

9.1

IBM MQ in containers

IBM

Note

Before using this information and the product it supports, read the information in [“Notices” on page 51.](#)

This edition applies to version 9 release 1 of IBM® MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

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

© **Copyright International Business Machines Corporation 2007, 2026.**

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


Contents

IBM MQ in containers.....	5
Planning for IBM MQ in containers.....	5
Choosing how you want to use IBM MQ in containers.....	5
Support for IBM MQ certified containers.....	6
Support for building your own IBM MQ container images and charts.....	8
Storage considerations for IBM MQ Advanced certified container.....	8
High availability for IBM MQ Advanced certified container.....	10
User authentication and authorization for IBM MQ Advanced certified container.....	11
Installing and uninstalling the IBM MQ Operator on OpenShift.....	12
Installing the IBM MQ Operator using the OpenShift web console.....	12
Installing the IBM MQ Operator using the OpenShift CLI.....	13
Deploying IBM MQ certified containers.....	15
Preparing your OpenShift project for IBM MQ using the OpenShift CLI.....	15
Deploying a queue manager using the IBM Cloud Pak for Integration Platform Navigator.....	16
Deploying a queue manager using the OpenShift web console.....	17
Deploying a queue manager using the OpenShift CLI.....	18
Integrating with the IBM Cloud Pak for Integration Operations Dashboard.....	20
Building an image with custom MQSC and INI files, using the OpenShift CLI.....	21
Deploying IBM MQ certified containers using Helm.....	22
Deploying previous CD releases of IBM MQ into an IBM Cloud Private cluster.....	26
Adding previous CD releases of an IBM MQ image into an IBM Cloud Private cluster.....	28
Adding previous CD releases of an IBM MQ image into an IBM Cloud Kubernetes Service cluster..	28
Connecting to a queue manager deployed in an OpenShift cluster	29
Connecting to the IBM MQ Console deployed in an OpenShift cluster.....	31
Backing up and restoring queue manager configuration using the OpenShift CLI.....	31
Building your own IBM MQ container.....	32
Planning your own IBM MQ queue manager image using a container.....	33
Building a sample IBM MQ queue manager image using Docker.....	33
Running local binding applications in separate containers.....	36
API reference for the IBM MQ Operator.....	38
API reference for mq.ibm.com/v1beta1.....	38
Notices.....	51
Programming interface information.....	52
Trademarks.....	52

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 in the pre-packaged container provided in IBM MQ Advanced and IBM MQ Advanced for Developers. This IBM MQ Advanced certified container offers a supported image and Helm chart, and can be used to deploy a production-ready IBM MQ image into Red Hat® OpenShift®, IBM Cloud® Private, or IBM Cloud Kubernetes Service.

You can also run IBM MQ in an IBM Cloud Pak® for Integration container, or in a container you build yourself.

 For more information about the IBM MQ Advanced certified container, see the following links.

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 both the basic IBM MQ concepts (see [IBM MQ Technical overview](#)) as well as basic Kubernetes/OpenShift concepts (see [OpenShift Container Platform architecture](#)).

Procedure

- [“Choosing how you want to use IBM MQ in containers” on page 5.](#)
- [“High availability for IBM MQ Advanced certified container” on page 10.](#)
- [“User authentication and authorization for IBM MQ Advanced certified container” on page 11.](#)

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

Using the IBM MQ Advanced certified containers

If you are planning to deploy on Red Hat OpenShift Container Platform, then you probably want to use the certified containers. There are three varieties of certified container:

- IBM MQ Advanced certified container for IBM Cloud Pak for Integration. This is a separate IBM product that includes a version of a certified container.
- IBM MQ Advanced certified container
- IBM MQ Advanced for Developers certified container (unwarranted)

IBM MQ 9.1.4 and earlier CD releases were also supported on IBM Cloud Private and IBM Cloud Kubernetes Service.

Note that the certified containers are evolving rapidly, and are therefore only supported under [Continuous Delivery](#) releases.

The certified containers include both pre-built container images, as well as deployment code for running on Red Hat OpenShift Container Platform. From IBM MQ 9.1.5 onwards, queue managers are managed using an IBM MQ Operator. Prior versions of IBM MQ, up to and including version 9.1.5, are managed using Helm charts.

Some IBM MQ features are not supported when using the certified containers. You will need to build your own images and charts if you want to do any of the following:

- Use the REST APIs for administration or messaging
- Use any of the following MQ components:
 - Managed File Transfer Agents and its resources. However you can use the certified containers to provide one or more Coordination, Command, or Agent queue managers.
 - AMQP
 - IBM MQ Bridge to Salesforce
 - IBM MQ Bridge to blockchain (not supported in containers)
- Use the web server when you're deploying using Helm charts (except for IBM Cloud Pak for Integration)
- Customize options used with **crtmqm**, **strmqm** and **endmqm**, such as configuring recovery logs

Building your own images and charts

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 [“Building your own IBM MQ container” on page 32](#). The Helm charts provided as part of the certified containers are published on GitHub, and can be used as samples for when you are building your own images:

- [Helm chart for IBM MQ Advanced certified container](#)
- [Helm chart for IBM MQ Advanced for Developers certified container](#)

Related concepts

[“Support for IBM MQ certified containers” on page 6](#)


The IBM MQ certified containers are only supported in certain Kubernetes environments

[“Support for building your own IBM MQ container images and charts” on page 8](#)

Information to consider if you are using containers on a Linux system.

Linux Support for IBM MQ certified containers

The IBM MQ certified containers are only supported in certain Kubernetes environments

 For CD release V9.1.4 and later, the IBM MQ Advanced certified container is supported for use with Red Hat OpenShift. See [“Deploying a queue manager using the Helm CLI” on page 24](#).

CD releases earlier than V9.1.4 were supported in the following Kubernetes environments:

- IBM Cloud Kubernetes Service
- IBM Cloud Private
- IBM Cloud Private with Red Hat OpenShift

For specific supported versions of Kubernetes, see the files `qualification.yaml` and `Chart.yaml` inside a downloaded IBM MQ Advanced Helm chart. These versions vary from release to release.

The IBM MQ Advanced certified container is only supported when deployed using the IBM MQ Operator or when using one of the following Helm charts:

- `ibm-mqadvanced-server-prod`

- `ibm-mqadvanced-server-integration-prod` in the IBM Cloud Pak for Integration

Note: The use of Helm charts is deprecated, following the release of the IBM MQ Operator.

Because container technology is evolving rapidly, the IBM MQ Advanced certified container is only supported on the latest version of the platforms that this chart supports at the time of release. If you want to use an older platform version, then you might need to use an older version of the IBM MQ Advanced certified container.

The IBM MQ Advanced certified container image is based on IBM MQ Continuous Delivery (CD) releases. These are supported for up to one year, or for two CD releases, whichever is longer. Long Term Support releases of IBM MQ are not available as a certified container.

From IBM MQ Advanced certified container V4.0 onwards, the image provides an installation of IBM MQ on a Red Hat Universal Base Image (UBI), which includes key Linux libraries and utilities used by IBM MQ. The UBI is supported by Red Hat when run on a Red Hat Enterprise Linux host. Earlier versions of the IBM MQ Advanced certified container used an unsupported Ubuntu base image.

Related concepts

“Support for building your own IBM MQ container images and charts” on page 8
Information to consider if you are using containers on a Linux system.

Linux → MQ Adv. → CD **Version support for the IBM MQ Advanced certified container**

A set of tables showing the mapping between supported versions of the IBM MQ Advanced certified container, IBM MQ, IBM Cloud Kubernetes Service, IBM Cloud Pak for Integration, and IBM Cloud Private.

IBM MQ Operator

V 9.1.5

The IBM MQ Operator is supported for use as part of IBM Cloud Pak for Integration version 2020.2, or independently, with IBM MQ version 9.1.5 and above.

The IBM MQ Operator is supported on Red Hat OpenShift Container Platform version 4.4 or above.

IBM MQ Advanced certified container **V 9.1.5** (Helm chart) – deprecated

Includes the Helm chart `ibm-mqadvanced-server-prod`.

V 9.1.5 From IBM MQ Advanced certified container V5.0.x onwards, the Helm chart, image, and fixes, are shipped via the IBM Entitled Catalog and Registry. Earlier versions were shipped via Passport Advantage®, and fix releases are available from IBM Fix Central.

Version	IBM MQ version	End of support	Supported platforms
6.0.x	9.1.5 Continuous Delivery Release	March 2021	Detailed System Requirements
5.0.x	9.1.4 Continuous Delivery Release	December 2020	Detailed System Requirements
4.1.x	9.1.3 Continuous Delivery Release	July 2020	Detailed System Requirements

IBM MQ Advanced certified container software for the IBM Cloud Pak for Integration

V 9.1.5 (Helm chart) – deprecated

Includes the Helm chart `ibm-mqadvanced-server-integration-prod`.

