Red Hat AMQ 2020.Q4

Deploying AMQ Interconnect on OpenShift

For Use with AMQ Interconnect 1.9

Last Updated: 2020-11-23

Red Hat AMQ 2020.Q4 Deploying AMQ Interconnect on OpenShift

For Use with AMQ Interconnect 1.9

Legal Notice

Abstract

Learn how to install and deploy AMQ Interconnect on OpenShift Container Platform.

Table of Contents

CHAPTER 1. GETTING STARTED WITH AMQ INTERCONNECT ON OPENSHIFT CONTAINER PLATFORM . 3
1.1. WHAT OPERATORS ARE 3
1.2. PROVIDED CUSTOM RESOURCES 3

CHAPTER 2. PREPARING TO DEPLOY AMQ INTERCONNECT ON OPENSHIFT CONTAINER PLATFORM . 5
2.1. CREATING SECRETS FOR SSL/TLS AUTHENTICATION 5
2.2. ADDING THE RED HAT INTEGRATION - AMQ CERTIFICATE MANAGER OPERATOR 7

CHAPTER 3. ADDING THE RED HAT INTEGRATION - AMQ INTERCONNECT 8

CHAPTER 4. CREATING A ROUTER NETWORK 9
4.1. CREATING AN INTERIOR ROUTER DEPLOYMENT 9
4.2. CREATING AN EDGE ROUTER DEPLOYMENT 12
4.3. CREATING AN INTER-CLUSTER ROUTER NETWORK 13
4.3.1. Creating an inter-cluster router network using a Certificate Authority 13
4.3.2. Creating an inter-cluster router network using AMQ Certificate Manager 15

CHAPTER 5. CONNECTING CLIENTS TO THE ROUTER NETWORK 18
5.1. EXPOSING PORTS FOR CLIENTS OUTSIDE OF OPENSHIFT CONTAINER PLATFORM 18
5.2. AUTHENTICATION FOR CLIENT CONNECTIONS 19
5.3. CONFIGURING CLIENTS TO CONNECT TO THE ROUTER NETWORK 19

CHAPTER 6. CONNECTING TO EXTERNAL SERVICES 21

CHAPTER 7. CONFIGURING THE ADDRESS SPACE FOR MESSAGE ROUTING 23
7.1. ROUTING MESSAGES BETWEEN CLIENTS 23
7.2. ROUTING MESSAGES THROUGH BROKERS 23

CHAPTER 8. USING PROMETHEUS AND GRAFANA TO MONITOR THE ROUTER NETWORK 26
8.1. SETTING UP PROMETHEUS AND GRAFANA 26
8.2. VIEWING AMQ INTERCONNECT DASHBOARDS IN GRAFANA 26
8.3. ROUTER METRICS 28

CHAPTER 9. USING THE AMQ INTERCONNECT WEB CONSOLE TO MONITOR THE ROUTER NETWORK 30

CHAPTER 1. GETTING STARTED WITH AMQ INTERCONNECT ON OPENSHIFT CONTAINER PLATFORM

AMQ Interconnect is a lightweight AMQP 1.0 message router for building large, highly resilient messaging networks for hybrid cloud and IoT/edge deployments. AMQ Interconnect automatically learns the addresses of messaging endpoints (such as clients, servers, and message brokers) and flexibly routes messages between them.

This document describes how to deploy AMQ Interconnect on OpenShift Container Platform by using the AMQ Interconnect Operator and the **Interconnect** Custom Resource Definition (CRD) that it provides. The CRD defines an AMQ Interconnect deployment, and the Operator creates and manages the deployment in OpenShift Container Platform.

1.1. WHAT OPERATORS ARE

Operators are a method of packaging, deploying, and managing a Kubernetes application. They take human operational knowledge and encode it into software that is more easily shared with consumers to automate common or complex tasks.

In OpenShift Container Platform 4.0, the *Operator Lifecycle Manager (OLM)* helps users install, update, and generally manage the life cycle of all Operators and their associated services running across their clusters. It is part of the Operator Framework, an open source toolkit designed to manage Kubernetes native applications (Operators) in an effective, automated, and scalable way.

The OLM runs by default in OpenShift Container Platform 4.0, which aids cluster administrators in installing, upgrading, and granting access to Operators running on their cluster. The OpenShift Container Platform web console provides management screens for cluster administrators to install Operators, as well as grant specific projects access to use the catalog of Operators available on the cluster.

OperatorHub is the graphical interface that OpenShift Container Platform cluster administrators use to discover, install, and upgrade Operators. With one click, these Operators can be pulled from OperatorHub, installed on the cluster, and managed by the OLM, ready for engineering teams to self-service manage the software in development, test, and production environments.

Additional resources

- For more information about Operators, see the OpenShift documentation.

1.2. PROVIDED CUSTOM RESOURCES

The AMQ Interconnect Operator provides the **Interconnect** Custom Resource Definition (CRD), which allows you to interact with an AMQ Interconnect deployment running on OpenShift Container Platform just like other OpenShift Container Platform API objects.

The **Interconnect** CRD represents a deployment of AMQ Interconnect routers. The CRD provides elements for defining many different aspects of a router deployment in OpenShift Container Platform such as:

- Number of AMQ Interconnect routers
- Deployment topology
- Connectivity

- Address semantics

CHAPTER 2. PREPARING TO DEPLOY AMQ INTERCONNECT ON OPENSHIFT CONTAINER PLATFORM

Before deploying AMQ Interconnect on OpenShift Container Platform, perform one of the following procedures:

- Section 2.1, “Creating secrets for SSL/TLS authentication”
- Section 2.2, “Adding the Red Hat Integration - AMQ Certificate Manager Operator”

If you are evaluating AMQ Interconnect, you can skip these steps however Red Hat recommends always securing AMQ Interconnect communication.

2.1. CREATING SECRETS FOR SSL/TLS AUTHENTICATION

NOTE

If you installed the Red Hat Integration - AMQ Certificate Manager Operator you can skip this procedure, instructions for securing your network with AMQ Certificate Manager are included in the associated procedures. OpenShift uses objects called **Secrets** to hold sensitive information such as SSL/TLS certificates. If you want to secure inter-router traffic, client traffic, or both, then you must create the SSL/TLS certificates and private keys and provide them to OpenShift as secrets.

For OpenShift Container Platform 4.6 and greater, this procedure is the only supported method of securing AMQ Interconnect communication.

Procedure

1. If you do not have an existing certificate authority (CA) certificate for inter-router connections, create one.
   These commands create a self-signed CA certificate for inter-router connections:

   ```
   # Create a new directory for the inter-router certificates.
   $ mkdir internal-certs

   # Create a private key for the CA.
   $ openssl genrsa -out internal-certs/ca-key.pem 2048

   # Create a certificate signing request for the CA.
   $ openssl req -new -batch -key internal-certs/ca-key.pem -out internal-certs/ca-csr.pem

   # Self sign the CA certificate.
   $ openssl x509 -req -in internal-certs/ca-csr.pem -signkey internal-certs/ca-key.pem -out
   internal-certs/ca.crt
   ```

2. Create a certificate for the router signed by the CA.
   These commands create a private key and a certificate, and sign the certificate using the CA created in the previous step:

   ```
   # Create a private key.
   $ openssl genrsa -out internal-certs/tls.key 2048
   ```

```
# Create a certificate signing request for the router.
$ openssl req -new -batch -subj "/CN=amq-interconnect.<project_name>.svc.cluster.local" -
key internal-certs/tls.key -out internal-certs/server-csr.pem

# Sign the certificate using the CA.
$ openssl x509 -req -in internal-certs/server-csr.pem -CA internal-certs/ca.crt -CAkey
internal-certs/ca-key.pem -out internal-certs/tls.crt -CAcreateserial
```

where **<project_name>** is the name of the current OpenShift project.

3. Create a secret containing the private key, router certificate, and CA certificate.
   This command creates the secret using the key and certificates that were created in the previous steps:

```
$ oc create secret generic inter-router-certs-secret --from-file=tls.crt=internal-certs/tls.crt  --
from-file=tls.key=internal-certs/tls.key  --from-file=ca.crt=internal-certs/ca.crt
```

4. If you want to use SSL/TLS to authenticate client connections (as opposed to authenticating clients using SASL), create a CA certificate for client connections.
   These commands create a self-signed CA certificate for client connections:

```
# Create a new directory for the client certificates.
$ mkdir client-certs

# Create a private key for the CA.
$ openssl genrsa -out client-certs/ca-key.pem 2048

# Create a certificate signing request for the CA.
$ openssl req -new -batch -key client-certs/ca-key.pem -out client-certs/ca-csr.pem

# Self sign the certificate.
$ openssl x509 -req -in client-certs/ca-csr.pem -signkey client-certs/ca-key.pem -out client-
certs/ca.crt
```

5. Create a certificate for client connections signed by the CA.
   These commands create a private key and a certificate, and then sign the certificate using the CA created in the previous step:

```
# Create a private key.
$ openssl genrsa -out client-certs/tls.key 2048

# Create a certificate signing request for the client connections
$ openssl req -new -batch -subj "/CN=<client_name>" -key client-certs/tls.key -out client-
certs/client-csr.pem

# Sign the certificate using the CA.
$ openssl x509 -req -in client-certs/client-csr.pem -CA client-certs/ca.crt -CAkey client-
certs/ca-key.pem -out client-certs/tls.crt -CAcreateserial
```

where **<client_name>** is unique for each router client.

6. Create a secret containing the CA certificate used to sign client certificates using the certificate that was created in the previous steps:

```
$ oc create secret generic client-ca-secret --from-file=ca.crt=client-certs/ca.crt --from-file=tls.crt=client-certs/ca.crt --from-file=tls.key=client-certs/ca-key.pem
```

2.2. ADDING THE RED HAT INTEGRATION - AMQ CERTIFICATE MANAGER OPERATOR

The Red Hat Integration - AMQ Certificate Manager Operator (cert-manager) is an optional Kubernetes add-on that issues and manages TLS certificates. The Red Hat Integration - AMQ Interconnect uses it to automatically create the TLS certificates needed to secure the router network.

You use *OperatorHub* to add the Operator to your OpenShift Container Platform cluster.

NOTE

Installing an Operator requires administrator-level privileges for your OpenShift cluster.

If you are using OpenShift Container Platform 4.6 you must create and manage TLS certificates as described in Section 2.1, “Creating secrets for SSL/TLS authentication”.

When installed, the operator is available to all users and projects in the cluster.

Prerequisites

- Access to an OpenShift Container Platform 4.5 cluster using a **cluster-admin** account.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators → OperatorHub**.
2. Choose **Red Hat Integration - AMQ Certificate Manager Operator** from the list of available Operators, and then click **Install**.
3. On the **Operator Installation** page, select **All namespaces on the cluster (default)** and then click **Install**.
   The **Installed Operators** page appears displaying the status of the Operator installation.
4. Verify that the Red Hat Integration - AMQ Certificate Manager Operator Operator is displayed and wait until the **Status** changes to **Succeeded**.
5. If the installation is not successful, troubleshoot the error:
   a. Click **Red Hat Integration - AMQ Certificate Manager Operator** on the **Installed Operators** page.
   b. Select the **Subscription** tab and view any failures or errors.

Additional resources

- For more information about **cert-manager**, see the cert-manager documentation.

CHAPTER 3. ADDING THE RED HAT INTEGRATION - AMQ INTERCONNECT

The Red Hat Integration - AMQ Interconnect creates and manages AMQ Interconnect router networks in OpenShift Container Platform. This Operator must be installed separately for each project that uses it.

You use *OperatorHub* to add the Operator to your OpenShift Container Platform cluster.

NOTE

Installing an Operator requires administrator-level privileges for your OpenShift cluster.

Prerequisites

- Access to an OpenShift Container Platform 4.5 or 4.6 cluster using a **cluster-admin** account.
- Red Hat Integration - AMQ Certificate Manager Operator is installed in the OpenShift Container Platform cluster if required.

Procedure

1. In the OpenShift Container Platform web console, navigate to **Operators** → **OperatorHub**.
2. Choose **Red Hat Integration - AMQ Interconnect** from the list of available Operators, and then click **Install**.
3. On the **Operator Installation** page, select the namespace into which you want to install the Operator, and then click **Install**.
   The **Installed Operators** page appears displaying the status of the Operator installation.
4. Verify that the AMQ Interconnect Operator is displayed and wait until the **Status** changes to **Succeeded**.
5. If the installation is not successful, troubleshoot the error:
   a. Click **Red Hat Integration - AMQ Interconnect** on the **Installed Operators** page.
   b. Select the **Subscription** tab and view any failures or errors.

CHAPTER 4. CREATING A ROUTER NETWORK

To create a network of AMQ Interconnect routers, you define a deployment in an **Interconnect** Custom Resource, and then apply it. The AMQ Interconnect Operator creates the deployment by scheduling the necessary Pods and creating any needed Resources.

The procedures in this section demonstrate the following router network topologies:

- Interior router mesh
- Interior router mesh with edge routers for scalability
- Inter-cluster router network that connects two OpenShift clusters

Prerequisites

- The AMQ Interconnect Operator is installed in your OpenShift Container Platform project.

4.1. CREATING AN INTERIOR ROUTER DEPLOYMENT

Interior routers establish connections with each other and automatically compute the lowest cost paths across the network.

Procedure

This procedure creates an interior router network of three routers. The routers automatically connect to each other in a mesh topology, and their connections are secured with mutual SSL/TLS authentication.

1. Create an **Interconnect** Custom Resource YAML file that describes the interior router deployment.

   Sample router-mesh.yaml file

   ```
   apiVersion: interconnectedcloud.github.io/v1alpha1
   kind: Interconnect
   metadata:
     name: router-mesh
   spec:
     deploymentPlan:
       role: interior  (1)
       size: 3  (2)
       placement:  Any  (3)
   ```

   (1) The operating mode of the routers in the deployment. The Operator will automatically connect interior routers in a mesh topology.

   (2) The number of routers to create.

   (3) Each router runs in a separate Pod. The placement defines where in the cluster the Operator should schedule and place the Pods. You can choose the following placement options:

   Any
   : The Pods can run on any node in the OpenShift Container Platform cluster.

   Every

The Operator places a router Pod on each node in the cluster. If you choose this option, the **Size** property is not needed - the number of routers corresponds to the number of nodes in the cluster.

Anti-Affinity
: The Operator ensures that multiple router Pods do not run on the same node in the cluster. If the size is greater than the number of nodes in the cluster, the extra Pods that cannot be scheduled will remain in a **Pending** state.

2. Create the router deployment described in the YAML file.

```
$ oc apply -f router-mesh.yaml
```

   The Operator creates a deployment of interior routers in a mesh topology that uses default address semantics. It also creates a Service through which the routers can be accessed, and a Route through which you can access the web console.

3. Verify that the router mesh was created and the Pods are running.
   Each router runs in a separate Pod. They connect to each other automatically using the Service that the Operator created.

```
$ oc get pods
NAME                                      READY   STATUS    RESTARTS   AGE
interconnect-operator-587f94784b-4bzdx   1/1     Running   0          52m
router-mesh-6b48f89bd-588r5              1/1     Running   0          40m
router-mesh-6b48f89bd-bdjc4              1/1     Running   0          40m
router-mesh-6b48f89bd-h6d5r              1/1     Running   0          40m
```

4. Review the router deployment.

```
$ oc get interconnect/router-mesh -o yaml
apiVersion: interconnectedcloud.github.io/v1alpha1
kind: Interconnect
...
spec:
  addresses: 1
  - distribution: closest
    prefix: closest
  - distribution: multicast
    prefix: multicast
  - distribution: closest
    prefix: unicast
  - distribution: closest
    prefix: exclusive
  - distribution: multicast
    prefix: broadcast
  deploymentPlan: 2
    livenessPort: 8888
    placement: Any
    resources: {}
    role: interior
    size: 3
  edgeListeners: 3
  - port: 45672
  interRouterListeners: 4
```

```
- authenticatePeer: true
  expose: true
  port: 55671
  saslMechanisms: EXTERNAL
  sslProfile: inter-router
listeners:  5
- port: 5672
- authenticatePeer: true
  expose: true
  http: true
  port: 8080
- port: 5671
  sslProfile: default
sslProfiles:  6
- credentials: router-mesh-default-tls
  name: default
- caCert: router-mesh-inter-router-tls
  credentials: router-mesh-inter-router-tls
  mutualAuth: true
  name: inter-router
users: router-mesh-users  7
```

1 The default address configuration. All messages sent to an address that does not match any of these prefixes are distributed in a balanced anycast pattern .

2 A router mesh of three interior routers was deployed.

3 Each interior router listens on port **45672** for connections from edge routers.

4 The interior routers connect to each other on port **55671**. These inter-router connections are secured with SSL/TLS mutual authentication. The **inter-router** SSL Profile contains the details of the certificates that the Operator generated.

5 Each interior router listens for connections from external clients on the following ports:

- **5672** - Unsecure connections from messaging applications.
- **5671** - Secure connections from messaging applications.
- **8080** - AMQ Interconnect web console access. Default user name/password security is applied.

6 Using the Red Hat Integration - AMQ Certificate Manager Operator, the Red Hat Integration - AMQ Interconnect automatically creates two SSL profiles:

- **inter-router** - The Operator secures the inter-router network with mutual TLS authentication by creating a Certificate Authority (CA) and generating certificates signed by the CA for each interior router.
- **default** - The Operator creates TLS certificates for messaging applications to connect to the interior routers on port **5671**.

7 The AMQ Interconnect web console is secured with user name/password authentication. The Operator automatically generates the credentials and stores them in the **router-mesh-users** Secret.

4.2. CREATING AN EDGE ROUTER DEPLOYMENT

You can efficiently scale your router network by adding an edge router deployment. Edge routers act as connection concentrators for messaging applications. Each edge router maintains a single uplink connection to an interior router, and messaging applications connect to the edge routers to send and receive messages.

Prerequisites

- The interior router mesh is deployed. For more information, see Section 4.1, "Creating an interior router deployment".

Procedure

This procedure creates an edge router on each node of the OpenShift Container Platform cluster and connects them to the previously created interior router mesh.

1. Create an **Interconnect** Custom Resource YAML file that describes the edge router deployment.

   Sample edge-routers.yaml file

   ```
   apiVersion: interconnectedcloud.github.io/v1alpha1
   kind: Interconnect
   metadata:
     name: edge-routers
   spec:
     deploymentPlan:
       role: edge
       placement: Every (1)
     edgeConnectors: (2)
       - host: router-mesh (3)
         port: 45672 (4)
   ```

   (1) An edge router Pod will be deployed on each node in the OpenShift Container Platform cluster. This placement helps to balance messaging application traffic across the cluster. The Operator will create a DaemonSet to ensure that the number of Pods scheduled always corresponds to the number of nodes in the cluster.

   (2) Edge connectors define the connections from the edge routers to the interior routers.

   (3) The name of the Service that was created for the interior routers.

   (4) The port on which the interior routers listen for edge connections. The default is **45672**.

2. Create the edge routers described in the YAML file:

   ```
   $ oc apply -f edge-routers.yaml
   ```

   The Operator deploys an edge router on each node of the OpenShift Container Platform cluster, and connects them to the interior routers.

3. Verify that the edge routers were created and the Pods are running.

Each router runs in a separate Pod. Each edge router connects to any of the previously created interior routers.

```
$ oc get pods
NAME                                  READY   STATUS    RESTARTS   AGE
edge-routers-2jz5j                      1/1     Running   0          33s
edge-routers-fhlxv                      1/1     Running   0          33s
edge-routers-gg2qb                      1/1     Running   0          33s
edge-routers-hj72t                      1/1     Running   0          33s
interconnect-operator-587f94784b-4bzdx   1/1     Running   0          54m
router-mesh-6b48f89bd-588r5             1/1     Running   0          42m
router-mesh-6b48f89bd-bdjc4             1/1     Running   0          42m
router-mesh-6b48f89bd-h6d5r             1/1     Running   0          42m
```

4.3. CREATING AN INTER-CLUSTER ROUTER NETWORK

Depending on whether you are using AMQ Certificate Manager, there are different procedures for creating an inter-cluster router network.

- Section 4.3.1, “Creating an inter-cluster router network using a Certificate Authority”
- Section 4.3.2, “Creating an inter-cluster router network using AMQ Certificate Manager”

4.3.1. Creating an inter-cluster router network using a Certificate Authority

You can create a router network from routers running in different OpenShift Container Platform clusters. This enables you to connect applications running in separate clusters.

Prerequisites

- You have already created secrets defining an existing certificate for each router.

Procedure

This procedure creates router deployments in two different OpenShift Container Platform clusters (**cluster1** and **cluster2**) and connects them together to form an inter-cluster router network. The connection between the router deployments is secured with SSL/TLS mutual authentication.

1. In the first OpenShift Container Platform cluster (**cluster1**), create an **Interconnect** Custom Resource YAML file that describes the interior router deployment.
   This example creates a single interior router with a default configuration.

   Sample cluster1-router-mesh.yaml file

   ```
   apiVersion: interconnectedcloud.github.io/v1alpha1
   kind: Interconnect
   metadata:
     name: cluster1-router-mesh
   spec:
     interRouterListeners:
       - authenticatePeer: true 1
         host: 0.0.0.0 2
         port: 55672 3
         saslMechanisms: EXTERNAL 4
   ```

```
      sslProfile: inter-router-profile 5
      expose: true 6
  sslProfiles:
    - caCert: inter-router-certs-secret 7
      credentials: inter-router-certs-secret 8
      name: inter-router-profile 9
```

1 **authenticatePeer** must be set to **true** to authenticate using TLS certificates

2 listener host

3 listener port

4 SASL mechanism to authenticate, use EXTERNAL for TLS certificates

5 ssl-profile name to use for authenticating clients

6 exposes a route so that the port is accessible from outside the cluster

7 name of cluster secret or your CA containing a **ca.crt** name (in case you're using the same secret used in credentials, otherwise it must have a tls.crt)

8 name of cluster secret with the CA certificate containing **tls.crt** and **tls.key** files

9 ssl-profile name to use for the interRouterListener

2. Create the router deployment described in the YAML file.

```
$ oc apply -f cluster1-router-mesh.yaml
```

The Red Hat Integration - AMQ Interconnect creates an interior router with a default configuration and a listener to authenticate other routers.

3. Log in to the second OpenShift Container Platform cluster (**cluster2**), and switch to the project where you want to create the second router deployment.

4. In **cluster2**, create an **Interconnect** Custom Resource YAML file to describe the router deployment.

```
apiVersion: interconnectedcloud.github.io/v1alpha1
kind: Interconnect
metadata:
  name: cluster2-router-mesh
spec:
  sslProfiles:
  - name: inter-router-profile 1
    credentials: inter-router-certs-secret
    caCert: inter-router-certs-secret
  interRouterConnectors:
  - host: cluster1-router-mesh-port-55672-myproject.cluster1.openshift.com 2
    port: 443
    verifyHostname: false
    sslProfile: inter-router-profile
    name: cluster1
```

1. This SSL Profile defines the certificate needed to connect to the router deployment in **cluster1**.

2. The URL of the Route for the inter-router listener on **cluster1**.

5. Create the router deployment described in the YAML file.

```
$ oc apply -f cluster2-router-mesh.yaml
```

6. Verify that the routers are connected.
   This example displays the connections from the router in **cluster2** to the router in **cluster1**.

```
$ oc exec cluster2-fb6bc5797-crvb6 -it -- qdstat -c
Connections
  id    host                                                        container                              role          dir
security                                    authentication  tenant

=================================================================================
=================================================================================
=============================================
  1     cluster1-router-mesh-port-55672-myproject.cluster1.openshift.com:443  cluster1-router-
mesh-54cffd9967-9h4vq  inter-router  out  TLSv1/SSLv3(DHE-RSA-AES256-GCM-SHA384)
x.509
```

4.3.2. Creating an inter-cluster router network using AMQ Certificate Manager

You can create a router network from routers running in different OpenShift Container Platform clusters. This enables you to connect applications running in separate clusters.

Procedure

This procedure creates router deployments in two different OpenShift Container Platform clusters (**cluster1** and **cluster2**) and connects them together to form an inter-cluster router network. The connection between the router deployments is secured with SSL/TLS mutual authentication.

1. In the first OpenShift Container Platform cluster (**cluster1**), create an **Interconnect** Custom Resource YAML file that describes the interior router deployment.
   This example creates a single interior router with a default configuration.

   Sample cluster1-router-mesh.yaml file

```
apiVersion: interconnectedcloud.github.io/v1alpha1
kind: Interconnect
metadata:
  name: cluster1-router-mesh
spec: {}
```

2. Create the router deployment described in the YAML file.

```
$ oc apply -f cluster1-router-mesh.yaml
```

The Red Hat Integration - AMQ Interconnect creates an interior router with a default configuration. It uses the Red Hat Integration - AMQ Certificate Manager Operator to create a Certificate Authority (CA) and generate a certificate signed by the CA.

3. Generate an additional certificate for the router deployment in the second OpenShift Container Platform cluster (**cluster2**).
   The router deployment in **cluster2** requires a certificate issued by the CA of **cluster1**.

   a. Create a **Certificate** Custom Resource YAML file to request a certificate.

      Sample certificate-request.yaml file

      ```
      apiVersion: certmanager.k8s.io/v1alpha1
      kind: Certificate
      metadata:
        name: cluster2-inter-router-tls
      spec:
        commonName: cluster1-router-mesh-myproject.cluster2.openshift.com
        issuerRef:
          name: cluster1-router-mesh-inter-router-ca  1
        secretName: cluster2-inter-router-tls-secret
      ---
      ```

      1 The name of the Issuer that created the inter-router CA for **cluster1**. By default, the name of the Issuer is ***<application-name>*-inter-router-ca**.

   b. Create the certificate described in the YAML file.

      ```
      $ oc apply -f certificate-request.yaml
      ```

   c. Extract the certificate that you generated.

      ```
      $ mkdir /tmp/cluster2-inter-router-tls
      $ oc extract secret/cluster2-inter-router-tls-secret --to=/tmp/cluster2-inter-router-tls
      ```

4. Log in to the second OpenShift Container Platform cluster (**cluster2**), and switch to the project where you want to create the second router deployment.

5. In **cluster2**, create a Secret containing the certificate that you generated.

   ```
   $ oc create secret generic cluster2-inter-router-tls-secret --from-file=/tmp/cluster2-inter-router-tls
   ```

6. In **cluster2**, create an **Interconnect** Custom Resource YAML file to describe the router deployment.

   ```
   apiVersion: interconnectedcloud.github.io/v1alpha1
   kind: Interconnect
   metadata:
     name: cluster2-router-mesh
   spec:
     sslProfiles:
     - name: inter-cluster-tls  1
   ```

```
    credentials: cluster2-inter-router-tls-secret
    caCert: cluster2-inter-router-tls-secret
  interRouterConnectors:
  - host: cluster1-router-mesh-port-55671-myproject.cluster1.openshift.com (2)
    port: 443
    verifyHostname: false
    sslProfile: inter-cluster-tls
```

(1) This SSL Profile defines the certificate needed to connect to the router deployment in **cluster1**.

(2) The URL of the Route for the inter-router listener on **cluster1**.

7. Create the router deployment described in the YAML file.

```
$ oc apply -f cluster2-router-mesh.yaml
```

8. Verify that the routers are connected.
   This example displays the connections from the router in **cluster2** to the router in **cluster1**.

```
$ oc exec cluster2-fb6bc5797-crvb6 -it -- qdstat -c
Connections
  id   host                                                        container                              role          dir
security                                  authentication  tenant

=============================================================================================================
=============================================================================================================
=================================================
  1    cluster1-router-mesh-port-55671-myproject.cluster1.openshift.com:443  cluster1-router-
mesh-54cffd9967-9h4vq  inter-router  out  TLSv1/SSLv3(DHE-RSA-AES256-GCM-SHA384)
x.509
```

CHAPTER 5. CONNECTING CLIENTS TO THE ROUTER NETWORK

After creating a router network, you can connect clients (messaging applications) to it so that they can begin sending and receiving messages.

By default, the Red Hat Integration - AMQ Interconnect creates a Service for the router deployment and configures the following ports for client access:

- **5672** for plain AMQP traffic without authentication
- **5671** for AMQP traffic secured with TLS authentication

To connect clients to the router network, you can do the following:

- If any clients are outside of the OpenShift cluster, expose the ports so that they can connect to the router network.
- Configure your clients to connect to the router network.

5.1. EXPOSING PORTS FOR CLIENTS OUTSIDE OF OPENSHIFT CONTAINER PLATFORM

You expose ports to enable clients outside of the OpenShift Container Platform cluster to connect to the router network.

Procedure

1. Start editing the **Interconnect** Custom Resource YAML file that describes the router deployment for which you want to expose ports.

```
$ oc edit -f router-mesh.yaml
```

2. In the **spec.listeners** section, expose each port that you want clients outside of the cluster to be able to access.
   In this example, port **5671** is exposed. This enables clients outside of the cluster to authenticate with and connect to the router network.

Sample router-mesh.yaml file

```
apiVersion: interconnectedcloud.github.io/v1alpha1
kind: Interconnect
metadata:
  name: router-mesh
spec:
  ...
  listeners:
    - port: 5672
    - authenticatePeer: true
      expose: true
      http: true
      port: 8080
    - port: 5671
```

```
      sslProfile: default
      expose: true
  ...
```

The Red Hat Integration - AMQ Interconnect creates a Route, which clients from outside the cluster can use to connect to the router network.

5.2. AUTHENTICATION FOR CLIENT CONNECTIONS

When you create a router deployment, the Red Hat Integration - AMQ Interconnect uses the Red Hat Integration - AMQ Certificate Manager Operator to create default SSL/TLS certificates for client authentication, and configures port **5671** for SSL encryption.

5.3. CONFIGURING CLIENTS TO CONNECT TO THE ROUTER NETWORK

You can connect messaging clients running in the same OpenShift cluster as the router network, a different cluster, or outside of OpenShift altogether so that they can exchange messages.

Prerequisites

- If the client is outside of the OpenShift Container Platform cluster, a connecting port must be exposed. For more information, see Section 5.1, "Exposing ports for clients outside of OpenShift Container Platform".

Procedure

- To connect a client to the router network, use the following connection URL format:

  ```
  <scheme>://[<username>@]<host>[:<port>]
  ```

  <scheme>

  Use one of the following:

  - **amqp** - unencrypted TCP from within the same OpenShift cluster
  - **amqps** - for secure connections using SSL/TLS authentication
  - **amqpws** - AMQP over WebSockets for unencrypted connections from outside the OpenShift cluster

  <username>

  If you deployed the router mesh with user name/password authentication, provide the client's user name.

  <host>

  If the client is in the same OpenShift cluster as the router network, use the OpenShift Service host name. Otherwise, use the host name of the Route.

  <port>

  If you are connecting to a Route, you must specify the port. To connect on an unsecured connection, use port **80**. Otherwise, to connect on a secured connection, use port **443**.

NOTE

To connect on an unsecured connection (port **80**), the client must use AMQP over WebSockets (**amqpws**).

The following table shows some example connection URLs.

URL	Description
amqp://admin@router-mesh:5672	The client and router network are both in the same OpenShift cluster, so the Service host name is used for the connection URL. In this case, user name/password authentication is implemented, which requires the user name (**admin**) to be provided.
amqps://router-mesh-myproject.mycluster.com:443	The client is outside of OpenShift, so the Route host name is used for the connection URL. In this case, SSL/TLS authentication is implemented, which requires the **amqps** scheme and port **443**.
amqpws://router-mesh-myproject.mycluster.com:80	The client is outside of OpenShift, so the Route host name is used for the connection URL. In this case, no authentication is implemented, which means the client must use the **amqpws** scheme and port **80**.

CHAPTER 6. CONNECTING TO EXTERNAL SERVICES

You can connect a router to an external service such as a message broker. The services may be running in the same OpenShift cluster as the router network, or running outside of OpenShift.

Prerequisites

- You must have access to a message broker.

Procedure

This procedure describes how to connect a router to a broker and configure a link route to connect messaging clients to it.

1. Start editing the **Interconnect** Custom Resource YAML file that describes the router deployment that you want to connect to a broker.

   ```
   $ oc edit -f router-mesh.yaml
   ```

2. In the **spec** section, configure the connection and link route.

   Sample router-mesh.yaml file

   ```
   apiVersion: interconnectedcloud.github.io/v1alpha1
   kind: Interconnect
   metadata:
     name: router-mesh
   spec:
     ...
     connectors: (1)
     - name: my-broker
       host: broker
       port: 5672
       routeContainer: true
     linkRoutes: (2)
     - prefix: q1
       direction: in
       connection: my-broker
     - prefix: q1
       direction: out
       connection: my-broker
   ```

   (1) The connection to be used to connect this router to the message broker. The Operator will configure this connection from every router defined in this router deployment to the broker. If you only want a single connection between the router network and the broker, then configure a **listener** instead of a connector and have the broker establish the connection.

   (2) The link route configuration. It defines the incoming and outgoing links and connection to be used to connect messaging applications to the message broker.

3. Verify that the router has established the link route to the message broker.

   ```
   $ oc exec router-mesh-fb6bc5797-crvb6 -it -- qdstat --linkroutes
   ```

```
Link Routes
  address  dir  distrib       status
  ====================================
  q1       in   linkBalanced  active
  q1       out  linkBalanced  active
```

Additional resources

- For more information about link routes, see Creating link routes.

CHAPTER 7. CONFIGURING THE ADDRESS SPACE FOR MESSAGE ROUTING

AMQ Interconnect provides flexible application-layer addressing and delivery semantics. By configuring addresses, you can route messages in anycast (closest or balanced) or multicast patterns.

7.1. ROUTING MESSAGES BETWEEN CLIENTS

By default, AMQ Interconnect distributes messages in a balanced anycast pattern (each message is delivered to a single consumer, and AMQ Interconnect attempts to balance the traffic load across the network). This means you only need to change the address configuration if you want to apply non-default semantics to an address or range of addresses.

Procedure

This procedure configures an address to use multicast distribution. The router network will distribute a copy of each message sent to this address to every consumer that is subscribed to the address.

1. Start editing the **Interconnect** Custom Resource YAML file that describes the router deployment.

   ```
   $ oc edit -f router-mesh.yaml
   ```

2. In the **spec** section, define the semantics to be applied to addresses.

   Sample router-mesh.yaml file

   ```
   apiVersion: interconnectedcloud.github.io/v1alpha1
   kind: Interconnect
   metadata:
     name: router-mesh
   spec:
     ...
     addresses:
     - pattern: */orders (1)
       distribution: multicast
   ```

   (1) Messages sent to any address that ends with “**orders**” will be distributed in a multicast pattern.

   The Operator applies the changes to the router network and restarts each Pod.

3. If you have additional router deployment Custom Resources that define routers in the router network, repeat this procedure for each CR.
   Each router in the router network must have the same address configuration.

Additional resources

- For more information about address semantics that you can configure, see Configuring message routing.

7.2. ROUTING MESSAGES THROUGH BROKERS

If you need to store and forward messages, you can route them through a queue on a message broker. In this scenario, message producers send messages to a router, and the router sends the messages to a broker queue. When a consumer connects to the router to receive the messages, the router retrieves them from the broker queue.

You can route messages to brokers running in the same OpenShift cluster as the router network, or to brokers that are running outside of the cluster.

Prerequisites

- You must have access to a message broker.

Procedure

1. Start editing the Interconnect Custom Resource YAML file that describes the router deployment.

   ```
   $ oc edit -f router-mesh.yaml
   ```

2. In the **spec** section, add a connector to connect to the broker, a waypoint address to point to the broker queue, and autolinks to create the links to the queue.

   Sample router-mesh.yaml file

   ```
   apiVersion: interconnectedcloud.github.io/v1alpha1
   kind: Interconnect
   metadata:
     name: router-mesh
   spec:
     ...
     addresses:
     - prefix: my-queue  1
       waypoint: true
     autoLinks:  2
     - address: my-queue
       direction: in
       connection: my-broker
     - address: my-queue
       direction: out
       connection: my-broker
     connectors:  3
     - name: my-broker
       host: broker
       port: 5672
       routeContainer: true
   ```

   1 The address (or set of addresses) for which messages should be stored on a broker queue.

   2 The autolink configuration. It defines the incoming and outgoing links and connection to be used to send and receive the messages on the broker.

   3 The connection to be used to connect the routers to the message broker.

   The Operator applies the changes to the router network and restarts each Pod.

3. Verify that the router has established the autolinks to the message broker.

```
$ oc exec router-mesh-6d6dccb57f-x5cqf -it -- qdstat --autolinks
AutoLinks
  addr      dir  phs  extAddr  link  status  lastErr
  ========================================================
  my-queue  in   1             26    active
  my-queue  out  0             27    active
```

4. If you have additional router deployment Custom Resources that define routers in the router network, repeat this procedure for each CR.
   Each router in the router network must have the same address configuration.

Additional resources

- For more information about routing messages to and from broker queues, see Routing Messages through broker queues.

CHAPTER 8. USING PROMETHEUS AND GRAFANA TO MONITOR THE ROUTER NETWORK

Prometheus is container-native software built for storing historical data and for monitoring large, scalable systems such as AMQ Interconnect. It gathers data over an extended time, rather than just for the currently running session.

You use Prometheus and Alertmanager to monitor and store AMQ Interconnect data so that you can use a graphical tool, such as Grafana, to visualize and run queries on the data.

8.1. SETTING UP PROMETHEUS AND GRAFANA

Before you can view AMQ Interconnect dashboards, you must deploy and configure Prometheus, Alertmanager, and Grafana in the OpenShift project in which AMQ Interconnect is deployed. All of the required configuration files are provided in a GitHub repository.

Procedure

1. Clone the **qdr-monitoring** GitHub repository.
   This repository contains example configuration files needed to set up Prometheus and Grafana to monitor AMQ Interconnect.

   ```
   $ git clone https://github.com/interconnectedcloud/qdr-monitoring
   ```

2. Set the **NAMESPACE** environment variable to the name of the project where you deployed AMQ Interconnect.
   For example, if you deployed AMQ Interconnect in the **example** project, set the **NAMESPACE** environment variable as follows:

   ```
   $ export NAMESPACE=example
   ```

3. Run the **deploy-monitoring.sh** script.
   This script creates and configures the OpenShift resources needed to deploy Prometheus, Alertmanager, and Grafana in your OpenShift project. It also configures two dashboards that provide metrics for the router network.

   ```
   $ ./deploy-monitoring.sh
   ```

   An alternative method of running this script is to to specify the target project as a parameter. For example:

   ```
   $ ./deploy-monitoring.sh example
   ```

Additional resources

- For more information about Prometheus, see the Prometheus documentation.
- For more information about Grafana, see the Grafana documentation.

8.2. VIEWING AMQ INTERCONNECT DASHBOARDS IN GRAFANA

After setting up Prometheus and Grafana, you can visualize the AMQ Interconnect data on the following Grafana dashboards:

Qpid Dispatch Router
: Shows metrics for:

Qpid Dispatch Router
: Shows metrics for:

- **Deliveries ingress**
- **Deliveries egress**
- **Deliveries ingress route container**
- **Deliveries egress route container**
- **Deliveries redirected to fallback destination**
- **Dropped presettled deliveries**
- **Presettled deliveries**
- **Auto links**
- **Link routes**
- **Address count**
- **Connection count**
- **Link count**

Qpid Dispatch Router - Delayed Deliveries
: Shows metrics for:

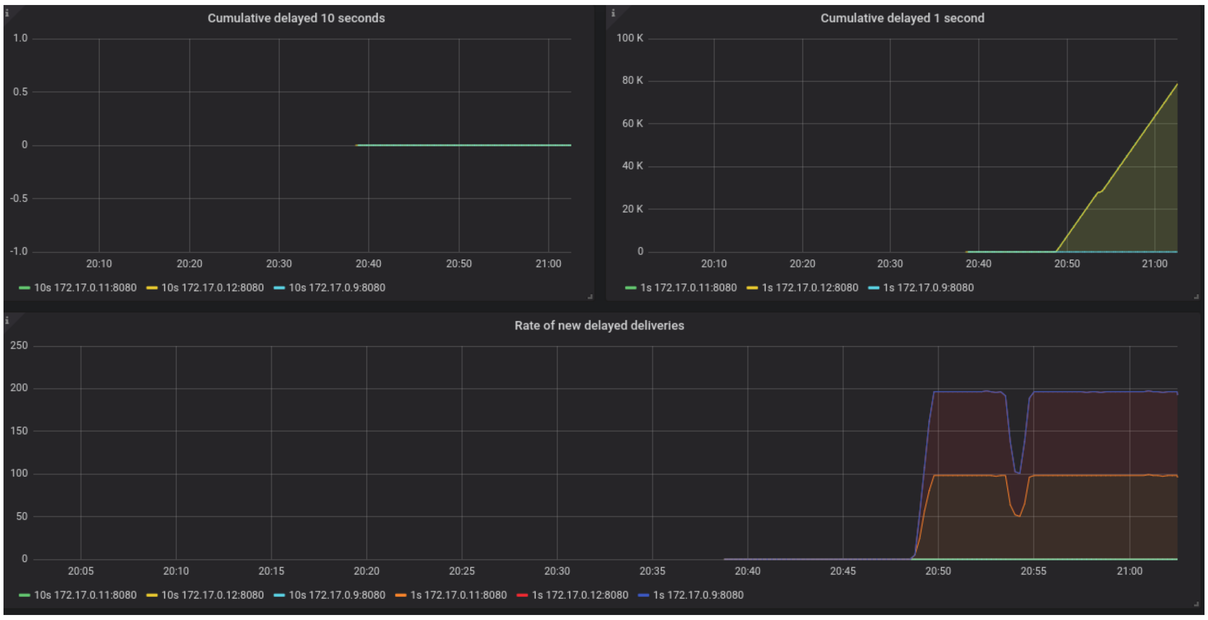
- **Cumulative delayed 10 seconds**
- **Cumulative delayed 1 second**
- **Rate of new delayed deliveries**

For more information about these metrics, see Section 8.3, “Router metrics”.

Procedure

1. In the OpenShift web console, switch to **Networking → Routes**, and click the URL for the **grafana** Route.
   The Grafana Log In page appears.
2. Enter your user name and password, and then click **Log In**.
   The default Grafana user name and password are both **admin**. After logging in for the first time, you can change the password.
3. On the top header, click the dashboard drop-down menu, and then select the **Qpid Dispatch Router** or **Qpid Dispatch Router - Delayed Deliveries** dashboard.

Figure 8.1. Delayed Deliveries dashboard



8.3. ROUTER METRICS

The following metrics are available in Prometheus:

qdr_connections_total

The total number of network connections to the router. This includes connections from and to any AMQP route container.

qdr_links_total

The total number of incoming and outgoing links attached to the router.

qdr_addresses_total

The total number of addresses known to the router.

qdr_routers_total

The total number of routers known to the router.

qdr_link_routes_total

The total number of active and inactive link routes configured for the router. See Understanding link routing for more details.

qdr_auto_links_total

The total number of incoming and outgoing auto links configured for the router. See Configuring brokered messaging for more details about autolinks.

qdr_presettled_deliveries_total

The total number of presettled deliveries arriving at the router. The router settles the incoming deliveries and propagates the settlement to the message destination, also known as *fire and forget*.

qdr_dropped_presettled_deliveries_total

The total number of presettled deliveries that the router dropped due to congestion. The router settles the incoming deliveries and propagates the settlement to the message destination, also known as fire and forget.

qdr_accepted_deliveries_total

The total number of deliveries accepted at the router. See Understanding message routing for more information on accepted deliveries.

qdr_released_deliveries_total

The total number of deliveries released at the router. See Understanding message routing for more information on released deliveries.

qdr_rejected_deliveries_total

The total number of deliveries rejected at the router. See Understanding message routing for more information on rejected deliveries.

qdr_modified_deliveries_total

The total number of deliveries modified at the router. See Understanding message routing for more information on modified deliveries.

qdr_deliveries_ingress_total

The total number of messages delivered to the router from clients. This includes management messages, but not route control messages.

qdr_deliveries_egress_total

The total number of messages sent from the router to clients. This includes management messages, but not route control messages.

qdr_deliveries_transit_total, qdr_deliveries_ingress_route_container_total

The total number of messages passing through the router for delivery to a different router.

qdr_deliveries_egress_route_container_total

The total number of deliveries sent to AMQP route containers from the router This includes messages to an AMQ Broker instance and management messages, but not route control messages.

qdr_deliveries_delayed_1sec_total

The total number of deliveries forwarded by the router that were unsettled for more than one second.

qdr_deliveries_delayed_10sec_total

The total number of deliveries forwarded by the router that were unsettled for more than ten seconds.

qdr_deliveries_stuck_total

The total number of deliveries that cannot be delivered. Typically, deliveries cannot be delivered due to lack of credit as described in Message routing flow control

qdr_links_blocked_total

The total number of links that are blocked.

qdr_deliveries_redirected_to_fallback_total

The total number of deliveries that were forwarded to a fallback destination. See Handling undeliverable messages for more information.

Additional information

See Section 8.2, “Viewing AMQ Interconnect dashboards in Grafana” .

CHAPTER 9. USING THE AMQ INTERCONNECT WEB CONSOLE TO MONITOR THE ROUTER NETWORK

You can use the AMQ Interconnect web console to monitor the status and performance of your router network. By default, when you create a router deployment, the AMQ Interconnect Operator generates the credentials to access the console and stores them in a Secret.

Procedure

1. In OpenShift, switch to **Networking → Routes**, and click the console Route.
   The web console opens in a new tab.

2. To connect to the web console, complete the following fields:

   Port

   Enter **443**.

   User name

   Enter the user name.
   To find the user name and password for accessing the web console, navigate to **Workloads → Secrets**. The Secret containing the web console credentials is called **<*application-name*>-users** (for example, **router-mesh-users**).

   The syntax for the user name is <*user*>@<*domain*> (the domain is the OpenShift application name, which is the name of the Custom Resource that describes the router deployment). For example, **guest@router-mesh**.

   Password

   Enter the password defined in the **<*application-name*>-users** Secret.

3. Click **Connect**.
   The **Routers** page is displayed showing all of the routers in the router network.

4. Use the web console tabs to monitor the router network.

This tab…	Provides…
Overview	Aggregated information about routers, addresses, links, connections, and logs.
Entities	Detailed information about each AMQP management entity for each router in the router network. Some of the attributes have charts that you can add to the **Charts** tab.
Topology	A graphical view of the router network, including routers, clients, and brokers. The topology shows how the routers are connected, and how messages are flowing through the network.
Charts	Graphs of the information selected on the **Entities** tab.
Message Flow	A chord diagram showing the real-time message flow by address.

This tab...	Provides...
Schema	The management schema that controls each of the routers in the router network.

Revised on 2020-11-23 18:32:57 UTC