Red Hat AMQ 7.2

Using AMQ Online on OpenShift Container Platform

For use with AMQ Online 1.0

Last Updated: 2019-01-29

Red Hat AMQ 7.2 Using AMQ Online on OpenShift Container Platform

For use with AMQ Online 1.0

Legal Notice

Abstract

This guide describes how to use AMQ Online.

Table of Contents

CHAPTER 1. INTRODUCTION

1.1. AMQ ONLINE OVERVIEW

Red Hat AMQ Online is an OpenShift-based mechanism for delivering messaging as a managed service. With Red Hat AMQ Online, administrators can configure a cloud-native, multi-tenant messaging service either in the cloud or on premise. Developers can provision messaging using the AMQ Online console. Multiple development teams can provision the brokers and queues from the console, without requiring each team to install, configure, deploy, maintain, or patch any software.

AMQ Online can provision different types of messaging depending on your use case. A user can request messaging resources by creating an address space. AMQ Online currently supports two address space types, standard and brokered, each with different semantics. The following diagrams illustrate the high-level architecture of each address space type:

Figure 1.1. Standard address space

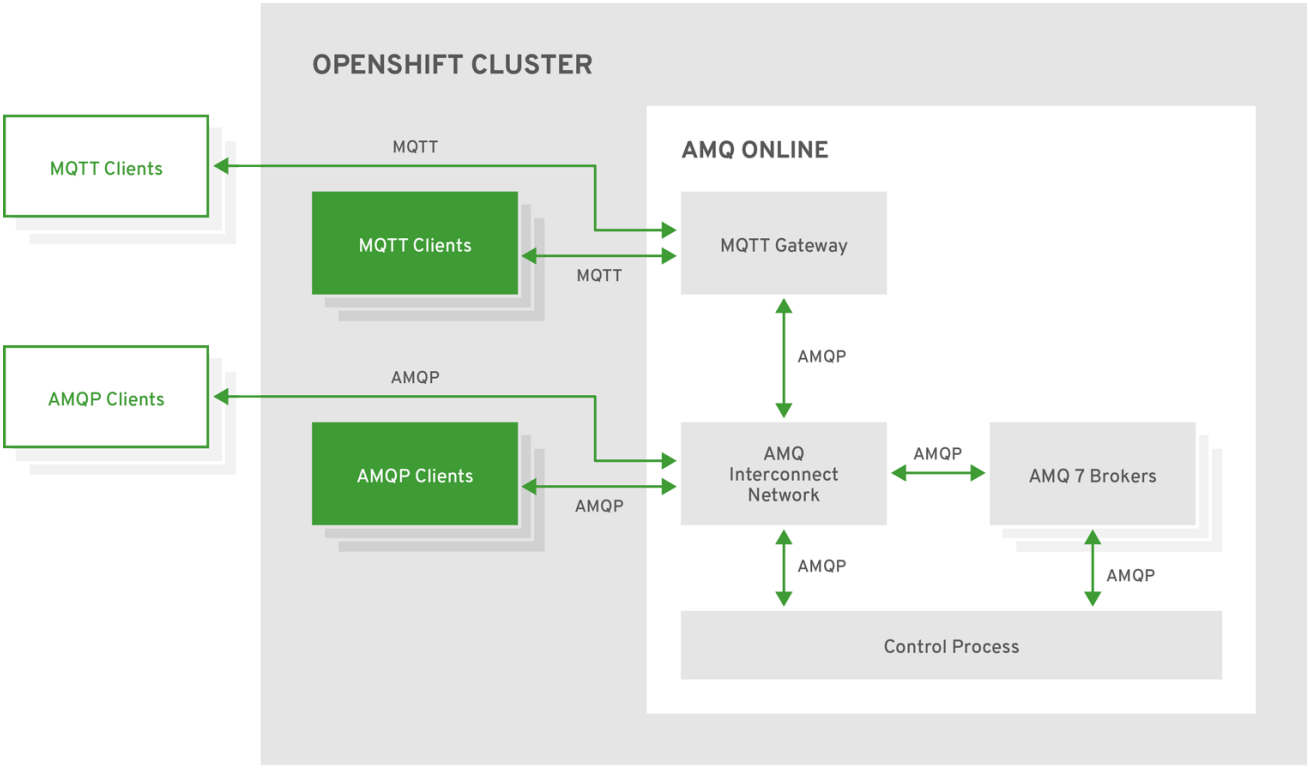
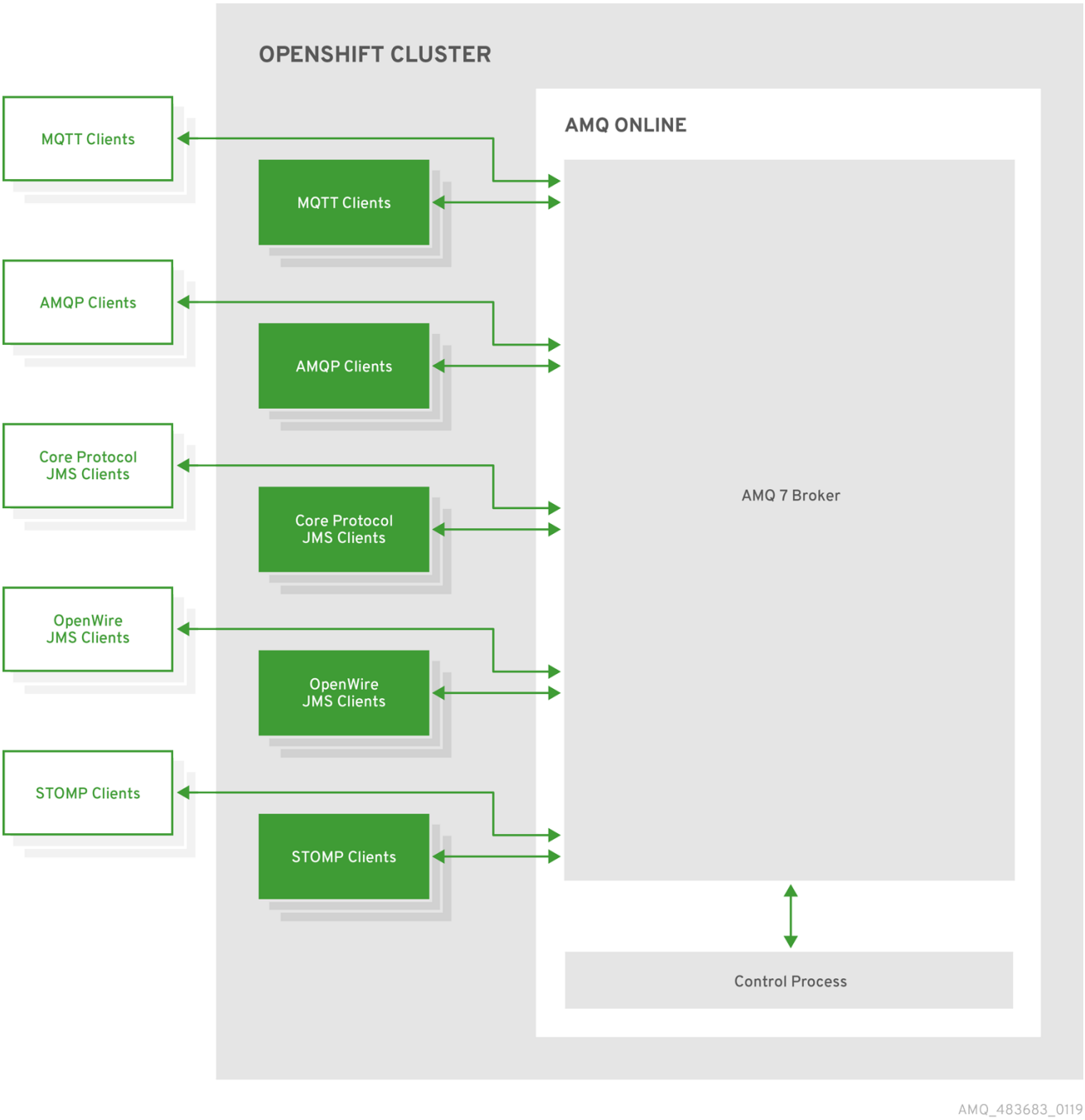


Figure 1.2. Brokered address space



1.2. SUPPORTED FEATURES

The following table shows the supported features for AMQ Online 1.0:

Table 1.1. Supported features reference table

Feature		Brokered address space	Standard address space
Address type	Queue	Yes	Yes
	Topic	Yes	Yes

Feature		Brokered address space	Standard address space
	Multicast	No	Yes
	Anycast	No	Yes
	Subscription	No	Yes
Messaging protocol	AMQP	Yes	Yes
	MQTT	Yes	Technology preview only
	CORE	Yes	No
	OpenWire	Yes	No
	STOMP	Yes	No
Transports	TCP	Yes	Yes
	WebSocket	Yes	Yes
Durable subscriptions	JMS durable subscriptions	Yes	No
	"Named" durable subscriptions	No	Yes
JMS	Transaction support	Yes	No
	Selectors on queues	Yes	No
	Message ordering guarantees (including prioritization)	Yes	No
Scalability	Scalable distributed queues and topics	No	Yes

1.3. AMQ ONLINE USER ROLES

AMQ Online users can be defined broadly in terms of two user roles: service administrator and messaging tenant. Depending on the size of your organization, these roles might be performed by the same person or different people.

The messaging tenant can request messaging resources, using both cloud-native APIs and tools. The messaging tenant can also manage the users and permissions of a particular address space within the

messaging system as well as create address spaces and addresses. *Using AMQ Online on OpenShift Container Platform* provides information about how to accomplish these tasks.

The service administrator role performs the initial installation and any subsequent upgrades. The service administrator might also deploy and manage the messaging infrastructure, such as monitoring the routers, brokers, and administration components; and creating the address space plans and address plans. For more information about how to set up and manage AMQ Online as well as configure the infrastructure and plans, see *Installing and Managing AMQ Online on OpenShift Container Platform*.

1.4. SUPPORT STATEMENT AND RESOURCES

The AMQ Online 1.0 release is provided as a service to customers who want to try the AMQ Online features and work with support when problems are encountered. Support for Beta releases is limited to commercially reasonable effort and non-production use cases, and all support cases must be opened with a severity of 4. Patches will not be provided, but bug fixes might be incorporated in future releases. To contact support, visit Open a Support Case.

1.5. SUPPORTED CONFIGURATIONS

For more information about AMQ Online supported configurations see Red Hat AMQ 7 Supported Configurations.

1.6. DOCUMENT CONVENTIONS

1.6.1. Variable text

This document contains code blocks with variables that you must replace with values specific to your installation. In this document, such text is styled as italic monospace.

For example, in the following code block, replace ***my-namespace*** with the namespace used in your installation:

```
sed -i 's/amq-online-infra/my-namespace/' install/bundles/enmasse-with-
standard-authservice/*.yaml
```

CHAPTER 2. MANAGING ADDRESS SPACES

AMQ Online is configured to support managing address spaces using the OpenShift command-line tools. Address spaces are managed like any other OpenShift resource using **oc**.

2.1. ADDRESS SPACE

An address space is a group of addresses that can be accessed through a single connection (per protocol). This means that clients connected to the endpoints of an address space can send messages to or receive messages from any authorized address within that address space. An address space can support multiple protocols, as defined by the address space type.

AMQ Online has two types of address spaces:

- Standard
- Brokered

2.2. STANDARD ADDRESS SPACE

The standard address space is the default address space in AMQ Online. It consists of an AMQP router network in combination with attachable storage units. Clients connect to a message router, which forwards messages to or from one or more message brokers. This address space type is appropriate when you have many connections and addresses. However, the standard address space has the following limitations:

- No transaction support
- No message ordering
- No selectors on queues
- No browsing on queues
- No message groups

Clients connect and send and receive messages in this address space using the AMQP or MQTT protocols. Note that MQTT does not support qos2 or retained messages.

2.2.1. Standard address types

The standard address space supports five different address types:

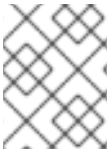
- queue
- topic
- anycast
- multicast
- subscription

2.2.1.1. Queue

The queue address type is a store-and-forward queue. This address type is appropriate for implementing a distributed work queue, handling traffic bursts, and other use cases where you want to decouple the producer and consumer. A queue can be sharded across multiple storage units. Message ordering might be lost for queues in the standard address space.

2.2.1.2. Topic

The topic address type supports the publish-subscribe messaging pattern where there are 1..N producers and 1..M consumers. Each message published to a topic address is forwarded to all subscribers for that address. A subscriber can also be durable, in which case messages are kept until the subscriber has acknowledged them.



NOTE

If you create a subscription on a topic, any senders to that topic must specify the **`topic`** capability.

2.2.1.2.1. Hierarchical topics and wildcards

A client receiving from a topic address can specify a wildcard address with the topic address as the root. The wildcard behavior follows the MQTT syntax:

- **`/`** is a separator
- **`+`** matches one level
- # matches one or more levels

So, for example:

- **`a/#/b`** matches **`a/foo/b`**, **`a/bar/b`**, and **`a/foo/bar/b`**
- **`a/+/b`** matches **`a/foo/b`** and **`a/bar/b`**, but would not match **`a/foo/bar`**

In the standard address space, the first level must always be a defined topic address; that is, # and **`+`** are not valid as the first characters of a subscribing address.

2.2.1.3. Anycast

The anycast address type is a scalable direct address for sending messages to one consumer. Messages sent to an anycast address are not stored, but are instead forwarded directly to the consumer. This method makes this address type ideal for request-reply (RPC) uses or even work distribution. This is the cheapest address type as it does not require any persistence.

2.2.1.4. Multicast

The multicast address type is a scalable direct address for sending messages to multiple consumers. Messages sent to a multicast address are forwarded to all consumers receiving messages on that address. Because message acknowledgments from consumers are not propagated to producers, only pre-settled messages can be sent to multicast addresses.

2.2.1.5. Subscription

The subscription address type allows a subscription to be created for a topic that holds messages

published to the topic even if the subscriber is not attached. The subscription is accessed by the consumer using <topic-address>::<subscription-address>. For example, for a subscription **mysub** on a topic **mytopic** the consumer consumes from the address **mytopic::mysub**.

2.3. BROKERED ADDRESS SPACE

The brokered address space is designed to support broker-specific features, at the cost of limited scale in terms of the number of connections and addresses. This address space supports JMS transactions, message groups, and selectors on queues and topics.

Clients can connect and send and receive messages in this address space using the AMQP, CORE, OpenWire, and MQTT protocols.

2.3.1. Brokered address types

The brokered address space supports two address types:

- queue
- topic

2.3.1.1. Queue

The queue address type is a store-and-forward queue. This address type is appropriate for implementing a distributed work queue, handling traffic bursts, and other use cases where you want to decouple the producer and consumer. A queue in the brokered address space supports selectors, message groups, transactions, and other JMS features. Message order can be lost with released messages.

2.3.1.2. Topic

The topic address type supports the publish-subscribe messaging pattern in which there are 1..N producers and 1..M consumers. Each message published to a topic address is forwarded to all subscribers for that address. A subscriber can also be durable, in which case messages are kept until the subscriber has acknowledged them.

2.3.1.2.1. Hierarchical topics and wildcards

A client receiving from a topic address can specify a wildcard address with the topic address as the root. The wildcard behavior follows the MQTT syntax:

- **/** is a separator
- **+** matches one level
- # matches one or more levels

So, for example:

- **a/#/b** matches **a/foo/b**, **a/bar/b**, **a/foo/bar/b**
- **a/+/b** matches **a/foo/b** and **a/bar/b**, but would not match **a/foo/bar**

2.4. ADDRESS SPACE PLANS

An address space is configured with an address space plan, which describes the allowed resource usage of that address space. The address space plans are configured by the service administrator and can vary between AMQ Online installations.

The address space plan can be changed if the address space requires more, or less, resources.

2.5. LISTING AVAILABLE PLANS USING THE COMMAND LINE

You can list the address space plans available for your address space.

Procedure

1. Log in as a messaging tenant:

```
oc login -u developer
```

2. Retrieve the schema with available plans (replace **`standard`** with **`brokered`** for the brokered address space type):

```
oc get addressspaceschema standard -o yaml
```

2.6. ADDRESS SPACE EXAMPLE

This address space example shows only the required options to create an **`AddressSpace`**.

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard 1
  plan: standard-unlimited 2
```

1. The address space type can be either **`brokered`** or **`standard`**.
2. The address space plan depends on the address space type and what has been configured by the AMQ Online administrator. See Listing available plans for details on how to get the available plans.

2.7. ADDRESS SPACE EXAMPLE USING AUTHENTICATION SERVICE

This address space example shows how you can configure the authentication service and endpoints of an **`AddressSpace`**.

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard 1
  plan: standard-unlimited 2
  authenticationService:
```

```
    type: standard 3
  endpoints: 4
    - name: messaging
      service: messaging
      expose: 5
        type: route
        routeHost: messaging.example.com
        routeTlsTermination: passthrough
        routeServicePort: amqps
      cert: 6
        provider: certBundle
        tlsKey: ZXhhbXBsZSAtbgo=
        tlsCert: ZXhhbXBsZSAtbgo=
    - name: mqtt
      service: mqtt
      expose:
        type: loadbalancer
        loadBalancerPorts:
        - mqtt
        - mqtts
      cert:
        provider: selfsigned
    - name: console
      service: console
      expose:
        type: route
        routeTlsTermination: reencrypt
        routeServicePort: https
      cert:
        provider: openshift
```

1. The address space type can be either **`brokered`** or **`standard`**.

2. The address space plan depends on the address space type and what has been configured by the AMQ Online administrator. See Listing available plans for details on how to get the available plans.

3. The authentication service type can be **`none`**, **`standard`** or **`external`**. **`External`** requires an additional field, **`details`**, which includes **`host`** and **`port`** information.

4. The endpoints that must be configured for the **`AddressSpace`**. Endpoints can specify either **`messaging`**, **`console`**, or **`mqtt`** services. However, the **`mqtt`** service is supported for the **`standard`** address space type only. Endpoints can be changed by replacing the address space.

5. Exposes the configuration of an endpoint. Endpoints can be kept internal to the cluster, or exposed using an OpenShift route or a load-balancer service. Exposed configuration can be changed by replacing the address space.

6. The certificate configuration can be used to specify how the endpoint certificate is provided. The provider can be **`selfsigned`** (default), **`wildcard`** (requires it to be enabled by the AMQ Online administrator), **`openshift`** (signed by the OpenShift cluster CA), or **`certBundle`** (base64-encoded PEM key and certificate). When using the **`certBundle`** provider, the certificate can be rotated by updating the **`tlsKey`** and **`tlsCert`** values and replacing the address space.

2.8. EXAMPLE ADDRESS SPACE STATUS OUTPUT

The **AddressSpace** resource contains a **status** field that can be used to retrieve information about its state and endpoints. The following output is an example of the output you can get from running **oc get addressspace myspace -o yaml**:

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  ...
status:
  isReady: false (1)
  messages:
    - "One or more deployments are not ready: "
  endpointStatuses: (2)
    - name: messaging
      cert: aGVsbG8= (3)
      serviceHost: messaging-123.enmasse-infra.svc (4)
      servicePorts: (5)
        - name: amqp
          port: 5672
        - name: amqps
          port: 5671
      externalHost: messaging.example.com (6)
      externalPorts: (7)
        - name: amqps
          port: 443
```

1. The **status.isReady** field can be either **true** or **false**.
2. The **status.endpointStatuses** field provides information about available endpoints for this address space.
3. The **cert** field contains the base64-encoded certificate for a given endpoint.
4. The **serviceHost** field contains the cluster-internal host name for a given endpoint.
5. The **servicePorts** field contains the available ports for the cluster-internal host.
6. The **externalHost** field contains the external host name for a given endpoint.
7. The **externalPorts** field contains the available ports for the external host.

2.9. CREATING ADDRESS SPACES USING THE COMMAND LINE

In AMQ Online, you create address spaces using standard command-line tools.

Procedure

1. Log in as a messaging tenant:
   -

```
oc login -u developer
```

2. Create the project for the messaging application:

```
oc new-project myapp
```

3. Create an address space definition:

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard
  plan: standard-unlimited
```

4. Create the address space:

```
oc create -f standard-address-space.yaml
```

5. Check the status of the address space:

```
oc get addressspace myspace -o jsonpath={.status.isReady}
```

The address space is ready for use when the previous command outputs **`true`**.

2.10. EXAMPLE COMMANDS FOR RETRIEVING ADDRESS SPACE INFORMATION

The following table shows the commands for retrieving address space information, such as the AMQ Online Console host name.

Table 2.1. Retrieving address space information commands table

To retrieve the…	Run this command:
AMQ Online Console host name	**`oc get addressspace myspace -o jsonpath={.status.endpointStatuses[?(@.name==\'console\')].externalHost}`**
status of an address space	**`oc get addressspace myspace -o jsonpath={.status.isReady}`**
base64-encoded PEM certificate for the messaging endpoint	**`oc get addressspace myspace -o jsonpath={.status.endpointStatuses[?(@.name==\'messaging\')].cert}`**
host name for the messaging endpoint	**`oc get addressspace myspace -o jsonpath={.status.endpointStatuses[?(@.name==\'messaging\')].externalHost}`**

2.11. REPLACING ADDRESS SPACES USING THE COMMAND LINE

Address spaces can be replaced in order to change the plan, endpoints, or network policies, or to replace certificates if using the **certBundle** certificate provider. When changing the plan, AMQ Online will attempt to apply the new plan if the current set of addresses fits within the new quota. If it does not, an error is provided on the **AddressSpace** resource.

Procedure

1. Log in as a messaging tenant:

```
oc login -u developer
```

2. Select the project for the messaging application:

```
oc project myapp
```

3. Update address space definition:

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard
  plan: standard-small
```

4. Replace the address space:

```
oc replace -f standard-address-space-replace.yaml
```

5. Check the status of the address space:

```
oc get addressspace myspace -o jsonpath={.status.isReady}
```

   The address space is ready for use when the above command outputs **true**.

CHAPTER 3. MANAGING ADDRESSES

AMQ Online is configured to support managing addresses using the OpenShift command-line tools and the AMQ Online Console. **`Address`** resources can be managed like any other OpenShift API resource using **`oc`**.

3.1. ADDRESS

An address is part of an address space and represents a destination for sending and receiving messages. An address has a type, which defines the semantics of sending messages to and receiving messages from that address.

The types of addresses available in AMQ Online depend on the address space type.

3.2. ADDRESS PLANS

An address is configured with an address plan, which describes the resource usage of that address. The address plans are configured by the service administrator and can vary between AMQ Online installations. The number of addresses that can be created, and what plans are available, depends on quota enforced by the address space plan.

Some address types also support changing the **`plan`** field: **`queue`**, **`anycast`**, and **`multicast`** address types in the **`standard`** address space support changing the plan as long as the new plan does not exceed the allowed quota. For queues, addresses are dynamically migrated across brokers, which might cause reordering of messages.

3.2.1. Address example

```
apiVersion: enmasse.io/v1beta1
kind: Address
metadata:
    name: myspace.myqueue (1)
spec:
    address: myqueue (2)
    type: queue (3)
    plan: standard-small-queue (4)
```

1. The address name must be prefixed with the address space name and a dot. Address names can only include alphanumeric characters.
2. The address is the messaging address this address resource represents.
3. The address type dictates the semantics of this address.
4. The address plan describes the resource usage for the address. See Listing available plans for details on how to get the available plans.

3.3. CREATING ADDRESSES USING THE COMMAND LINE

You can create addresses using the command line.

Procedure

1. Create an address definition:

```
apiVersion: enmasse.io/v1beta1
kind: Address
metadata:
    name: myspace.myqueue
spec:
    address: myqueue
    type: queue
    plan: standard-small-queue
```

NOTE

Prefixing the name with the address space name is required to ensure addresses from different address spaces do not collide.

2. Create the address:

```
oc create -f standard-small-queue.yaml
```

3. List the addresses:

```
oc get addresses -o yaml
```

3.4. REPLACING ADDRESSES USING THE COMMAND LINE

Procedure

1. Update an address definition:

```
apiVersion: enmasse.io/v1beta1
kind: Address
metadata:
    name: myspace.myqueue
spec:
    address: myqueue
    type: queue
    plan: standard-xlarge-queue
```

2. Replace the address:

```
oc replace -f standard-xlarge-queue.yaml
```

3. List the addresses:

```
oc get addresses -o yaml
```

CHAPTER 4. USING THE AMQ ONLINE CONSOLE

You can use the AMQ Online Console to perform tasks such as creating an address and viewing message and connection statistics.

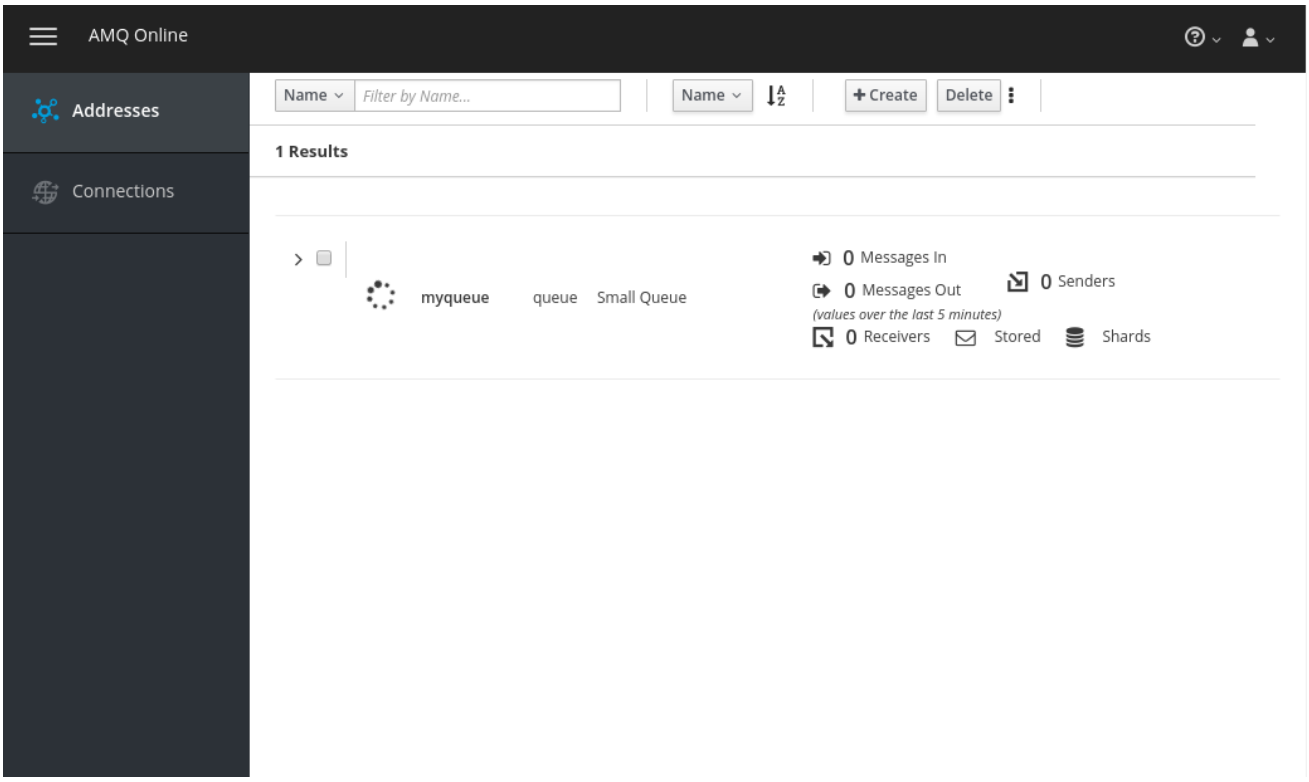
4.1. ACCESSING THE AMQ ONLINE CONSOLE

Prerequisites

- You must have obtained the host name for the AMQ Online Console. For more information about how to obtain the host name, see Example commands for retrieving address space information.

Procedure

1. In a web browser, navigate to **`https://console-host-name`** where ***`console-host-name`*** is the AMQ Online Console host name.
2. Log in with your OpenShift user credentials. The AMQ Online console opens.



4.2. CREATING ADDRESSES USING THE AMQ ONLINE CONSOLE

You can create new addresses using the AMQ Online Console. The type of addresses that you can create are determined by the type of address space.

Prerequisites

- You must have created an address space. For more information see Creating an address space.

Procedure

1. Log in to the AMQ Online Console. For more information, see Accessing the AMQ Online Console.
2. Click **New**. The Create new address window opens.
3. Type a name and select the address type. If selecting **subscription**, from the **Topic** list select the topic name to which you want to create a subscription.
4. Click **Next**.
5. Select a plan and click **Next**.
6. Click **Create**. Your address is displayed in the AMQ Online Console.

4.3. USING THE AMQ ONLINE CONSOLE ADDRESS FILTERING

The address name filtering feature in the AMQ Online Console uses regular expressions. Also, filters are cumulative.

Table 4.1. AMQ Online Console address name filtering behavior

To match…	Use…	Results in…
The beginning of an expression only	A caret followed by an expression: `^my`	All addresses beginning with `my`
An expression	The matching string: `my`	All addresses containing `my`
The end of an expression only	An expression followed by the dollar sign: `my$`	All addresses ending with `my`
An exact expression	A caret followed by an expression and a dollar sign: `^my$`	Only the address `my`

4.4. VIEWING MESSAGE AND CONNECTION STATISTICS USING THE AMQ ONLINE CONSOLE

Prerequisites

- You must be logged into the AMQ Online Console.

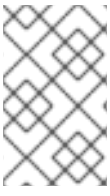
Table 4.2. Message statistics reference table

To view…	On the Addresses page see…
Address status	The first column (the symbol preceding the address name)
Address type	The third column
Address plan	The fourth column

To view…	On the Addresses page see…
Message ingress rate (during the last 5 minutes)	**Messages In**
Message egress rate (during the last 5 minutes)	**Messages Out**
Number of senders attached	**Senders**
Number of receivers attached	**Receivers**
Queue and topic address types only: Number of stored messages on the broker or brokers	**Stored**
Standard address space only: Message deliveries per second	For the desired address, expand the twisty on the left to show the **Senders** table; see the **Delivery Rate** column.

Table 4.3. Connection statistics reference table

To view…	On the Connections page see…
Total number of messages received as long the connection has existed	**Messages In**
Standard address space only: Total number of messages sent as long the connection has existed	**Messages Out**
Total number of messages delivered	For the desired connection, expand the twisty on the left to show the **Senders** and **Receivers** tables; see the **Deliveries** columns.
Standard address space only: Username used by the client to connect	The third column



NOTE

For the brokered address space only, on the Connections page, the number of senders is either `0` or `1`. As soon as one or more senders exist, `1` is displayed rather than reflecting the actual number of senders.

CHAPTER 5. USER MODEL

You can create users to access messaging clients and the AMQ Online Console through user definition files, which specify the authentication type and authorization policies.

Users are configured as **`MessagingUser`** resources. Users can only be created, deleted, read, updated, and listed.

The following example shows the **`user-example1.yaml`** file:

```
apiVersion: user.enmasse.io/v1beta1
kind: MessagingUser
metadata:
  name: myspace.user1
spec:
  username: user1
  authentication:
    type: password
    password: cGFzc3dvcmQ= # Base64 encoded
  authorization:
    - addresses: ["myqueue", "queue1", "queue2", "topic*"]
      operations: ["send", "recv"]
    - addresses: ["anycast1"]
      operations: ["send"]
```

The following fields are required:

- **`metadata.name`**
- **`metadata.namespace`**
- **`spec.authentication`**
- **`spec.authorization`**

The **`spec.authentication`** object defines how the user is authenticated, whereas **`spec.authorization`** defines the authorization policies for that user.

5.1. AUTHENTICATION

The supported values for the authentication type are **`password`** and **`federated`**.

NOTE

In the default configuration, **`password`** authentication type users can only access messaging endpoints, whereas **`federated`** authentication type users can only access the AMQ Online Console.

5.1.1. Password authentication type

For the **`password`** type, an additional field **`password`** must be set to a base64-encoded value of the password for that user. The password will not be printed out when reading the resource.

A password can be base64-encoded on the command line. To encode **`my-password`**, for example:

```
$ echo -n my-password | base64
bXktcGFzc3dvcmQ=
```

5.1.2. Federated authentication type

For the **federated** type, the **provider** field must also be specified. The supported values depend on the identity providers configured for the address space. The **federatedUsername** and **federatedUserid** fields must be set, and must map to the **username** and **userid** in the federated identity provider.

5.2. AUTHORIZATION

In addition, authorization policies can be defined using operations and addresses. Valid operations are **send**, **recv**, **view**, and **manage**.

The **manage** and **view** operations apply to all addresses in the address space.

In the **standard** address space, the asterisk wildcard can be used at the end of an address. The address **top*** matches addresses **topic** and **topic/sub**.

In the **brokered** address space, the plus sign and asterisk wildcards can be used at the end of an address to match a single word (plus sign) or all words (asterisk) after the forward slash delimiter. So, the address **topic/+** matches **topic/sub** but not **topic/s/sub**. The address **topic/*** matches **topic/sub** and **topic/s/sub**.

5.3. MANAGING USERS

AMQ Online user management is only supported when using the **standard** authentication service. On OpenShift, users can be managed using the OpenShift command-line tools.

Prerequisites

- You must have already created an address space.

5.3.1. Creating users using the command line

In AMQ Online users can be created using standard command-line tools.

Prerequisites

- You must have already created an address space.

Procedure

1. Save the user definition to a file:

```
apiVersion: user.enmasse.io/v1beta1
kind: MessagingUser
metadata:
  name: myspace.user1
spec:
  username: user1
```

```
  authentication:
    type: password
    password: cGFzc3dvcmQ= # Base64 encoded
  authorization:
    - addresses: ["myqueue", "queue1", "queue2", "topic*"]
      operations: ["send", "recv"]
    - addresses: ["anycast1"]
      operations: ["send"]
```

2. Create the user and associated user permissions:

```
oc create -f user-example1.yaml
```

3. Confirm that the user was created:

```
oc get messagingusers
```

5.3.2. Deleting users using the command line

Users can be deleted using standard command-line tools.

Prerequisites

- An address space must have been created.
- A user must have been created.

Procedure

1. List the current users:

```
oc get messagingusers
```

2. Delete the desired user:

```
oc delete messaginguser myspace.user1
```

5.3.3. Managing user permissions using the command line

You can edit the permissions for an existing user using the command line.

Prerequisites

- You must have already created a user. For more information see Creating users using the command line.

Procedure

1. Retrieve the user whose permissions you want to edit:

```
oc get messaginguser myspace.user1 -o yaml > user1-example.yaml
```

2. Make the desired permissions change and save the file.
3. From the command line, run the following command to apply the change:

```
oc apply -f user-example1.yaml
```

The new user permissions are applied.

CHAPTER 6. CONNECTING APPLICATIONS TO AMQ ONLINE

You can connect your application to AMQ Online using one of the following client examples.

- AMQ Online Python
- AMQ Online JMS
- AMQ Online JavaScript
- AMQ Online C++
- AMQ Online .NET

To connect to the messaging service from outside the OpenShift cluster, TLS must be used with SNI set to specify the fully qualified host name for the address space. The port used is 443.

The messaging protocols supported depends on the type of address space used. For more information about address space types, see Address space.

6.1. CLIENT EXAMPLES

6.1.1. AMQ Online Python example

You can use the following AMQ Online Python example to connect your application to AMQ Online. This example assumes you have created an address of type **queue** named **myqueue**.

```
from future import print_function, unicode_literals
from proton import Message
from proton.handlers import MessagingHandler
from proton.reactor import Container

class HelloWorld(MessagingHandler):
    def init(self, server, address):
        super(HelloWorld, self).init()
        self.server = server
        self.address = address

    def on_start(self, event):
        conn = event.container.connect(self.server)
        event.container.create_receiver(conn, self.address)
        event.container.create_sender(conn, self.address)

    def on_sendable(self, event):
        event.sender.send(Message(body="Hello World!"))
        event.sender.close()

    def on_message(self, event):
        print(event.message.body)
        event.connection.close()

Container(HelloWorld("amqps://messaging-route-hostname:443",
"myqueue")).run()
```

6.1.2. AMQ Online JMS example

You can use the following AMQ Online JMS example to connect your application to AMQ Online. This example assumes you have created an address of type **queue** named **myqueue**.

```
package org.apache.qpid.jms.example;

import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.ExceptionListener;
import javax.jms.JMSException;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.naming.Context;
import javax.naming.InitialContext;

public class HelloWorld {
    public static void main(String[] args) throws Exception {
        try {
            // The configuration for the Qpid InitialContextFactory has
been supplied in
            // a jndi.properties file in the classpath, which results in
it being picked
            // up automatically by the InitialContext constructor.
            Context context = new InitialContext();

            ConnectionFactory factory = (ConnectionFactory)
context.lookup("myFactoryLookup");
            Destination queue = (Destination)
context.lookup("myQueueLookup");

            Connection connection =
factory.createConnection(System.getProperty("USER"),
System.getProperty("PASSWORD"));
            connection.setExceptionListener(new MyExceptionListener());
            connection.start();

            Session session = connection.createSession(false,
Session.AUTO_ACKNOWLEDGE);

            MessageProducer messageProducer =
session.createProducer(queue);
            MessageConsumer messageConsumer =
session.createConsumer(queue);

            TextMessage message = session.createTextMessage("Hello
world!");
            messageProducer.send(message, DeliveryMode.NON_PERSISTENT,
Message.DEFAULT_PRIORITY, Message.DEFAULT_TIME_TO_LIVE);
            TextMessage receivedMessage = (TextMessage)
messageConsumer.receive(2000L);
```

```
            if (receivedMessage != null) {
                System.out.println(receivedMessage.getText());
            } else {
                System.out.println("No message received within the given
timeout!");
            }

            connection.close();
        } catch (Exception exp) {
            System.out.println("Caught exception, exiting.");
            exp.printStackTrace(System.out);
            System.exit(1);
        }
    }

    private static class MyExceptionListener implements ExceptionListener
{
        @Override
        public void onException(JMSException exception) {
            System.out.println("Connection ExceptionListener fired,
exiting.");
            exception.printStackTrace(System.out);
            System.exit(1);
        }
    }
}
```

with jndi.properties:

```
connectionfactory.myFactoryLookup = amqps://messaging-route-hostname:443?
transport.trustAll=true&transport.verifyHost=false
queue.myQueueLookup = myqueue
```

6.1.3. AMQ Online JavaScript example

You can use the following AMQ Online JavaScript example to connect your application to AMQ Online. This example assumes you have created an address of type **queue** named **myqueue**.

```
var container = require('rhea');
container.on('connection_open', function (context) {
    context.connection.open_receiver('myqueue');
    context.connection.open_sender('myqueue');
});
container.on('message', function (context) {
    console.log(context.message.body);
    context.connection.close();
});
container.on('sendable', function (context) {
    context.sender.send({body:'Hello World!'});
    context.sender.detach();
});
container.connect({username: 'username', password: 'password', port:443,
host:'messaging-route-hostname', transport:'tls',
rejectUnauthorized:false});
```

-

6.1.3.1. AMQ Online JavaScript example using WebSockets

```
var container = require('rhea');
var WebSocket = require('ws');

container.on('connection_open', function (context) {
    context.connection.open_receiver('myqueue');
    context.connection.open_sender('myqueue');
});
container.on('message', function (context) {
    console.log(context.message.body);
    context.connection.close();
});
container.on('sendable', function (context) {
    context.sender.send({body:'Hello World!'});
    context.sender.detach();
});

var ws = container.websocket_connect(WebSocket);
container.connect({username: 'username', password: 'password',
connection_details: ws("wss://messaging-route-hostname:443", ["binary"],
{rejectUnauthorized: false})});
```

6.1.4. AMQ Online C++ example

The C++ client has equivalent **simple_recv** and **simple_send** examples with the same options as Python. However, the C++ library does not perform the same level of processing on the URL; in particular it will not accept **amqps://** to imply using TLS, so the example needs to be modified as follows:

```
#include <proton/connection.hpp>
#include <proton/container.hpp>
#include <proton/default_container.hpp>
#include <proton/delivery.hpp>
#include <proton/message.hpp>
#include <proton/messaging_handler.hpp>
#include <proton/ssl.hpp>
#include <proton/thread_safe.hpp>
#include <proton/tracker.hpp>
#include <proton/url.hpp>

#include <iostream>

#include "fake_cpp11.hpp"

class hello_world : public proton::messaging_handler {
  private:
    proton::url url;

  public:
    hello_world(const std::string& u) : url(u) {}

    void on_container_start(proton::container& c) OVERRIDE {
```

```
        proton::connection_options co;
        co.ssl_client_options(proton::ssl_client_options());
        c.client_connection_options(co);
        c.connect(url);
    }

    void on_connection_open(proton::connection& c) OVERRIDE {
        c.open_receiver(url.path());
        c.open_sender(url.path());
    }

    void on_sendable(proton::sender &s) OVERRIDE {
        proton::message m("Hello World!");
        s.send(m);
        s.close();
    }

    void on_message(proton::delivery &d, proton::message &m) OVERRIDE {
        std::cout << m.body() << std::endl;
        d.connection().close();
    }
};

int main(int argc, char **argv) {
    try {
        std::string url = argc > 1 ? argv[1] : "messaging-route-
hostname:443/myqueue";

        hello_world hw(url);
        proton::default_container(hw).run();

        return 0;
    } catch (const std::exception& e) {
        std::cerr << e.what() << std::endl;
    }

    return 1;
}
```

6.1.5. AMQ Online .NET example

You can use the following AMQ Online .NET example to connect your application to AMQ Online. This example assumes you have created an address of type **queue** named **myqueue**.

```
using System;
using Amqp;

namespace Test
{
    public class Program
    {
        public static void Main(string[] args)
        {
            String url = (args.Length > 0) ? args[0] :
"amqps://messaging-route-hostname:443";
```

```
            String address = (args.Length > 1) ? args[1] : "myqueue";

            Connection.DisableServerCertValidation = true;
            Connection connection = new Connection(new Address(url));
            Session session = new Session(connection);
            SenderLink sender = new SenderLink(session, "test-sender",
address);

            Message messageSent = new Message("Test Message");
            sender.Send(messageSent);

            ReceiverLink receiver = new ReceiverLink(session, "test-
receiver", address);
            Message messageReceived =
receiver.Receive(TimeSpan.FromSeconds(2));
            Console.WriteLine(messageReceived.Body);
            receiver.Accept(messageReceived);

            sender.Close();
            receiver.Close();
            session.Close();
            connection.Close();
        }
    }
}
```

APPENDIX A. USING YOUR SUBSCRIPTION

AMQ Online is provided through a software subscription. To manage your subscriptions, access your account at the Red Hat Customer Portal.

Accessing your account

1. Go to access.redhat.com.
2. If you do not already have an account, create one.
3. Log in to your account.

Activating a subscription

1. Go to access.redhat.com.
2. Navigate to **My Subscriptions**.
3. Navigate to **Activate a subscription** and enter your 16-digit activation number.

Downloading zip and tar files

To access zip or tar files, use the Red Hat Customer Portal to find the relevant files for download. If you are using RPM packages, this step is not required.

1. Open a browser and log in to the Red Hat Customer Portal **Product Downloads** page at access.redhat.com/downloads.
2. Locate the **Red Hat AMQ Online** entries in the **JBOSS INTEGRATION AND AUTOMATION** category.
3. Select the desired AMQ Online product. The Software Downloads page opens.
4. Click the **Download** link for your component.

Registering your system for packages

To install RPM packages on Red Hat Enterprise Linux, your system must be registered. If you are using zip or tar files, this step is not required.

1. Go to access.redhat.com.
2. Navigate to **Registration Assistant**.
3. Select your OS version and continue to the next page.
4. Use the listed command in your system terminal to complete the registration.

To learn more see How to Register and Subscribe a System to the Red Hat Customer Portal.

Revised on 2019-01-29 20:16:03 UTC