Table 2. Version support for the IBM MQ Advanced certified container software for the IBM Cloud Pak for Integration

Version	IBM MQ version	IBM Cloud Pak for Integration version
6.0.x	9.1.4 Continuous Delivery Release	2020.1.1 (System Requirements)
5.0.x	9.1.3 Continuous Delivery Release	2019.4.1 (System Requirements)
4.1.x	9.1.3 Continuous Delivery Release	2019.3.2.2 (System Requirements)
4.0.x	9.1.3 Continuous Delivery Release	2019.3.2 (System Requirements)
3.0.x	9.1.3 Continuous Delivery Release	2019.3.1 (System Requirements)

See the [IBM Cloud Pak for Integration release notes](#) for supported version information.

Linux Support for building your own IBM MQ container images and charts

Information to consider if you are using containers on a Linux system.

- The base image used by the container image must use a Linux operating system that is supported.
- You must use the IBM MQ installers to install the product inside the container image.
- For a list of supported packages, see [IBM MQ rpm components for Linux systems](#).
- **V 9.1.0** The following packages are not supported:
 - MQSeriesBCBridge
 - MQSeriesRDQM
- The queue manager data directory (`/var/mqm` by default) must be stored on a container volume that keeps persistent state.

Important: You cannot use the union file system.

You must either mount a host directory as a data volume, or use a data volume container. For more information, see [Manage data in containers](#).

- You must be able to run IBM MQ control commands, such as `endmqm`, within the container.
- You must be able to get files and directories from within the container for diagnostic purposes.
- **V 9.1.0** You can use namespacing to share the namespaces of the container for the queue manager with other containers, in order to locally bind applications to a queue manager running in separate containers. For more information, see [“Running local binding applications in separate containers”](#) on page 36.

Related concepts

[“Support for IBM MQ certified containers”](#) on page 6

The IBM MQ certified containers are only supported in certain Kubernetes environments

Linux MQ Adv. V 9.1.5 CD Storage considerations for IBM MQ

Advanced certified container

The IBM MQ Advanced certified container runs in two storage modes:

- **Ephemeral storage** is used when all container state can be disposed of 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 queue manager state will be lost on restart. This includes:

- All messages
- All queue manager to queue manager communication state (channel message sequence numbers)
- The queue manager's MQ Cluster identity
- 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 queue manager configuration 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 OpenShift CLI” on page 21](#)). 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 persistence storage to assure the queue manager retains its persistent messages and configuration after a restart. Therefore, this is the default behavior. Due to the various storage providers and different capabilities each support, this often means that customization of the configuration is required. The below outlines the common fields that customize the MQ storage configuration in the v1beta1 API:

- `spec.queueManager.availability` controls the availability mode. If you are using `SingleInstance` you only require `ReadWriteOnce` storage, while `multiInstance` requires 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 [“High availability for IBM MQ Advanced certified container” on page 10](#)
- `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:
        enabled: true
        class: ibmc-file-gold-gid
        persistedData:
          enabled: true
          class: ibmc-file-gold-gid
        recoveryLogs:
          enabled: true
          class: ibmc-file-gold-gid
      securityContext:
        supplementalGroups: [99]
```

Linux

MQ Adv.

CD

High availability for IBM MQ Advanced certified container

You have two main choices for high availability with IBM MQ Advanced certified container: **Multi-instance queue manager** (which is an active-standby pair, using a shared, networked file system) and **Single resilient queue manager** (which offers a simple approach for HA using networked storage).

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.

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 container is ready. This is configurable in the Helm chart.
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 in the Helm chart.
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 in the Helm chart.
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. If the 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.

Related information

[High availability configurations](#)

Linux

MQ Adv.

CD

User authentication and authorization for IBM MQ Advanced certified container

IBM MQ can be configured to use LDAP users and groups for authorization. This is the recommended approach for the IBM MQ Advanced certified container.

In a multi-tenant containerized environment such as Red Hat OpenShift Container Platform, security constraints are put in place to prevent potential security issues. For example, in Red Hat OpenShift Container Platform the default `SecurityContextConstraints` (called `restricted`) uses a randomized user ID, discouraging any users local to the container itself. IBM MQ typically uses privilege escalation to check the passwords of users, which is also not recommended in multi-tenant container environments. For these reasons, the use of users defined on the operating system libraries inside a running container is not supported in the IBM MQ certified containers.

You need to configure your queue manager to use LDAP for user authentication and authorization. For information about configuring IBM MQ to do this, see [Connection authentication: User repositories and LDAP authorization](#)

Installing and uninstalling the IBM MQ Operator on OpenShift

The IBM MQ Operator can be installed onto OpenShift using the Operator Hub.

Before you begin

Procedure

- [“Installing the IBM MQ Operator using the OpenShift CLI” on page 13.](#)
- [“Installing the IBM MQ Operator using the OpenShift web console” on page 12.](#)

Installing the IBM MQ Operator using the OpenShift web console

The IBM MQ Operator can be installed onto OpenShift using the Operator Hub.

Before you begin

Log in to your OpenShift cluster's web console.

Procedure

1. Add the IBM Common Services operators to the list of installable operators
 - a) Click the plus icon. You see the **Import YAML** dialog box.
 - b) Paste the following resource definition in the dialog box.

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: opencloud-operators
  namespace: openshift-marketplace
spec:
  displayName: IBMCS Operators
  publisher: IBM
  sourceType: grpc
  image: docker.io/ibmcom/ibm-common-service-catalog:latest
  updateStrategy:
    registryPoll:
      interval: 45m
```

- c) Click **Create**.
2. Add the IBM operators to the list of installable operators
 - a) Click the plus icon. You see the **Import YAML** dialog box.
 - b) Paste the following resource definition in the dialog box.

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

- c) Click **Create**.
3. 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 which does not already exist.

- a) From the navigation pane, click the **Home > Projects**.
The Projects page is displayed.
 - b) Click **Create Project**. A Create Project area is displayed.
 - c) Enter details of the namespace that you are creating. For example, you can specify "ibm-mq" as the name.
 - d) Click **Create**. The namespace for your IBM MQ Operator is created.
4. Install the IBM MQ Operator.
- a) From the navigation pane, click **Operators > OperatorHub**.
The OperatorHub page is displayed.
 - b) In the **All Items** field, enter "IBM MQ".
The IBM MQ catalog entry is displayed.
 - c) Select **IBM MQ**.
The IBM MQ window is displayed.
 - d) Click **Install**.
You see the Create Operator Subscription page.
 - e) Set Installation Mode to either the specific namespace that you created, or the cluster wide scope
 - f) Click **Subscribe**.
You will see IBM MQ on the Installed Operators page.
 - g) Check the status of the Operator on the Installed Operators page, the status will change to Succeeded when the installation is complete.

What to do next

["Deploying IBM MQ certified containers" on page 15](#)

Linux MQ Adv. V 9.1.5 CD Installing the IBM MQ Operator using the OpenShift CLI

The IBM MQ Operator can be installed onto OpenShift using the Operator Hub.

Before you begin

Log into the OpenShift command line interface (CLI) using **oc login**. For these steps, you will need to be a cluster administrator.

Procedure

1. Create an OperatorSource for the IBM Common Services operators

- a) Create a YAML file defining the OperatorSource resource

Create a file called "operator-source-cs.yaml" with the following contents:

```
apiVersion: operators.coreos.com/v1alpha1
kind: CatalogSource
metadata:
  name: opencloud-operators
  namespace: openshift-marketplace
spec:
  displayName: IBMCS Operators
  publisher: IBM
  sourceType: grpc
  image: docker.io/ibmcom/ibm-common-service-catalog:latest
  updateStrategy:
```

```
registryPoll:
  interval: 45m
```

b) Apply the `OperatorSource` to the server.

```
oc apply -f operator-source-cs.yaml -n openshift-marketplace
```

2. Create an `OperatorSource` for the IBM operators

a) Create a YAML file defining the `OperatorSource` resource

Create a file called "operator-source-ibm.yaml" with the following contents:

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

b) Apply the `OperatorSource` to the server.

```
oc apply -f operator-source-ibm.yaml -n openshift-marketplace
```

3. 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 which does not already exist.

```
oc new-project ibm-mq
```

4. View the list of Operators available to the cluster from the OperatorHub

```
oc get packagemanifests -n openshift-marketplace
```

5. Inspect the IBM MQ Operator to verify its supported `InstallModes` and available Channels

```
oc describe packagemanifests ibm-mq -n openshift-marketplace
```

6. Create an `OperatorGroup` object YAML file

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 intend to install uses the `AllNamespaces`, then the `openshift-operators` namespace already has an appropriate `OperatorGroup` in place.

However, if the Operator uses the `SingleNamespace` mode and you do not already have an appropriate `OperatorGroup` in place, you must create one.

a) Create a file called "mq-operator-group.yaml" with the following contents:

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: <operatorgroup_name>
  namespace: <namespace>
spec:
  targetNamespaces:
  - <namespace>
```

b) Create the `OperatorGroup` object

```
oc apply -f mq-operator-group.yaml
```

7. Create a Subscription object YAML file to subscribe a namespace to the MQ Operator

a) Create a file called "mq-sub.yaml" with the following contents:

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: ibm-mq
  namespace: openshift-operators
spec:
  channel:
    name: ibm-mq
  source: ibm-operator-catalog
  sourceNamespace: openshift-marketplace
```

For AllNamespaces **InstallMode** usage, specify `openshift-operators` namespace. Otherwise, specify the relevant single namespace for SingleNamespace **InstallMode** usage.

b) Create the Subscription object

```
oc apply -f mq-sub.yaml
```

8. Check the status of the Operator

Once the installation of the Operator has succeeded, the pod status shows as *Running*.

For AllNamespaces **InstallMode** usage, specify `openshift-operators` as the namespace. Otherwise, specify the relevant single namespace for SingleNamespace **InstallMode** usage.

What to do next

[“Deploying IBM MQ certified containers” on page 15](#)

Linux

MQ Adv.

CD

Deploying IBM MQ certified containers

IBM MQ version 9.1.5 and above can be deployed to Red Hat OpenShift using the IBM MQ Operator. IBM MQ versions 9.1.5 and 9.1.4 can be deployed to Red Hat OpenShift using Helm. Earlier CD versions can be deployed to an IBM Cloud Private cluster or an IBM Cloud Kubernetes Service cluster, using Helm.

About this task

Procedure

- [“Deploying a queue manager using the Helm CLI” on page 24.](#)
- [“Deploying previous CD releases of IBM MQ into an IBM Cloud Private cluster” on page 26.](#)
- [“Adding previous CD releases of an IBM MQ image into an IBM Cloud Private cluster” on page 28.](#)
- [“Adding previous CD releases of an IBM MQ image into an IBM Cloud Kubernetes Service cluster” on page 28.](#)

Linux

MQ Adv.

CD

Preparing your OpenShift project for IBM MQ using the OpenShift CLI

Prepare your Red Hat OpenShift Container Platform cluster, so that it's ready to deploy a queue manager using the IBM MQ Operator. This task should be completed by a project administrator.

Before you begin

Note: If you plan to use IBM MQ in a project with other IBM Cloud Pak for Integration components already installed, you may not need to follow these instructions.

Log into your cluster using `cloudctl login` (for IBM Cloud Pak for Integration), or `oc login`.

About this task

The IBM MQ Advanced certified container images 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, follow these instructions to get an entitlement key and create a pull secret.

Procedure

1. Get the entitlement key that is assigned to your ID.
 - a) Log in to [MyIBM Container Software Library](#) with the IBM ID and password that are associated with the entitled software.
 - b) In the **Entitlement keys** section, select **Copy key** to copy the entitlement key to the clipboard.
2. Create a secret containing your entitlement key, in the project where you want to deploy your queue manager.

Run the following command, where `<entitlement-key>` is the key retrieved in step 1, and `<user-email>` is the IBM ID associated with the entitled software.

```
oc create secret docker-registry ibm-entitlement-key \
--docker-server=cp.icr.io \
--docker-username=cp \
--docker-password=<entitlement-key> \
--docker-email=<user-email>
```

What to do next

[“Deploying a queue manager using the OpenShift CLI” on page 18](#)

Linux MQ Adv. V 9.1.5 CD **Deploying a queue manager using the IBM Cloud Pak for Integration Platform Navigator**

Use the QueueManager custom resource to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster using the IBM Cloud Pak for Integration Platform Navigator. This task should be completed by a project administrator

Before you begin

In a browser, launch the IBM Cloud Pak for Integration Platform Navigator.

If this is the first time deploying a queue manager into this Red Hat OpenShift project, then follow the steps for [“Preparing your OpenShift project for IBM MQ using the OpenShift CLI” on page 15](#).

Procedure

1. Deploy a queue manager.

The following example deploys a "quick start" queue manager, which uses ephemeral (non-persistent) storage, and turns off MQ security. Messages will not be persisted across restarts of the queue manager. You can adjust the configuration to change many queue manager settings.

 - a) In the IBM Cloud Pak for Integration Platform Navigator, click **Runtime and instances**.
 - b) Click **Create instance**.
 - c) Select **Queue Manager**, and click **Next**.

The form to create an instance of a QueueManager is displayed.

Note: You can also click **Code** to view or change the QueueManager configuration YAML.
 - d) In the **Details** section, check or update the **Name** field, and specify the **Namespace** in which to create the queue manager instance.
 - e) If you accept the IBM Cloud Pak for Integration license agreement, change **License acceptance** to **On**.

You must accept the license to deploy a queue manager.

- f) In the **Queue Manager Config** section, check or update the **Name** of the underlying queue manager.

By default, the name of the queue manager used by IBM MQ client applications will be the same as the name of the QueueManager, but with any invalid characters (such as hyphens) removed. If you want to force the use of a particular name, you can edit this here.

- g) Click **Create**

The list of queue managers in the current project (namespace) is now displayed. The new QueueManager should have a status of Pending

2. Check the queue manager is running

The creation is complete when the QueueManager status is Running.

Related tasks

[“Connecting to a queue manager deployed in an OpenShift cluster” on page 29](#)

A set of configuration examples for connecting to a queue manager deployed in a Red Hat OpenShift cluster.

[“Connecting to the IBM MQ Console deployed in an OpenShift cluster” on page 31](#)

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

Linux MQ Adv. V 9.1.5 CD **Deploying a queue manager using the OpenShift web console**

Use the QueueManager custom resource to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster using the Red Hat OpenShift web console. This task should be completed by a project administrator

Before you begin

Log in to your OpenShift cluster's web console. You will need to select an existing Project (namespace) to use, or create a new one.

If this is the first time deploying a queue manager into this Red Hat OpenShift project, then follow the steps for [“Preparing your OpenShift project for IBM MQ using the OpenShift CLI” on page 15.](#)

Procedure

1. Deploy a queue manager.

The following example deploys a "quick start" queue manager, which uses ephemeral (non-persistent) storage, and turns off MQ security. Messages will not be persisted across restarts of the queue manager. You can adjust the configuration to change many queue manager settings.

- a) In the OpenShift web console, from the navigation pane click **Operators > Installed Operators**
- b) Click **IBM MQ**.
- c) Click on the **Queue Manager** tab.
- d) Click on the **Create QueueManager** button.

A YAML editor is displayed, containing example YAML for a QueueManager resource.

Note: You can also click **Edit Form** to view or change the QueueManager configuration.

- e) If you accept the license agreement, change **License acceptance** to **On**.

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 38.](#) You must accept the license to deploy a queue manager.

- f) Click **Create**

The list of queue managers in the current project (namespace) is now displayed. The new QueueManager should be in a Pending state.

2. Check the queue manager is running

The creation is complete when the QueueManager status is Running.

Related tasks

[“Connecting to a queue manager deployed in an OpenShift cluster” on page 29](#)

A set of configuration examples for connecting to a queue manager deployed in a Red Hat OpenShift cluster.

[“Connecting to the IBM MQ Console deployed in an OpenShift cluster” on page 31](#)

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

Linux

MQ Adv.

V 9.1.5

CD

Deploying a queue manager using the OpenShift CLI

Use the QueueManager custom resource to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster using the command line interface (CLI). 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 this is the first time deploying a queue manager into this Red Hat OpenShift project, then follow the steps for [“Preparing your OpenShift project for IBM MQ using the OpenShift CLI” on page 15](#).

Procedure

1. Deploy a queue manager.

The following example deploys a "quick start" queue manager, which uses ephemeral (non-persistent) storage, and turns off MQ security. Messages will not be persisted across restarts of the queue manager. You can adjust the contents of the YAML to change many queue manager settings.

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: 9.1.5.0-r2
  license:
    accept: false
    license: L-RJON-BN7PN3
    use: NonProduction
  web:
    enabled: true
  queueManager:
    name: "QUICKSTART"
    storage:
      queueManager:
        type: ephemeral
  template:
    pod:
      containers:
        - name: qmgr
          env:
            - name: MQSNOAUT
              value: "yes"
```

Important:If you accept the IBM Cloud Pak for Integration license agreement, change `accept: false` to `accept: true`. See [“Licensing reference for mq.ibm.com/v1beta1”](#) on page 38 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 with the Cloud Pak Identity and Access Manager.

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: 9.1.5.0-r2
  license:
    accept: false
    license: L-APIG-BM7GDH
    use: Development
  web:
    enabled: true
  queueManager:
    name: "QUICKSTART"
    storage:
      queueManager:
        type: ephemeral
  template:
    pod:
      containers:
        - name: qmgr
          env:
            - name: MQSNOAUT
              value: "yes"
```

Important:If you accept the MQ license agreement, change `accept: false` to `accept: true`. See [“Licensing reference for mq.ibm.com/v1beta1”](#) on page 38 for details on the license.

b) Create the QueueManager object

```
oc apply -f mq-quickstart.yaml
```

2. Check the queue manager is running

You can validate the deployment by running

```
oc describe queuemanager <QueueManagerResourceName>
```

, and then checking the status.

For example, run

```
oc describe queuemanager quickstart
```

, and check that the status.Phase field indicates Running

Related tasks

[“Connecting to a queue manager deployed in an OpenShift cluster”](#) on page 29

A set of configuration examples for connecting to a queue manager deployed in a Red Hat OpenShift cluster.

[“Connecting to the IBM MQ Console deployed in an OpenShift cluster”](#) on page 31

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

Linux

MQ Adv.

CD

V 9.1.4

Integrating with the IBM Cloud Pak for

Integration Operations Dashboard

The ability to trace transactions through IBM Cloud Pak for Integration is provided by the Operations Dashboard.

About this task

Enabling integration with the Operations Dashboard installs an MQ API exit to your queue manager. The API exit will send tracing data to the Operations Dashboard data store, about messages which are flowing through the queue manager.

Note that only messages which are sent using MQ client bindings are traced.

Procedure

1. Deploy a queue manager with tracing enabled

By default, the tracing feature is disabled.

If you are deploying using the IBM Cloud Pak for Integration Platform Navigator, then you can enable tracing while deploying, by setting **Enable Tracing** to **On**, and setting the **Tracing Namespace** to the namespace where the Operations Dashboard is installed. For more information on deploying a queue manager, see [“Deploying a queue manager using the IBM Cloud Pak for Integration Platform Navigator” on page 16](#)

If you are deploying using the [OpenShift CLI](#) or [OpenShift web console](#), then you can enable tracing with the following YAML snippet:

```
spec:
  tracing:
    enabled: true
    namespace: <Operations_Dashboard_Namespace
```

If you are deploying using Helm, then you can enable tracing by setting `odTracingConfig.enabled=true` and `odTracingConfig.odTracingNamespace=<Operations_Dashboard_Namespace`. If you want to enable Operations Dashboard integration on an existing queue manager, then you can apply this setting during when upgrading the Helm release.

Important: The queue manager will not start until MQ has been registered with the Operations Dashboard (see the next step).

Note that when this feature is enabled, it will run two sidecar containers ("Agent" and "Collector") in addition to the queue manager container. The images for these sidecar containers will be available in the same registry as the main MQ image, and will use the same pull policy and pull secret. There are additional settings available to configure CPU and memory limits.

2. If this is the first time a queue manager with Operations Dashboard integration has been deployed in this namespace, then you need to [Register](#) with the Operations Dashboard.

Registering creates a Secret object which the queue manager Pod needs to successfully start.

and INI files, using the 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 an OpenShift Secret for the IBM Entitled Registry in your Red Hat OpenShift project, then follow the steps for [“Preparing your OpenShift project for IBM MQ using the OpenShift CLI”](#) on page 15.

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:9.1.5.0-r2-amd64
      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:9.1.5.0-r2-amd64"
      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. 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 queue manager using the OpenShift CLI” on page 18](#), 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 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 queue manager using the OpenShift CLI” on page 18](#)

Use the QueueManager custom resource to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster using the command line interface (CLI). This task should be completed by a project administrator

Linux

MQ Adv.

CD

Deploying IBM MQ certified containers using

Helm

From IBM MQ 9.1.5.0, the recommended way to deploy a queue manager is to use the IBM MQ Operator. IBM MQ 9.1.5.0 and previous CD releases can be deployed using Helm, using the following instructions.

About this task

Procedure

- [“Preparing your OpenShift cluster for IBM MQ on OpenShift using Helm” on page 23.](#)
- [“Deploying a queue manager using the Helm CLI” on page 24.](#)

OpenShift using Helm

Prepare your Red Hat OpenShift Container Platform cluster, so that it's ready to deploy a queue manager using Helm. This task should be completed by a cluster administrator.

Before you begin

Note: If you are using IBM Cloud Pak for Integration, then the installer should have prepared an OpenShift project (namespace) for you to use with IBM MQ, so you may not need to follow these instructions.

Log into your cluster using `cloudctl login` (for IBM Cloud Pak for Integration), or `oc login`.

Procedure

1. Ensure that you've added the IBM Helm repository to your local copy of Helm.
For example, you can run the following command:

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

2. Ensure that you have a Helm server (called "Tiller") installed on your cluster.
Follow the instructions in [Getting started with Helm on OpenShift](#) to install Helm on your cluster.
3. Ensure that Service Accounts in your OpenShift project (namespace) are authorized to use the right Security Context Constraints (SCCs).

V 9.1.5

IBM MQ works under the default SCC of "restricted", so this step can normally be skipped.

Applying changes to SCCs needs to be done by an OpenShift cluster administrator. Each Helm chart version has different requirements for SCCs, which are documented in the individual README file for that Helm chart:

```
helm inspect readme ibm-entitled-charts/ibm-mqadvanced-server-prod
```

There are instructions in each README for setting up authorization for SCCs. Note that the IBM MQ Helm charts create a Service Account for their own use, which means that SCC permissions need to be applied at the "group" level (for all Service Accounts in the namespace).

4. Ensure that you have a valid "image pull secret" to pull images from your chosen container registry
The IBM MQ Advanced certified container images 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, follow these instructions to get an entitlement key and create a pull secret.
 - a) Get the entitlement key that is assigned to your ID.
 - i) Log in to [MyIBM Container Software Library](#) with the IBM ID and password that are associated with the entitled software.
 - ii) In the *Entitlement keys* section, select **Copy key** to copy the entitlement key to the clipboard.
 - b) Create the secret in the namespace in which you want to deploy your queue manager.
 - Run the following command, where `<entitlement-key>` is the key retrieved in step 1, and `<user-email>` is the IBM ID associated with the entitled software.

```
oc create secret docker-registry ibm-entitlement-key \
--docker-server=cp.icr.io \
--docker-username=cp \
--docker-password=<entitlement-key> \
--docker-email=<user-email>
```

What to do next

[“Deploying a queue manager using the Helm CLI” on page 24](#)

Linux > MQ Adv. > CD > V 9.1.4 **Deploying a queue manager using the Helm CLI**

Use Helm to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster. This task should be completed by a project administrator.

Before you begin

You need to install [Helm V2](#) and the [Red Hat OpenShift Container Platform command-line interface](#). If you're not using IBM Cloud Pak for Integration, then follow the steps for [“Preparing your OpenShift cluster for IBM MQ on OpenShift using Helm” on page 23](#).

Log into your cluster using **cloudctl login** (for IBM Cloud Pak for Integration), or **oc login**.

Procedure

1. Ensure that you've added the IBM Helm repository to your local copy of Helm.
For example, you can run the following command:

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

2. Review the configuration options for your queue manager

The deployment step includes both installation and configuration steps. Some settings for your queue manager must be set at deployment time, and changing them requires a re-deployment.

You can view the Helm chart README for details of all the available deployment options, by running one of the following commands:

- For IBM MQ Advanced certified container in IBM Cloud Pak for Integration:

```
helm inspect readme ibm-entitled-charts/ibm-mqadvanced-server-integration-prod
```

- For IBM MQ Advanced certified container:

```
helm inspect readme ibm-entitled-charts/ibm-mqadvanced-server-prod
```

You will typically need at least the following parameters:

- a. Release name. For example: `my-release`
 - b. Remote Helm repository. For example: `ibm-entitled-charts`
 - c. Helm chart: for example `ibm-mqadvanced-server-prod` or `ibm-mqadvanced-server-integration-prod`
 - d. Image pull secret name. For example: `entitled-registry`. Note this is not needed if you are deploying into the pre-defined project for MQ in IBM Cloud Pak for Integration
3. Deploy a queue manager.

Note that by default, the Helm chart assumes that you have a default [Storage Class](#) set in your Red Hat OpenShift Container Platform cluster.

For example, to install a basic queue manager in IBM Cloud Pak for Integration, run the following command:

```
helm install \
--tls \
--name my-release \
ibm-entitled-charts/ibm-mqadvanced-server-integration-prod \
--set license=accept \
```

```
--set tls.hostname=my.cluster \  
--set tls.generate=true
```

You can enter any hostname in the `tls.hostname` field (this is a required field but will not be used as in this example we are generating a new self-signed certificate)

To install a basic queue manager independently of IBM Cloud Pak for Integration, you could run the following command:

```
helm install \  
--name my-release \  
ibm-entitled-charts/ibm-mqadvanced-server-prod \  
--set license=accept \  
--set image.pullSecret=ibm-entitlement-key
```

Related tasks

[“Connecting to a queue manager deployed in an OpenShift cluster” on page 29](#)

A set of configuration examples for connecting to a queue manager deployed in a Red Hat OpenShift cluster.

[“Connecting to the IBM MQ Console deployed in an OpenShift cluster” on page 31](#)

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

Linux MQ Adv. V 9.1.5 CD *Deploying a queue manager with IBM Cloud File Storage, using the Helm CLI*

Example scenario to use Helm to deploy a queue manager onto a Red Hat OpenShift on IBM Cloud cluster, using IBM Cloud File Storage. This task should be completed by a project administrator

Before you begin

You need to install [Helm V2](#) and the Red Hat OpenShift Container Platform command-line interface. If you're not using IBM Cloud Pak for Integration, then follow the steps for [“Preparing your OpenShift cluster for IBM MQ on OpenShift using Helm” on page 23](#).

Log into your cluster using **cloudctl login** (for IBM Cloud Pak for Integration), or **oc login**.

Procedure

1. Ensure that you've added the IBM Helm repository to your local copy of Helm.
For example, you can run the following command:

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

2. Deploy a queue manager.

When using IBM Cloud File Storage, you will typically see the best results using the `ibmc-file-gold-gid` storage class. This storage class enables storage that can be written to by users in the correct file system group.

For example, to install a basic queue manager in IBM Cloud Pak for Integration, run the following command:

```
helm install \  
--tls \  
--name my-release \  
ibm-entitled-charts/ibm-mqadvanced-server-integration-prod \  
--set license=accept \  
--set tls.hostname=my.cluster \  
--set tls.generate=true \  
--set dataPVC.storageClassName=ibmc-file-gold-gid \  
--set security.context.supplementalGroups={99}
```

You can enter any hostname in the `tls.hostname` field (this is a required field but is not used here, because in this example we are generating a new self-signed certificate).

To install a basic queue manager independently of IBM Cloud Pak for Integration, you could run the following command:

```
helm install \
--name my-release \
ibm-entitled-charts/ibm-mqadvanced-server-prod \
--set license=accept \
--set image.pullSecret=ibm-entitlement-key \
--set dataPVC.storageClassName=ibmc-file-gold-gid \
--set security.context.supplementalGroups={99}
```

Related tasks

[“Connecting to a queue manager deployed in an OpenShift cluster” on page 29](#)

A set of configuration examples for connecting to a queue manager deployed in a Red Hat OpenShift cluster.

[“Connecting to the IBM MQ Console deployed in an OpenShift cluster” on page 31](#)

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

Linux

MQ Adv.

CD

Deploying previous CD releases of IBM MQ into an IBM Cloud Private cluster

For CD versions of IBM MQ earlier than 9.1.4, use the IBM Cloud Private management console to deploy a queue manager into IBM Cloud Private.

Before you begin



Attention: V 9.1.4 This deployment is not supported in IBM MQ 9.1.4 or later versions.

This task assumes that you have already [added an IBM MQ image into an IBM Cloud Private cluster](#).

The Helm chart README .md file is available from the IBM Cloud Private catalog entry, which is displayed after you complete this substep, or from the command line by adding your IBM Cloud Private's **local-charts** repository as a remote Helm repository and running the following command:

```
helm inspect readme remote_repo_name/ibm-mqadvanced-server-prod
```

You must have a [PodSecurityPolicy](#), or a [SecurityContextConstraint](#) (for IBM Cloud Private on Red Hat OpenShift) that supports the necessary security context. Details, including examples, can be found from the Helm chart README .md file.

Details on how to configure your Helm release can also be found in the Helm chart README .md file.

Note:

- If you are deploying to an IBM Cloud Private environment that does not support the required security settings by default, enable your deployment by following the instructions in [Deploying Helm charts that require elevated privileges in a non-default namespace](#) in the IBM Cloud Private product documentation.
- If you are using SELinux, you must meet the IBM MQ requirements described in [IBM MQ support for SELinux on Red Hat Enterprise Linux](#).

About this task

IBM Cloud Private offers a platform for managing on-premises, containerized applications. After you have added an IBM MQ image into an IBM Cloud Private cluster, you can use either the IBM Cloud Private management console or the command line to deploy a queue manager.

Procedure

- Using the IBM Cloud Private Management Console

- a) Open the IBM Cloud Private management console in a web browser, and click **Catalog**.
See [Accessing your IBM Cloud Private cluster by using the management console](#) in the IBM Cloud Private product documentation.
- b) Select the `ibm-mqadvanced-server-prod` chart from the list.
- c) Select **Configure**, then complete the following configuration steps:
 - a. Enter a release name.
 - b. Read and accept the license agreements.
 - c. Under the **dataPVC** section, set **storageclass** to your desired storage class. Leave blank to select the default storage class.
 - d. Under the **image** section, set the repository to the full image path. For example:


```
mycluster.icp:8500/namespace_name/ibm-mqadvanced-server-prod
```
 - e. Under the **image** section, set the tag to the image tag. For example:


```
9.1.3.0-r1
```
 - f. If you need a Kubernetes pull secret to access the image registry, add it as the **pullSecret**.
 - g. Under the **queueManager** section, set the name of the queue manager.
- d) Click **Install** to deploy your queue manager as a *Helm release*.
- Using the command line
 - a) Configure **cloudctl** to access your IBM Cloud Private cluster.
See [Installing the IBM Cloud Private CLI](#) in the IBM Cloud Private product documentation.
 - b) Ensure that you have [added your IBM Cloud Private's local-charts repository](#) as a remote Helm repository.
 - c) Install the chart.
Run the following command, specifying these parameters:
 - a. Release name (for example `my-release`)
 - b. Name of the remote helm repository that contains the `ibm-mqadvanced-server-prod` chart (for example `my-repo`)
 - c. Image repository (for example `mycluster.icp:8500/namespace_name/ibm-mqadvanced-server-prod`)
 - d. Image tag (for example `9.1.3.0-r1`)

```
helm install --name my-release --repo my-repo ibm-mqadvanced-server-prod --set license=accept --set image.repository=mycluster.icp:8500/namespace_name/ibm-mqadvanced-server-prod --set image.tag=9.1.3.0-r1 --tls
```

Related tasks

[“Deploying a queue manager using the Helm CLI” on page 24](#)

Use Helm to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster. This task should be completed by a project administrator.

[“Adding previous CD releases of an IBM MQ image into an IBM Cloud Private cluster” on page 28](#)

For CD versions of IBM MQ earlier than 9.1.4, prepare your IBM Cloud Private cluster to deploy a production-ready image for IBM MQ.

[“Adding previous CD releases of an IBM MQ image into an IBM Cloud Kubernetes Service cluster” on page 28](#)

For CD versions of IBM MQ earlier than 9.1.4, import a production-ready image for IBM MQ into IBM Cloud Kubernetes Service.

Linux

MQ Adv.

CD

Adding previous CD releases of an IBM MQ image into an IBM Cloud Private cluster

For CD versions of IBM MQ earlier than 9.1.4, prepare your IBM Cloud Private cluster to deploy a production-ready image for IBM MQ.

About this task



Attention: **V 9.1.4** This import is not supported in IBM MQ 9.1.4 or later versions.

You can download an IBM MQ image from Passport Advantage and import it into an IBM Cloud Private container.

Procedure

1. Download the latest IBM MQ image from the [Passport Advantage and Passport Advantage Express web site](#).

For details of available downloads, go to [Downloading IBM MQ 9.1](#) then click the tab for the release that you want to download. The name and number of the part to download are listed in a table.

2. Import the downloaded archive file into IBM Cloud Private.

See [Adding IBM software to the IBM Cloud Private Catalog](#) in the IBM Cloud Private product documentation.

What to do next

You are now ready to [Deploy a queue manager into IBM Cloud Private](#).

Related tasks

[“Deploying a queue manager using the Helm CLI” on page 24](#)

Use Helm to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster. This task should be completed by a project administrator.

[“Deploying previous CD releases of IBM MQ into an IBM Cloud Private cluster” on page 26](#)

For CD versions of IBM MQ earlier than 9.1.4, use the IBM Cloud Private management console to deploy a queue manager into IBM Cloud Private.

[“Adding previous CD releases of an IBM MQ image into an IBM Cloud Kubernetes Service cluster” on page 28](#)

For CD versions of IBM MQ earlier than 9.1.4, import a production-ready image for IBM MQ into IBM Cloud Kubernetes Service.

Linux

MQ Adv.

CD

Adding previous CD releases of an IBM MQ image into an IBM Cloud Kubernetes Service cluster

For CD versions of IBM MQ earlier than 9.1.4, import a production-ready image for IBM MQ into IBM Cloud Kubernetes Service.

About this task



Attention: **V 9.1.4** This import is not supported in IBM MQ 9.1.4 or later versions.

You can download an IBM MQ image from Passport Advantage and import it into an IBM Cloud Kubernetes Service cluster.

Procedure

1. Download the latest IBM MQ image from the [Passport Advantage and Passport Advantage Express web site](#).

For details of available downloads, go to [Downloading IBM MQ 9.1](#) then click the tab for the release that you want to download. The name and number of the part to download are listed in a table.

2. Import the downloaded archive file into IBM Cloud Kubernetes Service.

See [Running IBM Cloud Private images in public Kubernetes containers](#).

Related tasks

[“Deploying a queue manager using the Helm CLI” on page 24](#)

Use Helm to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster. This task should be completed by a project administrator.

[“Deploying previous CD releases of IBM MQ into an IBM Cloud Private cluster” on page 26](#)

For CD versions of IBM MQ earlier than 9.1.4, use the IBM Cloud Private management console to deploy a queue manager into IBM Cloud Private.

[“Adding previous CD releases of an IBM MQ image into an IBM Cloud Private cluster” on page 28](#)

For CD versions of IBM MQ earlier than 9.1.4, prepare your IBM Cloud Private cluster to deploy a production-ready image for IBM MQ.

Linux

MQ Adv.

CD

V 9.1.4

Connecting to a queue manager deployed in an OpenShift cluster

A set of configuration examples for connecting to a queue manager deployed in a Red Hat OpenShift cluster.

About this task

You need an [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 [Server Name Indication \(SNI\)](#) is only available in the TLS protocol. The Red Hat OpenShift Container Platform Router uses SNI for routing requests to the IBM MQ queue manager.

The required configuration of the OpenShift Route depends on the SNI behavior of your client application.

To set the SNI header as TLS 1.2 or higher, a CipherSpec or CipherSuite must be used for your TLS communication.

The SNI is set to the MQ channel if the following conditions are met:

- The IBM MQ C Client is V8 or later.
- The Java/JMS Client is V9.1.1 or later, and the Java installation supports the `javax.net.ssl.SNIHostName` class.
- The .NET Client is in unmanaged mode.

The SNI is set to the Hostname if a hostname is supplied as the connection name, and the following conditions are met:

- The .NET Client is in managed mode.
- The AMQP or XR client is used.
- The Java/JMS Clients are used with **AllowOutboundSNI** set to NO.

The SNI is not set and is blank under the following conditions:

- The IBM MQ C Client is V7.5 or earlier.
- IBM MQ C Client is used with **AllowOutboundSNI** set to NO.

- The Java/JMS Clients are used with a Java installation that does not support the `javax.net.ssl.SNIHostName` class.

Example

Host name based OpenShift Routes : For client applications that set the SNI to the host name

The following Helm charts automatically create a host name based OpenShift Route for connecting an application to an IBM MQ queue manager. Client applications that set the SNI to the host name can use this OpenShift Route.

- `ibm-mqadvanced-server-dev`
- `ibm-mqadvanced-server-prod`
- `ibm-mqadvanced-server-integration-prod` in the IBM Cloud Pak for Integration.

If you are not using these charts and need to create your own host name based OpenShift Route, you can apply the following `yaml` in your cluster:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: <provide a unique name for the Route>
  namespace: <namespace of your MQ deployment>
spec:
  to:
    kind: Service
    name: <name of the Kubernetes Service for your MQ deployment (for example "<Helm Release>-ibm-
mq")>
  port:
    targetPort: 1414
  tls:
    termination: passthrough
```

MQ channel based OpenShift Routes : For client applications that set the SNI to the MQ channel

Client applications that set the SNI to the MQ channel require a new 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 cluster, to allow routing to the correct queue manager.

To determine the required host name for each of your new OpenShift Routes, you need to map each channel name to an SNI address as documented here: <https://www.ibm.com/support/pages/ibm-websphere-mq-how-does-mq-provide-multiple-certificates-certlabl-capability>

You must then create a new OpenShift Route (for each channel) by applying the following `yaml` in your cluster:

```
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: <provide a unique name for the Route>
  namespace: <the namespace of your MQ deployment>
spec:
  host: <SNI address mapping for the channel>
  to:
    kind: Service
    name: <the name of the Kubernetes Service for your MQ deployment (for example "<Helm Release>-
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 "<Helm Release>-ibm-mq-qn")>
-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 OpenShift Container Platform (OCP) Router - normally 443.

Related tasks

[“Deploying a queue manager using the Helm CLI” on page 24](#)

Use Helm to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster. This task should be completed by a project administrator.

[“Connecting to the IBM MQ Console deployed in an OpenShift cluster” on page 31](#)

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

Linux > MQ Adv. > CD > V 9.1.4 Connecting to the IBM MQ Console deployed in an OpenShift cluster

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

About this task

If you are using the IBM MQ Operator, the IBM MQ Console URL can be found on the QueueManager details page in the OpenShift web console or in the IBM Cloud Pak for Integration Platform Navigator. Alternatively, it can be found from the OpenShift CLI by running the following command:

```
oc get queuemanager <QueueManager Name> -n <namespace of your MQ deployment> --output jsonpath='{.status.adminUiUrl}'
```

Example

The following Helm charts automatically create an OpenShift Route for accessing the IBM MQ Console

- ibm-mqadvanced-server-dev
- ibm-mqadvanced-server-integration-prod in the IBM Cloud Pak for Integration.

You can get the hostname of the OpenShift Route by running the following command:

```
oc get route <Route Name (for example "<Helm Release>-ibm-mq-web")> -n <namespace of your MQ deployment> --output jsonpath='{.spec.host}'
```

You can access the IBM MQ Console using the following URL:

```
https://<Route Hostname>/ibmmq/console
```

Related tasks

[“Deploying a queue manager using the Helm CLI” on page 24](#)

Use Helm to deploy a queue manager onto a Red Hat OpenShift Container Platform cluster. This task should be completed by a project administrator.

[“Connecting to a queue manager deployed in an OpenShift cluster” on page 29](#)

A set of configuration examples for connecting to a queue manager deployed in a Red Hat OpenShift cluster.

Linux > MQ Adv. > CD > Backing up and restoring queue manager configuration using the 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 **cloudctl login** (for IBM Cloud Pak for Integration), or **oc login**.

Procedure

- Back up queue manager configuration.

You can use the **dmpmqcfig** command to dump the configuration of an IBM MQ queue manager.

- a) Get the name of the pod for your queue manager.

For example, if you are using the Operator, 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
```

For example, if you are using Helm, you could run the following command, where *release_name* is the name of your Helm release.

```
oc get pods --selector release=release_name
```

- b) Run the **dmpmqcfig** command on the pod, directing the output into a file on your local machine.

dmpmqcfig outputs the queue manager's MQSC configuration.

```
oc exec -it pod_name -- dmpmqcfig > 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, if you are using the Operator, 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
```

For example, if you are using Helm, you could run the following command, where *release_name* is the name of your Helm release.

```
oc get pods --selector release=release_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
```

Building your own IBM MQ container

Develop a self-built container, formerly referred to as the "Docker container image". 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 one of the pre-packaged containers provided by IBM. See ["IBM MQ in containers" on page 5](#)

About this task

When you package IBM MQ as a container image, changes to your application can be deployed to test and staging systems quickly and easily. This can be a major benefit to continuous delivery in your enterprise.

Procedure

- For information on how to build an IBM MQ container image by using Docker, see the following subtopics:
 -  [“Support for building your own IBM MQ container images and charts” on page 8](#)
 - [“Planning your own IBM MQ queue manager image using a container” on page 33](#)
 - [“Building a sample IBM MQ queue manager image using Docker” on page 33](#)
 - [“Running local binding applications in separate containers” on page 36](#)

Related concepts

[“IBM MQ in containers” on page 5](#)

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.

Planning your own IBM MQ queue manager image using a container

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.

Building a sample IBM MQ queue manager image using Docker

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 34](#)
 - [“Building a sample configured IBM MQ queue manager image” on page 34](#)

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](#).

Results

You now have a base container image with IBM MQ installed.

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, create your own container image layer, using the generic image as a parent.

Before you begin

For an IBM MQ 9.1 image, you cannot configure secure access using the Red Hat OpenShift Container Platform "restricted" Security Context Constraint (SCC). The "restricted" SCC uses random user IDs, and prevents privilege escalation by changing to a different user. The IBM MQ 9.1 RPM-based installer relies on an `mqm` user and group, and also uses `setuid` bits on executable programs.

This restriction is removed in IBM MQ 9.2.

Procedure

1. Create a new directory, and add a file called `config.mqsc`, with the following contents:

```
DEFINE CHANNEL(PASSWORD.SVRCONN) CHLTYPE(SVRCONN)
SET CHLAUTH(PASSWORD.SVRCONN) TYPE(BLOCKUSER) USERLIST('nobody') +
DESCR('Allow privileged users on this channel')
SET CHLAUTH('*') TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(NOACCESS) DESCR('BackStop rule')
SET CHLAUTH(PASSWORD.SVRCONN) TYPE(ADDRESSMAP) ADDRESS('*') USERSRC(CHANNEL) CHCKCLNT(REQUIRED)
ALTER AUTHINFO(SYSTEM.DEFAULT.AUTHINFO.IDPWOS) AUTHTYPE(IDPWOS) ADOPTCTX(YES)
REFRESH SECURITY TYPE(CONNAUTH)
```

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
RUN useradd johndoe -G mqm && \
    echo johndoe:passwd | chpasswd
COPY config.mqsc /etc/mqm/
```

where:

- johndoe is the user ID that you want to add
- passwd is the original password

3. Build your custom container image using the following command:

```
sudo 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.

The **RUN** command adds a user named johndoe with password passwd and the **COPY** command adds the config.mqsc file into a specific location known by the parent image.

Note: On Red Hat Enterprise Linux (RHEL), you use the command **docker** (RHEL V7) or **podman** (RHEL V7 or RHEL V8). In the case of **podman**, you don't need **sudo** at the beginning of the command.

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:

```
sudo 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.
- You can set the maximum number of open files by using the command **sysctl fs.file-max=524288**.

V 9.1.0

Running local binding applications in separate containers

With process namespace sharing between containers in Docker, 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

This functionality is supported in IBM MQ 9.0.3 and later queue managers.

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.

You can try this functionality out, by completing the following steps on a Linux system that already has Docker installed.

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 `dspmqr` command on the second container, to see the status for both queue managers, by issuing the following command:

```
docker exec secondaryContainer dspmqr
```

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 `dspmqr`, `amqspqr`, `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 37 for more information.

V 9.1.0 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 36 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 `dspmqr` commands, you see the following:

```
docker exec container dspmqr
QMNAME(QM1)                STATUS(Status not available)
```

- For `runmqsc` commands, or other commands that try to connect to the queue manager, you are likely to receive an AMQ8146 error message:

```
docker exec -it container runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2026.
Starting MQSC for queue manager QM1.
AMQ8146: IBM MQ queue manager not available
```

- If you share all the required namespaces but you do not mount a shared volume to the `/var/mqm` directory, and you have a valid IBM MQ data path, then your commands also receive AMQ8146 error messages.

However, **dspmq** is not able to see your queue manager at all, and instead returns a blank response:

```
docker exec container dspmq
```

- If you share all the required namespaces but you do not mount a shared volume to the `/var/mqm` directory, and you do not have a valid IBM MQ data path (or no IBM MQ data path), then you see various errors as the data path is a key component of an IBM MQ installation. Without the data path, IBM MQ cannot operate.

If you run any of the following commands, and you see responses similar to those shown in these examples, you should verify that you have mounted the directory or created an IBM MQ data directory:

```
docker exec container dspmq
'No such file or directory' from /var/mqm/mqs.ini
AMQ6090: IBM MQ was unable to display an error message FFFFFFFF.
AMQffff

docker exec container dspmqver
AMQ7047: An unexpected error was encountered by a command. Reason code is 0.

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

docker exec container crtmqm QM1
AMQ8101: IBM MQ error (893) has occurred.

docker exec container strmqm QM1
AMQ6239: Permission denied attempting to access filesystem location '/var/mqm'.
AMQ7002: An error occurred manipulating a file.

docker exec container endmqm QM1
AMQ8101: IBM MQ error (893) has occurred.

docker exec container dltmqm QM1
AMQ7002: An error occurred manipulating a file.

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

Linux

MQ Adv.

V 9.1.5

CD

API reference for the IBM MQ

Operator

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

Linux

MQ Adv.

V 9.1.5

CD

API reference for mq.ibm.com/v1beta1

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

Linux

MQ Adv.

V 9.1.5

CD

Licensing reference for mq.ibm.com/v1beta1

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.use</code>	License information
L-RJON-BN7PN3	Production or NonProduction	IBM Cloud Pak for Integration 2020.2
L-RJON-BPHL2Y		IBM Cloud Pak for Integration Limited Edition 2020.2

Value of <code>spec.license.license</code>	Value of <code>spec.license.use</code>	License information
L-APIG-BJAKBF	Production or Development	IBM MQ Advanced 9.1.5
L-APIG-BM7GDH	Development	IBM MQ Advanced for Developers 9.1.5

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

Linux > MQ Adv. > V 9.1.5 > CD **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.

Field	Description
<code>apiVersion</code> string	APIVersion defines the versioned schema of this representation of an object. Servers should convert recognized schemas to the latest internal value, and may reject unrecognized values. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#resources .
<code>kind</code> string	Kind is a string value representing the REST resource this object represents. Servers may infer this from the endpoint the client submits requests to. Cannot be updated. In CamelCase. More info: https://git.k8s.io/community/contributors/devel/sig-architecture/api-conventions.md#types-kinds .
<code>metadata</code>	
<code>spec</code> “QueueManagerSpec” on page 43	The desired state of the QueueManager.
<code>status</code> “QueueManagerStatus” on page 44	The observed state of the QueueManager.

Availability

Availability settings for the Queue Manager, such as whether or not to use an active-standby pair.

Appears in:

- [“QueueManagerConfig” on page 41](#)

Field	Description
<code>type</code> string	The type of availability to use. Use "SingleInstance" for a single Pod, which will be restarted automatically (in some cases) by Kubernetes. Use "MultiInstance" for a pair of Pods, one of which is the "active" Queue Manager, and the other of which is a standby. See High availability for IBM MQ in containers in the latest version of IBM MQ.

License

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

Appears in:

- [“QueueManagerSpec” on page 43](#)

Field	Description
use string	Setting that controls how the software will to be used, where the license supports multiple uses. See https://ibm.biz/BdqvCF for valid values.
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 https://ibm.biz/BdqvCF for valid values.
metric string	Setting that specifies which license metric to use. For example, "ProcessorValueUnit", "VirtualProcessorCore" or "ManagedVirtualServer".

Limits

QueueManagerResourceList defines CPU & memory settings.

Appears in:

- “Resources” on page 46

Field	Description
cpu	
memory	

LocalObjectReference

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

Appears in:

- “QueueManagerSpec” on page 43

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

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:

- “QueueManagerSpec” on page 43

Field	Description
keys “PKISource” on page 40 array	Private keys to add to the Queue Manager's key repository.
trust “PKISource” on page 40 array	Certificates to add to the Queue Manager's key repository.

PKISource

PKISource defines a source of Public Key Infrastructure information, such as keys or certificates.

Appears in:

- [“PKI” on page 40](#)

Field	Description
name string	Name is used as the label for the key or certificate. Must be a lowercase alphanumeric string.
secret “Secret” on page 46	Supply a key using a Kubernetes Secret.

QueueManagerConfig

QueueManagerConfig defines the settings for the Queue Manager container and underlying Queue Manager.

Appears in:

- [“QueueManagerSpec” on page 43](#)

Field	Description
logFormat string	Which log format to use for this container. Use "JSON" for JSON-formatted logs from the container. Use "Basic" for text-formatted messages.
metrics “QueueManagerMetrics” on page 42	Settings for Prometheus-style metrics.
readinessProbe “QueueManagerReadinessProbe” on page 42	Settings that control the readiness probe.
resources “Resources” on page 46	Settings that control resource requirements.
storage “QueueManagerStorage” on page 45	Storage settings to control the Queue Manager's use of Persistent Volumes and Storage Classes.
availability “Availability” on page 39	Availability settings for the Queue Manager, such as whether or not to use an active-standby pair.
imagePullPolicy string	Setting that controls when the kubelet attempts to pull the specified image.
livenessProbe “QueueManagerLivenessProbe” on page 41	Settings that control the liveness probe.
debug boolean	Whether or not to log debug messages from the container-specific code, to the container log.
image string	The container image that will be used.
name string	Name of the underlying MQ Queue Manager, if different from metadata.name. Use this field if you want a Queue Manager name which does not conform to the Kubernetes rules for names (for example, a name which includes capital letters).

QueueManagerLivenessProbe

Settings that control the liveness probe.

Appears in:

- [“QueueManagerConfig” on page 41](#)

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

QueueManagerMetrics

Settings for Prometheus-style metrics.

Appears in:

- [“QueueManagerConfig” on page 41](#)

Field	Description
enabled boolean	Whether or not to enable a Prometheus-compatible metrics endpoint.

QueueManagerOptionalVolume

PersistentVolume details for MQ recovery logs. Required when using multi-instance Queue Manager.

Appears in:

- [“QueueManagerStorage” on page 45](#)

Field	Description
class string	Storage class to use for this volume. Only valid if "type" is "persistent-claim".
enabled boolean	Whether or not this volume should be enabled as a separate volume, or be placed on the default "queueManager" volume.
size string	Size of the PersistentVolume to pass to Kubernetes. Only valid if "type" is "persistent-claim".
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".
type string	Type of volume to use. Choose ephemeral to create a non-persistent "emptyDir" volume, or persistent-claim to use a persistent volume.

QueueManagerReadinessProbe

Settings that control the readiness probe.

Appears in:

- [“QueueManagerConfig” on page 41](#)

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

QueueManagerSpec

The desired state of the QueueManager.

Appears in:

- [“QueueManager” on page 39](#)

Field	Description
license “License” on page 39	Settings that control your acceptance of the license, and which license metrics to use.
pki “PKI” on page 40	Public Key Infrastructure settings, for defining keys and certificates for use with Transport Layer Security (TLS) or MQ Advanced Message Security (AMS).
queueManager “QueueManagerConfig” on page 41	QueueManagerConfig defines the settings for the Queue Manager container and underlying Queue Manager.
securityContext “SecurityContext” on page 47	Security settings to add to the Queue Manager Pod's securityContext.
tracing “TracingConfig” on page 48	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.
affinity	Standard Kubernetes affinity rules. For more information, see https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.17/#affinity-v1-core .
imagePullSecrets “LocalObjectReference” on page 40 array	An optional list of references to secrets in the same namespace to use for pulling any of the images used by this QueueManager. If specified, these secrets will be passed to individual puller implementations for them to use. For example, in the case of docker, only DockerConfig type secrets are honored. For more information, see https://kubernetes.io/docs/concepts/containers/images#specifying-imagepullsecrets-on-a-pod .

Field	Description
template “Template” on page 47	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.
terminationGracePeriod 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.
web “WebServerConfig” on page 49	Settings for the MQ web server.

QueueManagerStatus

The observed state of the QueueManager.

Appears in:

- [“QueueManager” on page 39](#)

Field	Description
endpoints “QueueManagerStatusEndpoint” on page 44 array	Information on the endpoints that this Queue Manager is exposing, such as API or UI endpoints.
name string	The name of the Queue Manager.
versions “QueueManagerStatusVersion” on page 45	Version of MQ being used, and other versions available from the IBM Entitled Registry.
adminUiUrl string	URL for the Admin UI.
conditions “QueueManagerStatusCondition” on page 44 array	Conditions represent the latest available observations of the Queue Manager's state.

QueueManagerStatusCondition

QueueManagerStatusCondition defines the conditions of the Queue Manager.

Appears in:

- [“QueueManagerStatus” on page 44](#)

Field	Description
message string	Human-readable message indicating details about last transition.
type string	Type of condition.
lastTransitionTime string	Last time the condition transitioned from one status to another.

QueueManagerStatusEndpoint

QueueManagerStatusEndpoint defines the endpoints for the QueueManager.

Appears in:

- [“QueueManagerStatus” on page 44](#)

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.

QueueManagerStatusVersion

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

Appears in:

- [“QueueManagerStatus” on page 44](#)

Field	Description
available “QueueManagerStatusVersion Available” on page 45	Other versions of MQ available from the IBM Entitled Registry.
reconciled string	The specific version of IBM MQ being used. If a custom image is specified, then this may not match the version of MQ actually being used.

QueueManagerStatusVersionAvailable

Other versions of MQ available from the IBM Entitled Registry.

Appears in:

- [“QueueManagerStatusVersion” on page 45](#)

Field	Description
channels array	Channels which are available for automatically updating the MQ version.
versions “Versions” on page 49 array	Specific versions of MQ which are available.

QueueManagerStorage

Storage settings to control the Queue Manager's use of Persistent Volumes and Storage Classes.

Appears in:

- [“QueueManagerConfig” on page 41](#)

Field	Description
persistedData “QueueManagerOptionalVolume” on page 42	PersistentVolume details for MQ persisted data, including configuration, queues and messages. Required when using multi-instance Queue Manager.
queueManager “QueueManagerVolume” on page 46	Default PersistentVolume for any data normally under /var/mqm. Will contain all persisted data and recovery logs, if no other volumes are specified.

Field	Description
recoveryLogs “QueueManagerOptionalVolume” on page 42	PersistentVolume details for MQ recovery logs. Required when using multi-instance Queue Manager.

QueueManagerVolume

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:

- [“QueueManagerStorage” on page 45](#)

Field	Description
class string	Storage class to use for this volume. Only valid if "type" is "persistent-claim".
size string	Size of the PersistentVolume to pass to Kubernetes. Only valid if "type" is "persistent-claim".
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".
type string	Type of volume to use. Choose <code>ephemeral</code> to create a non-persistent "emptyDir" volume, or <code>persistent-claim</code> to use a persistent volume.

Requests

QueueManagerResourceList defines CPU & memory settings.

Appears in:

- [“Resources” on page 46](#)

Field	Description
memory	
cpu	

Resources

Settings that control resource requirements.

Appears in:

- [“QueueManagerConfig” on page 41](#)

Field	Description
limits “Limits” on page 40	QueueManagerResourceList defines CPU & memory settings.
requests “Requests” on page 46	QueueManagerResourceList defines CPU & memory settings.

Secret

Supply a key using a Kubernetes Secret.

Appears in:

- [“PKISource” on page 40](#)

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.

SecurityContext

Security settings to add to the Queue Manager Pod's securityContext.

Appears in:

- [“QueueManagerSpec” on page 43](#)

Field	Description
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.
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. For more information, see https://docs.openshift.com/container-platform/latest/authentication/managing-security-context-constraints.html .

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:

- [“QueueManagerSpec” on page 43](#)

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

TracingAgent

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

Appears in:

- [“TracingConfig” on page 48](#)

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.
livenessProbe “TracingProbe” on page 48	Settings that control the liveness probe.
readinessProbe “TracingProbe” on page 48	Settings that control the readiness probe.

TracingCollector

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

Appears in:

- [“TracingConfig” on page 48](#)

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.
livenessProbe “TracingProbe” on page 48	Settings that control the liveness probe.
readinessProbe “TracingProbe” on page 48	Settings that control the readiness probe.

TracingConfig

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

Appears in:

- [“QueueManagerSpec” on page 43](#)

Field	Description
agent “TracingAgent” on page 47	In Cloud Pak for Integration only, you can configure settings for the optional Tracing Agent.
collector “TracingCollector” on page 48	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.
namespace string	Namespace where the Cloud Pak for Integration Operations Dashboard is installed.

TracingProbe

Settings that control the readiness probe.

Appears in:

- [“TracingCollector” on page 48](#)

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

Versions

QueueManagerStatusVersion defines a version of MQ.

Appears in:

- [“QueueManagerStatusVersionAvailable” on page 45](#)

Field	Description
name string	Version "name" for this version of QueueManager. These are valid values for the <code>spec.version</code> field.

WebServerConfig

Settings for the MQ web server.

Appears in:

- [“QueueManagerSpec” on page 43](#)

Field	Description
enabled boolean	Whether or not to enable the web server.

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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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 WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Important: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks

IBM, the IBM logo, ibm.com[®], are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" www.ibm.com/legal/copytrade.shtml. Other product and service names might be trademarks of IBM or other companies.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

This product includes software developed by the Eclipse Project (<http://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: