



Red Hat AMQ 7.2

Using AMQ Online on OpenShift Container Platform

For use with AMQ Online 1.1

Red Hat AMQ 7.2 Using AMQ Online on OpenShift Container Platform

For use with AMQ Online 1.1

Legal Notice

Copyright © 2019 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide describes how to use AMQ Online.

Table of Contents

CHAPTER 1. INTRODUCTION	4
1.1. AMQ ONLINE OVERVIEW	4
1.2. SUPPORTED FEATURES	5
1.3. AMQ ONLINE USER ROLES	6
1.4. SUPPORTED CONFIGURATIONS	7
1.5. DOCUMENT CONVENTIONS	7
1.5.1. Variable text	7
CHAPTER 2. MANAGING ADDRESS SPACES	8
2.1. ADDRESS SPACE	8
2.2. STANDARD ADDRESS SPACE	8
2.2.1. Standard address types	8
2.2.1.1. Queue	8
2.2.1.2. Topic	9
2.2.1.2.1. Hierarchical topics and wildcards	9
2.2.1.2.2. Known issue with creating a subscriber on a hierarchical topic	9
2.2.1.3. Anycast	9
2.2.1.4. Multicast	9
2.2.1.5. Subscription	10
2.3. BROKERED ADDRESS SPACE	10
2.3.1. Brokered address types	10
2.3.1.1. Queue	10
2.3.1.2. Topic	10
2.3.1.2.1. Hierarchical topics and wildcards	10
2.3.1.2.2. Known issue with creating a subscriber on a hierarchical topic	11
2.4. ADDRESS SPACE PLANS	11
2.5. LISTING AVAILABLE ADDRESS SPACE PLANS USING THE COMMAND LINE	11
2.6. LISTING AVAILABLE AUTHENTICATION SERVICES USING THE COMMAND LINE	11
2.7. ADDRESS SPACE EXAMPLES	12
2.7.1. Address space example	12
2.7.2. Address space example using an authentication service	12
2.7.3. Address space example exposing endpoints externally	12
2.7.3.1. OpenShift route example	12
2.7.3.2. OpenShift LoadBalancer service example	13
2.7.4. Configuring an address space certificate provider	14
2.7.4.1. openshift provider	14
2.7.4.2. selfsigned provider	15
2.7.4.3. certBundle provider	15
2.7.5. Address space example exports	16
2.7.5.1. ConfigMap and Secret type export examples	16
2.7.5.2. Service type export example	16
2.8. EXAMPLE ADDRESS SPACE STATUS OUTPUT	16
2.9. EXAMPLE OF EXPORTING ADDRESS SPACE INFORMATION INTO THE APPLICATION NAMESPACE	17
2.10. CREATING ADDRESS SPACES USING THE COMMAND LINE	18
2.11. CREATING AN ADDRESS SPACE USING THE RED HAT AMQ CONSOLE	19
2.12. DELETING AN ADDRESS SPACE USING THE RED HAT AMQ CONSOLE	19
2.13. EXAMPLE COMMANDS FOR RETRIEVING ADDRESS SPACE INFORMATION	20
2.14. REPLACING ADDRESS SPACES USING THE COMMAND LINE	20
CHAPTER 3. MANAGING ADDRESSES	22
3.1. ADDRESS	22

3.2. ADDRESS PLANS	22
3.2.1. Address example	22
3.3. LISTING AVAILABLE ADDRESS PLANS USING THE COMMAND LINE	22
3.4. CREATING ADDRESSES USING THE COMMAND LINE	23
3.5. CREATING ADDRESSES USING THE RED HAT AMQ CONSOLE	23
3.6. REPLACING ADDRESSES USING THE COMMAND LINE	24
CHAPTER 4. USING THE RED HAT AMQ CONSOLE	25
4.1. RED HAT AMQ CONSOLE USER PERMISSIONS	25
4.2. ACCESSING THE RED HAT AMQ CONSOLE	25
4.3. USING THE RED HAT AMQ CONSOLE ADDRESS FILTERING	26
4.4. VIEWING MESSAGE AND CONNECTION STATISTICS USING THE RED HAT AMQ CONSOLE	26
CHAPTER 5. USER MODEL	28
5.1. AUTHENTICATION	28
5.1.1. Password authentication type	28
5.1.2. Serviceaccount authentication type	29
5.2. AUTHORIZATION	29
5.3. MANAGING USERS	29
5.3.1. Creating users using the command line	29
5.3.2. Deleting users using the command line	30
5.3.3. Managing user permissions using the command line	30
CHAPTER 6. CONNECTING APPLICATIONS TO AMQ ONLINE	32
6.1. CLIENT EXAMPLES	32
6.1.1. AMQ Online Python example	32
6.1.1.1. Known issue with creating a subscriber on a hierarchical topic	32
6.1.2. AMQ Online JMS example	33
6.1.3. AMQ Online JavaScript example	34
6.1.3.1. AMQ Online JavaScript example using WebSockets	35
6.1.4. AMQ Online C++ example	35
6.1.4.1. Known issue with creating a subscriber on a hierarchical topic	36
6.1.5. AMQ Online .NET example	37
APPENDIX A. USING YOUR SUBSCRIPTION	38
Accessing your account	38
Activating a subscription	38
Downloading zip and tar files	38
Registering your system for packages	38
APPENDIX B. AMQ ONLINE RESOURCES FOR MESSAGING TENANTS	39

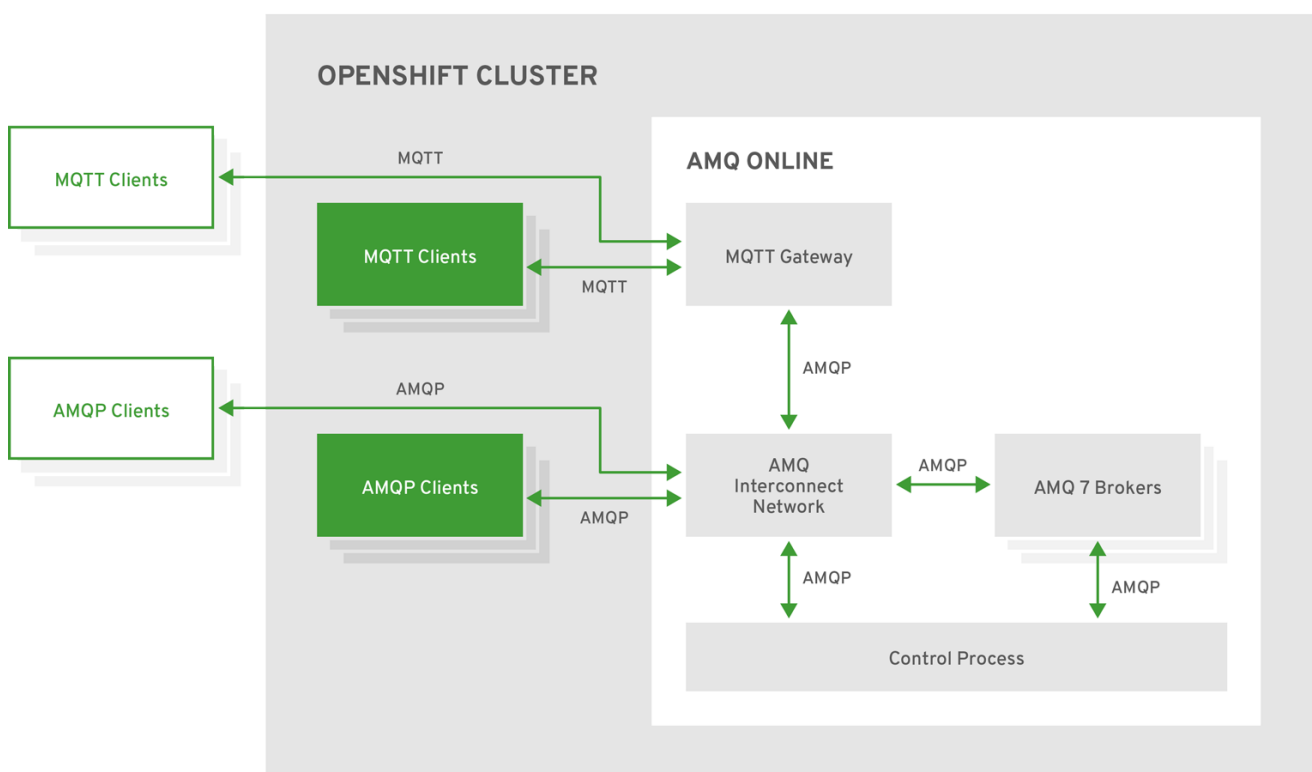
CHAPTER 1. INTRODUCTION

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 Red Hat AMQ 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.

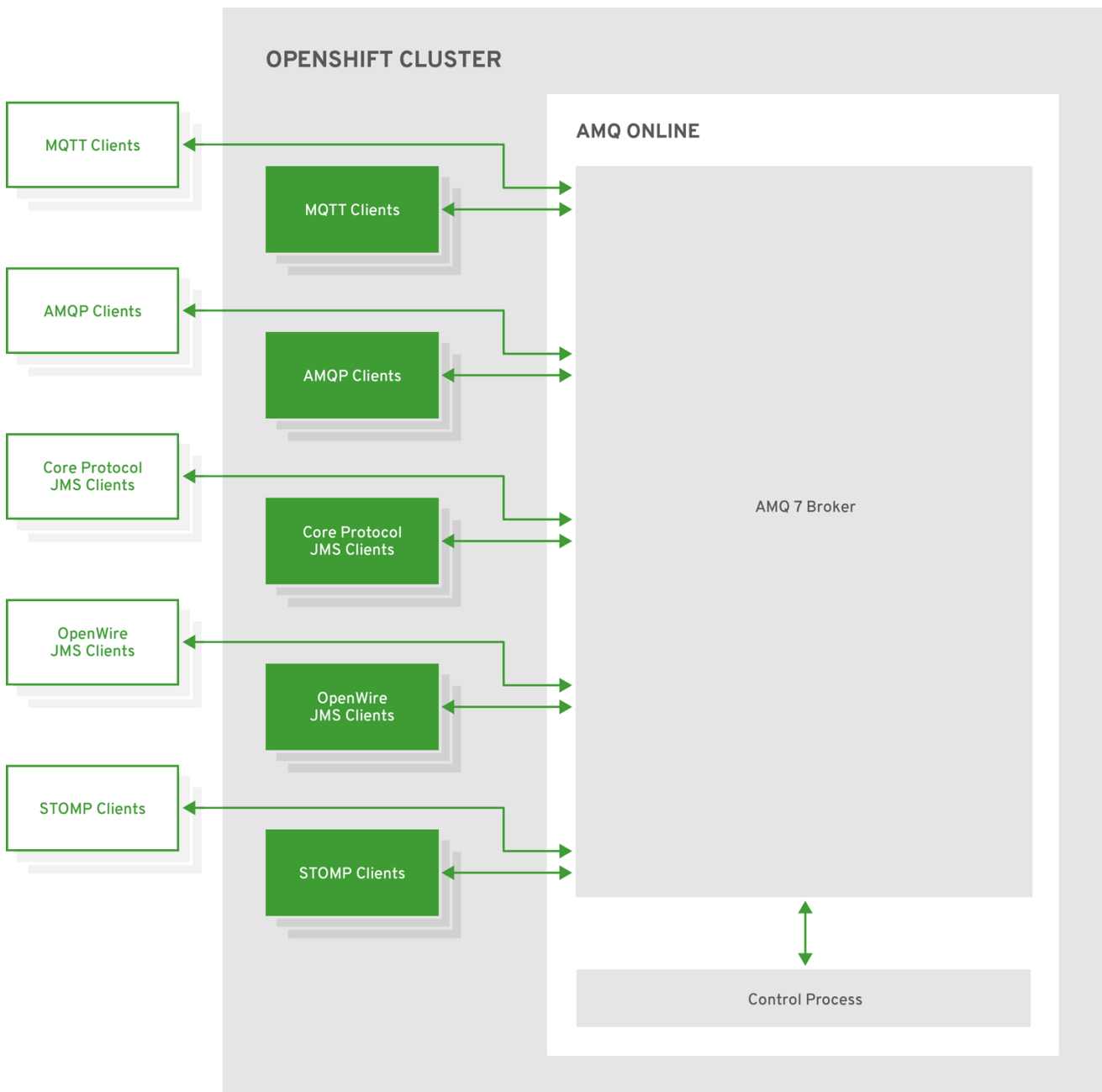
AMQ Online can provision different types of messaging depending on your use case. A user can request messaging resources by creating an address space. AMQ Online currently supports two address space types, standard and brokered, each with different semantics. The following diagrams illustrate the high-level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

Figure 1.1. Standard address space



AMQ_483683_0119

Figure 1.2. Brokered address space



AMQ_483683_0119

1.2. SUPPORTED FEATURES

The following table shows the supported features for AMQ Online 1.1:

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. SUPPORTED CONFIGURATIONS

For more information about AMQ Online supported configurations see [Red Hat AMQ 7 Supported Configurations](#).

1.5. DOCUMENT CONVENTIONS

1.5.1. 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:

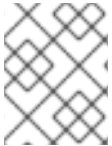
- queue
- topic
- anycast
- multicast
- subscription

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.

2.2.1.2.2. Known issue with creating a subscriber on a hierarchical topic

A known issue exists where creating a subscriber on a hierarchical topic in AMQ Online causes the broker to instead create it as a competing consumer (handling the address like a queue rather than a topic). For more information about the specific workaround for your client, see the applicable client example section in [Connecting applications to AMQ Online](#).

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**

2.3.1.2.2. Known issue with creating a subscriber on a hierarchical topic

A known issue exists where creating a subscriber on a hierarchical topic in AMQ Online causes the broker to instead create it as a competing consumer (handling the address like a queue rather than a topic). For more information about the specific workaround for your client, see the applicable client example section in [Connecting applications to AMQ Online](#).

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 ADDRESS SPACE PLANS USING THE COMMAND LINE

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

Procedure

1. Log in as a messaging tenant:

```
oc login -u developer
```

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

```
oc get addressspaceschema standard -o jsonpath='{.spec.plans[*].name}'
```

2.6. LISTING AVAILABLE AUTHENTICATION SERVICES USING THE COMMAND LINE

You can list the authentication services available for your address space type.

Procedure

1. Log in as a messaging tenant:

```
oc login -u developer
```

2. Retrieve the schema with the authentication services listed (replace **standard** with **brokered** for the brokered address space type):

```
oc get addressspaceschema standard -o jsonpath='{.spec.authenticationServices}'
```

2.7. ADDRESS SPACE EXAMPLES

2.7.1. 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 ❶
  plan: standard-unlimited ❷
```

- ❶ The address space type can be either **brokered** or **standard**.
- ❷ The address space plan depends on the address space type and what has been configured by the AMQ Online administrator. To view your available address space plans, see [Listing available address space plans](#).

2.7.2. Address space example using an authentication service

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

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

- ❶ The authentication service name depends on the available authentication services configured by the AMQ Online administrator. To view the available authentication services for your address space type, see [Listing available authentication services](#).

2.7.3. Address space example exposing endpoints externally

This address space example shows how you can configure the external endpoints of an **AddressSpace** to access messaging endpoints outside the OpenShift cluster.

2.7.3.1. OpenShift route example

To expose **AddressSpace** endpoints through OpenShift **LoadBalancer** services, the **loadbalancer** type is used:

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
```



```

metadata:
  name: myspace
spec:
  type: standard
  plan: standard-unlimited
  authenticationService:
    name: standard-authservice
  endpoints:
    - name: messaging 1
      service: messaging 2
      expose:
        type: loadbalancer 3
        loadBalancerPorts: 4
          - amqp
          - amqps
      annotations: 5
        mykey: myvalue
      loadBalancerSourceRanges: 6
        - 10.0.0.0/8

```

- 1** (Required) The name of the endpoint. The name specified affects the name of the OpenShift service to be created as well as the name of the endpoint in the status section of the **AddressSpace**.
- 2** (Required) The service names that must be configured for the **AddressSpace**. The service can either be **messaging**, **console**, or **mqtt**. However, the **mqtt** service is supported for the **standard** address space type only.
- 3** (Required) The type of endpoint being exposed. The **loadbalancer** type will create an {Kubernetes} **LoadBalancer** service. Valid values are **route** and **loadbalancer**.
- 4** (Required) A list of the ports to be exposed on the **LoadBalancer** service. For **messaging**, **amqp** and **amqps** can be specified.
- 5** (Optional) A set of key-value annotation pairs that are added to the **LoadBalancer Service** object.
- 6** (Optional) The allowed source ranges that are accepted by the load balancer.

2.7.3.2. OpenShift LoadBalancer service example

To expose **AddressSpace** endpoints as OpenShift routes, the **route** type is used:

```

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

```

```

service: messaging 2
expose:
  type: route
  routeServicePort: amqps 3
  routeTlsTermination: passthrough 4
  routeHost: messaging.example.com 5

```

- 1 (Required) The name of the endpoint. The name specified affects the name of the OpenShift service to be created as well as the name of the endpoint in the status section of the **AddressSpace**.
- 2 (Required) The service names that must be configured for the **AddressSpace**. The service can either be **messaging**, **console**, or **mqtt**. However, the **mqtt** service is supported for the **standard** address space type only.
- 3 (Required) The name of the port to be exposed. With the **route** type, only a single TLS-enabled port can be specified. For the **messaging** service, either **amqps** and **https** can be specified. For the **console** service, only **https** can be specified.
- 4 (Required) The TLS termination policy to be used for the OpenShift route. For the **messaging** service, the **amqps** port requires **passthrough** to be used, whereas **https** (websockets) also allows **reencrypt**. For the **console** service, **reencrypt** can be used.
- 5 (Optional) The hostname to use for the created route.

2.7.4. Configuring an address space certificate provider

This address space example shows how you can configure the endpoints of an **AddressSpace** to be configured using different certificate providers. The certificate provider determines how certificates are issued for the endpoints of an **AddressSpace**.

2.7.4.1. openshift provider

The **openshift** certificate provider can be used to configure endpoints with certificates signed by the OpenShift Cluster CA.

```

apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard
  plan: standard-unlimited
  authenticationService:
    name: standard-authservice
  endpoints:
    - name: messaging
      service: messaging
  cert:
    provider: openshift 1

```

- 1 (Required) The certificate provider type. The available types are **openshift** (on OpenShift only), **certBundle**, and **selfsigned** (default).

2.7.4.2. selfsigned provider

The **selfsigned** certificate provider can be used to configure endpoints with self-signed certificates. The CA for these certificates can be found in the **status.caCert** field of the **AddressSpace** resource.

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard
  plan: standard-unlimited
  authenticationService:
    name: standard-authservice
  endpoints:
    - name: messaging
      service: messaging
  cert:
    provider: selfsigned 1
```

- 1 (Required) The certificate provider type. The available types are **openshift** (on OpenShift only), **certBundle**, and **selfsigned** (default).

2.7.4.3. certBundle provider

The **certBundle** certificate provider can be used to configure endpoints with user-supplied certificates signed by your own CA. Certificate rotation can be done by updating the **tlsKey** and **tlsCert** fields with updated certificates, and then updating the **AddressSpace** resource.

```
apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard
  plan: standard-unlimited
  authenticationService:
    name: standard-authservice
  endpoints:
    - name: messaging
      service: messaging
  cert:
    provider: certBundle 1
    tlsKey: Y2VydGJ1bmRsZXByb3ZpZGVyY2VydA== 2
    tlsCert: Y2VydGJ1bmRsZXByb3ZpZGVyY2VydA== 3
```

- 1 (Required) The certificate provider type. The available types are **openshift** (on OpenShift only), **certBundle**, and **selfsigned** (default).

- 2 (Required) The base64-encoded value of the PEM private key (including the preamble)

- 3 (Required) The base64-encoded value of the PEM certificate (including the preamble)

2.7.5. Address space example exports

You can export your address space information using the following three export types:

- **ConfigMap**
- **Secret**
- **Service**

2.7.5.1. ConfigMap and Secret type export examples

This example shows the format used by the **ConfigMap** export type. The format of the **Secret** export type uses the same keys as the **ConfigMap** export type, but the values are Base64-encoded.

```
service.host: messaging.svc
service.port.amqp: 5672
external.host: external.example.com
external.port: 5671
ca.crt: // PEM formatted CA
```

2.7.5.2. Service type export example

This example shows the format used by the **Service** export type.

```
externalName: messaging.svc
ports:
- name: amqp
  port: 5672
  protocol: TCP
  targetPort: 5672
```

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. EXAMPLE OF EXPORTING ADDRESS SPACE INFORMATION INTO THE APPLICATION NAMESPACE

This address space example shows how you can export the endpoint information of an **AddressSpace** resource to a **ConfigMap**, **Secret**, or **Service** in the same namespace as the messaging application.

```

apiVersion: enmasse.io/v1beta1
kind: AddressSpace
metadata:
  name: myspace
spec:
  type: standard
  plan: standard-unlimited
  authenticationService:
    name: standard-authservice
  endpoints:
    - name: messaging
      service: messaging
  exports:
    - kind: ConfigMap 1
      name: my-config 2

```

- 1 (Required) The type of export: **ConfigMap**, **Secret**, or **Service**. The resulting **ConfigMap** contains the values in the format shown in [example exports format](#). For **Secret**, the same keys are used, but the values are base64-encoded. For **Service**, a OpenShift service of the type **ExternalName** is created. This provides applications running on OpenShift with a way to inject endpoint information or provide a proxy service in the same namespace as the application. For more information see [example exports format](#).

- 2 (Required) The name of the resource to create and update.

When exporting endpoint information, the AMQ Online **address-space-controller** service account running in the AMQ Online namespace must be granted privileges to create, update, and delete the configmap specified in the exports list. You can do this by creating an RBAC role and role-binding such as this one:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: rbac
rules:
- apiGroups: [ "" ]
  resources: [ "configmaps" ]
  verbs: [ "create" ]
- apiGroups: [ "" ]
  resources: [ "configmaps" ]
  resourceName: [ "my-config" ]
  verbs: [ "get", "update", "patch" ]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: rbac-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: rbac
subjects:
- kind: ServiceAccount
  name: address-space-controller
  namespace: amq-online-infra
```

2.10. 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**.

2.11. CREATING AN ADDRESS SPACE USING THE RED HAT AMQ CONSOLE

You can create a new address space using the Red Hat AMQ Console.

Procedure

1. Log in to the Red Hat AMQ Console.
For more information about how to access the Red Hat AMQ Console, see [Accessing the Red Hat AMQ Console](#).
2. Click **Create**. The Create an instance wizard opens.
3. Complete the required fields and when you are finished, click **Finish** to create the new address space.

When the address space has been successfully created, you can click the address space name to go to the Red Hat AMQ Console and view information about the newly created address space.

2.12. DELETING AN ADDRESS SPACE USING THE RED HAT AMQ CONSOLE

You can delete an existing address space using the Red Hat AMQ Console.

Procedure

1. Log in to the Red Hat AMQ Console.
For more information about how to access the Red Hat AMQ Console, see [Accessing the Red Hat AMQ Console](#).
2. Locate the address space that you want to delete.
3. In the far right column, click the vertical ellipsis icon and select **Delete**. The delete confirmation window opens.
4. Confirm your selection by clicking **Delete**. The address space is deleted.

2.13. EXAMPLE COMMANDS FOR RETRIEVING ADDRESS SPACE INFORMATION

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

Table 2.1. Retrieving address space information commands table

To retrieve the...	Run this command:
Red Hat AMQ Console host name	oc get routes console -o jsonpath={.spec.host}
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.caCert}'
host name for the messaging endpoint	oc get addressspace myspace -o 'jsonpath={.status.endpointStatuses[?(@.name=="messaging")].externalHost}'

2.14. 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 Red Hat AMQ 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. For more information about how to view the available plans see [Listing available address plans](#).

3.3. LISTING AVAILABLE ADDRESS PLANS USING THE COMMAND LINE

You can list the address plans available for an address type, such as **queue**.

Procedure

1. Log in as a messaging tenant:

```
oc login -u developer
```

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

```
oc get addressspaceschema standard -o 'jsonpath={.spec.addressTypes[?(@.name=="queue")].plans[*].name}'
```

3.4. 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
```

3.5. CREATING ADDRESSES USING THE RED HAT AMQ CONSOLE

You can create new addresses using the Red Hat AMQ 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 Red Hat AMQ Console. For more information, see [Accessing the Red Hat AMQ Console](#).
2. Click the address space link for the address space where you want to create a new address.
3. Click **New**. The Create new address window opens.
4. 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.
5. Click **Next**.
6. Select a plan and click **Next**.
7. Click **Create**. Your address is displayed in the Red Hat AMQ Console.

3.6. 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 RED HAT AMQ CONSOLE

You can use the Red Hat AMQ Console to perform tasks such as [creating](#) and [deleting an address space](#), [creating an address](#), and [viewing message and connection statistics](#) .

4.1. RED HAT AMQ CONSOLE USER PERMISSIONS

Red Hat AMQ Console uses the OpenShift RBAC permissions model. For more information about the OpenShift RBAC permissions model, see the [OpenShift documentation](#).

To use Red Hat AMQ Console, the OpenShift user requires a role that grants access to **addressspace** and **address** resources. For example, for edit access, **edit** permissions need be to given to the associated role object, and for view-only access, **list** permissions need to be granted. For more information about the AMQ Online example roles, see [AMQ Online example roles](#) .

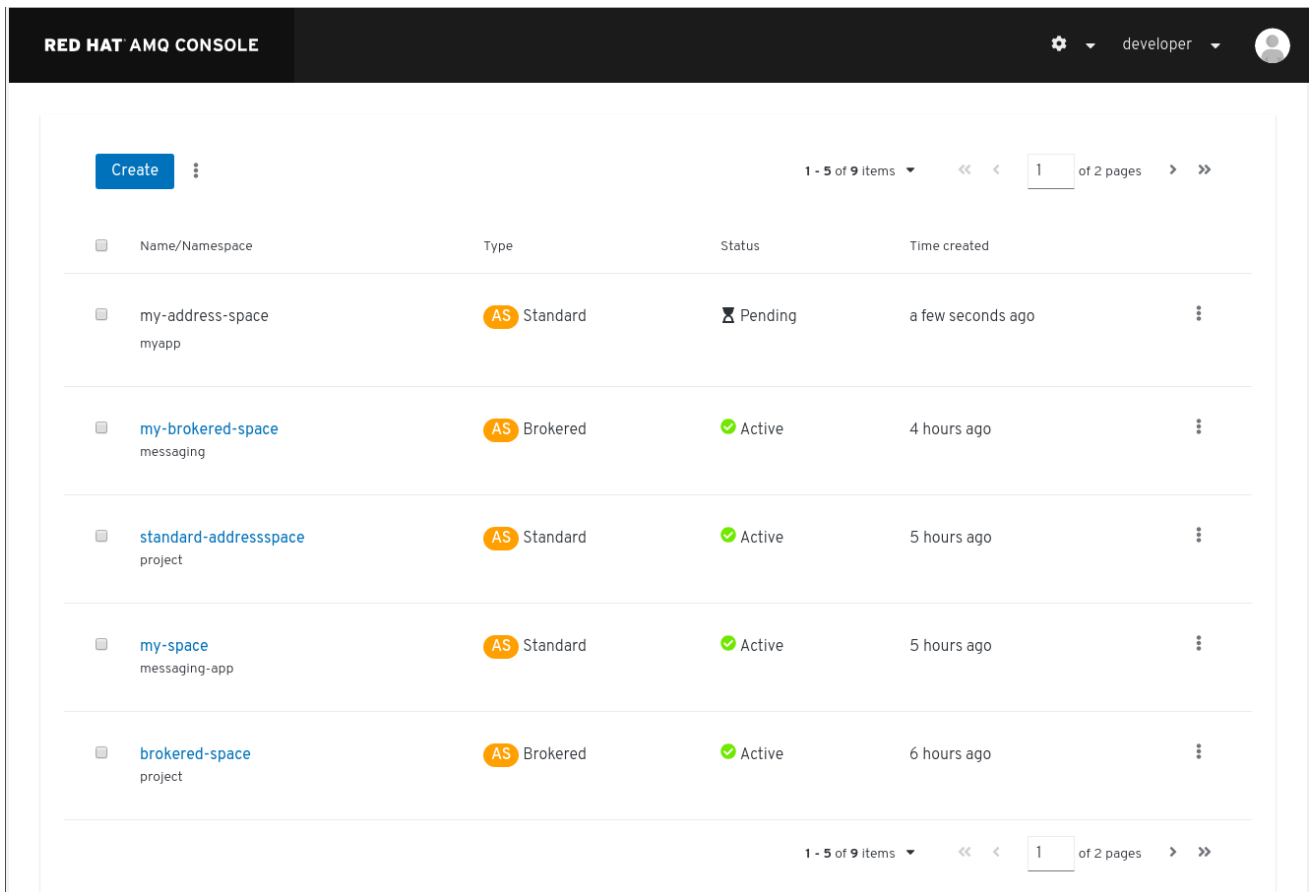
4.2. ACCESSING THE RED HAT AMQ CONSOLE

Prerequisites

- You must have obtained the host name for the Red Hat AMQ 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 Red Hat AMQ Console host name.
2. Log in with your OpenShift user credentials. The Red Hat AMQ Console opens.



4.3. USING THE RED HAT AMQ CONSOLE ADDRESS FILTERING

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

Table 4.1. Red Hat AMQ Console address name filtering behavior

To match...	Use...	Results in...
The beginning of an expression only	A caret followed by an expression: <code>^my</code>	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: <code>my\$</code>	All addresses ending with my
An exact expression	A caret followed by an expression and a dollar sign: <code>^my\$</code>	Only the address my

4.4. VIEWING MESSAGE AND CONNECTION STATISTICS USING THE RED HAT AMQ CONSOLE

Prerequisites

- You must be logged into the Red Hat AMQ 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
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

A messaging client connects using a `MessagingUser`. A `MessagingUser` specifies an authorization policy that controls which addresses may be used and the operations that may be performed on those addresses.

Users are configured as **`MessagingUser`** resources. Users can 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 **`serviceaccount`**. When using the **`password`** authentication type, you specify the username and password to be used by your messaging client when connecting. With the **`serviceaccount`** authentication type, you use the special string **`@@serviceaccount@@`** as the username, and an OpenShift service account token as the password.

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
bXktdGFzc3dvcmQ=
```


5.1.2. Serviceaccount authentication type

For the **serviceaccount** type, the **username** field must contain the OpenShift serviceaccount name that will be used to authenticate. When connecting with the messaging client, use the string **@@serviceaccount@@** as the username, and the service account token as the password.

5.2. AUTHORIZATION

In addition, authorization policies can be defined using operations and addresses. Valid operations are **send**, **rcv**, **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. To correctly base64 encode a password for the user definition file, run the following command:

```
echo -n password | base64 #cGFzc3dvcmQ=
```



NOTE

Be sure to use the **-n** parameter when running this command. Not specifying that parameter will result in an improperly coded password and cause log-in issues.

2. 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"]

```

3. Create the user and associated user permissions:

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

4. 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 > user-example1.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()
```

6.1.1.1. Known issue with creating a subscriber on a hierarchical topic

A known issue exists where creating a subscriber on a hierarchical topic in AMQ Online causes the broker to instead create it as a competing consumer (handling the address like a queue rather than a topic).

The workaround for this issue involves setting the capability **"topic"** in the source.

Procedure

1. In the **simple_recv.py** file, modify the **from proton.reactor import Container** to add the **ReceiverOption**:

```
class CapabilityOptions(ReceiverOption):
    def apply(self, receiver):
        receiver.source.capabilities.put_object(symbol("topic"))
```

1. Modify the following line to add **options=CapabilityOptions()**:

```
def on_start(self, event):
    event.container.create_receiver(conn, self.address, options=CapabilityOptions())
```

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(JMSEException exception) {
        System.out.println("Connection ExceptionListener fired, exiting.");
        exception.printStackTrace(System.out);
        System.exit(1);
    }
}
}
}

```

with jndi.properties:

```

connectionfactory.myFactoryLookup = amqs://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;
}

```

6.1.4.1. Known issue with creating a subscriber on a hierarchical topic

A known issue exists where creating a subscriber on a hierarchical topic in AMQ Online causes the broker to instead create it as a competing consumer (handling the address like a queue rather than a topic).

The workaround involves setting the capability **"topic"** in the source.

Procedure

- In the **topic_receive.cpp** file, edit the code so that it is similar to what is shown in this example:

```

void on_container_start(proton::container& cont) override {

```



```

proton::connection conn = cont.connect(conn_url_);
proton::receiver_options opts {};
proton::source_options sopts {};

sopts.capabilities(std::vector<proton::symbol> { "topic" });
opts.source(sopts);

conn.open_receiver(address_, opts);
}

```

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](#) .

APPENDIX B. AMQ ONLINE RESOURCES FOR MESSAGING TENANTS

The following table describes the AMQ Online resources that pertain to the messaging tenant role.

Table B.1. AMQ Online messaging tenant resources table

Resource	Description
addresses	Specifies the address.
addressspaces	Specifies the address space.
messagingusers	Specifies the authorization policy that controls which addresses may be used and the operations that may be performed on those addresses.

Revised on 2019-05-30 16:50:45 UTC