedData” on page 299 fields. Metrics access log files are written in JSON format. The filenames have the form `metricaudit<nn>.json`, where nn is a number. The `metricaudit01.json` file contains the most recent events.

Embedded IBM WebSphere Application Server Liberty webserver

You can configure the embedded WebSphere Liberty webserver to produce audit events. The `audit-1.0` and `audit-2.0` features are installed and available for configuration, depending on your requirements. For more information, see Audit 1.0 and Audit 2.0 in the WebSphere Liberty documentation.

You can enable and configure the feature by using a configuration file. If you are using the IBM MQ Operator, you can load the file by using the “.spec.web.manualConfig” on page 308 mounts. For example, the following code enables default configuration of the Audit 2.0 feature.

Create a configuration file called `mqwebuser.xml`, to enable the `audit-2.0` feature:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: example-enable-audit
data:
  mqwebuser.xml: |
    <?xml version="1.0" encoding="UTF-8"?>
    <server>
      <featureManager>
        <feature>audit-2.0</feature>
```

```
      </featureManager>
    </server>
```

Add the previous code to the `queueManager` configuration by using the `spec.web.manualConfig` mount:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
[...]
spec:
  [...]
  web:
    enabled: true
    manualConfig:
      configMap:
        name: example-enable-audit
```

Note: By default, audit logs are mirrored to the container log in basic (plain text) format. This format contains a reduced set of audit information. To get more information, either change the IBM MQ container log to JSON format or disable the web log mirroring and gather the WebSphere Liberty JSON audit log file directly. You can control the log format and log mirroring sources by using the `MQ_LOGGING_CONSOLE_FORMAT` and `MQ_LOGGING_CONSOLE_SOURCE` environment variables. For valid values, see "Environment variables supported" on page 322.

OpenShift CP4I Integrating IBM MQ with IBM Instana tracing

You can use IBM Instana to trace transactions.

Before you begin

Important:

- This feature is not available with IBM MQ Advanced for Developers.
- This feature is supported only on the Red Hat OpenShift Container Platform.

This document covers IBM Instana tracing, which is the process of tracing messages through a system. It does not cover IBM Instana monitoring, in which detail is retrieved about the state of an IBM MQ queue manager. For information regarding monitoring of IBM MQ by IBM Instana see Monitoring IBM MQ. For detailed instructions on authenticated monitoring, see "Configuring authenticated IBM Instana monitoring with TLS" on page 179.

Note: This feature is not supported when using IPv6 networking.

Before you can perform IBM Instana tracing with the IBM MQ Operator, you must deploy both an IBM Instana backend and IBM Instana agents. By default, an IBM MQ queue manager communicates with an IBM Instana agent deployed on the same node as the queue manager pod.

About this task

Enabling integration with IBM Instana causes an IBM MQ API exit to be installed in your queue manager. The API exit sends tracing data to IBM Instana agents about messages that are flowing through the queue manager.

The API exit adds RFH2 headers to each message. These headers contain tracing information.

The IBM Instana agents are responsible for sending the tracing data to the IBM Instana backend.

For information about deploying an IBM Instana backend and IBM Instana agents, see Enabling Instana monitoring links in the Platform UI in the IBM Cloud Pak for Integration documentation.

Procedure

Standard Deployment

- Deploy a queue manager with IBM Instana tracing enabled.

By default, IBM Instana tracing is disabled.

If you are using the IBM Cloud Pak for Integration Platform UI or the OpenShift web console:

1. Click **Operators** > **Installed Operators** > **IBM MQ** > **Details** > **Create instance**. Provide all necessary details for your queue manager.
2. Under the same location, click **Telemetry** > **Tracing** > **Instana**.
3. Set the **Enable Instana tracing** toggle to `true`.

If you are deploying through YAML, use the following snippet:

```
spec:
  telemetry:
    tracing:
      instana:
        enabled: true
```

Advanced Deployment

- Communicate with the IBM Instana agent over https.

  By default, the IBM Instana exit for IBM MQ communicates with the IBM Instana agent over http. The agent's host address is set to the IP address of the node the queue manager is running on. This matches the configuration described in Enabling IBM Instana monitoring in the IBM Cloud Pak for Integration documentation, where IBM Instana agents are deployed by the IBM Instana Agent Operator as a daemonset.

  Currently the communication between the IBM Instana exit for IBM MQ and the IBM Instana agent supports http or https protocols. To use https, the IBM Instana agent must first be configured to use TLS encryption. See Setting up TLS encryption for Agent endpoint in the IBM Instana documentation. The protocol can then be set to https as follows:

  If you are using the OpenShift web console:

  1. Click **Operators** > **Installed Operators** > **IBM MQ** > **Details** > **Create instance**. Provide all necessary details for your queue manager.
  2. Under the same location, click **Telemetry** > **Instana**.
  3. Expand the **Advanced configuration** dropdown list.
  4. Set the **Instana agent communication protocol** to `https`.

  If you are deploying through YAML, use the following snippet:

  ```
  spec:
    telemetry:
      instana:
        enabled: true
        protocol: https
  ```

- Set the **`agentHost`**

  If IBM Instana agents have not been deployed as a daemonset on the Openshift cluster where the queue manager is running, then you must set the **`agentHost`** value to the hostname or IP address where the IBM Instana agent is running. The **`agentHost`** value should not include a protocol or port.

  If you are using the OpenShift web console:

  1. Click **Operators** > **Installed Operators** > **IBM MQ** > **Details** > **Create instance**. Provide all necessary details for your queue manager.
  2. Under the same location, click **Telemetry** > **Instana**.
  3. Expand the **Advanced configuration** dropdown list.
  4. Type your hostname into the **Instana agent host** text box.

  If you are deploying through YAML, use the following snippet:

  ```
  spec:
    telemetry:
  ```

```
    instana:
      enabled: true
      agentHost: 9.9.9.9
```

What to do next

See also “Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97.

OpenShift CP4I Operator2.2.0 Configuring authenticated IBM Instana monitoring with TLS

To be able to monitor a queue manager through an IBM Instana agent on Red Hat OpenShift, you must configure both the agent and the queue manager.

Before you begin

Important: Integration with IBM Instana tracing is supported **only** on the Red Hat OpenShift Container Platform.

The "Configuration" section of "Monitoring IBM MQ" in the IBM Instana documentation provides general information regarding IBM Instana monitoring configuration. However it does not include details on configuring the queue manager.

Before you can perform IBM Instana tracing with the IBM MQ Operator, you must deploy both an IBM Instana backend and IBM Instana agents. To do this, see Enabling IBM Instana monitoring in the CP4I Platform UI in the IBM Instana documentation.

Procedure

1. Generate certificates.
2. Configure the IBM Instana agents.
3. Configure the queue manager.
4. Verify and debug.

Related tasks

“Integrating IBM MQ with IBM Instana tracing” on page 177
You can use IBM Instana to trace transactions.

OpenShift CP4I Operator2.2.0 *Generate a certificate and key for the IBM Instana agent and the queue manager*

For TLS communication between the IBM Instana agent and the queue manager on Red Hat OpenShift, both must have a certificate and corresponding private key.

Before you begin

Important: Integration with IBM Instana tracing is supported **only** on the Red Hat OpenShift Container Platform.

This is the first of four tasks to configure authenticated IBM Instana monitoring with TLS.

Note: The values used in the generation of these certificates are for demonstration purposes. When deploying in a production environment ensure that the subject and expiry of the certificate are appropriate.

Procedure

IBM MQ Queue manager

To communicate with the IBM Instana agent through TLS, the queue manager must have a certificate and corresponding private key. If you already have these, then skip this section.

1. Generate a certificate and private key for the queue manager.

   Run the following command:

```
openssl req \
       -newkey rsa:2048 -nodes -keyout server.key \
       -subj "/CN=mq queuemanager/OU=ibm mq" \
       -x509 -days 3650 -out server.crt
```

IBM Instana agent

For the agent to perform TLS communication with the IBM MQ queue manager, the agent must have a certificate and corresponding private key. If you already have a private key and certificate in a JKS keystore that you would like to use, then skip this section.

2. Generate a certificate and private key for the IBM Instana agent.

   Run the following command:

```
openssl req \
       -newkey rsa:2048 -nodes -keyout application.key \
       -subj "/CN=instana-agent/OU=app team1" \
       -x509 -days 3650 -out application.crt
```

3. Store the certificate and private key in a PKCS12 keystore.

   Run the following command, replacing *your_password* with the password you want to use to secure the keystore. Perform this replacement in all subsequent steps.

```
openssl pkcs12 -export -out application.p12 -inkey application.key -in application.crt
-passout pass:your_password
```

4. Convert the PKCS12 keystore into a JKS keystore.

   Run the following command:

```
keytool -importkeystore \
        -srckeystore application.p12 \
        -srcstoretype pkcs12 \
        -destkeystore application.jks \
        -deststoretype JKS \
        -srcstorepass your_password \
        -deststorepass your_password \
        -noprompt
```

5. Label the certificate.

   Run the following command:

```
keytool -changealias -alias "1" -destalias "instana" -keypass your_password -keystore
application.jks -storepass your_password -noprompt
```

6. Import the queue manager certificate into the keystore.

   Run the following command:

```
keytool -importcert -file server.crt -keystore application.jks -storepass your_password
-alias myca -noprompt
```

What to do next

You are now ready to configure the agents for IBM Instana monitoring.

OpenShift CP4I Operator2.2.0 *Instana monitoring: Configuring agents*

Mount the keystore to the IBM Instana agents, then configure monitoring for a specific queue manager on the Red Hat OpenShift Container Platform.

Before you begin

Important: Integration with IBM Instana tracing is supported **only** on the Red Hat OpenShift Container Platform.

This task assumes that you have generated a certificate and key for the IBM Instana agents and the queue manager.

Procedure

Mounting the keystore to the IBM Instana agents

1. Create a secret from your JKS keystore in the IBM Instana agent namespace.

   Run the following command, replacing *keystore_secret_name* with the name you want to use. Perform this replacement in all subsequent steps.

   ```
   oc create secret generic keystore_secret_name --from-file=./application.jks -n instana-agent
   ```

2. Mount the secret as a volume into the Instana agent pod.

   Do this by specifying the volume and volume mounts in the agent `CustomResource` or in the `values.yaml` Helm chart:

   ```
   agent:
     pod:
       volumeMounts:
       - mountPath: /opt/instana/agent/etc/jks-file-name.jks
         name: mq-key-jks-name
         subPath: jks-file-name.jks
       volumes:
       - name: mq-key-jks-name
         secret:
           secretName: keystore_secret_name
   ```

   To make this edit to an Instana agent that is deployed by the Instana Agent Operator, run a command similar to the following:

   ```
   oc edit agent [instana-agent-name] -n [instana-agent-namespace]
   ```

   For more information about mounting secrets into the Instana agent pod, see Volumes and volumeMounts in the Instana agent Helm chart documentation.

Configuring monitoring for a specific queue manager

3. In the instana-agent namespace, use the `oc edit configmap instana-agent` command to edit the instana-agent configmap.
4. Add the following section under `configuration.yaml: |`. If you have already defined this section, then just add the new queue manager to the list.

   ```
   com.instana.plugin.ibmmq:
     enabled: true
     poll_rate: 60
     queueManagers:
        QUEUE_MANAGER_NAME:
           channel: 'INSTANA.A.SVRCONN'
           keystorePassword: 'your_password'
           keystore: '/opt/instana/agent/etc/application.jks'
           cipherSuite: 'TLS_RSA_WITH_AES_256_CBC_SHA256'
   ```

   where

   - *your_password* is the password to your JKS keystore

- *QUEUE_MANAGER_NAME* is the name of the underlying IBM MQ queue manager to deploy, rather than the name of the Queue Manager Operand.

  Note: If *QUEUE_MANAGER_NAME* is not set to the underlying queue manager name, and is instead set to the Operand, monitoring will not work. The underlying name is defined in `spec.queuemanager.name` for the Queue Manager Operand.

5. Delete the instana-agent pods in the instana-agent namespace. This causes them to restart, and to begin monitoring with the new settings.

What to do next

You are now ready to configure the queue manager for IBM Instana monitoring.

Instana monitoring: Configuring the queue manager

Set up a queue manager that uses TLS to communicate with the IBM Instana agent on the Red Hat OpenShift Container Platform. The authentication for this connection is done using an SSLPEERMAP.

Before you begin

Important: Integration with IBM Instana tracing is supported **only** on the Red Hat OpenShift Container Platform.

This task assumes that you have configured the agents for IBM Instana monitoring.

Procedure

1. Configure the queue manager through both MQSC and INI.

   MQSC is used to set up a new TLS enabled channel, and then configure that channel to authenticate the connecting IBM Instana agent if it has a certificate with the required fields. In this case, we map any connecting client with a certificate containing the fields `CN=instana-agent,OU=app team1` to user `app1`. MQSC then grants permission for user `app1` to perform the required operations for IBM Instana monitoring.

   The INI file is used to grant permissions to our external user `app1`.

   The following configmap contains the required MQSC and INI settings. Deploy it into your queue manager namespace.

```
apiVersion: v1
data:
  channel.mqsc: |-
    DEFINE CHANNEL('INSTANA.A.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS12_OR_HIGHER')
    ALTER QMGR CONNAUTH(' ')
    REFRESH SECURITY
    SET CHLAUTH('INSTANA.A.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
ACTION(REPLACE)
    SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS) ACTION(REPLACE)
    SET CHLAUTH('INSTANA.A.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=instana-agent,OU=app
team1') USERSRC(MAP) MCAUSER('app1')
    SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(ALL)
    SET AUTHREC PROFILE('SYSTEM.ADMIN.COMMAND.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(PUT,INQ,DSP,CHG)
    SET AUTHREC PROFILE('SYSTEM.**') PRINCIPAL('app1') OBJTYPE(TOPIC) AUTHADD(DSP)
    SET AUTHREC PROFILE('*') PRINCIPAL('app1') OBJTYPE(TOPIC) AUTHADD(DSP)
    SET AUTHREC PROFILE('SYSTEM.**') PRINCIPAL('app1') OBJTYPE(QUEUE) AUTHADD(DSP, CHG, GET)
    SET AUTHREC PROFILE('SYSTEM.**') PRINCIPAL('app1') OBJTYPE(LISTENER) AUTHADD(DSP)
    SET AUTHREC PROFILE('AMQ.*') PRINCIPAL('app1') OBJTYPE(QUEUE) AUTHADD(DSP, CHG)
    REFRESH SECURITY TYPE(CONNAUTH)
  auth.ini: |-
    Service:
      Name=AuthorizationService
      EntryPoints=14
      SecurityPolicy=UserExternal
kind: ConfigMap
metadata:
```

```
  namespace: your-queue-manager-namespace
  name: qmgr-monitoring-config
```

where *your-queue-manager-namespace* is the namespace in which your queue manager will be deployed.

Note: If you are monitoring user-defined queues then you must add additional lines to the configmap MQSC, granting DSP, CHG and GET permissions to those queues. For example:

```
SET AUTHREC PROFILE('MYQUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE) AUTHADD(DSP, CHG, GET).
```

This example uses a configmap for the MQSC and INI data, but you can use a secret if any additions you make are confidential. For general information regarding deploying with MQSC and INI, see “Example: Supplying MQSC and INI files” on page 102.

2. For a TLS connection to be made, the queue manager must trust the certificate of the IBM Instana agent. To achieve this, create a secret containing just the certificate of the IBM Instana agent:

```
oc create secret generic instana-certificate-secret --from-file=./application.crt -n your-
queue-manager-namespace
```

3. The queue manager must present its own certificate for the TLS handshake, and requires access to the associated private key. Deploy a secret containing the key and certificate that you either created earlier or already possess:

```
oc create secret tls qm-tls-secret --cert server.crt --key server.key -n your-queue-manager-
namespace
```

With the configmap and secret created, you are ready to create the queue manager itself.

4. Ensure that your queue manager YAML does not set the environment variable **MQSNOAUT** in the queue manager container.

Otherwise, after it is enabled, the authentication mechanism will not work. Removing the variable after deployment does not cause the mechanism to be re-enabled, and the queue manager has to be recreated.

5. Add the following sections to your queue manager definition, where *MYQM* is the name of your queue manager:

```
spec:
  queueManager:
    name: MYQM   #(a)
    ini:   #(b)
      - configMap:
          items:
            - auth.ini
          name: qmgr-monitoring-config
    mqsc:   #(c)
      - configMap:
          items:
            - channel.mqsc
          name: qmgr-monitoring-config
  pki:
    keys:   #(d)
        - name: default
          secret:
            items:
            - tls.key
            - tls.crt
            secretName: qm-tls-secret
    trust:   #(e)
      - name: app
        secret:
          items:
            - application.crt
          secretName: instana-certificate-secret
```

The flagged sections of the specification are described as follows:

a. Ensure that you have given your underlying queue manager a unique name. If the underlying queue manager does not have a unique name, then monitoring might not work as intended. This name must match the name in the IBM Instana agent configmap that was edited earlier.
b. The INI information that was written to the configmap is added to the queue manager.
c. The MQSC information that was written to the configmap is added to the queue manager.
d. The queue manager certificate and private key are added to the queue manager keystore.
e. The IBM Instana agent certificate is added to the queue manager trust store.

6. Optional: Enable IBM Instana Tracing on your monitored queue manager.

   If you want to do this, see "Integrating IBM MQ with IBM Instana tracing" on page 177.

7. Deploy the queue manager.

What to do next

You are now ready to verify and debug the IBM Instana monitoring.

Instana monitoring: Verifying and debugging

To be able to monitor a queue manager through an IBM Instana agent on the Red Hat OpenShift Container Platform, you must configure both the agent and the queue manager.

Before you begin

Important: Integration with IBM Instana tracing is supported **only** on the Red Hat OpenShift Container Platform.

This task assumes that you have configured the queue manager for IBM Instana monitoring.

Procedure

Verifying

1. To verify that you have been successful in your deployment, view your queue manager in the IBM Instana dashboard.

   The queue manager should be visible in the services section of the application page, and also in the Infrastructure view.

Debugging

Note: These debugging steps assume an Openshift deployment of the IBM Instana agent running as a daemonset.

If you cannot see your queue manager in the IBM Instana dashboard, then you might have misconfigured your queue manager. Use the following steps to investigate.

2. Identify the node on which your active queue manager pod is running.

   Run the following command in your queue manager namespace:

   ```
   oc get pods -o wide -n your-queue-manager-namespace
   ```

3. To determine which IBM Instana agent pod is running on the same node as your queue manager, run the same command in the instana-agent namespace:

   ```
   oc get pods -o wide -n instana-agent-namespace
   ```

4. To help understand any issues from the IBM Instana agent side, get the logs of the IBM Instana agent pod and look for entries relating to 'mq' or to the name of your queue manager.

   Run the following command:

   ```
   oc logs instana-agent-pod -c instana-agent -n instana-agent
   ```

5. Check the queue manager logs.

If the agent has made an attempt to connect to the queue manager, then the queue manager logs should indicate why the connection was not successful. Run the following command:

```
oc logs your-queue-manager-name -n your-queue-manager-namespace
```

Results

You have completed all four tasks to configure authenticated IBM Instana monitoring with TLS.

OpenShift CP4I Building an image with custom MQSC and INI files, using the Red Hat OpenShift CLI

Use an Red Hat OpenShift Container Platform Pipeline to create a new IBM MQ container image, with MQSC and INI files you want to be applied to queue managers using this image. This task should be completed by a project administrator

Before you begin

You need to install the Red Hat OpenShift Container Platform command-line interface.

Log into your cluster using **`cloudctl login`** (for IBM Cloud Pak for Integration), or **`oc login`**.

If you don't have a Red Hat OpenShift Secret for the IBM Entitled Registry in your Red Hat OpenShift project, then follow the steps for Create the entitlement key secret.

Procedure

1. Create an `ImageStream`

   An image stream and its associated tags provide an abstraction for referencing container images from within Red Hat OpenShift Container Platform. The image stream and its tags allow you to see what images are available and ensure that you are using the specific image you need even if the image in the repository changes.

   ```
   oc create imagestream mymq
   ```

2. Create a `BuildConfig` for your new image

   A `BuildConfig` will allow builds for your new image, which will be based off the IBM official images, but will add any MQSC or INI files you want to be run on container start-up.

   a) Create a YAML file defining the `BuildConfig` resource

   For example, create a file called "mq-build-config.yaml" with the following contents:

   ```
   apiVersion: build.openshift.io/v1
   kind: BuildConfig
   metadata:
     name: mymq
   spec:
     source:
       dockerfile: |-
         FROM cp.icr.io/cp/ibm-mqadvanced-server-integration:10.0.0.0-r3
         RUN printf "DEFINE QLOCAL(foo) REPLACE\n" > /etc/mqm/my.mqsc \
           && printf "Channels:\n\tMQIBindType=FASTPATH\n" > /etc/mqm/my.ini
         LABEL summary "My custom MQ image"
     strategy:
       type: Docker
       dockerStrategy:
         from:
           kind: "DockerImage"
           name: "cp.icr.io/cp/ibm-mqadvanced-server-integration:10.0.0.0-r3"
         pullSecret:
           name: ibm-entitlement-key
     output:
       to:
         kind: ImageStreamTag
         name: 'mymq:latest-amd64'
   ```

You will need to replace the two places where the base IBM MQ is mentioned, to point at the correct base image for the version and fix you want to use (see “Release history for IBM MQ Operator” on page 5 for details). As fixes are applied, you will need to repeat these steps to re-build your image.

This example creates a new image based on the IBM official image, and adds files called "my.mqsc" and "my.ini" into the `/etc/mqm` directory. Any MQSC or INI files found in this directory will be applied by the container at start-up. INI files are applied using the **`crtmqm -ii`** option, and merged with the existing INI files. MQSC files are applied in alphabetical order.

It is important that your MQSC commands are repeatable, as they will be run *every time* the queue manager starts up. This typically means adding the `REPLACE` parameter on any `DEFINE` commands, and adding the `IGNSTATE(YES)` parameter to any `START` or `STOP` commands.

b) Apply the `BuildConfig` to the server.

```
oc apply -f mq-build-config.yaml
```

3. Run a build to create your image

a) Start the build

```
oc start-build mymq
```

You should see output similar to the following:

```
build.build.openshift.io/mymq-1 started
```

b) Check the status of the build
For example, you can run the following command, using the build identifier returned in the previous step:

```
oc describe build mymq-1
```

4. Deploy a queue manager, using your new image

Follow the steps described in “Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97, adding your new custom image into the YAML.

You could add the following snippet of YAML into your normal `QueueManager` YAML, where *my-namespace* is the Red Hat OpenShift project/namespace you are using, and *image* is the name of the image you created earlier (for example, "mymq:latest-amd64"):

```
spec:
  queueManager:
    image: image-registry.openshift-image-registry.svc:5000/my-namespace/my-image
```

Related tasks

“Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97
This example deploys a "quick start" queue manager using the IBM MQ Operator on Red Hat OpenShift. The queue manager uses ephemeral (non-persistent) storage, and turns off IBM MQ security. Messages

are not persisted across restarts of the queue manager. You can adjust the configuration to change many queue manager settings.

Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator

To use an IBM MQ license (base entitlement), certain license annotations must be applied to queue managers. How these licenses are applied depends on the version of your IBM MQ Operator and the version of your IBM MQ Operator queue managers.

Before you begin

Note: These instructions apply only if you are using the IBM MQ Operator. If you are self-deploying (in other words, without using the IBM MQ Operator) queue managers on Kubernetes, see IBM MQ container license annotations instead.

About this task

You might not need to complete this task, depending on the license that you want to use:

- If you want to use an IBM MQ or IBM MQ for Non-Production Environment license, you do not need to complete this task. Instead, select your license when you deploy a queue manager, as described in “Deploying a simple queue manager using the IBM MQ Operator on Red Hat OpenShift” on page 97 or “Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS” on page 100. The IBM MQ Operator applies the necessary annotations for you.
- If you want to use an IBM Cloud Pak for Integration license with your base IBM MQ license, complete the steps in this task to specify both licenses when you deploy a queue manager. The IBM MQ Operator then applies the relevant annotations for you.
- If you want to add a Native HA and Cross-Region Replication Add-On license to an IBM MQ licensed queue manager, you do not need to complete this task. Instead, follow the steps in “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124.

Important: Updating the queue manager YAML triggers a pod restart.

Procedure

- If you want to use an IBM Cloud Pak for Integration license with your base IBM MQ license, specify both licenses when you deploy a queue manager:
  - Specify an IBM Cloud Pak for Integration license in the `spec.license.license` field.
  - Specify an IBM MQ license in the `spec.license.additionalLicenses` field.

  For information about the available licenses, see “Licensing and API references for the IBM MQ Operator” on page 281.

  The following example shows how to configure a queue manager with an IBM Cloud Pak for Integration license and an IBM MQ additional license, for production use:

```
...
spec:
  license:
    accept: false
    license: L-VEMB-RABZSN
    additionalLicenses:
      - L-MKDD-7KHY2Q
    use: Production
...
```

  When you deploy the queue manager, the IBM MQ Operator applies one of the following annotations for you, depending on the licenses that you specified:

- IBM MQ with CP4I entitlement
- IBM MQ for Non-Production Environment with CP4I entitlement
- IBM MQ Container Multi Instance with CP4I entitlement
- IBM MQ Container Multi Instance for Non-Production Environment with CP4I entitlement

In the previous example, the operator applies the IBM MQ with CP4I entitlement annotation.

Adding annotations to queue managers by using the IBM MQ Operator

You can add annotations to queue managers either on creation, or by updating existing queue managers by using the `spec.annotations`.

About this task

These instructions apply only if both the following statements are true:

- You are using the IBM MQ Operator. If you are self-deploying queue managers on Kubernetes, see "IBM MQ container license annotations" on page 314 instead.
- You are editing an existing queue manager. For new queue managers, add the annotations on creation instead.

Important: Updating the queue manager YAML triggers a pod restart.

Procedure

- **Option 1:** Add annotations by using the command line interface (CLI).
  a) Log into your cluster from the command line.
  b) Set your `project` (namespace) to the one in which your queue manager is deployed:
    - For deployments on the Red Hat OpenShift Container Platform:

      ```
      oc project <QUEUE-MANAGER-NAMESPACE>
      ```

    - For deployments on Amazon EKS:

      ```
      kubectl config set-context --current --namespace=<QUEUE-MANAGER-NAMESPACE>
      ```

  c) Identify the name of the queue manager that you want to update:
    - For deployments on the Red Hat OpenShift Container Platform:

      ```
      oc get queuemanager
      ```

    - For deployments on Amazon EKS:

      ```
      kubectl get queuemanager
      ```

  d) Add the required annotations to your queue manager by editing the queue manager:
    - For deployments on the Red Hat OpenShift Container Platform

      ```
      oc edit queuemanager <QUEUE-MANAGER-NAME>
      ```

    - For deployments on Amazon Amazon EKS:

      ```
      kubectl edit queuemanager <QUEUE-MANAGER-NAME>
      ```

  e) Repeat these steps for other queue managers as required.
- **Option 2:** Add annotations by using the Red Hat OpenShift console, for deployments on Red Hat OpenShift.
  a) Navigate to your queue manager

a. Log in to the Red Hat OpenShift console with your Red Hat OpenShift Container Platform cluster administrator credentials.
b. Change the **Project** to the namespace where you installed the IBM MQ Operator. Select the namespace from the **Project** drop-down list.
c. In the navigation pane, click **Operators** > **Installed Operators**.
d. In the list on the Installed Operators panel, find and click **IBM MQ**.
e. Click the **Queue Manager** tab.
f. Click the required **Queue Manager name**.

b) Add the required annotations to your queue manager.

a. Click the **YAML** tab.
b. Locate the `spec` field.
c. Add the required annotations to the queue manager. Create the `annotations` field if it is not already present.
d. Click **Save**.

The queue manager pod restarts and the annotations are applied.

c) Repeat these steps for other queue managers as required.

OpenShift CP4I Adding custom annotations and labels to queue manager resources

You add custom annotations and labels to the QueueManager metadata.

About this task

Custom annotations and labels are added to all resources except PVCs. If a custom annotation or label matches an existing key, the value set by the IBM MQ Operator is used.

Procedure

- Add custom annotations.

  To add custom annotations to queue manager resources, including the pod, add the annotations under `metadata`. For example:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: quickstart-cp4i
  annotations:
    annotationKey: "value"
```

- Add custom labels.

  To add custom labels to queue manager resources, including the pod, add the labels under `metadata`. For example:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: quickstart-cp4i
  labels:
    labelKey: "value"
```

Disabling runtime webhook checks on Red Hat OpenShift

Runtime webhook checks on Red Hat OpenShift ensure that the storage classes are viable for your queue manager. You disable them to improve performance, or because they are not valid for your environment.

About this task

Runtime webhook checks are done on the queue manager configuration. They check that the storage classes are suitable for your selected queue manager type.

You might choose to disable these checks to decrease time taken for queue manager creation, or because the checks are not valid for your specific environment.

Note: After you disable runtime webhook checks, any storage class values are allowed. This could result in a broken queue manager.

Procedure

- Disable runtime webhook checks.

  Add the following annotation under `metadata`. For example:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: quickstart-cp4i
  annotations:
    "com.ibm.cp4i/disable-webhook-runtime-checks" : "true"
```

Disabling default value updates to the queue manager specification

The IBM MQ Operator updates any unspecified values in the queue manager specification with their default values. You can disable this behavior if you want to avoid any modifications to the queue manager specification. The queue manager status fields are still updated.

Procedure

- Disable queue manager default value updates.

  Add the following annotation under `metadata`. For example:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: quickstart-cp4i
  annotations:
    "com.ibm.mq/write-defaults-spec" : "false"
```

  Note: The quickstart examples have this annotation applied by default.

Running the IBM MQ container with a read-only root file system

You can configure the IBM MQ container to run with a read-only root file system. This prevents attackers from copying and running malicious code in the container.

About this task

Enabling the read-only root file system makes the container files immutable. That is, on the container file system, files can be viewed but not modified and no new files can be created. Files can only be modified or created on a mounted file system.

Scratch and Tmp volumes

Data to be written or modified is divided into two ephemeral volumes: Scratch and Tmp. These volumes are created and mounted into the /run and /tmp directories respectively, in the queue manager container.

- The Scratch volume contains the files, keystores, and other files that are used for configuring the queue manager.
- The Tmp volume contains diagnostic files, for example the queue manager RAS files.

When a read-only root file system is enabled and the **enabled** property of the Scratch or Tmp volume is set to true (the default value), an ephemeral volume of type emptyDir is created and mounted to the queue manager container for the respective ephemeral data.

V 10.0.0 You can disable the Scratch and Tmp volumes. When you set the **enabled** property of the Scratch or Tmp volume to false, a sub-directory of the queue manager data volume (/scratch or /tmp) is mounted to the queue manager container for the respective ephemeral data. If the queue manager data volume is of **type** persistent-claim, the queue manager data volume is a Persistent Volume Claim. If the **type** is set to ephemeral, the volume is an emptyDir volume.

When an emptyDir ephemeral volume is created and used for Scratch, Tmp, or the queue manager data, any files on this volume are lost on pod restart. If the size of the data in any of these volumes exceeds the value specified for the volume's **sizeLimit** property, Kubernetes can eject the container and create a new one. The **sizeLimit** property applies only to emptyDir ephemeral volumes.

V 10.0.0 When Scratch or Tmp data is stored on the queue manager data volume (when the **enabled** property is set to false), and the data volume is a Persistent Volume Claim, this data is by default considered ephemeral because the queue manager container erases this data on startup. You can disable the deletion of Tmp data by setting the MQ_ENABLE_CLEAN_TMP_ON_START environment variable to false. For information about setting queue manager environment variables, see “Example: Supplying additional environment variables for a queue manager” on page 106.

Considerations for choosing a backing volume

Data for an emptyDir volume is stored on the backing medium of the node the associated pod is running on. Although the aforementioned **sizeLimit** property can limit the use of this storage, the emptyDir volume might run out of capacity before this limit is reached, if the backing medium is filled from another source. V 10.0.0 If the size of the backing medium for the node is insufficient, you might want to store Scratch and Tmp data on the queue manager data volume.

V 10.0.0 Storing Scratch and Tmp data on the queue manager data volume has other consequences. This storage might be accessed over the network, which means access to Scratch and Tmp data is slower. The default size of this volume is 2Gi. The default **sizeLimit** for Scratch and Tmp volumes are 100M and 2Gi respectively, so you should increase the size of the data volume when storing this additional data. If the size of the volume is not large enough, Scratch and Tmp data might consume the volume, preventing the queue manager from functioning normally. Setting MQ_ENABLE_CLEAN_TMP_ON_START to false might also lead to this problem, if the data in /tmp is added to over time.

Procedure

A read-only root file system is not enabled by default. To enable it, complete one of the following procedures:

- Enable a read-only root file system with emptyDir ephemeral volumes.
  a) Enable a read-only root file system by setting the **readOnlyRootFilesystem** property under “.spec.securityContext” on page 302 to true:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
...
spec:
```

```
  securityContext:
    readOnlyRootFilesystem: true
```

b) Optional: Set or change the queue manager data storage type.

By default, a Persistent Volume Claim is mounted at `/mnt/mqm`. Or, if the **type** property is set to `ephemeral` in ".spec.queueManager.storage.queueManager" on page 300, an ephemeral volume is created and mounted.

c) For each ephemeral volume, carefully consider by how much the data might grow. Set the value of the **sizeLimit** property accordingly, including SI units.

- For the `Scratch` ephemeral volume, set the **sizeLimit** property in ".spec.queueManager.storage.scratch" on page 301. The default value is `100M`.
- For the `Tmp` ephemeral volume, set the **sizeLimit** property in ".spec.queueManager.storage.tmp" on page 301. The default value is `2Gi`.
- If the **type** of queue manager volume is set to `ephemeral`, set the **sizeLimit** property in ".spec.queueManager.storage.queueManager" on page 300. The default value is `2Gi`.

d) Optional: Although the default value of the **enabled** property for both ".spec.queueManager.storage.scratch" on page 301 and ".spec.queueManager.storage.tmp" on page 301 is `true`, you can also explicitly set the value to `true`.

In either case, `emptyDir` ephemeral volumes are created and used.

- V 10.0.0

Enable a read-only root file system by using the queue manager data volume.

a) Enable a read-only root file system by setting the `readOnlyRootFilesystem` property under ".spec.securityContext" on page 302 to `true`:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
...
spec:
  securityContext:
    readOnlyRootFilesystem: true
```

b) Configure the queue manager data volume.

- Set the **type** to `persistent-claim` to ensure that the queue manager data volume is not an `emptyDir` ephemeral volume.
- Set the **class** to be a StorageClass of type ReadWriteOnce because Security-Enhanced Linux (SELinux) might prevent sockets from being written to other types of storage.
- Carefully consider by how much the data might grow on the volume. Account for the queue manager data and `Scratch` and `Tmp` data. The default value for **size** is `2Gi`, so set a larger value that includes `Scratch` and `Tmp` requirements. Set the value of **size** accordingly, including SI units.
- Refer to ".spec.queueManager.storage.queueManager" on page 300 to set any additional values for the data volume.

For example:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
...
spec:
  queueManager:
    storage:
      queueManager:
        type: persistent-claim
        size: 5Gi
```

c) Set the **enabled** property for both ".spec.queueManager.storage.scratch" on page 301 and ".spec.queueManager.storage.tmp" on page 301 to `false`.
For example:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
...
spec:
  queueManager:
    storage:
      queueManager:
        type: persistent-claim
        size: 5Gi
      scratch:
        enabled: false
      tmp:
        enabled: false
```

The queue manager data volume has the sub-directories of `/scratch` and `/tmp` mounted into the queue manager container.

OpenShift Amazon EKS Configuring the IBM MQ Console with a basic registry using the IBM MQ Operator

To log in to the IBM MQ Console, you can supply your own configuration to the queue manager.

Before you begin

If you are deploying a queue manager with an IBM MQ Advanced for Developers license, there is a simple configuration built in. See "Example queue manager YAML that describes how to specify passwords for admin and app users" on page 327. If you are deploying an IBM Cloud Pak for Integration license queue manager on the Red Hat OpenShift Container Platform **only**, you can enable integration with the IBM Cloud Pak for Integration Keycloak to log in to the IBM MQ Console using Single Sign-On. See "Connecting to the IBM MQ Console" on page 244.

Procedure

1. **Create a password and hash it using `securityUtility`.**

   A `ConfigMap` can be used to store the credentials that you use to access your queue manager. Alternatively you can use a Secret, which protects credentials in the Kubernetes layer. However, monitoring or troubleshooting tools might expose the underlying file insecurely.

   For improved security with either option, hash these credentials by using the `securityUtility encode` command with the `--encoding=hash` option.

2. **Log into your cluster from the command line.**

   Alternatively, if you are using Red Hat OpenShift, you can use the Red Hat OpenShift console.

3. **Create a `ConfigMap` with your configuration.**

   For help with creating the XML configuration, see IBM MQ Console and REST API security.

   The following example creates a user within the group `MQWebAdminGroup`. Members of the `MQWebAdminGroup` are assigned the `MQWebAdmin` role. In this example:

   - You **must** replace the *USERNAME* and *PASSWORD* with your own values. Note that *USERNAME* is used twice in the example.

     You **must** specify the *NAMESPACE* as the one in which your IBM MQ Operator is deployed and where your queue manager will be, or already is, deployed.

   a) Use the command line, or the Red Hat OpenShift console for deployments on the Red Hat OpenShift Container Platform, to create the following `ConfigMap`:

```
kind: ConfigMap
apiVersion: v1
metadata:
  name: mqwebuserconfigmap
  namespace: NAMESPACE
data:
```

```
mqwebuser.xml: |
  <?xml version="1.0" encoding="UTF-8"?>
  <server>
    <featureManager>
      <feature>appSecurity-2.0</feature>
      <feature>basicAuthenticationMQ-1.0</feature>
    </featureManager>
    <enterpriseApplication id="com.ibm.mq.console">
      <application-bnd>
        <security-role name="MQWebAdmin">
          <group name="MQWebAdminGroup" realm="defaultRealm"/>
        </security-role>
      </application-bnd>
    </enterpriseApplication>
    <basicRegistry id="basic" realm="defaultRealm">
      <user name="USERNAME" password="PASSWORD"/>
      <group name="MQWebAdminGroup">
        <member name="USERNAME"/>
      </group>
    </basicRegistry>
  </server>
```

b) Optional: If using the command line, apply the `ConfigMap`:

- For deployments on the Red Hat OpenShift Container Platform:

```
oc apply -f mqwebuserconfigmap.yaml
```

- For deployments on Amazon EKS:

```
kubectl apply -f mqwebuserconfigmap.yaml
```

For the remaining steps, choose one of the following options:

- Deploy a new queue manager with the configuration to access the IBM MQ Console.
- Apply configuration that gives the IBM MQ Console access to an existing queue manager.

4. Optional: **Deploy a new queue manager with the configuration to access the IBM MQ Console.**

a) Create your queue manager.

Set the authentication and authorization providers to `manual` and supply the newly created `ConfigMap` `mqwebuserconfigmap` though one of the following options:

- Option 1: Through the queue manager YAML

For deployments on Red Hat OpenShift, add the following code under the `web` section of the queue manager YAML:

```
...
web:
    enabled: true
    console:
      authentication:
        provider: manual
      authorization:
        provider: manual
    manualConfig:
      configMap:
        name: mqwebuserconfigmap
```

For deployments on Amazon EKS, add the following code under the `web` section of the queue manager YAML:

```
...
web:
  enabled: true
  console:
    authentication:
      provider: manual
    authorization:
      provider: manual
  manualConfig:
    configMap:
      name: mqwebuserconfigmap
```

```
  route:
    enabled: false
```

`spec.web.route.enabled` is set to `false` because it is a Red Hat OpenShift specific feature that is enabled by default. This feature must be disabled explicitly on Amazon EKS.

- Option 2: Through the OpenShift console Form view:

  i) On the OpenShift console, select **Operators** > **Installed Operators**.

  ii) Select your deployment of the IBM MQ Operator.

  iii) Select **Queue Manager** and click **Create QueueManager**.

  iv) Select the relevant options for your queue manager.

  v) Select **Web** and set **Enable web server** to `true`.

  vi) Open the **Advanced configuration** list box.

  vii) Under the **Console** list box, set the **provider** for both **Authentication** and **Authorization** to `manual`.

  viii) Open the **Configuration** list box.

  ix) Open the **ConfigMap** list box and select the ConfigMap `mqwebuserconfigmap` that was created in step "3" on page 193.

  x) Click **Create**.

  You can now access the IBM MQ Console of your new queue manager through the credentials specified in the `ConfigMap` created in step "3" on page 193.

5. Optional: **Apply configuration that enables the IBM MQ Console for an existing queue manager.**

   There are two options for applying the configuration:

   - Option 1: by using the CLI

     OCP: `oc edit qmgr <QUEUEMANAGER_NAME>`

     EKS: `kubectl edit qmgr <QUEUEMANAGER_NAME>`

   - Option 2: OCP users, by using the OCP console

     a. On the Red Hat OpenShift console, select **Operators** > **Installed Operators**.

     b. Select your deployment of the IBM MQ Operator.

     c. Select **Queue Mananger** and select the name of your queue manager.

     d. Select **YAML**.

     e. Edit the YAML and click **Save**

   - For both options, replace the existing web section of the queue manager YAML with the following code:

     For deployments on Red Hat OpenShift:

```
...
web:
    enabled: true
    console:
      authentication:
        provider: manual
      authorization:
        provider: manual
    manualConfig:
      configMap:
        name: mqwebuserconfigmap
```

     For deployments on Amazon EKS:

```
...
web:
  enabled: true
  console:
```

```
    authentication:
      provider: manual
    authorization:
      provider: manual
  manualConfig:
    configMap:
      name: mqwebuserconfigmap
  route:
    enabled: false
```

`spec.web.route.enabled` is set to `false` because it is a Red Hat OpenShift specific feature that is enabled by default. This feature must be disabled explicitly on Amazon EKS.

6. **Amazon EKS deployments only:** Configure an `Ingress` to connect to the IBM MQ Console.

   Follow the steps in "Configuring Ingress to connect to a queue manager IBM MQ Console and IBM MQ Operator REST API from outside an Amazon EKS cluster" on page 173.

Results

You can now access the IBM MQ Console of your queue manager through the credentials specified in the `ConfigMap` that you created in step "3" on page 193.

OpenShift Expanding persistent volumes

If your storage provider supports volume expansion, use this task to expand a persistent volume. Depending on the storage provider, expansion might occur online or offline.

Before you begin

Successful volume expansion relies on your storage provider to fulfill the expansion request. Refer to your storage providers documentation to determine if online resizing is supported, and for information about offline resizing procedures.

If your storage provider cannot fulfill the expansion request, your Persistent Volume Claim might enter a state with warnings or errors. If expansion fails, an administrator can manually recover the Persistent Volume Claim state and cancel the expansion.

- For Red Hat OpenShift users, see Recovering from failure when expanding volumes in the Red Hat OpenShift documentation.
- For Amazon EKS users, see Recovery From Volume Expansion Failure in the Kubernetes blog.

About this task

To help with managing persistent storage, Kubernetes defines two API resources:

- A PersistentVolume (PV), which is a piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes. It can be provisioned statically or dynamically.
- A PersistentVolumeClaim (PVC), which is a request for storage by a user. It also acts as a claim check to the resource.

For more information, see Persistent Volumes in the Kubernetes documentation.

Warning:

- If the storage class used to create queue manager PVCs does not support online resizing, offline resizing takes place. During offline resizing user intervention is required to complete volume expansion, so queue managers experience downtime.
- For offline resizing of shared volumes for multi-instance queue managers, both active and standby pods must be brought down at the same time when performing the user intervention.
- Kubernetes platforms, including Red Hat OpenShift, do not support reducing the size of PVCs. Attempting to reduce the size of persistent volumes will put the queue manager into a 'Failed' state.

- This procedure does not apply to ephemeral volumes.

To expand a PV that is used by the IBM MQ container, complete the following steps.

Procedure

1. **Prepare to expand volumes**

   a) Decide which volumes to expand.

   b) Determine the storage class or classes being used by your volumes.

   For example:

```
spec:
  queueManager:
    storage:
      persistedData:
        enabled: true
        type: persistent-claim
        class: ocs-storagecluster-cephfs (1)
      queueManager:
        type: persistent-claim
      recoveryLogs:
        enabled: true
        type: persistent-claim
      defaultClass: ocs-storagecluster-ceph-rbd (2)
```

   Notes:

   - (1) If the volume defines a specific storage class, then this is used by PVCs of this type.
   - (2) If **`defaultClass`** is set, this storage class is used for all volumes without a specific storage class. If **`defaultClass`** is not set, and a volume type has not specified a class, then the default storage class for the cluster is used.

   You can also confirm the storage class in use by describing the underlying PVCs. For example:

   - For deployments on the Red Hat OpenShift Container Platform:

```
oc describe pvc <PVC_NAME>
```

   - For deployments on Amazon EKS:

```
kubectl describe pvc <PVC_NAME>
```

   c) Validate that your storage class supports volume expansion.

   A storage class might have the property **`.allowVolumeExpansion`** defined:

   - If this property is set to `true`, then volume expansion is supported.
   - If this property is set to `false`, or this property is not defined, then the storage class does not allow volume expansion. In this case, refer to your storage provider documentation to see if this feature can be enabled.

   You can also describe a storage class to determine if it supports volume expansion. For example:

   - For deployments on the Red Hat OpenShift Container Platform:

```
oc describe sc <STORAGE_CLASS_NAME>
```

   - For deployments on Amazon EKS:

```
kubectl describe sc <STORAGE_CLASS_NAME>
```

   d) Refer to your storage provider documentation to see if an online or offline procedure is used for volume expansion.

   An offline procedure requires queue manager pods to be manually restarted, whereas an online procedure does not. Refer to your storage provider documentation for offline resizing procedures.

e) Check if your queue manager has a status condition with the reason 'StorageMismatch'.

If your queue manager has this status condition, the volumes listed in the condition are expanded if you enable volume expansion. If you do not want this to happen, change the size fields associated with each volume type in your queue manager definition to match the provisioned PVCs. The status condition is removed when this is done for all mismatched volumes.

2. **Expand volumes**

Warning:

- If you have previously modified any of the volume size fields in your queue manager definition, volumes begin expanding when **`.allowVolumeExpansion`** is set to `true` in your queue manager definition.
- Your storage provider might have restrictions on the maximum size of a volume because of file system limitations or availability of local hardware. To avoid failures, validate these limitations in your storage provider documentation before you expand volumes.
- Reductions in PVC size are not supported by Kubernetes platforms, including Red Hat OpenShift. If you expand the size of a volume you cannot reduce it. If your attempt to do so fails, the IBM MQ Operator cannot return the PVC to its original state.

Example queue manager definition illustrating volume expansion:

```
spec:
  queueManager:
    storage:
      allowVolumeExpansion: true (A)
      persistedData:
        enabled: true
        type: persistent-claim
        size: 3Gi (B)
      queueManager:
        type: persistent-claim
        size: 4Gi (B)
      recoveryLogs:
        enabled: true
        type: persistent-claim
        size: 3Gi (B)
```

a) To allow volume expansion for the queue manager, set the field **`.spec.queueManager.storage.allowVolumeExpansion`** (A) on your queue manager to `true`.

b) You can now increase the size fields (B) for any of your enabled volume types. Applying these changes will start volume expansion.

3. **Validate that your PVCs have been resized.**

Notes:

- Volume expansion can take some time. If validation is not successful the first time consider waiting a few minutes and validating again.
- Volume expansion only completes without user action when an online resize is performed.
- Some storage providers round up the storage size you have requested. The expanded volume should have the same or greater size than your request.

a) Check your queue manager for status conditions. Refer to the following table for conditions, explanations, and suggested actions.

Table 1. Status conditions for storage

CONDITION	MESSAGE	EXPLANATION
`StorageMismatch`	`Storage sizes defined in the QueueManager resource do not match the capacity of one or more provisioned PVCs [pvc-list]. AllowVolumeExpansion is set to false in the QueueManager resource so the MQ Operator will not attempt to reconcile these differences.`	Volume expansion does not occur because **`.allowVolumeExpansion`** has not been set to `true` in the queue manager definition.
`StorageExpansionPending`	`Volume expansion is pending for the following PVCs [pvc-list]`	Volume expansion is still taking place. If this status condition persists for an extended period of time then follow the steps below to gather more information because an offline resize, or failure to resize, might be taking place.
`Failed`	There are many possible storage related messages which can created a `'Failed'` status condition. For example: `'MQ Queue Manager failed to deploy: persistentvolumeclaims "<pvc>" is forbidden: only dynamically provisioned pvc can be resized and the storageclass the provisions the pvc must support resize.'`	If the queue manager has `'Failed'` status conditions with text that refers to storage, refer to the message within the status condition. The example message given here is caused by using a storage class that does not support expansion.

b) For each PVC that you have expanded, check that the capacity has increased to match or be greater than the value specified in the queue manager definition.

HA queue managers might have multiple PVCs of each type. To get the capacity of a PVC, run the following command:

- For deployments on the Red Hat OpenShift Container Platform:

```
oc get pvc <PVC_NAME> -o template --template '{{.status.capacity.storage}}'
```

- For deployments on Amazon EKS:

```
kubectl get pvc <PVC_NAME> -o template --template '{{.status.capacity.storage}}'
```

c) Check that the PVC does not have any status conditions or events that suggest a failed resize:

- For deployments on the Red Hat OpenShift Container Platform:

```
oc describe pvc <PVC_NAME>
```

- For deployments on Amazon EKS:

```
kubectl describe pvc <PVC_NAME>
```

- Your PVC might have the status condition `FileSystemResizePending` with message 'Waiting for user to (re-)start a pod to finish file system resize of volume on node'. This status condition is raised for both online and offline resizes. For an online resize, this status condition disappears without user action after the online resize completes.
- If your PVC has an event or status condition that indicates a failed resize, see Recovering from failure when expanding volumes in the Red Hat OpenShift documentation.

d) Check that the queue manager pods do not have any status conditions or events that suggest a failed resize. For HA deployments, check each replica.

- For deployments on the Red Hat OpenShift Container Platform:

```
oc describe pod <QUEUE_MANAGER_POD>
```

- For deployments on Amazon EKS:

```
kubectl describe pod <QUEUE_MANAGER_POD>
```

If your pod has an event or status condition that indicates a failed resize, you can attempt to recover from that failure.

- For Red Hat OpenShift users, see Recovering from failure when expanding volumes in the Red Hat OpenShift documentation.
- For Amazon EKS users, see Recovery From Volume Expansion Failure in the Kubernetes blog.

The error text might help you resolve the problem, or prevent the same problem occurring if you try to resize again after recovery.

4. **Restart pods when resizing offline**

   If your storage provider uses an offline resizing procedure when expanding volumes, then for volume expansion to complete you need to restart the queue manager pods that mount the volumes being resized.

   For multi-instance queue managers the recovery logs and persisted data volumes are shared between both the active and standby pods. For resizing of these volumes to complete, bring down both pods at the same time.

   Refer to your storage provider documentation for their offline resizing procedure.

Kubernetes Self-deploying and configuring queue managers on Kubernetes

How to self-deploy queue managers on Kubernetes using native resources.

Before you begin

Note: The IBM MQ Operator is available on Amazon EKS from version 3.9.0, and can be used to deploy and configure queue managers. For more information see “Deploying and configuring queue managers using the IBM MQ Operator” on page 97.

About this task

Kubernetes Example: Self-deploying a simple queue manager in Kubernetes

This example deploys a simple "quick start" queue manager that uses ephemeral (non-persistent) storage and turns off IBM MQ security. Messages are not persisted across restarts of the queue manager. You can adjust the configuration to change many queue manager settings.

Before you begin

Note: The IBM MQ Operator is available on Amazon EKS and can be used to deploy and configure queue managers. For more information see "Deploying a simple queue manager using the IBM MQ Operator on Amazon EKS" on page 100.

To complete this example, you must first have completed the following prerequisites:

- Create a Kubernetes namespace for this example.
- On the command line, log into the Kubernetes cluster, and switch to the above namespace.
- Ensure that you have followed the steps in "Preparing to use Kubernetes by creating a pull secret" on page 86 in the above namespace.

About this task

This example provides `Service`, `ServiceAccount`, and `StatefulSet` YAML to define a queue manager to be deployed into Kubernetes.

Procedure

1. Create a `Service` to provide a consistent IP address for IBM MQ traffic.

   Ensure you are in the namespace you created earlier (see Before you begin), then apply the following YAML using the Kubernetes command line or console.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: quickstart
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
  name: quickstart-ibm-mq
spec:
  ports:
  - name: qmgr
    port: 1414
    protocol: TCP
    targetPort: 1414
  - name: metrics
    port: 9157
    protocol: TCP
    targetPort: 9157
  selector:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: quickstart
    app.kubernetes.io/name: ibm-mq
  type: ClusterIP
```

2. Create a `ServiceAccount` to use for the queue manager.

   The example `ServiceAccount` below refers to an `imagePullSecret` that is used for an entitlement check when the image is pulled. See "Preparing to use Kubernetes by creating a pull secret" on page 86 for more information.

```
kind: ServiceAccount
apiVersion: v1
metadata:
  name: quickstart-ibm-mq
  labels:
    app.kubernetes.io/component: integration
```

```
    app.kubernetes.io/instance: quickstart
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
imagePullSecrets:
  - name: ibm-entitlement-key
```

3. Create the queue manager Pod, managed by a `StatefulSet`.

   In the following YAML, the `namespace` name is assumed to be `mq-test`. Change it to match the name that you are using in your Kubernetes environment for this example.

   This YAML creates a single instance queue manager in your Kubernetes environment in the namespace specified. No persistence volume is created for this instance because our example is for ephemeral storage.

   Note: This example uses IBM MQ Advanced annotations. To use an IBM MQ license, edit the annotations under `spec.template.metadata` to the values that are specified under IBM MQ or IBM MQ for Non-Production Environment in "IBM MQ container license annotations" on page 314.

```
kind: StatefulSet
apiVersion: apps/v1
metadata:
  name: quickstart-ibm-mq
  namespace: mq-test
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: quickstart
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
spec:
  persistentVolumeClaimRetentionPolicy:
    whenDeleted: Retain
    whenScaled: Retain
  replicas: 1
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app.kubernetes.io/component: integration
      app.kubernetes.io/instance: quickstart
      app.kubernetes.io/name: ibm-mq
  template:
    metadata:
      labels:
        app.kubernetes.io/component: integration
        app.kubernetes.io/instance: quickstart
        app.kubernetes.io/name: ibm-mq
        app.kubernetes.io/version: 10.0.0.0
        statefulSetName: quickstart-ibm-mq
      annotations:
# These annotations are critical for the IBM License Service to charge you correctly
        productChargedContainers: qmgr
        productID: 208423bb063c43288328b1d788745b0c
        productMetric: PROCESSOR_VALUE_UNIT
        productName: IBM MQ Advanced
        productVersion: 10.0.0.0
    spec:
      restartPolicy: Always
      serviceAccountName: quickstart-ibm-mq
      terminationGracePeriodSeconds: 30
      securityContext: {}
      containers:
        - resources:
            limits:
              cpu: '1'
              memory: 1Gi
            requests:
              cpu: '1'
              memory: 1Gi
          readinessProbe:
            exec:
              command:
                - chkmqready
            initialDelaySeconds: 10
            timeoutSeconds: 3
            periodSeconds: 5
            successThreshold: 1
            failureThreshold: 1
          terminationMessagePath: /run/termination-log
```

```
          name: qmgr
          livenessProbe:
            exec:
              command:
                - chkmqhealthy
            initialDelaySeconds: 90
            timeoutSeconds: 5
            periodSeconds: 10
            successThreshold: 1
            failureThreshold: 1
          env:
            - name: LICENSE
              # if you accept the license, change the following value to "accept"
              value: view
            - name: MQ_QMGR_NAME
              value: QUICKSTART
            - name: MQ_MULTI_INSTANCE
              value: 'false'
            - name: MQ_ENABLE_METRICS
              value: 'true'
            - name: MQ_ENABLE_EMBEDDED_WEB_SERVER
              value: 'false'
            - name: MQ_LOGGING_CONSOLE_FORMAT
              value: basic
            - name: DEBUG
              value: 'false'
            - name: MQ_ENABLE_TRACE_STRMQM
              value: 'false'
            - name: MQ_EPHEMERAL_PREFIX
              value: /run/mqm
            - name: MQ_GRACE_PERIOD
              value: '29'
            - name: MQ_NATIVE_HA
              value: 'false'
          securityContext:
            allowPrivilegeEscalation: false
            capabilities:
              drop:
                - ALL
            privileged: false
            runAsNonRoot: true
            readOnlyRootFilesystem: false
          ports:
            - containerPort: 1414
              protocol: TCP
            - containerPort: 9157
              protocol: TCP
          terminationMessagePolicy: File
# The image to pull.  This example uses a prebuilt MQ Advanced image from IBM Container
Registry
          image: cp.icr.io/cp/ibm-mqadvanced-server:9.4.1.0-r1
          imagePullPolicy: IfNotPresent
      serviceAccount: quickstart-ibm-mq
      dnsPolicy: ClusterFirst
  serviceName: qm
  podManagementPolicy: OrderedReady
  updateStrategy:
    type: OnDelete
```

4. Confirm that the queue manager is running

   The queue manager is now being deployed. Confirm it is in Running state, for example by listing the queue manager pods:

   ```
   kubectl get pods --selector app.kubernetes.io/instance=quickstart
   ```

   Note that this must return a single pod, and the READY field must show the value 1/1.

5. View the queue manager status

   Run the following command, specifying the full name of the queue manager pod:

   ```
   kubectl exec -t <qmgr_pod_name> -- dspmq -m QUICKSTART
   ```

   This should return a response similar to the following:

   ```
   QMNAME(QUICKSTART) STATUS(Running)
   ```

Results

You have successfully deployed a simple IBM MQ queue manager, of type single instance, on your Kubernetes platform.

##

Example: Configuring Native HA in Kubernetes for self-deployed queue managers

This example deploys a queue manager using the native high availability feature into Kubernetes by using YAML resources directly. Mutual TLS is used for authentication, to map from a TLS certificate to an identity in the queue manager.

Before you begin

Note: The IBM MQ Operator is available on Amazon EKS, and can be used to deploy and configure native HA queue managers. For more information see "Configuring high availability for queue managers using the IBM MQ Operator" on page 118.

Note: This example uses an IBM MQ Advanced license. You can also configure IBM MQ licensed queue managers to use Native HA and Cross-Region Replication features by adding additional licenses to a queue manager. For more information, see "Configuring Native HA and Cross-Region Replication on IBM MQ licensed self-deployed queue managers in Kubernetes" on page 213.

To complete this example, you must first have completed the following prerequisites:

- Create a Kubernetes namespace for this example.
- On the command line, log into the Kubernetes cluster, and switch to the above namespace.
- Ensure that you have followed the steps in "Preparing to use Kubernetes by creating a pull secret" on page 86 in the above namespace.

About this task

This example provides `Service`, `ServiceAccount`, and `StatefulSet` YAML to define a queue manager to be deployed into Kubernetes. It also details the additional steps required to deploy the queue manager with TLS enabled.

Procedure

1. Create a pair of certificates as described in "Creating a self-signed PKI using OpenSSL" on page 107.
2. Create a config map containing MQSC commands and an INI file.

   Create a Kubernetes `ConfigMap` containing the MQSC commands to create a new queue and a SVRCONN channel, and to add a channel authentication record that allows access to the channel.

   Ensure you are in the namespace you created earlier (see Before you begin), then apply the following YAML using the Kubernetes command line.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: example-nativeha-configmap
data:
  example.mqsc: |
    DEFINE CHANNEL('MTLS.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS13_OR_HIGHER') REPLACE
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
ACTION(REPLACE)
    SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=example-app1') USERSRC(MAP)
MCAUSER('app1') ACTION(REPLACE)
    SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)
    DEFINE QLOCAL('EXAMPLE.QUEUE') REPLACE
    SET AUTHREC PROFILE('EXAMPLE.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(BROWSE,PUT,GET,INQ)
  example.ini: |
    Service:
```

```
        Name=AuthorizationService
        EntryPoints=14
        SecurityPolicy=UserExternal
    NativeHAInstance:
      Name=exampleqm-ibm-mq-0
      ReplicationAddress=exampleqm-ibm-mq-replica-0(9414)
    NativeHAInstance:
      Name=exampleqm-ibm-mq-1
      ReplicationAddress=exampleqm-ibm-mq-replica-1(9414)
    NativeHAInstance:
      Name=exampleqm-ibm-mq-2
      ReplicationAddress=exampleqm-ibm-mq-replica-2(9414)
```

The MQSC defines a channel called *MTLS.SVRCONN* and a queue called *EXAMPLE.QUEUE*. The channel is configured to allow access only to clients which present a certificate with a "common name" of *example-app1*. This is the common name used in one of the certificates created in Step “1” on page 204. Connections on this channel with this common name are mapped to a user ID of *app1*, which is authorized to connect to the queue manager, and to access the example queue. The INI file enables a security policy which means that the *app1* user ID does not need to exist in an external user registry — it exists only as a name in this configuration. It also configures the replication addresses for use with Native HA.

3. Create a `Service` to provide a consistent IP address for IBM MQ traffic.

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
  name: exampleqm-ibm-mq
spec:
  ports:
  - name: qmgr
    port: 1414
    protocol: TCP
    targetPort: 1414
  - name: metrics
    port: 9157
    protocol: TCP
    targetPort: 9157
  selector:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
  type: ClusterIP
```

4. Create an internal `Service` for each of the Native HA replicas to use to communicate with each other.

```
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
  name: exampleqm-ibm-mq-replica-0
spec:
  ports:
  - port: 9414
    protocol: TCP
    targetPort: 9414
  publishNotReadyAddresses: true
  selector:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    statefulset.kubernetes.io/pod-name: exampleqm-ibm-mq-0
  type: ClusterIP
---
apiVersion: v1
kind: Service
metadata:
```

```
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
  name: exampleqm-ibm-mq-replica-1
spec:
  ports:
  - port: 9414
    protocol: TCP
    targetPort: 9414
  publishNotReadyAddresses: true
  selector:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    statefulset.kubernetes.io/pod-name: exampleqm-ibm-mq-1
  type: ClusterIP
---
apiVersion: v1
kind: Service
metadata:
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
  name: exampleqm-ibm-mq-replica-2
spec:
  ports:
  - port: 9414
    protocol: TCP
    targetPort: 9414
  publishNotReadyAddresses: true
  selector:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    statefulset.kubernetes.io/pod-name: exampleqm-ibm-mq-2
  type: ClusterIP
```

5. Create a `ServiceAccount` to use for the queue manager.

   The example `ServiceAccount` below refers to a `imagePullSecret` that is used for an entitlement check when the image is pulled. See “Preparing to use Kubernetes by creating a pull secret” on page 86 for more information.

```
kind: ServiceAccount
apiVersion: v1
metadata:
  name: exampleqm-ibm-mq
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
imagePullSecrets:
  - name: ibm-entitlement-key
```

6. Create the queue manager Pod, managed by a `StatefulSet`.

   Use the following YAML as an example, making these changes:

   - Change `namespace` name from `mq-test` to match the name that you are using in your Kubernetes environment for this example.
   - If you are trying the example on a Kubernetes cluster with less than three available nodes, delete the `affinity` section.
   - Optionally, change the license. The example uses an IBM MQ Advanced license. From IBM MQ 9.4.2, you can also configure IBM MQ licensed queue managers to use Native HA and Cross-Region Replication features, by adding additional licenses to a queue manager.

     a. Change the annotations under `spec.template.metadata` to the values that are specified under IBM MQ or IBM MQ for Non-Production Environment in “IBM MQ container license annotations” on page 314.

b. Complete step “3” on page 214 in “Configuring Native HA and Cross-Region Replication on IBM MQ licensed self-deployed queue managers in Kubernetes” on page 213. This step adds an `IBMLicensingDefinition` CR that adds additional licensing annotations to your queue manager for the Native HA and Cross-Region Replication features

```
apiVersion: apps/v1
kind: StatefulSet
metadata:
  labels:
    app.kubernetes.io/component: integration
    app.kubernetes.io/instance: exampleqm
    app.kubernetes.io/name: ibm-mq
    app.kubernetes.io/version: 10.0.0.0
  name: exampleqm-ibm-mq
spec:
  persistentVolumeClaimRetentionPolicy:
    whenDeleted: Retain
    whenScaled: Retain
  updateStrategy:
    type: OnDelete
  podManagementPolicy: Parallel
  replicas: 3
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app.kubernetes.io/component: integration
      app.kubernetes.io/instance: exampleqm
      app.kubernetes.io/name: ibm-mq
  serviceName: qm
  template:
    metadata:
      # These annotations are critical for the IBM License Service to charge you correctly
      annotations:
        productChargedContainers: All
        productID: 208423bb063c43288328b1d788745b0c
        productMetric: PROCESSOR_VALUE_UNIT
        productName: IBM MQ Advanced
        productVersion: 10.0.0.0
      labels:
        app.kubernetes.io/component: integration
        app.kubernetes.io/instance: exampleqm
        app.kubernetes.io/name: ibm-mq
        app.kubernetes.io/version: 10.0.0.0
        statefulSetName: exampleqm-ibm-mq
    spec:
      # Create an anti-affinity rule, so the Native HA replicas don't end up on the same Node
      affinity:
        podAntiAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
          - labelSelector:
              matchExpressions:
              - key: statefulSetName
                operator: In
                values:
                - exampleqm-ibm-mq
            topologyKey: kubernetes.io/hostname
      containers:
      # Environment variables to supply to MQ container
      - env:
        - name: LICENSE
          # if you accept the license, change the following value to "accept"
          value: view
        - name: MQ_QMGR_NAME
          value: EXAMPLEQM
        - name: MQ_MULTI_INSTANCE
          value: "false"
        - name: MQ_ENABLE_METRICS
          value: "true"
        - name: MQ_ENABLE_EMBEDDED_WEB_SERVER
          value: "false"
        - name: MQ_LOGGING_CONSOLE_FORMAT
          value: basic
        - name: DEBUG
          value: "false"
        - name: MQ_ENABLE_TRACE_STRMQM
          value: "false"
        - name: MQ_EPHEMERAL_PREFIX
          value: /run/mqm
        - name: MQ_GRACE_PERIOD
          value: "29"
```

```
        - name: MQ_NATIVE_HA
          value: "true"
        # The image to pull.  This example uses a prebuilt MQ Advanced image from IBM
Container Registry
        image: cp.icr.io/cp/ibm-mqadvanced-server:10.0.0.0-r3
        imagePullPolicy: IfNotPresent
        livenessProbe:
          exec:
            command:
            - chkmqhealthy
          failureThreshold: 1
          periodSeconds: 10
          successThreshold: 1
          timeoutSeconds: 5
        name: qmgr
        ports:
        - containerPort: 1414
          protocol: TCP
        - containerPort: 9157
          protocol: TCP
        - containerPort: 9414
          protocol: TCP
        readinessProbe:
          exec:
            command:
            - chkmqready
          failureThreshold: 1
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 3
        # CPU and memory resources to allocate
        resources:
          limits:
            cpu: "1"
            memory: 1Gi
          requests:
            cpu: "1"
            memory: 1Gi
        # Container security context.  Disable all root capabilities and privileges.
        securityContext:
          allowPrivilegeEscalation: false
          capabilities:
            drop:
            - ALL
          privileged: false
          readOnlyRootFilesystem: true
          runAsNonRoot: true
        startupProbe:
          exec:
            command:
            - chkmqstarted
          failureThreshold: 24
          periodSeconds: 5
          successThreshold: 1
          timeoutSeconds: 5
        terminationMessagePath: /run/termination-log
        terminationMessagePolicy: File
        volumeMounts:
        # Mount the main queue manager persistent volume.  Sym-linked from /var/mqm
        - mountPath: /mnt/mqm
          name: data
        # Mount space for temporary files
        - mountPath: /run
          name: scratch
        - mountPath: /tmp
          name: tmp
        # Mount the "default" key for TLS
        - mountPath: /etc/mqm/pki/keys/default
          name: default
          readOnly: true
        # Mount the key to use for Native HA replication traffic
        - mountPath: /etc/mqm/ha/pki/keys/ha
          name: ha-tls
          readOnly: true
        # Mount MQSC to be applied when the queue manager starts
        - mountPath: /etc/mqm/example.mqsc
          name: cm-example-nativeha-configmap
          readOnly: true
          subPath: example.mqsc
        # Mount an INI file to be applied when the queue manager starts
        - mountPath: /etc/mqm/example.ini
          name: cm-example-nativeha-configmap
```

```
          readOnly: true
          subPath: example.ini
      restartPolicy: Always
      securityContext: {}
      serviceAccount: exampleqm-ibm-mq
      serviceAccountName: exampleqm-ibm-mq
      automountServiceAccountToken: false
      terminationGracePeriodSeconds: 30
      volumes:
      - emptyDir:
          sizeLimit: 100M
        name: scratch
      # Temporary directory is used by default to store runmqras output, so make it larger
      - emptyDir:
          sizeLimit: 2Gi
        name: tmp
      - name: default
        secret:
          defaultMode: 288
          items:
          - key: tls.key
            path: tls.key
          - key: tls.crt
            path: tls.crt
          - key: ca.crt
            path: ca.crt
          secretName: example-qm-tls
      - name: ha-tls
        secret:
          defaultMode: 288
          secretName: example-qm-tls
      - configMap:
          defaultMode: 420
          items:
          - key: example.mqsc
            path: example.mqsc
          - key: example.ini
            path: example.ini
          name: example-nativeha-configmap
        name: cm-example-nativeha-configmap
  volumeClaimTemplates:
  - apiVersion: v1
    kind: PersistentVolumeClaim
    metadata:
      labels:
        app.kubernetes.io/component: integration
        app.kubernetes.io/instance: exampleqm
        app.kubernetes.io/name: ibm-mq
      name: data
      namespace: mq-test
    spec:
      accessModes:
      - ReadWriteOnce
      resources:
        requests:
          storage: 2Gi
      volumeMode: Filesystem
```

Note that the `Secret` *example-qm-tls* was created in Step "1" on page 204, and the `ConfigMap` *example-nativeha-configmap* was created in Step "2" on page 204.

A persistent volume is created for each instance. The default storage class configured in your Kubernetes cluster is used. If you do not have a storage class configured as default, or you want to use a different storage class, add `storageClassName: storage_class_name` under `spec.volumeClaimTemplates[0].spec`.

The three pods in a Native HA queue manager replicate data over the network. This link is not encrypted by default, but this example uses the queue manager's certificate for encrypting traffic. You can specify a different certificate for additional security. The Native HA TLS Secret must be a Kubernetes TLS Secret, which has a particular structure (for example, the private key must be called *tls.key*).

Note that the `updateStrategy` is set to `OnDelete`, which impacts how updates are rolled-out. For more information, see "Considerations for performing your own rolling update of a Native HA queue manager" on page 24

7. Confirm that the queue manager is running.

   The queue manager is now being deployed. Confirm it is in `Running` state before proceeding, for example by listing the queue manager pods:

   ```
   kubectl get pods --selector app.kubernetes.io/instance=exampleqm
   ```

   Ideally you should connect to the queue manager to confirm that it is running. However connecting to the queue manager from outside the Kubernetes cluster is outside of the scope of these instructions, because it can vary significantly between Kubernetes distributions and cloud environments.
8. Force the active pod to fail

   To validate the automatic recovery of the queue manager, simulate a pod failure:

   a) View the active and standby pods

   Rerun the previous command:

   ```
   kubectl get pods --selector app.kubernetes.io/instance=exampleqm
   ```

   Note that, in the **READY** field, the active pod returns the value `1/1`, whereas the replica pods return the value `0/1`.

   b) Delete the active pod

   Run the following command, specifying the full name of the active pod:

   ```
   kubectl delete pod exampleqm-ibm-mq-<active_pod_value>
   ```

   c) View the pod status again

   Run the following command:

   ```
   kubectl get pods --selector app.kubernetes.io/instance=exampleqm
   ```

   d) View the queue manager status

   Run the following command, specifying the full name of one of the other pods:

   ```
   kubectl exec -t <qmgr_pod_name> -- dspmq -o nativeha -x -m EXAMPLEQM
   ```

   You should see the status shows that the active instance has changed, for example:

   ```
   QMNAME(EXAMPLEQM) ROLE(Active) INSTANCE(inst1) INSYNC(Yes) QUORUM(3/3)
   INSTANCE(inst1) ROLE(Active) REPLADDR(9.20.123.45) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
   CONNINST(Yes) ALTDATE(2022-01-12) ALTTIME(12.03.44)
   INSTANCE(inst2) ROLE(Replica) REPLADDR(9.20.123.46) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
   CONNINST(Yes) ALTDATE(2022-01-12) ALTTIME(12.03.44)
   INSTANCE(inst3) ROLE(Replica) REPLADDR(9.20.123.47) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
   CONNINST(Yes) ALTDATE(2022-01-12) ALTTIME(12.03.44)
   ```

Results

You have successfully deployed a queue manager with native high availability and mutual TLS authentication, and verified that it automatically recovers when the active Pod fails.

V 10.0.0 Example: Configuring Native HA Cross-Region Replication in Kubernetes for self-deployed queue managers

This example deploys a queue manager using the native high availability cross-region replication (Native HA CRR) feature into Kubernetes by using YAML resources directly.

Before you begin

Note: The IBM MQ Operator is available on Amazon EKS from version 3.9.0, and can be used to deploy and configure native HA queue managers. For more information see “Configuring Native HA CRR using the IBM MQ Operator” on page 123.

To complete this example, you must first have completed the following prerequisites:

- Create a Kubernetes namespace in each of the clusters where the Live and Recovery groups will be deployed.
- On the command line, be able to log into either Kubernetes cluster, and switch to the relevant namespace.
- Ensure that you have followed the steps in "Preparing to use Kubernetes by creating a pull secret" on page 86 in the above namespaces.
- Configure an appropriate environment specific Ingress/NLB (Network Load Balancer). This is required to enable communication between the two Native HA CRR groups, with TLS passthrough to the Service ClusterIP port that is defined in step "2" on page 212.

About this task

This example deploys two queue managers using the Native HA CRR feature into Kubernetes using YAML resources directly. One of the Native HA queue managers will be Live and the other will be the Recovery. This example builds upon the YAML files created in "Example: Configuring Native HA in Kubernetes for self-deployed queue managers" on page 204.

This example provides the modifications required on top of those defined for NativeHA: ConfigMap, Service and StatefulSet. You will additionally need to configure an appropriate environment specific Ingress/NLB. This is required to enable communication between the two Native HA CRR groups, with TLS passthrough to the Service ClusterIP port that is defined in Step "2" on page 212.

Procedure

1. Create a new config map, or define a new .ini data definition within the existing ConfigMap (see Step 2 in the Native HA example), to define the INI file for a Live or a Recovery queue manager.

   Create a Kubernetes ConfigMap containing the definitions to configure the following groups:

   a) A native high availability group as Recovery:

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: example-nativeha-configmap
   data:
   ...
     example-crr: |
       NativeHALocalInstance:
         GroupName=beta
         GroupRole=Recovery
         GroupLocalAddress=(9415)
         GroupCipherSpec=ANY_TLS12_OR_HIGHER
   ```

   When the Recovery group is deployed, determine the external address that routes traffic to the Recovery group Service ClusterIP port 9415. This address is determined by your environment-specific Ingress or LoadBalancer configuration (as described Before you begin). Use this address with the appropriate port (typically 443 for TLS passthrough) in the ReplicationAddress parameter mentioned in the snippet in step b.

   b) A Native HA CRR group as Live (pointing to the Recovery group):

   ```
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: example-nativeha-configmap
   data:
   ...
     example-crr: |
       NativeHALocalInstance:
         GroupName=alpha
         GroupRole=Live
         GroupLocalAddress=(9415)
         GroupCipherSpec=ANY_TLS12_OR_HIGHER
       NativeHARecoveryGroup:
   ```

```
      Enabled=Yes
      GroupName=beta
      ReplicationAddress=<address.to.recovery.cluster>(443)
```

2. Expose an additional Service ClusterIP port for the Native HA CRR traffic to flow into the CRR group. This can be done by creating a new YAML definition, or by extending the Service definition in step 3 of the Native HA example:

```
apiVersion: v1
kind: Service
metadata:
...
  name: exampleqm-ibm-mq
spec:
  ports:
...
  - name: ha-crr
    port: 9415
    protocol: TCP
    targetPort: 9415
...
  selector:
...
  type: ClusterIP
```

3. Define a new environment specific Ingress/NLB to enable communication from the other Native HA CRR group to reach this Native HA CRR group and enable TLS 'passthrough' to the new Service ClusterIP port defined in step 2.
4. Modify the StatefulSet created in step 6 of the Native HA example, to ensure that the following requirements are met:
   a) The new ConfigMap INI definition for Native HA CRR is mounted so that the queue manager can pick it up on start up:

```
...
        volumeMounts:
...
        # Mount the NativeHA CRR INI file to be applied when the queue manager starts
        - mountPath: /etc/mqm/example-crr.ini
          name: cm-example-nativeha-configmap
          readOnly: true
          subPath: example-crr.ini
...
      volumes:
...
      - configMap:
          defaultMode: 420
          items:
...
          - key: example-crr.ini
            path: example-crr.ini
          name: example-nativeha-configmap
        name: cm-example-nativeha-configmap
```

   b) Communication between the two Native HA CRR groups must be encrypted. Therefore, declarations of the TLS secret key for the current NativeHA CRR group, and the trust certificate for the other NativeHA CRR group, need to be defined and mounted in the `/etc/mqm/groupha` directory.

      Using the group names alpha (for Live) and beta (for Recovery), the example modifications for the YAML StatefulSet for the Live group would be:

```
...
        volumeMounts:
...
          - mountPath: /etc/mqm/groupha/pki/keys/ha-group
            name: groupha-tls
            readOnly: true
          - name: groupha-tls-trust-0
            mountPath: /etc/mqm/groupha/pki/trust/0
            readOnly: true
...
      volumes:
...
```

```
      - name: groupha-tls
        secret:
          secretName: nha-crr-secret-alpha
          defaultMode: 288
      - name: groupha-tls-trust-0
        secret:
          secretName: nha-crr-secret-beta
          defaultMode: 420
          items:
            - key: tls.crt
              path: native-crr-nha-crr-secret-beta-0-tls.crt
```

5. Confirm that the Native HA CRR groups are running, in-sync, connected, and in quorum, by using the following **dspmq** command on the Ready and Running pods of the Live and Recovery queue manager instance:

```
kubectl exec -t <qmgr_pod_name> -- dspmq -o nativeha -g -m EXAMPLEQM
```

The status should show that the two Native HA CRR groups are connected and acting as Live and Recovery groups, for example:

```
QMNAME(EXAMPLEQM) ROLE(Active) INSTANCE(testmqqm-ibm-mq-0) INSYNC(yes) QUORUM(3/3)
GRPLSN(<0:0:10:30665>) GRPNAME(alpha) GRPROLE(Live)
 GRPNAME(alpha) GRPROLE(Live) GRPADDR(Unknown) GRPVER(9.4.2.0) GRSTATUS(Normal)
RCOVLSN(<0:0:10:30665>) RCOVTIME(2025-02-17T16:25:16.527089Z) INITLSN(<0:0:9:4684>)
INITTIME(2025-02-17T16:25:01.226650Z) LIVETIME(2025-02-17T16:25:02.974677Z)
ALTDATE(2025-02-17) ALTTIME(16.25.19)
 GRPNAME(beta) GRPROLE(Recovery) GRPADDR(<address.to.recovery.cluster>) GRPVER(9.4.2.0)
CONNGRP(yes) GRSTATUS(Normal) RCOVLSN(<0:0:10:30665>) RCOVTIME(2025-02-17T16:25:16.527089Z)
BACKLOG(0) INSYNC(yes) SYNCTIME(2025-02-17T16:29:38.643916Z) ALTDATE(2025-02-17)
ALTTIME(16.29.29)
```

Results

You have successfully deployed two queue manager groups with native high availability and TLS authentication within and between the groups. You have also verified that the Live and Recovery groups are connected and performing cross-region replication.

Kubernetes Configuring Native HA and Cross-Region Replication on IBM MQ licensed self-deployed queue managers in Kubernetes

From IBM MQ 9.4.2, IBM MQ licensed queue managers can be configured to use Native HA Cross-Region Replication (CRR) features by adding an additional license to a queue manager.

Before you begin

Note: These instructions apply only if you are deploying queue managers on Kubernetes. If you deployed the IBM MQ Operator on Red Hat OpenShift, or on Amazon EKS from IBM MQ Operator 3.9.0, see “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124 instead.

- The IBM License Service is required to use Native HA CRR features on IBM MQ licensed queue managers. For more information, see Installing License Service on the Kubernetes cluster in the IBM Cloud Pak foundational services documentation.
- On the `StatefulSet` that manages your queue manager pod, the `metadata.labels.app.kubernetes.io/instance` label must be set to a unique value.

Procedure

1. On the command line, log into the Kubernetes cluster then switch to the namespace in which your queue manager is deployed.
2. Add the relevant annotations to your queue manager's annotations.

Note that the instructions in this step are for editing an existing queue manager. For new queue managers, add the annotations on creation.

Important: Updating the queue manager annotations requires a pod restart.

a) Identify the `StatefulSet` of the queue manager that you want to update:

```
kubectl get statefulset
```

b) Open the `StatefulSet` of the queue manager for editing:

```
kubectl edit statefulset <QUEUE-MANAGER-STATEFUL-SET>
```

c) Add the annotation that is relevant to your chargeable component to your `StatefulSet`. Note that there are other annotations under `spec.template.metadata`; change only the following annotations:

- For IBM MQ:

```
spec:
  template:
    metadata:
      productID: c661609261d5471fb4ff8970a36bccea
      productName: IBM MQ
```

- For IBM MQ for Non-Production Environment:

```
spec:
  template:
    metadata:
      productID: 151bec68564a4a47a14e6fa99266deff
      productName: IBM MQ for Non-Production Environment
```

d) Save your changes. A message similar to the following is shown on the command line:

```
statefulset.apps/<QUEUE-MANAGER-STATEFUL-SET> edited
```

e) Delete all queue manager pods that are managed by the `StatefulSet` that you edited.

f) Repeat these steps for other queue managers as required.

3. Add the relevant `IBMLicensingDefinition` CR that will add additional licensing annotations to your queue manager for the Native HA CRR features.

a) Create an `IBMLicensingDefinition` YAML file.

In the following examples, change these values:

- <IBMLicensingDefinition-UNIQUE-NAME>: A unique name for the `IBMLicensingDefinition` CR. A good practice is to include your queue manager name in this name.
- <QUEUE-MANAGER-NAMESPACE>: The namespace in which your queue manager is deployed.
- <QUEUE-MANAGER-NAME>: The value of the `metadata.labels.app.kubernetes.io/instance` of your queue manager.
- Example YAML for an `IBMLicensingDefinition` CR for production use:

```
apiVersion: operator.ibm.com/v1
kind: IBMLicensingDefinition
metadata:
  name: <IBMLicensingDefinition-UNIQUE-NAME>
  namespace: <QUEUE-MANAGER-NAMESPACE>
spec:
  action: cloneModify
  condition:
    metadata:
      labels:
        app.kubernetes.io/instance: <QUEUE-MANAGER-NAME>
  scope: cluster
  set:
```

```
    productID: 46a57a4a66ee4b904e9b5544b53a2ba2
    productName: IBM MQ Native HA and Cross-Region Replication Add-on
```

- Example YAML for an `IBMLicensingDefinition` CR for non-production use:

```
apiVersion: operator.ibm.com/v1
kind: IBMLicensingDefinition
metadata:
  name: <IBMLicensingDefinition-UNIQUE-NAME>
  namespace: <QUEUE-MANAGER-NAMESPACE>
spec:
  action: cloneModify
  condition:
    metadata:
      labels:
        app.kubernetes.io/instance: <QUEUE-MANAGER-NAME>
  scope: cluster
  set:
    productID: 525b1331a74d4a209d9f6e9c908e374e
    productName: IBM MQ Native HA and Cross-Region Replication Add-on for Non-
Production Env
```

b) Apply the `IBMLicensingDefinition` YAML file:

```
kubectl apply -f <IBMLicensingDefinition-UNIQUE-NAME>.yaml
```

c) Repeat these steps for other queue managers as required.

Kubernetes Self-deploying and configuring queue managers using the sample Helm chart

You can deploy and configure a queue manager on Kubernetes using the sample Helm chart.

About this task

The IBM MQ Operator is supported on the following platforms:

- Red Hat OpenShift Container Platform
- Amazon EKS (from IBM MQ Operator 3.9.0)

If you are not using one of these platforms, you can use the sample Helm chart to deploy onto other types of Kubernetes clusters.

Procedure

- For information on how to use Helm to deploy your own IBM MQ container image, see Sample IBM MQ Helm chart.

Related reference

“Support for IBM MQ in containers” on page 10
Not all IBM MQ features are available and supported in the same way in containers.

Podman Self-deploying queue managers on Podman

This example runs a prebuilt IBM MQ Advanced Container image, to deploy a queue manager into Podman. The same example can also be used with Docker.

Before you begin

To validate the TLS configuration at the end of this example:

- Install the IBM MQ client. You need the **amqsputc** and **amqsgetc** commands to test putting and getting messages. These commands can be installed as part of the IBM MQ client.
  - For Windows and Linux, install the IBM MQ redistributable client for your operating system from https://ibm.biz/mq94redistclients.

  - For Mac, download and set up the IBM MQ MacOS Toolkit from https://developer.ibm.com/tutorials/mq-macos-dev/.
- Note there is no prebuilt ARM64 IBM MQ Advanced Container image.

This example uses Podman, but can be used with Docker instead.

About this task

This example runs a prebuilt IBM MQ Advanced Container image, to deploy a queue manager into Podman. It details the additional steps required to configure TLS for messaging, as well as configuring the IBM MQ Console and the REST API. You can choose which of these to configure, by skipping steps that are not required to meet your needs.

When the queue manager is running, this example uses the IBM MQ sample programs running on a machine outside Podman, such as your laptop, to validate using mutual TLS between the sample client and the queue manager to put and get messages. It also validates logging into the IBM MQ Console, and making administrative and messaging REST API calls.

Note: You can instead run a prebuilt IBM MQ Advanced for Developers container image, which has some additional default developer configuration. See "IBM MQ Advanced for Developers container image" on page 326.

Procedure

- **Configure TLS for messaging between a client and the queue manager**

  a) Create certificates as described in "Creating a self-signed PKI using OpenSSL" on page 107, following all steps but skipping the **kubectl** command.

  a. Create a directory, for example called `qm-pki` containing the key and certificate files generated for the queue manager:

     i) `example-qm.key`

     ii) `example-qm.crt`

     iii) `ca.crt`

     The user running the container (uid 1001) needs read access to `example-qm.key`. By default, OpenSSL creates it such that the user that creates the key has access to it.

  b. Create a directory, for example called `mq-client`, containing files generated for the client:

     i) `example-app1.p12`

     ii) On Mac, you also need `example-app1-chain.crt`

  b) Create a file containing the MQSC commands, for example **example-tls.mqsc**, to create a new queue and a SVRCONN channel, and to add a channel authentication record that allows access to the channel.

```
DEFINE CHANNEL('MTLS.SVRCONN') CHLTYPE(SVRCONN) SSLCAUTH(REQUIRED)
SSLCIPH('ANY_TLS13_OR_HIGHER') REPLACE
SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=*') USERSRC(NOACCESS)
ACTION(REPLACE)
SET CHLAUTH('MTLS.SVRCONN') TYPE(SSLPEERMAP) SSLPEER('CN=example-app1') USERSRC(MAP)
MCAUSER('app1') ACTION(REPLACE)
SET AUTHREC PRINCIPAL('app1') OBJTYPE(QMGR) AUTHADD(CONNECT,INQ)
DEFINE QLOCAL('EXAMPLE.QUEUE') REPLACE
SET AUTHREC PROFILE('EXAMPLE.QUEUE') PRINCIPAL('app1') OBJTYPE(QUEUE)
AUTHADD(BROWSE,PUT,GET,INQ)
```

  The MQSC command defines a channel called `MTLS.SVRCONN` and a queue called `EXAMPLE.QUEUE`. The channel is configured to allow access only to clients that present a certificate with a "common name" of `example-app1`. This is the common name used in one of the certificates created in the previous step. Connections on this channel with this common name are mapped to a user ID of `app1`, which is authorized to connect to the queue manager and to access the example queue.

c) Create an INI file, for example `example-tls.ini`, that enables a security policy such that the app1 user ID does not need to exist in an external user registry - it exists only as a name in this configuration.

```
Service:
    Name=AuthorizationService
    EntryPoints=14
    SecurityPolicy=UserExternal
```

- **Configure the IBM MQ Console and REST API**

  In this example, you configure a basic registry within the `mqwebuser.xml` file. The user names, passwords, and roles in the XML file are used to authenticate and authorize users of the IBM MQ Console and REST API. For more information, see Configuring a basic registry for the IBM MQ Console and REST API.

  You should consider whether using a basic registry is appropriate for your production environments. See IBM MQ Console and REST API security.

  a) Create a file called `mqwebuser.xml`, to configure the IBM MQ Console.

  In this example a basic registry is created, containing users with the roles web admin and web user.

  You **must** replace *ADMIN_PASSWORD* and *APP_PASSWORD* with your own values. You can change the admin and app usernames too if you wish, though note the app username should match the app username specified in the previous MQSC command. Note that the user names are each used twice in the example.

```
<?xml version="1.0" encoding="UTF-8"?>
<server>
    <featureManager>
    <feature>appSecurity-2.0</feature>
    <feature>basicAuthenticationMQ-1.0</feature>
    </featureManager>
    <enterpriseApplication id="com.ibm.mq.console">
        <application-bnd>
            <security-role name="MQWebAdmin">
                <group name="MQWebAdminGroup" realm="defaultRealm"/>
            </security-role>
        </application-bnd>
    </enterpriseApplication>
    <enterpriseApplication id="com.ibm.mq.rest">
        <application-bnd>
            <security-role name="MQWebAdmin">
                <group name="MQWebAdminGroup" realm="defaultRealm"/>
            </security-role>
            <security-role name="MQWebUser">
                <group name="MQWebMessaging" realm="defaultRealm"/>
            </security-role>
        </application-bnd>
    </enterpriseApplication>
    <basicRegistry id="basic" realm="defaultRealm">
        <user name="admin" password="ADMIN_PASSWORD"/>
        <user name="app1" password="APP_PASSWORD"/>
        <group name="MQWebAdminGroup">
            <member name="admin"/>
        </group>
        <group name="MQWebMessaging">
            <member name="app1"/>
        </group>
    </basicRegistry>
    <sslDefault sslRef="mqDefaultSSLConfig"/>
</server>
```

  b) Log into the container registry.

  Get an entitlement key, then use it to log into the container registry. See “Preparing to use Podman or Docker by pulling the prebuilt container image” on page 87.

  c) Run the queue manager in Podman.

  Complete one of the following options:

– **Option 1: Run the queue manager in Podman by mounting the directory containing the queue manager certificates.**

  The following command includes the configuration for TLS messaging as well as configuring the IBM MQ Console and REST API. If you skipped any of the previously described configuration, because it wasn't required for your needs, remove the parameters relevant to the sections you skipped.

  Ensure the command points to the correct location for the files and directory created earlier.

  To run the queue manager in Podman, run the following command:

```
podman run --env MQ_ENABLE_EMBEDDED_WEB_SERVER=true  \
-v "./mqwebuser.xml:/etc/mqm/web/installations/Installation1/servers/mqweb/
mqwebuser.xml" \
-v "./example-tls.ini:/etc/mqm/example-tls.ini"  \
-v "./example-tls.mqsc:/etc/mqm/example-tls.mqsc"  \
-v "./qm-pki:/etc/mqm/pki/keys/qmlabel" --env LICENSE=accept --env MQ_QMGR_NAME=QM1 \
--publish 1414:1414 --publish 9443:9443 --detach --name QM1 cp.icr.io/cp/ibm-
mqadvanced-server:9.4.1.0-r1
```

  The command pulls the IBM MQ Advanced queue manager image, and starts the container.

– **Option 2: Run the queue manager in Podman using Podman secrets**

  You can mount the queue manager certificates using Podman secrets, rather than directly mounting the directories containing them into the container.

  Note: This option does not apply when using Docker, unless you are using Docker Swarm, because Docker secrets can only be used with Docker Swarm. See the docker secret create command in dockerdocs.

  Complete the following steps:

  a. Create Podman secrets for each of the queue manager TLS files. For example, from within the create `qm-pki` directory:

```
podman secret create example-qm.key ./example-qm.key
podman secret create example-qm.crt ./example-qm.crt
podman secret create ca.crt ./ca.crt
```

  The following command includes the configuration for TLS messaging as well as configuring the IBM MQ Console and REST API. If you skipped any of the previously described configuration, because it wasn't required for your needs, remove the parameters relevant to the sections you skipped.

  Ensure the command points to the correct location for the files and directory created earlier.

  To run the queue manager in Podman, run the following command:

```
podman run --env MQ_ENABLE_EMBEDDED_WEB_SERVER=true \
-v "./mqwebuser.xml:/etc/mqm/web/installations/Installation1/servers/mqweb/
mqwebuser.xml" \
-v "./example-tls.ini:/etc/mqm/example-tls.ini" \
-v "./example-tls.mqsc:/etc/mqm/example-tls.mqsc" \
--secret example-qm.key,target=/etc/mqm/pki/keys/qmlabel/example-
qm.key,uid=1001,mode=0440 \
--secret example-qm.crt,target=/etc/mqm/pki/keys/qmlabel/example-qm.crt,uid=1001 \
--secret ca.crt,target=/etc/mqm/pki/keys/qmlabel/ca.crt,uid=1001 --env
LICENSE=accept --env MQ_QMGR_NAME=QM1 \
--publish 1414:1414 --publish 9443:9443 --detach --name QM1 cp.icr.io/cp/ibm-
mqadvanced-server:9.4.1.0-r1
```

  Note: If the user that the container is running as is overridden to a different user, change the UID to match.

A queue manager is running in Podman.

```
podman ps
```

Validation

- **Validate messaging with TLS**

  a) Create a client INI file, called `mqclient.ini`, in the client directory created above, for example `mq-client`. Specify the *PASSWORD* used for the p12 keystore in step 1 of Configuring TLS for messaging between a client and the queue manager:

```
Channels:
  ChannelDefinitionDirectory=.
  ChannelDefinitionFile=CCDT.json
SSL:
  OutboundSNI=HOSTNAME
  SSLKeyRepository=example-app1.p12
  SSLKeyRepositoryPassword=PASSWORD
```

  b) In the same directory, create a CCDT file with a name matching the reference in the previously-defined `mqclient.ini` file, for example `CCDT.json`:

```
{
    "channel":
    [
        {
            "name": "MTLS.SVRCONN",
            "clientConnection":
            {
                "connection":
                [
                    {
                        "host": "localhost",
                        "port": 1414
                    }
                ],
                "queueManager": "QM1"
            },
            "transmissionSecurity":
            {
              "cipherSpecification": "ANY_TLS13",
              "certificateLabel": "example-app1"
            },
            "type": "clientConnection"
        }
   ]
}
```

  c) On Mac, run the following command:

```
export MQSSLTRUSTSTORE=example-app1-chain.crt
```

  d) From within the same directory, for example `mq-client`, use the sample application to put a message:

```
amqsputc EXAMPLE.QUEUE QM1
```

  e) Use the sample application to get a message:

```
amqsgetc EXAMPLE.QUEUE QM1
```

- **Validate the IBM MQ Console**

  Connect to the console in a browser:

  a) Open: https://localhost:9443/ibmmq/console

  b) Accept the self-signed certificate.

  c) Log in:

  – User: `admin`

  – Password: The value you previously specified for *ADMIN_PASSWORD*.

- **Validate the REST API**

  In the following examples, specify the values you previously configured for *ADMIN_PASSWORD* and *APP_PASSWORD*.

An example of calling the administrative REST API to get queue manager status:

```
curl -k -X GET -u admin:ADMIN_PASSWORD https://localhost:9443/ibmmq/rest/v1/admin/qmgr/QM1
```

An example of calling the messaging REST API to put and then get a message from a queue.

Put a message:

```
curl -k -X POST -u app1:APP_PASSWORD \
-H 'Content-Type: application/json' -H "ibm-mq-rest-csrf-token: value" \
-d 'msg from REST' https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/queue/
EXAMPLE.QUEUE/message
```

Browse a message:

```
curl -k -X GET -u app1:APP_PASSWORD \
-H "ibm-mq-rest-csrf-token: value" https://localhost:9443/ibmmq/rest/v2/messaging/qmgr/QM1/
queue/EXAMPLE.QUEUE/message
```

Note: The REST API **GET** operation is actually an IBM MQ browse operation - the message is left on the queue. To do an IBM MQ **GET**, use the REST API **DELETE** verb.

Troubleshooting

- If the **amqputc** or **amqgetc** sample applications fail, view error logs at `~/IBM/MQ/data/errors/AMQERR01.LOG` on the client machine, for example your laptop.
- View queue manager error logs in their usual places, by exec'ing onto the container in Podman.

  For example:

  ```
  podman exec -it QM1 /bin/bash
  ```

OpenShift CP4I-SC2 CD Migrating to the IBM MQ Operator on Red Hat OpenShift

This set of topics describes the key steps to migrate an existing IBM MQ queue manager into a container environment using the IBM MQ Operator in Red Hat OpenShift Container Platform.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

About this task

Clients who deploy IBM MQ on Red Hat OpenShift can be separated into the following scenarios:

1. Creating a new IBM MQ deployment in Red Hat OpenShift for new applications.
2. Extending an IBM MQ network into Red Hat OpenShift for new applications in Red Hat OpenShift.
3. Moving an IBM MQ deployment into Red Hat OpenShift to continue to support existing applications.

It is only for scenario 3 that you need to migrate your IBM MQ configuration. The other scenarios are considered new deployments.

This set of topics focuses on scenario 3, and describes the key steps to migrate an existing IBM MQ queue manager into a container environment using the IBM MQ Operator. Because of the flexibility and extensive use of IBM MQ, there are several optional steps. Each of these includes a "Do I need to do this" section. Verifying your need should save you time during your migration.

You also need to consider what data to migrate:

1. Migrate IBM MQ with the same configuration but without any existing queued messages.
2. Migrate IBM MQ with the same configuration and existing messages.

A typical version to version migration can use either approach. In a typical IBM MQ queue manager at the point of migration there are few if any messages stored on queues, which makes option 1 appropriate for many cases. In the case of migration to a container platform it is even more common to use option 1, to reduce the complexity of the migration and allow a blue green deployment. Therefore, the instructions focus on this scenario.

The objective of this scenario is to create a queue manager in the container environment that matches the definition of the existing queue manager. This allows existing network attached applications to simply be reconfigured to point to the new queue manager, without changing any other configuration or application logic.

Throughout this migration you generate multiple configuration files to be applied to the new queue manager. To simplify the management of these files, you should create a directory and generate them into that directory.

Procedure

1. “Checking that required functions are available” on page 221
2. “ Extracting the queue manager configuration” on page 222
3. Optional: “Optional: Extracting and acquiring the queue manager keys and certificates” on page 222
4. Optional: “Optional: Configuring LDAP” on page 224
5. Optional: “Optional: Changing the IP addresses and host names in the IBM MQ configuration” on page 232
6. “Updating the queue manager configuration for a container environment” on page 233
7. “Selecting the target HA architecture for IBM MQ running in containers” on page 236
8. “Creating the resources for the queue manager” on page 236
9. “Creating the new queue manager on Red Hat OpenShift” on page 238
10. “Verifying the new container deployment” on page 242

OpenShift CP4I-SC2 CD Checking that required functions are available

The IBM MQ Operator does not include all the features available within IBM MQ Advanced, and you must verify that these features are not required. Other features are partially supported, and can be reconfigured to match what is available in the container.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This is the first step in the “Migrating to the IBM MQ Operator on Red Hat OpenShift” on page 220.

Procedure

1. Verify that the target container image includes all the functions required.

   For the latest information, see “Choosing how you want to use IBM MQ in containers” on page 9.
2. The IBM MQ Operator has a single IBM MQ traffic port, known as a listener. If you have multiple listeners, simplify this to using a single listener in the container. Because this is not a common scenario this modification is not documented in detail.
3. If IBM MQ exits are used, migrate them into the container by layering in the IBM MQ exit binaries. This is an advanced migration scenario and therefore not included here. For an outline of the steps, see “Building an image with custom MQSC and INI files, using the Red Hat OpenShift CLI” on page 185.
4. If your IBM MQ system includes High Availability, review the available options.

   See “Planning high availability for IBM MQ in containers” on page 20.

What to do next

You are now ready to extract the queue manager configuration.

OpenShift CP4I-SC2 CD Extracting the queue manager configuration

The majority of configuration is portable between queue managers. For example the things that applications interact with, such as definitions of queues, topics and channels. Use this task to extract the configuration from the existing IBM MQ queue manager.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have checked that required functions are available.

Procedure

1. Log into the machine with the existing IBM MQ installation.
2. Back up the configuration.

   Run the following command:

   ```
   dmpmqcfg -m QMGR_NAME > /tmp/backup.mqsc
   ```

   Usage notes for this command:

   - This command stores the backup in the `tmp` directory. You can store the backup in another location, but this scenario assumes the `tmp` directory for subsequent commands.
   - Replace *QMGR_NAME* with the queue manager name from your environment. If you are unsure of the value, run the **dspmq** command to view the available queue managers on the machine. Here is sample **dspmq** command output for a queue manager named `qm1`:

     ```
     QMNAME(qm1)                   STATUS(Running)
     ```

     The **dspmq** command requires the IBM MQ queue manager to be started, otherwise you receive the following error:

     ```
     AMQ8146E: IBM MQ queue manager not available.
     ```

     If required, start the queue manager by running the following command:

     ```
     strmqm QMGR_NAME
     ```

What to do next

You are now ready to extract and acquire the queue manager keys and certificates.

OpenShift CP4I-SC2 CD Optional: Extracting and acquiring the queue manager keys and certificates

IBM MQ can be configured to encrypt network traffic into the queue manager with TLS. Use this task to verify that your queue manager is using TLS, to extract keys and certificates, and to configure TLS on the migrated queue manager.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have extracted the queue manager configuration.

About this task

Do I need to do this?

IBM MQ can be configured to encrypt traffic into the queue manager. This encryption is completed using a key repository that is configured on the queue manager. IBM MQ channels then enable the TLS communication. If you are unsure whether TLS communication is configured in your environment, run the following command to verify:

```
grep 'SECCOMM(ALL\|SECCOMM(ANON\|SSLCIPH' backup.mqsc
```

If no results are found, TLS is not being used. However, this does not mean that TLS should not be configured in the migrated queue manager. There are several reasons why you might want to change this behavior:

- The security approach on the Red Hat OpenShift environment should be enhanced compared to the previous environment.
- If you need to access the migrated queue manager from outside of the Red Hat OpenShift environment, TLS is required to pass through the Red Hat OpenShift Route.

Note: Queue manager certificates with the same Subject Distinguished Name (DN) as the issuer (CA) certificate are not supported. A certificate must have a unique Subject Distinguished Name. The product checks that the DNs are not the same.

Procedure

1. Extract any trusted certificates from the existing store.

   If TLS is currently in use on the queue manager, the queue manager might have a number of trusted certificates stored. These need to be extracted and copied to the new queue manager. Complete one of the following optional steps:

   - To streamline the extraction of the certificates, run the following script on the local system:

     ```
     #!/bin/bash

     keyr=$(grep SSLKEYR $1)
     if [ -n "${keyr}" ]; then
        keyrlocation=$(sed -n "s/^.*'\(.*\)'.*$/\1/ p" <<< ${keyr})
        mapfile -t runmqakmResult < <(runmqakm -cert -list -db ${keyrlocation}.kdb -stashed)
        cert=1
        for i in "${runmqakmResult[@]:2}"
        do
           certlabel=$(echo ${i:2} | xargs)
           echo Extracting certificate $certlabel to $cert.cert
           runmqakm -cert -extract -db ${keyrlocation}.kdb -label "$certlabel" -target $
     {cert}.cert -stashed
           cert=$[$cert+1]
        done
     fi
     ```

     When running the script, specify the location of the IBM MQ backup as an argument and the certificates are extracted. For instance, if the script is called `extractCert.sh` and the IBM MQ backup is located at `/tmp/backup.mqsc` then run the following command:

     ```
     extractCert.sh /tmp/backup.mqsc
     ```

   - Alternatively, run the following commands in the order shown:

     a. Identify the location of the queue manager's TLS key repository:

     ```
     grep SSLKEYR /tmp/backup.mqsc
     ```

     Sample output:

     ```
     SSLKEYR('/run/runmqserver/tls/key') +
     ```

where the key store is located at `/run/runmqserver/tls/key.kdb`

b. Based on this location information, query the key store to determine the stored certificates:

```
runmqakm -cert -list -db /run/runmqserver/tls/key.kdb -stashed
```

Sample output:

```
Certificates in database /run/runmqserver/tls/key.kdb:
           default
           CN=cs-ca-certificate,O=cert-manager
```

c. Extract each of the listed certificates. Do this by running the following command:

```
runmqakm -cert -extract -db KEYSTORE_LOCATION -label "LABEL_NAME" -target OUTPUT_FILE
-stashed
```

In the samples previously shown, this equates to the following commands:

```
runmqakm -cert -extract -db /run/runmqserver/tls/key.kdb -label "CN=cs-ca-
certificate,O=cert-manager" -target /tmp/cert-manager.crt -stashed
runmqakm -cert -extract -db /run/runmqserver/tls/key.kdb -label "default" -target /tmp/
default.crt -stashed
```

2. Acquire a new key and certificate for the queue manager

To configure TLS on the migrated queue manager, you generate a new key and certificate. This is then used during the deployment. In many organizations this means contacting your security team to request a key and certificate. In some organizations this option is not available, and self-signed certificates are used.

The following example generates a self-signed certificate where the expiry is set to 10 years:

```
openssl req \
       -newkey rsa:2048 -nodes -keyout qmgr.key \
       -subj "/CN=mq queuemanager/OU=ibm mq" \
       -x509 -days 3650 -out qmgr.crt
```

Two new files are created:

- `qmgr.key` is the private key for the queue manager
- `qmgr.crt` is the public certificate

What to do next

You are now ready to configure LDAP.

OpenShift CP4I-SC2 CD Optional: Configuring LDAP

The IBM MQ Operator can be configured to use several different security approaches. Typically LDAP is the most effective for an enterprise deployment, and LDAP is used for this migration scenario.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have extracted and acquired the queue manager keys and certificates.

About this task

Do I need to do this?

If you are already using LDAP for authentication and authorization then no changes are required.

If you are not sure if LDAP is being used, run the following command:

```
connauthname="$(grep CONNAUTH backup.mqsc | cut -d "(" -f2 | cut -d ")" -f1)"; grep -A 20
AUTHINFO\($connauthname\) backup.mqsc
```

Sample output:

```
DEFINE AUTHINFO('USE.LDAP') +
   AUTHTYPE(IDPWLDAP) +
   ADOPTCTX(YES) +
   CONNAME('ldap-service.ldap(389)') +
   CHCKCLNT(REQUIRED) +
   CLASSGRP('groupOfUniqueNames') +
   FINDGRP('uniqueMember') +
   BASEDNG('ou=groups,dc=ibm,dc=com') +
   BASEDNU('ou=people,dc=ibm,dc=com') +
   LDAPUSER('cn=admin,dc=ibm,dc=com') +
*  LDAPPWD('******************************') +
   SHORTUSR('uid') +
   GRPFIELD('cn') +
   USRFIELD('uid') +
   AUTHORMD(SEARCHGRP) +
*  ALTDATE(2020-11-26) +
*  ALTTIME(15.44.38) +
   REPLACE
```

There are two attributes in the output that are of particular interest:

AUTHTYPE
If this has the value IDPWLDAP, then you are using LDAP for authentication.

If the value is blank, or another value, then LDAP is not configured. In this case, check the AUTHORMD attribute to see if LDAP users are being used for authorization.

AUTHORMD
If this has the value OS, then you are not using LDAP for authorization.

To modify the authorization and authentication to use LDAP, complete the following tasks:

Procedure

1. Update the IBM MQ backup for the LDAP server.
2. Update the IBM MQ backup for LDAP authorization information.

OpenShift CP4I-SC2 CD LDAP part 1: Updating the IBM MQ backup for the LDAP server

A comprehensive description of how to set up LDAP is outside the scope of this scenario. This topic gives a summary of the process, a sample, and references to further information.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have extracted and acquired the queue manager keys and certificates.

About this task

Do I need to do this?

If you are already using LDAP for authentication and authorization then no changes are required. If you are not sure if LDAP is being used, see “Optional: Configuring LDAP” on page 224.

There are two parts to setting up the LDAP server:

1. Define an LDAP configuration.

2. Associate the LDAP configuration with the queue manager definition.

Further information to help you with this configuration:

- User Repository Overview
- Reference guide to the AUTHINFO command

Procedure

1. Define an LDAP configuration.

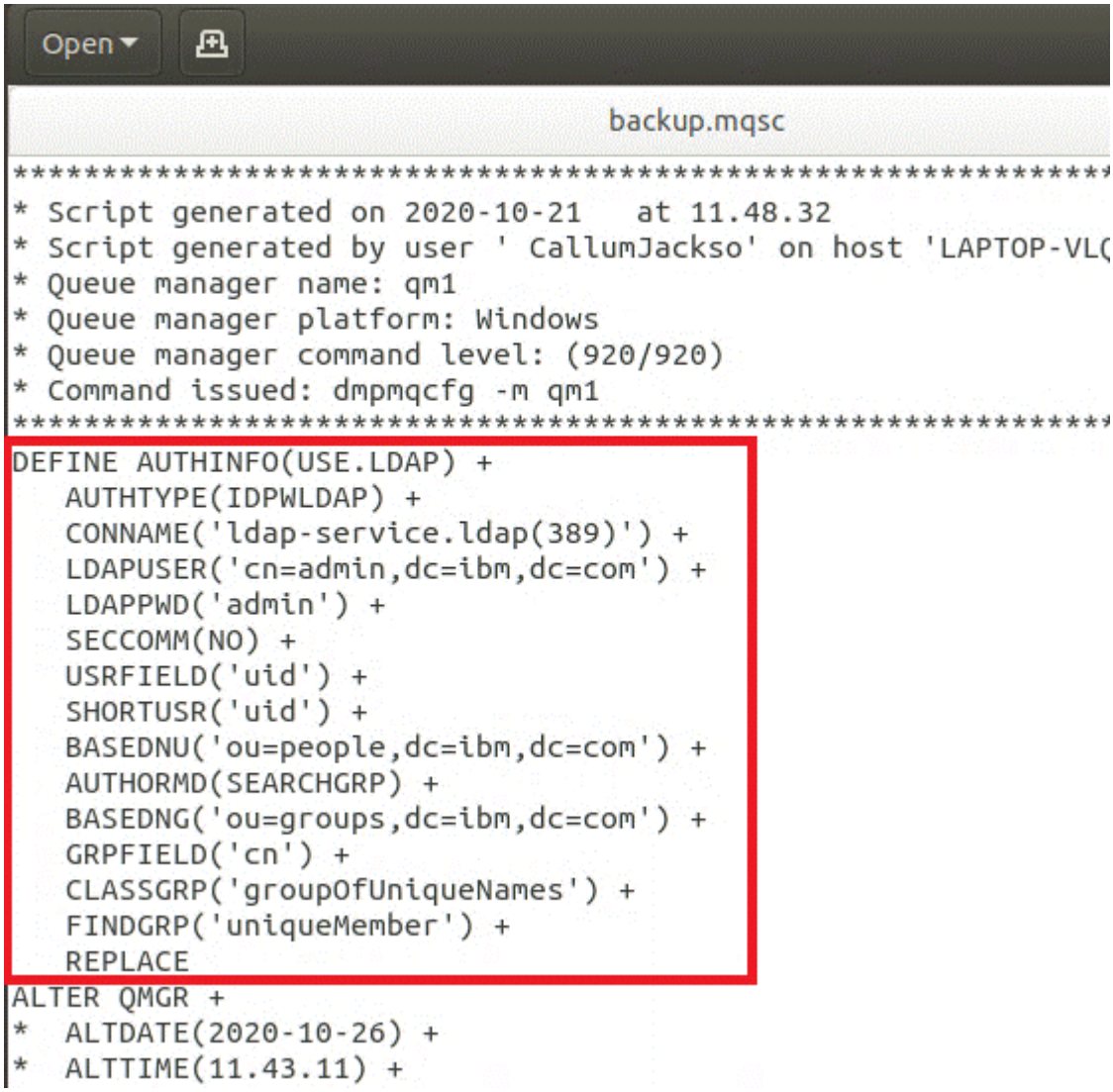
   Edit the `backup.mqsc` file to define a new **AUTHINFO** object for the LDAP system. For example:

```
DEFINE AUTHINFO(USE.LDAP) +
   AUTHTYPE(IDPWLDAP) +
   CONNAME('ldap-service.ldap(389)') +
   LDAPUSER('cn=admin,dc=ibm,dc=com') +
   LDAPPWD('admin') +
   SECCOMM(NO) +
   USRFIELD('uid') +
   SHORTUSR('uid') +
   BASEDNU('ou=people,dc=ibm,dc=com') +
   AUTHORMD(SEARCHGRP) +
   BASEDNG('ou=groups,dc=ibm,dc=com') +
   GRPFIELD('cn') +
   CLASSGRP('groupOfUniqueNames') +
   FINDGRP('uniqueMember')
   REPLACE
```

   where

   - **CONNAME** is the hostname and port corresponding to the LDAP server. If multiple addresses exist for resilience then these can be configured using a comma-separated list.
   - **LDAPUSER** is the distinguished name corresponding to the user that IBM MQ uses when connecting to LDAP to query user records.
   - **LDAPPWD** is the password that corresponds to the **LDAPUSER** user.
   - **SECCOM** specifies whether the communication to the LDAP server should use TLS. Possible values:
     - YES: TLS is used and a certificate is presented by the IBM MQ server.
     - ANON: TLS is used without a certificate being presented by the IBM MQ server.
     - NO: TLS is not used during the connection.
   - **USRFIELD** specifies the field in the LDAP record that the presented username be matched against.
   - **SHORTUSR** is a field within the LDAP record that does not exceed 12 characters in length. The value within this field be the asserted identity if authentication is successful.
   - **BASEDNU** is the base DN that should be used for searching LDAP.
   - **BASEDNG** is the base DN for groups within LDAP.
   - **AUTHORMD** defines the mechanism used to resolve group membership for the user. There are four options:
     - OS: Query the operating system for the groups associated with the short name.
     - SEARCHGRP: Search the group entries in LDAP for the authenticated user.
     - SEARCHUSR: Search the authenticated user record for group membership information.
     - SRCHGRPSN: Search the group entries in LDAP for the authenticated users short user name (defined by the SHORTUSR field).
   - **GRPFIELD** is the attribute within the LDAP group record that corresponds to a simple name. If specified this can be used for defining authorization records.
   - **CLASSUSR** is the LDAP object class that corresponds to a user.
   - **CLASSGRP** is the LDAP object class that corresponds to a group.
   - **FINDGRP** is the attribute within the LDAP record that corresponds to group membership.

The new entry can be placed anywhere within the file, however you might find it helpful to have any new entries at the beginning of the file:



2. Associate the LDAP configuration with the queue manager definition.

   You need to associate the LDAP configuration with the queue manager definition. Immediately below the `DEFINE AUTHINFO` entry is an `ALTER QMGR` entry. Modify the CONNAUTH entry to correspond to the newly created AUTHINFO name. For example in the previous example `AUTHINFO(USE.LDAP)` was defined, meaning the name is `USE.LDAP`. Therefore change `CONNAUTH('SYSTEM.DEFAULT.AUTHINFO.IDPWOS')` to `CONNAUTH('USE.LDAP')`:

To cause the switch to LDAP to occur immediately, call a REFRESH SECURITY command by adding a line immediately after the ALTER QMGR command:

*backup.mqsc

```
********************************************************************
* Script generated on 2020-10-21   at 11.48.32
* Script generated by user ' CallumJackso' on host 'LAPTOP-VLQKJ5UH'
* Queue manager name: qm1
* Queue manager platform: Windows
* Queue manager command level: (920/920)
* Command issued: dmpmqcfg -m qm1
********************************************************************
DEFINE AUTHINFO(USE.LDAP) +
   AUTHTYPE(IDPWLDAP) +
   CONNAME('ldap-service.ldap(389)') +
   LDAPUSER('cn=admin,dc=ibm,dc=com') +
   LDAPPWD('admin') +
   SECCOMM(NO) +
   USRFIELD('uid') +
   SHORTUSR('uid') +
   BASEDNU('ou=people,dc=ibm,dc=com') +
   AUTHORMD(SEARCHGRP) +
   BASEDNG('ou=groups,dc=ibm,dc=com') +
   GRPFIELD('cn') +
   CLASSGRP('groupOfUniqueNames') +
   FINDGRP('uniqueMember') +
   REPLACE
ALTER QMGR +
*  ALTDATE(2020-10-26) +
*  ALTTIME(11.43.11) +
   CCSID(850) +
   CERTLABL('default') +
   CLWLUSEQ(LOCAL) +
*  COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE) +
   CONNAUTH('USE.LDAP') +
*  CRDATE(2020-10-26) +
*  CRTIME(11.43.11) +
*  QMID(qm1_2020-10-26_11.43.11) +
   SSLCRYP(' ') +
   SSLKEYR('/run/runmqserver/tls/key') +
   SUITEB(NONE) +
*  VERSION(09020000) +
   FORCE
REFRESH SECURITY
```

What to do next

You are now ready to update the IBM MQ backup for LDAP authorization information.

OpenShift CP4I-SC2 CD LDAP part 2: Updating the IBM MQ backup for LDAP authorization information

IBM MQ provides fine grained authorization rules that control access to the IBM MQ objects. If you changed the authentication and authorization to LDAP, the authorization rules might be invalid and require updating.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have updated the backup for the LDAP server.

About this task

Do I need to do this?

If you are already using LDAP for authentication and authorization then no changes are required. If you are not sure if LDAP is being used, see “Optional: Configuring LDAP” on page 224.

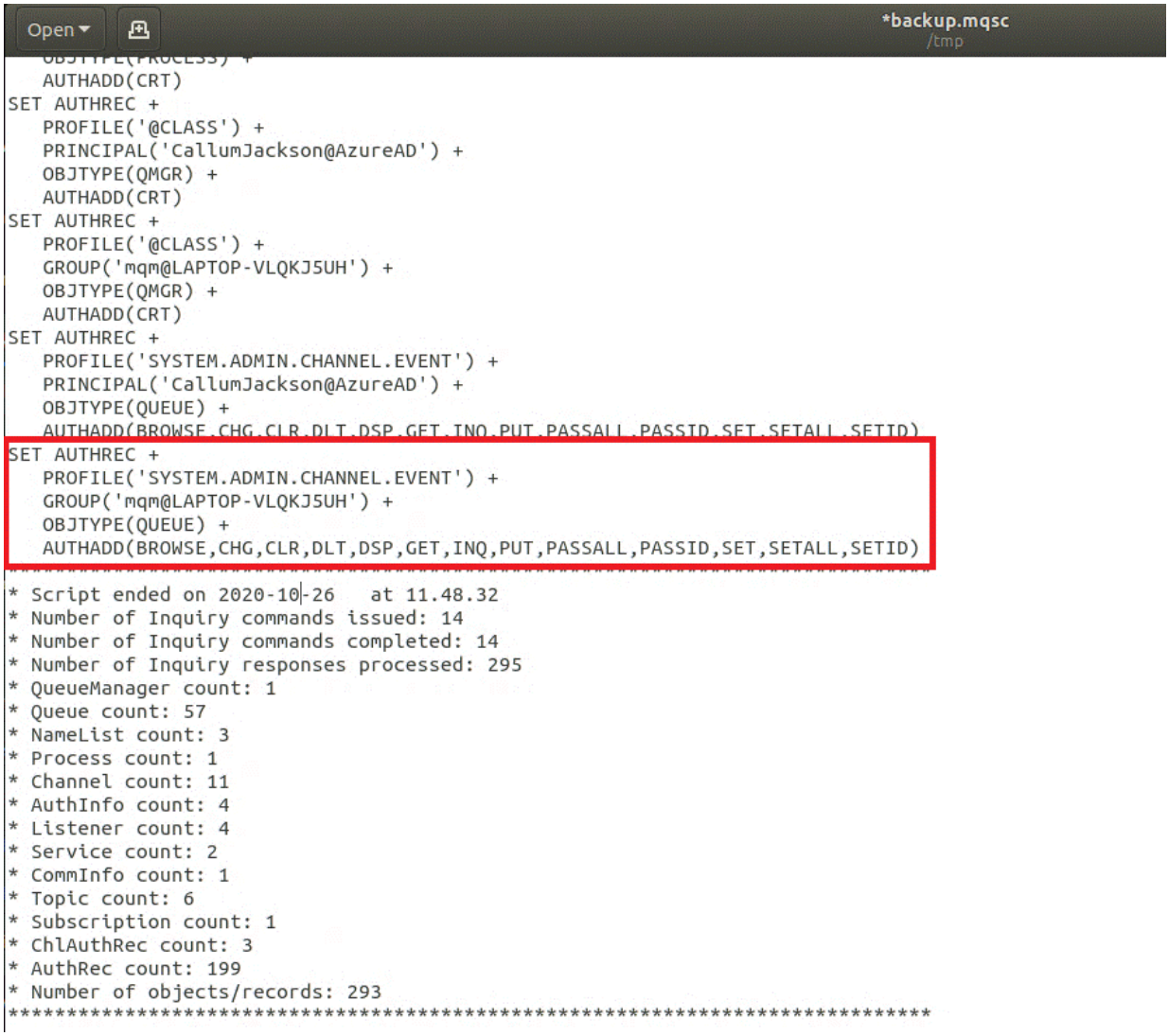
There are two parts to updating the LDAP authorization information:

1. Remove all existing authorization from the file.
2. Define new authorization information for LDAP.

Procedure

1. Remove all existing authorization from the file.

   In the backup file, near to the end of the file, you should see several entries that start with SET AUTHREC:



Find the existing entries and delete them. The most straightforward approach is to remove all the existing SET AUTHREC rules, then create new entries based on the LDAP entries.

2. Define new authorization information for LDAP

Depending on your queue manager configuration, and the number of resources and groups, this could be either a time consuming or straightforward activity. The following example assumes that your queue manager has only a single queue called Q1, and you want to allow the LDAP group apps to have access.

```
SET AUTHREC GROUP('apps') OBJTYPE(QMGR) AUTHADD(ALL)
SET AUTHREC PROFILE('Q1') GROUP('apps') OBJTYPE(QUEUE) AUTHADD(ALL)
```

The first AUTHREC command adds permission to access the queue manager, and the second provides access to the queue. If access to a second queue is required then a third AUTHREC command is needed, unless you decided to use wildcards to provide more generic access.

Here is another example. If an administrator group (called admins) needs full access to the queue manager, add the following commands:

```
SET AUTHREC PROFILE('*') OBJTYPE(QUEUE) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(TOPIC) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(CHANNEL) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(CLNTCONN) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(AUTHINFO) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(LISTENER) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(NAMELIST) GROUP('admins') AUTHADD(ALL)
```

```
SET AUTHREC PROFILE('*') OBJTYPE(PROCESS) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(SERVICE) GROUP('admins') AUTHADD(ALL)
SET AUTHREC PROFILE('*') OBJTYPE(QMGR) GROUP('admins') AUTHADD(ALL)
```

What to do next

You are now ready to change the IP addresses and host names in the IBM MQ configuration.

OpenShift CP4I-SC2 CD Optional: Changing the IP addresses and host names in the IBM MQ configuration

The IBM MQ configuration might have IP addresses and host names specified. In some situations these can remain, while in other situations they need to be updated.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have configured LDAP.

About this task

Do I need to do this?

First, determine if you have any IP addresses or host names specified, apart from the LDAP configuration defined in the previous section. To do this, run the following command:

```
grep 'CONNAME\|LOCLADDR\|IPADDRV' -B 3  backup.mqsc
```

Sample output:

```
************************************************
DEFINE AUTHINFO(USE.LDAP) +
   AUTHTYPE(IDPWLDAP) +
   CONNAME('ldap-service.ldap(389)') +
--
DEFINE AUTHINFO('SYSTEM.DEFAULT.AUTHINFO.IDPWLDAP') +
   AUTHTYPE(IDPWLDAP) +
   ADOPTCTX(YES) +
   CONNAME(' ') +
--
   REPLACE
DEFINE AUTHINFO('SYSTEM.DEFAULT.AUTHINFO.CRLLDAP') +
   AUTHTYPE(CRLLDAP) +
   CONNAME(' ') +
```

In this example the search returns three results. One result corresponds to the LDAP configuration defined previously. This can be ignored, because the hostname of the LDAP server is remaining the same. The other two results are empty connection entries, so these can be ignored as well. If you do not have any additional entries, you can skip the remainder of this topic.

Procedure

1. Understand the entries returned.

   IBM MQ can include IP addresses, host names and ports within many aspects of the configuration. We can classify these into two categories:

   a. **Location of this queue manager**: Location information that this queue manager uses or publishes, that other queue managers or applications within an IBM MQ network can use for connectivity.

   b. **Location of queue manager dependencies**: The locations of other queue managers or systems that this queue manager needs to be aware of.

   Because this scenario is focused only on the changes to this queue manager configuration, we only handle the configuration updates for category (a). However, if this queue manager location is

referenced by other queue managers or applications, their configurations might need updating to match this queue manager's new location.

There are two key objects that might contain information that needs to be updated:

- Listeners: These represent the network address that IBM MQ is listening on.
- CLUSTER RECEIVER channel: If the queue manager is part of an IBM MQ cluster, then this object exists. It specifies the network address that other queue managers can connect to.

2. In the original output from the `grep 'CONNAME\|LOCLADDR\|IPADDRV' -B 3 backup.mqsc` command, identify if any CLUSTER RECEIVER channels are defined. If so,update the IP addresses.

   To identify if any CLUSTER RECEIVER channels are defined, find any entries with CHLTYPE(CLUSRCVR) in the original output:

   ```
   DEFINE CHANNEL(ANY_NAME) +
      CHLTYPE(CLUSRCVR) +
   ```

   If entries do exist, update the CONNAME with the IBM MQ Red Hat OpenShift Route. This value is based on the Red Hat OpenShift environment and uses a predictable syntax:

   ```
   queue_manager_resource_name-ibm-mq-qm-openshift_project_name.openshift_app_route_hostname
   ```

   For example, if the queue manager deployment is named qm1 within the cp4i namespace, and the *openshift_app_route_hostname* is `apps.callumj.icp4i.com`, then the route URL is this:

   ```
   qm1-ibm-mq-qm-cp4i.apps.callumj.icp4i.com
   ```

   The port number for the route is typically 443. Unless your Red Hat OpenShift Administrator tells you differently, this is normally the correct value. Using this information, update the CONNAME fields. For example:

   ```
   CONNAME('qm1-ibm-mq-qm-cp4i.apps.callumj.icp4i.com(443)')
   ```

   In the original output from the `grep 'CONNAME\|LOCLADDR\|IPADDRV' -B 3 backup.mqsc` command, verify if any entries exist for LOCLADDR or IPADDRV. If they do, delete them. They are not relevant in a container environment.

What to do next

You are now ready to update the queue manager configuration for a container environment.

OpenShift CP4I-SC2 CD Updating the queue manager configuration for a container environment

When running in a container, certain configuration aspects are defined by the container, and might conflict with the exported configuration.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have changed the IBM MQ configuration of IP addresses and hostnames.

About this task

The following configuration aspects are defined by the container:

- The listener definitions (which correspond to the ports exposed).
- The location of any potential TLS store.

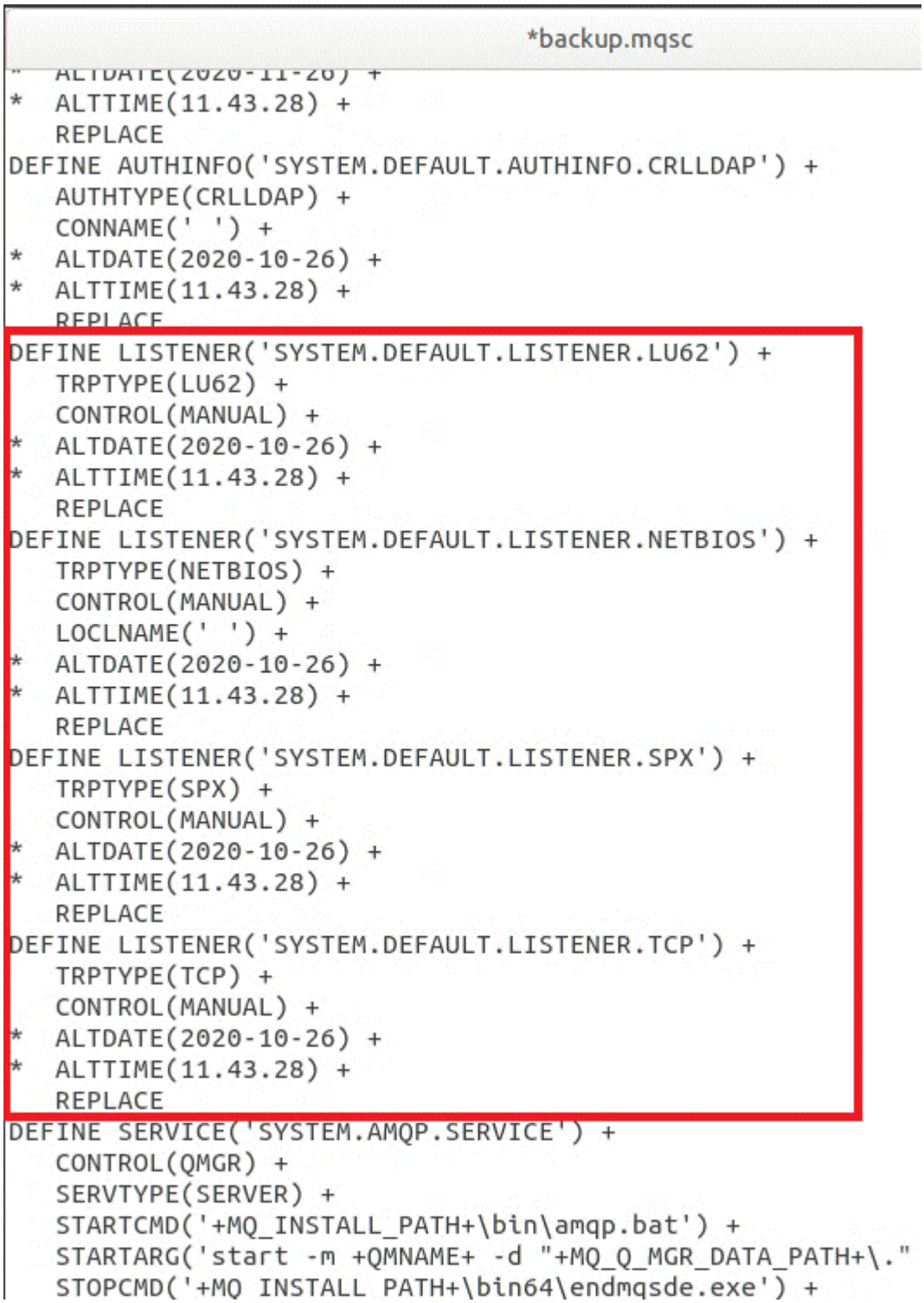
Therefore you need to update the exported configuration:

1. Remove any listener definitions.
2. Define the location of the TLS key repository.

Procedure

1. Remove any listener definitions.

   In the backup configuration, search for `DEFINE LISTENER`. This should be between the `AUTHINFO` and `SERVICE` definitions. Highlight the area, and delete it.



2. Define the location of the TLS key repository.

The queue manager backup contains the TLS configuration for the original environment. This is different from the container environment, and therefore a couple of updates are required:

- Change the **CERTLABL** entry to `default`
- Change the location of the TLS key repository (**SSLKEYR**) to: `/run/runmqserver/tls/key`

To find the location of the **SSLKEYR** attribute in the file, search for **SSLKEYR**. Typically only one entry is found. If multiple entries are found, check that you are editing the **QMGR** object as shown in the following illustration:

*backup.mqsc

```
************************************************************************
* Script generated on 2020-10-21   at 11.48.32
* Script generated by user ' CallumJackso' on host 'LAPTOP-VLQKJ5UH'
* Queue manager name: qm1
* Queue manager platform: Windows
* Queue manager command level: (920/920)
* Command issued: dmpmqcfg -m qm1
************************************************************************
DEFINE AUTHINFO(USE.LDAP) +
   AUTHTYPE(IDPWLDAP) +
   CONNAME('ldap-service.ldap(389)') +
   LDAPUSER('cn=admin,dc=ibm,dc=com') +
   LDAPPWD('admin') +
   SECCOMM(NO) +
   USRFIELD('uid') +
   SHORTUSR('uid') +
   BASEDNU('ou=people,dc=ibm,dc=com') +
   AUTHORMD(SEARCHGRP) +
   BASEDNG('ou=groups,dc=ibm,dc=com') +
   GRPFIELD('cn') +
   CLASSGRP('groupOfUniqueNames') +
   FINDGRP('uniqueMember') +
   REPLACE
ALTER QMGR +
*  ALTDATE(2020-10-26) +
*  ALTTIME(11.43.11) +
   CCSID(850) +
   CERTLABL('default') +
   CLWLUSEQ(LOCAL) +
*  COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE) +
   CONNAUTH('USE.LDAP') +
*  CRDATE(2020-10-26) +
*  CRTIME(11.43.11) +
*  QMID(qm1_2020-10-26_11.43.11) +
   SSLCRYP(' ') +
   SSLKEYR('/run/runmqserver/tls/key') +
   SUITEB(NONE) +
*  VERSION(09020000) +
   FORCE
REFRESH SECURITY
```

What to do next

You are now ready to select the target architecture for IBM MQ running in containers.

OpenShift CP4I-SC2 CD Selecting the target HA architecture for IBM MQ running in containers

Choose between single instance (a single Kubernetes Pod), multi-instance (two Pods), and Native HA (one active replica Pod and two standby replica Pods) to meet your high availability requirements.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have updated the queue manager configuration for a container environment.

About this task

The IBM MQ Operator provides three high availability options:

- **Single instance**: A single container (Pod) is started and it is the responsibility of Red Hat OpenShift to restart in the event of a failure. Because of the characteristics of a stateful set within Kubernetes, there are several situations in which this failover might take an extended period of time, or require an administrative action to be completed.
- **Multi-instance**: Two containers (each in a separate Pod) are started, one in active mode and another on standby. This topology enables much faster failover. It requires a Read Write Many file system that meets IBM MQ requirements.
- **Native HA**: Three containers (each in a separate Pod), each with an instance of the queue manager. One instance is the active queue manager, processing messages and writing to its recovery log. Whenever the recovery log is written, the active queue manager sends the data to the other two instances, known as replicas. If the Pod running the active queue manager fails, one of the replica instances of the queue manager takes over the active role and has current data to operate with.

In this task you only choose the target HA architecture. Steps for configuring your chosen architecture are described in a subsequent task in this scenario ("Creating the new queue manager on Red Hat OpenShift" on page 238).

Procedure

1. Review the three options.

   For a comprehensive description of these options, see "Planning high availability for IBM MQ in containers" on page 20.
2. Select the target HA architecture.

   If you are unsure which option to choose, start with the **Single instance** option, and verify if this meets your high availability requirements.

What to do next

You are now ready to create the queue manager resources.

OpenShift CP4I-SC2 CD Creating the resources for the queue manager

Import the IBM MQ configuration, and the TLS certificates and keys, into the Red Hat OpenShift environment.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have selected the target architecture for IBM MQ running in containers.

About this task

In the previous sections you have extracted, updated and defined two resources:

- IBM MQ configuration
- TLS certificates and keys

You need to import these resources into the Red Hat OpenShift environment before the queue manager is deployed.

Procedure

1. Import the IBM MQ configuration into Red Hat OpenShift.

   The following instructions assume that you have the IBM MQ configuration in the current directory, in a file called `backup.mqsc`. Otherwise, you need to customize the filename based on your environment.

   a) Log into your cluster using `oc login`.

   b) Load the IBM MQ configuration into a `configmap`.

      Run the following command:

      ```
      oc create configmap my-mqsc-migrated --from-file=backup.mqsc
      ```

   c) Verify that the file has loaded successfully.

      Run the following command:

      ```
      oc describe configmap my-mqsc-migrated
      ```

2. Import the IBM MQ TLS resources

   As discussed in "Optional: Extracting and acquiring the queue manager keys and certificates" on page 222, TLS might be required for the queue manager deployment. If so, you should already have a number of files ending with `.crt` and `.key`. You need to add these into Kubernetes secrets for the queue manager to reference at deployment time.

   For example, if you had a key and certificate for the queue manager they might be called:

   - `qmgr.crt`
   - `qmgr.key`

   To import these files, run the following command:

   ```
   oc create secret tls my-tls-migration --cert=qmgr.crt --key=qmgr.key
   ```

   Kubernetes provides this helpful utility when you are importing a matching public and private key. If you have additional certificates to add, for instance into the queue manager trust store, run the following command:

   ```
   oc create secret generic my-extra-tls-migration --from-file=comma_separated_list_of_files
   ```

   For example, if the files to be imported are `trust1.crt`, `trust2.crt` and `trust3.crt`, the command is this:

   ```
   oc create secret generic my-extra-tls-migration --from-file=trust1.crt,trust2.crt,trust3.crt
   ```

What to do next

You are now ready to create the new queue manager on Red Hat OpenShift.

OpenShift CP4I-SC2 CD Creating the new queue manager on Red Hat OpenShift

Deploy either a single instance or multi-instance queue manager on Red Hat OpenShift.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have created the queue manager resources, and installed the IBM MQ Operator into Red Hat OpenShift.

About this task

As outlined in "Selecting the target HA architecture for IBM MQ running in containers" on page 236, there are three possible deployment topologies. Therefore this topic provides three different templates:

- Template 1: Deploy a single instance queue manager.
- Template 2: Deploy a multi instance queue manager.
- Template 3: Deploy a Native HA queue manager.

Important: Only complete one of the three templates, based on your preferred topology.

Procedure

- **Template 1: Deploy a single instance queue manager.**

  The migrated queue manager is deployed to Red Hat OpenShift using a YAML file. Here is a sample, based on the names used in previous topics:

  Note: This example uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: qm1
spec:
  version: 10.0.0.0-r3
  license:
    accept: true
    license: L-VEMB-RABZSN
    use: "Production"
  pki:
    keys:
    - name: default
      secret:
        secretName: my-tls-migration
        items:
          - tls.key
          - tls.crt
  web:
    enabled: true
  queueManager:
    name: QM1
    mqsc:
      - configMap:
          name: my-mqsc-migrated
          items:
            - backup.mqsc
```

  Depending on the steps that you performed, the previous YAML might need to be customized. To help you with this, here is an explanation of this YAML:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
```

```
metadata:
  name: qm1
```

This defines the Kubernetes object, type and name. The only field requiring customization is the `name` field.

```
 spec:
  version: 10.0.0.0-r3
  license:
    accept: true
    license: L-VEMB-RABZSN
    use: "Production"
```

This corresponds to the version and license information for the deployment. If you need to customize this, use the information provided in “mq.ibm.com/v1beta1: Current license versions” on page 281.

```
  pki:
    keys:
    - name: default
      secret:
        secretName: my-tls-migration
        items:
          - tls.key
          - tls.crt
```

For the queue manager to be configured to use TLS, it must reference the relevant certificates and keys. The `secretName` field references the Kubernetes secret created within the Import the IBM MQ TLS resources section, and the list of items (`tls.key` and `tls.crt`) are the standard names Kubernetes assigns when using the `oc create secret tls` syntax. If you have additional certificates to add into the trust store then these can be added in a similar way, but the items are the corresponding file names used during the import. For example, the following code can be used to create the trust store certificates:

```
oc create secret generic my-extra-tls-migration --from-file=trust1.crt,trust2.crt,trust3.crt
```

```
  pki:
    trust:
    - name: default
      secret:
        secretName: my-extra-tls-migration
        items:
          - trust1.crt
          - trust2.crt
          - trust3.crt
```

Important: If TLS is not required, delete the TLS section of the YAML.

```
  web:
    enabled: true
```

This enables the web console for the deployment

```
  queueManager:
    name: QM1
```

This defines the name of the queue manager as QM1. The queue manager is customized based on your requirements, for instance what was the original queue manager name.

```
    mqsc:
      - configMap:
          name: my-mqsc-migrated
          items:
            - backup.mqsc
```

The previous code pulls in the queue manager configuration that was imported in the Import the IBM MQ configuration section. If you used different names, you need to modify `my-mqsc-migrated` and `backup.mqsc`.

Note that the sample YAML assumes that the default storage class for the Red Hat OpenShift environment is defined as either a RWX or RWO storage class. If a default is not defined within your environment, then you need to specify the storage class to be used. You can do this by extending the YAML as follows:

```
  queueManager:
    name: QM1
    storage:
      defaultClass: my_storage_class
      queueManager:
        type: persistent-claim
```

Add the highlighted text, with the class attribute customized to match your environment. To discover the storage class names within your environment, run the following command:

```
oc get storageclass
```

Here is sample output returned by this command:

```
NAME          PROVISIONER             RECLAIMPOLICY
aws-efs       openshift.org/aws-efs   Delete
gp2 (default) kubernetes.io/aws-ebs   Delete
```

The following code shows how to reference the IBM MQ configuration that was imported in the Import the IBM MQ configuration section. If you used different names, you need to modify `my-mqsc-migrated` and `backup.mqsc`.

```
    mqsc:
      - configMap:
          name: my-mqsc-migrated
          items:
            - backup.mqsc
```

You have deployed your single instance queue manager. This completes the template. You are now ready to verify the new container deployment.

- **Template 2: Deploy a multi instance queue manager.**

  The migrated queue manager is deployed to Red Hat OpenShift using a YAML file. The following sample is based on the names used in previous sections.

  Note: This example uses an IBM MQ Advanced license, though you can also use an IBM MQ license. For more information, see “Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator” on page 187.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: qm1mi
spec:
  version: 10.0.0.0-r3
  license:
    accept: true
    license: L-VEMB-RABZSN
    use: "Production"
  pki:
    keys:
    - name: default
      secret:
        secretName: my-tls-migration
        items:
          - tls.key
          - tls.crt
  web:
    enabled: true
  queueManager:
    name: QM1
    availability: MultiInstance
    storage:
      defaultClass: aws-efs
      persistedData:
        enabled: true
```

```
      queueManager:
        enabled: true
      recoveryLogs:
        enabled: true
    mqsc:
      - configMap:
          name: my-mqsc-migrated
          items:
            - backup.mqsc
```

Here is an explanation of this YAML. The majority of the configuration follows the same approach as deploying a single instance queue manager, therefore only the queue manager availability and storage aspects are explained here.

```
queueManager:
    name: QM1
    availability: MultiInstance
```

This specifies the queue manager name as `QM1`, and sets the deployment to be `MultiInstance` instead of the default single instance.

```
    storage:
      defaultClass: aws-efs
      persistedData:
        enabled: true
      queueManager:
        enabled: true
      recoveryLogs:
        enabled: true
```

An IBM MQ multi-instance queue manager depends on RWX storage. By default, a queue manager is deployed in single instance mode and therefore additional storage options are required when changing to multi instance mode. In the previous YAML sample, three storage persistent volumes and a persisted volume class are defined. This persisted volume class needs to be a RWX storage class. If you are unsure of the storage class names within your environment, you can run the following command to discover them:

```
oc get storageclass
```

Here is sample output returned by this command:

```
NAME          PROVISIONER             RECLAIMPOLICY
aws-efs       openshift.org/aws-efs   Delete
gp2 (default) kubernetes.io/aws-ebs   Delete
```

The following code shows how to reference the IBM MQ configuration that was imported in the Import the IBM MQ configuration section. If you used different names, you need to modify `my-mqsc-migrated` and `backup.mqsc`.

```
    mqsc:
      - configMap:
          name: my-mqsc-migrated
          items:
            - backup.mqsc
```

You have deployed your multi instance queue manager. This completes the template. You are now ready to verify the new container deployment.

- **Template 3: Deploy a Native HA queue manager.**

  For a worked example of creating a Native HA queue manager, see “Example: Configuring Native HA using the IBM MQ Operator” on page 119.

OpenShift CP4I-SC2 CD Verifying the new container deployment

Now that IBM MQ is deployed on Red Hat OpenShift, you can verify the environment using the IBM MQ samples.

Before you begin

Important: This task applies only to users on Red Hat OpenShift.

This task assumes that you have created the new queue manager on Red Hat OpenShift.

Important: This task assumes that TLS is not enabled in the queue manager.

About this task

In this task you run the IBM MQ samples from inside the migrated queue manager's container. However you might prefer to use your own applications running from another environment.

You need the following information:

- LDAP Username
- LDAP Password
- IBM MQ Channel name
- Queue name

This example code uses the following settings. Please note that your settings will differ.

- LDAP Username: mqapp
- LDAP Password: mqapp
- IBM MQ Channel name: DEV.APP.SVRCONN
- Queue name: Q1

Procedure

1. Exec into the running IBM MQ container.

   Use the following command:

   ```
   oc exec -it qm1-ibm-mq-0 /bin/bash
   ```

   where `qm1-ibm-mq-0` is the Pod that we deployed in “Creating the new queue manager on Red Hat OpenShift” on page 238. If you called the deployment something different, then customize this value.
2. Send a message.

   Run the following commands:

   ```
   cd /opt/mqm/samp/bin
   export IBM MQSAMP_USER_ID=mqapp
   export IBM MQSERVER=DEV.APP.SVRCONN/TCP/'localhost(1414)'
   ./amqsputc Q1 QM1
   ```

   You are prompted for a password, then you can send a message.
3. Verify that the message has been received successfully.

   Run the GET sample:

   ```
   ./amqsgetc Q1 QM1
   ```

Results

You have completed the “Migrating to the IBM MQ Operator on Red Hat OpenShift” on page 220.

What to do next

Use the following information to help you with more complex migration scenarios:

Migrating queued messages
To migrate existing queued messages, follow guidance in the following topic for exporting and importing messages after the new queue manager is in place: Using the dmpmqmsg utility between two systems.

Connecting to IBM MQ from outside the Red Hat OpenShift environment
The deployed queue manager can be exposed to IBM MQ clients and queue managers outside the Red Hat OpenShift environment. The process depends on the version of IBM MQ connecting into the Red Hat OpenShift environment. See “Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster ” on page 169.

Operating IBM MQ in containers

If you need to operate or interact with IBM MQ queue managers running in containers, then see the following topics for more information.

Procedure

- “Operating IBM MQ using the IBM MQ Operator” on page 243.
- “Viewing the status of Native HA queue managers” on page 258.
- “Ending Native HA queue managers” on page 268.

OpenShift CP4I Operating IBM MQ using the IBM MQ Operator

Procedure

- “Giving permissions for the IBM MQ Console” on page 243.
- “Connecting to the IBM MQ Console” on page 244.
- “Monitoring when using the IBM MQ Operator” on page 244.
- “Stopping a queue manager (mq.ibm.com/stop)” on page 256.
- “Backing up and restoring queue manager configuration” on page 257.

OpenShift CP4I Giving permissions for the IBM MQ Console

Permissions for the IBM MQ Console are managed differently based on your license usage.

About this task

Choose the relevant option based on the license used for your queue manager.

Procedure

- **Option 1:** Configure users and roles through Keycloak.

  If you are using an IBM Cloud Pak for Integration license, and are deployed on Red Hat OpenShift **only**, the IBM MQ Console uses Keycloak for identity and access management.

  – For information about configuring Keycloak, see Identity and Access management in the IBM Cloud Pak for Integration documentation.

  – For information about *MQ Roles* and their corresponding Keycloak roles, see Cloud Pak for Integration roles and permissions in the IBM Cloud Pak for Integration documentation. Roles can be configured for the whole installation, or for each individual instance (queue manager). *MQ Role* equivalents are listed under **Queue manager**.

  - If you previously configured users with IAM on older versions of the IBM MQ Operator, see Migrating users from IAM to Keycloak in the IBM Cloud Pak for Integration documentation.
- **Option 2:** Configure users and roles manually.

  If you are using an IBM MQ license on Red Hat OpenShift, or any license on Amazon EKS, the IBM MQ Console is not preconfigured and you need to configure it yourself.

  - For information on users and roles, see Configuring users and roles.
  - For a simple example, see "Configuring the IBM MQ Console with a basic registry using the IBM MQ Operator" on page 193.
  - Alternatively, if you change a queue manager to use an IBM Cloud Pak for Integration license on Red Hat OpenShift **only**, you can then configure Keycloak as previously described.

OpenShift CP4I Connecting to the IBM MQ Console

How to connect to the IBM MQ Console of a queue manager that has been deployed with the IBM MQ Operator.

Before you begin

You need permission to access the IBM MQ Console. See "Giving permissions for the IBM MQ Console" on page 243.

About this task

On Red Hat OpenShift

The IBM MQ Console URL can be found on the `QueueManager` details page in the Red Hat OpenShift web console or in the IBM Cloud Pak for Integration Platform UI. Alternatively, it can be found from the Red Hat OpenShift CLI by running the following command:

```
oc get queuemanager <QUEUEMANAGER_NAME> -n <NAMESPACE> --output
jsonpath='{.status.adminUiUrl}'
```

On Amazon EKS

To access the IBM MQ Console URL, follow the steps in "Configuring Ingress to connect to a queue manager IBM MQ Console and IBM MQ Operator REST API from outside an Amazon EKS cluster" on page 173. After you configure the `Ingress`, retrieve its `Address` by running the following command:

```
kubectl get ingress -n <QUEUE_MANAGER_NAMESPACE> <INGRESS_NAME> -o
jsonpath="{.status.loadBalancer.ingress[*].hostname}{'\n'}"
```

Related tasks

"Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster " on page 169

You need a Red Hat OpenShift Route to connect an application to an IBM MQ queue manager from outside a Red Hat OpenShift cluster. You must enable TLS on your IBM MQ queue manager and client application because SNI is available in the TLS protocol only when a TLS 1.2 or higher protocol is used. The Red Hat OpenShift Container Platform Router uses SNI for routing requests to the IBM MQ queue manager.

OpenShift CP4I Monitoring when using the IBM MQ Operator

Queue managers managed by the IBM MQ Operator can produce metrics compatible with Prometheus. Queue manager metrics are enabled and served through HTTP by default.

Before you begin

Using HTTPS for queue manager metrics

From 9.4.2.0-r1, queue managers can serve metrics through HTTPS. Prior to 9.4.2.0-r1 queue managers can only serve metrics through HTTP.

Note: All queue managers serve metrics through HTTP by default.

About this task

Prometheus is an open source time-series database and a rule evaluation engine for metrics. The IBM MQ containers expose a metrics endpoint that can be queried by Prometheus. The metrics are generated from the system topics of an IBM MQ queue manager.

OpenShift Container Platform includes a pre-configured, pre-installed, and self-updating monitoring stack that uses a Prometheus server. The OpenShift Container Platform monitoring stack needs to be configured to monitor user-defined projects. See Enabling monitoring for user-defined projects in the Red Hat OpenShift documentation.

If you are deploying on Amazon EKS, you can deploy Prometheus then manually configure the scrape target in Prometheus.

When you create a `QueueManager` with metrics enabled, by default the IBM MQ Operator creates a `ServiceMonitor` that the Prometheus Operator can then discover. If you are using a different method to deploy Prometheus, you must disable the ServiceMonitor and manually configure the scrape target in Prometheus.

Procedure

- View queue manager metrics.

  Red Hat OpenShift

  Use the Red Hat OpenShift Container Platform (OCP) monitoring stack

  1. Open the **Metrics** tab in OCP.
  2. Click **Observe** > **Metrics**.

  Amazon EKS

  1. Set `.spec.queueManager.metrics.serviceMonitor.enabled` to `false` because `ServiceMonitors` are a Red Hat OpenShift specific feature that is enabled by default. This feature must be disabled explicitly on Amazon EKS

     ```
     spec:
       queueManager:
         metrics:
           enabled: true
           serviceMonitor:
             enabled: false
     ```

  2. Deploy Prometheus then manually configure the scrape target in Prometheus.
  3. View metrics. Queue manager metrics are served on port 9157. You can view the metrics by querying this endpoint from within the queue manager pod. For example, run the following command, where <POD> is the name of your queue manager pod:

     ```
     kubectl exec -t <POD> -- curl http://0.0.0.0:9157/metrics
     ```

- Serve queue manager metrics through HTTPS

  Red Hat OpenShift

  Set **`.spec.queueManager.metrics.tls.provider`** to `openshift`.

  ```
  spec:
    queueManager:
      metrics:
        enabled: true
        tls:
          provider: openshift
  ```

  Note: Using this setting creates a Red Hat OpenShift certificate that is used by the HTTPS metrics server of a queue manager. No other providers can be configured for use by the IBM MQ Operator.

Amazon EKS (and optionally, but not recommended, Red Hat OpenShift)
Provide your own certificates to serve queue manager metrics through HTTPS.

If you are using Red Hat OpenShift, you can choose this option but it is not recommended because it requires disabling the `ServiceMonitor` and losing the benefits of that feature.

Set the following values in the queue manager YAML:

- `.spec.queueManager.metrics.tls.provider`: `manual`
- `.spec.queueManager.metrics.tls.secretName`: the name of a Kubernetes TLS secret that contains your own certificates. For example, you can follow the instructions to create a pair of certificates and store them in a `secret` as described in "Creating a self-signed PKI using OpenSSL" on page 107.
- `.spec.queueManager.metrics.serviceMonitor.enabled`: `false`

```
spec:
  queueManager:
    metrics:
      enabled: true
      tls:
        provider: manual
        secretName: <my-tls-secret>
      serviceMonitor:
        enabled: false
```

You can view the metrics by querying the metrics endpoint from within the queue manager pod. For example, when using self-signed certificates, run the following command, where <POD> is the name of your queue manager pod:

```
kubectl exec -t <POD> -- curl https://0.0.0.0:9157/metrics --insecure
```

- Disable queue manager metrics.

  Set **`.spec.queueManager.metrics.enabled`** to `false`.

Related tasks
"Troubleshooting container clusters with the OpenShift Monitoring CLI" on page 278
As a troubleshooting aid, you can query the available container cluster metrics using the OpenShift Monitoring CLI.

OpenShift CP4I *Monitoring container clusters with the OpenShift Monitoring console*

You can query the available container cluster metrics on deployments on Red Hat OpenShift by using the Red Hat OpenShift Monitoring web console.

Before you begin

OpenShift Monitoring provides a web console and a CLI. For day-to-day operations you might prefer to use the console, as described in this task. For gathering troubleshooting information you might prefer to use the CLI, as described in "Troubleshooting container clusters with the OpenShift Monitoring CLI" on page 278.

About this task

OpenShift Container Platform includes a pre-configured, pre-installed and self-updating monitoring stack that uses an open source Prometheus server. Queue managers running under an IBM MQ Operator can produce metrics compatible with Prometheus. For a complete list of the metrics available for IBM MQ, see "Metrics published by the IBM MQ container" on page 248.

OpenShift Monitoring is the default cluster monitoring method. It is installed with the cluster. It provides Prometheus, Thanos Querier, and a Grafana dashboard that shows queue manager metrics.

To query the available container cluster metrics using the OpenShift Monitoring console, complete the following steps.

Procedure

1. Enable OpenShift Monitoring for the operands deployed in an IBM MQ container cluster.
   a) Log in to the cluster as a cluster administrator.
   b) Add "`enableUserWorkload: true`" to the configMap `cluster-monitoring-config` in the `openshift-monitoring` namespace. To do this, follow the steps in Enabling monitoring for user-defined projects in the Red Hat OpenShift documentation. If the `cluster-monitoring-config` ConfigMap does not exist in the `openshift-monitoring` namespace, follow the instructions to create it.
2. Query the IBM MQ metrics using the OpenShift Monitoring console.

   Use the static Grafana dashboard, which presents cluster metrics to cluster administrators.

   Query IBM MQ metrics, and examine the data plotted on a chart, as described in Accessing metrics in the Red Hat OpenShift documentation.

   For example, if you query the metric **`ibmmq_qmgr_cpu_load_fifteen_minute_average_percentage`** you get a chart similar to the following screen capture:

   This screen capture also shows the web console prompt. You can run PromQL queries from this prompt. For example:

   Get a list of queue managers that have a positive FDC count:

```
ibmmq_qmgr_fdc_files > 0
```

   Pass the `namespace` filter in the query to get the results for a specific namespace:

```
ibmmq_qmgr_fdc_files{namespace='your-namespace'} > 0
```

   Return the count of all such queue managers:

```
count(ibmmq_qmgr_fdc_files{namespace='your-namespace'} > 0)
```

   Get the Write latency of queue manager pods over the last 5 hours:

```
ibmmq_qmgr_log_write_latency_seconds{namespace='your-namespace'}[5h]
```

Get the RAM usage estimate in bytes for a queue manager for a particular namespace summed by "Job":

```
sum by (job)(ibmmq_qmgr_ram_usage_estimate_for_queue_manager_bytes{qmgr='your-queue-manager', namespace='your-namespace'})
```

The previous queries illustrate use of filters "namespace", "job" and "qmgr". For this metric you can also filter on "container", "endpoint", "instance", "pod", and "prometheus". Note that the available filters are the column headings shown in the screen capture, except for “Name” which is the name of the metric itself.

Related tasks

“Troubleshooting container clusters with the OpenShift Monitoring CLI” on page 278
As a troubleshooting aid, you can query the available container cluster metrics using the OpenShift Monitoring CLI.

Thanos Querier Versus Thanos Querier

Related reference

“Metrics published by the IBM MQ container” on page 248
Queue manager containers can publish metrics compatible with Red Hat OpenShift Monitoring.

OpenShift CP4I *Metrics published by the IBM MQ container*

Queue manager containers can publish metrics compatible with Red Hat OpenShift Monitoring.

Metric	Type	Description
ibmmq_nha_recovery_average_network_round_trip_time_seconds	gauge	Average network round trip time
ibmmq_nha_recovery_backlog_average_bytes	gauge	Backlog average bytes
ibmmq_nha_recovery_backlog_bytes	gauge	Backlog bytes
ibmmq_nha_recovery_compressed_log_sent_bytes	gauge	Compressed log bytes sent
ibmmq_nha_recovery_log_data_average_compression_time_seconds	gauge	Log data average compression time
ibmmq_nha_recovery_log_data_average_decompression_time_seconds	gauge	Log data average decompression time
ibmmq_nha_recovery_log_decompressed_bytes	gauge	Log bytes decompressed
ibmmq_nha_recovery_log_sent_bytes	counter	Log bytes sent
ibmmq_nha_recovery_rebase_count	gauge	Rebase count

Metric	Type	Description
ibmmq_nha_recovery_recovery_log_sequence_number	gauge	Recovery log sequence number
ibmmq_nha_replication_acknowledged_log_sequence_number_total	gauge	Acknowledged log sequence number
ibmmq_nha_replication_average_network_round_trip_time_seconds	gauge	Average network round trip time
ibmmq_nha_replication_backlog_average_bytes	gauge	Backlog average bytes
ibmmq_nha_replication_backlog_bytes	gauge	Backlog bytes
ibmmq_nha_replication_catch_up_compressed_log_sent_bytes	counter	Catch-up compressed log bytes sent
ibmmq_nha_replication_catch_up_log_data_average_compression_time_seconds	gauge	Catch-up log data average compression time
ibmmq_nha_replication_catch_up_log_data_average_decompression_time_seconds	gauge	Catch-up log data average decompression time
ibmmq_nha_replication_catch_up_log_decompressed_bytes	gauge	Catch-up log bytes decompressed
ibmmq_nha_replication_catch_up_uncompressed_log_sent_bytes	counter	Catch-up uncompressed log bytes sent
ibmmq_nha_replication_catchup_log_sent_bytes	counter	Catch-up log bytes sent
ibmmq_nha_replication_log_file_system_free_space_percent	gauge	Log file system - free space
ibmmq_nha_replication_log_file_system_in_use_bytes	gauge	Log file system - bytes in use

Metric	Type	Description
ibmmq_nha_replication_log_write_average_acknowledgement_latency_seconds	gauge	Log write average acknowledgement latency
ibmmq_nha_replication_log_write_average_acknowledgement_size_bytes	gauge	Log write average acknowledgement size
ibmmq_nha_replication_mq_fdc_file_count	gauge	MQ FDC file count
ibmmq_nha_replication_queue_manager_file_system_free_space_percent	gauge	Queue Manager file system - free space
ibmmq_nha_replication_queue_manager_file_system_in_use_bytes	gauge	Queue Manager file system - bytes in use
ibmmq_nha_replication_synchronous_compressed_log_sent_bytes	counter	Synchronous compressed log bytes sent
ibmmq_nha_replication_synchronous_log_data_average_compression_time_seconds	gauge	Synchronous log data average compression time
ibmmq_nha_replication_synchronous_log_data_average_decompression_time_seconds	gauge	Synchronous log data average decompression time
ibmmq_nha_replication_synchronous_log_decompressed_bytes	gauge	Synchronous log bytes decompressed
ibmmq_nha_replication_synchronous_log_sent_bytes	counter	Synchronous log bytes sent
ibmmq_nha_replication_synchronous_uncompressed_log_sent_bytes	counter	Synchronous uncompressed log bytes sent
ibmmq_qmgr_commit_total	counter	Commit count

Metric	Type	Description
ibmmq_qmgr_cpu_load_fifteen_minute_average_percentage	gauge	CPU load - fifteen minute average
ibmmq_qmgr_cpu_load_five_minute_average_percentage	gauge	CPU load - five minute average
ibmmq_qmgr_cpu_load_one_minute_average_percentage	gauge	CPU load - one minute average
ibmmq_qmgr_destructive_get_bytes_total	counter	Interval total destructive get - byte count
ibmmq_qmgr_destructive_get_total	counter	Interval total destructive get- count
ibmmq_qmgr_durable_subscription_alter_total	counter	Alter durable subscription count
ibmmq_qmgr_durable_subscription_create_total	counter	Create durable subscription count
ibmmq_qmgr_durable_subscription_delete_total	counter	Delete durable subscription count
ibmmq_qmgr_durable_subscription_resume_total	counter	Resume durable subscription count
ibmmq_qmgr_errors_file_system_free_space_percentage	gauge	MQ errors file system - free space
ibmmq_qmgr_errors_file_system_in_use_bytes	gauge	MQ errors file system - bytes in use
ibmmq_qmgr_expired_message_total	counter	Expired message count
ibmmq_qmgr_failed_browse_total	counter	Failed browse count
ibmmq_qmgr_failed_mqcb_total	counter	Failed MQCB count
ibmmq_qmgr_failed_mqclose_total	counter	Failed MQCLOSE count
ibmmq_qmgr_failed_mqconn_mqconnx_total	counter	Failed MQCONN/MQCONNX count
ibmmq_qmgr_failed_mqget_total	counter	Failed MQGET - count

Metric	Type	Description
ibmmq_qmgr_failed_mqinq_total	counter	Failed MQINQ count
ibmmq_qmgr_failed_mqopen_total	counter	Failed MQOPEN count
ibmmq_qmgr_failed_mqput1_total	counter	Failed MQPUT1 count
ibmmq_qmgr_failed_mqput_total	counter	Failed MQPUT count
ibmmq_qmgr_failed_mqset_total	counter	Failed MQSET count
ibmmq_qmgr_failed_mqsubrq_total	counter	Failed MQSUBRQ count
ibmmq_qmgr_failed_subscription_create_alter_resume_total	counter	Failed create/alter/resume subscription count
ibmmq_qmgr_failed_subscription_delete_total	counter	Subscription delete failure count
ibmmq_qmgr_failed_topic_mqput_mqput1_total	counter	Failed topic MQPUT/MQPUT1 count
ibmmq_qmgr_fdc_files	gauge	MQ FDC file count
ibmmq_qmgr_log_file_system_free_space_percentage	gauge	Log file system - free space
ibmmq_qmgr_log_file_system_in_use_bytes	gauge	Log file system - bytes in use
ibmmq_qmgr_log_file_system_max_bytes	gauge	Log file system - bytes max
ibmmq_qmgr_log_in_use_bytes	gauge	Log - bytes in use
ibmmq_qmgr_log_logical_written_bytes_total	counter	Log - logical bytes written
ibmmq_qmgr_log_max_bytes	gauge	Log - bytes max
ibmmq_qmgr_log_occupied_by_extents_waiting_to_be_archived_bytes	gauge	Log - occupied by extents waiting to be archived

Metric	Type	Description
ibmmq_qmgr_log_occupied_by_reusable_extents_bytes	gauge	Log - bytes occupied by reusable extents
ibmmq_qmgr_log_physical_written_bytes_total	counter	Log - physical bytes written
ibmmq_qmgr_log_primary_space_in_use_percentage	gauge	Log - current primary space in use
ibmmq_qmgr_log_required_for_media_recovery_bytes	gauge	Log - bytes required for media recovery
ibmmq_qmgr_log_sequence_number_disk_total	gauge	Log - disk written log sequence number
ibmmq_qmgr_log_sequence_number_quorum_total	gauge	Log - quorum log sequence number
ibmmq_qmgr_log_slowest_write_since_restart_seconds	gauge	Log - slowest write since restart
ibmmq_qmgr_log_workload_primary_space_utilization_percentage	gauge	Log - workload primary space utilization
ibmmq_qmgr_log_write_latency_seconds	gauge	Log - write latency
ibmmq_qmgr_log_write_size_bytes	gauge	Log - write size
ibmmq_qmgr_mqcb_total	counter	MQCB count
ibmmq_qmgr_mqclose_total	counter	MQCLOSE count
ibmmq_qmgr_mqconn_mqconnx_total	counter	MQCONN/MQCONNX count
ibmmq_qmgr_mqctl_total	counter	MQCTL count
ibmmq_qmgr_mqdisc_total	counter	MQDISC count
ibmmq_qmgr_mqinq_total	counter	MQINQ count
ibmmq_qmgr_mqopen_total	counter	MQOPEN count
ibmmq_qmgr_mqput_mqput1_bytes_total	counter	Interval total MQPUT/MQPUT1 byte count

Metric	Type	Description
ibmmq_qmgr_mqput_mqput1_total	counter	Interval total MQPUT/MQPUT1 count
ibmmq_qmgr_mqset_total	counter	MQSET count
ibmmq_qmgr_mqstat_total	counter	MQSTAT count
ibmmq_qmgr_mqsubrq_total	counter	MQSUBRQ count
ibmmq_qmgr_non_durable_subscription_create_total	counter	Create non-durable subscription count
ibmmq_qmgr_non_durable_subscription_delete_total	counter	Delete non-durable subscription count
ibmmq_qmgr_non_persistent_message_browse_bytes_total	counter	Non-persistent message browse - byte count
ibmmq_qmgr_non_persistent_message_browse_total	counter	Non-persistent message browse - count
ibmmq_qmgr_non_persistent_message_destructive_get_total	counter	Non-persistent message destructive get - count
ibmmq_qmgr_non_persistent_message_get_bytes_total	counter	Got non-persistent messages - byte count
ibmmq_qmgr_non_persistent_message_mqput1_total	counter	Non-persistent message MQPUT1 count
ibmmq_qmgr_non_persistent_message_mqput_total	counter	Non-persistent message MQPUT count
ibmmq_qmgr_non_persistent_message_put_bytes_total	counter	Put non-persistent messages - byte count
ibmmq_qmgr_non_persistent_topic_mqput_mqput1_total	counter	Non-persistent - topic MQPUT/MQPUT1 count
ibmmq_qmgr_persistent_message_browse_bytes_total	counter	Persistent message browse - byte count
ibmmq_qmgr_persistent_message_browse_total	counter	Persistent message browse - count

Metric	Type	Description
ibmmq_qmgr_persistent_message_destructive_get_total	counter	Persistent message destructive get - count
ibmmq_qmgr_persistent_message_get_bytes_total	counter	Got persistent messages - byte count
ibmmq_qmgr_persistent_message_mqput1_total	counter	Persistent message MQPUT1 count
ibmmq_qmgr_persistent_message_mqput_total	counter	Persistent message MQPUT count
ibmmq_qmgr_persistent_message_put_bytes_total	counter	Put persistent messages - byte count
ibmmq_qmgr_persistent_topic_mqput_mqput1_total	counter	Persistent - topic MQPUT/MQPUT1 count
ibmmq_qmgr_published_to_subscribers_bytes_total	counter	Published to subscribers - byte count
ibmmq_qmgr_published_to_subscribers_message_total	counter	Published to subscribers - message count
ibmmq_qmgr_purged_queue_total	counter	Purged queue count
ibmmq_qmgr_queue_manager_file_system_free_space_percentage	gauge	Queue Manager file system - free space
ibmmq_qmgr_queue_manager_file_system_in_use_bytes	gauge	Queue Manager file system - bytes in use
ibmmq_qmgr_ram_free_percentage	gauge	RAM free percentage
ibmmq_qmgr_ram_usage_estimate_for_queue_manager_bytes	gauge	RAM total bytes - estimate for queue manager
ibmmq_qmgr_rollback_total	counter	Rollback count
ibmmq_qmgr_system_cpu_time_estimate_for_queue_manager_percentage	gauge	System CPU time - percentage estimate for queue manager
ibmmq_qmgr_system_cpu_time_percentage	gauge	System CPU time percentage

Metric	Type	Description
ibmmq_qmgr_topic_mqput_mqput1_total	counter	Topic MQPUT/MQPUT1 interval total
ibmmq_qmgr_topic_put_bytes_total	counter	Interval total topic bytes put
ibmmq_qmgr_trace_file_system_free_space_percentage	gauge	MQ trace file system - free space
ibmmq_qmgr_trace_file_system_in_use_bytes	gauge	MQ trace file system - bytes in use
ibmmq_qmgr_user_cpu_time_estimate_for_queue_manager_percentage	gauge	User CPU time - percentage estimate for queue manager
ibmmq_qmgr_user_cpu_time_percentage	gauge	User CPU time percentage

Related information

Metrics published on the system topics

Stopping a queue manager (mq.ibm.com/stop)

Stop a queue manager by adding an annotation to the queue manager definition.

About this task

Queue managers created by the IBM MQ Operator have an associated StatefulSet. This StatefulSet declares the number of Pods to be deployed for a given queue manager availability type through the '.replicas' field. This takes the value of 1 (Single Instance), 2 (Multi Instance) or 3 (NativeHA).

Note: Manually changing the value in the '.replicas' field prevents the queue manager from functioning correctly.

In some cases you might want to stop your queue manager so that the StatefulSet has a replica count of 0 and no Pods are deployed. Examples of when you might want to do this include during maintenance or a backup procedure.

Note: Because there are no queue manager Pods deployed when the queue manager is stopped, you and your applications will not be able to access the queue manager until it is started again.

Procedure

- To stop your queue manager, add the following annotation to the queue manager definition under the '.metadata.annotations' section.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: my-qm
  annotations:
    "mq.ibm.com/stop" : "true"
```

- To restart the queue manager and return it to its correct number of replicas, remove the annotation from the queue manager or set its value to 'false'.

OpenShift CP4I Backing up and restoring queue manager configuration

Backing up queue manager configuration can help you to rebuild a queue manager from its definitions if the queue manager configuration is lost. This procedure does not back up queue manager log data. Because of the transient nature of messages, historical log data is likely to be irrelevant at the time of restore.

Before you begin

Log into your cluster from the command line.

Procedure

- Back up queue manager configuration.

  You can use the **dmpmqcfg** command to dump the configuration of an IBM MQ queue manager.

  a) Get the name of the pod for your queue manager.
  For example, run the following command, where <QUEUEMANAGER_NAME> is the name of your `QueueManager` resource:

  – On Red Hat OpenShift:

  ```
  oc get pods --selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/
  instance=<QUEUEMANAGER_NAME>
  ```

  – On Amazon EKS:

  ```
  kubectl get pods --selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/
  instance=<QUEUEMANAGER_NAME>
  ```

  b) Run the **dmpmqcfg** command on the pod, directing the output into a file on your local machine.
  dmpmqcfg outputs the queue manager's MQSC configuration.

  Run the following command, where <POD_NAME> is the name of your pod, which you obtained in the previous step:

  – On Red Hat OpenShift:

  ```
  oc exec -it <POD_NAME> -- dmpmqcfg > backup.mqsc
  ```

  – On Amazon EKS:

  ```
  kubectl exec -it <POD_NAME> -- dmpmqcfg > backup.mqsc
  ```

- Restore queue manager configuration.

  Having followed the backup procedure outlined in the previous step, you should have a `backup.mqsc` file that contains the queue manager configuration. You can restore the configuration by applying this file to a new queue manager.

  a) Get the name of the pod for your queue manager.
  For example, run the following command, where <QUEUEMANAGER_NAME> is the name of your `QueueManager` resource:

  – On Red Hat OpenShift:

  ```
  oc get pods --selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/
  instance=<QUEUEMANAGER_NAME>
  ```

  – On Amazon EKS:

  ```
  kubectl get pods --selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/
  instance=<QUEUEMANAGER_NAME>
  ```

b) Run the **runmqsc** command on the pod, directing in the content of the `backup.mqsc` file.

Run the following command, where <POD_NAME> is the name of your pod, which you obtained in the previous step:

– On Red Hat OpenShift:

```
oc exec -i <POD_NAME> -- runmqsc < backup.mqsc
```

– On Amazon EKS:

```
kubectl exec -i <POD_NAME> -- runmqsc < backup.mqsc
```

MQ Adv. Viewing the status of Native HA queue managers

For custom-built containers, you can view the status of the Native HA instances by using the **dspmq** command.

About this task

You can use the **dspmq** command in one of the running Pods to view the operational status of a queue manager instance. The information returned depends on whether the instance is active or a replica. The information supplied by the active instance is definitive, information from replica instances might be out of date.

For help when a node failure occurs, see Recovering from a node failure on Native HA.

V 10.0.0 For a Native HA Cross-Region Replication (CRR) configuration, the information supplied also depends on whether the instance is part of a Live or a Recovery group. The information from the Live group is more definitive, but can still be out of date as the Recovery group sends back only periodic information.

You can perform the following actions:

- View whether the queue manager instance on the current node is active or a replica.
- View the Native HA operational status of the instance on the current node.
- View the operational status of all three instances in a Native HA configuration.
- V 10.0.0 View the status of the groups that the instance belongs to or is connected to.

The following status fields are used to report Native HA and Native HA Cross-Region Replication (CRR) configuration status of a queue manager instance. These fields always appear:

ROLE
Specifies the current role of the instance. Is one of `Active`, `Replica`, Unknown, (for a recovery group) `Leader`, or `Not Configured`.

INSTANCE
The name provided for this instance of the queue manager when it was created using the **-lr** option of the **crtmqm** command.

INSYNC
Indicates whether the instance is able to take over as the active instance if required.

QUORUM
Reports the quorum status in the form *number_of_instances_in-sync*/*number_of_instances_configured*.

V 10.0.0 **GRPLSN**
Specifies the log sequence number (LSN) that has been replicated between a quorum of instances in the HA group, in *nnnnn:nnnnn:nnnnn:nnnnn* format. Blank if the LSN is not known.

V 10.0.0 **GRPNAME**
Specifies the name of the HA group. Blank if a name has not been configured.

V 10.0.0 **GRPROLE**
Specifies the current role of the HA group and is one of:

- `Live` - A live group

- `Recovery` - A recovery group
- `Pending live` - A group waiting to become `Live`
- `Pending recovery` - A group waiting to become `Recovery`
- `Not configured` - Native HA has not been configured
- `Unknown` - The current role cannot be determined

These fields appear when extended information is requested for all instances by using the `-x` parameter:

REPLADDR
The replication address of the queue manager instance.

CONNACTV
Indicates whether the instance is connected to the active instance.

BACKLOG
Indicates the number of KB that the instance is behind the active instance.

CONNINST
Indicates whether the named instance is connected to this instance.

ALTDATE
Indicates the date when this information was last updated. Blank if it has never been updated.

ALTTIME
Indicates the time when this information was last updated. Blank if it has never been updated.

V 10.0.0 **ACKLSN**
Indicates the LSN that the instance has acknowledged as having been written to the recovery log, in *nnnnn:nnnnn:nnnnn:nnnnn* format. Blank if the LSN value is not known.

V 10.0.0 **HASTATUS**

Indicates the operational status of this instance and is one of the following values:

- `Normal` - the instance is operating normally.
- `Checking` - the instance is being checked to ensure the consistency of the recovery log.
- `Synchronizing` - the instance is being sent data to bring it in-sync.
- `Rebasing` - a new recovery log is being sent to the instance.
- `Disk full` - the instance has a full disk.
- `Disconnected` - cannot communicate with the instance.
- `Unknown` - the instance status is unknown.

V 10.0.0 **SYNCTIME**
Indicates the time when this instance was last in-sync with the active queue manager, in ISO 8601 format. Blank if the time is not known.

V 10.0.0 These fields appear when group information is requested by using the `-g` parameter. These fields are useful to view the status when the group is part of a Native HA CRR configuration:

V 10.0.0 **GRPNAME**
Specifies the name of the HA group. Blank if a name has not been configured.

V 10.0.0 **GRPROLE**
Specifies the current role of the HA group and is one of:

- `Live` - A live group
- `Recovery` - A recovery group
- `Pending live` - A group waiting to become `Live`
- `Pending recovery` - A group waiting to become `Recovery`
- `Unknown` - The current role cannot be determined
- `Not configured` - Native HA has not been configured

GRPADDR
The IP address that is used to connect to the group. This is set to Unknown if the address is not known because a connection to the group has not occurred.

GRPVER
Indicates the version of the current group leader in V.R.M.F format, where V.R.M.F is the Version, Release, Modification, and Fix Pack number. If the value is not known, this is set to ?.?.?.?

CONNGRP
Indicates whether the group is connected to the local group and is one of:

- yes - the group is connected.
- no - the group is disconnected.
- unknown - the connection status is unknown.
- suspended - the group is connected, but the group cannot replicate data until a configuration incompatibility between the groups is resolved.

This field is displayed only for groups other than the local group.

INSYNC
This is only displayed for a group with role Recovery or Pending live. It indicates whether the group could become Live with no loss of data if a failover occurred. (Note that a Recovery group might not be in-sync with the Live group very often because the log is replicated asynchronously. Use the RCOVLSN and RCOVTIME values to indicate if data loss might occur in an unplanned failover).

SYNCTIME
This is only displayed for a group with role Recovery or Pending live. It indicates the time when this group was last in-sync with the Live group, in ISO 8601 format (blank if it is not known). (Note that a Recovery group might not be in-sync with the Live group very often because the log is replicated asynchronously. Use the RCOVLSN and RCOVTIME values to indicate if data loss might occur in an unplanned failover).

BACKLOG
This is only displayed for a group with role Recovery or Pending live. It indicates the number of KB that the group is behind the live group.

GRSTATUS
Indicates the operational status of this group and is one of the following values:

- Normal - the group is operating normally.
- Checking - the group is being checked to ensure the consistency of the recovery log.
- Synchronizing - the group is being sent data to bring it in-sync.
- Rebasing - a new recovery log is being sent to the instance.
- Waiting for connection - the group is waiting for a connection from a group with the role Live or Pending Recovery.
- Partitioned - two groups believe they are the Live group (that is, a 'split brain' status).
- Unknown - the group status is unknown.

RCOVLSN
A log sequence number (LSN) that the group could recover to, in *nnnnn:nnnnn:nnnnn:nnnnn* format. Any data written to the recovery log after this point might be lost if an unplanned failover occurs. This value can be passed into the -s option of the **dmpmqlog** command.

RCOVTIME
A time that the group could recover to, in ISO 8601 format. Any data written to the recovery log after this time might be lost if an unplanned failover occurs. This value can be passed into the -t or -u options of the **dmpmqlog** command.

INITLSN
This is only displayed for a group with role Live or Pending recovery. It indicates the log sequence number (LSN) of the last log record recovered when the Native HA group initially became live, in the format *nnnnn:nnnnn:nnnnn:nnnnn*, or blank if the group is not live.

INITTIME
This is only displayed for a group with role `Live` or `Pending recovery`. It indicates the time of the last log record recovered when the Native HA group initially became live, in ISO 8601 format. Blank, if the group is not live.

LIVETIME
This is only displayed for a group with role `Live` or `Pending recovery`. It indicates the time the group first became the live group, in ISO 8601 format. Blank, if the group is not live.

ALTDATE
Indicates the date when this information was last updated.

ALTTIME
Indicates the time when this information was last updated.

Procedure

- Find the pods that are part of your queue manager.
  - On Red Hat OpenShift:

    ```
    oc get pod --selector app.kubernetes.io/instance=nativeha-qm
    ```

  - On Amazon EKS:

    ```
    kubectl get pod --selector app.kubernetes.io/instance=nativeha-qm
    ```

- Run the `dspmq` in one of the pods
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq
    ```

    For an interactive shell, where you can run `dspmq` directly.

    ```
    oc rsh <POD>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq
    ```

    For an interactive shell, where you can run `dspmq` directly.

    ```
    kubectl exec -it <POD> -- sh
    ```

- To determine whether a queue manager instance is running as the active instance or as a replica:
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o status -m <QMGR_NAME>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o status -m <QMGR_NAME>
    ```

  An active instance of a queue manager named BOB would report the following status:

  ```
  QMNAME(BOB)                STATUS(Running)
  ```

  An instance of BOB that is a recovery group leader in a Native HA CRR configuration would report the following status:

  ```
  QMNAME(BOB)                STATUS(Recovery group leader)
  ```

  A replica instance of a queue manager named BOB would report the following status:

```
QMNAME(BOB)                 STATUS(Replica)
```

An inactive instance would report the following status:

```
QMNAME(BOB)                 STATUS(Ended Immediately)
```

- To determine Native HA operational status of the instance in the specified pod:
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o nativeha -m <QMGR_NAME>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o nativeha -m <QMGR_NAME>
    ```

  The active instance of a queue manager named BOB might report the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(Yes) QUORUM(3/3) V 10.0.0
  GRPLSN(<0:25:365:1000>) GRPNAME() GRPROLE(Live)
  ```

  A replica instance of a queue manager named BOB might report the following status:

  ```
  QMNAME(BOB)         ROLE(Replica) INSTANCE(inst2) INSYNC(Yes) QUORUM(2/3) V 10.0.0
  GRPLSN(<0:25:365:1000>) GRPNAME() GRPROLE(Live)
  ```

  An inactive instance of a queue manager named BOB might report the following status:

  ```
  QMNAME(BOB)         ROLE(Unknown) INSTANCE(inst3) INSYNC(no) QUORUM(0/3) V 10.0.0GRPLSN()
  GRPNAME() GRPROLE(Live)
  ```

  V 10.0.0An active instance of a queue manager named BOB belonging to the Live group in a Native HA CRR configuration might report the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(yes) QUORUM(3/3)
  GRPLSN(<0:25:365:1000>) GRPNAME(alpha) GRPROLE(Live)
  ```

  V 10.0.0The same instance when the group was waiting to become the recovery group might report the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(yes) QUORUM(3/3)
  GRPLSN(<0:25:365:1000>) GRPNAME(alpha) GRPROLE(Pending recovery)
  ```

  V 10.0.0A queue manager named CAROL that is not an instance of a Native HA or Native HA CRR configuration would report the following status:

  ```
  QMNAME(CAROL)       ROLE(Not configured) INSTANCE() INSYNC() QUORUM() GRPLSN() GRPNAME()
  GRPROLE(Not configured)
  ```

- To determine the Native HA operational status of all the instances in the Native HA configuration:
  - On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o nativeha -x -m <QMGR_NAME>
    ```

  - On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o nativeha -x -m <QMGR_NAME>
    ```

  For example, if you issue this command on the node running the active instance of queue manager BOB, where all instances are in-sync, you might receive the following status:

  ```
  QMNAME(BOB)         ROLE(Active) INSTANCE(inst1) INSYNC(yes) QUORUM(2/3)
  GRPLSN(<0:25:365:1000>) GRPROLE(Live) GRPNAME()
   INSTANCE(inst1) ROLE(Active)  REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
  BACKLOG(0)  CONNINST(yes) ACKLSN(<0:26:365:1000>)  HASTATUS(Normal)
  SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
  ```

```
 INSTANCE(inst3) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN (<0:24:365:1000>) HASTATUS(Normal)
SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
 INSTANCE(inst2) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN (<0:25:365:1000>) HASTATUS(Synchronizing)
SYNCTIME(2023-12-20T17:13:22.123321Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

In another example, if you issue this command on a node running a replica instance of queue manager BOB, which is lagging behind, you might receive the following status:

```
QMNAME(BOB)         ROLE(Replica) INSTANCE(inst2) INSYNC(no) QUORUM(2/3)
GRPLSN(<0:25:365:1000>) GRPROLE(Live) GRPNAME()
 INSTANCE(inst1) ROLE(Active)  REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN(<0:26:365:1000>)  HASTATUS(Normal)
SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
 INSTANCE(inst3) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(yes)
BACKLOG(0)  CONNINST(yes) ACKLSN (<0:24:365:1000>) HASTATUS(Normal)
SYNCTIME(2023-12-20T17:13:27.853412Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
 INSTANCE(inst2) ROLE(Replica) REPLADDR(localhost) CONNACTV(yes) INSYNC(no)
BACKLOG(435) CONNINST(yes) ACKLSN (<0:25:365:1000>) HASTATUS(Synchronizing)
SYNCTIME(2023-12-20T17:13:22.123321Z) ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

If you issue this command on a node running an inactive instance of queue manager BOB, you might receive the following status:

```
QMNAME(BOB)            ROLE(Unknown) INSTANCE(inst3) INSYNC(no) QUORUM(0/3) GRPLSN()
GRPROLE() GRPNAME()
 INSTANCE(inst1) ROLE(Unknown) REPLADDR(9.20.123.45) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ACKLSN()  HASTATUS(Unknown) SYNCTIME() ALTDATE() ALTTIME()
ALTDATE() ALTTIME()
 INSTANCE(inst2) ROLE(Unknown) REPLADDR(9.20.123.46) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ACKLSN()  HASTATUS(Unknown) SYNCTIME() ALTDATE() ALTTIME()
ALTDATE() ALTTIME()
 INSTANCE(inst3) ROLE(Unknown) REPLADDR(9.20.123.47) CONNACTV(No)      INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ACKLSN()  HASTATUS(Unknown) SYNCTIME() ALTDATE() ALTTIME()
ALTDATE() ALTTIME()
```

If you issue the command when the instances are still negotiating which is active and which are replicas, you would receive the following status:

```
QMNAME(BOB)             STATUS(Negotiating)
```

- V 10.0.0

  To determine the operational status of all the Native HA groups in a Native HA CRR configuration:

  – On Red Hat OpenShift:

    ```
    oc exec -t <POD> -- dspmq -o nativeha -g -m <QMGR_NAME>
    ```

  – On Amazon EKS:

    ```
    kubectl exec -t <POD> -- dspmq -o nativeha -g -m <QMGR_NAME>
    ```

  If you issue this command on a member of the Live group, named alpha, you might receive the following status for alpha, and the Recovery group, beta:

```
GRPNAME(alpha) GRPROLE(Live) GRPADDR(9.20.135.129) GRPVER(09040200)
GRSTATUS(Normal) RCOVLSN(<0:0:104:65163>) RCOVTIME(2023-12-20T17:37:27.114787Z)
INITLSN(<0:0:101:352>) INITTIME(2023-12-19T09:13:27.512294)
LIVETIME(2023-12-19T06:14:27.512294) ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

```
GRPNAME(beta) GRPROLE(Recovery) GRPADDR(9.20.135.130) GRPVER(09040300)
CONNGRP(yes) GRSTATUS(Synchronizing) RCOVLSN(<0:0:19:16723>)
RCOVTIME(2023-12-20T17:36:23.463209Z) BACKLOG(5619) INSYNC(no) SYNCTIME()
ALTDATE(2023-12-20) ALTTIME(17.13.27)
```

Related reference

dspmq (display queue managers) command

Native HA CRR switchover and failover

A Native HA Cross-Region Replication (CRR) configuration enables you to switch the running of a queue manager to a different region.

You can use a Native HA CRR configuration to provide a disaster recovery solution. If some disaster occurs at your live site, you can perform a failover and continue running the queue manager at the recovery site. In this situation, you cannot guarantee that the recovery logs are synchronized when the failover occurs, so there might be some loss of data. Similarly, when the failed site is restored, you might have two live instances of the queue manager and so might accrue some partitioned (split-brain) data.

You can use a Native HA CRR configuration to enable you to provide continuity while performing maintenance or testing. In this circumstance, the switchover is controlled in a way that no data loss occurs. Similarly, when you switch back there is no partitioned data to resolve.

Complete a planned switchover on a Native HA CRR configuration

You can perform a planned switchover between the live and recovery groups in a Native HA Cross-Region Replication (CRR) configuration.

About this task

You might require to switch from your live group to your recovery group if you want to test your configuration, or need to apply maintenance.

When you perform a planned switchover, the system takes steps to ensure that you do not lose data or encounter a partitioned (split-brain) problem. You request the switchover by changing the group role on each of the groups. For a while both groups enter a pending status while their recovery logs are synchronized. When this is achieved, the groups adopt their new roles, a new leader is elected in the new live group and takes the active role and runs the queue manager.

Procedure

1. Specify that the live group should become the recovery group by setting its `GroupRole` to `Recovery` in the `NativeHALocalInstance` stanza of the `qm.ini` file.
   a) Set the `GroupRole` to `Recovery`.

      If you are using the IBM MQ Operator, change the `GroupRole` by setting the `.spec.queueManager.availability.nativeHAGroups.local.role` field in the QueueManager YAML. For example, to change the role of the Live group to Recovery, set the value of the `.spec.queueManager.availability.nativeHAGroups.local.role` field to `Recovery` for the group that is currently live.

   b) To verify the switchover was successful, get the pod states by using the relevant command.

      - On Red Hat OpenShift:

        ```
        oc get pods
        ```

      - On Amazon EKS:

        ```
        kubectl get pods
        ```

      One pod should be in ready state:

      ```
      NAME                 READY   STATUS    RESTARTS   AGE
      exampleqm-ibm-mq-0   1/1     Running   0          2m45s
      exampleqm-ibm-mq-1   0/1     Running   0          2m45s
      exampleqm-ibm-mq-2   0/1     Running   0          2m45s
      ```

   c) Check the status of the recovery group by issuing the following command, replacing <LEADER_POD> with the name of the `READY 1/1` pod (`exampleqm-ibm-mq-0` in the example):

- On Red Hat OpenShift:

```
oc exec -t <LEADER_POD> -- dspmq -o nativeha -g
```

- On Amazon EKS:

```
kubectl exec -t <LEADER_POD> -- dspmq -o nativeha -g
```

Check the GRPROLE is `Pending recovery`.

2. Specify that the recovery group should become the live group by setting its `GroupRole` to `Live` in the `NativeHALocalInstance` stanza of the `qm.ini` file.
   a) Set the `GroupRole` to `Live`.

   If you are using the IBM MQ Operator, change the `GroupRole` by setting the `.spec.queueManager.availability.nativeHAGroups.local.role` field in the QueueManager YAML. For example, to change the role of the Live group to Live, set the value of the `.spec.queueManager.availability.nativeHAGroups.local.role` field to `Live` for the group that is currently `Recovery`.

   If you issue a `dspmq -o nativeha -g` command for this queue manager after each instance in the group has restarted, you will see a GRPROLE of `Pending live`.

   b) After each instance in the group has restarted, get the pod states by using the relevant command.

   - On Red Hat OpenShift:

```
oc get pods
```

   - On Amazon EKS:

```
kubectl get pods
```

   One pod should be in ready state:

```
NAME                 READY   STATUS    RESTARTS   AGE
exampleqm-ibm-mq-0   1/1     Running   0          2m45s
exampleqm-ibm-mq-1   0/1     Running   0          2m45s
exampleqm-ibm-mq-2   0/1     Running   0          2m45s
```

   c) Check the status of the live group by issuing the following command, replacing <ACTIVE_POD> with the name of the `READY 1/1` pod displayed by the previous command (`exampleqm-ibm-mq-0` in the example):

   - On Red Hat OpenShift:

```
oc exec -t <ACTIVE_POD> -- dspmq -o nativeha -g
```

   - On Amazon EKS:

```
kubectl exec -t <ACTIVE_POD> -- dspmq -o nativeha -g
```

   Observe that GRPROLE is `Pending live` followed by `live` after the groups are in sync.

3. When the recovery logs on the live and recovery site are synchronized, the recovery group becomes live and elects a leader to be the active queue manager instance. From here, apps will automatically reconnect to this instance and any workload will resume as normal.

What to do next

When the maintenance is complete, you can switch back to the original live group by repeating the process.

Complete an unplanned failover on a Native HA CRR configuration

You perform an unplanned failover between the live and recovery groups in a Native HA Cross-Region Replication (CRR) configuration when some disaster occurs at the live site and it can no longer operate.

About this task

When the live group goes down, the recovery group cannot detect this as it simply stops receiving replicated log data, which could just mean there is no ongoing workload. An unplanned failover must therefore be manually carried out by changing the recovery group's role to live and notifying it that the previously live group is disabled so that it does not attempt to replicate data to it.

The recovery group might not be in-sync with the live group as log data is replicated asynchronously. If the recovery group is not in-sync with the live group, data is lost when an unplanned failover occurs. Before initiating an unplanned failover, issue the **dspmq -o nativeha -g** command on the recovery group, and use the RCOVLSN and RCOVTIME values to view the log sequence number (LSN) and time that the group can recover to.

Procedure

1. Ensure that applications have disconnected from the live group, and are configured to reconnect to the group that is currently the recovery group.
2. Specify that the recovery group should become the live group by completing the following steps:
   a. Set its `GroupRole` to `Live` in the `NativeHALocalInstance` stanza of the `qm.ini` file. For example, if you are using the MQ operator, change the setting of `.spec.queueManager.availability.nativeHAGroups.local.role` to `Live`.
   b. Set the `Enabled` field to `False` in the `NativeHARecoveryGroup` stanza of the `qm.ini` file. For example, if you are using the MQ operator, change the setting of `.spec.queueManager.availability.nativeHAGroups.remotes.enabled` to `false`.

   If you issue a **dspmq -o nativeha -g** command for this queue manager after each instance in the group has restarted you will see a GRPROLE of `Live`. An election is held among group members and one of them becomes the active instance and handles the queue manager's work.
3. To restart replication after the failover, you can either use the original live group as the recovery group, or create a new recovery group at a new site.
   - If the original live site is restored, before restarting the original live group, change the role of the group to recovery. This ensures that the group does not resume its live role.
   - If the original live site cannot be restored, create a new Native HA recovery group at a new site.

   To configure the recovery group, complete the steps that are described in "Restoring replication after an unplanned failover" on page 267.
4. If the configuration on the original live site is not changed, and the group resumes its live role when it is restored, you will have two active instances of the queue manager. If this is the case, the partitioned state is detected and no more replication occurs between the two groups. You must resolve which data to keep before you restore the original live and recovery roles. See "Resolving a partitioned (split brain) problem" on page 267 for guidance.

Related information

dspmq (display queue managers)

Restoring replication after an unplanned failover

Restore replication from the live group to a recovery group after an unplanned failover.

About this task

After an unplanned failover, if the original live site is restored it can be reconfigured to become the recovery group. To prevent the queue manager from entering a partitioned ("split-brain") state, complete this task before restarting the Native HA group on the original live site.

You can also complete these steps when you create a new recovery group after an unplanned failover, if the original live site cannot be restored.

Procedure

1. Specify that the group should become the recovery group by completing the following steps:
   a) Set its GroupRole to Recovery in the NativeHALocalInstance stanza of the `qm.ini` file. For example, if you are using the MQ operator, change the setting of the `.spec.queueManager.availability.nativeHAGroups.local.role` field in the QueueManager YAML to Recovery.
   b) Set the Enabled field to False in the NativeHARecoveryGroup stanza of the qm.ini file. For example, if you are using the MQ operator, change the setting of the `.spec.queueManager.availability.nativeHAGroups.remotes.enabled` field in the QueueManager YAML to false.

      If you issue a **`dspmq -o nativeha -g`** command on an instance of the group after each instance has restarted you will see a GRPROLE of `Recovery`. An election is held among group members and one of them becomes the recovery group leader.
2. Restart replication from the live group to the recovery group by setting the `Enabled` field to `True` in the NativeHARecoveryGroup stanza of the `qm.ini` file of the live group. For example, if you are using the MQ operator, change the setting of the `.spec.queueManager.availability.nativeHAGroups.remotes.enabled` field in the QueueManager YAML to true.

Resolving a partitioned (split brain) problem

A partitioned, or 'split-brain', condition can occur if a queue manager in a Native HA Cross-Region Replication (CRR) configuration runs in both the HA groups at the same time.

The condition can arise when an unplanned failover occurs. Communication with the Live group is lost and so you make the queue manager the primary on the Recovery group and it starts running. Meanwhile, the queue manager is still running on the original Live group. There is a possibility that the old Live group is still able to update its log and this creates an unresolvable branch in the data maintained by both two groups. You must discard the data from one of the groups before they can be rejoined.

To resolve the partitioned condition, you choose which group data to retain and which to delete. It might be that the groups were not in sync when the unplanned failover was triggered, or it might be that application workload was able to be processed by both groups The definition of "best" data is largely subjective. For example, while one group might have stored more data in its log, this might be largely automatically recorded media images of IBM MQ objects; whereas the other group might have processed more valuable business transactions. Whichever group is chosen, it is likely that, in the recovery from the partitioned condition, the unreplicated message and transaction outcomes on the group to be deleted will be useful when reconciling the possible data loss.

IBM MQ detects a partitioned condition when a group with a Live role connects to a another group that is running with a Live role. If you issue the command `dspmq -o nativeha -g` in both groups, the status `GRSTATUS(Partitioned)` is returned. The replication of log data is suspended between both groups. The `Partitioned` state continues to be reported until the group is able to replicate sufficient log data to another group with a Recovery role.

If it is not an obvious choice which group to retain and, if log extents are still available from both groups at the time of the unplanned failover, you can compare the log contents.

Use the **dspmq -o nativeha -g** command to identify the LSN (log sequence number) of the exact point in the log that a group became Live. The LSN is returned in the form `INITLSN (nnnnn:nnnnn:nnnnn:nnnnn)`. The time of this event is also reported by the `INITTIME` attribute. Then use the **dmpmqlog -s** *startLSN* command to examine the contents of a log from that point. Use **dspmq** and **dmpmqlog** in combination to identify log records that were written by one group and not replicated to the other. The **dmpmqlog** command also reports a summary of the messaging and transaction activity, which can be useful when making the choice.

Example

The groups 'alpha' and 'beta' were configured initially in a Live and Recovery configuration. A decision to convert the 'beta' group from Recovery to Live was made and an unplanned failover occurred. However the 'alpha' group remained able to progress some application workload which was not replicated to 'beta'. Meanwhile, some applications connected to 'beta' and put some messaging workload through. The INITLSN from the 'beta' group indicates the point in the log that the 'beta' group branched away from the content of the 'alpha' group.

A copy of the queue manager data is needed for the comparison process and the queue manager needs to be inactive. This can be achieved in a Kubernetes environment by scaling the queue manager stateful set size to zero or by taking a copy of the data, for example, by way of a persistent volume snapshot.

After the analysis of the log data has been completed and the choice of which group data to discard is made, you can discard it by changing the group role to Recovery and either deleting all of the queue manager persistent volumes, or removing the Recovery group information.

Related information

dspmq

dmpmqlog

OpenShift MQ Adv. Ending Native HA queue managers

For IBM MQ in containers, you can use the **endmqm** command to end an active or a replica queue manager that is part of a Native HA group.

Before you begin

Note: This information applies only to container environments.

About this task

The procedure for stopping a queue manager that is part of a Native HA group depends on whether it is an active or a replica instance. When you end either type of instance, a check is made to ensure that ending the instance does not break the quorum of the Native HA group. If the quorum would be broken, the **endmqm** command fails.

When you issue an **endmqm** command, the other instances in the group are warned that this is happening, so that they do not report errors when the connection breaks.

If an active instance loses quorum because of the ending or disconnection of too many replica instances, the active instance waits for a configurable amount of time before ending completely. This allows a period of time to shut down processing gracefully, rather than applications just having their connections broken. This timeout value can be specified by the `QuorumConnectivityTimeout` attribute in the `NativeHALocalInstance` stanza of the `qm.ini` file. The default value is 0 seconds.

Procedure

- To end the active instance of a queue manager, issue the following command on the node where the active instance is running:

  ```
  endmqm -s QMgrName
  ```

  – Specify the `-r` option to help client applications to reconnect to another instance.
  – If this instance is not the active instance in the Native HA group then the command fails.
  – If ending this active instance would cause the group quorum to fail then the command fails. (If other instances end or become unavailable at the same time as you run this command, the quorum check might not detect this, the Native HA group ends and can only be restarted when enough instances are available.)

  When the active queue manager ends, one of the replica instances takes over the active role. You cannot specify which replica takes over, this is determined by negotiation within the group and depends on which has the most up-to-date transaction logs.
- To end a replica instance of a queue manager, issue the following command:

  ```
  endmqm -x QMgrName
  ```

  – If this instance is the active instance then the command fails.
  – If ending this replica instance would cause the group quorum to fail then the command fails. (If other instances end or become unavailable at the same time as you run this command, the quorum check might not detect this, the Native HA group ends and can only be restarted when enough instances are available.)

  Note: You can also use the -c, -i, -p, or -w switches with the **endmqm** command on Native HA instances, regardless of which role they are in. The queue manager instance ends, ignoring the effect that has on the group quorum. Information is still shared with the other instances in the group, however. You can use these switches together with the -s for the active instance. You cannot use these switches together with the -x switch for replica instances.

Troubleshooting IBM MQ in containers

If you are having problems with running IBM MQ in a container, you can use the techniques that are described here to help you diagnose and solve the problems.

Procedure

- “Troubleshooting unplanned restarts of IBM MQ in containers” on page 269.
- “Collecting troubleshooting information with IBM MQ container inspector” on page 271.
- “Troubleshooting problems with the IBM MQ Operator” on page 274.
- “Collecting troubleshooting information for self-deployed queue managers on Kubernetes” on page 280.

OpenShift CP4I Kubernetes Troubleshooting unplanned restarts of IBM MQ in containers

In most container management systems such as Red Hat OpenShift Container Platform and Kubernetes, containers are commonly restarted. It is not normal for a container to be long-lived. This topic explains

the container lifecycle, how you can investigate a restart, and the reasons behind an unplanned container restart.

If you have seen no issues with your IBM MQ deployment, and it continues to run as expected, it is likely the solution is performing as intended. You may see a log message like this in the container log:

```
Signal received: terminated
```

This means that the SIGTERM signal has been sent to the MQ container, asking it to terminate. Linux containers have a responsibility to respond to POSIX signals, which are standardized messages sent to a program to trigger behavior. The SIGTERM signal is used by Kubernetes (and thus Red Hat OpenShift Container Platform) to gracefully terminate a container.

When the IBM MQ container receives a SIGTERM signal, it issues an `endmqm -w -r -tp` command, to stop the queue manager. Once the queue manager has stopped, the container will then stop. If the queue manager takes a long time to stop, then a SIGKILL signal might be sent, which will immediately terminate the Linux processes. The amount of time between a SIGTERM and a SIGKILL is known as the “termination grace period” in Kubernetes, and is configurable on the `QueueManager` resource (if you are using the IBM MQ Operator), or directly on the Pod resource. The default is 30 seconds, of which one second is reserved for the container to shut down, and the remainder is given to IBM MQ. For example, in the default case, an `endmqm -w -tp 29` is issued, which tells the queue manager it’s got 29 seconds to shut down.

Reasons for container termination

Kubernetes can restart a container within an existing Pod, or it can delete the Pod. There are fields in the Pod's "status" section which help indicate whether this is a container restart, for example:

```
status:
  containerStatuses:
  - restartCount: 2
    started: true
    state:
      running:
        startedAt: "2025-01-20T12:03:15Z"
  startTime: "2024-12-28T13:46:12Z"
```

In this example, you can see that the container has restarted, and that the container is much newer than the Pod. This indicates that a single container restarted, which is likely to mean that the container failed its liveness probe. A liveness probe can be configured in Kubernetes to check that a container is still healthy. The IBM MQ Operator sets up a queue manager liveness probe which calls the **dspmq** command to check for a valid running state. If the queue manager is not in a healthy state, or if running the probe itself takes too long, then the kubelet will consider that a failure. Thresholds for the number of failures to tolerate are configurable, either on the `QueueManager` resource (if you are using the IBM MQ Operator), or directly on the Pod resource.

If you don't see container restarts, then the whole Pod has probably been replaced. See Termination of Pods in the Kubernetes documentation. Kubernetes uses the terms “Pod Disruption” and “eviction" for the process by which Pods on Nodes are terminated either voluntarily or involuntarily. There are many reasons why a Pod might be evicted, including:

- **Rolling restart**. The IBM MQ Operator or Stateful Set controller can trigger a rolling update of queue manager Pods, if a configuration change is made which requires a rollout. For example, changing image version, mounted files, environment variables, annotations, labels, and more.
- **Termination by kubelet**. This can be for a number of reasons, including the following:
  - The Pod can be terminated because the Node is being shut down (perhaps as part of a rolling cluster update)
  - The Pod can be terminated due to Node “pressure” (where the kubelet proactively terminates Pods to reclaim resources on a Node). The Kubernetes cluster administrator can configure eviction thresholds which might vary between clusters.
- **Preemption by the Kubernetes scheduler**. This can happen if Kubernetes scheduler has to run a higher priority Pod

- **Tainted Node**. A Node can be “tainted”, and Pods which do not tolerate the taint are evicted. Taints are used by Kubernetes administrators to “repel” Pods from specific Nodes. For example, in order to say that IBM MQ Pods should no longer run on Nodes which have special hardware which is now reserved for other workloads.
- **Request via the Eviction API**. This can be called by an administrator to evict Pods
- **Pod garbage collection**. This can happen if the Node goes out of service, or is removed via the Kubernetes API.

Determining why a queue manager Pod was evicted

Potential sources of information to help understand why a Pod was evicted include:

- **Cluster events**. For example:
  - Red Hat OpenShift: Viewing system event information in OpenShift Container Platform clusters
  - Amazon EKS: Monitor your cluster performance and view logs
- **Cluster audit events**. For example:
  - Red Hat OpenShift: Viewing audit logs.
  - Amazon EKS: Auditing and logging
- **Nodes under pressure**. Look for Nodes under CPU, network, or memory pressure. You can see this in the Node status. Note that by the time you come to look, the Node may no longer be under pressure.
- **Red Hat OpenShift Container Platform Monitoring** or other monitoring metrics might be able to show things like disk latency issues. A useful Prometheus metric is `ibmmq_qmgr_log_write_latency_seconds`. This information comes from the MQ statistics topics.

Related information

Kubernetes documentation on Scheduling, Preemption and Eviction

OpenShift CP4I Podman Collecting troubleshooting information with IBM MQ container inspector

The IBM MQ container inspector tool collects troubleshooting information about IBM MQ queue managers.

Before you begin

Check that you meet the following prerequisites:

1. You are logged into the cluster that hosts your IBM MQ deployment.
2. You have permission to access the target namespace.

The commands in this topic use the following variables:

- *<queue_manager_name>*: the name of the QueueManager resource (`metadata.name` in the queue manager YAML)
- *<queue_manager_namespace>*: the namespace where the QueueManager resource is deployed (`metadata.namespace` in the queue manager YAML)
- *<output_directory>*: the directory to which the IBM MQ container inspector tool writes the collected information

Note: For deployments that don't use an operator, replace the parameter `--qm-name <queue_manager_name>` in the commands in this topic with `--pod-name <pod_name>`, where <pod_name> is the name of one of the queue manager pods.

About this task

The IBM MQ container inspector tool includes the following commands:

- **mustgather**: gathers information for you to provide to IBM Support when you raise a support case.
- **pvctool**: provides access to the files on a queue manager PVC where a remote shell cannot be established to the queue manager pod. This restriction might be because the pod is in an `Error` or `CrashLoopBackOff` state.

You can view the IBM MQ container inspector source code and license in the mq-container-inspector GitHub repository.

For detailed usage instructions and information about the data that the **mustgather** command collects, see the MQ container inspector **mustgather** command page in the mq-container-inspector GitHub repository.

You can run the IBM MQ container inspector in the following ways, which are described in this topic:

- As a Go binary file that you run locally
- Podman As an image that you run in Podman or Docker
- OpenShift CP4I As an image that you run in a pod on Red Hat OpenShift Container Platform

Alternatively, you can gather the required files manually by running commands against the cluster, as described in the following topics:

- OpenShift CP4I "Collecting troubleshooting information for queue managers deployed with the IBM MQ Operator" on page 274
- Kubernetes "Collecting troubleshooting information for self-deployed queue managers on Kubernetes" on page 280
- OpenShift CP4I "Troubleshooting: Gaining access to queue manager data" on page 277

Procedure

Running the IBM MQ container inspector Go binary file

1. Run the IBM MQ container inspector binary file directly from your machine to collect troubleshooting information.
   a) Download or build the IBM MQ container inspector tool by following the instructions in the mq-container-inspector GitHub repository.
   b) Run the tool by entering the following command, substituting your own queue manager name and namespace, and the output directory.

```
mq-container-inspector mustgather --qm-name <queue_manager_name> --qm-namespace
<queue_manager_namespace>
        --output-dir <output_directory>
```

   c) Optional: If the queue manager is in `Error` or `CrashLoopBackOff` state, use the **pvctool** to collect data by using the **runmqras** command.

```
mq-container-inspector pvctool --qm-name <queue_manager_name> --qm-namespace
<queue_manager_namespace>
        --output-dir <output_directory> --runmqras --cleanup
```

   This command creates a new pod (or pods) for each queue manager, which is attached to the queue manager PVC. The tool then runs the **runmqras** command against that pod to collect the required IBM MQ troubleshooting information and write it to the output directory. After the data is collected, the command deletes the **pvctool** pods.

Podman Running IBM MQ container inspector as an image in Podman or Docker

2. IBM MQ container inspector is included in the IBM Cloud Pak foundational services MustGather image (`icr.io/cpopen/cpfs/must-gather`). If you have Podman or Docker running on your machine, you can use this image to collect IBM MQ troubleshooting information.
   a) Create a directory to send the troubleshooting information to.

```
mkdir -p <output_directory>
chmod 775 <output_directory>
```

b) Run the following command (if you are using Docker, replace podman with `docker`), substituting your own queue manager name and namespace, and the output directory that you created in the previous step.

```
podman run -v ~/.kube/config:/.kube/config -v $(pwd)/<output_directory>:/tmp/must-gather \
icr.io/cpopen/cpfs/must-gather:latest gather -m mq -- mustgather --qm-name
<queue_manager_name> --qm-namespace <queue_manager_namespace>
```

This command runs the IBM MQ container inspector **mustgather** tool from your Podman or Docker agent by using the `~/.kube/config` file to connect to your cluster. The output from the tool is written to the output directory that you created.

c) Optional: If the queue manager is in `Error` or `CrashLoopBackOff` state, use the **pvctool** to collect data by using the **runmqras** command.

```
podman run -v ~/.kube/config:/.kube/config -v $(pwd)/<output_directory>:/tmp/must-gather \
icr.io/cpopen/cpfs/must-gather:latest gather -m mq -- pvctool --qm-name
<queue_manager_name> --qm-namespace <queue_manager_namespace> --output-dir
<output_directory> --runmqras --cleanup
```

This command creates a new pod (or pods) for each queue manager, which is attached to the queue manager PVC. The tool then runs the **runmqras** command against that pod to collect the required IBM MQ troubleshooting information and write it to the output directory. After the data is collected, the command deletes the **pvctool** pods.

Running IBM MQ container inspector in a pod in Red Hat OpenShift Container Platform

3. IBM MQ container inspector is included in the IBM Cloud Pak foundational services MustGather image (`icr.io/cpopen/cpfs/must-gather`). You can run this image by using the OpenShift **oc adm must-gather** command, as described in Gathering data about your cluster in the Red Hat documentation. This command is available only on OpenShift and requires admin permissions for the OpenShift cluster.

a) Run the **oc adm must-gather** command, substituting the output directory and your own queue manager name and namespace.

```
oc adm must-gather --dest-dir=<output_directory> --image=icr.io/cpopen/cpfs/must-
gather:latest -- gather -m mq -- mustgather --qm-name <queue_manager_name> --qm-namespace
<queue_manager_namespace>
```

b) Verify the output.

The IBM MQ container inspector tool sends output to *<output_directory>*`/icr-io-cpopen-cpfs-must-gather-`*<image_sha>*`/cloudpak-must-gather-timestamp.tar.gz`. Extract the package then verify the contents of the `Must_Gather_<timestamp>` directory before you submit the `Must_Gather_<timestamp>.tar.gz` to IBM Support. If this file is missing, check the `gather.log` file (created by the `oc adm must-gather` command) for an error message that explains why the data gather operation was unsuccessful.

c) Optional: If the queue manager is in `Error` or `CrashLoopBackOff` state, use the **pvctool** to collect data by using the **runmqras** command.

```
oc adm must-gather --dest-dir=<output_directory> --image=icr.io/cpopen/cpfs/must-
gather:latest -- gather -m mq -- pvctool --qm-name <queue_manager_name> --qm-namespace
<queue_manager_namespace> --output-dir <output_directory> --runmqras --cleanup
```

This command creates a new pod (or pods) for each queue manager, which is attached to the queue manager PVC. The tool then runs the **runmqras** command against that pod to collect the required IBM MQ troubleshooting information and write it to the output directory. After the data is collected, the command deletes the **pvctool** pods.

Troubleshooting problems with the IBM MQ Operator

If you are having problems with IBM MQ Operator, use the techniques described to help you diagnose and solve them.

Procedure

- “Collecting troubleshooting information for queue managers deployed with the IBM MQ Operator” on page 274
- “Troubleshooting: Gaining access to queue manager data” on page 277
- “Troubleshooting container clusters with the OpenShift Monitoring CLI” on page 278

Collecting troubleshooting information for queue managers deployed with the IBM MQ Operator

If you need to raise a case with IBM Support, provide troubleshooting information to help diagnose the issue. If possible, use the IBM MQ container inspector tool to collect the information. If you cannot use the tool, you can collect the information yourself manually.

Before you begin

Note: To collect troubleshooting information, use the IBM MQ container inspector tool if possible, as described in “Collecting troubleshooting information with IBM MQ container inspector” on page 271. This tool collects information in a simplified and standardized way. If you cannot use the tool or prefer not to use it, you can use the following manual procedure instead.

Procedure

1. Collect cloud provider information.

   This is the cloud provider that hosts your cluster (for example, IBM Cloud).
2. Collect architecture information.

   The architecture of your cluster is one of the following:

   - Linux for x86-64
   - Linux on Power Systems (ppc64le)
   - Linux on IBM Z®
3. Collect IBM MQ deployment information.

   a) Log on to your cluster, using a `bash`/`zsh` shell.

   b) Set the following environment variables:

   ```
   export QM=QueueManager_name
   export QM_NAMESPACE=QueueManager_namespace
   export MQ_OPERATOR_NAMESPACE=mq_operator_namespace
   ```

   Where *QueueManager_name* is the name of your QueueManager resource (`metadata.name` in the queue manager YAML), *QueueManager_namespace* is the namespace where it is deployed (`metadata.namespace` in the queue manager YAML), and *mq_operator_namespace* is the namespace where the IBM MQ Operator is deployed. This might be the same as the QueueManager namespace.

   c) Run the following commands, and provide all of the resulting output files to IBM Support.

   Notes:

   - The majority of these commands require access to the namespace where the queue manager is deployed. However, collecting the IBM MQ Operator log might additionally require **cluster administrator** access if the IBM MQ Operator is installed **cluster-scoped**.

- If you have a running queue manager, ensure that the output files that are produced by these commands are not empty (except for the `qm-logs-previous-`*pod_name*`.txt` pod logs, which are expected to be empty if the container did not restart).
- On Red Hat OpenShift:

```
# OCP / Kubernetes: Version
oc version -o yaml > ocversion.yaml

# QueueManager: YAML
oc get qmgr $QM -n $QM_NAMESPACE -o yaml > "queue-manager-$QM.yaml"

# MQ Queue Manager: Pods
oc get pods -n $QM_NAMESPACE -o wide --selector "app.kubernetes.io/instance=$QM" > "qm-
pods-$QM.txt"

# MQ Queue Manager: Pod YAML
oc get pods -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/instance=$QM" > "qm-
pods-$QM.yaml"

# MQ Queue Manager: Pod Logs
for p in $(oc get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do oc logs -n $QM_NAMESPACE --previous "$p" > "qm-
logs-previous-$p.txt"; oc logs -n $QM_NAMESPACE $p > "qm-logs-$p.txt";done

# MQ Queue Manager: Describe Pods
for p in $(oc get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do oc describe pod $p -n $QM_NAMESPACE > "qm-pod-
describe-$p.txt"; done

# MQ Web UI: Console Log
for p in $(oc get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do oc cp -n $QM_NAMESPACE --retries=10
"$p:var/mqm/web/installations/Installation1/servers/mqweb/logs/console.log" "web-$p-
console.log"; done

# MQ Web UI: Messages Log
for p in $(oc get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do oc cp -n $QM_NAMESPACE --retries=10
"$p:var/mqm/web/installations/Installation1/servers/mqweb/logs/messages.log" "web-$p-
messages.log"; done

# MQ Queue Manager: routes defined by operator
oc get routes -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/instance=$QM" >
"qm-routes-$QM.yaml"

# MQ Queue Manager: routes to QM
oc get routes -n $QM_NAMESPACE -o yaml --field-selector "spec.to.name=$QM-ibm-mq" > "qm-
routes2-$QM.yaml"

# MQ Queue Manager: stateful set
oc get statefulset -n $QM_NAMESPACE -o yaml ${QM}-ibm-mq > "qm-statefulset-$QM.yaml"

# MQ Queue Manager: revisions of the stateful set
oc get controllerrevisions.apps -o yaml -n $QM_NAMESPACE --selector "app.kubernetes.io/
instance=$QM" > "qm-statefulset-revisions-$QM.yaml"

# MQ Queue Manager: Pod events
for p in $(oc get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do oc get -o custom-columns="LAST
SEEN:.lastTimestamp,TYPE:.type,REASON:.reason,KIND:.involvedObject.kind,NAME:.involvedOb
ject.name,MESSAGE:.message" event -n $QM_NAMESPACE --field-selector
involvedObject.name="$p" > "qm-pod-events-$p.txt"; done

# MQ Queue Manager: StatefulSet events
oc get events -n $QM_NAMESPACE -o custom-columns="LAST
SEEN:.lastTimestamp,TYPE:.type,REASON:.reason,KIND:.involvedObject.kind,NAME:.involvedOb
ject.name,MESSAGE:.message" --field-selector involvedObject.name="${QM}-ibm-mq" > "qm-
statefulset-events-$QM.txt"

# MQ Queue Manager: services
oc get services -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/instance=$QM" >
"qm-services-$QM.yaml"

# MQ Queue Manager: PVCs
oc get pvc -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/instance=$QM" > "qm-
pvcs-$QM.yaml"

# MQ Operator: Version
oc get csv -n $QM_NAMESPACE | grep "^ibm-mq\|NAME" > mq-operator-csv.txt
```

```
# Cloud Pak Foundational Services: Version
oc get csv -n $QM_NAMESPACE | grep "^ibm-common-service-operator\|NAME" > common-
services-csv.txt

# Cloud Pak for Integration: Version (if applicable)
oc get csv -n $QM_NAMESPACE | grep "^ibm-integration-platform-navigator\|NAME" > cp4i-
csv.txt

# Output from runmqras (this may take a while to execute)
for p in $(oc get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do timestamp=$(TZ=UTC date +"%Y%m%d_%H%M%S"); oc exec
-n $QM_NAMESPACE $p -- runmqras -workdirectory "/tmp/runmqras_$timestamp" -section
logger,mqweb,nativeha,trace; oc cp -n $QM_NAMESPACE --retries=10 "$p:tmp/
runmqras_$timestamp/" .; done

# MQ Operator: Pod Log
oc logs -n $MQ_OPERATOR_NAMESPACE $(oc get pods -n $MQ_OPERATOR_NAMESPACE --no-headers
--selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/managed-by=olm | cut -d ' '
-f 1) > mq-operator-log.txt
```

- On Amazon EKS:

```
# Kubernetes: Version
kubectl version -o yaml > kubeversion.yaml

# QueueManager: YAML
kubectl get qmgr $QM -n $QM_NAMESPACE -o yaml > "queue-manager-$QM.yaml"

# MQ Queue Manager: Pods
kubectl get pods -n $QM_NAMESPACE -o wide --selector "app.kubernetes.io/instance=$QM" >
"qm-pods-$QM.txt"

# MQ Queue Manager: Pod YAML
kubectl get pods -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/instance=$QM" >
"qm-pods-$QM.yaml"

# MQ Queue Manager: Pod Logs
for p in $(kubectl get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do kubectl logs -n $QM_NAMESPACE --previous "$p" >
"qm-logs-previous-$p.txt"; kubectl logs -n $QM_NAMESPACE $p > "qm-logs-$p.txt";done

# MQ Queue Manager: Describe Pods
for p in $(kubectl get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do kubectl describe pod $p -n $QM_NAMESPACE > "qm-pod-
describe-$p.txt"; done

# MQ Web UI: Console Log
for p in $(kubectl get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do kubectl cp -n $QM_NAMESPACE --retries=10
"$p:var/mqm/web/installations/Installation1/servers/mqweb/logs/console.log" "web-$p-
console.log"; done

# MQ Web UI: Messages Log
for p in $(kubectl get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do kubectl cp -n $QM_NAMESPACE --retries=10
"$p:var/mqm/web/installations/Installation1/servers/mqweb/logs/messages.log" "web-$p-
messages.log"; done

# MQ Queue Manager: stateful set
kubectl get statefulset -n $QM_NAMESPACE -o yaml ${QM}-ibm-mq > "qm-statefulset-
$QM.yaml"

# MQ Queue Manager: revisions of the stateful set
kubectl get controllerrevisions.apps -o yaml -n $QM_NAMESPACE --selector
"app.kubernetes.io/instance=$QM" > "qm-statefulset-revisions-$QM.yaml"

# MQ Queue Manager: Pod events
for p in $(kubectl get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do kubectl get -o custom-columns="LAST
SEEN:.lastTimestamp,TYPE:.type,REASON:.reason,KIND:.involvedObject.kind,NAME:.involvedOb
ject.name,MESSAGE:.message" event -n $QM_NAMESPACE --field-selector
involvedObject.name="$p" > "qm-pod-events-$p.txt"; done

# MQ Queue Manager: StatefulSet events
kubectl get events -n $QM_NAMESPACE -o custom-columns="LAST
SEEN:.lastTimestamp,TYPE:.type,REASON:.reason,KIND:.involvedObject.kind,NAME:.involvedOb
ject.name,MESSAGE:.message" --field-selector involvedObject.name="${QM}-ibm-mq" > "qm-
statefulset-events-$QM.txt"

# MQ Queue Manager: services
```

```
kubectl get services -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/
instance=$QM" > "qm-services-$QM.yaml"

# MQ Queue Manager: PVCs
kubectl get pvc -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/instance=$QM" >
"qm-pvcs-$QM.yaml"

# MQ Operator: Version
kubectl get deployment -n $QM_NAMESPACE -o yaml --selector "app.kubernetes.io/name=ibm-
mq" | grep productVersion > mq-operator-version.txt

# Output from runmqras (this may take a while to execute)
for p in $(kubectl get pods -n $QM_NAMESPACE --no-headers --selector "app.kubernetes.io/
instance=$QM" | cut -d ' ' -f 1); do timestamp=$(TZ=UTC date +"%Y%m%d_%H%M%S"); kubectl
exec -n $QM_NAMESPACE $p -- runmqras -workdirectory "/tmp/runmqras_$timestamp" -section
logger,mqweb,nativeha,trace; kubectl cp -n $QM_NAMESPACE --retries=10 "$p:tmp/
runmqras_$timestamp/" .; done

# MQ Operator: Pod Log
kubectl logs -n $MQ_OPERATOR_NAMESPACE $(kubectl get pods -n $MQ_OPERATOR_NAMESPACE --
no-headers --selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/managed-by=Helm |
cut -d ' ' -f 1) > mq-operator-log.txt
```

Related tasks

Collecting troubleshooting information

OpenShift CP4I Troubleshooting: Gaining access to queue manager data

If possible, use the IBM MQ container inspector tool to collect troubleshooting information. If you cannot use the tool, you can collect the information yourself manually. As part of the manual process, you can use the PVC inspector tool to gain access to the files on a queue manager PVC where a remote shell cannot be established to the queue manager pod. This might be because the pod is in an **`Error`** or **`CrashLoopBackOff`** state. This tool is designed for use with queue managers deployed by the IBM MQ Operator on Red Hat OpenShift **only**..

Before you begin

Notes:

- If your deployment is on Amazon EKS, use the IBM MQ container inspector tool instead.
- If your deployment is on Red Hat OpenShift, use the IBM MQ container inspector if possible because it collects information in a simplified and standardized way. If you cannot use the tool or prefer not to use it, you can use the following manual procedure.

If you continue with this task, ensure that you have access to your queue manager namespace.

About this task

If you are collecting troubleshooting information manually, you can access the data that is stored on the Persistent Volume Claims (PVCs) associated with a given queue manager by using a tool to mount the PVCs to a set of inspector pods. You can then get a remote shell into any of the inspector pods to read the files.

Depending on the type of deployment, between one and three inspector pods are created. Volumes specific to a given pod of a Native-HA or Multi-Instance queue manager are available on the associated PVC inspector pod. Shared volumes are available on all inspectors. The name of the inspector pod contains the name of the associated queue manager pod.

Procedure

1. Download the MQ PVC inspector tool.

   The tool is available here: https://github.com/ibm-messaging/mq-pvc-tool.
2. Make sure that you are logged into your cluster.
3. Find out the name of the queue manager, and the namespace the queue manager is running in.

4. Run the inspector tool against your queue manager.
   a) Run the following command, specifying your queue manager name and its namespace name.

```
./pvc-tool.sh queue_manager_name queue_manager_namespace_name
```

   b) After the tool has completed, run the following command to view the inspector pods being created.

```
oc get pods
```

5. View the files mounted to the inspector pod.
   a) Each PVC inspector pod is associated with a queue manager pod, so there might be multiple inspector pods. Access one of these pods, by running the following command:

```
oc rsh pvc-inspector-pod-name
```

   You are placed in the directory containing the mounted PVC directories.
   b) List the PVC directories by running the following command:

```
ls
```

   c) See a list of the PVCs by running the following command outside of the remote shell session:

```
oc get pvc
```

   d) Clean up the pods created by the tool, by running the following command:

```
oc delete pods -l tool=mq-pvc-inspector
```

OpenShift CP4I Troubleshooting container clusters with the OpenShift Monitoring CLI

As a troubleshooting aid, you can query the available container cluster metrics using the OpenShift Monitoring CLI.

Before you begin

OpenShift Monitoring provides a web console and a CLI. For day-to-day operations you might prefer to use the console, as described in “Monitoring container clusters with the OpenShift Monitoring console” on page 246. For gathering troubleshooting information you might prefer to use the CLI, as described in the rest of this task.

About this task

OpenShift Container Platform includes a pre-configured, pre-installed and self-updating monitoring stack that uses an open source Prometheus server. Queue managers running under an IBM MQ Operator can produce metrics compatible with Prometheus. For a complete list of the metrics available for IBM MQ, see “Metrics published by the IBM MQ container” on page 248.

OpenShift Monitoring is the default cluster monitoring method. It is installed with the cluster. To query the available container cluster metrics using the OpenShift Monitoring CLI, complete the following steps.

Procedure

1. Enable OpenShift Monitoring for the operands deployed in an IBM MQ container cluster.
   a) Log in to the cluster as a cluster administrator.
   b) Add "`enableUserWorkload: true`" to the configMap `cluster-monitoring-config` in the `openshift-monitoring` namespace. To do this, follow the steps in Enabling monitoring for user-defined projects in the Red Hat OpenShift documentation. If the `cluster-monitoring-config`

ConfigMap does not exist in the `openshift-monitoring` namespace, follow the instructions to create it.

2. Install the OpenShift command line tool (oc).

   See Getting started with the OpenShift CLI.

3. Query the IBM MQ metrics using the OpenShift Monitoring CLI.

   When using the CLI, first declare the variables by using the **export** command. For example:

```
export OCP_LOGIN_TOKEN=$(oc whoami -t)
export THANOS_QUERIER_HOST=$(oc -n openshift-monitoring get route thanos-querier -o
jsonpath={.spec.host})
export QM_NAMESPACE=namespace-in-which-the-target-queue-manager-is-installed
export QM_METRIC_NAME=IBM-MQ-metric
```

   To run PromQL queries from the CLI, use the **PROMQL_QUERY** command. For example:

   - Get a list of queue managers that have a positive FDC count:

```
export QM_METRIC_NAME=ibmmq_qmgr_fdc_files
export PROMQL_QUERY="$QM_METRIC_NAME{namespace=\"$QM_NAMESPACE\"} > 0"
```

   - Get data for a specific time period:

```
export START=start_time
export END=end_time
export STEP=step_interval
curl -G -s -k -H "Authorization: Bearer $OCP_LOGIN_TOKEN" --data-urlencode
"query=$PROMQL_QUERY" \
--data-urlencode "start=$START" --data-urlencode "end=$END" \
--data-urlencode "step=$STEP" "https://$THANOS_QUERIER_HOST/api/v1/query_range"
```

   where *start_time* and *end_time* specify the time in ISO 8601 format, and *step_interval* is (for example) 15m, or 30s, or 1h.

   You can also export the data to a CSV file:

   a. The following command uses the same headers that display in the web console output shown in the screen capture in "Monitoring container clusters with the OpenShift Monitoring console" on page 246. To export to a CSV file, and pass the same headers that display in the web console output, first download and install `jq`.

   b. Run the following command:

```
(
echo
"metric_name,container,endpoint,instance,job,namespace,pod,prometheus,qmgr,service,timesta
mp,value"
curl -G -s -k -H "Authorization: Bearer $OCP_LOGIN_TOKEN" --data-urlencode
"query=$PROMQL_QUERY" \
--data-urlencode "start=$START" --data-urlencode "end=$END" --data-urlencode "step=$STEP"
\
"https://$THANOS_QUERIER_HOST/api/v1/query_range"| jq -r '.data.result[] | .metric as
$metric | .values[] | \
[$metric.__name__,
$metric.container,
$metric.endpoint,
$metric.instance,
$metric.job,
$metric.namespace,
$metric.pod,
$metric.prometheus,
$metric.qmgr,
$metric.service,
(.[0] | strftime("%Y-%m-%dT%H:%M:%SZ")), /* converts the timestamp from milliseconds to
ISO 8601 format */
.[1] ] | @csv'
) > your_filename.csv
```

CLI example: Write latency of Queue manager pods over the last 5 hours with a resolution of 10 minutes

```
export QM_METRIC_NAME=ibmmq_qmgr_log_write_latency_seconds
export PROMQL_QUERY="$QM_METRIC_NAME{namespace=\"$QM_NAMESPACE\"} [5h:10m]"
```

Export the data to a CSV file:

```
(
echo
"metric_name,container,endpoint,instance,job,namespace,pod,prometheus,qmgr,service,timestamp,val
ue"
curl -G -s -k -H "Authorization: Bearer $OCP_LOGIN_TOKEN" \
--data-urlencode "query=$PROMQL_QUERY" \
"https://$THANOS_QUERIER_HOST/api/v1/query" | jq -r '.data.result[] | .metric as $metric
| .values[] | \
[$metric.__name__,
$metric.container,
$metric.endpoint,
$metric.instance,
$metric.job,
$metric.namespace,
$metric.pod,
$metric.prometheus,
$metric.qmgr,
$metric.service,
(.[0] | strftime("%Y-%m-%dT%H:%M:%SZ")), /* converts the timestamp from milliseconds to ISO
8601 format */
.[1] ] | @csv'
) > your_filename.csv
```

Related tasks

“Monitoring when using the IBM MQ Operator” on page 244
Queue managers managed by the IBM MQ Operator can produce metrics compatible with Prometheus. Queue manager metrics are enabled and served through HTTP by default.

“Monitoring container clusters with the OpenShift Monitoring console” on page 246
You can query the available container cluster metrics on deployments on Red Hat OpenShift by using the Red Hat OpenShift Monitoring web console.

Related reference

“Metrics published by the IBM MQ container” on page 248
Queue manager containers can publish metrics compatible with Red Hat OpenShift Monitoring.

Kubernetes Collecting troubleshooting information for self-deployed queue managers on Kubernetes

If you need to raise a case with IBM Support for self-deployed queue managers, provide troubleshooting information to help diagnose the issue. If possible, use the IBM MQ container inspector tool to collect the information. If you cannot use the tool, you can collect the information yourself manually.

Before you begin

Note: To collect troubleshooting information, use the IBM MQ container inspector tool if possible, as described in “Collecting troubleshooting information with IBM MQ container inspector” on page 271. This tool collects information in a simplified and standardized way. If you cannot use the tool or prefer not to use it, you can use the following manual procedure instead.

Procedure

1. Collect cloud provider information.

   This is the cloud provider that hosts your Kubernetes cluster (for example, IBM Cloud).
2. Collect the Kubernetes version information
   For example, run `kubectl version -o yaml > kubeversion.yaml`

3. Collect resource YAML describing key elements of how IBM MQ is deployed.
   Include the following:
   a. StatefulSet
   b. Pod (for each queue manager instance)
   c. Service
   d. PersistentVolumeClaim
   e. ConfigMaps

   For example, run `kubectl get pod` *PodName* `-o yaml` for each queue manager Pod.
4. Collect a description of each Pod, including events.
   For example, run `kubectl describe pod` for each queue manager pod.
5. Collect all Pod logs
   For example, run `kubectl logs` *PodName* for each queue manager Pod.
6. Collect the output from the **runmqras** command
   Use `kubectl exec` to run the **runmqras** command inside each queue manager container.

Related tasks

Collecting troubleshooting information

Reference information for IBM MQ in containers

IBM MQ provides a Kubernetes Operator, which you can deploy on Red Hat OpenShift Container Platform and, from IBM MQ Operator 3.9.0, Amazon EKS.

Licensing and API references for the IBM MQ Operator

IBM MQ provides a Kubernetes Operator, which you can deploy on Red Hat OpenShift Container Platform and, from IBM MQ Operator 3.9.0, Amazon EKS. The `v1beta1` API can be used to create and manage queue manager resources.

mq.ibm.com/v1beta1: Current license versions

The `spec.license.license` field must contain the license identifier for the license you are accepting. Valid values are as follows:

Table 2. Valid values for the `spec.license.license` field

Value of `spec.license.license`	Value of `spec.license.use`	License information	Applicable IBM MQ versions
`L-VEMB-RABZSN`	`Production` or `NonProduction`	IBM Cloud Pak for Integration 16.2.0	10.0.0
`L-UHEB-WGGTYF`	`Production` or `NonProduction`	IBM Cloud Pak for Integration Limited Edition 16.2.0	10.0.0
`L-CXDB-XTZ9J8`	`Production`	IBM MQ Advanced 10.0	10.0.0
`L-UMVX-XYNRAD`	`NonProduction`	IBM MQ Advanced for Non-Production Environment 10.0	10.0.0
`L-GWLM-M6WVZU`	`Development`	IBM MQ Advanced for Developers (Non-Warranted) 10.0	10.0.0
`L-MKDD-7KHY2Q`	`Production` or `NonProduction`	IBM MQ 10.0 and IBM MQ for Non-Production Environment 10.0	10.0.0

Note that the license *version* is specified, which is not always the same as the version of IBM MQ.

You can use the `spec.license.additionalLicenses` field to configure add-on licenses to use in combination with your selected `spec.license.license` value. Valid `spec.license.additionalLicenses` values are as follows:

Table 3. Valid values for the `spec.license.additionalLicenses` field

Value of `spec.license.additionalLicenses`	**Value of `spec.license.use`**	**License information**	**Applicable IBM MQ versions**
`L-KFCR-53UGK9`	`Production` or `NonProduction`	IBM MQ Native HA and Cross-Region Replication Add-On 10.0 and IBM MQ Native HA and Cross-Region Replication Add-On for Non-Production Environment 10.0	10.0.0
`L-MKDD-7KHY2Q`	`Production` or `NonProduction`	IBM MQ 10.0 and IBM MQ for Non-Production Environment 10.0	10.0.0

Valid combinations of the `spec.license.license` and `spec.license.additionalLicenses` fields are as follows:

Table 4. Valid combinations of values for the `spec.license.license` and `spec.license.additionalLicenses` fields

Value of `spec.license.license`	**Value of `spec.license.additionalLicenses`**	**Value of `spec.license.use`**	**Applicable IBM MQ versions**
`L-MKDD-7KHY2Q`	`L-KFCR-53UGK9`	`Production` or `NonProduction`	10.0.0
`L-VEMB-RABZSN`	`L-MKDD-7KHY2Q`	`Production` or `NonProduction`	10.0.0
`L-UHEB-WGGTYF`	`L-MKDD-7KHY2Q`	`Production` or `NonProduction`	10.0.0

mq.ibm.com/v1beta1: Older license versions

See Older license versions in the IBM MQ 9.4 documentation.

OpenShift CP4I API reference for QueueManager (mq.ibm.com/v1beta1)

QueueManager

A QueueManager is an IBM MQ server which provides queuing and publish/subscribe services to applications. IBM MQ Documentation: https://ibm.biz/BdPZqj. License reference: https://ibm.biz/Bdm9be.

Field	Description
`apiVersion` string	APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources .
`kind` string	Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds .
`metadata`	
`spec` QueueManagerSpec	The desired state of the QueueManager.
`status` QueueManagerStatus	The observed state of the QueueManager.

.spec

The desired state of the QueueManager.

Appears in:

- "QueueManager" on page 282

Field	Description
`affinity`	Standard Kubernetes affinity rules. For more information, see https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#affinity-v1-core .
`annotations` Annotations	The annotations field serves as a pass-through for Pod annotations. Users can add any annotation to this field and have it apply to the Pod. The annotations here overwrite the default annotations if provided. Requires MQ Operator 1.3.0 or higher.
`imagePullSecrets` LocalObjectReference array	An optional list of references to secrets in the same namespace to use for pulling any of the images used by this QueueManager. If specified, these secrets will be passed to individual puller implementations for them to use. For example, in the case of docker, only DockerConfig type secrets are honored. For more information, see https://kubernetes.io/docs/concepts/containers/images#specifying-imagepullsecrets-on-a-pod.
`labels` Labels	The labels field serves as a pass-through for Pod labels. Users can add any label to this field and have it apply to the Pod. The labels here overwrite the default labels if provided. Requires MQ Operator 1.3.0 or higher.
`license` License	Settings that control your acceptance of the license, and which license metrics to use.
`pki` PKI	Public Key Infrastructure settings, for defining keys and certificates for use with Transport Layer Security (TLS) or MQ Advanced Message Security (AMS).
`queueManager` QueueManagerConfig	Settings for the Queue Manager container and underlying Queue Manager.
`securityContext` Securitycontext	Security settings to add to the Queue Manager Pod's securityContext.

Field	Description
`service` Service	Settings that apply to all QueueManager services. Modification or removal of this section or the fields within it may result in the regeneration of all QueueManager services. Service regeneration will disrupt communication resulting in downtime for all QueueManager types including NativeHA. Regenerated Services are assigned new ClusterIP addresses by Kubernetes. Requires MQ Operator 3.6.0 or higher.
`telemetry` Telemetry	Settings for Open Telemetry configuration. Requires MQ Operator 2.2.0 or higher.
`template` Template	Advanced templating for Kubernetes resources. The template allows users to override how IBM MQ generates the underlying Kubernetes resources, such as StatefulSet, Pods and Services. This is for advanced users only, as it has the potential to disrupt normal operation of MQ if used incorrectly. Any values specified anywhere else in the QueueManager resource will be overridden by settings in the template.
`terminationGracePeriodSeconds` integer	Optional duration in seconds the Pod needs to terminate gracefully. Value must be non-negative integer. The value zero indicates delete immediately. The target time in which ending the queue manager is attempted, escalating the phases of application disconnection. Essential queue manager maintenance tasks are interrupted if necessary. Defaults to 30 seconds.
`tracing` TracingConfig	Settings for tracing integration with the Cloud Pak for Integration Operations Dashboard.
`version` string	Setting that controls the version of MQ that will be used (required). For example: `9.1.5.0-r2` would specify MQ version 9.1.5.0, using the second revision of the container image. Container-specific fixes are often applied in revisions, such as fixes to the base image.
`web` WebServerConfig	Settings for the MQ web server.

.spec.annotations

The annotations field serves as a pass-through for Pod annotations. Users can add any annotation to this field and have it apply to the Pod. The annotations here overwrite the default annotations if provided. Requires MQ Operator 1.3.0 or higher.

Appears in:

- ".spec" on page 283

.spec.imagePullSecrets

LocalObjectReference contains enough information to let you locate the referenced object inside the same namespace.

Appears in:

- ".spec" on page 283

Field	Description
`name` string	Name of the referent. This field is effectively required, but due to backwards compatibility is allowed to be empty. Instances of this type with an empty value here are almost certainly wrong. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names . Default: "".

.spec.labels

The labels field serves as a pass-through for Pod labels. Users can add any label to this field and have it apply to the Pod. The labels here overwrite the default labels if provided. Requires MQ Operator 1.3.0 or higher.

Appears in:

- ".spec" on page 283

.spec.license

Settings that control your acceptance of the license, and which license metrics to use.

Appears in:

- ".spec" on page 283

Field	Description
`accept` boolean	Whether or not you accept the license associated with this software (required).
`additionalLicenses` array	The identifiers of the additional licenses you are accepting. These must be the correct additional license identifiers for the version of MQ you are using. See https://ibm.biz/Bdm9be for valid values.
`license` string	The identifier of the license you are accepting. This must be the correct license identifier for the version of MQ you are using. See https://ibm.biz/Bdm9be for valid values.
`metric` string	Setting that specifies which license metric to use. For example, `ProcessorValueUnit`, `VirtualProcessorCore`, `ManagedVirtualServer`, or FREE. Defaults to `ProcessorValueUnit` when using an MQ Advanced license, an MQ Advanced for Non-Production license, or an IBM MQ license. Defaults to `VirtualProcessorCore` when using a Cloud Pak for Integration license. Defaults to FREE when using an MQ Advanced for Developers license.
`use` string	Setting that controls how the software will to be used, where the license supports multiple uses. See https://ibm.biz/Bdm9be for valid values.

.spec.pki

Public Key Infrastructure settings, for defining keys and certificates for use with Transport Layer Security (TLS) or MQ Advanced Message Security (AMS).

Appears in:

- ".spec" on page 283

Field	Description
`keys` PKISource array	Private keys to add to the Queue Manager's key repository.
`trust` PKISource array	Certificates to add to the Queue Manager's key repository.

.spec.pki.keys

PKISource defines a source of Public Key Infrastructure information, such as keys or certificates.

Appears in:

- ".spec.pki" on page 285

Field	Description
name string	Name is used as the label for the key or certificate. Must be a lowercase alphanumeric string.
secret Secret	Supply a key using a Kubernetes Secret.

.spec.pki.keys.secret

Supply a key using a Kubernetes Secret.

Appears in:

- ".spec.pki.keys" on page 285

Field	Description
items array	Keys inside the Kubernetes secret which should be added to the Queue Manager container.
secretName string	The name of the Kubernetes secret.

.spec.pki.trust

PKISource defines a source of Public Key Infrastructure information, such as keys or certificates.

Appears in:

- ".spec.pki" on page 285

Field	Description
name string	Name is used as the label for the key or certificate. Must be a lowercase alphanumeric string.
secret Secret	Supply a key using a Kubernetes Secret.

.spec.pki.trust.secret

Supply a key using a Kubernetes Secret.

Appears in:

- ".spec.pki.trust" on page 286

Field	Description
items array	Keys inside the Kubernetes secret which should be added to the Queue Manager container.
secretName string	The name of the Kubernetes secret.

.spec.queueManager

Settings for the Queue Manager container and underlying Queue Manager.

Appears in:

- ".spec" on page 283

Field	Description
availability Availability	Availability settings for the Queue Manager, such as whether or not to use an active-standby pair or native high availability.

Field	Description
debug boolean	Whether or not to log debug messages from the container-specific code, to the container log. Defaults to false.
env Env array	Settings for supplying environment variables for the Queue Manager. Requires MQ Operator 3.5.0 or higher.
files Files array	Settings for supplying files for the Queue Manager. Requires MQ Operator 3.5.0 or higher.
image string	The container image that will be used.
imagePullPolicy string	Setting that controls when the kubelet attempts to pull the specified image. Defaults to IfNotPresent.
ini INISource array	Settings for supplying INI for the Queue Manager. Requires MQ Operator 1.1.0 or higher.
livenessProbe QueueManagerLivenessProbe	Settings that control the liveness probe.
logFormat string	Which log format to use for this container. Use JSON for JSON-formatted logs from the container. Use Basic for text-formatted messages. Defaults to Basic.
metrics QueueManagerMetrics	Settings for Prometheus-style metrics.
mqsc MQSCSource array	Settings for supplying MQSC for the Queue Manager. Requires MQ Operator 1.1.0 or higher.
name string	Name of the underlying MQ Queue Manager, if different from metadata.name. Use this field if you want a Queue Manager name which does not conform to the Kubernetes rules for names (for example, a name which includes capital letters).
readinessProbe QueueManagerReadinessProbe	Settings that control the readiness probe.
recoveryLogs Recoverylogs	Settings for MQ recovery logs. Requires MQ Operator 2.4.0 or higher.
resources Resources	Settings that control resource requirements.
route Route	Settings for the Queue Manager route. Requires MQ Operator 1.4.0 or higher.
startupProbe Startupprobe	Settings that control the startup probe. Only applies to MultiInstance and NativeHA deployments. Requires MQ Operator 1.5.0 or higher.
storage QueueManagerStorage	Storage settings to control the Queue Manager's use of persistent volumes and storage classes.

.spec.queueManager.availability

Availability settings for the Queue Manager, such as whether or not to use an active-standby pair or native high availability.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
`nativeHAGroups` Nativehagroups	Settings for Native HA Groups. Requires `type of availability` to be `NativeHA`. Requires MQ Operator 3.5.0 or higher.
`tls` Tls	Optional TLS settings for configuring secure communication between NativeHA replicas. Requires MQ Operator 1.5.0 or higher.
`type` string	The type of availability to use. Use `SingleInstance` for a single Pod, which will be restarted automatically (in some cases) by Kubernetes. Use `MultiInstance` for a pair of Pods, one of which is the `active` Queue Manager, and the other of which is a standby. Use `NativeHA` for native high availability replication (requires MQ Operator 1.5.0 or higher). Defaults to `SingleInstance`. See https://ibm.biz/Bdm9TW for more details.
`updateStrategy` string	The update strategy to use for MultiInstance and NativeHA Queue Managers. Use `RollingUpdate` to enable automatic rolling updates whenever the Queue Manager configuration changes. Use `OnDelete` to disable automatic rolling updates, Queue Manager changes will only be applied when Pods are deleted (including Pod deletions triggered by external factors). Defaults to `RollingUpdate`. Requires MQ Operator 1.6.0 or higher.

.spec.queueManager.availability.nativeHAGroups

Settings for Native HA Groups. Requires `type of availability` to be `NativeHA`. Requires MQ Operator 3.5.0 or higher.

Appears in:

- ".spec.queueManager.availability" on page 287

Field	Description
`local` Local	Settings for this NativeHA group.
`remotes` Remotes array	Settings for connecting to remote NativeHA groups.

.spec.queueManager.availability.nativeHAGroups.local

Settings for this NativeHA group.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups" on page 288

Field	Description
`name` string	The Name of this NativeHA group.
`role` string	The role this NativeHA group should adopt. Either Live or Recovery. Valid values are: `"Live"; "Recovery"`.
`route` Route	Settings for the NativeHA group route.
`tls` Tls	TLS settings for configuring secure communication between NativeHA groups. Required if TLS is not configured in the parent Availability section.

.spec.queueManager.availability.nativeHAGroups.local.route

Settings for the NativeHA group route.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.local" on page 288

Field	Description
`enabled` boolean	Whether or not to enable the route. Defaults to true.

.spec.queueManager.availability.nativeHAGroups.local.tls

TLS settings for configuring secure communication between NativeHA groups. Required if TLS is not configured in the parent Availability section.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.local" on page 288

Field	Description
`cipherSpec` string	The name of the CipherSpec for NativeHA group TLS.
`key` Key	Settings for the TLS Key used in secure communication between Native HA groups.

.spec.queueManager.availability.nativeHAGroups.local.tls.key

Settings for the TLS Key used in secure communication between Native HA groups.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.local.tls" on page 289

Field	Description
`secretName` string	The name of the Kubernetes secret. The secret should be of type 'kubernetes.io/tls' containing a single private key, public certificate, and chain of trust.

.spec.queueManager.availability.nativeHAGroups.remotes

Appears in:

- ".spec.queueManager.availability.nativeHAGroups" on page 288

Field	Description
`addresses` array	An array of hostname and port addresses for communicating with the remote NativeHA group. Only effective when local group role is `Live`.
`enabled` boolean	Whether or not to begin communication with the remote. Only compatible with local NativeHA group with role 'Live'. If not set, MQ container will use internal default.
`name` string	The name of the remote NativeHA group. This must match the NativeHA group name configured in the remote cluster.
`retries` Retries	Configuration for retry behaviour for the remote.
`statusInterval` integer	The longest time interval in milliseconds that the Recovery group will wait before sending a Recovery log status.
`trust` Trust array	Certificates to add to the Queue Manager's key repository for communication with the remote NativeHA group.

.spec.queueManager.availability.nativeHAGroups.remotes.retries

Configuration for retry behaviour for the remote.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.remotes" on page 289

Field	Description
`long` Long	Retry configuration for the long retry phase.
`reporting` string	Whether to report all retry attempts or just the retry phase.
`short` Short	Retry configuration for the short retry phase.

.spec.queueManager.availability.nativeHAGroups.remotes.retries.long

Retry configuration for the long retry phase.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.remotes.retries" on page 289

Field	Description
`interval` integer	Interval in milliseconds to wait between attempts to connect to the Remote during the long retry phase.

.spec.queueManager.availability.nativeHAGroups.remotes.retries.short

Retry configuration for the short retry phase.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.remotes.retries" on page 289

Field	Description
`count` integer	The number of attempts to make a connection during the short retry phase.
`interval` integer	Interval in milliseconds to wait between attempts to connect to the Remote during the short retry phase.

.spec.queueManager.availability.nativeHAGroups.remotes.trust

NativeHAGroupTrust defines a source of certificate authority public keys.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.remotes" on page 289

Field	Description
`configMap` Configmap	Supply a CA certificate using a Kubenetes Config Map.
`secret` Secret	Supply a CA certificate using a Kubernetes Secret.

.spec.queueManager.availability.nativeHAGroups.remotes.trust.configMap

Supply a CA certificate using a Kubenetes Config Map.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.remotes.trust" on page 290

Field	Description
`items` array	Keys inside the Kubernetes configmap source which should be applied. If unspecified, uses all keys.

Field	Description
name string	The name of the Kubernetes configmap source.

.spec.queueManager.availability.nativeHAGroups.remotes.trust.secret

Supply a CA certificate using a Kubernetes Secret.

Appears in:

- ".spec.queueManager.availability.nativeHAGroups.remotes.trust" on page 290

Field	Description
items array	Keys inside the the Kubernetes secret which should be applied. If unspecified, uses all keys.
secretName string	The name of the Kubernetes secret.

.spec.queueManager.availability.tls

Optional TLS settings for configuring secure communication between NativeHA replicas. Requires MQ Operator 1.5.0 or higher.

Appears in:

- ".spec.queueManager.availability" on page 287

Field	Description
cipherSpec string	The name of the CipherSpec for NativeHA TLS.
secretName string	The name of the Kubernetes secret.

.spec.queueManager.env

EnvVar represents an environment variable present in a Container.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
name string	Name of the environment variable. Must be a C_IDENTIFIER.
value string	Variable references $(VAR_NAME) are expanded using the previously defined environment variables in the container and any service environment variables. If a variable cannot be resolved, the reference in the input string will be unchanged. Double $$ are reduced to a single $, which allows for escaping the $(VAR_NAME) syntax: i.e. $$(VAR_NAME) will produce the string literal $(VAR_NAME). Escaped references will never be expanded, regardless of whether the variable exists or not. Defaults to .
valueFrom Valuefrom	Source for the environment variable's value. Cannot be used if value is not empty.

.spec.queueManager.env.valueFrom

Source for the environment variable's value. Cannot be used if value is not empty.

Appears in:

- ".spec.queueManager.env" on page 291

Field	Description
`configMapKeyRef` Configmapkeyref	Selects a key of a ConfigMap.
`fieldRef` Fieldref	Selects a field of the pod: supports metadata.name, metadata.namespace, `metadata.labels['<KEY>']`, `metadata.annotations['<KEY>']`, spec.nodeName, spec.serviceAccountName, status.hostIP, status.podIP, status.podIPs.
`resourceFieldRef` Resourcefieldref	Selects a resource of the container: only resources limits and requests (limits.cpu, limits.memory, limits.ephemeral-storage, requests.cpu, requests.memory and requests.ephemeral-storage) are currently supported.
`secretKeyRef` Secretkeyref	Selects a key of a secret in the pod's namespace.

.spec.queueManager.env.valueFrom.configMapKeyRef

Selects a key of a ConfigMap.

Appears in:

- ".spec.queueManager.env.valueFrom" on page 291

Field	Description
key string	The key to select.
name string	Name of the referent. This field is effectively required, but due to backwards compatibility is allowed to be empty. Instances of this type with an empty value here are almost certainly wrong. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names . Default: "".
`optional` boolean	Specify whether the ConfigMap or its key must be defined.

.spec.queueManager.env.valueFrom.fieldRef

Selects a field of the pod: supports metadata.name, metadata.namespace, `metadata.labels['<KEY>']`, `metadata.annotations['<KEY>']`, spec.nodeName, spec.serviceAccountName, status.hostIP, status.podIP, status.podIPs.

Appears in:

- ".spec.queueManager.env.valueFrom" on page 291

Field	Description
`apiVersion` string	Version of the schema the FieldPath is written in terms of, defaults to v1.
`fieldPath` string	Path of the field to select in the specified API version.

.spec.queueManager.env.valueFrom.resourceFieldRef

Selects a resource of the container: only resources limits and requests (limits.cpu, limits.memory, limits.ephemeral-storage, requests.cpu, requests.memory and requests.ephemeral-storage) are currently supported.

Appears in:

- ".spec.queueManager.env.valueFrom" on page 291

Field	Description
`containerName` string	Container name: required for volumes, optional for env vars.

Field	Description
`divisor`	Specifies the output format of the exposed resources, defaults to 1.
`resource` string	Required: resource to select.

.spec.queueManager.env.valueFrom.secretKeyRef

Selects a key of a secret in the pod's namespace.

Appears in:

- ".spec.queueManager.env.valueFrom" on page 291

Field	Description
`key` string	The key of the secret to select from. Must be a valid secret key.
`name` string	Name of the referent. This field is effectively required, but due to backwards compatibility is allowed to be empty. Instances of this type with an empty value here are almost certainly wrong. More info: https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names . Default: " ".
`optional` boolean	Specify whether the Secret or its key must be defined.

.spec.queueManager.files

Source of files.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
`configMap` Configmap	ConfigMap represents a Kubernetes ConfigMap that contains files.
`secret` Secret	Secret represents a Kubernetes Secret that contains files.

.spec.queueManager.files.configMap

ConfigMap represents a Kubernetes ConfigMap that contains files.

Appears in:

- ".spec.queueManager.files" on page 293

Field	Description
`defaultMountPath` string	The default mount path to use for keys in the Kubernetes source. This can be overridden by specifying individual mount paths for keys. If not specified, `/etc/mqm/customfiles` will be used.
`items` Items array	The keys inside the Kubernetes source which should be applied.
`name` string	The name of the Kubernetes source.

.spec.queueManager.files.configMap.items

Appears in:

- ".spec.queueManager.files.configMap" on page 293

Field	Description
`item` string	The name of the key.
`mountPath` string	The mount path to use for the key, overrides the default mount path.

.spec.queueManager.files.secret

Secret represents a Kubernetes Secret that contains files.

Appears in:

- ".spec.queueManager.files" on page 293

Field	Description
`defaultMountPath` string	The default mount path to use for keys in the Kubernetes source. This can be overridden by specifying individual mount paths for keys. If not specified, `/etc/mqm/customfiles` will be used.
`items` Items array	The keys inside the Kubernetes source which should be applied.
`name` string	The name of the Kubernetes source.

.spec.queueManager.files.secret.items

Appears in:

- ".spec.queueManager.files.secret" on page 294

Field	Description
`item` string	The name of the key.
`mountPath` string	The mount path to use for the key, overrides the default mount path.

.spec.queueManager.ini

Source of INI configuration files.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
`configMap` ConfigMapINISource	ConfigMap represents a Kubernetes ConfigMap that contains INI information.
`secret` SecretINISource	Secret represents a Kubernetes Secret that contains INI information.

.spec.queueManager.ini.configMap

ConfigMap represents a Kubernetes ConfigMap that contains INI information.

Appears in:

- ".spec.queueManager.ini" on page 294

Field	Description
`items` array	The keys inside the Kubernetes source which should be applied.
`name` string	The name of the Kubernetes source.

.spec.queueManager.ini.secret

Secret represents a Kubernetes Secret that contains INI information.

Appears in:

- ".spec.queueManager.ini" on page 294

Field	Description
`items` array	The keys inside the Kubernetes source which should be applied.
`name` string	The name of the Kubernetes source.

.spec.queueManager.livenessProbe

Settings that control the liveness probe.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
`failureThreshold` integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
`initialDelaySeconds` integer	Number of seconds after the container has started before the probe is initiated. Defaults to 90 seconds for SingleInstance. Defaults to 0 seconds for MultiInstance and NativeHA deployments. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .
`periodSeconds` integer	How often (in seconds) to perform the probe. Defaults to 10 seconds.
`successThreshold` integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
`timeoutSeconds` integer	Number of seconds after which the probe times out. Defaults to 5 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .

.spec.queueManager.metrics

Settings for Prometheus-style metrics.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
`enabled` boolean	Whether or not to enable an endpoint for Prometheus-compatible metrics. Defaults to true.
`serviceMonitor` Servicemonitor	Settings for the service monitor for Prometheus-compatible metrics. Requires MQ Operator 3.6.0 or higher.
`tls` Tls	TLS settings for configuring secure communication for the Prometheus-compatible metrics. Requires MQ Operator 3.5.0 or higher.

.spec.queueManager.metrics.serviceMonitor

Settings for the service monitor for Prometheus-compatible metrics. Requires MQ Operator 3.6.0 or higher.

Appears in:

- ".spec.queueManager.metrics" on page 295

Field	Description
`enabled` boolean	Whether or not to enable the service monitor. Defaults to true.

.spec.queueManager.metrics.tls

TLS settings for configuring secure communication for the Prometheus-compatible metrics. Requires MQ Operator 3.5.0 or higher.

Appears in:

- ".spec.queueManager.metrics" on page 295

Field	Description
`provider` string	The provider to use for configuring secure communication for the Prometheus-compatible metrics. Defaults to none. Use none for insecure HTTP metrics. Use `openshift` to use OpenShift-provided certificates with a ServiceMonitor. Use `manual` to provide your own certificates using a Kubernetes secret (requires MQ Operator 3.7.0 or higher).
`secretName` string	The name of the Kubernetes secret to use when provider is `manual`. Requires MQ Operator 3.7.0 or higher.

.spec.queueManager.mqsc

Source of MQSC configuration files.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
`configMap` ConfigMapMQSCSource	ConfigMap represents a Kubernetes ConfigMap that contains MQSC information.
`secret` SecretMQSCSource	Secret represents a Kubernetes Secret that contains MQSC information.

.spec.queueManager.mqsc.configMap

ConfigMap represents a Kubernetes ConfigMap that contains MQSC information.

Appears in:

- ".spec.queueManager.mqsc" on page 296

Field	Description
`items` array	The keys inside the Kubernetes source which should be applied.
`name` string	The name of the Kubernetes source.

.spec.queueManager.mqsc.secret

Secret represents a Kubernetes Secret that contains MQSC information.

Appears in:

- ".spec.queueManager.mqsc" on page 296

Field	Description
items array	The keys inside the Kubernetes source which should be applied.
name string	The name of the Kubernetes source.

.spec.queueManager.readinessProbe

Settings that control the readiness probe.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
failureThreshold integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
initialDelaySeconds integer	Number of seconds after the container has started before the probe is initiated. Defaults to 10 seconds for SingleInstance. Defaults to 0 for MultiInstance and NativeHA deployments. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .
periodSeconds integer	How often (in seconds) to perform the probe. Defaults to 5 seconds.
successThreshold integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
timeoutSeconds integer	Number of seconds after which the probe times out. Defaults to 3 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .

.spec.queueManager.recoveryLogs

Settings for MQ recovery logs. Requires MQ Operator 2.4.0 or higher.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
logFilePages integer	The recovery log data is held in a series of files. The log file size is specified in units of 4 KB pages.

.spec.queueManager.resources

Settings that control resource requirements.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
limits Limits	CPU & memory settings.
requests Requests	CPU & memory settings.

.spec.queueManager.resources.limits

CPU & memory settings.

Appears in:

- “.spec.queueManager.resources” on page 297

Field	Description
cpu	
memory	

.spec.queueManager.resources.requests

CPU & memory settings.

Appears in:

- “.spec.queueManager.resources” on page 297

Field	Description
cpu	
memory	

.spec.queueManager.route

Settings for the Queue Manager route. Requires MQ Operator 1.4.0 or higher.

Appears in:

- “.spec.queueManager” on page 286

Field	Description
enabled boolean	Whether or not to enable the route. Defaults to true.

.spec.queueManager.startupProbe

Settings that control the startup probe. Only applies to MultiInstance and NativeHA deployments. Requires MQ Operator 1.5.0 or higher.

Appears in:

- “.spec.queueManager” on page 286

Field	Description
failureThreshold integer	Minimum consecutive failures for the probe to be considered failed. Defaults to 24.
initialDelaySeconds integer	Number of seconds after the container has started before the probe is initiated. Defaults to 0 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .
periodSeconds integer	How often (in seconds) to perform the probe. Defaults to 5 seconds.
successThreshold integer	Minimum consecutive successes for the probe to be considered successful. Defaults to 1.
timeoutSeconds integer	Number of seconds after which the probe times out. Defaults to 5 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .

.spec.queueManager.storage

Storage settings to control the Queue Manager's use of persistent volumes and storage classes.

Appears in:

- ".spec.queueManager" on page 286

Field	Description
`allowVolumeExpansion` boolean	Whether or not to allow volumes to be expanded.
`defaultClass` string	Storage class to apply to all persistent volumes of this Queue Manager by default. Specific persistent volumes can define their own storage class which will override this default storage class setting. If `type of availability` is `SingleInstance` or `NativeHA`, storage class can be of type ReadWriteOnce or ReadWriteMany. If `type of availability` is `MultiInstance`, storage class must be of type ReadWriteMany.
`defaultDeleteClaim` boolean	Whether or not all the volumes should be deleted when the Queue Manager is deleted. Specific persistent volumes can define their own value for deleteClaim which will override this defaultDeleteClaim setting. Defaults to false.
`persistedData` QueueManagerOptionalVolume	PersistentVolume details for MQ persisted data, including configuration, queues and messages. Required when using multi-instance Queue Manager.
`queueManager` QueueManagerVolume	Default PersistentVolume for any data normally under `/var/mqm`. Will contain all persisted data and recovery logs, if no other volumes are specified.
`recoveryLogs` QueueManagerOptionalVolume	Persistent volume details for MQ recovery logs. Required when using multi-instance Queue Manager.
`scratch` Scratch	Settings for the Queue Manager's Scratch ephemeral volume. These settings, combined with the read-only root filesystem settings, determine where ephemeral data written to '/run' is stored. Requires MQ Operator 3.0.0 or higher.
`tmp` Tmp	Settings for the Queue Manager's Tmp ephemeral volume. These settings, combined with the read-only root filesystem settings, determine where ephemeral data written to '/tmp' is stored. The diagnostic data files, such as the zip file produced by the runmqras command, will be created in this volume. Requires MQ Operator 3.0.0 or higher.

.spec.queueManager.storage.persistedData

PersistentVolume details for MQ persisted data, including configuration, queues and messages. Required when using multi-instance Queue Manager.

Appears in:

- ".spec.queueManager.storage" on page 298

Field	Description
`class` string	Storage class to use for this volume. Only valid if `type` is `persistent-claim`. If `type of availability` is `SingleInstance` or `NativeHA`, storage class can be of type ReadWriteOnce or ReadWriteMany. If `type of availability` is `MultiInstance`, storage class must be of type ReadWriteMany.
`deleteClaim` boolean	Whether or not this volume should be deleted when the Queue Manager is deleted.

Field	Description
enabled boolean	Whether or not this volume should be enabled as a separate volume, or be placed on the default queueManager volume. Defaults to false.
size string	Size of the PersistentVolume to pass to Kubernetes, including SI units. Only valid if type is persistent-claim. For example, 2Gi. Defaults to 2Gi.
sizeLimit string	Size limit when using an ephemeral volume. Files are still written to a temporary directory, so you can use this option to limit the size. Only valid if type is ephemeral and root filesystem is set to read-only. Requires MQ Operator 3.0.0 or higher.
type string	Type of volume to use. Choose ephemeral to use non-persistent storage, or persistent-claim to use a persistent volume. Defaults to persistent-claim.

.spec.queueManager.storage.queueManager

Default PersistentVolume for any data normally under /var/mqm. Will contain all persisted data and recovery logs, if no other volumes are specified.

Appears in:

- ".spec.queueManager.storage" on page 298

Field	Description
class string	Storage class to use for this volume. Only valid if type is persistent-claim. If type of availability is NativeHA, storage class must be of type ReadWriteOnce. If type of availability is MultiInstance, storage class must be of type ReadWriteMany.
deleteClaim boolean	Whether or not this volume should be deleted when the Queue Manager is deleted.
size string	Size of the PersistentVolume to pass to Kubernetes, including SI units. Only valid if type is persistent-claim. For example, 2Gi. Defaults to 2Gi.
sizeLimit string	Size limit when using an ephemeral volume. Files are still written to a temporary directory, so you can use this option to limit the size. Only valid if type is ephemeral and root filesystem is set to read-only. Requires MQ Operator 3.0.0 or higher.
type string	Type of volume to use. Choose ephemeral to use non-persistent storage, or persistent-claim to use a persistent volume. Defaults to persistent-claim.

.spec.queueManager.storage.recoveryLogs

Persistent volume details for MQ recovery logs. Required when using multi-instance Queue Manager.

Appears in:

- ".spec.queueManager.storage" on page 298

Field	Description
class string	Storage class to use for this volume. Only valid if type is persistent-claim. If type of availability is SingleInstance or NativeHA, storage class can be of type ReadWriteOnce or ReadWriteMany. If type of availability is MultiInstance, storage class must be of type ReadWriteMany.

Field	Description
`deleteClaim` boolean	Whether or not this volume should be deleted when the Queue Manager is deleted.
`enabled` boolean	Whether or not this volume should be enabled as a separate volume, or be placed on the default `queueManager` volume. Defaults to false.
`size` string	Size of the PersistentVolume to pass to Kubernetes, including SI units. Only valid if `type` is `persistent-claim`. For example, `2Gi`. Defaults to `2Gi`.
`sizeLimit` string	Size limit when using an `ephemeral` volume. Files are still written to a temporary directory, so you can use this option to limit the size. Only valid if `type` is `ephemeral` and root filesystem is set to read-only. Requires MQ Operator 3.0.0 or higher.
`type` string	Type of volume to use. Choose `ephemeral` to use non-persistent storage, or `persistent-claim` to use a persistent volume. Defaults to `persistent-claim`.

.spec.queueManager.storage.scratch

Settings for the Queue Manager's Scratch ephemeral volume. These settings, combined with the read-only root filesystem settings, determine where ephemeral data written to '/run' is stored. Requires MQ Operator 3.0.0 or higher.

Appears in:

- ".spec.queueManager.storage" on page 298

Field	Description
`enabled` boolean	Whether or not data is written to non-persistent storage (true), or to the default `queueManager` volume (false). When set to false, ReadWriteOnce storage should be used for the default `queueManager` volume as SELinux may prevent sockets being written to other types of storage. Defaults to true. Requires MQ Operator 3.9.0 or higher.
`sizeLimit` string	Size limit of the Scratch ephemeral volume, including SI units. For example, `100M`. Defaults to `100M` if not specified. Valid only when root filesystem is set to read-only and the Scratch ephemeral volume is enabled. Requires MQ Operator 3.0.0 or higher.

.spec.queueManager.storage.tmp

Settings for the Queue Manager's Tmp ephemeral volume. These settings, combined with the read-only root filesystem settings, determine where ephemeral data written to '/tmp' is stored. The diagnostic data files, such as the zip file produced by the runmqras command, will be created in this volume. Requires MQ Operator 3.0.0 or higher.

Appears in:

- ".spec.queueManager.storage" on page 298

Field	Description
`enabled` boolean	Whether or not data is written to non-persistent storage (true), or to the default `queueManager` volume (false). Defaults to true. Requires MQ Operator 3.9.0 or higher.

Field	Description
`sizeLimit` string	Size limit of the Tmp ephemeral volume, including SI units. For example, `2Gi`. Defaults to `2Gi` if not specified. Valid only when root filesystem is set to read-only and the Tmp ephemeral volume is enabled. Requires MQ Operator 3.0.0 or higher.

.spec.securityContext

Security settings to add to the Queue Manager Pod's securityContext.

Appears in:

- ".spec" on page 283

Field	Description
`fsGroup` integer	A special supplemental group that applies to all containers in a pod. Some volume types allow the Kubelet to change the ownership of that volume to be owned by the pod: 1. The owning GID will be the FSGroup 2. The setgid bit is set (new files created in the volume will be owned by FSGroup) 3. The permission bits are OR'd with 0660. If unset, the Kubelet will not modify the ownership and permissions of any volume.
`initVolumeAsRoot` boolean	This affects the securityContext used by the container which initializes the PersistentVolume. Set this to `true` if you are using a storage provider which requires you to be the root user to access newly provisioned volumes. Setting this to `true` affects which Security Context Constraints (SCC) object you can use, and the Queue Manager may fail to start if you are not authorized to use an SCC which allows the root user. Defaults to false. For more information, see https://docs.openshift.com/container-platform/latest/authentication/managing-security-context-constraints.html .
`readOnlyRootFilesystem` boolean	Whether or not to enable read-only root filesystem settings for the Queue Manager. Defaults to false. Requires MQ Operator 3.0.0 or higher.
`supplementalGroups` array	A list of groups applied to the first process run in each container, in addition to the container's primary GID. If unspecified, no groups will be added to any container.

.spec.service

Settings that apply to all QueueManager services. Modification or removal of this section or the fields within it may result in the regeneration of all QueueManager services. Service regeneration will disrupt communication resulting in downtime for all QueueManager types including NativeHA. Regenerated Services are assigned new ClusterIP addresses by Kubernetes. Requires MQ Operator 3.6.0 or higher.

Appears in:

- ".spec" on page 283

Field	Description
`ipFamilies` array	Specifies the IP Families to be used for all QueueManager Services. Valid options are IPv4 and IPv6. If unspecified Kubernetes will set a default value based on the ipFamilyPolicy setting. If set manually Service creation will fail if the IP Family is not available on the cluster, or the values are incompatible with the ipFamilyPolicy setting. Modifications to the first value in the array (the primary IP Family) or removal of this field will result in the regeneration of all QueueManager services. Service regeneration will disrupt communication resulting in downtime for all QueueManager types including NativeHA. Regenerated Services are assigned new ClusterIP addresses by Kubernetes.
`ipFamilyPolicy` string	Specifies the IP Family Policy to be used for all QueueManager Services. Valid options are SingleStack, PreferDualStack and RequireDualStack. SingleStack is for a single IP Family. PreferDualStack is for two IP families on dual-stack configured clusters or a single IP family on single-stack clusters. RequireDualStack is for two IP families on dual-stack configured clusters, otherwise fail. Removal of this field will result in the regeneration of all QueueManager services. Service regeneration will disrupt communication resulting in downtime for all QueueManager types including NativeHA. Regenerated Services are assigned new ClusterIP addresses by Kubernetes.

.spec.telemetry

Settings for Open Telemetry configuration. Requires MQ Operator 2.2.0 or higher.

Appears in:

- ".spec" on page 283

Field	Description
`tracing` Tracing	Settings for Open Telemetry tracing.

.spec.telemetry.tracing

Settings for Open Telemetry tracing.

Appears in:

- ".spec.telemetry" on page 303

Field	Description
`instana` Instana	Settings for Instana tracing.

.spec.telemetry.tracing.instana

Settings for Instana tracing.

Appears in:

- ".spec.telemetry.tracing" on page 303

Field	Description
`agentHost` string	The hostname of the Instana agent to send tracing data to. This should not include a protocol.
`enabled` boolean	Whether or not to enable Instana tracing. Defaults to false.

Field	Description
`protocol` string	The protocol to be used in communication with the Instana agent. http and https are supported.

.spec.template

Advanced templating for Kubernetes resources. The template allows users to override how IBM MQ generates the underlying Kubernetes resources, such as StatefulSet, Pods and Services. This is for advanced users only, as it has the potential to disrupt normal operation of MQ if used incorrectly. Any values specified anywhere else in the QueueManager resource will be overridden by settings in the template.

Appears in:

- ".spec" on page 283

Field	Description
pod	Overrides for the template used for the Pod. See https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#podspec-v1-core .

.spec.tracing

Settings for tracing integration with the Cloud Pak for Integration Operations Dashboard.

Appears in:

- ".spec" on page 283

Field	Description
`agent` TracingAgent	In Cloud Pak for Integration only, you can configure settings for the optional Tracing Agent.
`collector` TracingCollector	In Cloud Pak for Integration only, you can configure settings for the optional Tracing Collector.
`enabled` boolean	Whether or not to enable integration with the Cloud Pak for Integration Operations Dashboard, via tracing. Defaults to false.
`namespace` string	Namespace where the Cloud Pak for Integration Operations Dashboard is installed.

.spec.tracing.agent

In Cloud Pak for Integration only, you can configure settings for the optional Tracing Agent.

Appears in:

- ".spec.tracing" on page 304

Field	Description
`image` string	The container image that will be used.
`imagePullPolicy` string	Setting that controls when the kubelet attempts to pull the specified image. Defaults to `IfNotPresent`.
`livenessProbe` TracingProbe	Settings that control the liveness probe.
`readinessProbe` TracingProbe	Settings that control the readiness probe.

.spec.tracing.agent.livenessProbe

Settings that control the liveness probe.

Appears in:

- ".spec.tracing.agent" on page 304

Field	Description
`failureThreshold` integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
`initialDelaySeconds` integer	Number of seconds after the container has started before liveness probes are initiated. Defaults to 10 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .
`periodSeconds` integer	How often (in seconds) to perform the probe. Defaults to 10 seconds.
`successThreshold` integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
`timeoutSeconds` integer	Number of seconds after which the probe times out. Defaults to 2 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .

.spec.tracing.agent.readinessProbe

Settings that control the readiness probe.

Appears in:

- ".spec.tracing.agent" on page 304

Field	Description
`failureThreshold` integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
`initialDelaySeconds` integer	Number of seconds after the container has started before liveness probes are initiated. Defaults to 10 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .
`periodSeconds` integer	How often (in seconds) to perform the probe. Defaults to 10 seconds.
`successThreshold` integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
`timeoutSeconds` integer	Number of seconds after which the probe times out. Defaults to 2 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .

.spec.tracing.collector

In Cloud Pak for Integration only, you can configure settings for the optional Tracing Collector.

Appears in:

- ".spec.tracing" on page 304

Field	Description
`image` string	The container image that will be used.
`imagePullPolicy` string	Setting that controls when the kubelet attempts to pull the specified image. Defaults to `IfNotPresent`.

Field	Description
`livenessProbe` TracingProbe	Settings that control the liveness probe.
`readinessProbe` TracingProbe	Settings that control the readiness probe.

.spec.tracing.collector.livenessProbe

Settings that control the liveness probe.

Appears in:

- ".spec.tracing.collector" on page 305

Field	Description
`failureThreshold` integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
`initialDelaySeconds` integer	Number of seconds after the container has started before liveness probes are initiated. Defaults to 10 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .
`periodSeconds` integer	How often (in seconds) to perform the probe. Defaults to 10 seconds.
`successThreshold` integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
`timeoutSeconds` integer	Number of seconds after which the probe times out. Defaults to 2 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .

.spec.tracing.collector.readinessProbe

Settings that control the readiness probe.

Appears in:

- ".spec.tracing.collector" on page 305

Field	Description
`failureThreshold` integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
`initialDelaySeconds` integer	Number of seconds after the container has started before liveness probes are initiated. Defaults to 10 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .
`periodSeconds` integer	How often (in seconds) to perform the probe. Defaults to 10 seconds.
`successThreshold` integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
`timeoutSeconds` integer	Number of seconds after which the probe times out. Defaults to 2 seconds. More info: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes .

.spec.web

Settings for the MQ web server.

Appears in:

- ".spec" on page 283

Field	Description
`console` Console	Settings for the MQ web console. Requires MQ Operator 3.0.0 or higher.
`enabled` boolean	Whether or not to enable the web server. Defaults to false.
`manualConfig` Manualconfig	Settings for supplying web server XML configuration. Requires MQ Operator 3.0.0 or higher.
`route` Route	Settings for the web server route. Requires MQ Operator 3.6.0 or higher.

.spec.web.console

Settings for the MQ web console. Requires MQ Operator 3.0.0 or higher.

Appears in:

- ".spec.web" on page 306

Field	Description
`authentication` Authentication	Authentication settings for the MQ web console. Requires MQ Operator 3.0.0 or higher.
`authorization` Authorization	Authorization settings for the MQ web console. Requires MQ Operator 3.0.0 or higher.

.spec.web.console.authentication

Authentication settings for the MQ web console. Requires MQ Operator 3.0.0 or higher.

Appears in:

- ".spec.web.console" on page 307

Field	Description
`provider` string	The authentication provider to use for the MQ web console. Use `integration-keycloak` to use single sign-on with the Cloud Pak for Integration Platform UI (Keycloak). Defaults to `integration-keycloak` if you use a Cloud Pak for Integration license, or `manual` if you use an MQ license. Use `manual` if you want to provide your own configuration.

.spec.web.console.authorization

Authorization settings for the MQ web console. Requires MQ Operator 3.0.0 or higher.

Appears in:

- ".spec.web.console" on page 307

Field	Description
`provider` string	The authorization provider to use for the MQ web console. Use `integration-keycloak` to use roles provided by the Cloud Pak for Integration Keycloak. Use `manual` if you want to provide your own configuration. Defaults to `integration-keycloak` if you use a Cloud Pak for Integration license, or `manual` if you use an MQ license.

.spec.web.manualConfig

Settings for supplying web server XML configuration. Requires MQ Operator 3.0.0 or higher.

Appears in:

- ".spec.web" on page 306

Field	Description
`configMap` Configmap	ConfigMap represents a Kubernetes ConfigMap that contains web server XML configuration.
`secret` Secret	Secret represents a Kubernetes Secret that contains web server XML configuration. Using a Secret protects any credentials in the Kubernetes layer, but it's possible that monitoring or troubleshooting tools might expose the underlying file insecurely. For improved security, encode credentials using "securityUtility".

.spec.web.manualConfig.configMap

ConfigMap represents a Kubernetes ConfigMap that contains web server XML configuration.

Appears in:

- ".spec.web.manualConfig" on page 308

Field	Description
`name` string	The name of the Kubernetes source.

.spec.web.manualConfig.secret

Secret represents a Kubernetes Secret that contains web server XML configuration. Using a Secret protects any credentials in the Kubernetes layer, but it's possible that monitoring or troubleshooting tools might expose the underlying file insecurely. For improved security, encode credentials using "securityUtility".

Appears in:

- ".spec.web.manualConfig" on page 308

Field	Description
`name` string	The name of the Kubernetes source.

.spec.web.route

Settings for the web server route. Requires MQ Operator 3.6.0 or higher.

Appears in:

- ".spec.web" on page 306

Field	Description
`enabled` boolean	Whether or not to enable the route. Defaults to true.

.status

The observed state of the QueueManager.

Appears in:

- "QueueManager" on page 282

Field	Description
`adminUiUrl` string	URL for the Admin UI.
`availability` Availability	Availability status for the Queue Manager.
`conditions` QueueManagerStatusCondition array	Conditions represent the latest available observations of the Queue Manager's state.
`customImages` boolean	Whether or not the operator selected Queue Manager image has been overridden with a custom image.
`endpoints` QueueManagerStatusEndpoint array	Information on the endpoints that this Queue Manager is exposing, such as API or UI endpoints.
`metadata` Metadata	Metadata represents additional information for the Queue Manager, including Integration-Keycloak status.
`name` string	The name of the Queue Manager.
`nativeHACRR` Nativehacrr	Information on the configured Native HA Cross-Region Replication.
`phase` string	Phase of the Queue Manager's state.
`versions` QueueManagerStatusVersion	Version of MQ being used, and other versions available from the IBM Entitled Registry.

.status.availability

Availability status for the Queue Manager.

Appears in:

- ".status" on page 308

Field	Description
`initialQuorumEstablished` boolean	Whether or not an initial quorum has been established for NativeHA.

.status.conditions

QueueManagerStatusCondition defines the conditions of the Queue Manager.

Appears in:

- ".status" on page 308

Field	Description
`lastTransitionTime` string	Last time the condition transitioned from one status to another.
`message` string	Human-readable message indicating details about last transition.
`reason` string	Reason for last transition of this status.
`status` string	Status of the condition. Valid values are: `"True"; "False"; "Unknown"`.
`type` string	Type of condition.

.status.endpoints

QueueManagerStatusEndpoint defines the endpoints for the QueueManager.

Appears in:

- ".status" on page 308

Field	Description
`name` string	Name of the endpoint.
`type` string	The type of the endpoint, for example 'UI' for a UI endpoint, 'API' for an API endpoint, 'OpenAPI' for API documentation.
`uri` string	URI for the endpoint.

.status.metadata

Metadata represents additional information for the Queue Manager, including Integration-Keycloak status.

Appears in:

- ".status" on page 308

Field	Description
`integrationKeycloak` Integrationkeycloak	QueueManagerStatusIntegrationKeycloak defines the Integration-Keycloak status for the QueueManager.
`lastValidConfiguration` string	

.status.metadata.integrationKeycloak

QueueManagerStatusIntegrationKeycloak defines the Integration-Keycloak status for the QueueManager.

Appears in:

- ".status.metadata" on page 310

Field	Description
`clientName` string	

.status.nativeHACRR

Information on the configured Native HA Cross-Region Replication.

Appears in:

- ".status" on page 308

Field	Description
`local` Local	Local cluster status.

.status.nativeHACRR.local

Local cluster status.

Appears in:

- ".status.nativeHACRR" on page 310

Field	Description
`address` string	Address of the local cluster.

Field	Description
groupName string	Group name of the local cluster.

.status.versions

Version of MQ being used, and other versions available from the IBM Entitled Registry.

Appears in:

- ".status" on page 308

Field	Description
available QueueManagerStatusVersionAvailable	Other versions of MQ available from the IBM Entitled Registry.
reconciled string	The specific version of IBM MQ being used. If a custom image is specified, then this may not match the version of MQ actually being used.

.status.versions.available

Other versions of MQ available from the IBM Entitled Registry.

Appears in:

- ".status.versions" on page 311

Field	Description
channels array	Channels which are available for automatically updating the MQ version.
versions Versions array	Specific versions of MQ which are available.

.status.versions.available.versions

QueueManagerStatusVersion defines a version of MQ.

Appears in:

- ".status.versions.available" on page 311

Field	Description
licenses Licenses array	Licenses that are applicable for this version of QueueManager.
name string	Version name for this version of QueueManager. These are valid values for the spec.version field.

.status.versions.available.versions.licenses

QueueManagerStatusLicense defines a license.

Appears in:

- ".status.versions.available.versions" on page 311

Field	Description
displayName string	Display name for the license.
link string	Link to the license content.

Field	Description
`matchesCurrentType` boolean	Whether or not the license matches the type of license currently used.
`name` string	Name of the license.

Status conditions for QueueManager (mq.ibm.com/v1beta1)

The **`status.conditions`** fields are updated to reflect the condition of the `QueueManager` resource. In general, conditions describe abnormal situations. A queue manager in a healthy, ready state has no **`Error`** or **`Pending`** conditions, though it might have some advisory **`Warning`** conditions.

The following conditions are defined for a `QueueManager` resource. For each of three components (QueueManager, Pod, and Storage), a table lists the possible condition types and the associated reason code and message text.

QueueManager

The conditions `Creating` and `Failed` monitor the overall progress of the deployment of the queue manager. If you are using an IBM Cloud Pak for Integration license and the web console is enabled, then the `OidcPending` condition logs the status of the queue manager while waiting for OIDC client registration to complete with IAM.

Table 5. Status conditions for the QueueManager component

Condition type	Reason code	Message warnings
Blocked	OperatorDependency	• To install, this instance requires Keycloak to be configured by [IBM Cloud Pak for Integration]. This instance will remain in [Pending] status until Keycloak is reported as [KeycloakReady] in the Cp4iServicesBinding resource for this QueueManager. • To install, this instance requires the operator [IBM IAM]. This instance will remain in [Blocked] status until the operator is installed by [IBM Cloud Pak foundational services]. • To install, this instance requires API [IBMLicensingDefinition] which is provided by the operator [IBM Licensing]. This instance will remain in [Blocked] status until the required operator is installed.
Pending	Creating	MQ queue manager is being deployed
Pending	OidcPending	MQ queue manager is waiting for OIDC client registration
Pending	Stopped	The MQ queue manager has been stopped as the 'mq.ibm.com/stop' annotation is present and set to 'true' in the QueueManager definition. When stopped the QueueManager StatefulSet replica count is set to zero, removing all of the MQ queue manager pods.

Table 5. Status conditions for the QueueManager component (continued)		
Condition type	**Reason code**	**Message warnings**
Error	Failed	MQ queue manager failed to deploy
Warning	UnsupportedVersion	An operand has been installed by an operator which is not supported on OCP version *<ocp_version>*. This operand is not supported.
Warning	CP4I-LTS Support	A CP4I-LTS operand *<mq_version>* has been installed but is being managed by an operator that does not qualify for the extended support duration. This operand does not qualify for the extended support duration.
Warning	CP4I-LTS Support	A CP4I-LTS operand *<mq_version>* has been installed but the OCP version *<ocp_version>* does not qualify for the extended support duration. This operand does not qualify for the extended support duration.

Pod

Pod conditions monitor the status of pods during the deployment of a queue manager. If you see any `PodFailed` condition, then the overall queue manager condition will also be set to `Failed`.

Table 6. Status conditions for the Pod component		
Condition type	**Reason code**	**Message warning**
Pending	PodPending	Pod for MQ queue manager is being deployed
Error	PodFailed	Pod for MQ queue manager is being deployed

Storage

Storage conditions monitor the progress (`StoragePending` condition) of requests to create volumes for persistent storage, and report back binding errors and other failures. Storage conditions also monitor the progress of volume expansions, and warn of mismatches between storage sizes defined in the Queue Manager definition and the size of deployed PVCs. If any error occurs during storage provisioning, the `StorageFailed` condition is added to the conditions list, and the overall queue manager condition is set to `Failed`.

Table 7. Status conditions for the Storage component		
Condition type	**Reason code**	**Message warning**
Pending	StoragePending	Storage for MQ queue manager is being provisioned
Warning	StorageEphemeral	Using ephemeral storage for a production MQ queue manager
Warning	StorageExpansionPending	Volume expansion is pending for the following PVCs [<list of pvcs>]

Table 7. Status conditions for the Storage component (continued)		
Condition type	**Reason code**	**Message warning**
Warning	StorageMismatch	Storage sizes defined in the QueueManager resource do not match the capacity of one or more provisioned PVCs [<list of pvcs>]. AllowVolumeExpansion is set to false in the QueueManager resource so the MQ Operator will not attempt to reconcile these differences.
Error	StorageFailed	Storage for MQ queue manager failed to provision

Linux IBM MQ container license annotations

In Kubernetes and Red Hat OpenShift Container Platform environments, the IBM License Service uses Pod annotations to track usage based on the limits defined on the container, rather than on the underlying machine. The IBM MQ Operator sets these Pod annotations for you, but when you deploy using other means, you must set them yourself.

For information about how the IBM MQ Operator annotates pods based on the values of a queue manager's YAML, see “IBM MQ container license annotations that are applied by the IBM MQ Operator” on page 319.

When deploying an IBM MQ container, there are two common approaches to licensing:

- License the entire machine running the container.
- License the container based on the associated limits (Kubernetes and Red Hat OpenShift Container Platform only)

Both options are available to customers, and further details can be found on the IBM Container Licenses page on Passport Advantage®.

If the IBM MQ container is to be licensed based on the container limits, then the IBM License Service needs to be installed to track usage. Further information regarding the supported environments and installation instructions can be found on the ibm-licensing-operator page on GitHub.

The IBM License Service is installed on the Kubernetes cluster where the IBM MQ container is deployed, and pod annotations are used to track usage. Therefore clients need to deploy the pod with specific annotations that the IBM License Service then uses. Based on your entitlement and capabilities deployed within the container, use one or more of the following annotations.

Note: Many of the annotations contain one or both of the following lines:

```
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
```

You must edit these lines before using the annotation:

- For `productChargedContainers`, you must choose `"All"`, or substitute the actual name of the container.
- For `productMetric`, you must choose one of the values offered.

Annotations to use with an IBM MQ product entitlement

If you have an IBM MQ product entitlement, select the annotation below that matches the entitlement you have purchased and want to use.

- “IBM MQ” on page 316
- “IBM MQ Advanced” on page 317
- “IBM MQ for Non-Production Environment” on page 317

- "IBM MQ Advanced for Non-Production Environment" on page 317
- "IBM MQ Advanced for Developers" on page 317

The IBM MQ annotations to use with IBM MQ Multi Instance High Availability configurations are as follows. See also "Selecting the correct annotations for High Availability configurations" on page 315.

- "IBM MQ Container Multi Instance" on page 317
- "IBM MQ Advanced Container Multi Instance" on page 317
- "IBM MQ Container Multi Instance for Non-Production Environment" on page 317
- "IBM MQ Advanced Container Multi Instance for Non-Production Environment" on page 317

Annotations to use with CP4I product entitlement

If you have IBM Cloud Pak for Integration (CP4I) entitlement select the annotation below that matches the entitlement you have purchased and want to use.

- "IBM MQ with CP4I entitlement" on page 318
- "IBM MQ Advanced with CP4I entitlement" on page 318
- "IBM MQ for Non-Production Environment with CP4I entitlement" on page 318
- "IBM MQ Advanced for Non-Production Environment with CP4I entitlement" on page 318

The CP4I annotations to use with IBM MQ Multi Instance High Availability configurations are as follows. See also "Selecting the correct annotations for High Availability configurations" on page 315.

- "IBM MQ Container Multi Instance with CP4I entitlement" on page 318
- "IBM MQ Advanced Container Multi Instance with CP4I entitlement" on page 318
- "IBM MQ Container Multi Instance for Non-Production Environment with CP4I entitlement" on page 319
- "IBM MQ Advanced Container Multi Instance for Non-Production Environment with CP4I entitlement" on page 319

Selecting the correct annotations for High Availability configurations

IBM MQ Multi Instance

When you deploy a multi-instance queue manager, you should use the same annotations on both the active and standby instances. One of the following annotations should be selected, depending on the entitlement purchased:

- IBM MQ or IBM MQ Advanced standalone entitlement
  - "IBM MQ Container Multi Instance" on page 317
  - "IBM MQ Advanced Container Multi Instance" on page 317
  - "IBM MQ Container Multi Instance for Non-Production Environment" on page 317
  - "IBM MQ Advanced Container Multi Instance for Non-Production Environment" on page 317
- IBM Cloud Pak for Integration entitlement
  - "IBM MQ Container Multi Instance with CP4I entitlement" on page 318
  - "IBM MQ Advanced Container Multi Instance with CP4I entitlement" on page 318
  - "IBM MQ Container Multi Instance for Non-Production Environment with CP4I entitlement" on page 319
  - "IBM MQ Advanced Container Multi Instance for Non-Production Environment with CP4I entitlement" on page 319

When used with IBM Cloud Pak for Integration entitlement, the entitlement ratios in the annotations ensure that correct entitlement consumption is recorded. When used with standalone IBM MQ or IBM

MQ Advanced entitlements, the annotations reported in the License Service for each instance need to be mapped to the IBM MQ entitlement parts as follows:

- IBM MQ Advanced Container Multi Instance
  - 1 x IBM MQ Advanced **and** 1 x IBM MQ Advanced High Availability Replica **or**
  - 2 x IBM MQ Advanced[1]
- IBM MQ Advanced Container Multi Instance for Non-Production Environment
  - 1 x IBM MQ Advanced **and** 1 x IBM MQ Advanced High Availability Replica **or**
  - 2 x IBM MQ Advanced for Non-Production Environment)[1]
- IBM MQ Container Multi Instance
  - 1 x IBM MQ **and** 1 x IBM MQ High Availability Replica **or**
  - 2 x IBM MQ[1]
- IBM MQ Container Multi Instance for Non-Production Environment
  - 1 x IBM MQ **and** 1 x IBM MQ High Availability Replica **or**
  - 2 x IBM MQ for Non-Production Environment) [1]

IBM MQ Native HA

If you are deploying a Native HA queue manager, only the active instance consumes entitlement, the two replica instances do not. All instances should have the same annotation. One of the following should be selected, depending on the entitlement purchased:

- IBM MQ or IBM MQ Advanced standalone entitlement
  - “IBM MQ Advanced” on page 317
  - “IBM MQ Advanced for Non-Production Environment” on page 317
- IBM Cloud Pak for Integration entitlement
  - “IBM MQ Advanced with CP4I entitlement” on page 318
  - “IBM MQ Advanced for Non-Production Environment with CP4I entitlement” on page 318

Annotations

The rest of this topic details the contents of each annotation.

IBM MQ

```
productID: "c661609261d5471fb4ff8970a36bccea"
productName: "IBM MQ"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

To use Native HA and Cross-Region Replication features on queue managers that use these license annotations, see the topic that matches your environment:

- “Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator” on page 124
- “Configuring Native HA and Cross-Region Replication on IBM MQ licensed self-deployed queue managers in Kubernetes” on page 213

[1] This entitlement option is sub optimal and should only be used if no entitlement of the relevant High Availability Replica part is available.

IBM MQ Advanced

```
productID: "208423bb063c43288328b1d788745b0c"
productName: "IBM MQ Advanced"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

IBM MQ for Non-Production Environment

```
productID: "151bec68564a4a47a14e6fa99266deff"
productName: "IBM MQ for Non-Production Environment"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

To use Native HA and Cross-Region Replication features on queue managers that use these license annotations, see the topic that matches your environment:

- "Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator" on page 124
- "Configuring Native HA and Cross-Region Replication on IBM MQ licensed self-deployed queue managers in Kubernetes" on page 213

IBM MQ Advanced for Non-Production Environment

```
productID: "21dfe9a0f00f444f888756d835334909"
productName: "IBM MQ Advanced for Non-Production Environment"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

IBM MQ Advanced for Developers

```
productID: "2f886a3eefbe4ccb89b2adb97c78b9cb"
productName: "IBM MQ Advanced for Developers (Non-Warranted)"
productMetric: "FREE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

IBM MQ Container Multi Instance

```
productID: "2dea73b866b648b6b4abe2a85eb76964"
productName: "IBM MQ Container Multi Instance"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

IBM MQ Advanced Container Multi Instance

```
productID: "bd35bff411bb47c2a3f3a4590f33a8ef"
productName: "IBM MQ Advanced Container Multi Instance"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

IBM MQ Container Multi Instance for Non-Production Environment

```
productID: "af11b093f16a4a26806013712b860b60"
productName: "IBM MQ Container Multi Instance for Non-Production Environment"
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

IBM MQ Advanced Container Multi Instance for Non-Production Environment

```
productID: "31f844f7a96b49749130cd0708fdbb17"
productName: "IBM MQ Advanced Container Multi Instance for Non-Production Environment"
```

```
productMetric: "PROCESSOR_VALUE_UNIT" | "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
```

IBM MQ with CP4I entitlement

```
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
cloudpakName: "IBM Cloud Pak for Integration"
productID: "c661609261d5471fb4ff8970a36bccea"
productName: "IBM MQ"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productCloudpakRatio: "4:1"
```

IBM MQ Advanced with CP4I entitlement

```
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
cloudpakName: "IBM Cloud Pak for Integration"
productID: "208423bb063c43288328b1d788745b0c"
productName: "IBM MQ Advanced"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productCloudpakRatio: "2:1"
```

IBM MQ for Non-Production Environment with CP4I entitlement

```
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
cloudpakName: "IBM Cloud Pak for Integration"
productID: "151bec68564a4a47a14e6fa99266deff"
productName: "IBM MQ for Non-Production Environment"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers:  "All" | "NAME_OF_CONTAINER"
productCloudpakRatio: "8:1"
```

IBM MQ Advanced for Non-Production Environment with CP4I entitlement

```
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
cloudpakName: "IBM Cloud Pak for Integration"
productID: "21dfe9a0f00f444f888756d835334909"
productName: "IBM MQ Advanced for Non-Production Environment"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productCloudpakRatio: "4:1"
```

IBM MQ Container Multi Instance with CP4I entitlement

```
productName: "IBM MQ Container Multi Instance"
productID: "2dea73b866b648b6b4abe2a85eb76964"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productCloudpakRatio: "10:3"
cloudpakName: "IBM Cloud Pak for Integration"
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
```

IBM MQ Advanced Container Multi Instance with CP4I entitlement

```
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
cloudpakName: "IBM Cloud Pak for Integration"
productID: "bd35bff411bb47c2a3f3a4590f33a8ef"
productName: "IBM MQ Advanced Container Multi Instance"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productCloudpakRatio: "5:3"
```

IBM MQ Container Multi Instance for Non-Production Environment with CP4I entitlement

```
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
cloudpakName: "IBM Cloud Pak for Integration"
productID: "af11b093f16a4a26806013712b860b60"
productName: "IBM MQ Container Multi Instance for Non-Production Environment"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productCloudpakRatio: "20:3"
```

IBM MQ Advanced Container Multi Instance for Non-Production Environment with CP4I entitlement

```
cloudpakId: "c8b82d189e7545f0892db9ef2731b90d"
cloudpakName: "IBM Cloud Pak for Integration"
productID: "31f844f7a96b49749130cd0708fdbb17"
productName: "IBM MQ Advanced Container Multi Instance for Non-Production Environment"
productMetric: "VIRTUAL_PROCESSOR_CORE"
productChargedContainers: "All" | "NAME_OF_CONTAINER"
productCloudpakRatio: "10:3"
```

Linux IBM MQ container license annotations that are applied by the IBM MQ Operator

In Kubernetes and Red Hat OpenShift Container Platform environments, the IBM License Service uses Pod annotations to track usage based on the limits defined on the container, rather than on the underlying machine. The IBM MQ Operator sets these Pod annotations for you, but when you deploy using other means, you must set them yourself.

For the available annotations, see “IBM MQ container license annotations” on page 314.

Note: The tables in this topic refer to the latest CD licenses, which are relevant to IBM MQ 10.0.0.0. For licenses for other IBM MQ versions, see “Licensing and API references for the IBM MQ Operator” on page 281.

The following table describes which of the available annotations are applied to a pod by the IBM MQ Operator, based on these values in a queue manager's YAML:

- The value of `spec.license.license`
- The value of `spec.license.use`
- The value of `spec.queueManager.availability.type`

Annotation	**Value of `spec.license.license`**	**Value of `spec.license.use`**	**Value of `spec.queueManager.availability.type`**
IBM MQ Advanced	L-CXDB-XTZ9J8	`Production`	`SingleInstance` or `NativeHA`
IBM MQ Advanced for Non-Production Environment	L-UMVX-XYNRAD	`NonProduction`	`SingleInstance` or `NativeHA`
IBM MQ Advanced for Developers	L-GWLM-M6WVZU	`Development`	`SingleInstance` or `NativeHA`
IBM MQ Advanced Container Multi Instance	L-CXDB-XTZ9J8	`Production`	`MultiInstance`

Annotation	Value of spec.license.license	Value of spec.license.use	Value of spec.queueManager.availability.type
IBM MQ Advanced Container Multi Instance for Non-Production Environment	L-UMVX-XYNRAD	NonProduction	MultiInstance
IBM MQ Advanced with CP4I entitlement	L-VEMB-RABZSN	Production	SingleInstance or NativeHA
IBM MQ Advanced for Non-Production Environment with CP4I entitlement	L-VEMB-RABZSN	NonProduction	SingleInstance or NativeHA
IBM MQ Advanced Container Multi Instance with CP4I entitlement	L-VEMB-RABZSN	Production	MultiInstance
IBM MQ Advanced Container Multi Instance for Non-Production Environment with CP4I entitlement	L-VEMB-RABZSN	NonProduction	MultiInstance

From IBM MQ Operator version 3.7.0, the operator can also apply the following annotations, based on the same queue manager YAML values:

Annotation	Value of spec.license.license	Value of spec.license.use	Value of spec.queueManager.availability.type
IBM MQ	L-MKDD-7KHY2Q	Production	SingleInstance or NativeHA
IBM MQ for Non-Production Environment	L-MKDD-7KHY2Q	NonProduction	SingleInstance or NativeHA
IBM MQ Container Multi Instance	L-MKDD-7KHY2Q	Production	MultiInstance
IBM MQ Container Multi Instance for Non-Production Environment	L-MKDD-7KHY2Q	NonProduction	MultiInstance

From IBM MQ Operator 3.7.0, the operator also applies annotations to a pod when you configure a queue manager's `spec.license.additionalLicenses` field. For more information, see “Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator” on page 187.

The following table describes these extra annotations, which are based on these values in a queue manager's YAML:

- The value of `spec.license.license`
- The value of `spec.license.additionalLicenses`
- The value of `spec.license.use`
- The value of `spec.queueManager.availability.type`

Annotation	Value of spec.license.license	Value of spec.license.additionalLicenses	Value of spec.license.use	Value of spec.queueManager.availability.type
IBM MQ with CP4I entitlement	L-VEMB-RABZSN	L-MKDD-7KHY2Q	Production	SingleInstance or NativeHA
IBM MQ for Non-Production Environment with CP4I entitlement	L-VEMB-RABZSN	L-MKDD-7KHY2Q	NonProduction	SingleInstance or NativeHA
IBM MQ Container Multi Instance with CP4I entitlement	L-VEMB-RABZSN	L-MKDD-7KHY2Q	Production	MultiInstance
IBM MQ Container Multi Instance for Non-Production Environment with CP4I entitlement	L-VEMB-RABZSN	L-MKDD-7KHY2Q	NonProduction	MultiInstance

From IBM MQ Operator 3.7.0, you can use the `spec.license.additionalLicenses` field to configure add-on licenses to be used in combination with your selected `spec.license.license`. For more information, see "Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator" on page 124.

The IBM MQ Operator creates an `IBMLicensingDefinion` CR for each queue manager that is configured to use an add-on license with an IBM MQ license. This CR enables tracking of the additional license by the IBM License Service.

The following table describes add-on annotations that are applied to a pod by the IBM MQ Operator from version 3.7.0, based on these values in a queue manager's YAML:

- The value of `spec.license.license`
- The value of `spec.license.additionalLicenses`
- The value of `spec.license.use`
- The value of `spec.queueManager.availability.type`

Annotation	Value of spec.license.license	Value of spec.license.additionalLicenses	Value of spec.license.use	Value of spec.queueManager.availability.type
IBM MQ Native HA and Cross-Region Replication	L-MKDD-7KHY2Q	L-KFCR-53UGK9	Production	NativeHA
IBM MQ Native HA and Cross-Region Replication for Non-Production Environment	L-MKDD-7KHY2Q	L-KFCR-53UGK9	NonProduction	NativeHA

Linux IBM MQ Advanced container image

A prebuilt container image is available from the IBM Container Registry. You can use this image with either an IBM MQ Advanced or an IBM MQ license. You can also use this image with the IBM MQ Native HA and Cross-Region Replication Add-on for container deployments.

Usage

To be able to use the image, you must accept the terms of the IBM MQ Advanced or IBM MQ license by setting the **LICENSE** environment variable. IBM MQ licenses might require further configuration; for more information, see "Configuring Queue Managers with IBM MQ license annotations by using the IBM MQ Operator" on page 187.

Note: From IBM MQ 9.4.2, you can configure IBM MQ licensed queue managers to use Native HA and Cross-Region Replication features by adding additional licenses to a queue manager. For more information, see "Configuring Native HA and Cross-Region Replication on IBM MQ licensed Queue Managers by using the IBM MQ Operator" on page 124. You cannot use Native HA features on queue managers before IBM MQ 9.4.2 that use an IBM MQ license.

Environment variables supported

AMQ_AMS_FORCE_DISABLED
Set by the IBM MQ Operator on IBM MQ licensed queue managers to prevent accidental use of the IBM MQ Advanced feature AMQ Advanced Message Security (AMS).

LANG
Set the language you want the license to be printed in.

LICENSE
Set `accept` to agree to the license conditions.

Set `view` to view the license conditions.

MQ_CMDLEVEL
Set to the MQ command level to use.

V 10.0.0 **MQ_ENABLE_CLEAN_TMP_ON_START**
Set to `false` to disable the deletion of data in `/tmp`.

MQ_ENABLE_EMBEDDED_WEB_SERVER
Set to `true` to start the web server when the container starts.

MQ_ENABLE_FIPS
Set to `true` to enable FIPS mode. Set to `false` to disable FIPS mode. By default, FIPS mode is automatically enabled if the underlying Linux host has FIPS enabled, but this option can be useful for testing.

MQ_ENABLE_SOFT_FILE_LIMIT_INCREASE
This variable addresses a bug introduced in OpenShift Container Platform 4.21 (OCPBUGS-62327), which reduces the default soft limit for the number of open files to 1024. This limit does not meet the minimum requirement to run IBM MQ on Linux (10240 files). The default value of this variable is `true`, which causes the queue manager to automatically increase the soft limit for the number of open files to match the hard limit before starting IBM MQ. IBM MQ will run with the increased soft limit.

MQ_ENABLE_METRICS
Set `true` to generate Prometheus metrics for your queue manager.

V 10.0.0 V 10.0.0 **MQ_ENABLE_INSTANA_TRACING**

Set to `true` to enable the IBM MQ tracing user exit. See IBM MQ tracing in the IBM Instana documentation.

MQ_ENABLE_TRACE_CRTMQDIR
Set to `true` to create an IBM MQ diagnostic trace while running the **`crtmqdir`** command.

MQ_ENABLE_TRACE_CRTMQM
Set to `true` to create an MQ diagnostic trace while running the **crtmqm** command.

MQ_ENABLE_TRACE_STRMQM
Set to `true` to create an IBM MQ diagnostic trace while running the **strmqm** command.

MQ_GRACE_PERIOD
The target time in which ending the queue manager within is attempted, escalating the phases of application disconnection. Sets the **endmqm -tp** option used when the container terminates. See endmqm (end queue manager) for more information.

MQ_LOGGING_CONSOLE_SOURCE
Specify a comma-separated list of sources for logs that are mirrored to the container's **stdout** location.

Valid values are `qmgr`, `web`, `mqsc`, and `probes`.

Default value is `qmgr,web,probes`.

Optional value is `mqsc`. This option can be used to reflect the contents of `autocfgmqsc.LOG` in the container log.

MQ_LOGGING_CONSOLE_FORMAT
Change the format of the logs that are printed to the container's **stdout** location.

Set `basic` to use a simple human-readable format. This is the default value.

Set `json` to use JSON format (one JSON object on each line).

MQ_LOGGING_CONSOLE_EXCLUDE_ID
Specify a comma-separated list of message IDs for log messages that are excluded.

The log messages still appear in the log file on disk, but are not printed to the container's **stdout** location.

Default value is `AMQ5041I,AMQ5052I,AMQ5051I,AMQ5037I,AMQ5975I`.

V 10.0.0 **MQ_LOGGING_METRICS_AUDIT_ENABLED**
Set to "true" to enable audit logging of access to the Prometheus metrics endpoint. Log output is sent to a JSON file in `/var/mqm/errors/`. You must also set `MQ_ENABLE_METRICS=true`, to generate the Prometheus metrics.

MQ_MULTI_INSTANCE
Set to "true" to enable running as a multi-instance queue manager. This changes the options used with **endmqm**.

MQ_NATIVE_HA
Set to "true" to enable Native HA. This sets the **crtmqm -lr** option. See crtmqm (create queue manager) for more information. You can configure other Native HA settings by mounting an INI file. See `NativeHAInstance` stanza of the qm.ini file and `NativeHALocalInstance` stanza of the qm.ini file.

Deprecated **MQ_NATIVE_HA_INSTANCE_0_NAME**
Sets the `Name` attribute in the `NativeHAInstance` INI stanza for one of the three Native HA instances. See `NativeHAInstance` stanza of the qm.ini file. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply an INI file fragment in `/etc/mqm`.

Deprecated **MQ_NATIVE_HA_INSTANCE_1_NAME**
Sets the `Name` attribute in the `NativeHAInstance` INI stanza for one of the three Native HA instances. See `NativeHAInstance` stanza of the qm.ini file. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply an INI file fragment in `/etc/mqm`.

Deprecated **MQ_NATIVE_HA_INSTANCE_2_NAME**
Sets the `Name` attribute in the `NativeHAInstance` INI stanza for one of the three Native HA instances. See `NativeHAInstance` stanza of the qm.ini file. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply an INI file fragment in `/etc/mqm`.

Deprecated **MQ_NATIVE_HA_INSTANCE_0_REPLICATION_ADDRESS**
Sets the `ReplicationAddress` attribute in the `NativeHAInstance` INI stanza for one of the three Native HA instances. See `NativeHAInstance` stanza of the qm.ini file. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply an INI file fragment in `/etc/mqm`.

Deprecated **MQ_NATIVE_HA_INSTANCE_1_REPLICATION_ADDRESS**
Sets the `ReplicationAddress` attribute in the `NativeHAInstance` INI stanza for one of the three Native HA instances. See `NativeHAInstance` stanza of the qm.ini file. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply an INI file fragment in `/etc/mqm`.

Deprecated **MQ_NATIVE_HA_INSTANCE_2_REPLICATION_ADDRESS**
Sets the `ReplicationAddress` attribute in the `NativeHAInstance` INI stanza for one of the three Native HA instances. See `NativeHAInstance` stanza of the qm.ini file. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply an INI file fragment in `/etc/mqm`.

Deprecated **MQ_NATIVE_HA_CIPHERSPEC**
Sets the `CipherSpec` attribute in the `NativeHALocalInstance` INI stanza for one of the three Native HA instances. See `NativeHALocalInstance` stanza of the qm.ini file. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply an INI file fragment in `/etc/mqm`.

Deprecated **MQ_NATIVE_HA_KEY_REPOSITORY**
Ignore automatically generated key repository for Native HA, and use the one specified. From IBM MQ 9.4.1, this variable is deprecated, and you should instead supply the location of the key repository in an INI file fragment in `/etc/mqm`.

MQ_QMGR_LOG_FILE_PAGES
MQ recovery log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages. This environment variable sets the **`crtmqm -lf`** option. See crtmqm (create queue manager) for more information.

MQ_QMGR_NAME
Set the name you want your queue manager to be created with.

File system and mount points

`/etc/mqm`
Any MQSC or INI files in this directory will be processed by the queue manager when it starts. This is the directory used by the automatic configuration feature. See Automatic configuration from an MQSC script at startup and Automatic configuration of qm.ini at startup for more information.

`/etc/mqm/pki/keys`
Contains sub-directories with private keys in PEM format, using either PKCS #1 or unencrypted PKCS #8 key structure. Each sub-directory under `/etc/mqm/pki/keys` is scanned for any files with the `.key` extension. The public X.509 certificate chain should be in the same directory, in one or more files with the `.crt` extension, in PEM format. If keys are mounted in this way, then a key repository is automatically generated in `/run/runmqserver/tls`. The key repository will be set in the queue manager's SSLKEYR setting, and the default key (CERTLABL) will be the first name after sorting the list of key names lexicographically.

`/etc/mqm/pki/trust`
Contains sub-directories with public X.509 certificates in PEM format. Each sub-directory will be scanned for any `.crt` files, and added as trusted certificates. The certificates for any private keys are automatically trusted, and don't need to be added this way.

`/etc/mqm/ha/pki/keys`
Configures the keys and certificates used by Native HA for intra-cluster communication. Contains sub-directories with private keys in PEM format, using either PKCS #1 or unencrypted PKCS #8 key structure. Each sub-directory under `/etc/mqm/ha/pki/keys` is scanned for any files with the `.key` extension. The public X.509 certificate chain should be in the same directory, in one or more files with the `.crt` extension, in PEM format. If keys are mounted in this way, then a key repository is automatically generated, along with an INI file fragment to use that key repository. The key used will

be the first name after sorting the list of key names lexicographically. The generated key repository is written into `/run/runmqserver/ha/tls`.

/etc/mqm/groupha/pki/keys
Configures the keys used by Native HA CRR for inter-cluster communication. Merges into the same key repository, and follows the same rules as `/etc/mqm/ha/pki/keys`.

/etc/mqm/groupha/pki/trust
Configures the trusted certificates used by Native HA CRR for inter-cluster communication. Contains sub-directories with public X.509 certificates in PEM format. Each sub-directory will be scanned for any `.crt` files, and added as trusted certificates. The certificates for any private keys are automatically trusted, and don't need to be added this way.

/etc/mqm/metrics/pki/keys
From 9.4.2.0-r1, if TLS keys are provided in this directory, and metrics are enabled for a queue manager, an HTTPS server is started on `/metrics` port 9157. Files in this directory should be PEM formatted:

- `tls.crt`: Server's public X.509 certificate
- `tls.key`: Server's private key (PKCS #1 or unencrypted PKCS #8)
- `ca.crt`: CA's public X.509 certificate (optional)

If no TLS keys are provided, or earlier versions of IBM MQ are used, an HTTP server is started.

/etc/mqm/web
The directory tree under `/etc/mqm/web` is copied over the top of `/var/mqm/web` when the container starts. For example, you can create `/etc/mqm/web/installations/Installation1/servers/mqweb/mqwebuser.xml` to configure the web server. This file would be copied into right location under the `/var/mqm/web`, which is usually a mounted volume. See Configuring the IBM MQ Console and REST API.

/mnt/mqm
The `/var/mqm` directory is symbolically linked to this location. By mounting in a persistent volume into this location, you can persist the MQ data across container runs.

/mnt/mqm-log
Directory used to hold recovery log files for the queue manager. If a volume is mounted into this location, the container code sets the **crtmqm -ld** option. See crtmqm (create queue manager) for more information.

/mnt/mqm-data
Directory used to hold the data files for the queue manager. If a volume is mounted into this location, the container code sets the **crtmqm -md** option. See crtmqm (create queue manager) for more information.

/run
The container writes temporary files under this directory. If you want to use a read-only filesystem for the container, you need to mount a volume into this location, and into `/tmp`

/run/secrets/initial.key
The initial key used to generate the WebSphere Liberty AES encryption key for the IBM MQ mqweb server. This key is used to encrypt sensitive configuration data, such as passwords and keystores, in the web server configuration files. The initial key has a minimum length of 16 characters. If this file doesn't exist or is empty, the initial key is randomly generated.

/run/termination-log
If the container code encounters an error which will cause the queue manager to terminate, it will write the reason into this file. This file location is used by default in Kubernetes to retrieve termination information.

/tmp
The container writes temporary files under this directory. If you want to use a read-only filesystem for the container, you need to mount a volume into this location, and into `/run`

OpenShift CP4I Kubernetes IBM MQ Advanced for Developers container image

A prebuilt container image is available for IBM MQ Advanced for Developers. This image is available from the IBM Container Registry. This image is suitable for use with Docker, Podman, Kubernetes, and other container environments.

Available images

IBM MQ images are stored in the IBM Container Registry:

- IBM MQ Advanced for Developers 10.0.0.0: `icr.io/ibm-messaging/mq:10.0.0.0-r3`
- IBM MQ Advanced for Developers 10.0.0.0: `icr.io/ibm-messaging/mq:10.0.0.0-r3`

Quick reference

- License:
  - "mq.ibm.com/v1beta1: Current license versions" on page 281 and Apache License 2.0. Note that the IBM MQ Advanced for Developers license does not permit further distribution, and the terms restrict usage to a developer machine.
- Where to file issues:
  - GitHub
- Available for the following CPU architectures:
  - amd64
  - s390x
  - ppc64le

Usage

Run IBM MQ Advanced for Developers in a container.

See the usage documentation for details on how to run a container.

To be able to use the image, you must accept the terms of the IBM MQ license by setting the **LICENSE** environment variable.

Environment variables

All of the environment variables from the "IBM MQ Advanced container image" on page 322 are available, plus the following additional variables:

Deprecated **MQ_ADMIN_PASSWORD**

Specify the password of the admin user.

Must be at least 8 characters long.

There is no default password for the admin user.

From IBM MQ 9.4.0, this variable is no longer provided. The example YAML in this topic shows how you can provide the admin password by mounting a secret.

Deprecated **MQ_APP_PASSWORD**

Specify the password of the app user.

If set, this causes the **DEV.APP.SVRCONN** channel to become secured and only allow connections that supply a valid user ID and password.

Must be at least 8 characters long.

There is no default password for the app user.

From IBM MQ 9.4.0, this variable is no longer provided. The example YAML in this topic shows how you can provide the app password by mounting a secret..

MQ_DEV
Set `false` to stop the default objects being created.

For more information about the default developer configuration supported by the IBM MQ Advanced for Developers image, see the default developer configuration documentation.

Example queue manager YAML that describes how to specify passwords for admin and app users

For users of the **admin** and **app** user IDs, you must provide passwords when deploying a queue manager using the `Development` license. Here is an example queue manager YAML that shows you how to do this with the IBM MQ Operator.

The following command creates a secret containing passwords for **admin** and **app** users.

```
oc create secret generic my-mq-dev-passwords --from-literal=mqAdminPassword=passw0rd --from-
literal=mqAppPassword=passw0rd
```

The following YAML uses these passwords when deploying a queue manager.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: qm-dev
  annotations:
    "com.ibm.mq/write-defaults-spec" : "false"
spec:
  license:
    accept: true
    license: L-GWLM-M6WVZU
    use: Development
  web:
    enabled: true
  queueManager:
    env:
      - name: MQ_DEV
        value: "true"
      - name: MQ_CONNAUTH_USE_HTP
        value: "true"
    files:
      - secret:
          items:
            - item: mqAdminPassword
            - item: mqAppPassword
          name: my-mq-dev-passwords
          defaultMountPath: /run/secrets
    storage:
      queueManager:
        type: persistent-claim
    name: QUICKSTART
  version: 10.0.0.0-r3
```

OpenShift CP4I Podman IBM MQ MFT Agent for Developers container image

A prebuilt container image is available for IBM MQ MFT Agent for Developers. The image published on IBM Container Registry (`irc.io`) can be used with Podman, Docker, and OpenShift Container Platform environments.

Attention: Deprecated IBM MQ MFT Agent for Developers images were previously available from Docker Hub, but this is deprecated, and no further updates are available on Docker Hub.

Available images

The latest version of IBM MQ MFT Agent for Developers is published on IBM Container Registry: `icr.io/ibm-messaging/mqmft:latest`

Quick reference

- License:
  - IBM MQ Advanced for Developers and Apache License 2.0. The IBM MQ Advanced for Developers license does not permit further distribution, and the terms restrict usage to a developer machine.
- Where to file issues:
  - GitHub
- Available for the following platform:
  - Linux on amd64

Building a container image

See Building a container image for information on how to build your own agent container image with the IBM MQ Redistributable Managed File Transfer package on amd64 architectures.

Usage

See the usage documentation for details on how to run the image container with Podman runtime.

See the usage documentation for details on how to deploy the image in an OpenShift Container Platform.

To be able to use the image, you must accept the terms of the IBM MQ license by setting the **LICENSE** environment variable.

Environment variables

LICENSE
Set `accept` to agree to the IBM MQ Advanced for Developers license conditions.

Set `view` to view the license conditions.

MFT_AGENT_CONFIG_FILE
Required.

The path of the JSON file containing the information required for setting up an agent. The path must be on a mount point. For example, a `configMap` on OpenShift. See Additional MFT agent configuration files for a detailed description of the attributes.

MFT_AGENT_NAME
Required.

The name of the agent to configure.

BFG_JVM_PROPERTIES
Optional.

Any JVM property that needs to be set when running the agent JVM.

MFT_LOG_LEVEL
Optional.

The level of information displayed. The supported values are `info` and `verbose`. The default is `info`. Contents of agent's `output0.log` is displayed if **MFT_LOG_LEVEL** is set to `verbose`.

MFT_AGENT_START_WAIT_TIME
Optional.

An agent might take some time to start after the **fteStartAgent** command is issued. This is the time, in seconds, that the container waits for an agent to start. If an agent does not start within the specified wait time, the container ends.

MFT_MOUNT_PATH
Optional.

An environment variable pointing to the path from where the agent reads or writes files.

MFT_COORD_QMGR_CIPHER
Required.

The name of the CipherSpec to be used for securely connecting to the coordination queue manager.

MFT_CMD_QMGR_CIPHER
Required.

The name of the CipherSpec to be used for securely connecting to the command queue manager.

MFT_AGENT_QMGR_CIPHER
Required.

The name of the CipherSpec to be used for securely connecting to the agent queue manager.

For more information about the default developer configuration supported by the IBM MQ Advanced for Developers image, see the default developer configuration documentation.

Location of agent configuration files

The agent in the container will create agent configuration and log files under the fixed directory `/mnt/mftdata`. This folder can be on a persistent volume as well, in which case the volume must be mounted as `/mnt/mftdata` mount point in to the container.

IBM MQ CASE for air-gap install and upgrade

When installing the IBM MQ Operator and IBM MQ queue managers into an air-gapped cluster, the IBM MQ CASE is used. The IBM MQ CASE contains details of all the images and metadata that make up a particular release of the IBM MQ CASE.

Air-gap and IBM MQ CASE

The IBM MQ CASE contains images for only a single version of the queue manager. This version is the latest version of the queue manager that the corresponding IBM MQ Operator can deploy. See the IBM MQ CASE Lookup Tables for the exact version available in each IBM MQ CASE.

You can use an IBM MQ CASE to install, or upgrade to, the latest available queue manager that the operator can deploy; that is, the CD or SC2 queue manager that is released at the same time as the CASE and corresponding operator. When you deploy the latest available queue manager, the install or upgrade process runs quickly because it has only 1 image to mirror.

However, if you are installing one or more queue manager versions that are not the latest that the operator can deploy, you must repeat the air-gap image mirroring process using a separate IBM MQ CASE for each required queue manager version. See the IBM MQ CASE Lookup Tables for the exact version available in each IBM MQ CASE.

Note: The IBM MQ Operator is aware of the older queue manager versions so you can ask it to deploy those versions. For this reason, it is important that you consider which queue manager versions you need to mirror.

Queue manager-related images

The IBM MQ CASE contains only the images for the single latest queue manager version that the corresponding operator can deploy.

Note: The IBM MQ CASE no longer includes the Operations Dashboard images, because it is not an option for the latest queue manager versions to deploy Operations Dashboard.

Use the IBM MQ CASE Lookup Tables to identify which IBM MQ CASE contains the IBM MQ queue manager version you require.

Operator-related images

Use the IBM MQ CASE Lookup Tables to identify which IBM MQ CASE contains the IBM MQ Operator version you require.

Important:

- When installing the IBM MQ Operator in an air-gap environment, you should always complete the air-gap mirroring process for the IBM MQ CASE that corresponds to the IBM MQ Operator version you intend to install. Do not install, or upgrade to, an operator version from a different IBM MQ CASE.
- The IBM MQ CASE contains additional IBM MQ Operator versions, but only install the operator that corresponds to the IBM MQ CASE you have mirrored.
- When upgrading the IBM MQ Operator, if you are changing operator channels, follow the air-gap upgrade best practice to upgrade to the latest IBM MQ Operator version available on the current channel. It is important to do this before changing the subscription to a new IBM MQ Operator channel.

IBM MQ CASE Lookup Tables

Table 8. Contents of recent CD instances of IBM MQ CASE

IBM MQ CASE	IBM MQ Operator	IBM MQ Queue Manager
4.0.2	4.0.2	10.0.0.0-r3
4.0.1	4.0.1	10.0.0.0-r2
4.0.0	4.0.0	10.0.0.0-r1
3.9.2	3.9.2	9.4.5.1-r1
3.9.1	3.9.1	9.4.5.0-r2
3.9.0	3.9.0	9.4.5.0-r1
3.8.1	3.8.1	9.4.4.1-r1
3.8.0	3.8.0	9.4.4.0-r4
3.7.2	3.7.2	9.4.4.0-r3
3.7.1	3.7.1	9.4.4.0-r2
3.7.0	3.7.0	9.4.4.0-r1
3.6.4	3.6.4	9.4.3.1-r3
3.6.3	3.6.3	9.4.3.1-r2
3.6.2	3.6.2	9.4.3.1-r1
3.6.1	3.6.1	9.4.3.0-r2
3.6.0	3.6.0	9.4.3.0-r1
3.5.3	3.5.3	9.4.2.1-r2
3.5.2	3.5.2	9.4.2.1-r1
3.5.1	3.5.1	9.4.2.0-r2
3.5.0	3.5.0	9.4.2.0-r1

Table 8. Contents of recent CD instances of IBM MQ CASE (continued)		
IBM MQ CASE	**IBM MQ Operator**	**IBM MQ Queue Manager**
3.4.1	3.4.1	9.4.1.1-r1, 9.4.1.0-r2, 9.4.1.0-r1, 9.4.0.7-r1, 9.4.0.6-r2, 9.4.0.6-r1, 9.4.0.5-r2, 9.4.0.5-r1, 9.4.0.0-r3, 9.4.0.0-r2, 9.4.0.0-r1, 9.3.5.1-r2, 9.3.5.1-r1, 9.3.5.0-r2, 9.3.5.0-r1, 9.3.4.1-r1, 9.3.4.0-r1, 9.3.3.3-r2, 9.3.3.3-r1, 9.3.3.2-r3, 9.3.3.2-r2, 9.3.3.2-r1, 9.3.3.1-r2, 9.3.3.1-r1, 9.3.3.0-r2, 9.3.3.0-r1, 9.3.2.1-r2, 9.3.2.1-r1, 9.3.2.0-r2, 9.3.2.0-r1, 9.3.1.1-r1, 9.3.1.0-r3, 9.3.1.0-r2, 9.3.1.0-r1, 9.3.0.25-r1, 9.3.0.21-r3, 9.3.0.21-r2, 9.3.0.21-r1, 9.3.0.20-r2, 9.3.0.20-r1, 9.3.0.17-r3, 9.3.0.17-r2, 9.3.0.17-r1, 9.3.0.16-r2, 9.3.0.16-r1, 9.3.0.15-r1, 9.3.0.11-r2, 9.3.0.11-r1, 9.3.0.10-r2, 9.3.0.10-r1, 9.3.0.6-r1, 9.3.0.5-r3, 9.3.0.5-r2, 9.3.0.5-r1, 9.3.0.4-r2, 9.3.0.4-r1, 9.3.0.3-r1, 9.3.0.1-r4, 9.3.0.1-r3, 9.3.0.1-r2, 9.3.0.1-r1, 9.3.0.0-r3, 9.3.0.0-r2, 9.3.0.0-r1

Table 9. Contents of recent SC2 instances of IBM MQ CASE		
IBM MQ CASE	**IBM MQ Operator**	**IBM MQ Queue Manager**
4.0.2	4.0.2	10.0.0.0-r3, 10.0.0.0-r2, 10.0.0.0-r1

For information about the contents of the 3.2.x SC2 IBM MQ CASE, see the SC2 table in the IBM MQ CASE Lookup Tables section of the IBM MQ 9.4 documentation.

Previous operator removal

For reference, from IBM MQ CASE 3.4.0, the IBM MQ Operator images for the following channels were removed from the IBM MQ CASE:

- v1.1
- v1.2
- v1.4
- v1.5
- v1.6
- v1.7
- v2.1

Recreating Clusters and Image Registries

You can list all of the queue managers that are deployed in a cluster, and their versions:

For deployments on the Red Hat OpenShift Container Platform:

```
oc get queuemanager --all-namespaces -o=jsonpath='{range .items[*]}{.metadata.name}{":\t"}
{.spec.version}{"\n"}{end}'
```

For deployments on Amazon EKS:

```
kubectl get queuemanager --all-namespaces -o=jsonpath='{range .items[*]}{.metadata.name}{":\t"}
{.spec.version}{"\n"}{end}'
```

You should run this command periodically, and store the output. This list is valuable if you want to recreate either the cluster, or the Image Registry storing the images mirrored for air-gap.

Related tasks

"Upgrading to the 4.0.x CD or SC2 channel of IBM MQ Operator on Red Hat OpenShift" on page 57
Upgrade either a CD or SC2 deployment of the IBM MQ Operator where **only** IBM MQ licenses are used.

"Installing the IBM MQ Operator on Red Hat OpenShift" on page 43
The IBM MQ Operator can be installed onto Red Hat OpenShift using the OpenShift console or command line interface (CLI).

"Installing the IBM MQ Operator for use with IBM Cloud Pak for Integration on Red Hat OpenShift" on page 50
For use with the IBM Cloud Pak for Integration, the IBM MQ Operator can be installed onto Red Hat OpenShift through the OpenShift console or command line interface (CLI).

IBM MQ Operator Helm chart values

These default values for configuration properties are provided by the IBM MQ Operator when you deploy or upgrade the operator by using Helm. You can override these values when you run the `helm install` or `helm upgrade` commands.

Field	Type	Default Value	Description
`operator.env`	object	`{}`	Environment variables to pass to the IBM MQ Operator. (For example `{env1: "value1",env2: "value2"}`)
`operator.deployment.repository`	string	`cpopen/ibm-mq-operator`	Repository for the IBM MQ Operator image
`operator.deployment.sha`	string	Latest sha256 value of the IBM Operator image	SHA value of the IBM MQ Operator image
`operator.deployment.tag`	string		Tag value for the IBM MQ Operator image (optional; ignored when a SHA value is provided)
`operator.deployment.pullPolicy`	string	`IfNotPresent`	Image pull policy for the IBM MQ Operator (`IfNotPresent`, `Always` or `Never`)
`operator.deployment.resources.requests.cpu`	string	1	CPU requests setting for the IBM MQ Operator

Field	Type	Default Value	Description
operator.deployment.resources.requests.memory	string	1Gi	Memory requests setting for the IBM MQ Operator
operator.deployment.resources.limits.cpu	string	1	CPU limits setting for the IBM MQ Operator
operator.deployment.resources.limits.memory	string	1Gi	Memory limits setting for the IBM MQ Operator
operator.installMode	string	OwnNamespace	Install mode for the IBM MQ Operator to determine at what scope it operates (OwnNamespace or AllNamespaces)
operator.imagePullSecrets	list	[]	Image pull secrets for the IBM MQ Operator image, to allow pulling from authenticated registries. (For example, ["secret1","secret2"])
operand.repositories.integration	string	cp/ibm-mqadvanced-server-integration	Repository for the IBM Cloud Pak for Integration image. **Important:** Changing operand.repositories causes a **restart** of existing queue managers.
operand.repositories.production	string	cp/ibm-mqadvanced-server	Repository for the IBM MQ Advanced image. **Important:** Changing operand.repositories causes a **restart** of existing queue managers.
operand.repositories.developer	string	ibm-messaging/mq	Repository for the IBM MQ Advanced for Developers image. **Important:** Changing operand.repositories causes a **restart** of existing queue managers.

Field	Type	Default Value	Description
`privateRegistry`	string	`""`	Private registry allowing pulling of IBM MQ Operator and Queue Manager images from an alternative private registry (For example, `my-private-registry.com:5000`). **Important:** Changing `privateRegistry` causes a **restart** of existing queue managers and the IBM MQ Operator.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan, Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

© Copyright IBM Corp. 2007, 2026

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of IBM MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com®, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information"www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (https://www.eclipse.org/).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Part Number:

(1P) P/N: