Red Hat AMQ 2021.Q3

在 OpenShift 上部署 AMQ Broker

用于 AMQ Broker 7.9

Last Updated: 2023-02-20

Red Hat AMQ 2021.Q3 在 OpenShift 上部署 AMQ Broker

用于 AMQ Broker 7.9

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

了解如何在 OpenShift Container Platform 上安装和部署 AMQ Broker。

目录

使开源包含更多

红帽承诺替换我们的代码、文档和网页属性中存在问题的语言。我们从这四个术语开始： master、slave、blacklist 和 whitelist。这些更改将在即将发行的几个发行本中逐渐实施。详情请查看 CTO Chris Wright 的信息。

第 1 章 OPENSHIFT CONTAINER PLATFORM 中的 AMQ BROKER 介绍

Red Hat AMQ Broker 7.9 作为容器化镜像提供，用于 OpenShift Container Platform(OCP)4.6. 4.7、4.8 或 4.9。

AMQ 代理基于 Apache ActiveMQ Artemis。它提供符合 JMS 规范的消息代理。设置初始代理 pod 后，您可以使用 OpenShift Container Platform 功能快速部署重复项。

1.1. 版本兼容性和支持

如需有关 OpenShift Container Platform 镜像版本兼容性的详细信息，请参阅：

- OpenShift Container Platform 4.x 测试的集成

注意

在 OpenShift Container Platform 上部署 AMQ Broker 现在都使用基于 RHEL 8 的镜像。

1.2. 不支持的功能

- 基于主从的高可用性
  不支持通过配置 master 和从对实现高可用性(HA)。相反，当容器集缩减时，OpenShift 中会使用 scaledown 控制器来提供 HA，这将启用消息迁移。

  可能需要为 HA 配置相应的外部客户端（通过 OpenShift 代理或使用绑定端口连接到代理集群）。在集群场景中，代理会告知特定客户端所有代理的主机和端口信息的地址。由于这些仅可在内部访问，因此某些客户端功能将不起作用或需要禁用。

客户端	Configuration
核心 JMS 客户端	由于外部核心协议 JMS 客户端不支持 HA 或任何类型的故障转移，因此连接工厂必须使用 **useTopologyForLoadBalancing=false** 配置连接工厂。
AMQP 客户端	AMQP 客户端不支持故障切换列表

- 集群中的持久订阅
  创建持久订阅时，这表示为客户端所连接的代理上的持久队列。当集群在 OpenShift 中运行时，客户端不知道已在哪个代理上创建了持久订阅队列。如果订阅持久且客户端重新连接，目前负载均衡器无法将其重新连接到同一节点。发生这种情况时，客户端可能连接到其他代理并创建重复的订阅队列。因此，不建议将持久订阅与代理集群搭配使用。

1.3. 文档惯例

本文档对 **sudo** 命令、文件路径和可替换值使用以下惯例：

sudo 命令
在本文档中，**sudo** 用于任何需要 root 特权的命令。使用 **sudo** 时，您应始终谨慎操作，因为任何更改都可能会影响整个系统。

有关使用 **sudo** 的更多信息，请参阅 **sudo** 命令。

关于在此文档中使用文件路径
在这个文档中，所有文件路径都对 Linux、UNIX 和类似操作系统（例如 **/home/...**）有效。如果您使用的是 Microsoft Windows，则应使用等效的 Microsoft Windows 路径（例如，**C:\Users\...**）。

可替换值
本文档有时会使用可替换值，您必须将这些值替换为特定于环境的值。可替换的值为小写，用方括号(**<>**)括起，样式则使用斜体和 **单空间** 字体。用下划线(_)**分隔多个词语。**

例如，在以下命令中，将 *<project_name>* 替换为您自己的项目名称。

```
$ oc new-project <project_name>
```

第 2 章 计划在 OPENSHIFT CONTAINER PLATFORM 上部署 AMQ BROKER

本节论述了如何规划基于 Operator 的部署。

操作器 是允许您打包、部署和管理 OpenShift 应用的程序。Operator 通常自动执行常见或复杂的任务。通常，Operator 旨在提供：

- 一致、可重复的安装
- 系统组件健康检查
- OTA(OTA)更新
- 受管升级

Operator 可让您在代理实例运行时进行更改，因为它们始终侦听用于配置部署的自定义资源(CR)实例的更改。当您更改 CR 时，Operator 会与现有代理部署协调更改，并更新部署来反映更改。另外，Operator 还提供消息迁移功能，可确保消息传递数据的完整性。当集群部署中的代理因为部署失败或有意缩减而关闭时，此功能会将信息迁移到仍在同一代理集群中运行的代理 Pod。

2.1. AMQ BROKER OPERATOR 自定义资源定义概述

通常，自定义资源定义(CRD)是配置项目的架构，您可以针对使用 Operator 部署的自定义 OpenShift 对象进行修改。通过创建对应的自定义资源(CR)实例，您可以在 CRD 中指定配置项目的值。如果您是 Operator 开发人员，您通过 CRD 公开的内容基本上会变成如何配置和使用已部署对象的 API。您可以通过常规 HTTP curl 命令直接访问 CRD，因为 CRD 通过 Kubernetes 自动公开。

您可以通过 OperatorHub 图形界面，使用 OpenShift 命令行界面(CLI)或 Operator Lifecycle Manager 来安装 AMQ Broker Operator。在这两种情况下，AMQ Broker Operator 都包含下面描述的 CRD。

主要代理 CRD

您可以基于此 CRD 部署 CR 实例，以创建并配置代理部署。

根据您安装 Operator 的方式，此 CRD 是：

- Operator 安装存档的 crd s 目录中的 broker_activemqartemis_ crd 文件（OpenShift CLI 安装方法）
- OpenShift Container Platform Web 控制台的 自定义资源定义 部分中的 ActiveMQArtemis CRD（OperatorHub 安装方法）

地址 CRD

您可以基于此 CRD 部署 CR 实例，为代理部署创建地址和队列。

根据您安装 Operator 的方式，此 CRD 是：

- Operator 安装存档的 crd s 目录中的 broker_activemqartemisaddress_ crd 文件（OpenShift CLI 安装方法）
- OpenShift Container Platform Web 控制台的 自定义资源定义 部分中的 ActiveMQArtemisAddress CRD（OperatorHub 安装方法）

安全 CRD

您根据此 CRD 部署 CR 实例，以创建用户并将这些用户与安全上下文相关联。

根据您安装 Operator 的方式，此 CRD 是：

- Operator 安装存档的 crds 目录中的 broker_activemqartemissecurity_crd 文件（OpenShift CLI 安装方法）
- OpenShift Container Platform Web 控制台的 Custom Resource Definitions 部分中的 ActiveMQArtemisSecurity CRD（OperatorHub 安装方法）。

scaleDown CRD

当实例化控制器进行消息迁移时，Operator 会自动 基于此 CRD 创建 CR 实例。

根据您安装 Operator 的方式，此 CRD 是：

- Operator 安装存档的 crd s 目录中的 broker_activemqartemisscaledown_ crd 文件（OpenShift CLI 安装方法）
- OpenShift Container Platform Web 控制台的 自定义资源定义 部分中的 ActiveMQArtemisScaledown CRD（OperatorHub 安装方法）。

其它资源

- 使用以下方法了解如何安装 AMQ Broker Operator（以及所有包括的 CRD）：
  - OpenShift CLI，请参阅 第 3.2 节 “使用 CLI 安装 Operator”
  - Operator Lifecycle Manager 和 OperatorHub 图形界面，请参阅 第 3.3 节 “使用 OperatorHub 安装 Operator”。
- 有关在基于主代理和地址 CRD 创建 CR 实例时使用的完整配置引用，请参阅：
  - 第 8.1.1 节 “代理自定义资源配置参考”
  - 第 8.1.2 节 “地址自定义资源配置参考”

2.2. AMQ BROKER OPERATOR 示例自定义资源概述

您在安装过程中下载和提取的 AMQ Broker Operator 存档包括 deploy/crs 目录中的示例自定义资源 (CR)文件。这些 CR 文件示例允许您：

- 部署不带 SSL 或群集的最小代理.
- 定义地址.

您下载并提取的 broker Operator 存档还包括 deploy/examples 目录中部署等 CR，如下所列。

artemis-basic-deployment.yaml

基本代理部署.

artemis-persistence-deployment.yaml

使用持久存储进行代理部署。

artemis-cluster-deployment.yaml

集群代理的部署.

artemis-persistence-cluster-deployment.yaml

使用持久性存储部署集群代理。

artemis-ssl-deployment.yaml

使用 SSL 安全性进行代理部署.

artemis-ssl-persistence-deployment.yaml

使用 SSL 安全性和持久存储进行代理部署.

artemis-aio-journal.yaml

将异步 I/O(AIO)与代理日志结合使用。

address-queue-create.yaml

地址和队列创建.

2.3. 监视 CLUSTER OPERATOR 部署的选项

当 Cluster Operator 运行时，它开始 *监视* AMQ Broker 自定义资源(CR)的更新。

您可以选择部署 Cluster Operator 来监视 CR:

- 单个命名空间（包含 Operator 的同一命名空间）

- 所有命名空间

注意

如果您在集群中的命名空间中安装了 AMQ Broker Operator 的早期版本，红帽建议您不要安装 AMQ Broker Operator 7.9 版本来监视该命名空间以避免潜在的冲突。

2.4. OPERATOR 如何选择容器镜像

当您根据 Operator *的最低版本 7.9.3-opr-3 为代理部署创建自定义资源(CR)实例时，您不需要 在 CR 中明确指定代理或 Init 容器镜像名称。默认情况下，如果您部署 CR 但没有显式指定容器镜像值，Operator 会自动选择要使用的适当容器镜像。*

注意

如果使用 OpenShift 命令行界面安装 Operator，Operator 安装存档包含一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件。示例 CR 中包含 spec.deploymentPlan.image 属性，并将其设置为其 占位符 默认值。此值表示 Operator 在部署 CR 之前不会选择代理容器镜像。

指定初始容器镜像的 spec.deploymentPlan.initImage 属性 不包括在 broker_activemqartemis_cr.yaml 示例 CR 文件中。如果您没有在 CR 中明确包含 spec.deploymentPlan.initImage 属性并指定了一个值，Operator 会选择部署 CR 时使用的适当内置 Init 容器镜像。

本节介绍了 Operator 如何选择这些镜像。

要选择代理和初始容器镜像，Operator 首先决定镜像应对应的 AMQ Broker 版本。Operator 决定以下版本：

- *如果主 CR 中的 spec.upgrades.enabled 属性已设置为 true，spec .version 属性指定 7.7.0、78.0、7.8. 1 或 7.8.2，Operator 会使用该指定版本。*
- *如果 spec.upgrades.enabled 没有设置为 true，或者将 spec.version 设置为早于 7.7.0 的 AMQ Broker 版本，Operator 将使用 最新版本的 AMQ Broker（即 7.9.3）。*

然后，Operator 会检测容器平台。AMQ Broker Operator 可以在以下容器平台中运行：

- *OpenShift Container Platform (x86_64)*
- *IBM Z 上的 OpenShift Container Platform(s390x)*
- *IBM Power Systems 上的 OpenShift Container Platform(ppc64le)*

根据 AMQ Broker 和容器平台的版本，Operator 会在 operator.yaml 配置文件中引用两组环境变量。这些环境变量集为不同版本的 AMQ Broker 指定代理和 Init 容器镜像，如下小节所述。

2.4.1. 代理容器镜像的环境变量

代理容器镜像的 operator.yaml 配置文件中包含的环境变量具有以下命名约定：

OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>

IBM Z 上的 OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>_s390x

IBM Power 系统上的 OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>_ppc64le

表中显示了每个支持的容器平台和特定 AMQ Broker 版本的环境变量名称。

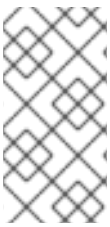
容器平台	环境变量名称

容器平台	环境变量名称
OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_790**
IBM Z 上的 OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781_s390x** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_782_s390x** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_790_s390x**
IBM Power 系统上的 OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781_ppc64le** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_782_ppc64le** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_790_ppc64le**

每个环境变量的值指定红帽提供的代理容器镜像。例如：

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_790
  #value: registry.redhat.io/amq7/amq-broker-rhel8:7.9
  value: registry.redhat.io/amq7/amq-broker-rhel8@sha256:979b59325aa0f34eb05625201beba53fccbb83bd5eb80a89dcb5261ae358138f
```

因此，Operator 根据 AMQ Broker 版本和容器平台决定适用的环境变量名称。在启动代理容器时，Operator 会使用对应的镜像值。



注意

在 operator.yaml 文件中，Operator 使用由 Secure Hash Algorithm (SHA)值表示的镜像。注释行以数字符号(#)符号开头，表示 SHA 值与特定的容器镜像标签对应。

2.4.2. Init 容器镜像的环境变量

Init 容器镜像的 operator.yaml 配置文件中包含的环境变量有以下命名约定：

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_<AMQ_Broker_version_identifier>

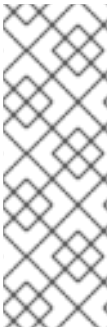
下面列出了特定 AMQ Broker 版本的环境变量名称。

- *RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_781*
- *RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_782*
- *RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_790*

每个环境变量的值指定红帽提供的 Init 容器镜像。例如：

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_790
  #value: registry.redhat.io/amq7/amq-broker-init-rhel8:0.4-17
  value: registry.redhat.io/amq7/amq-broker-init-
rhel8@sha256:b74d03ed852a3731467ffda95266ce49f2065972f1c37bf254f3d52b34c11991
```

因此，Operator 根据 AMQ Broker 版本决定适用的环境变量名称。在启动初始容器时，Operator 会使用对应的镜像值。



注意

如示例所示，Operator 使用由 Secure Hash Algorithm (SHA)值表示的镜像。注释行以数字符号(#)符号开头，表示 SHA 值与特定的容器镜像标签对应。观察对应的容器镜像标签 不是一个 floating 标签，格式为 0.4-17。这意味着 Operator 使用的容器镜像仍然被修复。当 Operator 可用时，它不会自动 拉取和使用新的 微版本 （即 0.4-17-n，其中 n 是最新的微版本）。

Init 容器镜像的 operator.yaml 配置文件中包含的环境变量有以下命名约定：

OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_<AMQ_Broker_version_identifier>

IBM Z 上的 OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_<AMQ_Broker_version_identifier>

IBM Power 系统上的 OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_<AMQ_Broker_version_identifier>

表中显示了每个支持的容器平台和特定 AMQ Broker 版本的环境变量名称。

容器平台	环境变量名称
OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_790**
IBM Z 上的 OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_790**
IBM Power 系统上的 OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_790**

每个环境变量的值指定红帽提供的 Init 容器镜像。例如：

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_790
  #value: registry.redhat.io/amq7/amq-broker-init-rhel8:0.4-17-1
```

```
value: registry.redhat.io/amq7/amq-broker-init-
rhel8@sha256:b74d03ed852a3731467ffda95266ce49f2065972f1c37bf254f3d52b34c11991
```

因此，Operator 根据 AMQ Broker 版本和容器平台决定适用的环境变量名称。在启动初始容器时，Operator 会使用对应的镜像值。

注意

如示例所示，Operator 使用由 Secure Hash Algorithm (SHA)值表示的镜像。注释行以数字符号(#)符号开头，表示 SHA 值与特定的容器镜像标签对应。观察对应的容器镜像标签 不是一个 floating 标签，格式为 0.4-17。这意味着 Operator 使用的容器镜像仍然被修复。当 Operator 可用时，它不会自动 拉取和使用新的 微版本 （即 0.4-17-n，其中 n 是最新的微版本）。

其它资源

- 要了解如何使用 AMQ Broker Operator 创建代理部署，请参阅 第 3 章 使用 AMQ Broker Operator 在 OpenShift Container Platform 上部署 AMQ Broker。
- 如需有关 Operator 如何使用初始容器生成代理配置的更多信息，请参阅 第 4.1 节 “Operator 如何生成代理配置”。
- 要了解如何构建和指定 自定义 初始容器镜像，请参阅 第 4.6 节 “指定自定义初始容器镜像”。

2.5. OPERATOR 部署备注

本节论述了规划基于 Operator 的部署时的一些重要注意事项

- 部署 AMQ Broker Operator 附带的自定义资源定义(CRD)需要 OpenShift 集群的集群管理员特权。部署 Operator 时，非管理员用户可以通过对应的自定义资源(CR)创建代理实例。要让常规用户部署 CR，集群管理员必须首先为 CRD 分配角色和权限。如需更多信息，请参阅 OpenShift Container Platform 文档中的 为自定义资源定义创建集群角色。
- 当您使用最新 Operator 版本的 CRD 更新集群时，此更新会影响 集群中的所有 项目。从以前版本的 Operator 中部署的所有代理 Pod 都可能无法更新其状态。当您点击 OpenShift Container Platform Web 控制台中正在运行的代理 Pod 的 Logs 选项卡时，您会看到指出"UpdatePodStatus"失败的信息。但是，该项目中的 broker Pod 和 Operator 会象预期一样工作。要修复受影响的项目的此问题，还必须将该项目升级为使用最新版本的 Operator。

- 虽然您可以通过部署多个自定义资源(CR)实例，在给定的 OpenShift 项目中创建多个代理部署，但通常在项目中创建单个代理部署，然后为地址部署多个 CR 实例。

  红帽建议在不同的项目中创建代理部署。

- 如果要使用持久性存储部署代理，且在 OpenShift 集群中没有容器原生虚拟化存储，则需要手动置备持久性卷(PV)，并确保 Operator 可以声明这些卷可用。例如，如果要创建一个包含两个代理的集群，且带有持久性存储（也就是说，在 CR 中设置 persistenceEnabled=true），则需要有两个持久性卷可用。默认情况下，每个代理实例都需要存储 2 GiB。

  如果您在 CR 中指定了 persistenceEnabled=false，部署的代理会使用 临时存储。临时存储意味着，每次重启代理 Pod 时，任何现有数据都会丢失。

  有关在 OpenShift Container Platform 中置备持久性存储的更多信息，请参阅：

  - 了解持久性存储 (OpenShift Container Platform 4.5)

- 在 首次部署 CR 前，您必须将以下列出的项目的配置添加到主代理 CR 实例中。您无法 将这些项目的配置添加到已在运行的代理部署中。

  - 持久性存储部署中每个代理所需的持久性卷声明(PVC)的大小
  - 部署中每个代理的内存和 CPU 限值和请求

下一节中的步骤演示了如何安装 Operator，并使用自定义资源(CR)在 OpenShift Container Platform 上创建代理部署。成功完成这些步骤后，您将让 Operator 在单个 Pod 中运行。您创建的每个代理实例都将在与 Operator 相同的项目中作为 StatefulSet 的独立 Pod 运行。稍后，您将了解如何使用专用寻址 CR 在代理部署中定义地址。

第 3 章 使用 AMQ BROKER OPERATOR 在 OPENSHIFT CONTAINER PLATFORM 上部署 AMQ BROKER

3.1. 先决条件

- 在安装 Operator 并使用它创建代理部署前，您应该查阅 第 2.5 节 “Operator 部署备注” 中的 Operator 部署说明。

3.2. 使用 CLI 安装 OPERATOR

注意

每个 Operator 版本都需要您下载最新的 AMQ Broker 7.9.3 Operator 安装和示例文件，如下所述。

本节中的步骤演示了如何使用 OpenShift 命令行界面(CLI)在给定 OpenShift 项目中为 AMQ Broker 7.9 安装和部署 Operator 的最新版本。在后续流程中，您将使用此 Operator 来部署一些代理实例。

- 有关安装使用 OperatorHub 图形界面的 AMQ Broker Operator 的替代方法，请参阅 第 3.3 节 “使用 OperatorHub 安装 Operator”。

- 要了解有关 升级 基于 Operator 的现有代理部署的信息，请参阅 第 6 章 升级基于 Operator 的代理部署。

3.2.1. 获取 Operator 代码

此流程演示了如何访问和准备为 AMQ Broker 7.9 安装最新版本的 Operator 所需的代码。

流程

1. 在网页浏览器中，导航到 AMQ Broker 7.9.3 的 Software Downloads 页面。

2. 确保 Version 下拉列表的值已设置为 7.9.3，并且选择了 Releases 选项卡。

3. 在 AMQ Broker 7.9.3 Operator 安装和示例文件旁边，点 Download。

   下载 amq-broker-operator-7.9.3-ocp-install-examples.zip 压缩存档会自动开始。

4. 下载完成后，将存档移动到您选择的安装目录中。以下示例将存档移至名为 ~/broker/operator 的目录。

```
$ mkdir ~/broker/operator
$ mv amq-broker-operator-7.9.3-ocp-install-examples.zip ~/broker/operator
```

5. 在您选择的安装目录中，提取 存档的内容。例如：

```
$ cd ~/broker/operator
$ unzip amq-broker-operator-7.9.3-ocp-install-examples.zip
```

6. 切换到您在提取存档时创建的目录。例如：

```
$ cd amq-broker-operator-7.9.3-ocp-install-examples
```

7. 以集群管理员身份登录 OpenShift Container Platform。例如：

```
$ oc login -u system:admin
```

8. 指定您要安装 Operator 的项目。您可以创建新项目或切换到现有项目。

   a. 创建一个新项目

   ```
   $ oc new-project <project_name>
   ```

   b. 或者，切换到现有项目：

   ```
   $ oc project <project_name>
   ```

9. 指定用于 Operator 的服务帐户。

a. 在您提取的 Operator 存档的 deploy 目录中，打开 service_account.yaml 文件。

b. 确保 kind 元素设置为 ServiceAccount。

c. 在 metadata 部分中，为服务帐户分配自定义名称，或者使用默认名称。默认名称为 amq-broker-operator。

d. 在项目中创建服务帐户。

```
$ oc create -f deploy/service_account.yaml
```

10. 为 Operator 指定角色名称。

a. 打开 role.yaml 文件。此文件指定 Operator 可以使用和修改的资源。

b. 确保 kind 元素设置为 Role。

c. 在 metadata 部分中，为角色分配自定义名称，或者使用默认名称。默认名称为 amq-broker-operator。

d. 在项目中创建角色。

```
$ oc create -f deploy/role.yaml
```

11. 为 Operator 指定角色绑定。角色绑定根据您指定的名称，将之前创建的服务帐户绑定到 Operator 角色。

a. 打开 role_binding.yaml 文件。确保 ServiceAccount 和 Role 的名称 值与 service_account.yaml 和 role.yaml 文件中指定的值匹配。例如：

```
metadata:
  name: amq-broker-operator
subjects:
  kind: ServiceAccount
```

```
    name: amq-broker-operator
roleRef:
    kind: Role
    name: amq-broker-operator
```

b. 在项目中创建角色绑定。

```
$ oc create -f deploy/role_binding.yaml
```

在以下流程中，您将在项目中部署 Operator。

3.2.2. 使用 CLI 部署 Operator

本节中的步骤演示了如何使用 OpenShift 命令行界面(CLI)在 OpenShift 项目中部署 AMQ Broker 7.9 的最新版本 Operator。

先决条件

- 您必须已经为 Operator 部署准备了 OpenShift 项目。请参阅 第 3.2.1 节 “获取 Operator 代码”。
- 从 AMQ Broker 7.3 开始，您可以使用新版本的红帽生态系统目录来访问容器镜像。这个新版本的 registry 需要您成为经过身份验证的用户，然后才能访问镜像。在按照本节中的步骤前，您必须先完成 Red Hat Container Registry Authentication 中描述的步骤。
- 如果要使用持久性存储部署代理，且在 OpenShift 集群中没有容器原生虚拟化存储，则需要手动置备持久性卷(PV)，并确保 Operator 可以声明这些代理可用。例如，如果要创建一个包含两个带有持久性存储的代理的集群（也就是说，在自定义资源中设置 persistenceEnabled=true），则需要有两个可用的 PV。默认情况下，每个代理实例都需要存储 2 GiB。

  如果在自定义资源中指定了 persistent Enabled=false，部署的代理将使用 临时存储。临时存储意味着，每次重启代理 Pod 时，任何现有数据都会丢失。

  有关置备持久性存储的更多信息，请参阅：

  - 了解持久性存储 (OpenShift Container Platform 4.5)

流程

1. 在 OpenShift 命令行界面(CLI)中，以集群管理员身份登录 OpenShift。例如：

```
$ oc login -u system:admin
```

2. 切换到之前为 Operator 部署准备的项目。例如：

```
$ oc project <project_name>
```

3. 切换到之前提取 Operator 安装存档时创建的目录。例如：

```
$ cd ~/broker/operator/amq-broker-operator-7.9.3-ocp-install-examples
```

4. 部署 Operator 中包含的 CRD。在部署和启动 Operator 之前，您必须在 OpenShift 集群中安装 CRD。

   a. 部署主代理 CRD。

   ```
   $ oc create -f deploy/crds/broker_activemqartemis_crd.yaml
   ```

   b. 部署地址 CRD。

   ```
   $ oc create -f deploy/crds/broker_activemqartemisaddress_crd.yaml
   ```

   c. 部署 scaledown controller CRD。

   ```
   $ oc create -f deploy/crds/broker_activemqartemisscaledown_crd.yaml
   ```

5. 将与红帽生态系统目录中用于身份验证的帐户关联的 pull secret 与 OpenShift 项目 的默认、部署 者和 构建器 服务帐户链接。

```
$ oc secrets link --for=pull default <secret_name>
$ oc secrets link --for=pull deployer <secret_name>
$ oc secrets link --for=pull builder <secret_name>
```

6. 在您下载并提取的 Operator 存档的 deploy 目录中，打开 operator.yaml 文件。确保 spec.containers.image 属性的值对应于 Operator 版本 7.9.3-opr-3，如下所示。

```
spec:
  template:
    spec:
      containers:
        #image: registry.redhat.io/amq7/amq-broker-rhel8-operator:7.9
        image: registry.redhat.io/amq7/amq-broker-rhel8-
operator@sha256:851ae51685e535317486b899eb0f80c3c5236464ae35ef3f9cde740173f7
286b
```

注意

在 operator.yaml 文件中，Operator 使用由 Secure Hash Algorithm (SHA)值表示的镜像。注释行以数字符号(#)符号开头，表示 SHA 值与特定的容器镜像标签对应。

7. *通过选择性地编辑 operator.yaml 文件的 WATCH_NAMESPACE 部分，确定 Operator 会监视哪些命名空间。*

- *要部署 Operator 来监视活跃命名空间，请不要编辑以下部分：*

```
- name: WATCH_NAMESPACE
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
```

- *部署 Operator 以监视所有命名空间：*

```
- name: WATCH_NAMESPACE
  value: '*'
```

- *部署 Operator 以监视多个命名空间，如 namespace1 和 namespace2 ：*

```
- name: WATCH_NAMESPACE
  value: 'namespace1,namespace2'
```

注意

如果您之前使用 Operator 的早期版本部署代理，并且希望部署 Operator 以监视多个命名空间，请参阅 升级前。

8. 部署 Operator。

```
$ oc create -f deploy/operator.yaml
```

在 OpenShift 项目中，Operator 在新 Pod 中启动。

在 OpenShift Container Platform Web 控制台中，Operator Pod 的 Events 选项卡上的信息确认 OpenShift 已部署了您指定的 Operator 镜像，已将新容器分配给 OpenShift 集群中的节点，并启动新容器。

另外，如果您点击 Pod 中的 Logs 选项卡，输出应包含类似以下内容的行：

```
...
{"level":"info","ts":1553619035.8302743,"logger":"kubebuilder.controller","msg":"Starting
Controller","controller":"activemqartemisaddress-controller"}
{"level":"info","ts":1553619035.830541,"logger":"kubebuilder.controller","msg":"Starting
Controller","controller":"activemqartemis-controller"}
{"level":"info","ts":1553619035.9306898,"logger":"kubebuilder.controller","msg":"Starting
workers","controller":"activemqartemisaddress-controller","worker count":1}
{"level":"info","ts":1553619035.9311671,"logger":"kubebuilder.controller","msg":"Starting
workers","controller":"activemqartemis-controller","worker count":1}
```

以上输出确认新部署的 Operator 与 Kubernetes 通信，代理和寻址的控制器正在运行，并且这些控制器启动了一些 worker。

注意

建议您在给定的 OpenShift 项目中仅部署 AMQ Broker Operator 的一个实例。不建议将 Operator 部署的 spec.replicas 属性设置为大于 1 值，或者 在同一项目中多次部署 Operator。

其它资源

- 有关安装使用 OperatorHub 图形界面的 AMQ Broker Operator 的替代方法，请参阅 第 3.3 节 “使用 OperatorHub 安装 Operator”。

3.3. 使用 OPERATORHUB 安装 OPERATOR

3.3.1. Operator Lifecycle Manager 概述

在 OpenShift Container Platform 4.5 及更高版本中，Operator Lifecycle Manager (OLM)可帮助用户安装、更新并普遍管理所有 Operator 以及在用户集群中运行的关联服务的生命周期。Operator Framework 是 Operator Framework 的一部分，后者是一个开源工具包，旨在以有效、自动化且可扩展的方式管理 Kubernetes 原生应用程序(Operator)。

OLM 默认在 OpenShift Container Platform 4.5 及更新的版本中运行，辅助集群管理员对集群上运行的 Operator 进行安装、升级和授予访问权。OpenShift Container Platform Web 控制台提供一些管理界面，供集群管理员安装 Operator，以及为特定项目授权以便使用集群上的可用 Operator 目录。

OperatorHub 是 OpenShift 集群管理员使用 OLM 发现、安装和升级 Operator 的图形界面。只需单击一次，即可从 OperatorHub 中拉取这些 Operator，在集群中安装并由 OLM 管理，为工程团队在开发、测试和生产环境中管理软件做好准备。

部署 Operator 后，您可以使用自定义资源(CR)实例来创建代理部署，如独立和集群代理。

3.3.2. 从 OperatorHub 部署 Operator

此流程演示了如何使用 OperatorHub 将 AMQ Broker 的 Operator 的最新版本部署到指定的 OpenShift 项目。

重要

使用 OperatorHub 部署 Operator 需要集群管理员特权。

先决条件

- OperatorHub 必须提供 Red Hat Integration - AMQ Broker for RHEL 8(Multiarch) Operator。

流程

1. 以集群管理员身份登录 OpenShift Container Platform Web 控制台。
2. 在左侧导航菜单中点击 Operators → OperatorHub。

3. 在 OperatorHub 页面顶部的 Project 下拉菜单中选择您要部署 Operator 的项目。

4. 在 OperatorHub 页面中，使用 Filter by keyword... 找到 Red Hat Integration - AMQ Broker for RHEL 8(Multiarch)Operator 的方框。

注意

在 OperatorHub 中，在其名称中可能会找到多个 Operator，而不是包含 AMQ Broker。确保点击 Red Hat Integration - AMQ Broker for RHEL 8(Multiarch) Operator。点这个 Operator 时，请查看打开的信息窗格。对于 AMQ Broker 7.9，此 Operator 的最新次要版本标签为 7.9.3-opr-3。

5. 点 Red Hat Integration - AMQ Broker for RHEL 8(Multiarch) Operator。在出现的对话框中，单击 Install。

6. 在 Install Operator 页面中：

   a. 在 Update Channel 下，从以下单选按钮中选择 7.x 来指定用于跟踪和接收 Operator 更新的频道：

      - 7.X - 如果可用，此频道将更新至 7.10。
      - 7.8.X - 这是长期支持(LTS)通道。

   b. 在 Installation Mode 中，选择 Operator 监控的命名空间：

      - 集群上的特定命名空间 - Operator 安装在该命名空间中，仅监控该命名空间是否有 CR 更改。
      - 所有命名空间 - Operator 监控所有命名空间的 CR 更改。

注意

如果您之前使用 Operator 的早期版本部署代理，并且希望部署 Operator 以监视多个命名空间，请参阅 升级前。

7. 在 Installed Namespace 下拉菜单中选择您要安装 Operator 的项目。

8. 在 Approval Strategy 下，确保选择了标题为 Automatic 的单选按钮。这个选项指定对 Operator 的更新不需要手动批准才能进行安装。

9. 点 Install。

当 Operator 安装完成后，Installed Operators 页会打开。您应该会看到 Red Hat Integration - AMQ Broker for RHEL 8(Multiarch) Operator 已安装在您指定的项目命名空间中。

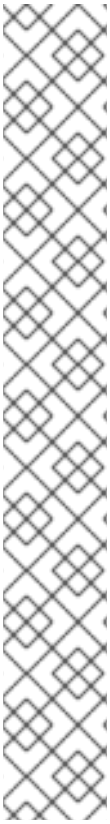
其它资源

- 要了解如何在安装了 AMQ Broker Operator 的项目中创建代理部署，请参阅 第 3.4.1 节 “部署基本代理实例”。

3.4. 创建基于 OPERATOR 的代理部署

3.4.1. 部署基本代理实例

以下流程演示了如何使用自定义资源(CR)实例来创建基本的代理部署。



注意

- 虽然您可以通过部署多个自定义资源(CR)实例，在给定的 OpenShift 项目中创建多个代理部署，但通常在项目中创建单个代理部署，然后为地址部署多个 CR 实例。

  红帽建议在不同的项目中创建代理部署。

- 在 AMQ Broker 7.9 中，如果要配置以下项目，您必须在首次部署 CR 前 将适当的配置添加到主代理 CR 实例中。

  - 持久性存储部署中每个代理所需的持久性卷声明(PVC)的大小
  - 部署中每个代理的内存和 CPU 限值和请求

先决条件

- 您必须已安装 AMQ Broker Operator。
  - 要使用 OpenShift 命令行界面(CLI)安装 AMQ Broker Operator，请参阅 第 3.2 节 “使用 CLI 安装 Operator”。
  - 要使用 OperatorHub 图形界面安装 AMQ Broker Operator，请参阅 第 3.3 节 “使用 OperatorHub 安装 Operator”。
- 您应该了解 Operator 如何选择用于代理部署的代理容器镜像。如需更多信息，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。
- 从 AMQ Broker 7.3 开始，您可以使用新版本的红帽生态系统目录来访问容器镜像。这个新版本的 registry 需要您成为经过身份验证的用户，然后才能访问镜像。在按照本节中的步骤前，您必须先完成 Red Hat Container Registry Authentication 中描述的步骤。

流程

成功安装 Operator 后，Operator 正在运行并侦听与 CR 相关的更改。本示例步骤演示了如何使用 CR 实例在项目中部署基本代理。

1. 为代理部署开始配置自定义资源(CR)实例。

   a. 使用 OpenShift 命令行界面：

      i. 以具有特权的用户身份登录 OpenShift，以在您要创建部署的项目中部署 CR。

      ```
      oc login -u <user> -p <password> --server=<host:port>
      ```

      ii. 打开名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载并提取的 Operator 安装存档的 deploy/crs 目录中。

   b. 使用 OpenShift Container Platform Web 控制台：

      i. 以具有特权的用户身份登录控制台，以在您要创建部署的项目中部署 CR。

      ii. 根据主代理 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 Administration → Custom Resource Definitions。

      iii. 单击 ActiveMQArtemis CRD。

      iv. 点 实例 选项卡。

      v. 单击 Create ActiveMQArtemis。

      在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

对于基本的代理部署，配置可能类似如下。此配置是 broker_activemqartemis_cr.yaml 示例 CR 文件的默认内容。

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
```

```
  application: ex-aao-app
spec:
  version: 7.9.3
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

请注意，在 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性被设置为默认值占位符。此值表示 image 属性默认不指定用于部署的代理容器镜像。要了解 Operator 如何决定要使用的适当代理容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。

注意

broker_activemqartemis_cr.yaml 示例 CR 使用 ex-aao 的命名约定。此命名规则表示 CR 是 AMQ Broker Operator 的示例资源。AMQ Broker 基于 ActiveMQ Artemis 项目。当您部署此示例 CR 时，生成的 StatefulSet 使用名称 ex-aao-s。另外，部署中的代理 Pod 直接基于 StatefulSet 名称，如 ex-aao-s-0、ex-aao-so-s-1 等。CR 中的应用程序名称作为 StatefulSet 上的标签出现在部署中。您可以在 Pod 选择器中使用此标签，例如：

2. size 属性指定要部署的代理数量。2 或以上值指定集群代理部署。但是，若要部署单个代理实例，请确保该值设置为 1。

3. 部署 CR 实例。

   a. 使用 OpenShift 命令行界面：

      i. 保存 CR 文件。

      ii. 切换到您要在其中创建代理部署的项目。

```
$ oc project <project_name>
```

      iii. 创建 CR 实例。

```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

b. **使用 OpenShift Web 控制台：**

i. **配置完 CR 后，点 Create。**

4. **在 OpenShift Container Platform Web 控制台中，单击 Workloads → StatefulSets。您会看到一个名为 ex-aao-s 的新 StatefulSet。**

a. **单击 ex-aao-ss StatefulSet。您会看到有一个 Pod，对应于您在 CR 中定义的单个代理。**

b. **在 StatefulSet 中，单击 Pods 选项卡。单击 ex-aao-ss Pod。在正在运行的 Pod 的 Events 选项卡中，您会看到代理容器已经启动。Logs 选项卡显示代理本身正在运行。**

5. **要测试代理是否正常运行，请访问代理 Pod 中的 shell 来发送一些测试信息。**

a. **使用 OpenShift Container Platform Web 控制台：**

i. **单击 Workloads → Pods。**

ii. **单击 ex-aao-ss Pod。**

iii. **点击 Terminal 选项卡。**

b. **使用 OpenShift 命令行界面：**

i. **获取项目的 Pod 名称和内部 IP 地址。**

```
$ oc get pods -o wide

NAME                    STATUS   IP
```

```
amq-broker-operator-54d996c   Running  10.129.2.14
ex-aao-ss-0                   Running  10.129.2.15
```

ii. 访问代理 Pod 的 shell。

```
$ oc rsh ex-aao-ss-0
```

6. 在 shell 中，使用 artemis 命令来发送一些测试消息。在 URL 中指定代理 Pod 的内部 IP 地址。例如：

```
sh-4.2$ ./amq-broker/bin/artemis producer --url tcp://10.129.2.15:61616 --destination
queue://demoQueue
```

上述命令会在代理上自动创建一个名为 demoQueue 的队列，并将默认数量的 1000 条消息发送到队列。

您应该看到类似如下的输出：

```
Connection brokerURL = tcp://10.129.2.15:61616
Producer ActiveMQQueue[demoQueue], thread=0 Started to calculate elapsed time ...

Producer ActiveMQQueue[demoQueue], thread=0 Produced: 1000 messages
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in second : 3 s
Producer ActiveMQQueue[demoQueue], thread=0 Elapsed time in milli second : 3492 milli
seconds
```

其它资源

- 有关主代理自定义资源(CR)的完整配置引用，请参阅 第 8.1 节 “自定义资源配置参考”。
- 要了解如何将正在运行的代理连接到 AMQ 管理控制台，请参阅 第 5 章 连接到基于 Operator 的代理部署的 AMQ 管理控制台。

3.4.2. 部署集群代理

如果项目中有两个或多个代理 Pod 运行，则 Pod 会自动组成代理集群。集群配置使代理能够互相连接，并根据需要重新分发消息，以实现负载平衡。

以下流程演示了如何部署集群代理。默认情况下，这个部署中的代理会 根据需求 负载平衡使用，这意

味着代理只会将消息转发到具有匹配消费者的其他代理。

先决条件

- *已部署了基本的代理实例。请参阅 第 3.4.1 节 “部署基本代理实例”。*

流程

1. *打开用于基本代理部署的 CR 文件。*

2. *对于集群部署，请确保 deploymentPlan.size 的值为 2 或更高。例如：*

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
        size: 4
        image: placeholder
        ...
```

注意

在 metadata 部分中，您需要包含 namespace 属性，只有在 使用 OpenShift Container Platform Web 控制台创建 CR 实例时才需要指定一个值。您应指定的值是代理部署的 OpenShift 项目的名称。

3. *保存修改后的 CR 文件。*

4. *以具有特权的用户身份登录 OpenShift，以在之前创建基本代理部署的项目中部署 CR。*

```
$ oc login -u <user> -p <password> --server=<host:port>
```

5. *切换到之前创建基本代理部署的项目。*

```
$ oc project <project_name>
```

6. 在命令行中应用更改：

```
$ oc apply -f <path/to/custom_resource_instance>.yaml
```

在 OpenShift Container Platform Web 控制台中，根据 CR 中指定的数量，在项目中启动其他代理 Pod。默认情况下，项目中运行的代理会被集群。

7. 打开每个 Pod 的 Logs 选项卡。日志显示 OpenShift 在各个代理上建立了集群连接网桥。特别是，日志输出包括以下行：

```
targetConnector=ServerLocatorImpl (identity=(Cluster-connection-
bridge::ClusterConnectionBridge@6f13fb88
```

3.4.3. 将自定义资源更改应用到运行代理部署

以下是将自定义资源(CR)更改应用到运行代理部署时需要注意的一些重要事项：

- 您无法动态更新 CR 中的 persistenceEnabled 属性。要更改此属性，请将集群缩减为零代理。删除现有的 CR。然后，重新创建并使用您的更改重新部署 CR，同时指定部署大小。
- CR 中的 deploymentPlan.size 属性的值覆盖您通过 oc scale 命令对代理部署大小所做的任何更改。例如，假设您使用 oc scale 将部署的大小从三个代理更改为两个，但 CR 中的 deploymentPlan.size 的值仍然是 3。在这种情况下，OpenShift 最初将部署缩减为两个代理。但是，当 scaledown 操作完成后，Operator 会根据 CR 中指定的将部署恢复到三个代理。
- 如 第 3.2.2 节 “使用 CLI 部署 Operator” 所述，如果您使用持久性存储创建代理部署（也就是说，通过在 CR 中设置 persistenceEnabled=true ），您可能需要为 AMQ Broker Operator 置备持久性卷(PV)来声明代理 Pod。如果缩小代理部署的大小，Operator 会释放之前为已关闭的代理 Pod 声明的所有 PV。但是，如果您通过删除 CR 来删除 代理部署，AMQ Broker Operator 不会在 删除时为仍在部署中的代理 Pod 发布持久性卷声明(PVC)。另外，这些未发布的 PV 可用于任何新部署。在这种情况下，您需要手动释放卷。如需更多信息，请参阅 OpenShift 文档中的发行持久性卷。
- 在 AMQ Broker 7.9 中，如果要配置以下项目，您必须在首次部署 CR 前 将适当的配置添加到主 CR 实例中。
  - 持久性存储部署中每个代理所需的持久性卷声明(PVC)的大小

- 部署中每个代理的内存和 CPU 限值和请求

- 在活跃的扩展事件中，您应用的任何进一步更改都将由 Operator 排队，并且仅在扩展完成后执行。例如，假设您将部署的大小从四个代理缩减到一个。然后，在进行缩减时，您还会更改代理管理员用户名和密码的值。在这种情况下，Operator 会把用户名和密码更改排队，直到部署通过一个活跃的代理运行。

- 除了更改部署的大小或更改接收器、连接器或控制台的 expose 属性值外，所有 CR 都会导致重启现有的代理。如果您在部署中有多个代理，一次只能重启一个代理。

第 4 章 配置基于 OPERATOR 的代理部署

4.1. OPERATOR 如何生成代理配置

在使用自定义资源(CR)实例来配置代理部署前，您应该了解 Operator 如何生成代理配置。

在创建基于 Operator 的代理部署时，每个代理的 Pod 在 OpenShift 项目中的 StatefulSet 中运行。代理的应用程序容器在每个 Pod 中运行。

在初始化每个 Pod 时，Operator 会运行一种名为 Init Container 的容器类型。在 OpenShift Container Platform 中，Init 容器是在应用程序容器之前运行的专用容器。初始容器可以包含应用程序镜像中不存在的实用程序或设置脚本。

默认情况下，AMQ Broker Operator 使用内置初始容器。Init 容器将主 CR 实例用于您的部署，以生成每个代理应用程序容器使用的配置。

如果您在 CR 中指定了地址设置，Operator 会生成默认配置，然后使用 CR 中指定的配置合并或替换该配置。后续小节中将描述此过程。

4.1.1. Operator 如何生成地址设置配置

如果您已在部署的主自定义资源(CR)实例中包含地址设置配置，Operator 会为每个代理生成地址设置配置，如下所述。

1. Operator 在代理应用程序容器之前运行初始容器。Init 容器生成默认的地址设置配置。默认地址设置配置如下所示：

```
<address-settings>
  <!--
  if you define auto-create on certain queues, management has to be auto-create
  -->
  <address-setting match="activemq.management#">
    <dead-letter-address>DLQ</dead-letter-address>
    <expiry-address>ExpiryQueue</expiry-address>
    <redelivery-delay>0</redelivery-delay>
    <!--
    with -1 only the global-max-size is in use for limiting
    -->
    <max-size-bytes>-1</max-size-bytes>
    <message-counter-history-day-limit>10</message-counter-history-day-limit>
```

```
        <address-full-policy>PAGE</address-full-policy>
        <auto-create-queues>true</auto-create-queues>
        <auto-create-addresses>true</auto-create-addresses>
        <auto-create-jms-queues>true</auto-create-jms-queues>
        <auto-create-jms-topics>true</auto-create-jms-topics>
      </address-setting>

      <!-- default for catch all -->
      <address-setting match="#">
        <dead-letter-address>DLQ</dead-letter-address>
        <expiry-address>ExpiryQueue</expiry-address>
        <redelivery-delay>0</redelivery-delay>
        <!--
        with -1 only the global-max-size is in use for limiting
        -->
        <max-size-bytes>-1</max-size-bytes>
        <message-counter-history-day-limit>10</message-counter-history-day-limit>
        <address-full-policy>PAGE</address-full-policy>
        <auto-create-queues>true</auto-create-queues>
        <auto-create-addresses>true</auto-create-addresses>
        <auto-create-jms-queues>true</auto-create-jms-queues>
        <auto-create-jms-topics>true</auto-create-jms-topics>
      </address-setting>
    <address-settings>
```

2. 如果您还在自定义资源(CR)实例中指定了地址设置配置，则 Init 容器会处理配置并将其转换为 XML。

3. 根据 CR 中 applyRule 属性的值，Init Container 会合并 或将上方显示 的默认地址设置配置替换为 您在 CR 中指定的配置。这个合并或替换的结果是代理将使用的最终地址设置配置。

4. 当初始容器完成生成代理配置（包括地址设置）后，代理应用程序容器会启动。启动时，broker 容器会从之前由初始容器使用的安装目录中复制其配置。您可以在 broker.xml 配置文件中检查地址设置配置。对于正在运行的代理，此文件位于 /home/jboss/amq-broker/etc 目录中。

其它资源

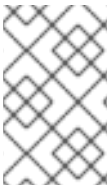
- 有关在 CR 中使用 applyRule 属性的示例，请参阅 第 4.2.3 节 “在基于 Operator 的代理部署中将地址与配置的地址匹配”。

4.1.2. 代理 Pod 的目录结构

在创建基于 Operator 的代理部署时，每个代理的 Pod 在 OpenShift 项目中的 StatefulSet 中运行。代理的应用程序容器在每个 Pod 中运行。

在初始化每个 Pod 时，Operator 会运行一种名为 Init Container 的容器类型。在 OpenShift Container Platform 中，Init 容器是在应用程序容器之前运行的专用容器。初始容器可以包含应用程序镜像中不存在的实用程序或设置脚本。

在为代理实例生成配置时，Init 容器使用默认安装目录中包含的文件。此安装目录位于 Operator 挂载到代理 Pod 的卷中，以及 Init Container 和代理容器共享的卷。Init 容器用于挂载共享卷的路径在名为 CONFIG_INSTANCE_DIR 的环境变量中定义。CONFIG_INSTANCE_DIR 的默认值为 /amq/init/config。在文档中，此目录被称为 <install_dir>。



注意

您无法更改 CONFIG_INSTANCE_DIR 环境变量的值。

默认情况下，安装目录有以下子目录：

子目录	内容
<install_dir>/bin	运行代理所需的二进制文件和脚本.
<install_dir>/etc	配置文件.
<install_dir>/data	代理日志.
<install_dir>/lib	运行代理所需的 jars 和库。
<install_dir>/log	代理日志文件.
<install_dir>/tmp	临时 Web 应用文件.

当初始容器完成生成代理配置时，代理应用程序容器会启动。启动时，broker 容器会从之前由初始容器使用的安装目录中复制其配置。代理 Pod 初始化并运行时，代理配置位于代理的 /home/jboss/amq-broker 目录（及子目录）。

其它资源

- 有关 Operator 如何为内置初始容器选择容器镜像的更多信息，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。

- 要了解如何构建和指定自定义初始容器镜像，请参阅 第 4.6 节 “指定自定义初始容器镜像”。

4.2. 为基于 OPERATOR 的代理部署配置地址和队列

对于基于 Operator 的代理部署，您可以使用两个单独的自定义资源(CR)实例来配置地址和队列，以及它们关联的设置。

- 要在代理上创建地址和队列，您需要根据地址自定义资源定义(CRD)部署 CR 实例。
  - 如果您使用 OpenShift 命令行界面(CLI)安装 Operator，地址 CRD 是您下载并提取的 Operator 安装存档 deploy/ crds 中的 broker_activemqartemisaddress_crd.yaml 文件。
  - 如果使用 OperatorHub 来安装 Operator，地址 CRD 是 OpenShift Container Platform Web 控制台中的 Administration → Custom Resource Definitions 中列出的 ActiveMQAretmisAddress CRD。
- 要配置您随后与特定地址匹配的地址和队列设置，您可以在用于创建代理部署的主自定义资源(CR)实例中包含配置。
  - 如果您使用 OpenShift CLI 安装 Operator，则 main broker CRD 是您下载并提取的 Operator 安装存档 deploy/ crds 中的 broker_activemqartemis_crd.yaml 文件。
  - 如果使用 OperatorHub 安装 Operator，则主代理 CRD 是 OpenShift Container Platform Web 控制台中的 Administration → Custom Resource Definitions 中列出的 ActiveMQAretmis CRD。

通常，您可以在 OpenShift Container Platform 上为代理部署配置的地址和队列设置 完全等同于 Linux 或 Windows 上的独立代理部署。但是，您应该了解这些设置的配置 方式 方面的一些差异。以下子部分描述了这些区别。

4.2.1. OpenShift 和独立代理部署之间地址和队列设置的不同

- 要在 OpenShift Container Platform 上为代理部署配置地址和队列设置，您可以在代理部署的主自定义资源(CR)实例的 addressSettings 部分添加配置。这与 Linux 或 Windows 上的独立部署不同，后者将配置添加到 broker.xml 配置文件中的 address-settings 元素中。

- 用于配置项目名称的格式因 OpenShift Container Platform 和独立代理部署而异。对于 OpenShift Container Platform 部署，配置项名称位于 camel 案例中，如 defaultQueueRoutingType。相比之下，独立部署的配置项名称为小写，使用短划线(-)分隔符，如 default-queue-routing-type。

下表显示了这一命名差异的一些进一步示例。

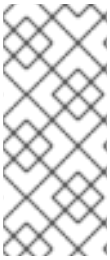
独立代理部署的配置项	OpenShift 代理部署的配置项
address-full-policy	addressFullPolicy
auto-create-queues	autoCreateQueues
default-queue-routing-type	defaultQueueRoutingType
last-value-queue	lastValueQueue

其它资源

- 有关为 OpenShift Container Platform 代理部署创建地址和队列以及匹配的设置示例，请参阅：
  - 为 OpenShift Container Platform 上的代理部署创建地址和队列
  - 为 OpenShift Container Platform 上的代理部署配置地址设置匹配
- 要了解有关 OpenShift Container Platform 代理部署的地址、队列和地址设置的所有配置选项，请参阅 第 8.1 节 “自定义资源配置参考”。
- 有关为 独立 代理部署配置地址、队列和相关地址设置的更多信息，请参阅 配置 AMQ Broker 中的地址 、队列和主题。您可以使用这些信息为 OpenShift Container Platform 上的代理部署创建等效的配置。

4.2.2. 为基于 Operator 的代理部署创建地址和队列

以下流程演示了如何使用自定义资源(CR)实例将地址和关联的队列添加到基于 Operator 的代理部署。



注意

要在代理部署中创建多个地址和/或队列，您需要创建单独的 CR 文件并单独部署它们，并在每次情况下指定新的地址和/或队列名称。此外，每个 CR 实例的 name 属性必须是唯一的。

先决条件

- 您必须已安装 AMQ Broker Operator，包括在代理上创建地址和队列所需的专用自定义资源定义(CRD)。有关安装 Operator 的两种替代方法的详情，请参考：
  - 第 3.2 节 “使用 CLI 安装 Operator”.
  - 第 3.3 节 “使用 OperatorHub 安装 Operator”.
- 您应该熟悉如何使用 CR 实例创建基本的代理部署。如需更多信息，请参阅 第 3.4.1 节 “部署基本代理实例”。

流程

1. 开始配置自定义资源(CR)实例，以定义代理部署的地址和队列。
   a. 使用 OpenShift 命令行界面：
      i. 以具有特权的用户身份登录 OpenShift，以便在项目中为代理部署部署 CR。

```
oc login -u <user> -p <password> --server=<host:port>
```

      ii. 打开名为 broker_activemqartemisaddress_cr.yaml 的示例 CR 文件，该文件包含在您下载并提取的 Operator 安装存档的 deploy/crs 目录中。
   b. 使用 OpenShift Container Platform Web 控制台：
      i. 以具有特权的用户身份登录到控制台，以便在用于代理部署的项目中部署 CR。

ii. 根据地址 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 **Administration** → **Custom Resource Definitions**。

iii. 单击 **ActiveMQArtemisAddress CRD**。

iv. 点 实例 选项卡。

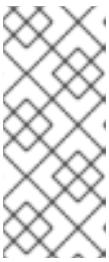
v. 单击 **Create ActiveMQArtemisAddress**。

在控制台中，会打开 **YAML** 编辑器，供您配置 **CR** 实例。

2. 在 **CR** 的 **spec** 部分中，添加行来定义地址、队列和路由类型。例如：

```
apiVersion: broker.amq.io/v2alpha2
kind: ActiveMQArtemisAddress
metadata:
  name: myAddressDeployment0
  namespace: myProject
spec:
  ...
  addressName: myAddress0
  queueName: myQueue0
  routingType: anycast
  ...
```

上述配置定义一个名为 **myAddress0** 的地址，其队列名为 **myQueue0**，以及 **anycast** 路由类型。



注意

在 **metadata** 部分中，您需要包含 **namespace** 属性，只有在 使用 **OpenShift Container Platform Web** 控制台创建 **CR** 实例时才需要指定一个值。您应指定的值是代理部署的 **OpenShift** 项目的名称。

3. 部署 **CR** 实例。

a. 使用 **OpenShift** 命令行界面：

i. 保存 CR 文件。

ii. 切换到代理部署的项目。

```
$ oc project <project_name>
```

iii. 创建 CR 实例。

```
$ oc create -f <path/to/address_custom_resource_instance>.yaml
```

b. 使用 OpenShift Web 控制台：

i. 配置完 CR 后，点 Create。

4. （可选）要删除之前使用 CR 实例添加到部署中的地址和队列，请使用以下命令：

```
$ oc delete -f <path/to/address_custom_resource_instance>.yaml
```

4.2.3. 在基于 Operator 的代理部署中将地址与配置的地址匹配

如果向客户端发送消息失败，您可能不希望代理不断尝试传递消息。为防止无限发送尝试，您可以定义一个 死信地址 和关联的 死信队列。在尝试了指定次数后，代理会从原始队列中删除未传送的消息，并将消息发送到配置的死信地址。之后，系统管理员可以使用死信队列中未传送的消息来检查消息。

以下示例演示了如何为基于 Operator 的代理部署配置死信地址和队列。这个示例演示了如何：

- 使用主代理自定义资源(CR)实例的 addressSetting 部分来配置地址设置。
- 将这些地址设置与代理部署中的地址匹配。

先决条件

- 您必须使用针对 AMQ Broker 7.9（即版本 7.9）的 Operator 的最新版本。要了解如何将

Operator 升级到最新版本，请参阅 第 6 章 升级基于 Operator 的代理部署。

- *您应该熟悉如何使用 CR 实例创建基本的代理部署。如需更多信息，请参阅 第 3.4.1 节 “部署基本代理实例”。*
- *您应该熟悉 Operator 合并或替换为 CR 实例中指定的配置 的默认 地址设置配置。如需更多信息，请参阅 第 4.1.1 节 “Operator 如何生成地址设置配置”。*

流程

1. *开始配置 CR 实例，以添加一个死信地址和队列，以接收部署中每个代理的未传送消息。*

   a. *使用 OpenShift 命令行界面：*

      i. *以具有特权的用户身份登录 OpenShift，以便在项目中为代理部署部署 CR。*

      ```
      oc login -u <user> -p <password> --server=<host:port>
      ```

      ii. *打开名为 broker_activemqartemisaddress_cr.yaml 的示例 CR 文件，该文件包含在您下载并提取的 Operator 安装存档的 deploy/crs 目录中。*

   b. *使用 OpenShift Container Platform Web 控制台：*

      i. *以具有特权的用户身份登录到控制台，以便在用于代理部署的项目中部署 CR。*

      ii. *根据地址 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 Administration → Custom Resource Definitions。*

      iii. *单击 ActiveMQArtemisAddress CRD。*

      iv. *点 实例 选项卡。*

v. 单击 **Create ActiveMQArtemisAddress**。

在控制台中，会打开 **YAML** 编辑器，供您配置 **CR** 实例。

2. 在 **CR** 的 **spec** 部分中，添加行来指定死信地址和队列，以接收未传送的消息。例如：

```
apiVersion: broker.amq.io/v2alpha2
kind: ActiveMQArtemisAddress
metadata:
  name: ex-aaoaddress
spec:
  ...
  addressName: myDeadLetterAddress
  queueName: myDeadLetterQueue
  routingType: anycast
  ...
```

上述配置定义了一个名为 **myDeadLetterAddress** 的死信地址，它带有名为 **myDeadLetterQueue** 和 **any cast** 路由类型的死信队列。

注意

在 **metadata** 部分中，您需要包含 **namespace** 属性，只有在 使用 **OpenShift Container Platform Web** 控制台创建 **CR** 实例时才需要指定一个值。您应指定的值是代理部署的 **OpenShift** 项目的名称。

3. 部署地址 **CR** 实例。

a. 使用 **OpenShift** 命令行界面：

i. 保存 **CR** 文件。

ii. 切换到代理部署的项目。

```
$ oc project <project_name>
```

...

iii. *创建地址 CR。*

```
$ oc create -f <path/to/address_custom_resource_instance>.yaml
```

b. *使用 OpenShift Web 控制台：*

i. *配置完 CR 后，点 Create。*

4. *为代理部署开始配置自定义资源(CR)实例。*

a. *示例 CR 文件：*

i. *打开名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载并提取的 Operator 安装存档的 deploy/crs 目录中。*

b. *使用 OpenShift Container Platform Web 控制台：*

i. *根据主代理 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 Administration → Custom Resource Definitions。*

ii. *单击 ActiveMQArtemis CRD。*

iii. *点 实例 选项卡。*

iv. *单击 Create ActiveMQArtemis。*

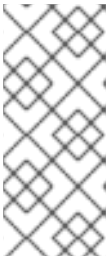
在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

对于基本的代理部署，配置可能类似如下。此配置是 broker_activemqartemis_cr.yaml 示例 CR 文件的默认内容。

```
apiVersion: broker.amq.io/v2alpha4
```

```
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
        size: 1
        image: placeholder
        requireLogin: false
        persistenceEnabled: true
        journalType: nio
        messageMigration: true
```

请注意，在 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性 被设置为默认值 占位符。此值表示 image 属性默认不指定用于部署的代理容器镜像。要了解 Operator 如何决定要使用的适当代理容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。



注意

在 metadata 部分中，您需要包含 namespace 属性，只有在 使用 OpenShift Container Platform Web 控制台创建 CR 实例时才需要指定一个值。您应指定的值是代理部署的 OpenShift 项目的名称。

5. 在 CR 的 deploymentPlan 部分中，添加一个包含单个 addressSettings 部分的新 addressSettings 部分，如下所示。

```
spec:
    version: 7.9.3
    deploymentPlan:
        size: 1
        image: placeholder
        requireLogin: false
        persistenceEnabled: true
        journalType: nio
        messageMigration: true
        addressSettings:
            addressSetting:
```

6. 将 match 属性的一个实例添加到 addressSetting 块。指定一个 address-matching 表达式。例如：

```
spec:
    version: 7.9.3
    deploymentPlan:
        size: 1
        image: placeholder
```

```
      requireLogin: false
      persistenceEnabled: true
      journalType: nio
      messageMigration: true
      addressSettings:
        addressSetting:
        - match: myAddress
```

匹配

指定代理应用以下配置的地址或地址 集合。在本例中，match 属性的值对应于一个名为 myAddress 的地址。

7. *添加与未传送消息相关的属性并指定值。例如：*

```
spec:
  version: 7.9.3
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      addressSetting:
      - match: myAddress
        deadLetterAddress: myDeadLetterAddress
        maxDeliveryAttempts: 5
```

deadLetterAddress

代理将未传送的信息发送到的地址。

maxDeliveryAttempts

代理在将消息移至配置的死信地址前尝试的最大发送次数。

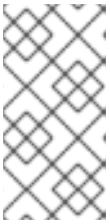
在前面的示例中，如果代理尝试向以 myAddress 开头的地址发送一条消息，代理会将消息移到指定的死信地址 myDeadLetterAddress。

8. *（可选）应用与其他地址或地址集类似的配置。例如：*

```
spec:
  version: 7.9.3
  deploymentPlan:
```

```
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      addressSetting:
      - match: myAddress
        deadLetterAddress: myDeadLetterAddress
        maxDeliveryAttempts: 5
      - match: 'myOtherAddresses*'
        deadLetterAddress: myDeadLetterAddress
        maxDeliveryAttempts: 3
```

在本例中，第二个 match 属性的值包含一个星号通配符。通配符表示，上述配置应用于以字符串 myOtherAddresses 开头的 任何 地址。



注意

如果您使用通配符表达式作为 match 属性的值，则必须在单引号中包含该值，例如： 'myOtherAddresses*'。

9. 在 addressSettings 部分的开头，添加 applyRule 属性并指定一个值。例如：

```
spec:
  version: 7.9.3
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      applyRule: merge_all
      addressSetting:
      - match: myAddress
        deadLetterAddress: myDeadLetterAddress
        maxDeliveryAttempts: 5
      - match: 'myOtherAddresses*'
        deadLetterAddress: myDeadLetterAddress
        maxDeliveryAttempts: 3
```

applyRule 属性指定 Operator 如何为每个匹配地址或一组地址应用您添加到 CR 中的配置。您可以指定的值有：

merge_all

- *对于在 CR 和 与同一地址组匹配的默认配置中指定的地址设置：*
  - *将默认配置中指定的任何属性值替换为 CR 中指定的任何属性值。*
  - *保留在 CR 或 默认配置中唯一指定的任何属性值。在最终合并的配置中包含每一个。*
- *对于 CR 中指定的地址设置， 或者唯一与特定地址或地址集匹配的默认配置，请在最终合并的配置中包括它们。*

merge_replace

- *对于 CR 和与同一地址组匹配的 CR 和 默认配置 中指定的地址设置，请在最终的 CR 中指定的设置中包括合并的配置。不要包含 默认配置中指定的任何属性，即使这些属性没有在 CR 中指定。*
- *对于 CR 中指定的地址设置， 或者唯一与特定地址或地址集匹配的默认配置，请在最终合并的配置中包括它们。*

replace_all

将默认配置中指定的 所有 地址设置替换为 CR 中指定的设置。最后，合并后的配置与 CR 中指定的配置完全对应。

注意

如果没有在 CR 中明确包含 applyRule 属性，Operator 将使用默认值 merge_all。

10. *部署代理 CR 实例。*

    a. *使用 OpenShift 命令行界面：*

i. 保存 CR 文件。

ii. 创建 CR 实例。

```
$ oc create -f <path/to/broker_custom_resource_instance>.yaml
```

b. 使用 OpenShift Web 控制台：

i. 配置完 CR 后，点 Create。

其它资源

- 要了解有关 OpenShift Container Platform 代理部署的地址、队列和地址设置的所有配置选项，请参阅 第 8.1 节 “自定义资源配置参考”。
- 如果使用 OpenShift 命令行界面(CLI)安装 AMQ Broker Operator，您下载并提取的安装存档包含一些配置地址设置的额外示例。在安装归档的 deploy/examples 目录中，请参考：
  - artemis-basic-address-settings-deployment.yaml
  - artemis-merge-replace-address-settings-deployment.yaml
  - artemis-replace-address-settings-deployment.yaml
- 有关为 独立 代理部署配置地址、队列和相关地址设置的更多信息，请参阅 配置 AMQ Broker 中的地址 、队列和主题。您可以使用这些信息为 OpenShift Container Platform 上的代理部署创建等效的配置。
- 如需有关 OpenShift Container Platform 中初始容器的更多信息，请参阅使用初始容器在部署 pod 前执行任务。

4.3. 为基于 OPERATOR 的代理部署创建安全配置

以下流程演示了如何使用自定义资源(CR)实例将用户和关联的安全配置添加到基于 Operator 的代理部署中。

先决条件

- 您必须已安装 AMQ Broker Operator。有关安装 Operator 的两种替代方法的详情，请参考：
  - 第 3.2 节 “使用 CLI 安装 Operator”.
  - 第 3.3 节 “使用 OperatorHub 安装 Operator”.
- 您应该熟悉代理安全，如安全 代理所述
- 您应该熟悉如何使用 CR 实例创建基本的代理部署。如需更多信息，请参阅 第 3.4.1 节 “部署基本代理实例”。

流程

您可以在创建代理部署前或之后部署安全 CR。但是，如果您在创建代理部署后部署安全 CR，则代理 Pod 会被重启以接受新配置。

1. 开始配置自定义资源(CR)实例，以定义代理部署的用户和相关安全配置。
   a. 使用 OpenShift 命令行界面：
      i. 以具有特权的用户身份登录 OpenShift，以便在项目中为代理部署部署 CR。

```
oc login -u <user> -p <password> --server=<host:port>
```

      ii. 打开名为 broker_activemqartemissecurity_cr.yaml 的示例 CR 文件，该文件包括在您下载并提取的 Operator 安装的 deploy/crs 目录中。

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有特权的用户身份登录到控制台，以便在用于代理部署的项目中部署 CR。

ii. 根据地址 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 Administration → Custom Resource Definitions。

iii. 单击 ActiveMQArtemisSecurity CRD。

iv. 点 实例 选项卡。

v. 单击 Create ActiveMQArtemisSecurity。

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

2. 在 CR 的 spec 部分中，添加行来定义用户和角色。例如：

```
apiVersion: broker.amq.io/v1alpha1
kind: ActiveMQArtemisSecurity
metadata:
  name: ex-prop
spec:
  loginModules:
    propertiesLoginModules:
      - name: "prop-module"
        users:
          - name: "sam"
            password: "samsecret"
            roles:
              - "sender"
          - name: "rob"
            password: "robsecret"
            roles:
              - "receiver"
  securityDomains:
    brokerDomain:
      name: "activemq"
      loginModules:
        - name: "prop-module"
          flag: "sufficient"
  securitySettings:
    broker:
```

```
    - match: "#"
      permissions:
        - operationType: "send"
          roles:
            - "sender"
        - operationType: "createAddress"
          roles:
            - "sender"
        - operationType: "createDurableQueue"
          roles:
            - "sender"
        - operationType: "consume"
          roles:
            - "receiver"
            ...
```

上述配置定义了两个用户：

- *一个名为 prop-module 的 propertiesLoginModule，它定义了名为 sam 的用户，其角色名为 sender。*
- *一个名为 prop-module 的 propertiesLoginModule，它定义了名为 rob 的用户，其角色名为 receiver。*

这些角色的属性在 securityDomains 部分的 broker Domain 和 broker 部分定义。例如，定义了 send 角色，以允许具有该角色的用户在任何地址上创建持久队列。默认情况下，配置会应用到当前命名空间中的 CR 定义的所有已部署代理。要将配置限制为特定的代理部署，请使用第 8.1.3 节 “安全自定义资源配置参考” 中描述的 applyToCrNames 选项。

注意

在 metadata 部分中，您需要包含 namespace 属性，只有在 使用 OpenShift Container Platform Web 控制台创建 CR 实例时才需要指定一个值。您应指定的值是代理部署的 OpenShift 项目的名称。

3. *部署 CR 实例。*

   a. *使用 OpenShift 命令行界面：*

      i. *保存 CR 文件。*

ii. 切换到代理部署的项目。

```
$ oc project <project_name>
```

iii. 创建 CR 实例。

```
$ oc create -f <path/to/address_custom_resource_instance>.yaml
```

b. 使用 OpenShift Web 控制台：

i. 配置完 CR 后，点 Create。

其它资源

- 第 8.1.3 节 “安全自定义资源配置参考”
- 第 3.4.1 节 “部署基本代理实例”

4.4. 配置代理存储要求

要在基于 Operator 的代理部署中使用持久性存储，您可以在用于创建部署的自定义资源(CR)实例中将 persistenceEnabled 设置为 true。如果您在 OpenShift 集群中没有容器原生虚拟化，则需要手动置备持久性卷(PV)，并确保 Operator 可以使用持久性卷声明(PVC)声明这些卷。如果要创建包含两个带有持久性存储的代理的集群，则需要有两个 PV 可用。默认情况下，部署中的每个代理都需要存储 2 GiB。但是，您可以为您的代理部署配置 CR，以指定每个代理所需的 PVC 大小。

重要

在首次部署 CR 之前，您必须将代理存储大小的配置添加到代理部署的主 CR 中。您不能 将配置添加到已在运行的代理部署中。

4.4.1. 配置代理存储大小

以下流程演示了如何为您的代理部署配置自定义资源(CR)实例，以指定代理为持久性存储所需的持久性卷声明(PVC)的大小。

重要

在首次部署 CR 之前，您必须将代理存储大小的配置添加到代理部署的主 CR 中。您不能 将配置添加到已在运行的代理部署中。

先决条件

- 您必须 至少 使用 AMQ Broker 7.7 的 Operator 最新版本（即 0.17 版本）。要了解如何将 Operator 升级到 AMQ Broker 7.9 的最新版本，请参阅 第 6 章 升级基于 Operator 的代理部署。

- 您应该熟悉如何使用 CR 实例创建基本的代理部署。请参阅 第 3.4.1 节 “部署基本代理实例”。

- 您必须已置备了持久性卷(PV)，并使它们可以被 Operator 声明使用。例如，如果要创建一个包含两个带有持久性存储的代理的集群，则需要有两个 PV 可用。

  有关置备持久性存储的更多信息，请参阅：

  - 了解持久性存储 (OpenShift Container Platform 4.5)

流程

1. 为代理部署开始配置自定义资源(CR)实例。

   a. 使用 OpenShift 命令行界面：

      i. 以具有特权的用户身份登录 OpenShift，以在您要创建部署的项目中部署 CR。

      ```
      oc login -u <user> -p <password> --server=<host:port>
      ```

      ii. 打开名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载并提取的 Operator 安装存档的 deploy/crs 目录中。

   b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有特权的用户身份登录控制台，以在您要创建部署的项目中部署 CR。

ii. 根据主代理 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 **Administration → Custom Resource Definitions**。

iii. 单击 **ActiveMQArtemis CRD**。

iv. 点 *实例* 选项卡。

v. 单击 **Create ActiveMQArtemis**。

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

对于基本的代理部署，配置可能类似如下。此配置是 broker_activemqartemis_cr.yaml 示例 CR 文件的默认内容。

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
        size: 1
        image: placeholder
        requireLogin: false
        persistenceEnabled: true
        journalType: nio
        messageMigration: true
```

请注意，在 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性 被设置为默认值 占位符。此值表示 image 属性默认不指定用于部署的代理容器镜像。要了解 Operator 如何决定要使用的适当代理容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。

2. 要指定代理存储要求，在 CR 的 deploymentPlan 部分添加 storage 部分。添加 大小 属性并指定值。例如：

```
spec:
```

```
    version: 7.9.3
    deploymentPlan:
      size: 1
      image: placeholder
      requireLogin: false
      persistenceEnabled: true
      journalType: nio
      messageMigration: true
      storage:
        size: 4Gi
```

storage.size

每个代理 Pod 所需的持久性卷声明(PVC)大小（以字节为单位）。此属性仅在 persistenceEnabled 设为 true 时应用。您指定的值 必须 包含一个单元。支持字节表示法（如 K、M、G）或二进制等效（Ki、Mi、Gi）。

3. 部署 CR 实例。

a. 使用 OpenShift 命令行界面：

i. 保存 CR 文件。

ii. 切换到您要在其中创建代理部署的项目。

```
$ oc project <project_name>
```

iii. 创建 CR 实例。

```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

b. 使用 OpenShift Web 控制台：

i. 配置完 CR 后，点 Create。

4.5. 为基于 OPERATOR 的代理部署配置资源限制和请求

当您创建基于 Operator 的代理部署时，部署中的代理 Pod 在 OpenShift 集群的一个节点上在 StatefulSet 中运行。您可以为部署配置自定义资源(CR)实例，以指定在每个 Pod 中运行的代理容器所使

用的 host-node 计算资源。通过为 CPU 和内存(RAM)指定限制和请求值，您可以确保代理 Pod 的满意性能。

重要

- 在 首次部署 CR 前，您必须为代理部署添加限制和请求的配置。您不能 将配置添加到已在运行的代理部署中。
- 红帽无法为限制和请求推荐值，因为它们基于您的特定消息传递系统用例和您已实施的架构。但是，建议您 在为生产环境配置这些值前测试和调整开发环境中的这些值。
- 在初始化每个代理 Pod 时，Operator 会运行名为 Init Container 的容器类型。您为每个代理容器配置的任何资源限制和请求也适用于每个初始容器。有关在代理部署中使用初始容器的更多信息，请参阅 第 4.1 节 “Operator 如何生成代理配置”。

您可以指定以下限制和请求值：

CPU 限制

对于 Pod 中运行的每个代理容器，这个值是容器可以使用的最大主机节点 CPU 量。如果代理容器尝试超过指定的 CPU 限制，OpenShift 会对容器进行限流。这样可确保容器具有一致的性能，无论节点上运行的 Pod 数量如何。

内存限制

对于 Pod 中运行的每个代理容器，这个值是容器可以消耗的最大主机节点内存量。如果代理容器尝试超过指定的内存限值，OpenShift 将终止该容器。代理 Pod 重启。

CPU 请求

对于 Pod 中运行的每个代理容器，这个值是容器请求的主机节点 CPU 的数量。OpenShift 调度程序在 Pod 放置期间考虑 CPU 请求值，以便将代理 Pod 绑定到具有充足计算资源的节点。

CPU 请求值是代理容器运行 的最小 CPU 量。但是，如果节点上没有 CPU 争用，容器可以使用所有可用的 CPU。如果您指定了 CPU 限值，容器不能超过这个 CPU 用量。如果节点上存在 CPU 争用，CPU 请求值为 OpenShift 提供了一种方式来衡量所有容器的 CPU 使用量。

内存请求

对于 Pod 中运行的每个代理容器，这个值是容器请求的主机节点内存量。OpenShift 调度程序在

Pod 放置期间考虑内存请求值，以便将代理 Pod 绑定到具有充足计算资源的节点。

内存请求值是代理容器运行 的最小 内存量。不过，容器可以消耗尽可能多的可用内存。如果您指定了内存限制，则代理容器不能超过该内存用量。

CPU 以名为 millicores 的单位来测量。OpenShift 集群中的每一节点检查操作系统，以确定节点上的 CPU 内核数。然后，节点将该值乘以 1000 来表示总容量。例如，如果节点有两个内核，节点的 CPU 容量表示为 2000m。因此，如果您要使用单个内核的十分之一，您可以将值指定为 100m。

内存以字节为单位计算。您可以使用字节表示法（E、P、T、G、M、K）或二进制等效（Ei、Pi、Ti、Gi、Mi、K）指定值。您指定的值必须包含一个单元。

4.5.1. 配置代理资源限制和请求

以下示例演示了如何为您的代理部署配置主自定义资源(CR)实例，以便为部署中 Pod 中运行的每个代理容器设置 CPU 和内存限值和请求。

重要

- 在 首次部署 CR 前，您必须为代理部署添加限制和请求的配置。您不能 将配置添加到已在运行的代理部署中。
- 红帽无法为限制和请求推荐值，因为它们基于您的特定消息传递系统用例和您已实施的架构。但是，建议您 在为生产环境配置这些值前测试和调整开发环境中的这些值。

先决条件

- 您应该熟悉如何使用 CR 实例创建基本的代理部署。请参阅 第 3.4.1 节 “部署基本代理实例”。

流程

1. 为代理部署开始配置自定义资源(CR)实例。

   a. 使用 OpenShift 命令行界面：

使用 OpenShift 命令行界面

i. 以具有特权的用户身份登录 OpenShift，以在您要创建部署的项目中部署 CR。

```
oc login -u <user> -p <password> --server=<host:port>
```

ii. 打开名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载并提取的 Operator 安装存档的 deploy/crs 目录中。

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有特权的用户身份登录控制台，以在您要创建部署的项目中部署 CR。

ii. 根据主代理 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 Administration → Custom Resource Definitions。

iii. 单击 ActiveMQArtemis CRD。

iv. 点 实例 选项卡。

v. 单击 Create ActiveMQArtemis。

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

对于基本的代理部署，配置可能类似如下。此配置是 broker_activemqartemis_cr.yaml 示例 CR 文件的默认内容。

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
        size: 1
        image: placeholder
```

```
      requireLogin: false
      persistenceEnabled: true
      journalType: nio
      messageMigration: true
```

请注意，在 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性 被设置为默认值 占位符。此值表示 image 属性默认不指定用于部署的代理容器镜像。要了解 Operator 如何决定要使用的适当代理容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。

2. 在 CR 的 deploymentPlan 部分中，添加一个 resources 部分。添加 限值 和 请求 子部分。在各个子部分中，添加一个 cpu 和 memory 属性并指定值。例如：

```
spec:
  version: 7.9.3
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    resources:
      limits:
        cpu: "500m"
        memory: "1024M"
      requests:
        cpu: "250m"
        memory: "512M"
```

limits.cpu

在部署中运行的 Pod 中运行的每个代理容器都不能超过这个数量的主机节点 CPU 用量。

limits.memory

在部署中运行的 Pod 中运行的每个代理容器都不能超过这个数量的主机节点内存用量。

requests.cpu

部署中在 Pod 中运行的每个代理容器都会请求这个数量的主机节点 CPU。这个值是代理容器运行 的最小 CPU 量。

requests.memory

部署中在 Pod 中运行的每个代理容器都会请求这个数量的主机节点内存。这个值是代理容器运行 的最小 内存量。

3. 部署 CR 实例。

a. 使用 OpenShift 命令行界面：

i. 保存 CR 文件。

ii. 切换到您要在其中创建代理部署的项目。

```
$ oc project <project_name>
```

iii. 创建 CR 实例。

```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

b. 使用 OpenShift Web 控制台：

i. 配置完 CR 后，点 Create。

4.6. 指定自定义初始容器镜像

如 第 4.1 节 “Operator 如何生成代理配置” 所述，AMQ Broker Operator 使用默认的内置初始容器来生成代理配置。要生成配置，Init 容器会为您的部署使用主自定义资源(CR)实例。您 唯一 可在 CR 中指定的项目是在主代理自定义资源定义(CRD)中公开的项目。

但是，在某些情况下，您可能需要包含 CRD 中未 公开的配置。在本例中，在主 CR 实例中，您可以指定一个 自定义 初始容器。自定义初始容器可以修改或添加到 Operator 已创建的配置中。例如，您可以使用自定义初始容器来修改代理日志记录设置。或者，您可以使用自定义初始容器在代理安装目录中包含额外的运行时依赖项（即 .jar 文件）。

在构建自定义初始容器镜像时，您必须遵循以下重要准则：

- 在您为自定义镜像创建的构建脚本（如 Docker Dockerfile 或 Podman Containerfile）中，FROM 指令必须指定 AMQ Broker Operator 内置初始容器的最新版本作为基础镜像。在脚本中，包含以下行：

```
FROM registry.redhat.io/amq7/amq-broker-init-
rhel8@sha256:b74d03ed852a3731467ffda95266ce49f2065972f1c37bf254f3d52b34c11991
```

- 自定义镜像必须包含一个名为 post-config.sh 的脚本，该脚本应包含在名为 /amq/scripts 的目录中。post-config.sh 脚本可在其中修改或添加到 Operator 生成的初始配置中。当您指定自定义初始容器时，Operator 在使用 CR 实例生成配置但启动代理应用程序容器前运行 post-config.sh 脚本。

- 如 第 4.1.2 节 “代理 Pod 的目录结构” 所述，Init Container 使用的安装目录的路径在名为 CONFIG_INSTANCE_DIR 的环境变量中定义。在引用安装目录时，post-config.sh 脚本应使用此环境变量名称（例如 ${CONFIG_INSTANCE_DIR}/lib），而不是此变量的实际值（如 /amq/init/config/lib）。

- 如果要在自定义代理配置中包含其他资源（如 .xml 或.jar 文件），您必须确保这些资源包含在自定义镜像中，并可以被 post-config.sh 脚本访问。

以下流程描述了如何指定自定义初始容器镜像。

先决条件

- 您必须使用 Operator 至少有 7.9. 3-opr-3 版本。要了解如何升级到最新的 Operator 版本，请参阅 第 6 章 升级基于 Operator 的代理部署。

- 您必须已构建了符合上述准则的自定义初始容器镜像。有关为 ArtemisCloud Operator 构建和指定自定义初始容器镜像的完整示例，请参阅 自定义 Init 容器镜像，以了解基于 JDBC 的持久性。

- 要为 AMQ Broker Operator 提供自定义初始容器镜像，您需要将该镜像添加到容器镜像仓库（如 Quay 容器 registry ）中的存储库。

- 您应该了解 Operator 如何使用初始容器生成代理配置。如需更多信息，请参阅 第 4.1 节 “Operator 如何生成代理配置”。

- 您应该熟悉如何使用 CR 创建代理部署。如需更多信息，请参阅 第 3.4 节 “创建基于 Operator 的代理部署”。

流程

1. 为代理部署开始配置自定义资源(CR)实例。

   a. 使用 OpenShift 命令行界面：

      i. 以具有特权的用户身份登录 OpenShift，以在您要创建部署的项目中部署 CR。

      ```
      oc login -u <user> -p <password> --server=<host:port>
      ```

      ii. 打开名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载并提取的 Operator 安装存档的 deploy/crs 目录中。

   b. 使用 OpenShift Container Platform Web 控制台：

      i. 以具有特权的用户身份登录控制台，以在您要创建部署的项目中部署 CR。

      ii. 根据主代理 CRD 启动一个新的 CR 实例。在左侧窗格中，单击 Administration → Custom Resource Definitions。

      iii. 单击 ActiveMQArtemis CRD。

      iv. 点 实例 选项卡。

      v. 单击 Create ActiveMQArtemis。

      在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

对于基本的代理部署，配置可能类似如下。此配置是 broker_activemqartemis_cr.yaml 示例 CR 文件的默认内容。

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
```

```
  application: ex-aao-app
spec:
  version: 7.9.3
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

请注意，在 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性被设置为默认值 占位符。此值表示 image 属性默认不指定用于部署的代理容器镜像。要了解 Operator 如何决定要使用的适当代理容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。

2. *在 CR 的 deploymentPlan 部分中，添加 initImage 属性。*

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
  version: 7.9.3
  deploymentPlan:
    size: 1
    image: placeholder
    initImage:
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

3. *将 initImage 属性的值设置为自定义初始容器镜像的 URL。*

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
  version: 7.9.3
  deploymentPlan:
    size: 1
    image: placeholder
    initImage: <custom_init_container_image_url>
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

- *initImage*

  指定自定义初始容器镜像的完整 URL，您必须将其添加到容器 registry 中的存储库。

4. *部署 CR 实例。*

   a. *使用 OpenShift 命令行界面：*

      i. *保存 CR 文件。*

      ii. *切换到您要在其中创建代理部署的项目。*

```
$ oc project <project_name>
```

      iii. *创建 CR 实例。*

```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

   b. *使用 OpenShift Web 控制台：*

      i. *配置完 CR 后，点 Create。*

其它资源

- *有关为 ArtemisCloud Operator 构建和指定自定义初始容器镜像的完整示例，请参阅 自定义 Init 容器镜像，以了解基于 JDBC 的持久性。*

4.7. 为客户端连接配置基于 OPERATOR 的代理部署

4.7.1. 配置接收器

要在 OpenShift 部署中启用与代理 Pod 的客户端连接，您可以为部署定义 接收器。接收器定义代理 Pod 如何接受连接。您可以在用于代理部署的主自定义资源(CR)中定义接收器。在创建接收器时，您可以指定要在接收器上启用的消息协议，以及用于这些协议的代理 Pod 上的端口。

以下流程演示了如何在 CR 中为您的代理部署定义新的接收器。

先决条件

- *要配置接收器，代理部署必须基于 AMQ Broker Operator 版本 0.9 或以上。有关安装 Operator 最新版本的详情请参考 第 3.2 节 “使用 CLI 安装 Operator”。*

流程

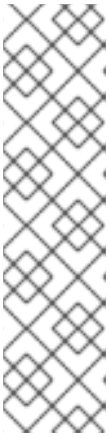
1. *在您在初始安装过程中下载并提取的 Operator 存档的 deploy/crs 目录中，打开 broker_activemqartemis_cr.yaml 自定义资源(CR)文件。*

2. *在 acceptors 元素中，添加命名的 acceptor。添加 协议 和端口 参数。设置值以指定要由接收器使用的消息传递协议，以及要为这些协议公开的每个代理 Pod 上的端口。例如：*

```
spec:
...
  acceptors:
  - name: my-acceptor
    protocols: amqp
    port: 5672
...
```

配置的接收器将端口 5672 公开给 AMQP 客户端。表中显示了您可以为 protocol 参数 指定的完整值集。

协议	值
核心协议	**core**
AMQP	**amqp**
OpenWire	**openwire**
MQTT	**mqtt**
STOMP	**stomp**
所有支持的协议	**all**



注意

- *对于部署中的每个代理 Pod，Operator 还会创建一个使用端口 61616 的默认接收器。代理集群需要这个默认接收器，并启用了内核协议。*
- *默认情况下，AMQ 代理管理控制台在代理 Pod 上使用端口 8161。部署中的每个代理 Pod 都有专用服务，提供对控制台的访问。如需更多信息，请参阅 第 5 章 连接到基于 Operator 的代理部署的 AMQ 管理控制台。*

3. *若要在同一接收器上使用其他协议，请修改 protocol 参数。指定以逗号分隔的协议列表。例如：*

```
spec:
...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
...
```

配置的接收器现在向 AMQP 和 OpenWire 客户端公开端口 5672。

4. *要指定接收器允许的并发客户端连接数量，请添加 connectionAllowed 参数并设置一个值。例如：*

```
spec:
...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
    connectionsAllowed: 5
...
```

5. *默认情况下，接收器仅公开给与代理部署相同的 OpenShift 集群中的客户端。若要同时将接收器公开给 OpenShift 之外的客户端，可添加 expose 参数，并将值设为 true。*

```
spec:
...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
```

```
    connectionsAllowed: 5
    expose: true
    ...
...
```

当您向 OpenShift 外的客户端公开接收器时，Operator 会自动为部署中的每个代理 Pod 创建专用服务和路由。

6. 若要启用来自 OpenShift 外部客户端的安全连接，可添加 sslEnabled 参数，并将值设为 true。

```
spec:
...
 acceptors:
 - name: my-acceptor
   protocols: amqp,openwire
   port: 5672
   connectionsAllowed: 5
   expose: true
   sslEnabled: true
   ...
...
```

当您在接收器（或连接器）上启用 SSL（即安全套接字层）安全性时，您可以添加相关配置，例如：

- 用于在 OpenShift 集群中存储身份验证凭据的机密名称。当您在接收器上启用 SSL 时，需要一个 secret。有关生成此 secret 的详情请参考 第 4.7.2 节 “保护 broker-client 连接”。
- 用于安全网络通信的传输层安全(TLS)协议。TLS 是更新的、更安全的 SSL 版本。您可以在 enabledProtocols 参数中指定 TLS 协议。
- 接收器是否在代理和客户端之间使用双向 TLS（也称为 互相身份验证 ）。您可以通过将 require ClientAuth 参数的值设置为 true 来指定此值。

其它资源

- 要了解如何配置 TLS 来保护代理-客户端连接的安全，包括生成 secret 来存储身份验证凭证，请参阅 第 4.7.2 节 “保护 broker-client 连接”。
-

有关完整的自定义资源配置参考，包括配置接收器和连接器，请参阅 第 8.1 节 “自定义资源配置参考”。

4.7.2. 保护 broker-client 连接

如果您已在接收器或连接器（即，通过将 sslEnabled 设为 true）上启用了安全性，您必须配置传输层安全(TLS)以允许代理和客户端之间进行基于证书的身份验证。TLS 是更新的、更安全的 SSL 版本。有两个主要 TLS 配置：

单向 TLS

只有代理才会显示证书。该证书供客户端用于验证代理。这是最常见的配置。

双向 TLS

代理和客户端都存在证书。这有时称为 相互身份验证。

以下部分描述：

- 单向和双向 TLS 使用的代理证书的配置要求
- 如何配置单向 TLS
- 如何配置双向 TLS

对于单向和双向 TLS，您可以通过生成一个 secret 来完成配置，该 secret 存储在代理和客户端之间成功 TLS 握手所需的凭证。这是您必须在安全接收器或连接器的 sslSecret 参数中指定的 secret 名称。secret 必须包含 Base64 编码的代理密钥存储（单向和双向 TLS）、Base64 编码的代理信任存储（仅限双向 TLS），以及这些文件的对应密码（也是 Base64 编码的）。单向和双向 TLS 配置程序演示了如何生成此机密。

注意

如果您没有在受保护的接收器或连接器的 sslSecret 参数中明确指定 secret 名称，则接收器或连接器会假定默认 secret 名称。默认 secret 名称使用格式 <custom_resource_name>-<acceptor_name>-secret 或 <custom_resource_name>-<connector_name>-secret。例如，my -broker-deployment-my-acceptor-secret。

即使接收器或连接器假设默认 secrete 名称，您仍然必须自己生成此 secret。它不会被自动创建。

4.7.2.1. 为主机名验证配置代理证书

注意

本节论述了配置单向或双向 TLS 时必须生成的代理证书的一些要求。

当客户端在部署中尝试连接到代理 Pod 时，客户端连接 URL 中的 verifyHost 选项决定了客户端是否将代理证书的 Common Name(CN)与其主机名进行比较，以验证它们是否匹配。如果您在客户端连接 URL 中指定了 verifyHost=true 或类似的内容，客户端会执行此验证。

在很少情况下，当您对连接的安全性不担心时，您可能会省略此验证，例如，代理部署在隔离的网络中 OpenShift 集群上。否则，对于安全连接，建议客户端执行此验证。在这种情况下，正确配置代理密钥存储证书是确保客户端连接成功。

通常，当客户端使用主机验证时，您生成代理证书时指定的 CN 必须与客户端连接的代理 Pod 上的 Route 的完整主机名匹配。例如，如果您部署了单一代理 Pod，CN 可能类似如下：

```
CN=my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain
```

为确保 CN 可以在带有多个代理 的部署中解析到任何 代理 Pod，您可以指定一个星号(*)通配符字符来代替代理 Pod 的序号。例如：

```
CN=my-broker-deployment-*-svc-rte-my-openshift-project.my-openshift-domain
```

上例中显示的 CN 成功解析到 my-broker-deployment 部署中的任何代理 Pod。

另外，您在生成代理证书时指定的 Subject Alternative Name(SAN)必须 单独列出 部署中的所有代理 Pod，作为用逗号分开的列表。例如：

```
"SAN=DNS:my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain,DNS:my-broker-deployment-1-svc-rte-my-openshift-project.my-openshift-domain,..."
```

4.7.2.2. 配置单向 TLS

本节中的步骤演示了如何配置单向传输层安全(TLS)来保护代理客户端连接。

在单向 TLS 中，只有代理才会显示证书。客户端使用此证书来验证代理。

先决条件

- 当客户端使用主机名验证时，您应该了解代理证书生成的要求。如需更多信息，请参阅 第 4.7.2.1 节 “为主机名验证配置代理证书”。

流程

1. 为代理密钥存储生成自签名证书。

```
$ keytool -genkey -alias broker -keyalg RSA -keystore ~/broker.ks
```

2. 从代理密钥存储导出证书，以便与客户端共享证书。以 Base64 编码 .pem 格式导出证书。例如：

```
$ keytool -export -alias broker -keystore ~/broker.ks -file ~/broker_cert.pem
```

3. 在客户端上，创建一个导入代理证书的客户端信任存储。

```
$ keytool -import -alias broker -keystore ~/client.ts -file ~/broker_cert.pem
```

4. 以管理员身份登录 OpenShift 容器平台。例如：

```
$ oc login -u system:admin
```

5.

5. 切换到包含代理部署的项目。例如：

```
$ oc project <my_openshift_project>
```

6. 创建用于存储 TLS 凭据的机密。例如：

```
$ oc create secret generic my-tls-secret \
--from-file=broker.ks=~/broker.ks \
--from-file=client.ts=~/broker.ks \
--from-literal=keyStorePassword=<password> \
--from-literal=trustStorePassword=<password>
```

注意

在生成机密时，OpenShift 要求您同时指定密钥存储和信任存储。trust store 键通常命名为 client.ts。对于代理和客户端之间的单向 TLS，实际不需要信任存储。但是，若要成功生成 secret，您需要将一些有效的存储文件指定为 client.ts 的值。前面的步骤通过重复使用之前生成的代理密钥存储文件为 client.ts 提供"dummy"值。这足以生成含有单向 TLS 所需所有凭证的 secret。

7. 将 secret 链接到您在安装 Operator 时创建的服务帐户。例如：

```
$ oc secrets link sa/amq-broker-operator secret/my-tls-secret
```

8. 在安全接收器或连接器的 sslSecret 参数中指定 secret 名称。例如：

```
spec:
...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
    sslEnabled: true
    sslSecret: my-tls-secret
    expose: true
    connectionsAllowed: 5
...
```

4.7.2.3. 配置双向 TLS

本节中的步骤演示了如何配置双向传输层安全(TLS)来保护代理客户端连接。

在双向 TLS 中，代理和客户端都显示证书。代理和客户端使用这些证书在有时称为相互 身份验证的过程中相互验证。

先决条件

- 当客户端使用主机名验证时，您应该了解代理证书生成的要求。如需更多信息，请参阅 第 4.7.2.1 节 “为主机名验证配置代理证书”。

流程

1. 为代理密钥存储生成自签名证书。

```
$ keytool -genkey -alias broker -keyalg RSA -keystore ~/broker.ks
```

2. 从代理密钥存储导出证书，以便与客户端共享证书。以 Base64 编码 .pem 格式导出证书。例如：

```
$ keytool -export -alias broker -keystore ~/broker.ks -file ~/broker_cert.pem
```

3. 在客户端上，创建一个导入代理证书的客户端信任存储。

```
$ keytool -import -alias broker -keystore ~/client.ts -file ~/broker_cert.pem
```

4. 在客户端上，为客户端密钥存储生成自签名证书。

```
$ keytool -genkey -alias broker -keyalg RSA -keystore ~/client.ks
```

5. 在客户端上，从客户端密钥存储导出证书，以便它可以与代理共享。以 Base64 编码 .pem 格式导出证书。例如：

```
$ keytool -export -alias broker -keystore ~/client.ks -file ~/client_cert.pem
```

6. 创建导入客户端证书的代理信任存储。

```
$ keytool -import -alias broker -keystore ~/broker.ts -file ~/client_cert.pem
```

7.

7. 以管理员身份登录 OpenShift 容器平台。例如：

```
$ oc login -u system:admin
```

8. 切换到包含代理部署的项目。例如：

```
$ oc project <my_openshift_project>
```

9. 创建用于存储 TLS 凭据的机密。例如：

```
$ oc create secret generic my-tls-secret \
--from-file=broker.ks=~/broker.ks \
--from-file=client.ts=~/broker.ts \
--from-literal=keyStorePassword=<password> \
--from-literal=trustStorePassword=<password>
```

注意

在生成机密时，OpenShift 要求您同时指定密钥存储和信任存储。trust store 键通常命名为 client.ts。对于代理和客户端之间的双向 TLS，您必须生成一个包含代理信任存储的 secret，因为这保存了客户端证书。因此，在上一步中，您为 client.ts 键指定的值实际上是 代理 信任存储文件。

10. 将 secret 链接到您在安装 Operator 时创建的服务帐户。例如：

```
$ oc secrets link sa/amq-broker-operator secret/my-tls-secret
```

11. 在安全接收器或连接器的 sslSecret 参数中指定 secret 名称。例如：

```
spec:
...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
    sslEnabled: true
    sslSecret: my-tls-secret
    expose: true
    connectionsAllowed: 5
...
```

4.7.3. 代理部署的网络服务

在用于代理部署的 OpenShift Container Platform Web 控制台的 Networking 窗格中，有两个正在运行的服务，即 无头 服务和 ping 服务。无头服务的默认名称格式为 <custom_resource_name>-hdls-svc，如 my-broker-deployment-hdls-svc。ping Service 的默认名称格式为 <custom_resource_name>-ping-svc，如 'my-broker-deployment-ping-svc。

无头服务允许访问每个代理 Pod 上的端口 8161 和 61616。代理管理控制台使用端口 8161，端口 61616 用于代理集群。您还可以使用无头服务从内部客户端（即与代理部署相同的 OpenShift 集群内客户端）连接到代理 Pod。

ping Service 由代理用于发现，并允许代理在 OpenShift 环境中组成集群。在内部，此服务公开端口 8888。

其它资源

- 要了解如何使用无头服务从内部客户端连接到代理 Pod，请参阅 第 4.7.4.1 节 “从内部客户端连接到代理”。

4.7.4. 从内部和外部客户端连接到代理

本节中的示例演示了如何从内部客户端（即与代理部署相同的 OpenShift 集群中的客户端）和外部客户端（即 OpenShift 集群外的客户端）连接到代理。

4.7.4.1. 从内部客户端连接到代理

内部客户端可以使用为代理部署运行的 无头 服务连接到代理 Pod。

要使用无头服务连接到代理 Pod，请使用 <Protocol>://<PodName>.<HeadlessServiceName>.<ProjectName>.svc.cluster.local 格式指定一个地址。例如：

```
$ tcp://my-broker-deployment-0.my-broker-deployment-hdls-svc.my-openshift-project.svc.cluster.local
```

OpenShift DNS 成功以此格式解析地址，因为基于 Operator 的代理部署创建的 StatefulSet 提供稳定的 Pod 名称。

其它资源

-

有关代理部署中默认运行的无头服务的更多信息，请参阅 第 4.7.3 节 “代理部署的网络服务”。

4.7.4.2. 从外部客户端连接到代理

当您向外部客户端公开接收器（即，通过将 expose 参数的值设置为 true）时，Operator 会自动为部署中的每个代理 Pod 创建专用服务和路由。要查看给定代理 Pod 上配置的路由，请在 OpenShift Container Platform Web 控制台中选择 Pod 并点击 Routes 选项卡。

外部客户端可以通过指定为代理 Pod 创建的路由的完整主机名来连接到代理。您可以使用基本的 curl 命令测试对此完整主机名的外部访问。例如：

```
$ curl https://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain
```

Route 的完整主机名必须解析到托管 OpenShift 路由器的节点。OpenShift 路由器使用主机名来确定流量在 OpenShift 内部网络内的发送位置。

默认情况下，OpenShift 路由器针对受保护（即 SSL 加密）流量和端口 443 侦听端口 80。对于 HTTP 连接，如果您指定了安全连接 URL（即 https），如果指定了非安全连接 URL（即 http），路由器会自动将流量定向到端口 443。

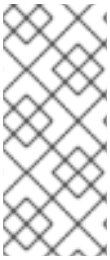
对于非 HTTP 连接：

- 客户端必须明确指定端口号（如端口 443）作为连接 URL 的一部分。
- 对于单向 TLS，客户端必须指定其信任存储的路径和对应的密码，作为连接 URL 的一部分。
- 对于双向 TLS，客户端还必须指定其 密钥 存储的路径和对应的密码，作为连接 URL 的一部分。

对于支持的消息传递代理，一些客户端连接 URL 示例如下所示。

外部核心客户端，使用单向 TLS

```
tcp://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
useTopologyForLoadBalancing=false&sslEnabled=true \
&trustStorePath=~/client.ts&trustStorePassword=<password>
```



注意

在连接 **URL** 中，使用**TopologyForLoadBalancing** 键明确设置为 **false**，因为外部核心客户端无法使用代理返回的拓扑信息。如果此键设为 **true**，或者您没有指定值，则会产生 **DEBUG** 日志消息。

外部核心客户端，使用双向 TLS

```
tcp://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
useTopologyForLoadBalancing=false&sslEnabled=true \
&keyStorePath=~/client.ks&keyStorePassword=<password> \
&trustStorePath=~/client.ts&trustStorePassword=<password>
```

外部 OpenWire 客户端，使用单向 TLS

```
ssl://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443"

# Also, specify the following JVM flags
-Djavax.net.ssl.trustStore=~/client.ts -Djavax.net.ssl.trustStorePassword=<password>
```

外部 OpenWire 客户端，使用双向 TLS

```
ssl://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443"

# Also, specify the following JVM flags
-Djavax.net.ssl.keyStore=~/client.ks -Djavax.net.ssl.keyStorePassword=<password> \
-Djavax.net.ssl.trustStore=~/client.ts -Djavax.net.ssl.trustStorePassword=<password>
```

使用单向 TLS 的外部 AMQP 客户端

```
amqps://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
transport.verifyHost=true \
&transport.trustStoreLocation=~/client.ts&transport.trustStorePassword=<password>
```

使用双向 TLS 的外部 AMQP 客户端

```
amqps://my-broker-deployment-0-svc-rte-my-openshift-project.my-openshift-domain:443?
transport.verifyHost=true \
&transport.keyStoreLocation=~/client.ks&transport.keyStorePassword=<password> \
&transport.trustStoreLocation=~/client.ts&transport.trustStorePassword=<password>
```

4.7.4.3. 使用 NodePort 连接到代理

除了使用路由外，OpenShift 管理员可以将 NodePort 配置为从 OpenShift 外部的客户端连接到代理 Pod。NodePort 应映射到由代理配置的接收器指定的协议指定的端口之一。

默认情况下，NodePort 的范围是 30000 到 32767，这意味着 NodePort 通常与代理 Pod 上的预期端口不匹配。

要通过 NodePort 从 OpenShift 外部的客户端连接到代理，您需要以 < protocol>:// < ocp_node_ip>:<node_port_number> 格式指定一个 URL。

其它资源

- *有关使用路由和 NodePort 等方法从 OpenShift 集群外部与集群中运行的服务进行通信的更多信息，请参阅：*

- 配置集群入口流量概述 (OpenShift Container Platform 4.5)

4.8. 为 AMQP 消息配置大型消息处理

客户端可能会发送可能会超过代理内部缓冲区大小的大型 AMQP 消息，从而导致意外错误。要防止这种情况，您可以将代理配置为在消息大于指定最小值时将消息存储为文件。以这种方式处理大量消息意味着代理不会将消息保存在内存中。相反，代理会将消息存储在用于存储大型消息文件的专用目录中。

对于 OpenShift Container Platform 上的代理部署，大型信息目录为 /opt/<custom_resource_name>/data/large-messages on the Persistent Volume(PV)。当代理将消息存储为大消息时，队列会在大型消息目录中保留对 文件的引用。

重要

对于 AMQ Broker 7.9 中基于 Operator 的代理部署，大型消息处理仅适用于 AMQP 协议。

4.8.1. 为大型消息处理配置 AMQP 接收器

以下流程演示了如何配置接收器来处理大于指定大小的 AMQP 消息作为大型消息。

先决条件

- 您应该熟悉如何为基于 Operator 的代理部署配置接收器。请参阅 第 4.7.1 节 “配置接收器”。
- 要在专用的大型消息目录中存储大型 AMQP 消息，代理部署必须使用持久性存储（即，用于创建部署的自定义资源(CR)实例中 persistenceEnabled 设置为 true ）。有关配置持久性存储的更多信息，请参阅：
  - 第 2.5 节 “Operator 部署备注”
  - 第 8.1 节 “自定义资源配置参考”

流程

1. 打开您之前定义了 AMQP 接收器的自定义资源(CR)实例。

   a. 使用 OpenShift 命令行界面：

   ```
   $ oc edit -f <path/to/custom_resource_instance>.yaml
   ```

   b. 使用 OpenShift Container Platform Web 控制台：

      i. 在左侧导航菜单中点击 Administration → Custom Resource Definitions

      ii. 单击 ActiveMQArtemis CRD。

      iii. 点 实例 选项卡。

      iv. 查找与项目命名空间对应的 CR 实例。

   之前配置的 AMQP 接收器可能类似如下：

   ```
   spec:
   ...
     acceptors:
     - name: my-acceptor
       protocols: amqp
       port: 5672
       connectionsAllowed: 5
       expose: true
       sslEnabled: true
   ...
   ```

2. 指定代理作为大型消息处理的 AMQP 消息的最小大小，以字节为单位。例如：

   ```
   spec:
   ...
     acceptors:
     - name: my-acceptor
       protocols: amqp
       port: 5672
       connectionsAllowed: 5
   ```

```
    expose: true
    sslEnabled: true
    amqpMinLargeMessageSize: 204800
    ...
...
```

在前面的示例中，代理配置为接受端口 5672 上的 AMQP 消息。根据 amqpMinLargeMessageSize 的值，如果接收器收到 AMQP 消息，其正文大于或等于 204800 字节（即 200 KB），代理会将消息存储为大消息。

代理将消息存储在大型信息目录中（默认为/opt/<custom_resource_name> /data/large-messages ）中。

如果您没有为 amqpMinLargeMessageSize 属性明确指定值，代理将使用默认值 102400（即 100 KB）。

如果您将 amqpMinLargeMessageSize 设置为 -1，则禁用 AMQP 消息的大型消息处理。

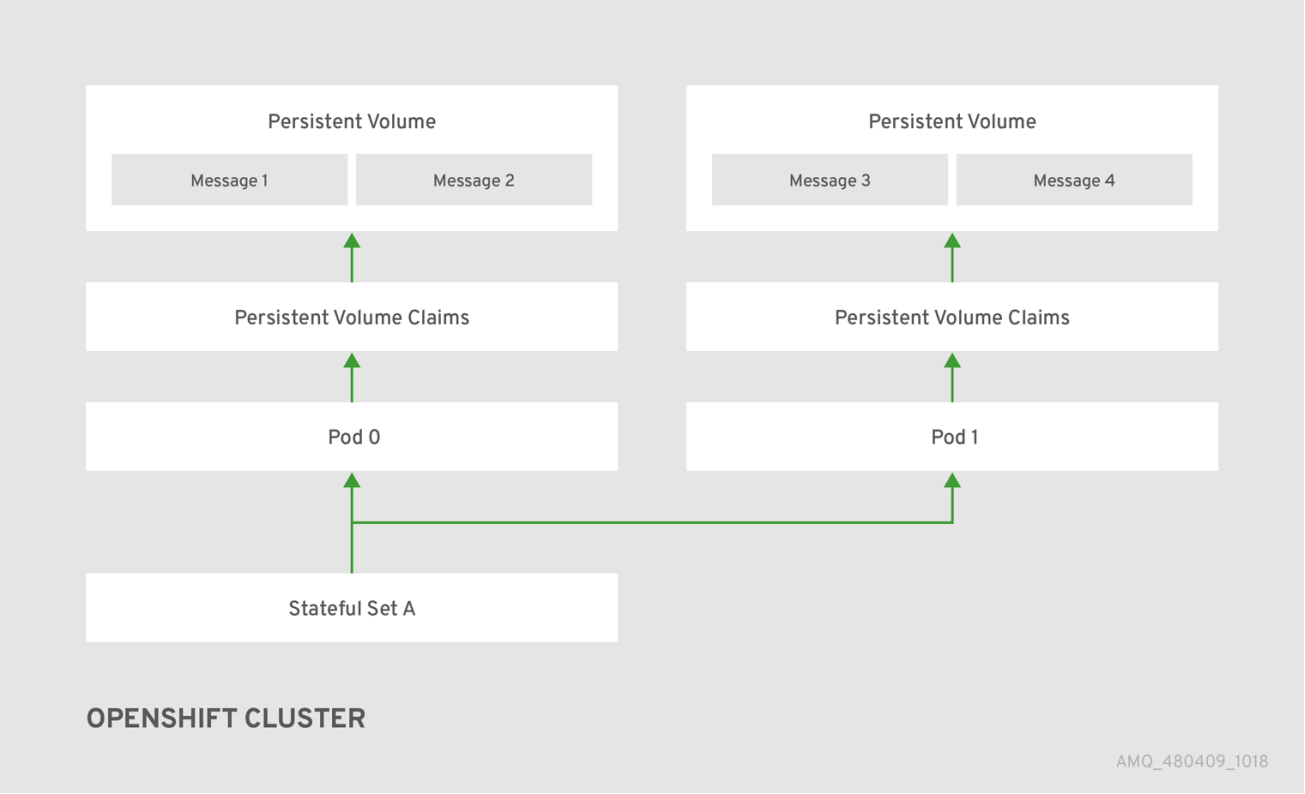
4.9. 高可用性和信息迁移

4.9.1. 高可用性

术语 高可用性 指的是即使该系统部分出现故障或已关闭，也能保持正常运行的系统。对于 OpenShift Container Platform 上的 AMQ Broker，这意味着在代理 Pod 失败时确保消息传递数据的完整性和可用性，或者因为您有意缩减部署而关闭。

要允许 AMQ Broker 在 OpenShift Container Platform 上高可用性，您可以在代理集群中运行多个代理 Pod。每个代理 Pod 将其消息数据写入一个声明用于持久性卷声明(PVC)的可用持久性卷(PV)。如果代理 Pod 失败或关闭，则存储在 PV 中的消息数据会迁移到代理集群中的另一个可用代理 Pod。其他代理 Pod 将消息数据存储在自己的 PV 中。

下图显示了基于 StatefulSet 的代理部署。在本例中，代理集群中的两个代理 Pod 仍在运行。



当代理 Pod 关闭时，AMQ Broker Operator 会自动启动一个 缩减控制器，执行消息迁移到仍然在代理集群中运行的另一代理 Pod。此消息迁移过程也称为 Pod draining。下面的部分描述了消息迁移。

4.9.2. 消息迁移

当集群部署中的代理因为部署失败或有意缩减而关闭时，消息迁移可确保消息传递数据的完整性。这个过程也称为 Pod 排空，指的是从已关闭的代理 Pod 中移除和重新发布信息。

注意

- *用于执行消息迁移的 scaledown 控制器只能在单个 OpenShift 项目中运行。控制器无法在单独的项目中的代理之间迁移消息。*
- *要使用消息迁移，在部署中必须至少有两个代理。默认情况下，带有两个或多个代理的代理会被集群。*

对于基于 Operator 的代理部署，您可以通过在部署的主代理自定义资源中将 messageMigration 设置为 true 来启用消息迁移。

消息迁移过程按照以下步骤执行：

1. 当部署中的代理 Pod 因部署失败或有意缩减而关闭时，Operator 会自动启动缩减控制器来准备消息迁移。scaledown 控制器在与代理集群相同的 OpenShift 项目名称中运行。

2. scaledown controller 注册自己，并侦听与项目中持久性卷声明(PVC)相关的 Kubernetes 事件。

3. 要检查卷声明中孤立的持久性卷(PV)，scaledown 控制器会查看卷声明的ordinal。控制器将卷声明中的序数与仍在项目中 StatefulSet（即代理集群）中运行的代理 Pod 的序数进行比较。

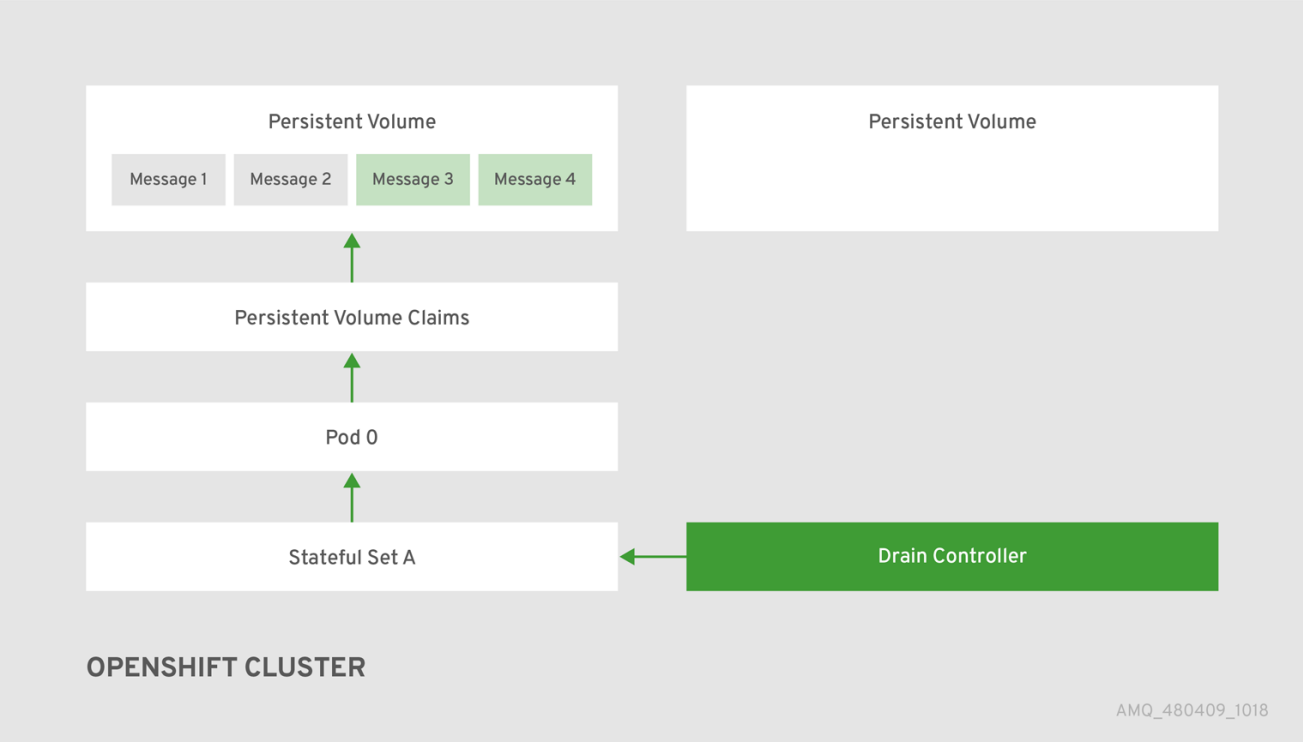
   如果卷声明中的序数大于仍在代理集群中运行的任何代理 Pod 的ordinal，缩减控制器会决定该序上的代理 Pod 已关闭，且该消息必须迁移到另一个代理 Pod。

4. scaledown 控制器启动一个排空程序 Pod。drainer Pod 运行代理并执行消息迁移。然后，drainer Pod 会识别一个可迁移孤立消息的替代代理 Pod。

注意

您的部署中必须至少有一个代理 Pod 运行，才能进行消息迁移。

下图演示了扩展控制器（也称为 排空控制器）如何将消息迁移到正在运行的代理 Pod。



成功将消息迁移到操作代理 Pod 后，drainer Pod 会关闭，缩减控制器会删除孤立 PV 的 PVC。PV 返回到"Released"状态。

注意

如果您将代理部署缩减为 0（零），则不会进行消息迁移，因为没有正在运行的代理 Pod 可以迁移到哪些消息传递数据。但是，如果您将部署缩减到零，然后备份到小于原始部署的大小，则会为保持关闭的代理启动排空 Pod。

其它资源

- *有关在缩减代理部署时的消息迁移示例，请参阅 在缩减时迁移消息。*

4.9.3. 在缩减时迁移消息

要在扩展代理部署时迁移消息，请使用主代理自定义资源(CR)来启用消息迁移。AMQ Broker Operator 自动运行专用缩减控制器，在缩减集群代理部署时执行消息迁移。

启用消息迁移后，Operator 中的 scaledown 控制器会检测到代理 Pod 的关闭，并启动一个排空 Pod 来执行消息迁移。drainer Pod 连接到集群中的另一个 live 代理 Pod，并将信息迁移到该 live 代理 Pod。迁移完成后，扩展控制器将关闭。

注意

- 扩展控制器仅在单个 OpenShift 项目中运行。控制器无法在单独的项目中的代理之间迁移消息。
- 如果您将代理部署缩减为 0（零），则不会进行消息迁移，因为没有正在运行的代理 Pod 可以迁移到消息传递数据。但是，如果您将部署缩减到零代理，然后只备份到原始部署中的一部分代理，则为保持关闭的代理启动排空 Pod。

以下示例步骤演示了 scaledown 控制器的行为。

先决条件

- 您已有基本的代理部署。请参阅 第 3.4.1 节 “部署基本代理实例”。
- 您应了解消息迁移的工作方式。如需更多信息，请参阅 第 4.9.2 节 “消息迁移”。

流程

1. 在您最初下载并提取的 Operator 存储库的 deploy/crs 目录中，打开主代理 CR broker_activemqartemis_cr.yaml。
2. 在主代理 CR 中，将 messageMigration 和 persistenceEnabled 设置为 true。

   这些设置意味着，当您稍后缩减集群代理部署的大小时，Operator 会自动启动扩展控制器，并将信息迁移到仍在运行的代理 Pod 中。
3. 在现有的代理部署中，验证哪些 Pod 正在运行。

```
$ oc get pods
```

您看到类似于如下的输出：

```
activemq-artemis-operator-8566d9bf58-9g25l   1/1   Running   0   3m38s
ex-aao-ss-0                                  1/1   Running   0   112s
ex-aao-ss-1                                  1/1   Running   0   8s
```

前面的输出显示有三个 Pod 正在运行；一个用于 broker Operator 本身，另一个用于部署中每个代理。

4. 登录到每个 Pod 并向每个代理发送一些信息。

   a. 使 Pod ex-aao-s-0 的集群 IP 地址为 172.17.0.6，运行以下命令：

   ```
   $ /opt/amq-broker/bin/artemis producer --url tcp://172.17.0.6:61616 --user admin --password admin
   ```

   b. 使 Pod ex-aao-s-1 的集群 IP 地址为 172.17.0.7，运行以下命令：

   ```
   $ /opt/amq-broker/bin/artemis producer --url tcp://172.17.0.7:61616 --user admin --password admin
   ```

   上述命令在每个代理上创建一个名为 TEST 的队列，并为每个队列添加 1000 消息。

5. 将集群从两个代理缩减到一个代理。

   a. 打开 main broker CR broker_activemqartemis_cr.yaml。

   b. 在 CR 中，将 deploymentPlan.size 设置为 1。

   c. 在命令行中应用更改：

   ```
   $ oc apply -f deploy/crs/broker_activemqartemis_cr.yaml
   ```

   您会看到 Pod ex-aao-ss-1 开始关闭。scaledown 控制器启动名称相同的新 drainer Pod。此排空 Pod 在将所有消息从代理 Pod ex-aao-ss-1 迁移到集群中的其他代理 Pod ex-aao-ss-0 后也会关闭。

6. 关闭 drainer Pod 时，检查代理 Pod ex-aao-ss-0 的 TEST 队列中的消息数。您会看到队列中的消息数量是 2000，这表示 drainer Pod 成功从已关闭的代理 Pod 中迁移了 1000 条信息。

第 5 章 连接到基于 OPERATOR 的代理部署的 AMQ 管理控制台

基于 Operator 的部署中的每个代理 Pod 在端口 8161 上托管自己的 AMQ 管理控制台实例。要为每个代理提供对控制台的访问，您可以为代理部署配置自定义资源(CR)实例，以指示 Operator 自动为每个代理 Pod 创建专用服务和路由。

以下流程描述了如何连接已部署的代理的 AMQ 管理控制台。

先决条件

- 您必须已使用 AMQ Broker Operator 创建代理部署。例如，要了解如何使用示例 CR 创建基本代理部署，请参阅 第 3.4.1 节 “部署基本代理实例”。
- 要指示 Operator 自动为部署中的每个代理 Pod 创建 Service 和 Route，您必须在用于创建部署的自定义资源(CR)实例中将 console.expose 属性的值设置为 true。此属性的默认值为 false。有关完整的自定义资源配置参考，包括 CR 的 console 部分的配置，请参阅 第 8.1 节 “自定义资源配置参考”。

5.1. 连接到 AMQ 管理控制台

当您在用于创建代理部署的自定义资源(CR)实例中将 console.expose 属性的值设置为 true 时，Operator 会自动为每个代理 Pod 创建一个专用服务和路由，以提供 AMQ 管理控制台的访问权限。

auto-created Service 的默认名称格式为 <custom-resource-name>-wconsj-<broker-pod-ordinal>-svc。例如，my -broker-deployment-wconsj-0-svc。自动创建的路由的默认名称格式为 <custom-resource-name>-wconsj-<broker-pod-ordinal>-svc-rte。例如，my -broker-deployment-wconsj-0-svc-rte。

此流程演示了如何访问正在运行的代理 Pod 的控制台。

流程

1. 在 OpenShift Container Platform web 控制台中点 Networking → Routes。

   在 Routes 页面中，找到给定代理 Pod 的 wconsj Route。例如，my -broker-deployment-wconsj-0-svc-rte。

2. 在 Location 下，单击与路由对应的链接。

   Web 浏览器中打开一个新选项卡。

3. 单击 管理控制台 链接。

   AMQ 管理控制台登录页面将打开。

4. 要登录到控制台，在用于创建代理部署的自定义资源(CR)实例中输入为 adminUser 和 adminPassword 属性指定的值。

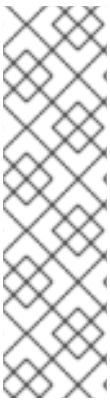
   如果没有为 CR 中的 adminUser 和 adminPassword 明确指定值，请按照 第 5.2 节 “访问 AMQ 管理控制台登录凭证” 中的说明检索登录到控制台所需的凭证。

注意

只有在 CR 的 requireLogin 属性设置为 true 时，才需要 adminUser 和 adminPassword 的值才能登录到控制台。此属性指定是否需要登录凭证来登录到代理 和 控制台。如果 requireLogin 设置为 false，您可以在提示输入用户名和密码时输入任何文本来登录控制台，而无需提供有效的用户名密码。

5.2. 访问 AMQ 管理控制台登录凭证

如果您没有在用于代理部署的自定义资源(CR)实例中为 adminUser 和 adminPassword 指定值，Operator 会自动生成这些凭证并将其存储在 secret 中。默认 secret 名称的格式为 <custom-resource-name>-credentials-secret，如 my-broker-deployment-credentials-secret。



注意

只有在 CR 的 requireLogin 参数设为 true 时，才需要 adminUser 和 adminPassword 的值才能登录到管理控制台。

如果 requireLogin 设置为 false，您可以在提示输入用户名和密码时输入任何文本来登录控制台，而无需提供有效的用户名密码。

此流程演示了如何访问登录凭证。

流程

1. 查看 OpenShift 项目中机密的完整列表。

   a. 在 OpenShift Container Platform web 控制台中点 Workload → Secrets。

   b. 在命令行中：

   ```
   $ oc get secrets
   ```

2. 打开适当的 secret，以显示 Base64 编码的控制台登录凭证。

   a. 在 OpenShift Container Platform web 控制台中点名称中包含代理自定义资源实例的 secret。点 YAML 标签。

   b. 在命令行中：

   ```
   $ oc edit secret <my-broker-deployment-credentials-secret>
   ```

3. 要解码 secret 中的值，请使用类似如下的命令：

   ```
   $ echo 'dXNlcl9uYW1l' | base64 --decode
   console_admin
   ```

其它资源

- 如需了解更多有关使用 AMQ Management Console 查看和管理代理的信息，请参阅在 管理 AMQ Broker 中使用 AMQ Management Console 管理代理

第 6 章 升级基于 OPERATOR 的代理部署

本节中的步骤显示如何升级：

- 使用 OpenShift 命令行界面(CLI)和 OperatorHub 的 AMQ Broker Operator 版本
- 基于 Operator 的代理部署的代理容器镜像

6.1. 开始前

本节介绍了在为基于 Operator 的代理部署升级 Operator 和代理容器镜像前的一些重要注意事项。

- 要将 OpenShift Container Platform 3.11 上运行的基于 Operator 的代理部署升级为在 OpenShift Container Platform 4.5 或更高版本中运行，您必须先升级 OpenShift Container Platform 安装。然后，您必须创建一个与现有部署匹配的基于 Operator 的代理部署。要了解如何创建新基于 Operator 的代理部署，请参阅 第 3 章 使用 AMQ Broker Operator 在 OpenShift Container Platform 上部署 AMQ Broker。
- 使用 OpenShift 命令行界面(CLI)或 OperatorHub 升级 Operator 需要 OpenShift 集群的集群管理员特权。
- 如果您最初使用 CLI 安装 Operator，则还应使用 CLI 来 升级 Operator。如果您最初使用 OperatorHub 来安装 Operator（也就是说，它在 OpenShift Container Platform Web 控制台中项目的 Operators → Installed Operators 下出现），您还应使用 OperatorHub 来升级 Operator。有关这些升级方法的详情，请参考：
  - 第 6.2 节 “使用 CLI 升级 Operator”
  - 第 6.3.3 节 “使用 OperatorHub 升级 Operator”
- 如果要部署 Operator 来监视多个命名空间，例如要监视所有命名空间，您必须：
  1. 确保您已备份了与集群中代理部署相关的所有 CR。

2. 卸载现有 Operator。

3. 部署 7.9 Operator 以观察您需要的命名空间。

4. 检查所有部署并在需要时重新创建。

6.2. 使用 CLI 升级 OPERATOR

本节中的步骤演示了如何使用 OpenShift 命令行界面(CLI)将 Operator 的不同版本升级到 AMQ Broker 7.9 提供的最新版本。

6.2.1. 先决条件

- 只有在最初使用 CLI 安装 Operator 时，才应使用 CLI 来升级 Operator。如果您最初使用 OperatorHub 来安装 Operator（即，Operator 会出现在 OpenShift Container Platform Web 控制台中项目的 Operators → Installed Operators 下），您应该使用 OperatorHub 来升级 Operator。要了解如何使用 OperatorHub 升级 Operator，请参阅 第 6.3 节 “使用 OperatorHub 升级 Operator”。

6.2.2. 升级 Operator 的 7.8.x 版本

此流程演示了如何使用 OpenShift 命令行界面(CLI)将 Operator 的 7.8.x 升级到 AMQ Broker 7.9 的最新版本。

流程

1. 在网页浏览器中，导航到 AMQ Broker 7.9.3 补丁的 Software Downloads 页面。

2. 确保将 Version 下拉列表的值设置为 7.9.3，并且选择了 Patches 选项卡。

3. 在 AMQ Broker 7.9.3 Operator 安装和示例文件旁边，点 Download。

   下载 amq-broker-operator-7.9.3-ocp-install-examples.zip 压缩存档会自动开始。

4. 下载完成后，将存档移动到您选择的安装目录中。以下示例将存档移至名为

~/broker/operator 的目录。

```
mkdir ~/broker/operator
mv amq-broker-operator-7.9.3-ocp-install-examples.zip ~/broker/operator
```

5. *在您选择的安装目录中，提取 存档的内容。例如：*

```
cd ~/broker/operator
unzip amq-broker-operator-7.9.3-ocp-install-examples.zip
```

6. *以包含现有 Operator 部署的项目的管理员身份登录到 OpenShift Container Platform。*

```
$ oc login -u <user>
```

7. *切换到您要在其中升级 Operator 版本的 OpenShift 项目。*

```
$ oc project <project-name>
```

8. *在您下载并提取的最新 Operator 存档的 deploy 目录中，打开 operator.yaml 文件。*

注意

在 operator.yaml 文件中，Operator 使用由 Secure Hash Algorithm (SHA)值表示的镜像。注释行以数字符号(#)符号开头，表示 SHA 值与特定的容器镜像标签对应。

9. *为 之前的 Operator 部署打开 operator.yaml 文件。检查您在先前配置中指定的任何非默认值是否 在新的 operator.yaml 配置文件中复制。*

10. *如果您 对新 operator.yaml 文件进行了任何更新，请保存该文件。*

11. *应用更新的 Operator 配置。*

```
$ oc apply -f deploy/operator.yaml
```

OpenShift 更新项目以使用最新的 Operator 版本。

12. 要重新创建以前的代理部署，创建一个新的 CR yaml 文件以匹配原始 CR 的用途并应用它。第 3.4.1 节 “部署基本代理实例” 描述了如何在 Operator 安装存档中应用 deploy/crs/broker_activemqartemis_cr.yaml 文件，您可以使用该文件作为新 CR yaml 文件的基础。

6.3. 使用 OPERATORHUB 升级 OPERATOR

本节论述了如何使用 OperatorHub 将 Operator 的不同版本升级到 AMQ Broker 7.9 提供的最新版本。

6.3.1. 先决条件

- 只有在最初使用 OperatorHub 安装 Operator 时，才应使用 OperatorHub 升级 Operator（即，Operator 在 OpenShift Container Platform Web 控制台中的 Operators → Installed Operators 下会出现在项目的 Operators Installed Operators 下）。相反，如果您最初使用 OpenShift 命令行界面(CLI)来安装 Operator，则还应使用 CLI 来升级 Operator。要了解如何使用 CLI 升级 Operator，请参阅 第 6.2 节 “使用 CLI 升级 Operator”。

- 使用 OperatorHub 升级 AMQ Broker Operator 需要 OpenShift 集群的集群管理员特权。

6.3.2. 开始前

本节论述了使用 OperatorHub 升级 AMQ Broker Operator 实例前的一些重要注意事项。

- 从 OperatorHub 安装最新的 Operator 版本时，Operator Lifecycle Manager 会自动 更新 OpenShift 集群中的 CRD。您不需要删除现有的 CRD。

- 当您使用最新 Operator 版本的 CRD 更新集群时，此更新会影响 集群中的所有 项目。从以前版本的 Operator 部署的所有代理 Pod 可能无法在 OpenShift Container Platform Web 控制台中更新其状态。当您点击正在运行的代理 Pod 的 Logs 选项卡时，您会看到指出"UpdatePodStatus"失败的信息。但是，该项目中的 broker Pod 和 Operator 会象预期一样工作。要修复受影响的项目的此问题，还必须将该项目升级为使用最新版本的 Operator。

6.3.3. 使用 OperatorHub 升级 Operator

此流程演示了如何使用 OperatorHub 升级 AMQ Broker Operator 的实例。

流程

1. *以集群管理员身份登录 OpenShift Container Platform Web 控制台。*
2. *删除项目中代理部署的主要自定义资源(CR)实例。此操作将删除代理部署。*
   a. *在左侧导航菜单中，点击 Administration → Custom Resource Definitions。*
   b. *在 Custom Resource Definitions 页面上，单击 ActiveMQArtemis CRD。*
   c. *点 实例 选项卡。*
   d. *查找与项目命名空间对应的 CR 实例。*
   e. *对于 CR 实例，点击右侧的 More Options 图标（三个垂直点）。选择 Delete ActiveMQArtemis。*
3. *从项目卸载现有的 AMQ Broker Operator。*
   a. *在左侧导航菜单中点 Operators → Installed Operators。*
   b. *在页面顶部的 Project 下拉菜单中选择您要卸载 Operator 的项目。*
   c. *找到您要卸载的 Red Hat Integration - AMQ Broker 实例。*
   d. *对于 Operator 实例，点击右侧的 More Options 图标（三个垂直点）。点击 Uninstall Operator。*

e. 在确认对话框中点击 **Uninstall**。

4. 使用 OperatorHub 为 AMQ Broker 7.9 安装 Operator 的最新版本。如需更多信息，请参阅 第 3.3.2 节 “从 OperatorHub 部署 Operator”。

5. 要重新创建以前的代理部署，创建一个新的 CR yaml 文件以匹配原始 CR 的用途并应用它。第 3.4.1 节 “部署基本代理实例” 描述了如何在 Operator 安装存档中应用 **deploy/crs/broker_activemqartemis_cr.yaml** 文件，您可以使用该文件作为新 CR yaml 文件的基础。

6.4. 通过指定 AMQ BROKER 版本来升级代理容器镜像

以下流程演示了如何通过指定 AMQ Broker 版本来为基于 Operator 的代理部署升级代理容器镜像。例如，您可以将 Operator 升级到 AMQ Broker 7.9.3 的最新版本，但 CR 中的 **spec.upgrades.enabled** 属性已被设为 **true**，**spec.version** 属性指定 7.8.0。要升级 代理容器镜像，您需要手动指定一个新的 AMQ Broker 版本（如 7.9.3）。当您指定新版本的 AMQ Broker 时，Operator 会自动选择与此版本对应的代理容器镜像。

先决条件

- 您必须对 7.9.3 使用最新版本的 Operator。要了解如何将 Operator 升级到最新版本，请参阅：
  - 第 6.2 节 “使用 CLI 升级 Operator”
  - 第 6.3.3 节 “使用 OperatorHub 升级 Operator”.
- 如 第 2.4 节 “Operator 如何选择容器镜像” 所述，如果您部署 CR 且没有明确指定代理容器镜像，Operator 会自动选择要使用的相应容器镜像。要使用本节中描述的升级过程，您必须使用这个默认行为。如果您通过直接在 CR 中指定代理容器镜像来覆盖默认行为，Operator 无法 自动升级代理容器镜像以对应 AMQ Broker 版本，如下所述。

流程

1. 编辑代理部署的主代理 CR 实例。

a. 使用 OpenShift 命令行界面：

i. 以具有特权的用户身份登录 OpenShift，以便在用于代理部署的项目中编辑和部署 CR。

```
$ oc login -u <user> -p <password> --server=<host:port>
```

ii. 在文本编辑器中，打开用于代理部署的 CR 文件。例如，这可能是之前下载并提取的 Operator 安装存档 deploy/ crs 目录中中包含的 broker_activemqartemis_cr.yaml 文件。

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有权限的用户身份登录到控制台，以便在用于代理部署的项目中编辑和部署 CR。

ii. 在左侧窗格中，单击 Administration → Custom Resource Definitions。

iii. 单击 ActiveMQArtemis CRD。

iv. 点 实例 选项卡。

v. 查找与项目命名空间对应的 CR 实例。

vi. 对于 CR 实例，点击右侧的 More Options 图标（三个垂直点）。选择 Edit ActiveMQArtemis。

在控制台中，会打开 YAML 编辑器，供您编辑 CR 实例。

2. 要指定要将代理容器镜像升级到的 AMQ Broker 版本，请为 CR 的 spec.version 属性设置值。例如：

```
spec:
  version: 7.9.3
  ...
```

3. 在 CR 的 spec 部分，找到 upgrade 部分。如果此部分尚未包含在 CR 中，请添加它。

```
spec:
  version: 7.9.3
  ...
  upgrades:
```

4. 确保 upgrade 部分 包含 enabled 和副 属性。

```
spec:
  version: 7.9.3
  ...
  upgrades:
    enabled:
    minor:
```

5. 要根据 AMQ Broker 的指定版本启用代理容器镜像升级，请将 enabled 属性的值设置为 true。

```
spec:
  version: 7.9.3
  ...
  upgrades:
    enabled: true
    minor:
```

6. 要定义代理的升级行为，请为 次 属性设置一个值。

   a. 要允许在 次 AMQ Broker 版本间升级，请将 minor 值设为 true。

```
spec:
  version: 7.9.0
  ...
  upgrades:
    enabled: true
    minor: true
```

例如，假设当前代理容器镜像对应于 7.8.0，并且有与为 spec.version 指定的 7.9.0 版本对应的新镜像。在本例中，Operator 确定在 7.8 和 7.9 次版本间有可用的升级。根据上述

设置（允许在次版本间升级），Operator 升级代理容器镜像。

相反，假设当前代理容器镜像与 7.9.0 对应，并且为 spec.version 指定 了一个新 值 7.9.1。如果存在与 7.9.1 对应的镜像，Operator 会确定在 7.9.0 和 7. 9.1 微版本间有可用的升级。根据以上设置，仅允许在次版本间升级，Operator 不会 升级代理容器镜像。

b. 要允许在 微 AMQ Broker 版本间升级，请将 minor 值设为 false。

```
spec:
  version: 7.9.0
  ...
  upgrades:
    enabled: true
    minor: false
```

例如，假设当前代理容器镜像对应于 7.8.0，并且有与为 spec.version 指定的 7.9.0 版本对应的新镜像。在本例中，Operator 确定在 7.8 和 7.9 次版本间有可用的升级。根据前面的设置，这些设置不允许在次版本间升级（即仅在微版本间升级），Operator 不会升级 代理容器镜像。

相反，假设当前代理容器镜像与 7.9.0 对应，并且为 spec.version 指定 了一个新 值 7.9.1。如果存在与 7.9.1 对应的镜像，Operator 会确定在 7.9.0 和 7. 9.1 微版本间有可用的升级。根据上述设置（允许在微版本间升级），Operator 升级代理容器镜像。

7. 将更改应用到 CR。

a. 使用 OpenShift 命令行界面：

i. 保存 CR 文件。

ii. 切换到代理部署的项目。

```
$ oc project <project_name>
```

iii. 应用 CR。

```
$ oc apply -f <path/to/broker_custom_resource_instance>.yaml
```

b. 使用 OpenShift Web 控制台：

i. 编辑完 CR 后，点击 Save。

应用 CR 更改时，Operator 首先验证现有部署是否提供了对 spec.version 指定的 AMQ Broker 版本升级。如果您已指定要升级到的 AMQ Broker 的无效版本（例如，尚未可用的版本），Operator 会记录警告信息，并且不再执行进一步操作。

但是，如果升级到指定版本，且为 upgrade .enabled 和 upgrade .minor 指定的值允许升级，则 Operator 升级部署中的每个代理以使用与新的 AMQ Broker 版本对应的代理容器镜像。

Operator 使用的代理容器镜像在 Operator 部署的 operator.yaml 配置文件中的环境变量中定义。环境变量名称包含 AMQ Broker 版本的标识符。例如，环境变量 RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_791 对应于 AMQ Broker 7.9.1。

当 Operator 应用了 CR 更改时，它会重启部署中的每个代理 Pod，以便每个 Pod 使用指定的镜像版本。如果您在部署中有多个代理，一次只能有一个代理 Pod 关闭并重启。

其它资源

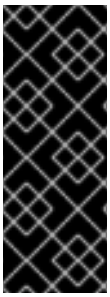
- 要了解 Operator 如何使用环境变量选择代理容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。

第 7 章 监控代理

7.1. 在 FUSE 控制台中查看代理

您可以配置基于 Operator 的代理部署，以将 Fuse Console 用于 OpenShift 而不是 AMQ 管理控制台。当您正确配置代理部署后，Fuse 控制台会发现代理并将其显示在专用的 Artemis 选项卡上。您可以查看 AMQ 管理控制台中相同的代理运行时数据。您还可以执行相同的基本管理操作，如创建地址和队列。

以下流程描述了如何为代理部署配置自定义资源(CR)实例，以便 OpenShift 的 Fuse 控制台在部署中发现并显示代理。



重要

从 Fuse 控制台查看代理只是一个技术预览功能。技术预览功能不被红帽产品服务等级协议 (SLA) 支持，且可能在功能方面有缺陷。红帽不推荐在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。有关红帽技术预览功能支持范围的更多信息，请参阅 技术预览功能支持范围。

先决条件

- OpenShift 的 Fuse 控制台必须部署到 OCP 集群，或部署到该集群上的特定命名空间。如果您已将控制台部署到特定命名空间中，代理部署必须 位于同一 命名空间中，以便控制台能够发现代理。否则，只需要将 Fuse 控制台和代理部署到同一 OCP 群集上即可。有关在 OCP 上安装 Fuse Online 的更多信息，请参阅在 OpenShift Container Platform 上安装和操作 Fuse Online。

- 您必须已创建了代理部署。例如，要了解如何使用自定义资源(CR)实例创建基于 Operator 的基本部署，请参阅 第 3.4.1 节 “部署基本代理实例”。

流程

1. 打开用于代理部署的 CR 实例。例如，基本部署的 CR 可能类似如下：

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
```

```
      size: 4
      image: registry.redhat.io/amq7/amq-broker-rhel8:7.9
      ...
```

2. 在 deploymentPlan 部分中，添加 jolokiaAgentEnabled 和 managementRBACEnabled 属性并指定值，如下所示：

```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
      size: 4
      image: registry.redhat.io/amq7/amq-broker-rhel8:7.9
      ...
      jolokiaAgentEnabled: true
      managementRBACEnabled: false
```

jolokiaAgentEnabled

指定 Fuse 控制台是否可以在部署中发现并显示代理的运行时数据。要使用 Fuse Console，请将值设为 true。

managementRBACEnabled

指定是否为部署中的代理启用了基于角色的访问控制(RBAC)。您必须将 值设为 false 才能使用 Fuse 控制台，因为 Fuse 控制台使用其基于角色的访问控制。

重要

如果您将 managementRBACEnabled 的值设置为 false 来启用 Fuse 控制台，则代理的管理 MBeans 不再需要授权。您 不应该 使用 AMQ 管理控制台，而 managementRBACEnabled 被设置为 false，因为这可能导致代理上的所有管理操作暴露给未经授权的使用。

3. 保存 CR 实例。

4. 切换到之前创建代理部署的项目。

```
$ oc project <project_name>
```

5. 在命令行中应用更改。

```
$ oc apply -f <path/to/custom_resource_instance>.yaml
```

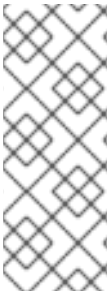
6. 在 Fuse 控制台中，若要查看 Fuse 应用程序，请单击 Online 选项卡。要查看正在运行的代理，请在左侧导航菜单中点击 Artemis。

其它资源

- 有关为 OpenShift 使用 Fuse 控制台的更多信息，请参阅在 OpenShift 上监控和管理红帽 Fuse 应用程序。
- 要了解使用 AMQ 管理控制台查看和管理代理的方式在 Fuse 控制台中，请参阅使用 AMQ 管理控制台管理代理。

7.2. 使用 PROMETHEUS 监控代理运行时指标

下面的部分描述了如何在 OpenShift Container Platform 上为 AMQ Broker 配置 Prometheus 指标插件。您可以使用插件来监控和存储代理运行时指标。您还可以使用 Grafana 等图形工具来配置 Prometheus 插件收集的数据的更多高级视觉化和仪表板。



注意

Prometheus metrics 插件允许您以 Prometheus 格式 收集和导出代理指标。但是，红帽 不支持 安装或配置 Prometheus 本身，也不支持 Grafana 等视觉化工具。如果您需要安装、配置或运行 Prometheus 或 Grafana 的支持，请访问产品网站以获取社区支持和文档等资源。

7.2.1. 指标概述

要监控代理实例的健康状况和性能，您可以使用 AMQ Broker 的 Prometheus 插件来监控和存储代理运行时指标。AMQ Broker Prometheus 插件将代理运行时指标导出到 Prometheus 格式，允许您使用 Prometheus 本身来视觉化并运行数据查询。

您还可以使用图形工具（如 Grafana）为 Prometheus 插件收集的指标配置更高级的视觉化和仪表板。

插件导出到 Prometheus 格式的指标如下所述。

代理指标

artemis_address_memory_usage

此代理上所有地址用于内存中消息的字节数。

artemis_address_memory_usage_percentage

此代理上所有地址使用的内存作为 global-max-size 参数的百分比。

artemis_connection_count

连接到此代理的客户端数。

artemis_total_connection_count

自这个代理启动以来已连接到此代理的客户端数。

地址指标

artemis_routed_message_count

路由到一个或多个队列绑定的消息数量。

artemis_unrouted_message_count

未 路由到任何队列绑定的消息数。

队列指标

artemis_consumer_count

来自给定队列使用消息的客户端数量.

artemis_delivering_durable_message_count

给定队列当前传送给消费者的持久消息数量。

artemis_delivering_durable_persistent_size

给定队列正在提供给消费者的持久消息的持久大小。

artemis_delivering_message_count

给定队列当前传送给消费者的消息数量。

artemis_delivering_persistent_size

给定队列当前传送给消费者的持久信息大小。

artemis_durable_message_count

当前在给定队列中持久消息的数量。这包括调度、分页和内交付消息。

artemis_durable_persistent_size

目前在给定队列中持久化消息的持久大小.这包括调度、分页和内交付消息。

artemis_messages_acknowledged

创建队列后从给定队列确认的消息数。

artemis_messages_added

自创建队列以来添加到给定队列的消息数。

artemis_message_count

当前在给定队列中的消息数.这包括调度、分页和内交付消息。

artemis_messages_killed

创建队列后从给定队列中删除的消息数。当消息超过配置的最大发送尝试数时，代理会终止消息。

artemis_messages_expired

自创建队列以来，来自给定队列的消息数量已过期。

artemis_persistent_size

给定队列中所有消息（持久和非持久）的持久大小。这包括调度、分页和内交付消息。

artemis_scheduled_durable_message_count

给定队列中持久、调度的消息数量。

artemis_scheduled_durable_persistent_size

给定队列中持久、调度消息的持久大小.

artemis_scheduled_message_count

给定队列中调度的消息数.

artemis_scheduled_persistent_size

给定队列中调度消息的持久大小.

对于上方未列出的更高级别代理指标，您可以通过聚合较低级别指标来计算这些指标。例如，若要计算总消息数，您可以聚合代理部署中的所有队列中的 artemis_message_count 指标。

对于 AMQ Broker 的内部部署，托管代理的 Java 虚拟机(JVM)的指标也会导出到 Prometheus 格式。这不适用于在 OpenShift Container Platform 上部署 AMQ Broker。

7.2.2. 使用 CR 启用 Prometheus 插件

安装 AMQ Broker 时，安装中包含 Prometheus 指标插件。启用后，插件会收集代理的运行时指标，并将其导出到 Prometheus 格式。

以下流程演示了如何使用 CR 为 AMQ Broker 启用 Prometheus 插件。此流程支持 AMQ Broker 7.9 或更高版本的新和现有部署。

有关运行代理的替代步骤，请参阅 第 7.2.3 节 “使用环境变量为正在运行的代理部署启用 Prometheus 插件”。

流程

1. 打开用于代理部署的 CR 实例。例如，基本部署的 CR 可能类似如下：

```
apiVersion: broker.amq.io/v2alpha5
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
        size: 4
        image: registry.redhat.io/amq7/amq-broker-rhel8:7.9
        ...
```

2. 在 deployment Plan 部分中，添加 enableMetricsPlugin 属性，并将值设为 true，如下所示：

```
apiVersion: broker.amq.io/v2alpha5
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
        size: 4
        image: registry.redhat.io/amq7/amq-broker-rhel8:7.9
        ...
        enableMetricsPlugin: true
```

enableMetricsPlugin

指定是否在部署中为代理启用了 Prometheus 插件。

3. 保存 CR 实例。

4. 切换到之前创建代理部署的项目。

```
$ oc project <project_name>
```

5. 在命令行中应用更改。

```
$ oc apply -f <path/to/custom_resource_instance>.yaml
```

指标插件开始以 Prometheus 格式收集代理运行时指标。

其它资源

- 有关更新正在运行的代理的详情请参考 第 3.4.3 节 “将自定义资源更改应用到运行代理部署”。

7.2.3. 使用环境变量为正在运行的代理部署启用 Prometheus 插件

以下流程演示了如何使用环境变量为 AMQ Broker 启用 Prometheus 插件。有关备选步骤，请参阅

第 7.2.2 节 “使用 CR 启用 Prometheus 插件”。

先决条件

- 您可以为使用 AMQ Broker Operator 创建的代理 Pod 启用 Prometheus 插件。但是，您部署的代理必须为 AMQ Broker 7.7 或更高版本使用代理容器镜像。

流程

1. 使用包含代理部署的项目的管理员权限登录到 OpenShift Container Platform Web 控制台。
2. 在 Web 控制台中，单击 Home → Projects。选择包含代理部署的项目。
3. 要查看项目中的 StatefulSets 或 DeploymentConfig，请点击 Workloads → StatefulSets 或 Workloads → DeploymentConfig。
4. 点与代理部署对应的 StatefulSet 或 DeploymentConfig。
5. 要访问代理部署的环境变量，请点击 Environment 选项卡。
6. 添加新的环境变量 AMQ_ENABLE_METRICS_PLUGIN。将 变量的值设为 true。

当您设置 AMQ_ENABLE_METRICS_PLUGIN 环境变量时，OpenShift 会重启 StatefulSet 或 DeploymentConfig 中的每个代理 Pod。当部署中有多个 Pod 时，OpenShift 将依次重新启动每个 Pod。当每个代理 Pod 重启时，代理的 Prometheus 插件开始收集代理运行时指标。

7.2.4. 访问正在运行的代理 Pod 的 Prometheus 指标

此流程演示了如何访问正在运行的代理 Pod 的 Prometheus 指标。

先决条件

- 您必须已为您的代理 Pod 启用了 Prometheus 插件。请参阅 第 7.2.3 节 “使用环境变量为正在运行的代理部署启用 Prometheus 插件”。

流程

1. 对于您要访问的指标的代理 Pod，您需要标识之前创建的路由，将 Pod 连接到 AMQ Broker 管理控制台。Route 名称构成了访问指标所需的 URL 的一部分。

   a. 单击 **Networking → Routes**。

   b. 对于所选代理 Pod，标识为将 Pod 连接到 AMQ Broker 管理控制台而创建的路由。在 **Hostname** 下，记下显示的完整 URL。例如：

   ```
   http://rte-console-access-pod1.openshiftdomain
   ```

2. 要访问 **Prometheus** 指标，在网页浏览器中输入之前记录的路由名称并附加 "/metrics"。例如：

   ```
   http://rte-console-access-pod1.openshiftdomain/metrics
   ```

注意

如果您的控制台配置不使用 SSL，请在 URL 中指定 http。在本例中，主机名的 DNS 解析将流量定向到 OpenShift 路由器的端口 80。如果您的控制台配置使用 SSL，请在 URL 中指定 https。在本例中，您的浏览器默认为 OpenShift 路由器的端口 443。如果 OpenShift 路由器也将端口 443 用于 SSL 流量，这允许成功连接控制台，路由器默认也使用端口 443。

7.3. 使用 JMX 监控代理运行时数据

本例演示了如何使用 Jolokia REST 接口对 JMX 监控代理。

先决条件

- 建议完成 基本代理部署。

流程

1. 获取正在运行的 pod 列表：

   ```
   $ oc get pods
   ```

```
NAME              READY    STATUS    RESTARTS   AGE
ex-aao-ss-1   1/1      Running   0          14d
```

2. *运行 **oc logs** 命令：*

```
$ oc logs -f ex-aao-ss-1

...
Running Broker in /home/jboss/amq-broker
...
2021-09-17 09:35:10,813 INFO  [org.apache.activemq.artemis.integration.bootstrap]
AMQ101000: Starting ActiveMQ Artemis Server
2021-09-17 09:35:10,882 INFO  [org.apache.activemq.artemis.core.server] AMQ221000: live
Message Broker is starting with configuration Broker Configuration
(clustered=true,journalDirectory=data/journal,bindingsDirectory=data/bindings,largeMessagesDi
rectory=data/large-messages,pagingDirectory=data/paging)
2021-09-17 09:35:10,971 INFO  [org.apache.activemq.artemis.core.server] AMQ221013:
Using NIO Journal
2021-09-17 09:35:11,114 INFO  [org.apache.activemq.artemis.core.server] AMQ221057:
Global Max Size is being adjusted to 1/2 of the JVM max size (-Xmx). being defined as
2,566,914,048
2021-09-17 09:35:11,369 WARNING [org.jgroups.stack.Configurator] JGRP000014:
BasicTCP.use_send_queues has been deprecated: will be removed in 4.0
2021-09-17 09:35:11,385 WARNING [org.jgroups.stack.Configurator] JGRP000014:
Discovery.timeout has been deprecated: GMS.join_timeout should be used instead
2021-09-17 09:35:11,480 INFO  [org.jgroups.protocols.openshift.DNS_PING] serviceName
[ex-aao-ping-svc] set; clustering enabled
2021-09-17 09:35:24,540 INFO  [org.openshift.ping.common.Utils] 3 attempt(s) with a
1000ms sleep to execute [GetServicePort] failed. Last failure was
[javax.naming.CommunicationException: DNS error]
...
2021-09-17 09:35:25,044 INFO  [org.apache.activemq.artemis.core.server] AMQ221034:
Waiting indefinitely to obtain live lock
2021-09-17 09:35:25,045 INFO  [org.apache.activemq.artemis.core.server] AMQ221035:
Live Server Obtained live lock
2021-09-17 09:35:25,206 INFO  [org.apache.activemq.artemis.core.server] AMQ221080:
Deploying address DLQ supporting [ANYCAST]
2021-09-17 09:35:25,240 INFO  [org.apache.activemq.artemis.core.server] AMQ221003:
Deploying ANYCAST queue DLQ on address DLQ
2021-09-17 09:35:25,360 INFO  [org.apache.activemq.artemis.core.server] AMQ221080:
Deploying address ExpiryQueue supporting [ANYCAST]
2021-09-17 09:35:25,362 INFO  [org.apache.activemq.artemis.core.server] AMQ221003:
Deploying ANYCAST queue ExpiryQueue on address ExpiryQueue
2021-09-17 09:35:25,656 INFO  [org.apache.activemq.artemis.core.server] AMQ221020:
Started EPOLL Acceptor at ex-aao-ss-1.ex-aao-hdls-svc.broker.svc.cluster.local:61616 for
protocols [CORE]
2021-09-17 09:35:25,660 INFO  [org.apache.activemq.artemis.core.server] AMQ221007:
Server is now live
2021-09-17 09:35:25,660 INFO  [org.apache.activemq.artemis.core.server] AMQ221001:
Apache ActiveMQ Artemis Message Broker version 2.16.0.redhat-00022 [amq-broker,
nodeID=8d886031-179a-11ec-9e02-0a580ad9008b]
2021-09-17 09:35:26,470 INFO  [org.apache.amq.hawtio.branding.PluginContextListener]
Initialized amq-broker-redhat-branding plugin
```

```
2021-09-17 09:35:26,656 INFO  [org.apache.activemq.hawtio.plugin.PluginContextListener]
Initialized artemis-plugin plugin
...
```

3. *运行查询来监控* ***MaxConsumers*** *的代理：*

```
$ curl -k -u admin:admin http://console-broker.amq-
demo.apps.example.com/console/jolokia/read/org.apache.activemq.artemis:broker=%22broker
%22,component=addresses,address=%22TESTQUEUE%22,subcomponent=queues,routing-
type=%22anycast%22,queue=%22TESTQUEUE%22/MaxConsumers

{"request":
{"mbean":"org.apache.activemq.artemis:address=\"TESTQUEUE\",broker=\"broker\",compone
nt=addresses,queue=\"TESTQUEUE\",routing-
type=\"anycast\",subcomponent=queues","attribute":"MaxConsumers","type":"read"},"value":-
1,"timestamp":1528297825,"status":200}
```

第 8 章 REFERENCE

8.1. 自定义资源配置参考

自定义资源定义(CRD)是使用 Operator 部署的自定义 OpenShift 对象的配置项的 schema。通过部署对应的自定义资源(CR)实例，您可以为 CRD 中显示的配置项目指定值。

以下子部分详细描述了可根据主代理 CRD 在自定义资源实例中设置的配置项。

8.1.1. 代理自定义资源配置参考

基于主代理 CRD 的 CR 实例允许您配置代理以便在 OpenShift 项目中部署。下表描述了您可以在 CR 实例中配置的项目。

重要

您部署的任何对应自定义资源(CR)中都需要标记为星号(*)的配置项目。如果没有为非必需项目明确指定值，配置将使用默认值。

条目	子条目	描述和使用
adminUser*		连接到代理和管理控制台所需的管理员用户名。 如果没有指定值，则该值会自动生成并存储在 secret 中。默认 secret 名称的格式为 ***<custom_resource_name>*-credentials-secret**。例如，my-**broker-deployment-credentials-secret**。 **类型** ：字符串 **示例** ：my-user **默认值** ：自动生成的随机值

条目	子条目	描述和使用
adminPassword*		连接到代理和管理控制台所需的管理员密码。 如果没有指定值，则该值会自动生成并存储在 secret 中。默认 secret 名称的格式为 ***<custom_resource_name>*-credentials-secret**。例如，my-**broker-deployment-credentials-secret**。 **类型** ：字符串 **示例** ：my-password **默认值** ：自动生成的随机值
deploymentPlan*		代理部署配置
	image*	在部署中用于每个代理的代理容器镜像的完整路径。 您不需要为 CR **中的镜像**显式指定值。**占位符** 的默认值表示 Operator 尚未决定要使用的适当镜像。 要了解 Operator 如何选择要使用的代理容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。 **类型** ：字符串 **示例**: registry.redhat.io/amq-broker-rhel8@sha256:979b59325aa0f34eb05625201beba53fccbb83bd5eb80a89dcb5261ae358138f **默认值** ：占位符

条目	子条目	描述和使用
	Size*	要在部署中创建的代理 Pod 数量。 如果您指定 2 个或更高级别的值，则默认集群代理部署。默认情况下，集群用户名和密码会自动生成并存储在与 **adminUser 和 admin Password** 相同的机密中。 **类型** ：int **示例** ：1 **默认值** ：2
	requireLogin	指定是否需要登录凭证来连接到代理。 **类型** ：布尔值 **示例** ： false **默认值为** ： true
	persistenceEnabled	指定是否将日志存储用于部署中的每个代理 Pod。如果设置为 **true**，每个代理 Pod 需要一个可用的持久性卷(PV)，Operator 可以使用持久性卷声明(PVC)声明。 **类型** ：布尔值 **示例** ： false **默认值为** ： true

条目	子条目	描述和使用
	initImage	用于配置代理的 init 容器镜像。 您不需要在 CR 中明确指定 **initImage** 的值，除非您要提供自定义镜像。 要了解 Operator 如何选择要使用的内置初始容器镜像，请参阅 第 2.4 节 “Operator 如何选择容器镜像”。 要了解如何指定 *自定义初始* 容器镜像，请参阅 第 4.6 节 “指定自定义初始容器镜像”。 **类型** ：字符串 **示例**: registry.redhat.io/amq-broker-init-rhel8@sha256:b74d03ed852a3731467ffda95266ce49f2065972f1c37bf254f3d52b34c11991 **默认值** ：未指定
	journalType	指定是否使用异步 I/O(AIO)还是非阻塞 I/O(NIO)。 **类型** ：字符串 **示例** ：aio **默认值为** ：nio
	messageMigration	当代理 Pod 因代理部署失败或有意缩减而关闭时，指定是否将消息迁移到仍然在代理集群中运行的另一代理 Pod。 **类型** ：布尔值 **示例** ： false **默认值为** ： true

条目	子条目	描述和使用
	resources.limits.cpu	部署中 Pod 中运行的每个代理容器可以消耗的最大 host-node CPU 数量，以 millicore 为单位。 **类型** ：字符串 **示例** ："500m" **默认值** ：使用与您的 OpenShift Container Platform 版本相同的默认值。咨询集群管理员.
	resources.limits.memory	部署中 Pod 中运行的每个代理容器可以消耗的最大主机节点内存量，以字节为单位。支持字节表示法（如 K、M、G）或二进制等效（Ki、Mi、Gi）。 **类型** ：字符串 **示例** ："1024M" **默认值** ：使用与您的 OpenShift Container Platform 版本相同的默认值。咨询集群管理员.
	resources.requests.cpu	在部署中在 Pod 中运行的每个代理容器以 millicores 为单位的主机节点 CPU 量。 **类型** ：字符串 **示例** ："250m" **默认值** ：使用与您的 OpenShift Container Platform 版本相同的默认值。咨询集群管理员.

条目	子条目	描述和使用
	resources.requests.memory	在部署中运行的 Pod 中运行的每个代理容器以字节为单位的主机节点内存量。支持字节表示法（如 K、M、G）或二进制等效（Ki、Mi、Gi）。 **类型** ：字符串 **示例** ："512M" **默认值** ：使用与您的 OpenShift Container Platform 版本相同的默认值。咨询集群管理员.
	storage.size	部署中每个代理所需的持久性卷声明(PVC)大小（以字节为单位）。此属性仅在 **persistenceEnabled** 设为 **true** 时应用。您指定的值 **必须** 包含一个单元。支持字节表示法（如 K、M、G）或二进制等效（Ki、Mi、Gi）。 **类型** ：字符串 **示例** ：4Gi **默认值** ：2Gi
	jolokiaAgentEnabled	指定是否为部署中的代理启用了 Jolokia JVM Agent。如果此属性的值设为 **true**，Fuse Console 可以发现并显示代理的运行时数据。 **类型** ：布尔值 **示例** ： true **默认值为** ： false

条目	子条目	描述和使用
	managementRBACEnabled	指定是否为部署中的代理启用了基于角色的访问控制 (RBAC)。要使用 Fuse 控制台，**您必须将** 值设置为 **false**，因为 Fuse 控制台使用其基于角色的访问控制。 **类型** ：布尔值 **示例** ： false **默认值为** ： true
console		代理管理控制台的配置.
	expose	指定是否在部署中为每个代理公开管理控制台端口。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	sslEnabled	指定是否在管理控制台端口中使用 SSL。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	sslSecret	代理密钥存储、信任存储及其相应密码（所有 Base64 编码）的机密。如果没有为 **sslSecret** 指定值，控制台将使用默认的 secret 名称。默认 secret 名称的格式为 ***<custom_resource_name>*-console-secret**。 **类型** ：字符串 **示例** ：my-broker-deployment-console-secret **默认值** ：未指定

条目	子条目	描述和使用
	useClientAuth	指定管理控制台是否需要客户端授权。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
acceptors.acceptor		单个接收器配置实例。
	name*	接收器名称. **类型** ：字符串 **示例** ：my-acceptor **默认值** ：Not applicable
	port	用于接收器实例的端口号。 **类型** ：int **示例** ：5672 **默认值** ：61626 用于您定义的第一个接收器：然后，您定义的每个后续接收器的默认值都会递增 10。
	protocols	要在接收器实例上启用的消息传递协议。 **类型** ：字符串 **示例** ： amqp,core **默认值为** ：all
	sslEnabled	指定是否在接收器端口上启用 SSL。如果设置为 **true**，请在 **sslSecret** 中指定的 secret 名称中查找 TLS/SSL 所需的凭证。 **类型** ：布尔值 **示例** ： true **默认值为** ： false

条目	子条目	描述和使用
	sslSecret	代理密钥存储、信任存储及其相应密码（所有 Base64 编码）的机密。 如果您没有为 **sslSecret** 指定自定义 secret 名称，则接收器假设默认 secret 名称。默认 secret 名称的格式为 ***<custom_resource_name>-<acceptor_name>*** **-secret**。 您必须始终自行创建此 secret，即使接收器假设默认名称。 **类型** ：字符串 **示例** ：my-broker-deployment-my-acceptor-secret **默认值为** ：*<custom_resource_name>-<acceptor_name>*-secret
	enabledCipherSuites	用于 TLS/SSL 通信的密码套件的逗号分隔列表。 指定客户端应用程序支持的最安全密码套件。如果您使用逗号分隔的列表指定代理和客户端通用的一组密码套件，或者您没有指定任何密码套件，代理和客户端会相互协商要使用的密码套件。如果您不知道要指定哪些密码套件，建议您首先与在 debug 模式中运行的客户端建立代理-客户端连接，以验证代理和客户端都常见的密码套件。然后，在代理中配置 **enabledCipherSuites**。 **类型** ：字符串 **默认值** ：未指定

条目	子条目	描述和使用
	enabledProtocols	用于 TLS/SSL 通信的协议的逗号分隔列表。 **类型** ：字符串 **示例** ：TLSv1,TLSv1.1,TLSv1.2 **默认值** ：未指定
	needClientAuth	指定是否代理通知客户端在接收器上是否需要双向 TLS。此属性覆盖 **wantClientAuth**。 **类型** ：布尔值 **示例** ： true **默认值** ：未指定
	wantClientAuth	指定是否向客户端告知在接收器上 *请求* 双向 TLS，但不是必需的。此属性将被 **needClientAuth 覆盖**。 **类型** ：布尔值 **示例** ： true **默认值** ：未指定
	verifyHost	指定是否将客户端证书的 Common Name(CN)与其主机名进行比较，以验证它们是否匹配。这个选项只适用于使用双向 TLS。 **类型** ：布尔值 **示例** ： true **默认值** ：未指定
	sslProvider	指定 SSL 提供程序是 JDK 还是 OPENSSL。 **类型** ：字符串 **示例** ：OPENSSL **默认值** ：JDK

条目	子条目	描述和使用
	sniHost	与传入连接上的 **server_name** 扩展匹配的正则表达式。如果名称不匹配，则拒绝与接收器的连接。 **类型** ：字符串 **示例** ：some_regular_expression **默认值** ：未指定
	expose	指定是否将接收器公开给 OpenShift Container Platform 之外的客户端。 当您向 OpenShift 外的客户端公开接收器时，Operator 会自动为部署中的每个代理 Pod 创建专用服务和路由。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	anycastPrefix	客户端用于指定应该使用 **anycast** 路由类型的前缀。 **类型** ：字符串 **示例** ： jms.queue **默认值** ：未指定
	multicastPrefix	客户端用于指定 **多播** 路由类型的前缀。 **类型** ：字符串 **示例** ：/topic/ **默认值** ：未指定

条目	子条目	描述和使用
	connectionsAllowed	接收器上允许的连接数。达到此限制时，会向日志发出 DEBUG 消息，连接将被拒绝。使用中的客户端类型决定了在连接被拒绝时会发生什么。 **类型** ：整数 **示例** ：2 **默认值** ：0（无限制连接）
	amqpMinLargeMessageSize	代理作为大型消息处理 AMQP 消息所需的最小消息大小，以字节为单位。如果 AMQP 消息的大小等于或大于这个值，代理会将消息存储在大型信息目录中（默认为**/opt/*<custom_resource_name>*/data/large-messages**），代理用于消息存储。将值设置为 **-1** 可禁用 AMQP 消息的大型消息处理。 **类型** ：整数 **示例** ：204800 **默认值** ：102400 (100 KB)
connectors.connector		单个连接器配置实例。
	name*	连接器的名称。 **类型** ：字符串 **示例** ：my-connector **默认值** ：Not applicable
	type	要创建的连接器的类型；tcp **或 vm**。 **类型** ：字符串 **示例** ：vm **默认值** ：tcp

条目	子条目	描述和使用
	主机*	要连接的主机名或 IP 地址。 **类型** ：字符串 **示例** ：192.168.0.58 **默认值** ：未指定
	port*	用于连接器实例的端口号。 **类型** ：int **示例** ：22222 **默认值** ：未指定
	sslEnabled	指定连接器端口是否启用了 SSL。如果设置为 **true**，请在 **sslSecret** 中指定的 secret 名称中查找 TLS/SSL 所需的凭证。 **类型** ：布尔值 **示例** ： true **默认值为** ： false

条目	子条目	描述和使用
	sslSecret	代理密钥存储、信任存储及其相应密码（所有 Base64 编码）的机密。 如果您没有为 **sslSecret** 指定自定义 secret 名称，则连接器会假定为默认 secret 名称。默认 secret 名称的格式为 ***<custom_resource_name>-<connector_name>*-secret**。 您必须始终自行创建此 secret，即使连接器假定默认名称时也是如此。 **类型** ：字符串 **示例** ：my-broker-deployment-my-connector-secret **默认值为** ：*<custom_resource_name>-<connector_name>*-secret
	enabledCipherSuites	用于 TLS/SSL 通信的密码套件的逗号分隔列表。 **类型** ：字符串 **注** ：对于连接器，建议您 **不要** 指定密码套件列表。 **默认值** ：未指定
	enabledProtocols	用于 TLS/SSL 通信的协议的逗号分隔列表。 **类型** ：字符串 **示例** ：TLSv1,TLSv1.1,TLSv1.2 **默认值** ：未指定

条目	子条目	描述和使用
	needClientAuth	指定代理是否告知客户端连接器上需要双向 TLS。此属性覆盖 **wantClientAuth**。 **类型** ：布尔值 **示例** ： true **默认值** ：未指定
	wantClientAuth	指定代理是否在连接器上*请求*双向 TLS，但不是必需的。此属性将被 **needClientAuth 覆盖**。 **类型** ：布尔值 **示例** ： true **默认值** ：未指定
	verifyHost	指定是否将客户端证书的 Common Name(CN)与其主机名进行比较，以验证它们是否匹配。这个选项只适用于使用双向 TLS。 **类型** ：布尔值 **示例** ： true **默认值** ：未指定
	sslProvider	指定 SSL 提供程序是 **JDK** 还是 **OPENSSL**。 **类型** ：字符串 **示例** ：OPENSSL **默认值** ：JDK

条目	子条目	描述和使用
	sniHost	与传出连接上的 **server_name** 扩展匹配的正则表达式。如果名称不匹配，则连接器连接将被拒绝。 **类型** ：字符串 **示例** ：some_regular_expression **默认值** ：未指定
	expose	指定是否向 OpenShift Container Platform 之外的客户端公开连接器。 **类型** ：布尔值 **示例** ： true **默认值为** ： false

条目	子条目	描述和使用
addressSettings.applyRule		指定 Operator 如何为每个匹配地址或一组地址应用添加到 CR 中的配置。 您可以指定的值有： **merge_all** 对于在 CR **和** 与同一地址组匹配的默认配置中指定的地址设置： ● 将默认配置中指定的任何属性值替换为 CR 中指定的任何属性值。 ● 保留在 CR **或** 默认配置中唯一指定的任何属性值。在最终合并的配置中包含每一个。 对于 CR 中指定的地址设置， 或者唯一与特定地址或地址集匹配的默认配置，请在最终合并的配置中包括它们。 **merge_replace** 对于 CR 和与同一地址组匹配的 CR 和 **默认配置** 中指定的地址设置，请在最终的 **CR** 中指定的设置中包括合并的配置。**不要包含** 默认配置中指定的任何属性，即使这些属性没有在 CR 中指定。 对于 CR 中指定的地址设置， 或者唯一与特定地址或地址集匹配的默认配置，请在最终合并的配置中包括它们。 **replace_all** 将默认配置中指定的 **所有** 地址设置替换为 CR 中指定的设置。最终的 lgred 配置与 CR 中指定的配置完全对应。 **类型** ：字符串 **示例** ： replace_all **默认值** ： merge_all

条目	子条目	描述和使用
addressSettings.addressSetting		匹配地址集的地址设置。
	addressFullPolicy	指定在配置了 **maxSizeBytes** 的地址已满时会发生什么。可用的策略有： **页面** 发送到完整地址的消息将分页到磁盘。 **DROP** 发送到完整地址的消息将被静默丢弃。 **FAIL** 发送到完整地址的消息将被丢弃，消息制作者会收到异常。 **BLOCK** 消息制作者在尝试发送任何其他消息时将阻止。 BLOCK 策略仅适用于 AMQP、OpenWire 和 Core 协议，因为这些协议支持流控制。 **类型** ：字符串 **示例** ：DROP **默认值** ：页面
	autoCreateAddresses	指定当客户端发送消息时代理是自动创建地址，还是尝试使用绑定到不存在的地址的队列。 **类型** ：布尔值 **示例** ： false **默认值为** ： true

条目	子条目	描述和使用
	autoCreateDeadLetterResources	指定代理是否自动创建一个死信地址和队列来接收未传送的信息。 如果参数设为 **true**，代理会自动创建一个死信地址和关联的死信队列。自动创建的地址的名称与您为 **deadLetterAddress** 指定的值匹配。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	autoCreateExpiryResources	指定代理是否自动创建一个地址和队列来接收过期的信息。 如果参数设为 **true**，代理会自动创建一个到期地址和关联的到期队列。自动创建的地址的名称与您为 expiration **Address** 指定的值匹配。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	autoCreateJmsQueues	此属性已弃用。相反，应使用 **autoCreateQueues**。
	autoCreateJmsTopics	此属性已弃用。相反，应使用 **autoCreateQueues**。
	autoCreateQueues	指定当客户端发送消息或尝试使用尚不存在的队列中的邮件时，代理是否会自动创建队列。 **类型** ：布尔值 **示例** ： false **默认值为** ： true

条目	子条目	描述和使用
	autoDeleteAddresses	指定代理是否会在代理不再有队列时自动删除创建的地址。 **类型** ：布尔值 **示例** ： false **默认值为** ： true
	autoDeleteAddressDelay	当地址没有队列时，代理会在自动删除自动创建的地址前等待，以毫秒为单位。 **类型** ：整数 **示例** ：100 **默认值** ：0
	autoDeleteJmsQueues	此属性已弃用。改为使用 **autoDeleteQueues**。
	autoDeleteJmsTopics	此属性已弃用。改为使用 **autoDeleteQueues**。
	autoDeleteQueues	指定当队列没有消费者且没有消息时，代理是否会自动删除创建的队列。 **类型** ：布尔值 **示例** ： false **默认值为** ： true
	autoDeleteCreatedQueues	指定当队列没有消费者且没有消息时，代理是否会自动删除手动创建的队列。 **类型** ：布尔值 **示例** ： true **默认值为** ： false

条目	子条目	描述和使用
	autoDeleteQueuesDelay	当队列没有消费者时，代理会在自动删除自动创建的队列前等待，以毫秒为单位。 **类型** ：整数 **示例** ：10 **默认值** ：0
	autoDeleteQueuesMessageCount	代理评估是否可以自动删除队列前可在队列中的最大消息数。 **类型** ：整数 **示例** ：5 **默认值** ：0
	configDeleteAddresses	重新载入配置文件时，此参数指定如何处理从 配置文件中删除的地址（及其队列）。您可以指定以下值： **OFF** 当重新载入配置文件时，代理不会删除该地址。 **FORCE** 重新载入配置文件时，代理会删除地址及其队列。如果队列中有任何消息，它们也会被删除。 **类型** ：字符串 **示例** ：FORCE **默认值** ：OFF

条目	子条目	描述和使用
	configDeleteQueues	重新载入配置文件时，此设置指定了代理如何处理从配置文件中删除的队列。您可以指定以下值： **OFF** 当重新载入配置文件时，代理不会删除队列。 **FORCE** 在重新加载配置文件时，代理会删除队列。如果队列中有任何消息，它们也会被删除。 **类型** ：字符串 **示例** ：FORCE **默认值** ：OFF
	deadLetterAddress	代理发送到的地址（即 *未传送*）信息。 **类型** ：字符串 **示例** ：DLA **默认值** ：无
	deadLetterQueuePrefix	代理应用到自动创建的死信队列名称的前缀。 **类型** ：字符串 **示例** ：myDLQ. **默认值** ：DLQ.
	deadLetterQueueSuffix	代理应用到自动创建的死信队列的后缀。 **类型** ：字符串 **示例** ：.DLQ **默认值** ：无

条目	子条目	描述和使用
	defaultAddressRoutingType	自动创建地址中使用的路由类型。 **类型** ：字符串 **示例** ：ANYCAST **默认值** ：多播
	defaultConsumersBeforeDispatch	在消息发送开始前所需的消费者数量，以查找地址上的队列。 **类型** ：整数 **示例** ：5 **默认值** ：0
	defaultConsumerWindowSize	使用者的默认窗口大小（以字节为单位）。 **类型** ：整数 **示例** ：300000 **默认值** ：1048576 (1024*1024)
	defaultDelayBeforeDispatch	默认时间，以毫秒为单位，如果未达到为 **defaultConsumersBeforeDispatch** 指定的值，代理会在分配消息前等待。 **类型** ：整数 **示例** ：5 **默认值为** ：-1（无延迟）
	defaultExclusiveQueue	指定地址上的所有队列是否默认是独占队列。 **类型** ：布尔值 **示例** ： true **默认值为** ： false

条目	子条目	描述和使用
	defaultGroupBuckets	用于消息分组的存储桶数量。 **类型** ：整数 **示例** ：0（消息分组禁用） **默认值为** ：-1（无限制）
	defaultGroupFirstKey	用于向消费者指明组中首先消息的密钥。 **类型** ：字符串 **示例** ： firstMessageKey **默认值** ：无
	defaultGroupRebalance	指定在新消费者连接到代理时是否重新平衡组。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	defaultGroupRebalancePauseDispatch	指定在代理重新平衡组时是否暂停消息分配。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	defaultLastValueQueue	指定地址中的所有队列是否默认是最后的值队列。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	defaultLastValueKey	用于最后一个值队列的默认键。 **类型** ：字符串 **示例** ：stock_ticker **默认值** ：无

条目	子条目	描述和使用
	defaultMaxConsumers	队列中允许的最大消费者数量. **类型** ：整数 **示例** ：100 **默认值为** ：-1（无限制）
	defaultNonDestructive	指定地址上的所有队列是否默认是非破坏性的。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	defaultPurgeOnNoConsumers	指定代理是否在没有消费者时清除队列的内容。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	defaultQueueRoutingType	自动创建的队列中使用的路由类型。默认值为 **MULTICAST**。 **类型** ：字符串 **示例** ：ANYCAST **默认值** ：多播
	defaultRingSize	匹配队列的默认环大小，该队列没有明确设置环大小。 **类型** ：整数 **示例** ：3 **默认值为** ：-1（无大小限制）

条目	子条目	描述和使用
	enableMetrics	指定配置的指标插件（如 Prometheus 插件）是收集匹配地址的指标或地址集。 **类型** ：布尔值 **示例** ： false **默认值为** ： true
	expiryAddress	收到过期邮件的地址。 **类型** ：字符串 **示例** ：myExpiryAddress **默认值** ：无
	expiryDelay	过期时间，以毫秒为单位应用到使用默认过期时间的消息。 **类型** ：整数 **示例** ：100 **默认值为** ：-1（未应用过期时间）
	expiryQueuePrefix	代理应用到自动创建的到期队列的名称的前缀。 **类型** ：字符串 **示例** ：myExp. **默认值** ：EXP.
	expiryQueueSuffix	代理应用到自动创建的到期队列的名称的后缀。 **类型** ：字符串 **示例** ：.EXP **默认值** ：无

条目	子条目	描述和使用
	lastValueQueue	指定队列是否只使用最后一个值。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	managementBrowsePageSize	指定管理资源可以浏览的消息数。 **类型** ：整数 **示例** ：100 **默认值** ：200
	匹配*	将地址设置与代理上配置的地址匹配的字符串。您可以指定确切的地址名称，或者使用通配符表达式将地址设置与 *一组* 地址匹配。 如果您使用通配符表达式作为 **match** 属性的值，则必须在单引号中包含该值，如 **'myAddresses*'**。 **类型** ：字符串 **示例** ： 'myAddresses*' **默认值** ：无
	maxDeliveryAttempts	指定代理在向配置的死信地址发送消息前尝试发送消息的次数。 **类型** ：整数 **示例** ：20 **默认值** ：10

条目	子条目	描述和使用
	maxExpiryDelay	过期时间，以毫秒为单位应用到使用大于此值的消息。 **类型** ：整数 **示例** ：20 **默认值为** ：-1（没有应用最长到期时间）
	maxRedeliveryDelay	代理尝试的消息重新传送之间的最大值，以毫秒为单位。 **类型** ：整数 **示例** ：100 **默认值** ：默认值为 **redeliveryDelay** 的值的十倍，默认值为 **0**。
	maxSizeBytes	地址的最大内存大小，以字节为单位。当 **addressFullPolicy** 设置为 **PAGING**、**BLOCK** 或 **FAIL** 时使用。还支持字节表示法，如 "K"、"Mb" 和 "GB"。 **类型** ：字符串 **示例** ：10Mb **默认值为** ：-1（无限制）
	maxSizeBytesRejectThreshold	代理开始拒绝信息之前，地址的最大大小，以字节为单位。**address-full-policy** 设置为 **BLOCK** 时使用。只适用于 AMQP 协议的 **maxSizeBytes**。 **类型** ：整数 **示例** ：500 **默认值为** ：-1（无最大大小）

条目	子条目	描述和使用
	messageCounterHistoryDayLimit	代理为地址保留消息计数器历史记录的天数。 **类型** ：整数 **示例** ：5 **默认值** ：0
	minExpiryDelay	过期时间，以毫秒为单位应用到使用少于这个值的消息。 **类型** ：整数 **示例** ：20 **默认值为** ：-1（应用的最短过期时间）
	pageMaxCacheSize	在分页导航期间保留在内存中的页面文件数，以优化 I/O。 **类型** ：整数 **示例** ：10 **默认值** ：5
	pageSizeBytes	以字节为单位的分页大小. 还支持 **K**、**Mb** 和 **GB** 等字节表示法。 **类型** ：字符串 **示例** ：20971520 **默认值** ：10485760（大约 10.5 MB）
	redeliveryDelay	以毫秒为单位，代理在重新传送已取消消息前等待的时间。 **类型** ：整数 **示例** ：100 **默认值** ：0

条目	子条目	描述和使用
	redeliveryDelayMultiplier	应用到 **redeliveryDelay** 的值的倍数。 **类型** ：数字 **示例** ：5 **默认值** ：1
	redeliveryCollisionAvoidanceFactor	要应用到 **redeliveryDelay 的值的倍数**，以避免冲突。 **类型** ：数字 **示例** ：1.1 **默认值** ：0
	redistributionDelay	在重新分发所有剩余消息前，代理会在队列中最后一个消费者关闭后等待几毫秒的时间。 **类型** ：整数 **示例** ：100 **默认值为** ：-1（未设置）
	retroactiveMessageCount	要保留用于将来在地址上创建的队列的消息数量。 **类型** ：整数 **示例** ：100 **默认值** ：0
	sendToDlaOnNoRoute	指定是否将消息发送到配置的死信地址（如果无法路由到任何队列）。 **类型** ：布尔值 **示例** ： true **默认值为** ： false

条目	子条目	描述和使用
	slowConsumerCheckPeriod	代理检查速度较慢的消费者的频率（**以秒为单位**）。 **类型**：整数 **示例**：15 **默认值**：5
	slowConsumerPolicy	指定识别慢消费者时发生的情况。有效选项为 **KILL** 或 **NOTIFY**。**KILL** 终止消费者的连接，这会影响使用同一连接的任何客户端线程。**NOTIFY** 向客户端发送 **CONSUMER_SLOW** 管理通知。 **类型**：字符串 **示例**：KILL **默认值**：通知
	slowConsumerThreshold	在每秒消息中消息的最小消息消耗率，然后消费者被视为速度较慢。 **类型**：整数 **示例**：100 **默认值为**：-1（未设置）
upgrades		
	enabled	当您更新 **version** 值以指定 AMQ Broker 的新目标版本时，请指定是否允许 Operator 自动将 **deploymentPlan.image** 值更新为与 AMQ Broker 版本对应的代理容器镜像。 **类型**：布尔值 **示例**：true **默认值为**：false

条目	子条目	描述和使用
	次	指定在将 **版本** 从 AMQ Broker 的值更新为另一个（*例如从* **7.8.0** 到 **7.9.** 3）时，指定是否允许 Operator 自动更新 **deploymentPlan.image** 值。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
version		指定 AMQ Broker 的目标 *次要版本*，您希望 Operator 自动更新 CR，以使用对应的代理容器镜像。例如，如果您将 **version** 的值从 **7.6.0** 改为 **7.7.0**（把 upgrade **.enabled 和 upgrade**. **minor** 被设置为 **true**），那么 Operator 将 **deploymentPlan.image** 更新为表单 **registry.redhat.io/amq7/amq-broker-rhel8:7.8-x** 的代理镜像。 **类型** ：字符串 **示例** ：7.7.0 **默认值** ：AMQ Broker 的当前版本

8.1.2. 地址自定义资源配置参考

基于地址 CRD 的 CR 实例可让您为部署中的代理定义地址和队列。下表详述了您可以配置的项目：

重要

您部署的任何对应自定义资源(CR)中都需要标记为星号(*)的配置项目。如果没有为非必需项目明确指定值，配置将使用默认值。

条目	描述和使用
addressName*	在代理中创建的地址名称。 **类型** ：字符串 **示例** ： address0 **默认值** ：未指定
queueName	在代理上创建的队列名称。如果没有指定 **queueName**，则 CR 只创建地址。 **类型** ：字符串 **示例** ： queue0 **默认值** ：未指定
removeFromBrokerOnDelete*	指定在删除该部署的地址 CR 实例时，Operator 是否删除部署中的所有代理的现有地址。默认值为 **false**，这意味着 Operator 在删除 CR 时不会删除现有地址。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
routingType*	要使用的路由类型； **任播** 或多 **播**. **类型** ：字符串 **示例** ： anycast **默认值** ：多播

8.1.3. 安全自定义资源配置参考

基于安全 CRD 的 CR 实例允许您定义部署中的代理安全配置，包括：

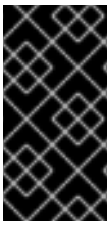
- *用户和角色*
- *登录模块，包括 propertiesLoginModule、guestLoginModule 和 keycloakLoginModule*
- *基于角色的访问控制*

- *控制台访问控制*

注意

许多选项要求您了解安全代理中描述的代理安全概念 https://access.redhat.com/documentation/en-us/red_hat_amq/2021.q3/html-single/configuring_amq_broker/#assembly-br-securing-brokers_configuring

下表详述了您可以配置的项目：



重要

您部署的任何对应自定义资源(CR)中都需要标记为星号(*)的配置项目。如果没有为非必需项目明确指定值，配置将使用默认值。

条目	子条目	描述和使用
loginModules		个或多个登录模块配置。 登录模块可以是以下类型之一： • **propertiesLoginModule** - 允许您直接定义代理用户。 • **guestLoginModule** - 对于没有登录凭证或凭证失败身份验证的用户，您可以使用客户机帐户授予对代理的有限访问权限。 • **keycloakLoginModule**. - 允许您使用 Red Hat Single Sign-On 安全代理。
propertiesLoginModule	name*	登录模块的名称。 **类型** ：字符串 **示例** ：my-login **默认值** ：Not applicable

条目	子条目	描述和使用
	users.name*	用户名称。 **类型** ：字符串 **示例** ：jdoe **默认值** ：Not applicable
	users.password*	用户的密码。 **类型** ：字符串 **示例** ：password **默认值** ：Not applicable
	users.roles	角色的名称。 **类型** ：字符串 **示例** ：查看器 **默认值** ：Not applicable
guestLoginModule	name*	guest 登录模块的名称。 **类型** ：字符串 **示例** ：guest-login **默认值** ：Not applicable
	guestUser	guest 用户的名称。 **类型** ：字符串 **示例** ：myguest **默认值** ：Not applicable
	guestRole	guest 用户的角色名称。 **类型** ：字符串 **示例** ：guest **默认值** ：Not applicable

条目	子条目	描述和使用
keycloakLoginModule	name	KeycloakLoginModule 的名称 **类型** ：字符串 **示例** ：sso **默认值** ：Not applicable
	moduleType	KeycloakLoginModule 的类型（directAccess 或 bearerToken） **类型** ：字符串 **示例** ： bearerToken **默认值** ：Not applicable
	配置	以下配置项目与红帽单点登录相关，并提供了 OpenID Connect 文档的详细信息。
	configuration.realm*	KeycloakLoginModule 的域 **类型** ：字符串 **示例** ：myrealm **默认值** ：Not applicable
	configuration.realmPublicKey	域的公钥 **类型** ：字符串 **默认值** ：Not applicable
	configuration.authServerUrl*	keycloak 身份验证服务器的 URL **类型** ：字符串 **默认值** ：Not applicable
	configuration.sslRequired	指定是否需要 SSL **类型** ：字符串 有效值为 'all', 'external' 和 'none'。
	configuration.resource*	资源名称 应用程序的 client-id。每个应用程序都有一个客户端 ID，用于识别应用程序。

条目	子条目	描述和使用
	configuration.publicClient	指定它是公共客户端。 **类型** ：布尔值 **默认值为** ： false **示例** ： false
	configuration.credentials.key	指定凭证密钥。 **类型** ：字符串 **默认值** ：Not applicable **类型** ：字符串 **默认值** ：Not applicable
	configuration.credentials.value	指定凭证值 **类型** ：字符串 **默认值** ：Not applicable
	configuration.useResourceRoleMappings	指定是否使用资源角色映射 **类型** ：布尔值 **示例** ： false
	configuration.enableCors	指定是否启用跨资源共享(CORS) 它将处理 CORS preflight 请求。它还会查看访问令牌来确定有效的来源。 **类型** ：布尔值 **默认值为** ： false
	configuration.corsMaxAge	CORS 最大年龄 如果启用了 CORS，这会设置 Access-Control-Max-Age 标头的值。
	configuration.corsAllowedMethods	CORS 允许的方法 如果启用了 CORS，这会设置 Access-Control-Allow-Methods 标头的值。这应该是以逗号分隔的字符串。

条目	子条目	描述和使用
	configuration.corsAllowedHeaders	CORS 允许的标头 如果启用了 CORS，这会设置 Access-Control-Allow-Headers 标头的值。这应该是以逗号分隔的字符串。
	configuration.corsExposedHeaders	CORS 公开的标头 如果启用了 CORS，这会设置 Access-Control-Expose-Headers 标头的值。这应该是以逗号分隔的字符串。
	configuration.exposeToken	指定是否公开访问令牌 **类型** ：布尔值 **默认值为** ： false
	configuration.bearerOnly	指定是否验证 bearer 令牌 **类型** ：布尔值 **默认值为** ： false
	configuration.autoDetectBearerOnly	指定是否仅自动探测 bearer 令牌 **类型** ：布尔值 **默认值为** ： false
	configuration.connectionPoolSize	连接池的大小 **类型** ：整数 **默认值** ：20
	configuration.allowAnyHostName	指定是否允许任何主机名 **类型** ：布尔值 **默认值为** ： false
	configuration.disableTrustManager	指定是否禁用信任管理器 **类型** ：布尔值 **默认值为** ： false
	configuration.trustStore*	信任存储的路径 这是 REQUIRED，除非 ssl-required 为 none 或 disable-trust-manager 为 true。

条目	子条目	描述和使用
	configuration.trustStorePassword*	信任存储密码 如果设置了 truststore，且信任存储需要密码，则这是 REQUIRED。
	configuration.clientKeyStore	客户端密钥存储的路径 **类型** ：字符串 **默认值** ：Not applicable
	configuration.clientKeyStorePassword	客户端密钥存储密码 **类型** ：字符串 **默认值** ：Not applicable
	configuration.clientKeyPassword	客户端密钥密码 **类型** ：字符串 **默认值** ：Not applicable
	configuration.alwaysRefreshToken	指定是否始终刷新令牌 **类型** ：布尔值 **示例** ： false
	configuration.registerNodeAtStartup	指定是否在启动时注册节点 **类型** ：布尔值 **示例** ： false
	configuration.registerNodePeriod	重新注册节点的周期 **类型** ：字符串 **默认值** ：Not applicable
	configuration.tokenStore	令牌存储类型（会话或 Cookie） **类型** ：字符串 **默认值** ：Not applicable
	configuration.tokenCookiePath	Cookie 存储的 Cookie 路径 **类型** ：字符串 **默认值** ：Not applicable

条目	子条目	描述和使用
	configuration.principalAttribute	OpenID Connect ID Token 属性，用于填充 UserPrincipal 名称 OpenID Connect ID Token 属性，用于填充 UserPrincipal 名称。如果 token 属性为空，则默认为 sub。可能的值有 sub、preferred_username、mail、name、nickname、given_name、family_name。
	configuration.proxyUrl	代理 URL
	configuration.turnOffChangeSessionIdOnLogin	指定是否在成功登录中更改会话 ID **类型** ：布尔值 **示例** ： false
	configuration.tokenMinimumTimeToLive	刷新活跃访问令牌的最短时间 **类型** ：整数 **默认值** ：0
	configuration.minTimeBetweenJwksRequests	到 Keycloak 的两个请求之间的最小间隔，以检索新的公钥 **类型** ：整数 **默认值** ：10
	configuration.publicKeyCacheTtl	到 Keycloak 的两个请求之间的最大间隔以检索新的公钥 **类型** ：整数 **默认值** ：86400
	configuration.ignoreOauthQueryParameter	是否关闭对 bearer 令牌处理的 access_token 查询参数的处理 **类型** ：布尔值 **示例** ： false
	configuration.verifyTokenAudience	验证令牌是否包含此客户端名称（资源）作为受众 **类型** ：布尔值 **示例** ： false

条目	子条目	描述和使用
	configuration.enableBasicAuth	是否支持基本身份验证 **类型** ：布尔值 **默认值为** ： false
	configuration.confidentialPort	Keycloak 服务器用于通过 SSL/TLS 进行安全连接的机密端口 **类型** ：整数 **示例** ：8443
	configuration.redirectRewriteRules.key	用于匹配 Redirect URI 的正则表达式。 **类型** ：字符串 **默认值** ：Not applicable
	configuration.redirectRewriteRules.value	替换字符串 **类型** ：字符串 **默认值** ：Not applicable
	configuration.scope	DirectAccessGrantsLoginModule 的 OAuth2 范围参数 **类型** ：字符串 **默认值** ：Not applicable
securityDomains		代理安全域
	brokerDomain.name	代理域名 **类型** ：字符串 **示例** ：activemq **默认值** ：Not applicable
	brokerDomain.loginModules	个或多个登录模块。在上面的 **loginModules** 部分中必须定义每个条目。

条目	子条目	描述和使用
	brokerDomain.loginModules.name	登录模块名称 **类型** ：字符串 **示例** ： prop-module **默认值** ：Not applicable
	brokerDomain.loginModules.flag	与 propertiesLoginModule、**必需**、**requisite**、**sufficient** 和 **optional** 都是有效的值。 **类型** ：字符串 **示例** ：足够 **默认值** ：Not applicable
	brokerDomain.loginModules.debug	Debug
	brokerDomain.loginModules.reload	reload
	consoleDomain.name	代理域名 **类型** ：字符串 **示例** ：activemq **默认值** ：Not applicable
	consoleDomain.loginModules	单个登录模块配置。
	consoleDomain.loginModules.name	登录模块名称 **类型** ：字符串 **示例** ： prop-module **默认值** ：Not applicable
	consoleDomain.loginModules.flag	与 propertiesLoginModule、**必需**、**requisite**、**sufficient** 和 **optional** 都是有效的值。 **类型** ：字符串 **示例** ：足够 **默认值** ：Not applicable

条目	子条目	描述和使用
	consoleDomain.loginModules.debug	Debug **类型** ：布尔值 **示例** ： false
	consoleDomain.loginModules.reload	reload **类型** ：布尔值 **示例** ： true **默认** ：false
securitySettings		要添加到 **broker.xml** 或 **management.xml**的额外安全设置
	broker.match	安全设置部分的地址匹配模式。有关匹配模式 语法的详情，请参阅 AMQ Broker 通配符语法。
	broker.permissions.operationType	安全设置的操作类型，如 设置权限 中所述。 **类型** ：字符串 **示例** ：创建Address **默认值** ：Not applicable
	broker.permissions.roles	安全设置应用到这些角色，如 设置权限 所述。 **类型** ：字符串 **示例** ：root **默认值** ：Not applicable
securitySettings.management		用于配置 **management.xml** 的选项。
	hawtioRoles	允许登录到 Broker 控制台的角色。 **类型** ：字符串 **示例** ：root **默认值** ：Not applicable

条目	子条目	描述和使用
	connector.host	用于连接到管理 API 的连接器主机。 **类型** ：字符串 **示例** ：myhost **默认值为**: localhost
	connector.port	用于连接到管理 API 的连接器端口。 **类型** ：整数 **示例** ：1099 **默认值** ：1099
	connector.jmxRealm	管理 API 的 JMX 域。 **类型** ：字符串 **示例** ：activemq **默认值**: activemq
	connector.objectName	管理 API 的 JMX 对象名称。 **类型** ：字符串 **示例** ：connector:name=rmi **默认** ：connector:name=rmi
	connector.authenticatorType	管理 API 身份验证类型。 **类型** ：字符串 **示例** ：password **默认** ：password
	connector.secured	管理 API 连接是否设有保护。 **类型** ：布尔值 **示例** ： true **默认值为** ： false
	connector.keyStoreProvider	用于管理连接器的密钥存储提供程序。如果您设置了 connector.secured="true"，则需要此项。默认值为 JKS。

条目	子条目	描述和使用
	connector.keyStorePath	密钥存储的位置。如果您设置了 connector.secured="true"，则需要此项。
	connector.keyStorePassword	管理连接器的密钥存储密码。如果您设置了 connector.secured="true"，则需要此项。
	connector.trustStoreProvider	如果您设置了 connector.secured="true"，则所需的管理连接器的信任存储供应商。 **类型** ：字符串 **示例** ：JKS **Default**:JKS
	connector.trustStorePath	管理连接器信任存储的位置。如果您设置了 connector.secured="true"，则需要此项。 **类型** ：字符串 **默认值** ：Not applicable
	connector.trustStorePassword	管理连接器的信任存储密码。如果您设置了 connector.secured="true"，则需要此项。 **类型** ：字符串 **默认值** ：Not applicable
	connector.passwordCodec	用于管理连接器的密码编码器 The password codec 的完全限定类名称，如在 配置文件 中加密密码 中所述。
	authorisation.allowedList.domain	allowedList 的域 **类型** ：字符串 **默认值** ：Not applicable
	authorisation.allowedList.key	allowedList 的密钥 **类型** ：字符串 **默认值** ：Not applicable

条目	子条目	描述和使用
	authorisation.defaultAccess.method	defaultAccess List 的方法 **类型** ：字符串 **默认值** ：Not applicable
	authorisation.defaultAccess.roles	defaultAccess List 的角色 **类型** ：字符串 **默认值** ：Not applicable
	authorisation.roleAccess.domain	roleAccess List 的域 **类型** ：字符串 **默认值** ：Not applicable
	authorisation.roleAccess.key	roleAccess List 键 **类型** ：字符串 **默认值** ：Not applicable
	authorisation.roleAccess.accessList.method	roleAccess List 方法 **类型** ：字符串 **默认值** ：Not applicable
	authorisation.roleAccess.accessList.roles	roleAccess List 角色 **类型** ：字符串 **默认值** ：Not applicable
	applyToCrNames	将此安全配置应用到当前命名空间中指定 CR 定义的代理。* 或空字符串的值表示应用于所有代理。 **类型** ：字符串 **示例** ：my-broker **默认值** ：由当前命名空间中的 CR 定义的所有代理。

8.2. 应用程序模板参数

在 OpenShift Container Platform 镜像上配置 AMQ Broker，通过指定应用程序模板参数的值来执

行。您可以配置以下参数：

表 8.1. 应用程序模板参数

参数	描述
AMQ_ADDRESSES	在其启动时，以逗号分隔列表中指定默认可用的地址。
AMQ_ANYCAST_PREFIX	指定应用到多路协议端口 61616 和 61617 的广播前缀。
AMQ_CLUSTERED	启用集群。
AMQ_CLUSTER_PASSWORD	指定用于集群的密码。AMQ Broker 应用程序模板使用存储在 AMQ_CREDENTIAL_SECRET 的 secret 中此参数的值。
AMQ_CLUSTER_USER	指定用于集群的集群用户。AMQ Broker 应用程序模板使用存储在 AMQ_CREDENTIAL_SECRET 的 secret 中此参数的值。
AMQ_CREDENTIAL_SECRET	指定存储中敏感凭证的 secret，如代理用户名/密码、集群用户名/密码和密钥存储密码。
AMQ_DATA_DIR	指定数据的目录。StatefulSets 中使用的。
AMQ_DATA_DIR_LOGGING	指定数据目录日志记录的目录。
AMQ_EXTRA_ARGS	指定要传递给 **artemis create** 的其他参数。
GLOBAL_MAX_SIZE	指定消息数据可消耗的最大内存量。如果没有指定值，则会分配一半系统内存。
AMQ_KEYSTORE	指定 SSL 密钥存储文件名。如果没有指定值，将生成随机密码，但不会配置 SSL。
AMQ_KEYSTORE_PASSWORD	（可选）指定用于解密 SSL 密钥存储的密码。AMQ Broker 应用程序模板使用存储在 AMQ_CREDENTIAL_SECRET 的 secret 中此参数的值。
AMQ_KEYSTORE_TRUSTSTORE_DIR	指定挂载 secret 的目录。默认值为 **/etc/amq-secret-volume**。
AMQ_MAX_CONNECTIONS	仅用于 SSL，指定接受者将接受的最大连接数。
AMQ_MULTICAST_PREFIX	指定应用到多路协议端口 61616 和 61617 的多播前缀。
AMQ_NAME	指定代理实例的名称。默认值为 **amq-broker**。

参数	描述
AMQ_PASSWORD	指定用于向代理进行身份验证的密码。AMQ Broker 应用程序模板使用存储在 AMQ_CREDENTIAL_SECRET 的 secret 中此参数的值。
AMQ_PROTOCOL	在以逗号分隔的列表中指定代理使用的消息传递协议。可用选项包括 **amqp**、**mqtt**、**Openwire**、**stomp** 和 **hornetq**。如果没有指定，则所有协议都可用。请注意，若要将映像与红帽 JBoss 企业应用平台集成，必须指定 OpenWire 协议，而且也可以选择指定其他协议。
AMQ_QUEUES	以逗号分隔列表指定在启动时代理中默认可用的队列。
AMQ_REQUIRE_LOGIN	如果设置为 **true**，则需要登录。如果未指定，或设置为 **false**，则允许匿名访问。默认情况下不指定此参数的值。
AMQ_ROLE	指定创建的角色的名称。默认值为 **amq**。
AMQ_TRUSTSTORE	指定 SSL 信任存储文件名。如果没有指定值，将生成随机密码，但不会配置 SSL。
AMQ_TRUSTSTORE_PASSWORD	（可选）指定用于解密 SSL 信任存储的密码。AMQ Broker 应用程序模板使用存储在 AMQ_CREDENTIAL_SECRET 的 secret 中此参数的值。
AMQ_USER	指定用于代理身份验证的用户名。AMQ Broker 应用程序模板使用存储在 AMQ_CREDENTIAL_SECRET 的 secret 中此参数的值。
APPLICATION_NAME	指定 OpenShift 内部使用的应用程序的名称。它用于应用中的服务、Pod 和其他对象的名称。
IMAGE	指定镜像。在 **持久性**、**持久和有状态** set **-clustered** 模板中使用。
IMAGE_STREAM_NAMESPACE	指定镜像流命名空间。在 **ssl** 和 **基本** 模板中使用。
OPENSHIFT_DNS_PING_SERVICE_PORT	指定 OpenShift DNS ping 服务的端口号。

参数	描述
PING_SVC_NAME	指定 OpenShift DNS ping 服务的名称。如果您指定了一个 **APPLICATION _NAME 的值，则默认值为 $APPLICATION** _NAME。否则，默认值是 **ping**。如果您为 **PING_SVC_NAME** 指定自定义值，则此值会覆盖默认值。如果要使用模板在同一个 OpenShift 项目命名空间中部署多个代理集群，您必须确保 **PING_SVC_NAME** 每个部署都有一个唯一值。
VOLUME_CAPACITY	指定数据库卷的持久性存储大小。

注意

如果您将 broker.xml 用于自定义配置，则该文件中指定的任何值将覆盖应用程序模板中相同参数值。

- ***AMQ_NAME***
- ***AMQ_ROLE***
- ***AMQ_CLUSTER_USER***
- ***AMQ_CLUSTER_PASSWORD***

8.3. 日志

除了查看 OpenShift 日志外，您还可以通过查看输出到容器控制台的 AMQ 日志来排除在 OpenShift Container Platform 镜像上运行的 AMQ Broker 的问题。

流程

- ***在命令行中运行以下命令：***

```
$ oc logs -f <pod-name> <container-name>
```

-

修订于 2022-04-09 22:49:00 +1000