

9.4

*IBM MQ in containers*

**IBM**

**Note**

Before using this information and the product it supports, read the information in [“Notices” on page 171](#).

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, 2024.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

---

# Contents

<b>IBM MQ in containers and IBM Cloud Pak for Integration.....</b>	<b>5</b>
About.....	5
Release history for IBM MQ Operator.....	5
Planning.....	8
Choosing how you want to use IBM MQ in containers.....	8
Support for IBM MQ in containers.....	9
Planning for licensing IBM MQ in containers.....	15
Planning storage for the IBM MQ Operator.....	16
Planning high availability for IBM MQ in containers.....	17
Disaster recovery for IBM MQ in containers.....	22
Planning security for IBM MQ in containers.....	23
Planning scalability and performance for IBM MQ in containers.....	29
Preparing, installing and upgrading.....	29
Installing and upgrading the IBM MQ Operator.....	30
Preparing for IBM MQ by building your own container image.....	53
Deploying and configuring.....	60
Deploying and configuring queue managers using the IBM MQ Operator.....	60
Deploying and configuring queue managers using Helm.....	101
Migrating to the IBM MQ Operator.....	101
Checking that required functions are available.....	102
Extracting the queue manager configuration.....	102
Optional: Extracting and acquiring the queue manager keys and certificates.....	103
Optional: Configuring LDAP.....	105
Optional: Changing the IP addresses and host names in the IBM MQ configuration.....	113
Updating the queue manager configuration for a container environment.....	114
Selecting the target HA architecture for IBM MQ running in containers.....	118
Creating the resources for the queue manager.....	118
Creating the new queue manager on Red Hat OpenShift.....	120
Verifying the new container deployment.....	124
Operating.....	125
Operating IBM MQ using the IBM MQ Operator.....	125
Viewing the status of Native HA queue managers.....	132
Manually ending Native HA queue manager instances.....	134
Reference.....	134
API reference for the IBM MQ Operator.....	135
License annotations when building your own IBM MQ container image.....	158
IBM MQ Advanced for Developers container image.....	163
Troubleshooting.....	166
Troubleshooting unplanned restarts of IBM MQ in containers.....	166
Troubleshooting problems with the IBM MQ Operator.....	167
<b>Notices.....</b>	<b>171</b>
Programming interface information.....	172
Trademarks.....	172




# 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®. 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 as 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 earlier IBM MQ Operators, see [Release history for IBM MQ Operator](#) in the IBM MQ 9.3 documentation.
- For information about future IBM MQ updates, see the overall [IBM MQ Recommended Fixes and Planned Maintenance release dates](#) page.

### IBM MQ Operator 3.2.2



#### IBM Cloud Pak for Integration version

IBM Cloud Pak for Integration 16.1.0

#### Operator channel

v3.2-sc2

#### Allowed values for .spec.version

[9.4.0.0-r2](#)

#### Allowed values for .spec.version during migration

9.3.0.0-r1, 9.3.0.0-r2, 9.3.0.0-r3, 9.3.0.1-r1, 9.3.0.1-r2, 9.3.0.1-r3, 9.3.0.1-r4, 9.3.0.3-r1, 9.3.0.4-r1, 9.3.0.4-r2, 9.3.0.5-r1, 9.3.0.5-r2, 9.3.0.5-r3, 9.3.0.6-r1, 9.3.0.10-r1, 9.3.0.10-r2, 9.3.0.11-r1, 9.3.0.11-r2, 9.3.0.15-r1, 9.3.0.16-r1, 9.3.0.16-r2, 9.3.0.17-r1, 9.3.0.17-r2, 9.3.0.17-r3, 9.3.0.20-r1, 9.3.1.0-r1, 9.3.1.0-r2, 9.3.1.0-r3, 9.3.1.1-r1, 9.3.2.0-r1, 9.3.2.0-r2, 9.3.2.1-r1, 9.3.2.1-r2, 9.3.3.0-r1, 9.3.3.0-r2, 9.3.3.1-r1, 9.3.3.1-r2, 9.3.3.2-r1, 9.3.3.2-r2, 9.3.3.3-r1, 9.3.3.3-r2, 9.3.4.0-r1, 9.3.4.1-r1, 9.3.5.0-r1, 9.3.5.0-r2, 9.3.5.1-r1, 9.3.5.1-r2, 9.4.0.0-r1

#### Red Hat OpenShift Container Platform versions

OpenShift Container Platform 4.12 and above.

## IBM Cloud Pak foundational services versions

IBM Cloud Pak foundational services version 4.6 only.

### What's changed

- Vulnerabilities that are addressed are detailed in this [Security Bulletin](#).

## IBM MQ Operator 3.2.1



### IBM Cloud Pak for Integration version

IBM Cloud Pak for Integration 16.1.0

### Operator channel

v3.2-sc2

### Allowed values for .spec.version

[9.4.0.0-r1](#)

### Allowed values for .spec.version during migration

9.3.0.0-r1, 9.3.0.0-r2, 9.3.0.0-r3, 9.3.3.2-r3 9.3.0.1-r1, 9.3.0.1-r2, 9.3.0.1-r3, 9.3.0.1-r4, 9.3.0.3-r1, 9.3.0.4-r1, 9.3.0.4-r2, 9.3.0.5-r1, 9.3.0.5-r2, 9.3.0.5-r3, 9.3.0.6-r1, 9.3.0.10-r1, 9.3.0.10-r2, 9.3.0.11-r1, 9.3.0.11-r2, 9.3.0.15-r1, 9.3.0.16-r1, 9.3.0.16-r2, 9.3.0.17-r1, 9.3.0.17-r2, 9.3.0.17-r3, 9.3.1.0-r1, 9.3.1.0-r2, 9.3.1.0-r3, 9.3.1.1-r1, 9.3.2.0-r1, 9.3.2.0-r2, 9.3.2.1-r1, 9.3.2.1-r2, 9.3.3.0-r1, 9.3.3.0-r2, 9.3.3.1-r1, 9.3.3.1-r2, 9.3.3.2-r1, 9.3.3.2-r2, 9.3.3.2-r3, 9.3.3.3-r1, 9.3.3.3-r2, 9.3.4.0-r1, 9.3.4.1-r1, 9.3.5.0-r1, 9.3.5.0-r2, 9.3.5.1-r1, 9.3.5.1-r2

### Red Hat OpenShift Container Platform versions

OpenShift Container Platform 4.12 and above.

## IBM Cloud Pak foundational services versions

IBM Cloud Pak foundational services version 4.6 only.

### What's changed

- Resolves an issue on OpenShift Container Platform 4.12 where upgrading to the v3.2-sc2 channel could cause unexpected behavior for IBM Cloud Pak for Integration users. For more information, see [Upgrading from 2023.4](#) in the IBM Cloud Pak for Integration documentation.

## IBM MQ Operator 3.2.0



### IBM Cloud Pak for Integration version

IBM Cloud Pak for Integration 16.1.0

### Operator channel

v3.2-sc2

### Allowed values for .spec.version

[9.4.0.0-r1](#)

### Allowed values for .spec.version during migration

9.3.0.0-r1, 9.3.0.0-r2, 9.3.0.0-r3, 9.3.3.2-r3 9.3.0.1-r1, 9.3.0.1-r2, 9.3.0.1-r3, 9.3.0.1-r4, 9.3.0.3-r1, 9.3.0.4-r1, 9.3.0.4-r2, 9.3.0.5-r1, 9.3.0.5-r2, 9.3.0.5-r3, 9.3.0.6-r1, 9.3.0.10-r1, 9.3.0.10-r2, 9.3.0.11-r1, 9.3.0.11-r2, 9.3.0.15-r1, 9.3.0.16-r1, 9.3.0.16-r2, 9.3.0.17-r1, 9.3.0.17-r2, 9.3.0.17-r3, 9.3.1.0-r1, 9.3.1.0-r2, 9.3.1.0-r3, 9.3.1.1-r1, 9.3.2.0-r1, 9.3.2.0-r2, 9.3.2.1-r1, 9.3.2.1-r2, 9.3.3.0-r1, 9.3.3.0-r2, 9.3.3.1-r1, 9.3.3.1-r2, 9.3.3.2-r1, 9.3.3.2-r2, 9.3.3.2-r3, 9.3.3.3-r1, 9.3.3.3-r2, 9.3.4.0-r1, 9.3.4.1-r1, 9.3.5.0-r1, 9.3.5.0-r2, 9.3.5.1-r1, 9.3.5.1-r2

### Red Hat OpenShift Container Platform versions

OpenShift Container Platform 4.12 and above.

## IBM Cloud Pak foundational services versions

IBM Cloud Pak foundational services version 4.6 only.

## What's new

- “Expanding persistent volumes” on page 96 is now supported.
- Queue managers can now be stopped by adding the `mq.ibm.com/stop` annotation and setting it to `true`. See [“Stopping a queue manager \(mq.ibm.com/stop\)”](#) on page 100

### Notes:

- A stopped queue manager has the `.replicas` field in its `StatefulSet` set to 0.
- Because the IBM MQ Operator now actively manages the `.replicas` field in the `StatefulSet`, if you modify this field it is immediately reverted by the operator.
- Older versions of IBM MQ enter a 'Failed' state if you modify the `.replicas` field, but still retain the modified value. If your existing operating procedures rely on this behavior, from IBM MQ 9.4 you must use the `mq.ibm.com/stop` annotation.

## What's changed

- Odd-numbered releases of Red Hat OpenShift Container Platform are now supported.
- IBM MQ Catalog image moved to file-based catalog format from SQLite database format.
- Based on Red Hat Universal Base Image 9.4-949.1716471857. **Note:** UBI 9 has pending FIPS 140-3 certification. UBI 9 is not supported on the Power® 8 architecture.
- Vulnerabilities that are addressed are detailed in this [Security Bulletin](#).

## Release history for Queue Manager Container images for use with IBM MQ Operator

**Note:** For information about earlier Queue Manager Container images, see [Release history for IBM MQ Operator](#) in the IBM MQ 9.3 documentation.

### 9.4.0.0-r2



#### Required operator version

[3.2.2](#) or higher

#### Supported architectures

amd64, s390x, ppc64le

#### Images

- `cp.icr.io/cp/ibm-mqadvanced-server-integration:9.4.0.0-r2`
- `cp.icr.io/cp/ibm-mqadvanced-server:9.4.0.0-r2`
- `icr.io/ibm-messaging/mq:9.4.0.0-r2`

## What's new

- [What's new in IBM MQ 9.4.0 for Multiplatforms - base and Advanced entitlement](#)

## What's changed

- [What's changed in IBM MQ 9.4.0.](#)
- Based on [Red Hat Universal Base Image 9.4-1134](#).

### 9.4.0.0-r1



#### Required operator version

[3.2.0](#) or higher

#### Supported architectures

amd64, s390x, ppc64le

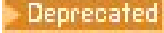
## Images

- [cp.icr.io/cp/ibm-mqadvanced-server-integration:9.4.0.0-r1](https://cp.icr.io/cp/ibm-mqadvanced-server-integration:9.4.0.0-r1)
- [cp.icr.io/cp/ibm-mqadvanced-server:9.4.0.0-r1](https://cp.icr.io/cp/ibm-mqadvanced-server:9.4.0.0-r1)
- [icr.io/ibm-messaging/mq:9.4.0.0-r1](https://icr.io/ibm-messaging/mq:9.4.0.0-r1)

## What's new

- [What's new in IBM MQ 9.4.0 for Multiplatforms - base and Advanced entitlement](#)

## What's changed

- [What's changed in IBM MQ 9.4.0](#)
-  **Deprecated** When using IBM MQ Advanced for Developers, setting the passwords for the admin and app users through an environment variable is deprecated. Use Secrets instead.
- A new optional value mqsc is added for the environment variable `MQ_LOGGING_CONSOLE_SOURCE`. This option can be used to reflect the contents of `autocfgmqsc.LOG` in the container log.
- Based on Red Hat Universal Base Image 9.4-949.1716471857. **Note:** UBI 9 has pending FIPS 140-3 certification. UBI 9 is not supported on the Power 8 architecture.

## 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 concepts (see [OpenShift Container Platform architecture](#)).

### Procedure

- [“Choosing how you want to use IBM MQ in containers” on page 8.](#)
- [“Support for IBM MQ in containers” on page 9.](#)
- [“Planning storage for the IBM MQ Operator” on page 16.](#)
- [“Planning high availability for IBM MQ in containers” on page 17.](#)
- [“Disaster recovery for IBM MQ in containers” on page 22.](#)
- [“User authentication and authorization for IBM MQ in containers” on page 23.](#)

## 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, which uses pre-packaged container images, or you can build your own images and deployment code.

### Using the IBM MQ Operator



If you are planning to deploy on Red Hat OpenShift Container Platform, then you probably want to use the IBM MQ Operator.

The IBM MQ Operator extends the Red Hat OpenShift Container Platform API to add a new `QueueManager` custom resource. The operator watches for new queue manager definitions, and then turns them into necessary low-level resources, such as `StatefulSet` and `Service` resources. In the case of Native HA, the operator can also perform the complex rolling update of queue manager instances. See [“Considerations for performing your own rolling update of a Native HA queue manager” on page 21](#)



Some IBM MQ features are not supported when using the IBM MQ Operator. See [“Support for IBM MQ in containers”](#) on page 9 for details of what is supported when using the IBM MQ Operator.

## Building your own images and deployment code

This is the most flexible container solution, but it requires you to have strong skills in configuring containers, and to "own" the resultant container. If you aren't planning to use Red Hat OpenShift Container Platform, then you will need to build your own images and deployment code.

Samples for building your own images are available. See [“Preparing for IBM MQ by building your own container image”](#) on page 53.

See [“Support for IBM MQ in containers”](#) on page 9 for details of what is supported when building your own image and deployment code.

### Related reference

[“Support for IBM MQ in containers”](#) on page 9

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.

Below is a table which shows in detail how IBM MQ features are supported with the IBM MQ Operator, or when you build your own containers and deployment code.

### Notes:

- The pre-built IBM MQ container images on IBM Container Registry (icr.io and cp.icr.io) are only supported and eligible for fixes if used with the IBM MQ Operator.
- From IBM MQ Operator channel v3.2, Long Term Support (LTS) is renamed to Support Cycle 2 (SC2). This is because the only available LTS path for IBM MQ in containers is two years support under IBM Cloud Pak for Integration entitlement, and IBM Cloud Pak for Integration has adopted the term SC2. Here is the full picture of entitlement:
  - With IBM MQ entitlement, the IBM MQ Operator can deploy only the IBM MQ Continuous Delivery (CD) images.
  - With IBM Cloud Pak for Integration entitlement, the IBM MQ Operator can deploy CD or SC2 (formerly LTS) images.

It is not possible to "upgrade" the license of the pre-built IBM MQ Advanced for Developers image to a different license. The IBM MQ Operator will deploy different images, depending on which license is selected.

In this table, the following terms apply:

### "Container enablement code"

The executables **runmqserver**, **runmqintegrationserver**, **chkmqhealthy**, **chkmqready** and **chkmqstarted**. This code is provided as a sample, and is only supported as part of the pre-built containers when used with the IBM MQ Operator.

	Using IBM MQ Operator and a IBM Cloud Pak for Integration license	Using IBM MQ Operator and an IBM MQ Advanced license	Using IBM MQ Operator and an IBM MQ Advanced for Developers license	Pre-built IBM MQ Advanced for Developers image	Build-your-own container
<b>Supported platforms</b>	Supported on Red Hat OpenShift Container Platform only. Releases of Red Hat OpenShift Container Platform are no longer supported by IBM MQ once Red Hat stops support.  See “Version support for the IBM MQ Operator” on page 14 for more details.		Available on Red Hat OpenShift Container Platform only, but not supported.	Works on any Docker, containerd or cri-o platform, but not supported. See <a href="#">System Requirements for IBM MQ</a> for details.	Any Docker, containerd or cri-o platform. See <a href="#">System Requirements for IBM MQ</a> for details. Native HA is only supported on Kubernetes or Red Hat OpenShift Container Platform. The sample container image uses a Red Hat Universal Base Image (UBI), which includes Linux® libraries and utilities used by IBM MQ. The UBI is supported by Red Hat when run on Red Hat OpenShift. The <i>container enablement code</i> is not supported.
<b>CPU architectures</b>	Supported on amd64 and s390x z/Linux. Also supported on ppc64le Power Systems version 9 and above systems. Note that all nodes in the Red Hat OpenShift Container Platform cluster must use the same CPU architecture.		Available on amd64 and s390x z/Linux, but not supported. Also available on ppc64le Power Systems version 9 and above systems, but not supported. Note that all nodes in the Red Hat OpenShift Container Platform cluster must use the same CPU architecture.		As per IBM MQ software.

	Using IBM MQ Operator and a IBM Cloud Pak for Integration license	Using IBM MQ Operator and an IBM MQ Advanced license	Using IBM MQ Operator and an IBM MQ Advanced for Developers license	Pre-built IBM MQ Advanced for Developers image	Build-your-own container
<b>Duration of support</b>	<p>IBM Cloud Pak for Integration - Support Cycle 2 (formerly Long Term Support) or Continuous Delivery.<sup>1</sup></p> <p>CD operator and queue managers are supported until next IBM Cloud Pak for Integration CD or CP4I-SC2 release.</p> <p>CP4I-SC2 operator and queue managers are supported until next IBM Cloud Pak for Integration CP4I-SC2 release, plus a grace period to allow for upgrade.</p>	<p>Continuous Delivery stream only, for both the IBM MQ Operator, and queue managers.</p> <p>Each IBM MQ Operator and queue manager version is only supported until the next CD release.</p>	Not supported		<p>As per IBM MQ software. See <a href="#">IBM MQ FAQ for Long Term Support and Continuous Delivery releases</a>. The <i>container enablement code</i> is not supported.</p>
<b>Security fix availability</b>	Periodic fixes available as container images on IBM Container Registry				<p>Fixes to IBM MQ software are available as software on <a href="#">Fix Central</a>. The <i>container enablement code</i> is not supported.</p>

<sup>1</sup> The IBM MQ Operator is supported either as an IBM MQ CD release, or as a IBM Cloud Pak for Integration - Support Cycle 2 (formerly Long Term Support) release:

- IBM MQ 9.4.0.x container images deployed with IBM MQ Operator 3.2.x, when used as part of IBM Cloud Pak for Integration 16.1.0, are eligible for CP4I-LTS support. The latest Support Cycle 2 (SC2) release of the IBM MQ Operator is 3.2.2, and the latest SC2 container image is 9.4.0.0-r2.
- IBM MQ 9.4.0.x container images deployed with IBM MQ Operator 3.2.x, when used as part of IBM Cloud Pak for Integration 16.1.0, are eligible for CD support. The latest Continuous Delivery (CD) release of the IBM MQ Operator is 3.2.2, and the latest CD container image is 9.4.0.0-r2.

	Using IBM MQ Operator and a IBM Cloud Pak for Integration license	Using IBM MQ Operator and an IBM MQ Advanced license	Using IBM MQ Operator and an IBM MQ Advanced for Developers license	Pre-built IBM MQ Advanced for Developers image	Build-your-own container
<b>Interim fix availability</b>	Queue manager fixes available as software, and a custom image build is necessary.  IBM MQ Operator fixes are not available as interim fixes.		No interim fixes available.		Fixes to IBM MQ software are available as software on <a href="#">Fix Central</a> or via IBM Support. The <i>container enablement code</i> is not supported.
<b>Feature: Advanced Message Security</b>	Supported. Note that it's not easy to use server-side encryption, because the IBM MQ Operator does not directly allow you to specify your own key store for Advanced Message Security.		Available but not supported.		Supported as per IBM MQ software, but no sample available.
<b>Feature: Managed File Transfer</b>	Not available and not supported. However, you can use the IBM MQ Operator to provide one or more Coordination, Command, or Agent queue managers.			Not available and not supported.	Supported as per IBM MQ software, with sample for agent.
<b>Feature: MQTT</b>	Not available and not supported.				Supported as per IBM MQ software, but no sample available.
<b>Feature: AMQP</b>	Not available and not supported.				Supported as per IBM MQ software, but no sample available.
<b>Feature: REST API</b>	Available and supported.				Available and supported as per IBM MQ software.
<b>Feature: Replicated Data Queue Managers</b>	Not available and not supported. Replicated Data Queue Managers (RDQM) are tightly coupled with the Linux kernel, and are not supported in containers.				

	Using IBM MQ Operator and a IBM Cloud Pak for Integration license	Using IBM MQ Operator and an IBM MQ Advanced license	Using IBM MQ Operator and an IBM MQ Advanced for Developers license	Pre-built IBM MQ Advanced for Developers image	Build-your-own container
<b>Feature: Native HA</b>	Available and supported.		Available, but not supported.		Available only on Kubernetes and Red Hat OpenShift Container Platform. Supported as per IBM MQ software.
<b>Feature: Multi-instance queue managers</b>	Available and supported.		Available, but not supported.		Available and supported as per IBM MQ software.
<b>Feature: Types of recovery logs</b>	Circular logging or replicated logs only. Linear logging is not supported.				Available and supported as per IBM MQ software. You need to configure the <b>crtmqm</b> options.
<b>Feature: specifying custom command line options for <code>crtmqdir</code>, <code>crtmqm</code>, <code>strmqm</code> and <code>endmqm</code></b>	Not available and not supported. Most options can be configured using an INI file, however some cannot be configured, such as the use of linear logging.				Optional, depending on how you implement your <i>container enablement code</i> .
<b>Feature: Operating system (OS) users</b>	Not available and not supported.				Possible and supported as per IBM MQ software, if you install IBM MQ using RPMs, but no sample is available. Not recommended due to security risk.

**Note:** The phrase "supported as per IBM MQ software" means that the IBM Technical Support is limited to the core IBM MQ software that is running inside the container.

#### **Related concepts**

[IBM MQ FAQ for Long Term Support and Continuous Delivery releases](#)

## Related reference

### IBM Cloud Pak for Integration Software Support Lifecycle Addendum

## **Version support for the IBM MQ**

### Operator


A mapping between supported versions of IBM MQ, OpenShift Container Platform and IBM Cloud Pak for Integration.

- [“Available IBM MQ versions” on page 14](#)
- [“Compatible Red Hat OpenShift Container Platform versions” on page 14](#)
- [“IBM Cloud Pak for Integration versions” on page 14](#)
- [“Available IBM MQ versions in older operators” on page 15](#)
- [“Compatible OpenShift Container Platform versions for older operators” on page 15](#)

### Available IBM MQ versions

Operator channel	Operator version	IBM MQ versions						
		9.4.0	9.3.5	9.3.4	9.3.3	9.3.2	9.3.1	9.3.0
v32-sc2	3.2	CD and SC2	DEP	DEP	DEP	DEP	DEP	MIG

Key:

- CD: Continuous Delivery support is available.
- SC2: IBM Cloud Pak for Integration - Support Cycle 2 (formerly Long Term Support) is available.
- MIG: Only available during migration from an IBM Cloud Pak for Integration - Support Cycle 2 (formerly Long Term Support) operand to a Continuous Delivery operand.
- DEP:  Deprecated. As IBM MQ releases go out of support, they might still be configurable in the operator, but are no longer eligible for support and might be removed in future releases.

See [“Release history for IBM MQ Operator” on page 5](#) for full details of each version, including detailed features, changes and fixes in each version.

### Compatible Red Hat OpenShift Container Platform versions

Operator channel	Operator version	OpenShift Container Platform versions <sup>2</sup>		
		4.15	4.14	4.12
v3.2-sc2	3.2.0 onwards	SC2	SC2	SC2

Key:

- CD: Continuous Delivery support is available.
- SC2: IBM Cloud Pak for Integration - Support Cycle 2 (formerly Long Term Support) is available.
- EOS: No longer supported. Please migrate to a later OpenShift Container Platform version.

### IBM Cloud Pak for Integration versions

Supported for use as part of IBM Cloud Pak for Integration version 16.1.0, or independently:

<sup>2</sup> OpenShift Container Platform versions are subject to their own support dates. See [OpenShift Container Platform Life Cycle Policy](#) for more information.

- IBM MQ Operator 3.2.x

## Available IBM MQ versions in older operators

See [Available IBM MQ versions in the IBM MQ 9.3 documentation](#).

## Compatible OpenShift Container Platform versions for older operators

See [Compatible OpenShift Container Platform versions in the IBM MQ 9.3 documentation](#).

### **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 need to override settings on the queue manager `Route` created by the IBM MQ Operator, then you need to disable the route entirely setting `.spec.route.enabled` setting to "false", and then creating your own `Route`.
- Settings such as labels and annotations, as well as Pod settings like `securityContext`, can all be set on the `QueueManager` resource.
- In other cases, the IBM MQ Operator may 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.

### Related reference

[“License annotations when building your own IBM MQ container image” on page 158](#)

License annotations let you track usage based on the limits defined on the container, rather than on the underlying machine. You configure your clients to deploy the container with specific annotations that the IBM License Service then uses to track usage.

### Related information

[IBM Container Licenses](#)

The IBM MQ Operator runs in 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 90](#)). If this is not completed, then IBM MQ will need to be configured each time the container restarts.

 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



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 17.

- **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:
        enabled: true
```

The following example shows a snippet of a multi-instance queue manager configuration, with a non-default storage class, and with file storage requiring supplemental groups:

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

For information about storage considerations for Native HA queue managers, see [“Native HA”](#) on page 19.

**Note:** You can also configure supplemental groups with single-instance queue managers.

## Storage capacity



When you use the IBM MQ Operator you should try to ensure that you request volumes that are large enough for your ongoing needs. However, should 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 96.

## Encryption



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.

## **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” on page 19](#).

## 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

[High availability configurations](#)

### Related tasks

[“Configuring high availability for queue managers using the IBM MQ Operator” on page 72](#)

## CP4I MQ Adv. Native HA

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.

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

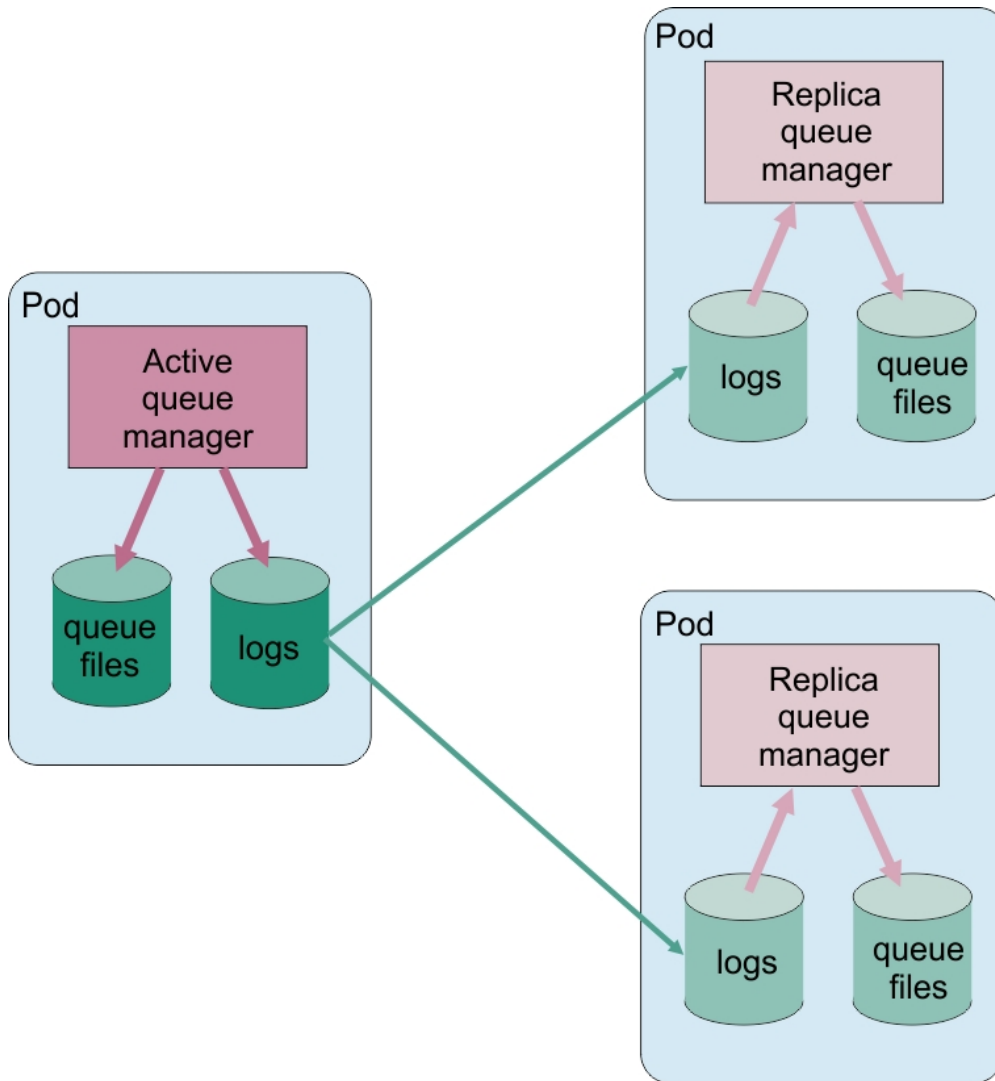


Figure 1. Example of Native HA configuration

**MQ Adv.** **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 REPLICATED 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 REPLICATED 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 REPLICATED 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 8](#)

There are multiple options for using IBM MQ in containers: you can choose to use the IBM MQ Operator, which uses pre-packaged container images, or you can build your own images and deployment code.

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](#)

OpenShift

CP4I

## 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 23](#)
  - [“Security constraints on the use of operating system users in containers” on page 24](#)
- [“Considerations for restricting network traffic to IBM MQ in containers” on page 24](#)

### 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 24](#).

### 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 67](#)


This example deploys a queue manager into the OpenShift Container Platform 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](#). 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 use on Red Hat OpenShift Container Platform.

```
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
    # 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.

**Note:** On AIX®, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status.

Customers should view the [IBM Crypto for C \(ICC\) certificate](#) and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the [NIST CMVP modules in process list](#).

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 is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the [NIST CMVP modules in process list](#).

## 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, 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.

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. Currently this endpoint is HTTP only. You can turn off the metrics endpoint to make the rest of IBM MQ FIPS compliant.
- 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.

**Note:** On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the [IBM Crypto for C \(ICC\) certificate](#) and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the [NIST CMVP modules in process list](#).

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 is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the [NIST CMVP modules in process list](#).

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:

## 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 OpenShift cluster without FIPS support, then the queue manager, IBM MQ web server, and Native HA components do not have their FIPS support automatically enabled. Only the x86-64 architecture is currently supported by the OpenShift platform for FIPS. For Power and Linux for IBM Z® architectures, OpenShift does not offer FIPS support. To explicitly enable FIPS support in the IBM MQ components for these architectures, 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:

```
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

**Note:** On AIX, Linux, and Windows, IBM MQ provides FIPS 140-2 compliance through the IBM Crypto for C (ICC) cryptographic module. The certificate for this module has been moved to the Historical status. Customers should view the [IBM Crypto for C \(ICC\) certificate](#) and be aware of any advice provided by NIST. A replacement FIPS 140-3 module is currently in progress and its status can be viewed by searching for it in the [NIST CMVP modules in process list](#).

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 is currently pending and its status can be viewed by searching for "Red Hat Enterprise Linux 9 - OpenSSL FIPS Provider" in the [NIST CMVP modules in process list](#).

## About this task

`MQ_ENABLE_FIPS` supports three values:

### 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 a sample queue manager YAML that describes enabling TLS and FIPS for the queue manager component:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  namespace: ibm-mq-fips
  name: ibm-mq-qm-ppcle
spec:
  license:
    accept: true
    license: L-EHXT-MQCRN9
    use: Production
  queueManager:
    name: PPCLEQM
    storage:
      queueManager:
        type: ephemeral
  template:
    pod:
      containers:
        - env:
            - name: MQ_ENABLE_FIPS
              value: "true"
          name: qmgr
  version: 9.4.0.0-r2
  web:
    enabled: false
  pki:
    keys:
      - name: ibm-mq-tls-certs
        secret:
          secretName: ibm-mq-tls-secret
          items:
            - tls.key
            - tls.crt
```

## 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. From Red Hat OpenShift Container Platform 4.11, 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.

The process limit in Kubernetes can be configured by a cluster administrator 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.

- **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.

## 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 IBM MQ Operator, then you prepare your Red Hat OpenShift Container Platform, cluster by installing the operator. See [“Installing and upgrading the IBM MQ Operator” on page 30](#)

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 53](#)

# Installing and upgrading the IBM MQ Operator

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

## 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

- [“Dependencies for the IBM MQ Operator” on page 30](#)
- [“Cluster-scoped permissions required by the IBM MQ Operator” on page 30](#)
- [“Verifying image signatures” on page 31](#)
- [“Installing the IBM MQ Operator” on page 31](#)
- [“Upgrading the IBM MQ Operator and queue managers” on page 41](#)
- [“Uninstalling the IBM MQ Operator” on page 51](#)

### **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. See [Deploying License Service](#) in the IBM Cloud Pak for Integration documentation.

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” on page 31](#) for more details on the dependencies.

### **Cluster-scoped permissions required by the IBM MQ Operator**

The IBM MQ Operator requires cluster-scoped permissions to manage admission webhooks and samples, and to read storage class and cluster version information.

The IBM MQ Operator requires the following cluster-scoped permissions:

- Permission to manage admission webhooks. This allows creating, retrieving, and updating specific webhooks that are used in the process of creating and managing containers provided by the Operator.
  - API Groups: **admissionregistration.k8s.io**
  - Resources: **validatingwebhookconfigurations**
  - Verbs: **get, delete**
- Permission to create and manage resources that are used in the Red Hat OpenShift console to provide samples and snippets when creating custom resources.
  - API Groups: **console.openshift.io**
  - Resources: **consoleyamlsamples**
  - Verbs: **create, get, update, delete**
- 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**

- Permission to read storage classes on the cluster. This allows the Operator to feed back any issues with selected storage classes in containers.
  - API Groups: **storage.k8s.io**
  - Resources: **storageclasses**
  - Verbs: **get, list**

**Note:** The IBM MQ Operator also requires namespace-scoped permissions. If the IBM MQ Operator is installed at a cluster scope, then the namespace-scoped permissions are present in all namespaces.

## **Verifying image signatures**

IBM MQ Operator and IBM MQ queue manager container images are digitally signed.

### About this task

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).

### Procedure

- Verify the signatures of IBM MQ Operator and IBM MQ queue manager container images:
  - See [Verifying image signatures](#) in the IBM Cloud Pak for Integration (CP4I) 16.1.0 documentation.

## **Installing the IBM MQ Operator**

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

### Before you begin

#### Important:

- This topic is for installing the IBM MQ Operator 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 CP4I”](#) on page 38.
- 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 8.

### About this task

The following steps represent the typical task flow for installing your IBM MQ Operator:

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 3.2 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 16](#). 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 using the following values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=3.2.2
```

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 33](#).

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 page 61.

### Related tasks

["Uninstalling the IBM MQ Operator"](#) on page 51

You can use the Red Hat OpenShift console or CLI to uninstall the IBM MQ Operator from Red Hat 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 [page 31](#).

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=3.2.2
export ARCH=ARCHITECTURE
```

where *ARCHITECTURE* refers to the architecture of the system on which you are deploying the IBM MQ Operator, and has a value of amd64, ppc64le, or s390x.

**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", in which case you can skip to step [“8”](#) on [page 34](#) "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](#).

- **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 page 31](#).

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](#).

## **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 page 31.

### 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.  
Review [“Version support for the IBM MQ Operator”](#) on page 14 to determine which operator channel to choose.
  - 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 page 31).

## **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 page 31.

## 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. Review [“Version support for the IBM MQ Operator”](#) on page 14 to determine which operator channel to choose.
7. Install the operator.

Use the following command, changing *ibm-mq-operator-channel* to match the channel for the version of the IBM MQ Operator you want to install, and changing *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.

```
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: ibm-operator-catalog
  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 page 31).

## Installing the IBM MQ Operator for use with CP4I

For use with the IBM Cloud Pak for Integration (CP4I), the IBM MQ Operator can be installed onto Red Hat OpenShift through the OpenShift console or command line interface (CLI).

## Before you begin

### Important:

- This topic is for installing the IBM MQ Operator for use with CP4I, or if you intend to deploy at least one of your queue managers using a CP4I license **only**. For instructions on installing the IBM MQ Operator for standalone use, see ["Installing the IBM MQ Operator"](#) on page 31.
- 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 8.

## About this task

The following steps represent the typical task flow for installing your IBM MQ Operator:

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 \(CP4I\) 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 3.2 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 16](#). 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. 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. Use the following values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=3.2.2
```

- 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 CP4I 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 foundational services
- 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=3.2.2
export ARCH=ARCHITECTURE
```

where *ARCHITECTURE* refers to your system architecture, and has a value of amd64, ppc64le, or s390x.

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 CP4I 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 foundational services
- 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 CP4I 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 CP4I license.
- The IBM MQ Console is enabled.
- You are using the CP4I 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 foundational services operator in the same installation mode as the CP4I Operator. For supported versions, see [Operator channel versions for this release](#).
- Install the CP4I 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 page 61.

### Related tasks

["Uninstalling the IBM MQ Operator"](#) on page 51

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

## **Upgrading the IBM MQ Operator and queue managers**

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

### 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 to the latest version on your current Operator channel, see ["Upgrading to an IBM MQ Operator channel latest security release"](#) on page 42.

- Option 2: **Upgrade the IBM MQ Operator for IBM MQ licenses.**

To upgrade deployments of the IBM MQ Operator where **only** IBM MQ licenses are used, see ["Upgrading the IBM MQ Operator"](#) on page 42.

- Option 3: **Upgrade the IBM MQ Operator for CP4I users.**

Upgrade deployments of the IBM MQ Operator 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 CP4I users” on page 47.](#)

## **Upgrading the IBM MQ Operator**

Upgrade deployments of the IBM MQ Operator where **only** IBM MQ licenses are used.

### Before you begin

**Important:** This task is for users of the IBM MQ Operator and **only** IBM MQ licenses. If you are an IBM Cloud Pak for Integration (CP4I) user, or you have deployed at least one of your queue managers using a CP4I license, see [“Upgrading the IBM MQ Operator for CP4I users” on page 47.](#)

### About this task

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

**Note:** Version 3.2.x of the IBM MQ Operator was released as both a CD and an SC2 release.

### Procedure

- Option 1: [“Upgrading to an IBM MQ Operator channel latest security release” on page 42](#)
- Option 2: [“Upgrading a 2.0.x LTS IBM MQ Operator to the 3.2.x SC2/CD channel” on page 44](#)
- Option 3: [“Upgrading a CD IBM MQ Operator to the 3.2.x SC2/CD channel” on page 44](#)

## **Upgrading to an IBM MQ Operator channel latest security release**

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 to the latest Security Release on the deployment's Channel. If this does not apply to your deployment, please refer to alternative upgrade paths described in [“Upgrading the IBM MQ Operator and queue managers” on page 41.](#)

### About this task

You first upgrade the catalogue source, then upgrade the queue managers. There are two options, 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 catalogue 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.](#)

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 45.

## Procedure

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

a) 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](#).

b) If you have your **Update approval** status for the IBM MQ Operator set to **Automatic**, your operator upgrades. If you have your **Update approval** set to **Manual**, follow these steps to 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.

c) Upgrade any IBM MQ queue managers.

Proceed to 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. If you do not have automatic upgrades set, then 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

Proceed to 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 49.

Operator channel	Latest IBM MQ queue manager
v3.2 (SC2/CD)	9.4.0.0-r2

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

## Before you begin

### Important:

- This task is for users of the IBM MQ Operator and **only** IBM MQ licenses. If you are an IBM Cloud Pak for Integration (CP4I) user, or you have deployed at least one of your queue managers using a CP4I license, see [“Upgrading the IBM MQ Operator for CP4I users”](#) on page 47.
- This topic is for upgrading deployments of the 2.0.x Long Term Support (LTS) IBM MQ Operator to the Support Cycle 2 (SC2) channel of IBM MQ Operator 3.2.x **only**. If this does not apply to your deployment, see the alternative upgrade paths described in [“Upgrading the IBM MQ Operator and queue managers”](#) on page 41.

To upgrade to IBM MQ Operator 3.2.2 you must be running Red Hat OpenShift Container Platform 4.12 or later. To verify the compatible versions for each IBM MQ Operator channel, see [“Compatible Red Hat OpenShift Container Platform versions”](#) on page 14. To upgrade the platform, see [Upgrading Red Hat OpenShift](#).

## Procedure

### 1. Mirror images (air-gap only).

You must mirror the IBM MQ images. Complete the steps at the following link, using only these values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=3.2.2
```

You should omit section 3.5 "Configure the cluster", because the connection to the image registry should have been set up during previous installs or upgrades.

**Link:** [Mirroring images for an air-gapped cluster](#).

### 2. Upgrade your IBM MQ Operator to 3.2.2.

See [“Upgrading the IBM MQ Operator using Red Hat OpenShift”](#) on page 47.

### 3. Upgrade the instances.

To receive the latest features and security fixes, upgrade the IBM MQ Operand (Queue Manager Container image) to the latest CD version (9.4.0.0-r2). See [“Upgrading an IBM MQ queue manager using Red Hat OpenShift”](#) on page 49.

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

## Before you begin

### Important:

- This task is for users of the IBM MQ Operator and **only** IBM MQ licenses. If you are an IBM Cloud Pak for Integration (CP4I) user, or you have deployed at least one of your queue managers using a CP4I license, see [“Upgrading the IBM MQ Operator for CP4I users”](#) on page 47.
- This topic is for upgrading Continuous Delivery (CD) deployments of the IBM MQ Operator prior to version 3.2.0, to version 3.2.2 **only**. If this does not apply to your deployment, see the alternative upgrade paths described in [“Upgrading the IBM MQ Operator and queue managers”](#) on page 41.

To upgrade to IBM MQ Operator 3.2.2 you must be running Red Hat OpenShift Container Platform 4.12 or later. To verify the compatible versions for each IBM MQ Operator channel, see [“Compatible Red Hat](#)

OpenShift Container Platform versions” on page 14. To upgrade the platform, see [Upgrading Red Hat OpenShift](#).

## Procedure

### 1. Optional: **Upgrade an IBM MQ Operator that is currently at a CD version prior to 3.0.0.**

If your IBM MQ Operator is currently at a CD version prior to 3.0.0, follow the relevant steps in [Migrating to the current CD channel of the IBM MQ Operator \(IBM MQ 9.3 documentation\)](#), then return here to upgrade to the latest CD version. Note that this is a mandatory prerequisite step before upgrading to version 3.2.2.

### 2. **Mirror images (air-gap only).**

You must mirror the IBM MQ images. Complete the steps at the following link, using only these values:

```
export OPERATOR_PACKAGE_NAME=ibm-mq
export OPERATOR_VERSION=3.2.2
```

You should omit section 3.5 "Configure the cluster", because the connection to the image registry should have been set up during previous installs or upgrades.

**Link:** [Mirroring images for an air-gapped cluster.](#)

### 3. **Upgrade your IBM MQ Operator to 3.2.2.**

See [“Upgrading the IBM MQ Operator using Red Hat OpenShift”](#) on page 47.

### 4. **Upgrade the instances.**

To receive the latest features and security fixes, upgrade the IBM MQ Operand (Queue Manager Container image) to the latest CD version (9.4.0.0-r2). See [“Upgrading an IBM MQ queue manager using Red Hat OpenShift”](#) on page 49.



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.

Installed Operators > Operator details

**IBM MQ**  
3.1.3 provided by IBM

Details | YAML | **Subscription** | Events | Queue Manager

**⚠ CatalogSource health unknown**  
This operator cannot be updated. The health of CatalogSource "ibm-operator-catalog" is unknown. It may have been disabled or removed from the cluster.  
[View CatalogSource](#)

**Subscription details**

<b>Update channel</b> ⓘ v3.1	<b>Update approval</b> ⓘ Automatic	<b>Upgrade status</b> ⚠ Cannot update CatalogSource not found
---------------------------------	---------------------------------------	---

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 45.

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 CP4I users**

Upgrade deployments of the IBM MQ Operator where an IBM Cloud Pak for Integration (CP4I) license is used.

## Before you begin

**Important:** This task is for CP4I users. This includes if you have deployed at least one of your queue managers under a CP4I license. If this does not apply to you, see [“Upgrading the IBM MQ Operator” on page 42](#).

## About this task

Complete one of the following options:

## Procedure

- **Option 1:** Upgrade deployments of the 2.0.x Long Term Support (LTS) IBM MQ Operator  
Follow the steps in [Upgrading from 2022.2 by generating an upgrade plan](#).
- **Option 2:** Upgrade a 3.0.x or 3.1.x deployment of the IBM MQ Operator  
Follow the steps in [Upgrading from 2023.4 by generating an upgrade plan](#).
- **Option 3:** Upgrade other deployments of the IBM MQ Operator  
Follow the relevant steps in [Migrating to the current CD channel of the IBM MQ Operator \(IBM MQ 9.3 documentation\)](#), then return here and proceed with **Option 2**. Note that this is a mandatory prerequisite step.

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

You can upgrade the IBM MQ Operator using either the Red Hat OpenShift web console or CLI.

## Procedure

To upgrade the IBM MQ Operator using Red Hat OpenShift, complete one of the following tasks:

- [“Upgrading the IBM MQ Operator using the Red Hat OpenShift console” on page 47](#)
- [“Upgrading the IBM MQ Operator using the Red Hat OpenShift CLI” on page 48](#)

### **Upgrading the IBM MQ Operator using the Red Hat OpenShift console**

The IBM MQ Operator can be upgraded using the Operator Hub.

## Before you begin

**Note:** The latest CD version of the IBM MQ Operator is 3.2.2, and is both an SC2 and CD version. For the latest IBM MQ Operator release notes, see [Release history for IBM MQ Operator](#).

Log in to your Red Hat OpenShift cluster console.

## Procedure

1. Review [“Version support for the IBM MQ Operator” on page 14](#) to determine which operator channel to upgrade to.
2. 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 45](#) before returning to complete step [“3” on page 48](#).

If you are using the IBM Operator catalog (some online installs only), proceed to step [“3” on page 48](#).

Follow the instructions in [“Adding the IBM MQ Operator catalog source” on page 33](#).

3. 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 will 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, and click **Save**.  
The operator will upgrade to the latest version available to the new channel. See [“Version support for the IBM MQ Operator” on page 14](#).

 *Upgrading the IBM MQ Operator using the Red Hat OpenShift CLI*  
The IBM MQ Operator can be upgraded from the command line.

## Before you begin

**Note:** The latest CD version of the IBM MQ Operator is 3.2.2, and is both an SC2 and CD version. For the latest IBM MQ Operator release notes, see [Release history for IBM MQ Operator](#).

Log into your cluster using **oc login**.

## Procedure

1. Review [“Version support for the IBM MQ Operator” on page 14](#) to determine which operator channel to upgrade to.
2. 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 45](#) before returning to complete step [“3” on page 48](#).

If you are using the IBM Operator catalog (some online installs only), proceed to step [“3” on page 48](#).

Follow the instructions in [“Adding the IBM MQ Operator catalog source” on page 33](#).

3. 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 or minor version, you will need 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
```

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

### Before you begin

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.



### Procedure

To upgrade the IBM MQ queue manager using Red Hat OpenShift, complete one of the following tasks:

- [“Upgrading an IBM MQ queue manager using the Red Hat OpenShift console” on page 49](#)
- [“Upgrading an IBM MQ queue manager using the Red Hat OpenShift CLI” on page 50](#)
- [“Upgrading an IBM MQ queue manager in Red Hat OpenShift using the Platform UI” on page 50](#)

### What to do next

To complete an IBM Cloud Pak for Integration upgrade, you might need to return to the IBM Cloud Pak for Integration documentation.

  *Upgrading an IBM MQ queue manager using the Red Hat OpenShift console*  
An IBM MQ queue manager, deployed using the IBM MQ Operator, can be upgraded in Red Hat OpenShift using the Operator Hub.

### Before you begin

**Note:** The latest version of the IBM MQ queue manager is 9.4.0.0-r2, and is both an SC2 and CD version. For the latest IBM MQ queue manager release notes, see [Release history for Queue Manager Container images for use with IBM MQ Operator](#).

- Log in to your Red Hat OpenShift cluster web console.
- Ensure that your IBM MQ Operator is using the desired Update Channel. See [“Upgrading the IBM MQ Operator using Red Hat OpenShift” on page 47](#).

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 in [Upgrading a CD IBM MQ Operator to the 3.2.x SC2/CD channel](#).

### Procedure

1. From the navigation pane, click **Operators > Installed Operators**.  
All installed Operators in the specified project are displayed.
2. Select the **IBM MQ Operator**.  
The **IBM MQ Operator** window is displayed.
3. Navigate to the **Queue Manager** tab.  
The **Queue Manager Details** window is displayed.
4. Select the queue manager that you want to upgrade.
5. Navigate to the YAML tab.

6. 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 for use with IBM MQ Operator”](#) on page 7 for a mapping of IBM MQ Operator versions and IBM MQ queue manager container images.

7. Save the updated queue manager YAML.

**OpenShift** **CP4I** *Upgrading an IBM MQ queue manager using the Red Hat OpenShift CLI*  
An IBM MQ queue manager, deployed using the IBM MQ Operator, can be upgraded in Red Hat OpenShift using the command line.

## Before you begin

**Note:** The latest version of the IBM MQ queue manager is 9.4.0.0-r2, and is both an SC2 and CD version. For the latest IBM MQ queue manager release notes, see [Release history for Queue Manager Container images for use with IBM MQ Operator](#).

You need to be a cluster administrator to complete these steps.

- Log in to the Red Hat OpenShift command line interface (CLI) using `oc login`.
- Ensure that your IBM MQ Operator is using the desired Update Channel. See [“Upgrading the IBM MQ Operator and queue managers”](#) on page 41.

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 in [Upgrading a CD IBM MQ Operator to the 3.2.x SC2/CD channel](#).

## Procedure

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 [“Version support for the IBM MQ Operator”](#) on page 14 for a mapping of channels to IBM MQ Operator versions and IBM MQ queue manager versions.

Use the following command:

```
oc edit queuemanager my_qmgr
```

where `my_qmgr` is the name of the QueueManager resource that you want to upgrade.

**CP4I** *Upgrading an IBM MQ queue manager in Red Hat OpenShift using the Platform UI*  
An IBM MQ queue manager, deployed using the IBM MQ Operator, can be upgraded in Red Hat OpenShift using the IBM Cloud Pak for Integration Platform UI.

## Before you begin

**Note:** The latest version of the IBM MQ queue manager is 9.4.0.0-r2, and is both an SC2 and CD version. For the latest IBM MQ queue manager release notes, see [Release history for Queue Manager Container images for use with IBM MQ Operator](#).

- Log in to the IBM Cloud Pak for Integration Platform UI in the namespace that contains the queue manager you want to upgrade.
- Ensure that your IBM MQ Operator is using the desired Update Channel. See [“Upgrading the IBM MQ Operator and queue managers”](#) on page 41.

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 in [Upgrading a CD IBM MQ Operator to the 3.2.x SC2/CD channel](#).

## Procedure

1. From the IBM Cloud Pak for Integration Platform UI home page, click the **Runtimes** tab.
2. Queue managers with available upgrades have a blue **i** next to the **Version**. Click the **i** to show **New version available**.
3. Click the three dots on the far right of the queue manager that you want to upgrade, then click **Change version**.
4. Under **Select a new channel or version**, select the required upgrade version.
5. Click **Change version**.

## Results

The queue manager is upgraded.

## **Uninstalling the IBM MQ Operator**

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

## 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
```

- h. If nothing else is using common services, you might want to uninstall the common services operator, and delete the operator group:

i) Uninstall the common services operator, by following the instructions in [Uninstalling foundational services](#) in the IBM Cloud Pak foundational services product documentation.

- ii) View the operator group:

```
oc get operatorgroup
```

- iii) Delete the operator group:

```
oc delete OperatorGroup operator_group_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
```

- g. Optional: If nothing else is using common services, you might want to uninstall the common services operator. To do so, follow the instructions in [Uninstalling foundational services](#) in the IBM Cloud Pak foundational services product documentation.

## 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 8

### About this task

#### Procedure

- ["General considerations when building your own queue manager image"](#) on page 53
- ["Building a sample IBM MQ queue manager container image"](#) on page 54
- ["Running local binding applications in separate containers"](#) on page 56
- [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 54](#)
  - [“Building a sample configured IBM MQ queue manager image” on page 54](#)

### ***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 WebSphere® MQ. You can also set the `LICENSE` variable 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 57 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 56 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, 2024.
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 dlrmqm QM1
AMQ7002: An error occurred manipulating a file.

docker exec container strmqweb
<file path>/mqrc.c[1152]
lpiObtainQMDetails --> 545261715
```

## **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 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 `ibmwebsphermqueue_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  $2 \times \text{HeartbeatInterval}$  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  $2 \times \text{HeartbeatInterval}$  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:

```
startmq 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)
```

## 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 60](#)
- [“Deploying and configuring queue managers using Helm” on page 101](#)

## Deploying and configuring queue managers using the IBM MQ Operator

Configuration examples; configuring HA; connecting from outside an OpenShift 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

- [“Examples for configuring a queue manager” on page 63.](#)
- [“Configuring high availability for queue managers using the IBM MQ Operator” on page 72.](#)
- [“Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster ” on page 81.](#)
- [“Integrating IBM MQ with IBM Instana tracing” on page 83.](#)
- [“Building an image with custom MQSC and INI files, using the Red Hat OpenShift CLI” on page 90.](#)
- [“Adding custom annotations and labels to queue manager resources” on page 92.](#)
- [“Disabling runtime webhook checks” on page 92.](#)
- [“Disabling default value updates to the queue manager specification” on page 93.](#)

## **Deploying a simple queue manager using the IBM MQ Operator**

This example deploys a "quick start" queue manager, which 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.

### 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 [“Licensing reference for mq.ibm.com/v1beta1” on page 135.](#) 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 [“Licensing reference for mq.ibm.com/v1beta1” on page 135](#) 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: 9.4.0.0-r2
  license:
    accept: false
    license: L-BMSF-5YDSLRL
    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 [“Licensing reference for mq.ibm.com/v1beta1” on page 135](#) 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 CP4I” on page 38](#).

To install a basic queue manager independently of 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
spec:
  version: 9.4.0.0-r2
  license:
```

```
accept: false
license: L-EHXT-MQCRN9
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 [“Licensing reference for mq.ibm.com/v1beta1”](#) on page 135 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 81

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 only available in the TLS protocol 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 deployed in a Red Hat OpenShift cluster”](#) on page 125

How to connect to the IBM MQ Console of a queue manager that has been deployed onto a Red Hat OpenShift Container Platform cluster.

[“Examples for configuring a queue manager”](#) on page 63

A queue manager can be configured by adjusting the contents of the QueueManager custom resource.

### **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 64
- [“Example: Configuring a queue manager with mutual TLS authentication”](#) on page 67

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 will deploy 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.

Example 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
```

Example QueueManager - 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: 9.4.0.0-r2
  license:
    accept: false
    license: L-EHXT-MQCRN9
    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
```

**Important:** If you accept the IBM MQ Advanced license agreement, change `accept: false` to `accept: true`. See [Licensing reference for mq.ibm.com/v1beta1](#) for details on the license.

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` will always be 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.

## OpenShift CP4I **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 `sample/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>
- **mac OS** 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

```
oc 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 67](#)

This example deploys a queue manager into the OpenShift Container Platform 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 73](#)

This example deploys a queue manager using the native high availability feature into the OpenShift Container Platform 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 78](#)

This example deploys a multi-instance queue manager using into the OpenShift Container Platform 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 into the OpenShift Container Platform 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 OpenShift Container Platform (OCP) project/namespace for this example.
- On the command line, log into the OCP cluster, and 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 into the OpenShift Container Platform. 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 65](#).
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-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 67. 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 the following custom resource YAML. Ensure you are in the namespace you created before you began this task, then enter the following YAML in the OCP web console, or using the command line. Check that the correct license is specified, and accept the license by changing `false` to `true`.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
spec:
  license:
    accept: false
    license: L-EHXT-MQCRN9
    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: 9.4.0.0-r2
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Note that the Secret *example-qm-tls* was created in Step “1” on page 67, and the ConfigMap *example-tls-configmap* was created in Step “2” on page 67

### 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:

```
oc 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 “Testing a mutual TLS connection to a queue manager from your laptop” on page 69.

## 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.

## from your laptop

After you have created a queue manager using the IBM MQ Operator, 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 Kubernetes cluster, such as your laptop.

### Before you begin

To complete this example, you must first have completed the following prerequisites:

- Install the IBM MQ client. You need the **amqspu`tc`** and **amqsget`c`** 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>
  - **mac OS** 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 65:
  - `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 67

### 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.

### Procedure

1. Confirm that the queue manager is running

The queue manager is now being deployed. Confirm it 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.hosts}}"
```

3. Create a 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, then you will also need to adjust that. 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 you used when creating the p12 file

```

Make sure to update the `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 81.](#)

#### 5. If you are using an arm64 Apple Mac, then 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 have been consumed, and are output. After a few seconds, the command exits.

## Results

Congratulations, you have successfully tested the connection a queue manager with TLS enabled, and shown that you can securely put and get messages to the queue manager from a client.

### **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
- 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 92.

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 for Production
    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 for Non-Production
    productMetric: VIRTUAL_PROCESSOR_CORE
```

## **Configuring high availability for queue managers using the IBM MQ Operator**

### About this task

#### Procedure

- [“Native HA” on page 19.](#)
- [“Example: Configuring Native HA using the IBM MQ Operator” on page 73.](#)
- [“Configuring a multi-instance queue manager using the IBM MQ Operator” on page 78.](#)

## **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, for example:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: nativeha-example
spec:
  license:
    accept: false
    license: L-EHXT-MQCRN9
    use: Production
  queueManager:
    availability:
      type: NativeHA
    version: 9.4.0.0-r2
```

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 73.](#)



## Related tasks

[“Example: Configuring Native HA using the IBM MQ Operator” on page 73](#)

This example deploys a queue manager using the native high availability feature into the OpenShift Container Platform 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 into the OpenShift Container Platform 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 OpenShift Container Platform (OCP) project/namespace for this example.
- On the command line, log into the OCP cluster, and 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 into the OpenShift Container Platform. 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 65](#).
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 73. 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 the following custom resource YAML. Ensure you are in the namespace you created before you began this task, then enter the following YAML in the OCP web console, or using the command line. Check that the correct license is specified, and accept the license by changing `false` to `true`.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
spec:
  license:
    accept: false
    license: L-EHXT-MQCRN9
    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: 9.4.0.0-r2
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt
```

Note that the Secret *example-qm-tls* was created in Step “1” on page 73, and the ConfigMap *example-nativeha-configmap* was created in Step “2” on page 73

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*).

### 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:

```
oc get qmgr exampleqm
```

### 5. Test the connection to the queue manager

To confirm the queue manager is configured and available, follow the steps in [“Testing a mutual TLS connection to a queue manager from your laptop”](#) on page 69.

## 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:

```
oc 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:

```
oc delete pod exampleqm-ibm-mq-value
```

### c) View the pod status again

Run the following command:

```
oc 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:

```
oc exec -t 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) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst2) ROLE(Replica) REPLADDR(9.20.123.46) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst3) ROLE(Replica) REPLADDR(9.20.123.47) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
```

### e) Test the connection to the queue manager again

To confirm the queue manager has recovered, follow the steps in [“Testing a mutual TLS connection to a queue manager from your laptop”](#) on page 69.

## Results

Congratulations, 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.

## **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 nodes might be out of date.

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.

The following status fields are used to report Native HA configuration status:

**ROLE**

Specifies the current role of the instance and is one of Active, Replica, or Unknown.

**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*.

**REPLADDR**

The replication address of the queue manager instance.

**CONNECTV**

Indicates whether the node is connected to the active instance.

**BACKLOG**

Indicates the number of KB that the instance is behind.

**CONNINST**

Indicates whether the named instance is connected to this instance.

**ALTDATE**

Indicates the date on which this information was last updated (blank if it has never been updated).

**ALTTIME**

Indicates the time at which this information was last updated (blank if it has never been updated).

**Procedure**

- Find the pods that are part of your queue manager.

```
oc get pod --selector app.kubernetes.io/instance=nativeha-qm
```

- Run the `dspm` in one of the pods

```
oc exec -t Pod dspm
```

```
oc rsh Pod
```

for an interactive shell, where you can run `dspm` directly.

- To determine whether a queue manager instance is running as the active instance or as a replica:

```
oc exec -t Pod dspm -o status -m QMgrName
```

An active instance of a queue manager named BOB would report the following status:

```
QMNAME(BOB)          STATUS(Running)
```

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:

```
oc exec -t Pod dspm -o nativeha -m QMgrName
```

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)
```

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)
```

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)
```

- To determine the Native HA operational status of all the instances in the Native HA configuration:

```
oc exec -t Pod dspmq -o nativeha -x -m QMgrName
```

If you issue this command on the node running the active instance of queue manager BOB, you might receive the following status:

```
QMNAME(BOB)          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) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst2) ROLE(Replica) REPLADDR(9.20.123.46) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst3) ROLE(Replica) REPLADDR(9.20.123.47) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
```

If you issue this command on a node running a replica instance of queue manager BOB, you might receive the following status, which indicates that one of the replicas is lagging behind:

```
QMNAME(BOB)          ROLE(Replica) INSTANCE(inst2) INSYNC(Yes) QUORUM(2/3)
INSTANCE(inst2) ROLE(Replica) REPLADDR(9.20.123.46) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst1) ROLE(Active) REPLADDR(9.20.123.45) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst3) ROLE(Replica) REPLADDR(9.20.123.47) CONNACTV(Yes) INSYNC(No) BACKLOG(435)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
```

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)
INSTANCE(inst1) ROLE(Unknown) REPLADDR(9.20.123.45) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ALTDATA() ALTTIME()
INSTANCE(inst2) ROLE(Unknown) REPLADDR(9.20.123.46) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ALTDATA() ALTTIME()
INSTANCE(inst3) ROLE(Unknown) REPLADDR(9.20.123.47) CONNACTV(No) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ALTDATA() 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)
```

## Related tasks

[“Example: Configuring Native HA using the IBM MQ Operator” on page 73](#)

This example deploys a queue manager using the native high availability feature into the OpenShift Container Platform 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 64 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 > MQ Adv. **Ending Native HA queue managers**

You can use the `endmqm` command to end an active or a replica queue manager that is part of a Native HA group.

### **Procedure**

- To end the active instance of a queue manager, see [Ending Native HA queue managers](#) in the [Configuring](#) section of this documentation.

## OpenShift > CP4I > MQ Adv. > Kubernetes **Configuring a multi-instance queue manager using the IBM MQ Operator**

This example deploys a multi-instance queue manager using into the OpenShift Container Platform 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 OpenShift Container Platform (OCP) project/namespace for this example.
- On the command line, log into the OCP cluster, and 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 into the OpenShift Container Platform. 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 65.

### 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 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-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 79. 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 using the following custom resource YAML. Ensure you are in the namespace you created before you began this task, then enter the following YAML in the OCP web console, or using the command line. Check that the correct license is specified, and accept the license by changing `false` to `true`.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: exampleqm
```

```

spec:
  license:
    accept: false
    license: L-EHXT-MQCRN9
    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: 9.4.0.0-r2
  pki:
    keys:
      - name: default
        secret:
          secretName: example-qm-tls
          items:
            - tls.key
            - tls.crt
            - ca.crt

```

Change `STORAGE_CLASS` to the storage class you identified in Step “1” on page 79.

Note that the Secret `example-qm-tls` was created in Step “2” on page 79, and the ConfigMap `example-miqm-configmap` was created in Step “3” on page 79

The availability type is set to `MultiInstance`, which causes persistent storage to be selected automatically.

#### 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:

```
oc get qmgr exampleqm
```

#### 6. Test the connection to the queue manager

To confirm the queue manager is configured and available, follow the steps in “[Testing a mutual TLS connection to a queue manager from your laptop](#)” on page 69.

#### 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:

```
oc 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:

```
oc delete pod exampleqm-ibm-mq-value
```

##### c) View the pod status again

Run the following command:

```
oc get pods --selector app.kubernetes.io/instance=exampleqm
```



d) View the queue manager status

Run the following command, specifying the full name of the other pod:

```
oc 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

To confirm the queue manager has recovered, follow the steps in [“Testing a mutual TLS connection to a queue manager from your laptop”](#) on page 69.

## Results

Congratulations, you have successfully deployed a multi-instance queue manager with mutual TLS authentication, and verified that it automatically recovers when the active Pod fails.

## 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 only available in the TLS protocol 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

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 CERTLABEL 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 CERTLABEL 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 tasks

“Connecting to the IBM MQ Console deployed in a Red Hat OpenShift cluster” on page 125  
How to connect to the IBM MQ Console of a queue manager that has been deployed onto a Red Hat OpenShift Container Platform cluster.

## Integrating IBM MQ with IBM Instana tracing

IBM Instana can be used to trace transactions within IBM Cloud Pak for Integration.

### Before you begin

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 85.

#### Note:

- This feature is supported only on Operands of IBM MQ version 9.3.1.0-r2 or later.
- You can run IBM Instana tracing on previous IBM MQ Operator and queue manager versions, but not natively. See [Configuring IBM MQ Tracing](#) in the IBM Instana documentation.

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 Instana 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 **Telemetry > Tracing > Instana**.
2. 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 Instana 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 **Telemetry > Instana**.
2. Expand the **Advanced configuration** dropdown list.
3. 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 **Telemetry > Instana**.
2. Expand the **Advanced configuration** dropdown list.
3. 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 page 61.

## Configuring authenticated IBM Instana monitoring with TLS

To be able to monitor a queue manager through an IBM Instana agent, you must configure both the agent and the queue manager.

### Before you begin

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 83](#)

IBM Instana can be used to trace transactions within IBM Cloud Pak for Integration.

## 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, both must have a certificate and corresponding private key.

### Before you begin

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](#).

### **Instana monitoring: Configuring agents**

Mount the keystore to the IBM Instana agents, then configure monitoring for a specific queue manager.

## Before you begin

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. In the `instana-agent` namespace, use the `oc edit daemonset instana-agent` command to edit the `instana-agent` daemonset to include the following additional `volumeMount` and `volume`:

```
volumeMounts:
- name: mq-key-jks-name
  subPath: application.jks
  mountPath: /opt/instana/agent/etc/application.jks
volumes:
- name: mq-key-jks-name
  secret:
    secretName: keystore_secret_name
```

### 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. The authentication for this connection is done using an [SSLPEERMAP](#).

## Before you begin

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 64](#).

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 83.
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, you must configure both the agent and the queue manager.

## Before you begin

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](#).

## **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:9.4.0.0-r2
      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:9.4.0.0-r2"
      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 page 61, 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 page 61](#)

This example deploys a "quick start" queue manager, which 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.

## 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

Runtime webhook checks 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"
```

## OpenShift CP4I Operator 2.1.0 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.

When a read-only root file system is enabled, two ephemeral volumes `Scratch` and `Tmp` are created, and mounted in `/run` and `/tmp` directories respectively in the container.

- The `Scratch` volume contains the files, keystores and other files used for configuring the queue manager.
- The `Tmp` volume contains diagnostic files, for example the queue manager RAS files.

Because these volumes are ephemeral, the files on these volumes are lost on pod restart.

The type of the volume created for queue manager data depends on the storage type. By default, a persistent volume is mounted. Or, if storage type is `ephemeral`, then an ephemeral volume is mounted. If the size of the data in the volume exceeds the value specified for the `sizeLimit` property, then Kubernetes can eject the container and create a new one.

A read-only root file system is not enabled by default. To enable it, complete the following steps:

## Procedure

1. Use the `spec.securityContext` API to enable the read-only root file system.

For your queue manager, set the **readOnlyRootFilesystem** property in [“.spec.securityContext” on page 147](#) to true.

IBM MQ Operator creates two ephemeral volumes, Scratch and Tmp.

2. 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 146](#), then an ephemeral volume is created and mounted.

3. 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 147](#). The default value is "100M".
  - For the Tmp ephemeral volume, set the **sizeLimit** property in [“.spec.queueManager.storage.tmp” on page 147](#). 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 146](#). The default value is "2Gi".

## **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 165](#). If you are deploying an IBM Cloud Pak for Integration license queue manager, 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 deployed in a Red Hat OpenShift cluster” on page 125](#).

## Procedure

1. **Create a password and encrypt it using `securityUtility`.**

A `ConfigMap` is used to store the credentials you use to access your queue manager. For improved security, you encode these credentials with the [`securityUtility` command](#).

Alternatively you can use a `Secret`, which protects credentials in the Kubernetes layer. However, monitoring or troubleshooting tools might expose the underlying file insecurely.

2. Optional: **Log into the Red Hat OpenShift command line interface (CLI).**

If using the OpenShift CLI, log in using `oc login`.

Alternatively you can use the 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 OpenShift console or the command line 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>
      <sslDefault sslRef="mqDefaultSSLConfig"/>
    </server>
```

b) Optional: If using the command line, apply the ConfigMap:

```
oc 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 through one of the following options:

- Option 1: Through the queue manager YAML

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
```

- 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 94.
- 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 94.

#### 5. Optional: **Apply configuration that enables the IBM MQ Console for an existing queue manager.**

Edit the YAML of the queue manager for which you are enabling the IBM MQ Console:

- a. On the OpenShift console select **Operators > Installed Operators**.
- b. Select your deployment of the IBM MQ Operator.
- c. Select **Queue Manager** and select the name of your queue manager.
- d. Select **YAML**.
- e. Replace the existing web section of the queue manager YAML with the following code:

```
...
web:
  enabled: true
  console:
    authentication:
      provider: manual
    authorization:
      provider: manual
  manualConfig:
    configMap:
      name: mqwebuserconfigmap
```

- f. Click **Save**.

You can now access the IBM MQ Console of your existing queue manager through the credentials specified in the ConfigMap created in step “3” on page 94.

## **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 OpenShift administrator can manually recover the Persistent Volume Claim state and cancel the expansion. See [Recovering from failure when expanding volumes](#) in the Red Hat OpenShift documentation.

### **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.
- OpenShift does 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:

```
oc 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:

```
oc 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 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.

<i>Table 1. Status conditions for storage</i>		
<b>CONDITION</b>	<b>MESSAGE</b>	<b>EXPLANATION</b>
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 <b>.allowVolumeExpansion</b> 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 create 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:

```
oc 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:

```
oc 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.

```
oc describe pod queue-manager-pod-name
```

- If your pod has an event or status condition that indicates a failed resize, see [Recovering from failure when expanding volumes in the Red Hat OpenShift documentation](#). 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.

### Stopping a queue manager ([mq.ibm.com/stop](http://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'`.

## Deploying and configuring queue managers using Helm

You can deploy and configure a queue manager on Kubernetes using the sample Helm chart.

### About this task

If you're not using Red Hat OpenShift Container Platform, then the IBM MQ Operator is not supported. 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 9](#)

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

## OpenShift CP4I-SC2 CD Migrating to the IBM MQ Operator

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.

### 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 102](#)

2. [“Extracting the queue manager configuration” on page 102](#)
3. Optional: [“Optional: Extracting and acquiring the queue manager keys and certificates” on page 103](#)
4. Optional: [“Optional: Configuring LDAP” on page 105](#)
5. Optional: [“Optional: Changing the IP addresses and host names in the IBM MQ configuration” on page 113](#)
6. [“Updating the queue manager configuration for a container environment” on page 114](#)
7. [“Selecting the target HA architecture for IBM MQ running in containers” on page 118](#)
8. [“Creating the resources for the queue manager” on page 118](#)
9. [“Creating the new queue manager on Red Hat OpenShift” on page 120](#)
10. [“Verifying the new container deployment” on page 124](#)

## **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**

This is the first step in the [“Migrating to the IBM MQ Operator” on page 101](#).

### **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 8](#).
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 90](#).
4. If your IBM MQ system includes High Availability, review the available options.  
See [“Planning high availability for IBM MQ in containers” on page 17](#).

### **What to do next**

You are now ready to [extract the queue manager configuration](#).

## **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**

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:

```
dmpmqcfig -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 `dspsmq` command to view the available queue managers on the machine. Here is sample `dspsmq` command output for a queue manager named `qm1`:

```
QMNAME(qm1)                STATUS(Running)
```

The `dspsmq` 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](#).

## **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

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,0=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,0=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

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) +
*  ALTDAT(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.

## 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

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 105](#).

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

- **CONNNAME** 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:

```
Open [icon]
backup.mqsc
*****
* Script generated on 2020-10-21 at 11.48.32
* Script generated by user ' CallumJackso' on host 'LAPTOP-VLQ
* 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) +
```

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'):

```
Open [icon]
backup.mqsc
*****
* Script generated on 2020-10-21 at 11.48.32
* Script generated by user ' CallumJackso' on host 'l
* 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) +
* COMMANDO(SYSTEM_ADMIN_COMMAND.QUEUE) +
  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](#).

## 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

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 105](#).

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:

```

Open [icon] *backup.mqsc
/tmp
OBJTYPE(PROCESS) +
AUTHADD(CRT)
SET AUTHREC +
PROFILE('@CLASS') +
PRINCIPAL('CallumJackson@AzureAD') +
OBJTYPE(QMGR) +
AUTHADD(CRT)
SET AUTHREC +
PROFILE('@CLASS') +
GROUP('mqm@LAPTOP-VLQKJ5UH') +
OBJTYPE(QMGR) +
AUTHADD(CRT)
SET AUTHREC +
PROFILE('SYSTEM.ADMIN.CHANNEL.EVENT') +
PRINCIPAL('CallumJackson@AzureAD') +
OBJTYPE(Queue) +
AUTHADD(BROWSE,CHG,CLR,DLT,DSP,GET,INQ,PUT,PASSALL,PASSID,SET,SETALL,SETID)
SET AUTHREC +
PROFILE('SYSTEM.ADMIN.CHANNEL.EVENT') +
GROUP('mqm@LAPTOP-VLQKJ5UH') +
OBJTYPE(Queue) +
AUTHADD(BROWSE,CHG,CLR,DLT,DSP,GET,INQ,PUT,PASSALL,PASSID,SET,SETALL,SETID)

* Script ended on 2020-10-26 at 11.48.32
* Number of Inquiry commands issued: 14
* Number of Inquiry commands completed: 14
* Number of Inquiry responses processed: 295
* QueueManager count: 1
* Queue count: 57
* NameList count: 3
* Process count: 1
* Channel count: 11
* AuthInfo count: 4
* Listener count: 4
* Service count: 2
* CommInfo count: 1
* Topic count: 6
* Subscription count: 1
* ChlAuthRec count: 3
* AuthRec count: 199
* Number of objects/records: 293
*****

```

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.

## 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

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](#).

## 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

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.

```

*backup.mqsc
** ALTDATA(2020-11-26) +
* ALTTIME(11.43.28) +
  REPLACE
DEFINE AUTHINFO('SYSTEM.DEFAULT.AUTHINFO.CRLLDAP') +
  AUTHTYPE(CRLLDAP) +
  CONNAME(' ') +
* ALTDATA(2020-10-26) +
* ALTTIME(11.43.28) +
  REPLACE
DEFINE LISTENER('SYSTEM.DEFAULT.LISTENER.LU62') +
  TRPTYPE(LU62) +
  CONTROL(MANUAL) +
* ALTDATA(2020-10-26) +
* ALTTIME(11.43.28) +
  REPLACE
DEFINE LISTENER('SYSTEM.DEFAULT.LISTENER.NETBIOS') +
  TRPTYPE(NETBIOS) +
  CONTROL(MANUAL) +
  LOCLNAME(' ') +
* ALTDATA(2020-10-26) +
* ALTTIME(11.43.28) +
  REPLACE
DEFINE LISTENER('SYSTEM.DEFAULT.LISTENER.SPX') +
  TRPTYPE(SPX) +
  CONTROL(MANUAL) +
* ALTDATA(2020-10-26) +
* ALTTIME(11.43.28) +
  REPLACE
DEFINE LISTENER('SYSTEM.DEFAULT.LISTENER.TCP') +
  TRPTYPE(TCP) +
  CONTROL(MANUAL) +
* ALTDATA(2020-10-26) +
* ALTTIME(11.43.28) +
  REPLACE
DEFINE SERVICE('SYSTEM.AMQP.SERVICE') +
  CONTROL(QMGR) +
  SERVTYPE(SERVER) +
  STARTCMD('+MQ_INSTALL_PATH+\bin\amqp.bat') +
  STARTARG('start -m +QMNAME+ -d "+MQ_Q_MGR_DATA_PATH+\.'
  STOPCMD('+MQ_INSTALL_PATH+\bin64\endmqsd.exe') +

```

## 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) +
  CCSTD(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.

## 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

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 120](#)).

### Procedure

1. Review the three options.

For a comprehensive description of these options, see [“Planning high availability for IBM MQ in containers” on page 17](#).

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](#).

Import the IBM MQ configuration, and the TLS certificates and keys, into the Red Hat OpenShift environment.

### Before you begin

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 103, 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

Deploy either a single instance or multi-instance queue manager on Red Hat OpenShift.

### Before you begin

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 118, 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:

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: qm1
spec:
  version: 9.4.0.0-r2
  license:
    accept: true
    license: L-BMSF-5YDSLRL
    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: 9.4.0.0-r2
  license:
    accept: true
    license: L-BMSF-5YDSLRL
    use: "Production"
```

This corresponds to the version and license information for the deployment. If you need to customize this, use the information provided in [“Licensing reference for mq.ibm.com/v1beta1”](#) on page 135.

```
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.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: qm1mi
spec:
  version: 9.4.0.0-r2
  license:
    accept: true
    license: L-BMSF-5YDSLRL
    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-efs	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 73.

Now that IBM MQ is deployed on Red Hat OpenShift, you can verify the environment using the IBM MQ samples.

### Before you begin

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 120](#). 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 [page 101](#).

## 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 81.

## 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 125.
- [“Viewing the status of Native HA queue managers”](#) on page 132.
- [“Manually ending Native HA queue manager instances”](#) on page 134.

OpenShift

CP4I

## Operating IBM MQ using the IBM MQ Operator

### Procedure

- [“Connecting to the IBM MQ Console deployed in a Red Hat OpenShift cluster”](#) on page 125.
- [“Monitoring when using the IBM MQ Operator”](#) on page 126.
- [“Backing up and restoring queue manager configuration using the Red Hat OpenShift CLI”](#) on page 132.

OpenShift

CP4I

## Connecting to the IBM MQ Console deployed in a Red Hat OpenShift cluster

How to connect to the IBM MQ Console of a queue manager that has been deployed onto a Red Hat OpenShift Container Platform cluster.

### About this task

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 of your MQ deployment --output jsonpath='{.status.adminUiUrl}'
```

If you are using an IBM Cloud Pak for Integration license, the IBM MQ Console uses Keycloak for identity and access management. See [Identity and Access management](#) in the IBM Cloud Pak for Integration documentation.

If you are using an IBM MQ license, then the IBM MQ Console is not pre-configured, and you need to configure it yourself. For more information, see [Configuring users and roles](#). For an example, see [“Configuring the IBM MQ Console with a basic registry using the IBM MQ Operator”](#) on page 94.

## Related tasks

[“Configuring a Route to connect to a queue manager from outside a Red Hat OpenShift cluster ” on page 81](#)

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 only available in the TLS protocol 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.

## **Giving permissions for the IBM MQ Console**

Permissions for the IBM MQ Console are managed differently based on your license usage.

## About this task

- If you are using an IBM Cloud Pak for Integration license, the IBM MQ Console uses Keycloak for identity and access management.
  - See [Identity and Access management](#) in the IBM Cloud Pak for Integration documentation.
  - If you previously configured users with IAM on older versions of the IBM MQ Operator, see [Migrating users from IAM to Keycloak](#).
- If you are using an IBM MQ license, then the IBM MQ Console is not pre-configured, and you need to configure it yourself.
  - For more 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 94](#).
  - Alternatively you can install the IBM Cloud Pak for Integration Operator to configure Keycloak as previously described.

## **Monitoring when using the IBM MQ Operator**

Queue managers managed by the IBM MQ Operator can produce metrics compatible with Prometheus.

You can view these metrics using the [Red Hat OpenShift Container Platform \(OCP\) monitoring stack](#). Open the **Metrics** tab in OCP, then click **Observe > Metrics**. Queue manager metrics are enabled by default, but can be disabled by setting `.spec.metrics.enabled` to `false`.

Prometheus is a time-series database and a rule evaluation engine for metrics. The IBM MQ containers expose a metrics endpoint which can be queried by Prometheus. The metrics are generated from the MQ system topics for monitoring and activity trace.

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. For more information, see [Enabling monitoring for user-defined projects](#). The IBM MQ Operator creates a `ServiceMonitor` when you create a `QueueManager` with metrics enabled, which the Prometheus Operator can then discover.

## **Metrics published when using the IBM MQ Operator**

Queue manager containers can publish metrics compatible with Red Hat OpenShift Monitoring.

Metric	Type	Description
<code>ibmmq_qmgr_commit_total</code>	counter	Commit count
<code>ibmmq_qmgr_cpu_load_fifteen_minute_average_percentage</code>	gauge	CPU load - fifteen minute average

<b>Metric</b>	<b>Type</b>	<b>Description</b>
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/MQCONN count
ibmmq_qmgr_failed_mqget_total	counter	Failed MQGET - count
ibmmq_qmgr_failed_mqinq_total	counter	Failed MQINQ count

<b>Metric</b>	<b>Type</b>	<b>Description</b>
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_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_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



<b>Metric</b>	<b>Type</b>	<b>Description</b>
ibmmq_qmgr_log_required_for_media_recovery_bytes	gauge	Log - bytes required for media recovery
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
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

<b>Metric</b>	<b>Type</b>	<b>Description</b>
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
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

<b>Metric</b>	<b>Type</b>	<b>Description</b>
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
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

## **Backing up and restoring queue manager configuration using the Red Hat OpenShift CLI**

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 using **oc login**.

### 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, you could run the following command, where *queue\_manager\_name* is the name of your `QueueManager` resource:

```
oc get pods --selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/instance=queue_manager_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.

```
oc 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, you could run the following command, where *queue\_manager\_name* is the name of your `QueueManager` resource:

```
oc get pods --selector app.kubernetes.io/name=ibm-mq,app.kubernetes.io/instance=queue_manager_name
```

- b) Run the **runmqsc** command on the pod, directing in the content of the `backup.mqsc` file.

```
oc exec -i pod_name -- runmqsc < backup.mqsc
```

## **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 to view the operational status of a queue manager instance on a node. 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 nodes might be out of date.

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.

The following status fields are used to report Native HA configuration status:

**ROLE**

Specifies the current role of the instance and is one of Active, Replica, or Unknown.

**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*.

**REPLADDR**

The replication address of the queue manager instance.

**CONNECTV**

Indicates whether the node is connected to the active instance.

**BACKLOG**

Indicates the number of KB that the instance is behind.

**CONNINST**

Indicates whether the named instance is connected to this instance.

**ALTDAT**

Indicates the date on which this information was last updated (blank if it has never been updated).

**ALTTIME**

Indicates the time at which this information was last updated (blank if it has never been updated).

**Procedure**

- To determine whether a queue manager instance is running as the active instance or as a replica:

```
dspmqr -o status -m QMgrName
```

An active instance of a queue manager named BOB would report the following status:

```
QMNAME(BOB)           STATUS(Running)
```

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 on the current node:

```
dspmqr -o nativeha -m QMgrName
```

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)
```

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)
```

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)
```

- To determine the Native HA operational status of all the instances in the Native HA configuration:

```
dspmqr -o nativeha -x -m QMgrName
```

If you issue this command on the node running the active instance of queue manager BOB, you might receive the following status:

```
QMNAME(BOB)                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) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst2) ROLE(Replica) REPLADDR(9.20.123.46) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst3) ROLE(Replica) REPLADDR(9.20.123.47) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
```

If you issue this command on a node running a replica instance of queue manager BOB, you might receive the following status, which indicates that one of the replicas is lagging behind:

```
QMNAME(BOB)                ROLE(Replica) INSTANCE(inst2) INSYNC(Yes) QUORUM(2/3)
INSTANCE(inst2) ROLE(Replica) REPLADDR(9.20.123.46) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst1) ROLE(Active) REPLADDR(9.20.123.45) CONNACTV(Yes) INSYNC(Yes) BACKLOG(0)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
INSTANCE(inst3) ROLE(Replica) REPLADDR(9.20.123.47) CONNACTV(Yes) INSYNC(No) BACKLOG(435)
CONNINST(Yes) ALTDATA(2022-01-12) ALTTIME(12.03.44)
```

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)
INSTANCE(inst1) ROLE(Unknown) REPLADDR(9.20.123.45) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ALTDATA() ALTTIME()
INSTANCE(inst2) ROLE(Unknown) REPLADDR(9.20.123.46) CONNACTV(Unknown) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ALTDATA() ALTTIME()
INSTANCE(inst3) ROLE(Unknown) REPLADDR(9.20.123.47) CONNACTV(No) INSYNC(Unknown)
BACKLOG(Unknown) CONNINST(No) ALTDATA() 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)
```

### Related reference

[dspmqr \(display queue managers\) command](#)

## MQ Adv. Manually ending Native HA queue manager instances

You can use the **endmqm** command to end an active or a replica queue manager that is part of a Native HA group.

### Procedure

- To end the active instance of a queue manager, see [Ending Native HA queue managers](#) in the Configuring section of this documentation.

## OpenShift CP4I API reference for the IBM MQ Operator

IBM MQ provides a Kubernetes Operator, which provides native integration with Red Hat OpenShift Container Platform.

### OpenShift CP4I API reference for mq.ibm.com/v1beta1

The v1beta1 API can be used to create and manage QueueManager resources.

### OpenShift CP4I CP4I-SC2 CD Licensing reference for 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:

Value of <code>spec.license.license</code>	Value of <code>spec.license.usage</code>	License information	Applicable IBM MQ versions
L-JTPV-KYG8TF	Production or NonProduction	<a href="#">IBM Cloud Pak for Integration 16.1.0</a>	9.4.0
L-BMSF-5YDSLRL	Production or NonProduction	<a href="#">IBM Cloud Pak for Integration Limited Edition 16.1.0</a>	9.4.0
L-EHXT-MQCRN9	Production	<a href="#">IBM MQ Advanced 9.4</a>	9.4.0
L-CLXQ-ADXTK3	Development	<a href="#">IBM MQ Advanced for Developers (Non-Warranted) 9.4</a>	9.4.0

Note that the license *version* is specified, which is not always the same as the version of IBM MQ.

#### Older license versions

See [Older license versions](#) in the IBM MQ 9.3 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/BdPZfq..>

Field	Description
<code>apiVersion</code> string	<code>APIVersion</code> 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: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources</a> .
<code>kind</code> string	<code>Kind</code> 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: <a href="https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds">https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds</a> .
<code>metadata</code>	
<code>spec QueueManagerSpec</code>	The desired state of the QueueManager.

Field	Description
<code>status</code> <a href="#">QueueManagerStatus</a>	The observed state of the QueueManager.

### .spec

The desired state of the QueueManager.

Appears in:

- [“QueueManager” on page 135](#)

Field	Description
<code>affinity</code>	Standard Kubernetes affinity rules. For more information, see <a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#affinity-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#affinity-v1-core</a> .
<code>annotations</code> <a href="#">Annotations</a>	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.
<code>imagePullSecrets</code> <a href="#">LocalObjectReference</a> 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 <a href="https://kubernetes.io/docs/concepts/containers/images#specifying-imagepullsecrets-on-a-pod">https://kubernetes.io/docs/concepts/containers/images#specifying-imagepullsecrets-on-a-pod</a> .
<code>labels</code> <a href="#">Labels</a>	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.
<code>license</code> <a href="#">License</a>	Settings that control your acceptance of the license, and which license metrics to use.
<code>pki</code> <a href="#">PKI</a>	Public Key Infrastructure settings, for defining keys and certificates for use with Transport Layer Security (TLS) or MQ Advanced Message Security (AMS).
<code>queueManager</code> <a href="#">QueueManagerConfig</a>	Settings for the Queue Manager container and underlying Queue Manager.
<code>securityContext</code> <a href="#">SecurityContext</a>	Security settings to add to the Queue Manager Pod's securityContext.
<code>telemetry</code> <a href="#">Telemetry</a>	Settings for Open Telemetry configuration. Requires MQ Operator 2.2.0 or higher.
<code>template</code> <a href="#">Template</a>	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.
<code>terminationGracePeriod</code> Seconds 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.



Field	Description
tracing <a href="#">TracingConfig</a>	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 <a href="#">WebServerConfig</a>	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 136](#)

### **.spec.imagePullSecrets**

LocalObjectReference contains enough information to let you locate the referenced object inside the same namespace.

Appears in:

- [“.spec” on page 136](#)

Field	Description
name string	Name of the referent. More info: <a href="https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names">https://kubernetes.io/docs/concepts/overview/working-with-objects/names/#names</a> TODO: Add other useful fields. apiVersion, kind, uid?.

### **.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 136](#)

### **.spec.license**

Settings that control your acceptance of the license, and which license metrics to use.

Appears in:

- [“.spec” on page 136](#)

Field	Description
accept boolean	Whether or not you accept the license associated with this software (required).
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 <a href="https://ibm.biz/BdPZfq">https://ibm.biz/BdPZfq</a> for valid values.

Field	Description
metric string	Setting that specifies which license metric to use. For example, <code>ProcessorValueUnit</code> , <code>VirtualProcessorCore</code> or <code>ManagedVirtualServer</code> . Defaults to <code>ProcessorValueUnit</code> when using an MQ license and <code>VirtualProcessorCore</code> when using a Cloud Pak for Integration license.
use string	Setting that controls how the software will to be used, where the license supports multiple uses. See <a href="https://ibm.biz/BdPZfq">https://ibm.biz/BdPZfq</a> 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 136](#)

Field	Description
keys <a href="#">PKISource</a> array	Private keys to add to the Queue Manager's key repository.
trust <a href="#">PKISource</a> 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 138](#)

Field	Description
name string	Name is used as the label for the key or certificate. Must be a lowercase alphanumeric string.
secret <a href="#">Secret</a>	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 138](#)

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 138](#)

Field	Description
name string	Name is used as the label for the key or certificate. Must be a lowercase alphanumeric string.
secret <a href="#">Secret</a>	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 138

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 136

Field	Description
availability <a href="#">Availability</a>	Availability settings for the Queue Manager, such as whether or not to use an active-standby pair or native high availability.
debug boolean	Whether or not to log debug messages from the container-specific code, to the container log. Defaults to false.
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 <code>IfNotPresent</code> .
ini <a href="#">INISource</a> array	Settings for supplying INI for the Queue Manager. Requires MQ Operator 1.1.0 or higher.
livenessProbe <a href="#">QueueManagerLivenessProbe</a>	Settings that control the liveness probe.
logFormat string	Which log format to use for this container. Use <code>JSON</code> for JSON-formatted logs from the container. Use <code>Basic</code> for text-formatted messages. Defaults to <code>Basic</code> .
metrics <a href="#">QueueManagerMetrics</a>	Settings for Prometheus-style metrics.
mjsc <a href="#">MQSCSource</a> 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 <code>metadata.name</code> . 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).

Field	Description
readinessProbe <a href="#">QueueManagerReadinessProbe</a>	Settings that control the readiness probe.
recoveryLogs <a href="#">RecoveryLogs</a>	Settings for MQ recovery logs. Requires MQ Operator 2.4.0 or higher.
resources <a href="#">Resources</a>	Settings that control resource requirements.
route <a href="#">Route</a>	Settings for the Queue Manager route. Requires MQ Operator 1.4.0 or higher.
startupProbe <a href="#">StartupProbe</a>	Settings that control the startup probe. Only applies to MultiInstance and NativeHA deployments. Requires MQ Operator 1.5.0 or higher.
storage <a href="#">QueueManagerStorage</a>	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 139

Field	Description
tls <a href="#">Tls</a>	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 <code>SingleInstance</code> for a single Pod, which will be restarted automatically (in some cases) by Kubernetes. Use <code>MultiInstance</code> for a pair of Pods, one of which is the active Queue Manager, and the other of which is a standby. Use <code>NativeHA</code> for native high availability replication (requires MQ Operator 1.5.0 or higher). Defaults to <code>SingleInstance</code> . See <a href="http://ibm.biz/BdqAQa">http://ibm.biz/BdqAQa</a> for more details.
updateStrategy string	The update strategy to use for MultiInstance and NativeHA Queue Managers. Use <code>RollingUpdate</code> to enable automatic rolling updates whenever the Queue Manager configuration changes. Use <code>OnDelete</code> 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 <code>RollingUpdate</code> . Requires MQ Operator 1.6.0 or higher.

### **.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 140

Field	Description
cipherSpec string	The name of the CipherSpec for NativeHA TLS.
secretName string	The name of the Kubernetes secret.

## **.spec.queueManager.ini**

Source of INI configuration files.

Appears in:

- [“.spec.queueManager” on page 139](#)

Field	Description
configMap <a href="#">ConfigMapINISource</a>	ConfigMap represents a Kubernetes ConfigMap that contains INI information.
secret <a href="#">SecretINISource</a>	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 141](#)

Field	Description
items array	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 141](#)

Field	Description
items array	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 139](#)

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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .
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.

Field	Description
timeoutSeconds integer	Number of seconds after which the probe times out. Defaults to 5 seconds. More info: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .

### **.spec.queueManager.metrics**

Settings for Prometheus-style metrics.

Appears in:

- [“.spec.queueManager” on page 139](#)

Field	Description
enabled boolean	Whether or not to enable an endpoint for Prometheus-compatible metrics. Defaults to true.

### **.spec.queueManager.mqsc**

Source of MQSC configuration files.

Appears in:

- [“.spec.queueManager” on page 139](#)

Field	Description
configMap <a href="#">ConfigMapMQSCSource</a>	ConfigMap represents a Kubernetes ConfigMap that contains MQSC information.
secret <a href="#">SecretMQSCSource</a>	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 142](#)

Field	Description
items array	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 142](#)

Field	Description
items array	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 139](#)

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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .
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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .

## **.spec.queueManager.recoveryLogs**

Settings for MQ recovery logs. Requires MQ Operator 2.4.0 or higher.

Appears in:

- [“.spec.queueManager” on page 139](#)

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 139](#)

Field	Description
limits <a href="#">Limits</a>	CPU & memory settings.
requests <a href="#">Requests</a>	CPU & memory settings.

## **.spec.queueManager.resources.limits**

CPU & memory settings.

Appears in:

- [“.spec.queueManager.resources” on page 143](#)

Field	Description
cpu	

Field	Description
memory	

### **.spec.queueManager.resources.requests**

CPU & memory settings.

Appears in:

- [“.spec.queueManager.resources” on page 143](#)

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 139](#)

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 139](#)

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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .
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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .

### **.spec.queueManager.storage**

Storage settings to control the Queue Manager's use of persistent volumes and storage classes.

Appears in:

- [“.spec.queueManager” on page 139](#)



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 <a href="#">QueueManagerOptionalVolume</a>	PersistentVolume details for MQ persisted data, including configuration, queues and messages. Required when using multi-instance Queue Manager.
queueManager <a href="#">QueueManagerVolume</a>	Default PersistentVolume for any data normally under /var/mqm. Will contain all persisted data and recovery logs, if no other volumes are specified.
recoveryLogs <a href="#">QueueManagerOptionalVolume</a>	Persistent volume details for MQ recovery logs. Required when using multi-instance Queue Manager.
scratch <a href="#">Scratch</a>	Settings for the Queue Manager's Scratch ephemeral volume. This volume will be mounted as the '/run' folder on the container. Only applicable if the root filesystem is set to read-only. Requires MQ Operator 3.0.0 or higher.
tmp <a href="#">Tmp</a>	Settings for the Queue Manager's Tmp ephemeral volume. This volume will be mounted on the container as the '/tmp' folder. The diagnostic data files, such as the zip file produced by the runmqras command, will be created in this volume. Only applicable if the root filesystem is set to read-only. 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 144

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.
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.

Field	Description
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 144](#)

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.
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 144](#)

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.scratch**

Settings for the Queue Manager's Scratch ephemeral volume. This volume will be mounted as the '/run' folder on the container. Only applicable if the root filesystem is set to read-only. Requires MQ Operator 3.0.0 or higher.

Appears in:

- [“.spec.queueManager.storage” on page 144](#)

Field	Description
sizeLimit string	Size limit of the ephemeral volume, including SI units. For example, 2Gi. Valid only when root filesystem is set to read-only. Requires MQ Operator 3.0.0 or higher.

### **.spec.queueManager.storage.tmp**

Settings for the Queue Manager's Tmp ephemeral volume. This volume will be mounted on the container as the '/tmp' folder. The diagnostic data files, such as the zip file produced by the runmqras command, will be created in this volume. Only applicable if the root filesystem is set to read-only. Requires MQ Operator 3.0.0 or higher.

Appears in:

- [“.spec.queueManager.storage” on page 144](#)

Field	Description
sizeLimit string	Size limit of the ephemeral volume, including SI units. For example, 2Gi. Valid only when root filesystem is set to read-only. 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 136](#)

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 rw-rw---- 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 <a href="https://docs.openshift.com/container-platform/latest/authentication/managing-security-context-constraints.html">https://docs.openshift.com/container-platform/latest/authentication/managing-security-context-constraints.html</a> .
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.telemetry**

Settings for Open Telemetry configuration. Requires MQ Operator 2.2.0 or higher.

Appears in:

- [“.spec” on page 136](#)

Field	Description
tracing <a href="#">Tracing</a>	Settings for Open Telemetry tracing.

### **.spec.telemetry.tracing**

Settings for Open Telemetry tracing.

Appears in:

- [“.spec.telemetry” on page 148](#)

Field	Description
instana <a href="#">Instana</a>	Settings for Instana tracing.

### **.spec.telemetry.tracing.instana**

Settings for Instana tracing.

Appears in:

- [“.spec.telemetry.tracing” on page 148](#)

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 136](#)

Field	Description
pod	Overrides for the template used for the Pod. See <a href="https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#podspec-v1-core">https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#podspec-v1-core</a> .

### **.spec.tracing**

Settings for tracing integration with the Cloud Pak for Integration Operations Dashboard.

Appears in:

- [“.spec” on page 136](#)

Field	Description
agent <a href="#">TracingAgent</a>	In Cloud Pak for Integration only, you can configure settings for the optional Tracing Agent.
collector <a href="#">TracingCollector</a>	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 149](#)

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 <a href="#">TracingProbe</a>	Settings that control the liveness probe.
readinessProbe <a href="#">TracingProbe</a>	Settings that control the readiness probe.

## **.spec.tracing.agent.livenessProbe**

Settings that control the liveness probe.

Appears in:

- [“.spec.tracing.agent” on page 149](#)

<b>Field</b>	<b>Description</b>
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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .
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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .

## **.spec.tracing.agent.readinessProbe**

Settings that control the readiness probe.

Appears in:

- [“.spec.tracing.agent” on page 149](#)

<b>Field</b>	<b>Description</b>
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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .
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: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .

## **.spec.tracing.collector**

In Cloud Pak for Integration only, you can configure settings for the optional Tracing Collector.

Appears in:

- [“.spec.tracing” on page 149](#)

<b>Field</b>	<b>Description</b>
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
<a href="#">livenessProbe</a> <a href="#">TracingProbe</a>	Settings that control the liveness probe.
<a href="#">readinessProbe</a> <a href="#">TracingProbe</a>	Settings that control the readiness probe.

### **.spec.tracing.collector.livenessProbe**

Settings that control the liveness probe.

Appears in:

- [“.spec.tracing.collector” on page 150](#)

Field	Description
<code>failureThreshold</code> integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
<code>initialDelaySeconds</code> integer	Number of seconds after the container has started before liveness probes are initiated. Defaults to 10 seconds. More info: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .
<code>periodSeconds</code> integer	How often (in seconds) to perform the probe. Defaults to 10 seconds.
<code>successThreshold</code> integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
<code>timeoutSeconds</code> integer	Number of seconds after which the probe times out. Defaults to 2 seconds. More info: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .

### **.spec.tracing.collector.readinessProbe**

Settings that control the readiness probe.

Appears in:

- [“.spec.tracing.collector” on page 150](#)

Field	Description
<code>failureThreshold</code> integer	Minimum consecutive failures for the probe to be considered failed after having succeeded. Defaults to 1.
<code>initialDelaySeconds</code> integer	Number of seconds after the container has started before liveness probes are initiated. Defaults to 10 seconds. More info: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .
<code>periodSeconds</code> integer	How often (in seconds) to perform the probe. Defaults to 10 seconds.
<code>successThreshold</code> integer	Minimum consecutive successes for the probe to be considered successful after having failed. Defaults to 1.
<code>timeoutSeconds</code> integer	Number of seconds after which the probe times out. Defaults to 2 seconds. More info: <a href="https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes">https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle#container-probes</a> .

### **.spec.web**

Settings for the MQ web server.

Appears in:

- [“.spec” on page 136](#)

Field	Description
console <a href="#">Console</a>	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 <a href="#">ManualConfig</a>	Settings for supplying web server XML configuration. Requires MQ Operator 3.0.0 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 151](#)

Field	Description
authentication <a href="#">Authentication</a>	Authentication settings for the MQ web console. Requires MQ Operator 3.0.0 or higher.
authorization <a href="#">Authorization</a>	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 152](#)

Field	Description
provider string	The authentication provider to use for the MQ web console. Use <code>integration-keycloak</code> to use single sign-on with the Cloud Pak for Integration Platform UI (Keycloak). Defaults to <code>integration-keycloak</code> if you use a Cloud Pak for Integration license, or <code>manual</code> if you use an MQ license. Use <code>manual</code> 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 152](#)

Field	Description
provider string	The authorization provider to use for the MQ web console. Use <code>integration-keycloak</code> to use roles provided by the Cloud Pak for Integration Keycloak. Use <code>manual</code> if you want to provide your own configuration. Defaults to <code>integration-keycloak</code> if you use a Cloud Pak for Integration license, or <code>manual</code> 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 151](#)

Field	Description
configMap <a href="#">ConfigMap</a>	ConfigMap represents a Kubernetes ConfigMap that contains web server XML configuration.
secret <a href="#">Secret</a>	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 152](#)

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 152](#)

Field	Description
name string	The name of the Kubernetes source.

### **.status**

The observed state of the QueueManager.

Appears in:

- [“QueueManager” on page 135](#)

Field	Description
adminUiUrl string	URL for the Admin UI.
availability <a href="#">Availability</a>	Availability status for the Queue Manager.
conditions <a href="#">QueueManagerStatusCondition</a> array	Conditions represent the latest available observations of the Queue Manager’s state.
endpoints <a href="#">QueueManagerStatusEndpoint</a> array	Information on the endpoints that this Queue Manager is exposing, such as API or UI endpoints.

Field	Description
metadata <a href="#">Metadata</a>	Metadata represents additional information for the Queue Manager, including Integration-Keycloak status.
name string	The name of the Queue Manager.
phase string	Phase of the Queue Manager's state.
versions <a href="#">QueueManagerStatusVersion</a>	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 153](#)

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 153](#)

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.
type string	Type of condition.

### **.status.endpoints**

QueueManagerStatusEndpoint defines the endpoints for the QueueManager.

Appears in:

- [“.status” on page 153](#)

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 153](#)

Field	Description
<code>integrationKeycloak</code> <code>IntegrationKeycloak</code>	QueueManagerStatusIntegrationKeycloak defines the Integration-Keycloak status for the QueueManager.

## **.status.metadata.integrationKeycloak**

QueueManagerStatusIntegrationKeycloak defines the Integration-Keycloak status for the QueueManager.

Appears in:

- [“.status.metadata” on page 155](#)

Field	Description
<code>clientName</code> string	

## **.status.versions**

Version of MQ being used, and other versions available from the IBM Entitled Registry.

Appears in:

- [“.status” on page 153](#)

Field	Description
<code>available</code> <code>QueueManagerStatusVersionA</code> <code>available</code>	Other versions of MQ available from the IBM Entitled Registry.
<code>reconciled</code> 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 155](#)

Field	Description
<code>channels</code> array	Channels which are available for automatically updating the MQ version.
<code>versions</code> <code>Versions</code> 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 155](#)

Field	Description
licenses <a href="#">Licenses</a> 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 <code>spec.version</code> field.

### **.status.versions.available.versions.licenses**

QueueManagerStatusLicense defines a license.

Appears in:

- [“.status.versions.available.versions” on page 155](#)

Field	Description
displayName string	Display name for the license.
link string	Link to the license content.
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. It might have some advisory **Warning** conditions.

The following conditions are defined for a QueueManager resource:

Table 2. Queue manager status conditions

Component	Condition type	Reason code	Message warning
QueueManager <sup>3</sup>	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].
	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.
	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.	

<sup>3</sup> 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 2. Queue manager status conditions (continued)

Component	Condition type	Reason code	Message warning
Pod <sup>4</sup>	Pending	PodPending	Pod for MQ queue manager is being deployed
	Error	PodFailed	Pod for MQ queue manager is being deployed
Storage <sup>5</sup>	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>]
	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 License annotations when building your own IBM MQ container image

License annotations let you track usage based on the limits defined on the container, rather than on the underlying machine. You configure your clients to deploy the container with specific annotations that the IBM License Service then uses to track usage.

When deploying a self-built IBM MQ container image, there are two common approaches to licensing:

- License the entire machine running the container.
- License the container based on the associated limits.

Both options are available to clients, 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

<sup>4</sup> 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.

<sup>5</sup> 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 alert 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.

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 161](#)
- ["IBM MQ Advanced" on page 161](#)
- ["IBM MQ for Non-Production Environment" on page 161](#)
- ["IBM MQ Advanced for Non-Production Environment" on page 161](#)
- ["IBM MQ Advanced for Developers" on page 161](#)

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 159](#).

- ["IBM MQ Container Multi Instance" on page 161](#)
- ["IBM MQ Advanced Container Multi Instance" on page 161](#)
- ["IBM MQ Container Multi Instance for Non-Production Environment" on page 162](#)
- ["IBM MQ Advanced Container Multi Instance for Non-Production Environment" on page 162](#)

## 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 162](#)
- ["IBM MQ Advanced with CP4I entitlement" on page 162](#)
- ["IBM MQ for Non-Production Environment with CP4I entitlement" on page 162](#)
- ["IBM MQ Advanced for Non-Production Environment with CP4I entitlement" on page 162](#)

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 159](#).

- ["IBM MQ Container Multi Instance with CP4I entitlement" on page 162](#)
- ["IBM MQ Advanced Container Multi Instance with CP4I entitlement" on page 163](#)
- ["IBM MQ Container Multi Instance for Non-Production Environment with CP4I entitlement" on page 163](#)
- ["IBM MQ Advanced Container Multi Instance for Non-Production Environment with CP4I entitlement" on page 163](#)

## Selecting the correct annotations for High Availability configurations

### IBM MQ Multi Instance

When you deploy a pair of queue managers in an IBM MQ multi-instance high availability configuration, you should use the same annotation on both 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 161](#)
  - [“IBM MQ Advanced Container Multi Instance” on page 161](#)
  - [“IBM MQ Container Multi Instance for Non-Production Environment” on page 162](#)
  - [“IBM MQ Advanced Container Multi Instance for Non-Production Environment” on page 162](#)
- IBM Cloud Pak for Integration entitlement
  - [“IBM MQ Container Multi Instance with CP4I entitlement” on page 162](#)
  - [“IBM MQ Advanced Container Multi Instance with CP4I entitlement” on page 163](#)
  - [“IBM MQ Container Multi Instance for Non-Production Environment with CP4I entitlement” on page 163](#)
  - [“IBM MQ Advanced Container Multi Instance for Non-Production Environment with CP4I entitlement” on page 163](#)

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<sup>6</sup>
- 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)<sup>6</sup>
- IBM MQ Container Multi Instance
  - 1 x IBM MQ **and** 1 x IBM MQ High Availability Replica **or**
  - 2 x IBM MQ<sup>6</sup>
- 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)<sup>6</sup>

### **IBM MQ Native HA**

If you are deploying three queue managers in a Native HA quorum, only the active instance consumes entitlement. 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 161](#)
  - [“IBM MQ Advanced for Non-Production Environment” on page 161](#)
- IBM Cloud Pak for Integration entitlement
  - [“IBM MQ Advanced with CP4I entitlement” on page 162](#)
  - [“IBM MQ Advanced for Non-Production Environment with CP4I entitlement” on page 162](#)

---

<sup>6</sup> This entitlement option is sub optimal and should only be used if no entitlement of the relevant High Availability Replica part is available.



## 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"
```

### 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"
```

### 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: "c661609261d5471fb4ff8970a36bceca"  
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"
```

## 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 9.4.0.0: [icr.io/ibm-messaging/mq:9.4.0.0-r2](https://icr.io/ibm-messaging/mq:9.4.0.0-r2)

### Quick reference

- License:
  - [Licensing reference for mq.ibm.com/v1beta1](#) 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 supported

### LANG

Set the language you want the license to be printed in.

### LICENSE

Set `accept` to agree to the IBM MQ Advanced for Developers license conditions.

Set `view` to view the license conditions.

### ▶ **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.

**V 9.4.0** ▶ **V 9.4.0** From IBM MQ 9.4.0, this variable is no longer provided. [The example YAML in this topic](#) shows how you can create this variable yourself and secure it with 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.

**V 9.4.0** ▶ **V 9.4.0** From IBM MQ 9.4.0, this variable is no longer provided. [The example YAML in this topic](#) shows how you can create this variable yourself and secure it with a secret.

### MQ\_DEV

Set `false` to stop the default objects being created.

### MQ\_ENABLE\_METRICS

Set `true` to generate Prometheus metrics for your queue manager.

### 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` and `mqsc`.

Default value is `qmgr,web`.

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.

## MQ\_QMGR\_NAME

Set the name you want your queue manager to be created with.

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=dev-admin-password=passw0rd --from-literal=dev-app-password=passw0rd
```

The following YAML uses these passwords when deploying a queue manager.

```
apiVersion: mq.ibm.com/v1beta1
kind: QueueManager
metadata:
  name: qm-dev
spec:
  license:
    accept: false
    license: L-CLXQ-ADXTK3
    use: Development
  web:
    enabled: true
  template:
    pod:
      containers:
        - env:
            - name: MQ_DEV
              value: "true"
            - name: MQ_CONNAUTH_USE_HTTP
              value: "true"
            - name: MQ_ADMIN_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: my-mq-dev-passwords
                  key: dev-admin-password
            - name: MQ_APP_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: my-mq-dev-passwords
                  key: dev-app-password
          name: qmgr
      queueManager:
        storage:
          queueManager:
            type: persistent-claim
          name: QUICKSTART
    version: 9.4.0.0-r2
```

## 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 166.](#)
- [“Troubleshooting problems with the IBM MQ Operator” on page 167.](#)

## 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.

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 Pod eviction

The SIGTERM signal is used by Kubernetes (and thus Red Hat OpenShift Container Platform) to gracefully terminate a Pod. 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:

- **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.
  - The Pod can be terminated because the Pod has failed its liveness probe. A liveness probe can be configured in Kubernetes to check that a Pod 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.

- **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, [Viewing system event information in an OpenShift Container Platform cluster.](#)
- **Cluster audit events.** See [Viewing audit logs in Red Hat OpenShift Container Platform.](#)
- **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 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 167](#)
- [“Troubleshooting: Gaining access to queue manager data” on page 169](#)

## OpenShift CP4I Collecting troubleshooting information for queue managers deployed with the IBM MQ Operator

Collecting troubleshooting information that should be provided to IBM Support when raising a new support case.

### Procedure

1. Collect cloud provider information.

This is the cloud provider that hosts your Red Hat OpenShift cluster (for example, IBM Cloud).

2. Collect architecture information.

The architecture of your Red Hat OpenShift cluster is one of the following:

- Linux for x86-64
- Linux on Power Systems (ppc64le)
- Linux for IBM Z

3. Collect IBM MQ deployment information.

- a) Log on to your Red Hat OpenShift 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, *QueueManager\_namespace* is the namespace where it is deployed, 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.

```
# 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: .involvedObject.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 $QM_NAMESPACE -o custom-columns="LAST SEEN: .lastTimestamp,TYPE: .type,REASON: .reason,KIND: .involvedObject.kind,NAME: .involvedObject.name,MESSAGE: .message" --field-selector involvedObject.name="${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

```

### Note:

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**.

### Related tasks

[Collecting troubleshooting information for IBM Support](#)

## Troubleshooting: Gaining access to queue manager data

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.

### Before you begin

To use the PVC inspector tool, you must have access to your queue manager namespace.

### About this task

To help with troubleshooting, you can access the data stored on the Persistent Volume Claims (PVCs) associated with a given queue manager. To do this, you use 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
```

## 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.

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](http://ibm.com)<sup>®</sup>, 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](http://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: