Red Hat AMQ Broker 7.12

在 OpenShift 上部署 AMQ Broker

用于 AMQ Broker 7.12

Last Updated: 2024-05-22

Red Hat AMQ Broker 7.12 在 OpenShift 上部署 AMQ Broker

用于 AMQ Broker 7.12

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

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 OPENSHIFT CONTAINER PLATFORM 上 AMQ BROKER 简介

Red Hat AMQ Broker 7.12 作为容器化镜像，用于 OpenShift Container Platform (OCP) 4.12、4.13、4.14 或 4.15。

AMQ Broker 基于 Apache ActiveMQ Artemis。它提供符合 JMS 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。

1.1. 版本兼容性和支持

有关 OpenShift Container Platform 镜像版本兼容性的详情，请参阅：

- OpenShift Container Platform 4.x Tested Integrations

注意

所有在 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(master-slave)的高可用性
  不支持通过配置主和从对来实现高可用性 (HA)。相反，AMQ Broker 使用 OpenShift Container Platform 中提供的 HA 功能。

- 外部客户端无法使用 AMQ Broker 提供的拓扑信息
  当 AMQ Core Protocol JMS 客户端或 AMQ JMS 客户端连接到 OpenShift Container Platform 集群中的代理时，代理可以向客户端发送集群中其他代理的 IP 地址和端口信息，如果与当前代理的连接丢失，则代理充当客户端的故障转移列表。

  为每个代理提供的 IP 地址是一个内部 IP 地址，它不能被 OpenShift Container Platform 集群外部的客户端访问。要防止外部客户端尝试使用内部 IP 地址连接到代理，请在客户端用来初始连接到代理的 URI 中设置以下配置。

客户端	配置
AMQ 核心协议 JMS 客户端	**useTopologyForLoadBalancing=false**
AMQ JMS Client	**failover.amqpOpenServerListAction=IGNORE**

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，请参阅管理 sudo 访问。

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，以尖括号(**<>**)括起，样式则使用斜体和 **monospace** 字体。用下划线(_)分隔多个词语。

例如，使用以下命令将 ***<project_name>*** 替换为您自己的项目名称。

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

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

本节论述了如何规划基于 Operator 的部署。

Operator 是使您能够打包、部署和管理 OpenShift 应用程序的程序。Operator 通常自动化常见或复杂任务。通常，Operator 旨在提供：

- 一致、可重复安装
- 系统组件的健康检查
- 无线 (OTA) 更新
- 受管升级

Operator 允许您在代理实例运行时进行更改，因为它们始终侦听您用来配置部署的自定义资源 (CR) 实例。当对 CR 进行更改时，Operator 会将更改与现有代理部署进行协调，并更新部署以反映更改。另外，Operator 提供了一个消息迁移功能，可确保消息数据的完整性。当因为部署有意缩减而在集群部署中代理关闭时，此功能会将信息迁移到仍然在同一代理集群中运行的代理 Pod。

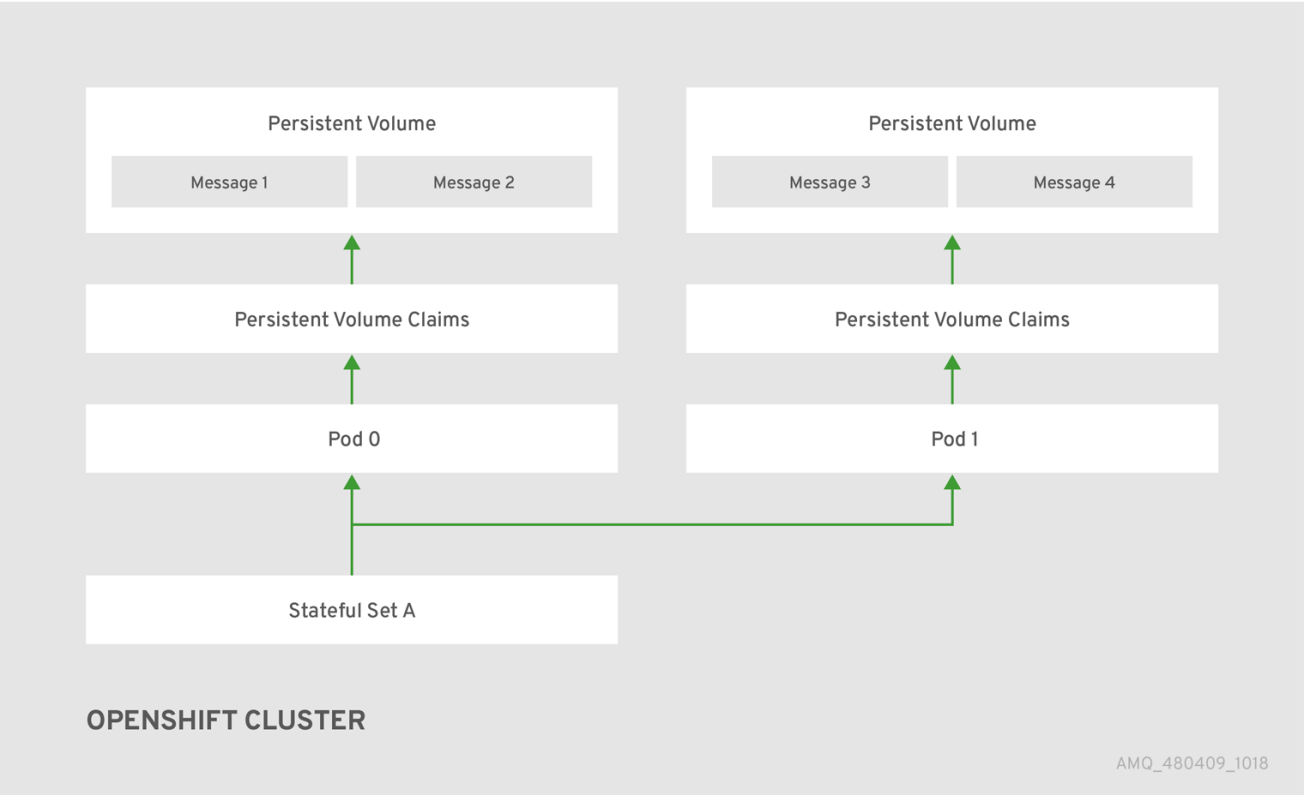
2.1. 高可用性概述(HA)

术语 *高可用性* 指的是可以保持正常运行的系统，即使该系统的一部分失败或关闭也是如此。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。

AMQ Broker 使用 OpenShift Container Platform 中提供的 HA 功能来缓解 Pod 失败：

- 如果在 AMQ Broker 上启用了持久性存储，则每个代理 Pod 将其数据写入使用持久性卷声明 (PVC)声明的持久性卷(PV)。即使 Pod 被删除后，PV 仍会保留。如果代理 Pod 失败，OpenShift Container Platform 会重启具有相同名称的 Pod，并使用包含消息传递数据的现有 PV。
- 您可以在集群中运行多个代理 Pod，并在单独的节点上分发 Pod 以防止节点失败。在集群中，每个代理 Pod 会将其消息数据写入自己的 PV，然后在不同节点上重启时该代理 Pod 可供该代理 Pod 使用。

下图显示了集群代理部署。在这种情况下，代理集群中的两个代理 Pod 仍在运行。



其他资源

有关如何使用持久性存储的详情，请参考 第 2.8 节 “Operator 部署备注”。

有关如何在独立节点上分发代理 Pod 的详情，请参考 第 4.17.2 节 “使用容限控制 pod 放置”。

2.2. AMQ BROKER OPERATOR 自定义资源定义概述

通常，自定义资源定义 (CRD) 是配置项目的模式，您可以针对通过 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，您可以为 CRD 中的配置项目指定值。如果您是 Operator 开发人员，通过 CRD 公开的内容实际上就成为配置和使用对象的 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。

您可以使用 OpenShift 命令行界面 (CLI) 或 Operator Lifecycle Manager（OperatorHub 图形界面）安装 AMQ Broker Operator。在这两种情况下，AMQ Broker Operator 都会包括下面描述的 CRD。

主代理 CRD

您可以根据此 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 为：

- Operator 安装存档（OpenShift CLI 安装方法）的 **crds** 目录中的 **broker_activemqartemis_crd** 文件
- OpenShift Container Platform Web 控制台的 **Custom Resource Definitions** 部分中的 **ActiveMQArtemis** CRD（OperatorHub 安装方法）

地址 CRD

您可以根据此 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 为：

- Operator 安装存档（OpenShift CLI 安装方法）的 **crds** 目录中的 **broker_activemqartemisaddress_crd** 文件
- OpenShift Container Platform Web 控制台的 **Custom Resource Definitions** 部分中的 **ActiveMQArtemisAddresss** CRD（OperatorHub 安装方法）

安全 CRD

您可以基于此 CRD 部署 CR 实例，以创建用户并将这些用户与安全上下文关联。
根据您安装 Operator 的方式，此 CRD 为：

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

scaleDown CRD

当实例化用于消息迁移的缩减控制器时，**Operator** 会根据这个 CRD 自动创建 CR 实例。
根据您安装 Operator 的方式，此 CRD 为：

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

其他资源

- 要了解如何安装 AMQ Broker Operator（以及所有包含的 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”。
- 有关基于主代理和地址 CRD 创建 CR 实例时使用的完整配置参考，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”

2.3. 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 归档包含 **deploy/crs** 目录中的示例自定义资源 (CR) 文件。这些 CR 文件示例可让您：

- 在没有 SSL 或集群的情况下部署最小代理。
- 定义地址。

您下载和提取的 broker Operator 归档还包括 deploy/examples/ **address 和 deploy/examples /artemis** 目录中的部署示例 CR，如下所示。

address_queue.yaml

部署具有不同名称的地址和队列。取消部署 CR 时删除队列。

address_topic.yaml

使用多播路由类型部署地址。取消部署 CR 时删除地址。

artemis_address_settings.yaml

使用特定地址设置部署代理。

artemis_cluster_persistence.yaml

使用持久性存储部署集群代理。

artemis_enable_metrics_plugin.yaml

启用 Prometheus metrics 插件来收集指标。

artemis_resources.yaml

为代理设置 CPU 和内存限值。

artemis_single.yaml

部署单个代理。

2.4. 观察 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：

- 单一命名空间（包含 Operator 的同一命名空间）
- 所有命名空间

注意

如果您已经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 7.12 版本以避免潜在的冲突。

2.5. OPERATOR 如何决定用于部署镜像的配置

在 **ActiveMQArtemis** CR 中，您可以使用以下任一配置来部署容器镜像：

- 指定 **spec.version** 属性中的版本号，并允许 Operator 选择该版本号部署的代理和 init 容器镜像。
- 指定您希望 Operator 在 **spec.deploymentPlan.image** 和 **spec.deploymentPlan.initImage** 属性中部署的特定代理和 init 容器镜像的 registry URL。
- 将 **spec.deploymentPlan.image** 属性的值设置为 **占位符**，这意味着 Operator 会选择 Operator 版本已知的最新的代理和 init 容器镜像。

注意

如果不使用这些配置来部署容器镜像，Operator 会选择 Operator 版本已知的最新的代理和 init 容器镜像。

保存 CR 后，Operator 会执行以下验证来确定要使用的配置。

- Operator 检查 CR 是否包含 **spec.version** 属性。

- 如果 CR 不包含 **spec.version** 属性，Operator 会检查 CR 是否包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性。
  - 如果 CR 包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性，Operator 将部署由 registry URL 标识的容器镜像。
  - 如果 CR 不包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性，Operator 会选择要部署的容器镜像。如需更多信息，请参阅 第 2.6 节 “Operator 如何选择容器镜像”。
- 如果 CR 包含 **spec.version** 属性，Operator 会验证指定的版本号是否在 Operator 支持的有效版本范围内。
  - 如果 **spec.version** 属性的值无效，Operator 会停止部署。
  - 如果 **spec.version** 属性的值有效，Operator 会检查 CR 是否包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性。
    - 如果 CR 包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性，Operator 将部署由 registry URL 标识的容器镜像。
    - 如果 CR 不包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性，Operator 会选择要部署的容器镜像。如需更多信息，请参阅 第 2.6 节 “Operator 如何选择容器镜像”。

注意

如果 CR 只包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性中的一个，Operator 会使用 **spec.version** number 属性为不在 CR 中的属性选择镜像，如果 **spec.version** 属性不在 CR 中，或者选择该属性的最新已知镜像。

红帽建议不要指定没有 **spec.deployment.Plan.initImage** 属性的 **spec.deploymentPlan.image** 属性，或者反之亦然，以防止部署不匹配的代理和 init 容器镜像版本。

2.6. OPERATOR 如何选择容器镜像

如果 CR 不包含 **spec.deploymentPlan.image** 和 **spec.deployment.Plan.initImage** 属性，它指定 Operator 必须部署的特定容器镜像的 registry URL，Operator 会自动选择要部署的适当容器镜像。

注意

如果使用 OpenShift 命令行界面安装 Operator，Operator 安装存档包括一个示例 CR 文件，名为 **broker_activemqartemis_cr.yaml**。在示例 CR 中，包含 **spec.deploymentPlan.image** 属性，并将其默认值设置为 **placeholder**。此值表示，在部署 CR 前，Operator 不会选择代理容器镜像。

spec.deploymentPlan.initImage 属性（用于指定 Init 容器镜像） **没有**包括在 **broker_activemqartemis_cr.yaml** 示例 CR 文件中。如果您没有在 CR 中显式包含 **spec.deploymentPlan.initImage** 属性并指定值，Operator 会选择一个与所选 Operator 容器镜像版本匹配的内置 Init 容器镜像。

要选择代理和初始容器镜像，Operator 首先决定所需的镜像的 AMQ Broker 版本。Operator 从 **spec.version** 属性的值获取版本。如果没有设置 **spec.version** 属性，Operator 将使用 AMQ Broker 的镜像的最新版本。

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

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

然后，Operator 会根据 AMQ Broker 和容器平台的版本，在 **operator.yaml** 配置文件中引用两组环境变量。这些环境变量集为 AMQ Broker 的各种版本指定代理和初始容器镜像，如以下部分所述。

2.6.1. 代理和 init 容器镜像的环境变量

operator.yaml 中包含的环境变量有以下命名约定。

容器平台	命名规则
OpenShift Container Platform	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<*AMQ_Broker_version*>**
IBM Z 上的 OpenShift Container Platform	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<*AMQ_Broker_version*>_s390x**
IBM Power 系统上的 OpenShift Container Platform	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<*AMQ_Broker_version*>_ppc64le**

以下是每个支持的容器平台的环境变量名称和 init 容器镜像的环境变量示例。

容器平台	环境变量名称
OpenShift Container Platform	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_7120** **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_7120**
IBM Z 上的 OpenShift Container Platform	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_7120_s390x** **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_7120**
IBM Power 系统上的 OpenShift Container Platform	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_7120_ppc64le** **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_7120**

每个环境变量的值都指定红帽提供的容器镜像的地址。镜像名称由 *安全哈希算法* (SHA)值表示。例如：

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_7120
  value: registry.redhat.io/amq7/amq-broker-
rhel8@sha256:3cc58e98c905d427d6be214a8df3c2687170f74abeacc33366f91bfc18e69a3a
```

因此，Operator 基于 AMQ Broker 版本和容器平台，Operator 决定代理和 init 容器的适用环境变量名称。Operator 在启动代理容器时使用对应的镜像值。

其他资源

- 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”。
- 要了解如何构建并指定 *自定义初始容器镜像*，请参阅 第 4.11 节 “指定自定义初始容器镜像”。

2.7. 在自定义资源(CR)中验证镜像和版本配置

保存 CR 后，Operator 会执行以下 CR 配置验证，并在 CR 中提供状态。

验证	验证目的	CR 中报告的状态
CR 是否包含没有 **spec.version** 属性的 **spec.deploymentPlan.image** 属性。	没有 **spec.version** 属性的 **spec.deploymentPlan.image** 属性会导致 Operator 每次升级 Operator 时重启代理 pod。Pod 重启是必需的，因为新 Operator 使用最新支持的代理版本更新 StatefulSet 中的标签，除非在 **spec.version** 属性中明确设置了版本号。	**Valid** 条件为 **Unknown**，并显示以下状态消息： **Unknown image version, 在指定镜像时在 spec.version 中设置受支持的代理** 版本。
CR 是否包含 **spec.deploymentPlan.image** 属性，没有 **spec.deploymentPlan.initImage** 属性，反之亦然。	使用这个配置，可以部署代理和 init 容器镜像的不同版本，这可能会阻止代理启动。	'Valid' 条件为 **Unknown**，且显示以下状态信息： **Init image 和 broker 镜像必须配置为相互独立的对。**
如果 CR 包含 **spec.version** 属性，是 Operator 支持的版本范围中指定的版本。	如果 **spec.version** 属性的值是 Operator 不支持的代理版本，Operator 不会进行代理 pod 部署。	**Valid** 条件为 **False**，并显示以下状态信息： **Spec.Version 不解析为受支持的代理版本，原因在 <*version*> 支持的列表中找不到匹配的代理。**
是否根据 **spec.deploymentPlan.image** 属性中的容器镜像的 URL 部署代理镜像版本，与 **spec.version** 属性中的代理版本匹配。	如果在 CR 中配置了两个属性，则标记部署的实际代理版本和 **spec.version** 属性中显示的版本不匹配。这是用于突出显示 **spec.version** 属性中显示的版本不是部署的版本。	**BrokerVersionAligned** 条件的状态为 **Unknown**，并显示以下信息： **broker*version*not aligned on pod <*pod name*>, detected version < version > doesn't match the spec.*version*< version > resolved as <*version*>.**

其他资源

有关在 CR 中查看状态信息的更多信息，请参阅查看代理部署的状态信息。

2.8. OPERATOR 部署备注

本节介绍了规划基于 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。当部署了 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。要启用常规用户来部署 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。

- 当使用最新 Operator 版本的 CRD 更新集群时，这个更新会影响 **集群中的所有项目**。从 Operator 之前的版本部署的任何代理 pod 可能无法更新其状态。当您在 OpenShift Container Platform Web 控制台中点正在运行的代理 pod 的 Logs 选项卡时，您会看到表示 'UpdatePodStatus' 失败的消息。但是，该项目中的代理 pod 和 Operator 将继续按预期工作。要为受影响的项目修复此问题，还必须升级该项目以使用最新版本的 Operator。

- 虽然您可以通过部署多个自定义资源(CR)实例，但在给定的 OpenShift 项目中创建多个代理部署，但通常会在项目中创建单个代理部署，然后为地址部署多个 CR 实例。
  红帽建议您在单独的项目中创建代理部署。

- 如果要使用持久性存储部署代理，且没有 OpenShift 集群中的容器原生存储，则需要手动置备持久性卷(PV)，并确保这些代理可以被 Operator 声明。例如，如果要创建具有持久性存储的两个代理（即，在 CR 中设置 **persistenceEnabled=true** ）的集群，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。
  如果在 CR 中指定 **persistenceEnabled=false**，则部署的代理 *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

- 在首次部署 CR **之前**，您必须将以下列出的项目的配置添加到主代理 CR 实例。**您不能将** 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)的大小和存储类

  - 为部署中的每个代理的限制和请求

- 如果更新 CR 中的参数，Operator 无法在 StatefulSet 中动态更新，Operator 会删除 StatefulSet，并使用更新的参数值重新创建 StatefulSet。删除 StatefulSet 会导致所有 pod 都被删除并重新创建，这会导致临时代理中断。如果 StatefulSet 中将 **persistenceEnabled=false** 改为 **persistenceEnabled=true**，则 Operator 无法在 StatefulSet 中动态更新 CR 更新示例。

2.9. 识别现有 OPERATOR 监控的命名空间

如果集群已经包含 AMQ Broker 的 Operator，并且希望新 Operator 监视所有或多个命名空间，您必须确保新 Operator 不会监控任何与现有 Operator 相同的命名空间。使用以下步骤识别现有 Operator 监控的命名空间。

流程

1. 在 OpenShift Container Platform Web 控制台左侧窗格中，单击 **Workloads → Deployments**。

2. 在 **Project** 下拉列表中，选择 **All Projects**。

3. 在 **Filter Name** 框中，指定一个字符串，例如 **amq** 来显示集群中安装的 AMQ Broker 的 Operator。

注意

namespace 列显示 **部署** 每个 Operator 的命名空间。

4. 检查每个已安装 Operator 的命名空间是否已配置为 **监视**。

   a. 点 Operator 名称显示 Operator 详情并点击 **YAML** 选项卡。

   b. 搜索 **WATCH_NAMESPACE** 并记下 Operator 监视的命名空间。

   - 如果 **WATCH_NAMESPACE** 部分有一个 **fieldPath** 字段，它的值为 **metadata.namespace**，Operator 会监视部署它的命名空间。
   - 如果 **WATCH_NAMESPACE** 部分有一个具有命名空间列表 **的值** 字段，Operator 会监视指定的命名空间。例如：

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

   - 如果 **WATCH_NAMESPACE** 部分有一个为空或带有一个星号的 **value** 字段时，Operator 会查看集群中的所有命名空间。例如：

```
- name: WATCH_NAMESPACE
  value: ""
```

   在这种情况下，在部署新 Operator 前，您必须卸载现有 Operator 或重新配置它以监视特定的命名空间。

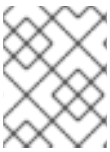
下一部分中的流程演示了如何安装 Operator 并使用自定义资源(CR)在 OpenShift Container Platform 上创建代理部署。完成这些步骤后，Operator 会在单独的 Pod 中运行，以及您创建的每个代理实例作为与 Operator 位于同一项目中的 StatefulSet 中的单个 Pod 运行。之后，您将了解如何使用专用寻址 CR 在代理部署中定义地址。

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 并使用它来创建代理部署前，您应该参阅 第 2.8 节 “Operator 部署备注” 中的 Operator 部署备注。

3.2. 使用 CLI 安装 OPERATOR



注意

每个 Operator 发行版本都要求您下载最新的 **AMQ Broker 7.12.0 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)在给定的 OpenShift 项目中安装和部署 AMQ Broker 7.12 的 Operator 的最新版本。在后续流程中，您可以使用此 Operator 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，请参考 第 3.3 节 “使用 OperatorHub 安装 Operator”。
- 要了解 *升级* 基于 Operator 的代理部署的信息，请参阅 *第 6 章 升级基于 Operator 的代理部署*。

3.2.1. 准备部署 Operator

在使用 CLI 部署 Operator 之前，您必须下载 Operator 安装文件并准备部署。

流程

1. 在 Web 浏览器中，导航到 AMQ Broker 7.12.0 版本的 **Software Downloads** 页面。
2. 确保 **Version** 下拉列表的值设为 **7.12.0**，并且选择了 **Releases** 选项卡。
3. 在最新的 **AMQ Broker 7.12.0 Operator Installation and Example Files**旁边，点 **Download**。下载 **amq-broker-operator-7.12.0-ocp-install-examples.zip** 压缩存档会自动开始。
4. 将存档移到您选择的目录中。以下示例将存档移到名为 **~/broker/operator** 的目录中。

```
$ mkdir ~/broker/operator
$ mv amq-broker-operator-7.12.0-ocp-install-examples.zip ~/broker/operator
```

5. 在您选择的目录中，提取存档的内容。例如：

```
$ cd ~/broker/operator
$ unzip amq-broker-operator-7.12.0-ocp-install-examples.zip
```

6. 切换到提取存档时创建的目录。例如：

```
$ cd amq-broker-operator-7.12.0-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. 指定您要在其中安装 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. 指定要与 Operator 搭配使用的服务帐户。

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

   b. 确保 **kind** 元素设置为 **ServiceAccount**。

   c. 如果要更改默认服务帐户名称，在 **metadata** 部分中，将 **amq-broker-controller-manager** 替换为自定义名称。

   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** 元素设为 **Role**。

    c. 如果要更改默认角色名称，在 **metadata** 部分中，将 **amq-broker-operator-role** 替换为自定义名称。

    d. 在项目中创建角色。

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

11. 为 Operator 指定角色绑定。角色绑定会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。

    b. 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-rolebinding
    subjects:
      kind: ServiceAccount
      name: amq-broker-controller-manager
    roleRef:
      kind: Role
      name: amq-broker-operator-role
    ```

c. 在项目中创建角色绑定。

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

12. 为 Operator 指定领导选举角色绑定。角色绑定会根据您指定的名称将之前创建的服务帐户绑定到领导选举角色。

a. 为 Operator 创建领导选举角色。

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

b. 在项目中创建领导选举角色绑定。

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

13. （可选） 如果您希望 Operator 监视多个命名空间，请完成以下步骤：

注意

如果 OpenShift Container Platform 集群已经包含为 AMQ Broker 安装的 Operator，您必须确保新 Operator 不会监视与现有 Operator 相同的命名空间。有关如何识别现有 Operator 监控的命名空间的详情，请参考 识别现有 Operator 监控的命名空间。

a. 在您下载和提取的 Operator 归档的 deploy 目录中，打开 **operator_yaml** 文件。

b. 如果您希望 Operator 监控集群中的所有命名空间，在 **WATCH_NAMESPACE** 部分，添加一个 **value** 属性，并将值设为星号。注释掉 **WATCH_NAMESPACE** 部分中的现有属性。例如：

```
- name: WATCH_NAMESPACE
  value: "*"
# valueFrom:
#   fieldRef:
#     fieldPath: metadata.namespace
```

注意

为避免冲突，请确保多个 Operator 不监视同一命名空间。例如，如果您部署 Operator **以监视** 集群中的所有命名空间，则无法部署另一个 Operator 来监视单独的命名空间。如果在集群中部署了 Operator，您可以指定新 Operator 监视的命名空间列表，如下所述。

c. 如果您希望 Operator 监控多个命名空间，但不是集群中的所有命名空间，在 **WATCH_NAMESPACE** 部分指定命名空间列表。确保您排除了现有 Operator 监控的任何命名空间。例如：

```
- name: WATCH_NAMESPACE
  value: "namespace1, namespace2"`.
```

d. 在您下载和提取的 Operator 归档的 deploy 目录中，打开 **cluster_role_binding.yaml** 文件。

e. 在 Subjects 部分中，指定一个与您要部署 Operator 的 OpenShift Container Platform 项目对应的命名空间。**例如：**

```
Subjects:
- kind: ServiceAccount
  name: amq-broker-controller-manager
  namespace: operator-project
```

注意

如果您之前使用早期版本的 Operator 部署代理，并且希望部署 Operator 以观察多个命名空间，请参阅 升级前。

f. **在项目中创建集群角色。**

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

g. **在项目中创建集群角色绑定。**

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

在以下步骤中，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 项目中为 AMQ Broker 7.12 部署最新版本的 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 “准备部署 Operator”。**
- **从 AMQ Broker 7.3 开始，您可以使用 Red Hat Ecosystem Catalog 的新版本来访问容器镜像。这个新版本的 registry 要求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操作前，您必须先完成 Red Hat Container Registry Authentication中所述的步骤。**
- **如果要使用持久性存储部署代理，且没有 OpenShift 集群中的容器原生存储，则需要手动置备持久性卷(PV)，并确保它们可以被 Operator 声明。例如，如果要创建具有持久性存储的两个代理（即，在自定义资源中设置 persistenceEnabled=true ）的集群，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 persistenceEnabled=false，则部署的代理 *将使用临时存储*。临时存储意味着每次重启代理 Pod 时，任何现有的数据都会丢失。

  有关置备持久性存储的更多信息，请参阅：

  - **了解持久性存储**

流程

1. **在 OpenShift 命令行界面(CLI)中，以集群管理员身份登录到 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-7.12.0-ocp-install-examples
```

4. 部署 Operator 中包含的 CRD。在部署并启动 Operator 前，您必须在 OpenShift 集群中安装 CRD。

   a. 部署主要代理 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 控制器 CRD。

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

   d. 部署安全 CRD：

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

5. 将与 Red Hat Ecosystem Catalog 进行身份验证时使用的账户相关联的 pull secret 与您的 OpenShift 项目的 default, deployer, 和 builder 服务账户进行链接。

```
$ 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 归档的 deploy 目录中，打开 operator.yaml 文件。确保 spec.containers.image 属性的值与 Operator 的 7.12.0-opr-1 版本对应，如下所示。

```
spec:
  template:
    spec:
      containers:
        #image: registry.redhat.io/amq7/amq-broker-rhel8-operator:7.10
        image: registry.redhat.io/amq7/amq-broker-rhel8-
operator@sha256:757c2f0129a6eadc2e427eb5bb3051b14d1770bc227214924ad4966a7f
412a1f
```

注意

在 operator.yaml 文件中，Operator 使用一个由 *Secure Hash Algorithm*(SHA) 值代表的镜像。注释行以数字符号 (#) 符号开头，注明 SHA 值对应于特定的容器镜像标签。

7. 部署 Operator。

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

在 OpenShift 项目中，Operator 会在新 Pod 中启动。

在 OpenShift Container Platform Web 控制台中，Operator Pod 的 Events 标签页上的信息确认 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，并启动了新容器。

另外，如果您点击 Pod 中的 Logs 选项卡，输出应包含重新排序以下内容的行：

```
...
{"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}
```

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

注意

建议在一个给定的 OpenShift 项目中仅部署 AMQ Broker Operator 的一个单个实例。不建议将 Operator 部署的 **spec.replicas** 属性设置为大于 1 的值，或者在同一项目中多次部署 Operator。

其他资源

- 有关安装使用 OperatorHub 图形界面的 AMQ Broker Operator 的替代方法，请参考 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 及更新的版本中，*Operator Lifecycle Manager* (OLM)可帮助用户安装、更新并通常管理所有 Operator 以及在用户集群中运行的关联服务的生命周期。它是 Operator Framework 的一部分，它是一个开源工具包，用于以有效、自动化且可扩展的方式管理 Kubernetes 原生应用程序(Operator)。

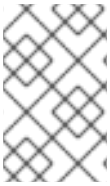
OLM 默认在 OpenShift Container Platform 4.5 及更新的版本中运行，辅助集群管理员对集群上运行的 Operator 进行安装、升级和授予访问权。OpenShift Container Platform Web 控制台提供一些管理界面，供集群管理员安装 Operator，以及为特定项目授权以便使用集群上的可用 Operator 目录。

OperatorHub 是 OpenShift 集群管理员使用 OLM 发现、安装和升级 Operator 的图形界面。只需点一个按钮，即可从 OperatorHub 拉取并在 OperatorHub 中安装，并由 OLM 管理，为工程团队在开发、测试和生产环境中自助管理软件。

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。



注意

在 OperatorHub 中，您只能安装每个频道中提供的最新 Operator 版本。如果要安装 Operator 的早期版本，则必须使用 CLI 安装 Operator。更多信息请参阅 第 3.2 节 “使用 CLI 安装 Operator”。

先决条件

- OperatorHub 中必须提供 **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator。
- 您需要有集群管理员特权。

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 **Operators** → **OperatorHub**。
3. 在 OperatorHub 页面顶部的 **Project** 下拉菜单中选择您要在其中部署 Operator 的项目。
4. 在 OperatorHub 页面中，使用 **Filter by keyword...** 复选框来查找 **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator。

注意

在 OperatorHub 中，您可能会发现多个 Operator 与其名称中包含 **AMQ Broker**。确保点 **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator。当您点此 Operator 时，查看打开的信息窗格。对于 AMQ Broker 7.12，Operator 的最新次要版本标签是 **7.12.0-opr-1**。

5. 点 **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator。在出现的对话框中，点 **Install**。
6. 在 **Install Operator** 页面中：
   a. 在 **Update Channel** 下，选择 **7.11.x** 频道以仅接收版本 7.11 的更新。**7.11.x** 频道是 Long Term Support (LTS)频道。
   根据 OpenShift Container Platform 集群安装的时间，您可能还会看到较旧版本的 AMQ Broker 的频道。唯一支持的频道是 **7.10.x**，它也是 LTS 频道。
   b. 在 **Installation Mode** 下，选择 Operator 监视的命名空间：
      - **A specific namespace on the cluster** - Operator 已安装在该命名空间中，仅监控该命名空间是否有 CR 更改。
      - **All namespaces** - Operator 监控所有命名空间是否有 CR 更改。

注意

如果您之前使用早期版本的 Operator 部署代理，而您希望部署 Operator 以观察多个命名空间，请参阅升级前的操作。

7. 在 Installed Namespace 下拉菜单中选择您要安装 Operator 的项目。
8. 在 Approval Strategy 下，确保选择了授权 **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 页面将打开。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。

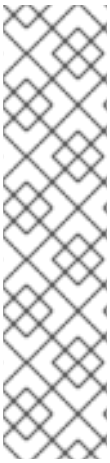
其他资源

- 要了解如何在安装了 AMQ Broker 的 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，但在给定的 OpenShift 项目中创建多个代理部署，但通常会在项目中创建单个代理部署，然后为地址部署多个 CR 实例。
  红帽建议您在单独的项目中创建代理部署。
- 在 AMQ Broker 7.12 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。
  - 持久性存储部署中每个代理所需的持久性卷声明(PVC)的大小和存储类
  - 为部署中的每个代理的限制和请求

先决条件

- 您必须已安装了 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。更多信息请参阅 第 2.6 节 “Operator 如何选择容器镜像”。

- 从 AMQ Broker 7.3 开始，您可以使用 Red Hat Ecosystem Catalog 的新版本来访问容器镜像。这个新版本的 registry 要求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操作前，您必须先完成 Red Hat Container Registry Authentication中所述的步骤。

流程

成功安装 Operator 时，Operator 会运行并侦听与 CR 相关的更改。本例流程演示了如何使用 CR 实例在项目中部署基本代理。

1. 开始为代理部署配置自定义资源(CR)实例。

   a. 使用 OpenShift 命令行界面：

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

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

      ii. 打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 Operator 安装的 deploy/crs 目录中。

   b. 使用 OpenShift Container Platform Web 控制台：

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

      ii. 根据主代理 CRD 启动新的 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。

   对于基本代理部署，配置可能类似如下。

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

   观察 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性被设置为 占位符 的默认值。此值表示，默认情况下 image 属性没有指定用于部署的代理容器镜像。要了解 Operator 如何确定要使用的适当代理容器镜像，请参阅 第 2.6 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。此命名惯例表示 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-ss。另外，部署中的代理 Pod 直接基于 StatefulSet 名称，如ex-aao-ss-0、ex-aao-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。值 2 或更高指定集群代理部署。但是，要部署单个代理实例，请确保将值设为 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 控制台中点 Workloads → StatefulSets。您会看到一个名为 ex-aao-ss 的新 StatefulSet。**

   a. **点 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，点 Pods 选项卡。点 ex-aao-ss Pod。在运行 Pod 的 Events 选项卡中，您会看到代理容器已启动。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 上的 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. **点 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
```

前面的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 1000 个消息发送到队列。

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
```

其他资源

- 有关主代理自定义资源(CR)的完整配置参考，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

如果您的项目中有两个或者多个代理 Pod，则 Pod 会自动组成一个代理集群。集群配置可让代理互相连接并根据需要重新分发信息，以进行负载平衡。

以下流程演示了如何部署集群代理。默认情况下，此部署中的代理会 *按需* 进行负载平衡，这意味着代理只会将消息转发到具有匹配使用者的其他代理。

先决条件

- 已部署基本代理实例。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/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  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 实例时才指定值。您应该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. 以具有在之前创建的基本代理部署项目中部署 CR 的用户身份登录 OpenShift。

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

5. 切换到之前创建的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，在项目中启动额外的代理 Pod。默认情况下，项目中运行的代理是集群的。

7. 打开每个 Pod 的 Logs 选项卡。日志显示 OpenShift 已在每个代理上建立了集群连接网桥。特别是，日志输出包含类似如下的行：

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

3.4.3. 将自定义资源更改应用到正在运行的代理部署

以下是有关应用自定义资源(CR)更改以运行代理部署的一些重要事项：

- 您无法动态更新 CR 中的 persistenceEnabled 属性。要更改此属性，将集群缩减为零个代理。删除现有 CR。然后，使用您的更改重新创建并重新部署 CR，同时指定部署大小。
- 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（即，通过在 CR 中设置 persistenceEnabled=true），您可能需要为代理 Pod 置备持久性卷(PV)来声明代理 Pod。如果您缩减代理部署的大小，Operator 会释放之前为被关闭的代理 Pod 声明的 PV。但是，如果您通过删除 CR 来*删除*代理部署，AMQ Broker Operator不会为在删除时仍在部署中的代理 Pod 释放 PVC。另外，这些未发布的 PV 不适用于任何新部署。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 发布持久性卷。
- 在 AMQ Broker 7.12 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。
  - 持久性存储部署中每个代理所需的持久性卷声明(PVC)的大小和存储类。
  - 部署 中每个代理的内存和 CPU 限制和请求。

- 在活跃的扩展事件中，您应用的任何进一步更改都由 Operator 排队，只有在扩展完成后执行。例如，假设您将部署的大小从四个代理缩减为一个。然后，在发生 scaledown 时，您也会更改代理管理员用户名和密码的值。在这种情况下，Operator 会对用户名和密码进行更改进行队列，直到部署使用一个活跃的代理运行为止。
- 所有 CR 更改 - 除更改部署大小外，或更改 acceptors、连接器或控制台的 **expose** 属性的值 - 会导致现有代理被重启。如果部署中有多个代理，则一次只重启一个代理。

3.5. 更改 OPERATOR 的日志记录级别

AMQ Broker Operator 的默认日志记录级别是 **info**，它会记录信息和错误消息。您可以更改默认日志记录级别，以增加或减少写入 Operator 日志的详细信息。

如果使用 OpenShift Container Platform 命令行界面安装 Operator，您可以在 Operator 配置文件 **operator.yaml** 中设置新的日志级别，可以是在安装 Operator 之前或之后。如果使用 Operator Hub，您可以在安装 Operator 后使用 OpenShift Container Platform Web 控制台在 Operator 订阅中设置日志级别。

Operator 的其他可用日志记录级别有：

错误

仅将错误消息写入日志。

debug

将所有消息写入日志，包括调试信息。

流程

1. 使用 OpenShift Container Platform 命令行界面：
   a. 您需要以集群管理员身份登录。例如：

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

   b. 如果没有安装 Operator，请完成以下步骤以更改日志级别。
      i. 在您下载并提取的 Operator 归档的 **deploy** 目录中，打开 **operator.yaml** 文件。
      ii. 将 **zap-log-level** 属性的值更改为 **debug** 或 **error**。例如：

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    control-plane: controller-manager
  name: amq-broker-controller-manager
  spec:
    containers:
    - args:
      - --zap-log-level=error
  ...
```

      iii. 保存 **operator.yaml** 文件。
      iv. 安装 Operator。

c. 如果已安装 Operator，请使用 sed 命令更改 deploy/operator.yaml 文件中的日志级别并重新部署 Operator。例如，以下命令可将日志级别从 info 改为 error 并重新部署 Operator：

```
$ sed 's/--zap-log-level=info/--zap-log-level=error/' deploy/operator.yaml | oc apply -f -
```

2. 使用 OpenShift Container Platform Web 控制台：

a. 以集群管理员身份登录 OpenShift Container Platform。

b. 在左侧窗格中，单击 Operators → Installed Operators。

c. 点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)Operator。

d. 点击 Subscriptions 选项卡。

e. 点 Actions。

f. 点 Edit Subscription。

g. 点 YAML 标签。
在控制台中，会打开 YAML 编辑器，供您编辑订阅。

h. 在 config 元素中，添加名为 ARGS 的环境变量，并指定 info、debug 或 error 的日志记录级别。在以下示例中，指定 debug 日志级别的 ARGS 环境变量传递给 Operator 容器。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
spec:
  ...
  config:
    env:
    - name: ARGS
      value: "--zap-log-level=debug"
  ...
```

i. 点 Save。

3.6. 为 OPERATOR 配置领导选举设置

您可以自定义 AMQ Broker Operator 用于领导选举机制的设置。

如果使用 OpenShift Container Platform 命令行界面安装 Operator，您可以在 Operator 配置文件 operator.yaml 中配置领导选举设置，可在安装前或安装后。如果使用 OperatorHub，您可以使用 OpenShift Container Platform Web 控制台在安装后在 Operator 订阅中配置领导选举设置。

流程

1. 使用 OpenShift Container Platform Web 控制台：

a. 以集群管理员身份登录 OpenShift Container Platform。

b. 在左侧窗格中，单击 Operators → Installed Operators。

c. 点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)Operator。

d. 点击 Subscriptions 选项卡。

e. 点 Actions。

f. 点 Edit Subscription。

g. 点 YAML 标签。
在控制台中，会打开 YAML 编辑器，供您编辑订阅。

h. 在 config 部分中，添加名为 ARGS 的环境变量，并在变量值中指定领导选举设置。例如：

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
spec:
  ..
  config:
    env:
    - name: ARGS
      value: "--lease-duration=18 --renew-deadline=12 --retry-period=3"
```

i. 点击 Save。

lease-duration

非领导操作器在尝试获取之前领导未续订的租期前等待的持续时间（以秒为单位）。默认值为 15。

renew-deadline

Operator 在尝试续订领导角色前等待持续时间（以秒为单位）。默认值为 10。

retry-period

Operator 在尝试获取和更新领导角色之间等待的时间（以秒为单位）。默认值为 2。

2. 使用 OpenShift Container Platform 命令行界面：

a. 您需要以集群管理员身份登录。例如：

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

b. 在您下载并提取的 operator 归档的 deploy 目录中，打开 operator.yaml 文件。

c. 设置领导选举设置的值。例如：

```
apiVersion: apps/v1
kind: Deployment
...
template
..
spec:
  containers:
  - args:
    - --lease-duration=60
    - --renew-deadline=40
    - --retry-period=5
..
```

d. 保存 operator.yaml 文件。

e. 如果已安装 Operator，请应用更新的设置。

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

f. 如果没有安装 Operator，请安装 Operator。

3.7. 查看代理部署的状态信息

您可以查看 OpenShift Container Platform 为代理部署报告的一系列标准条件的状态。您还可以查看代理部署的自定义资源(CR)中提供的其他状态信息。

流程

1. 为代理部署打开 CR 实例。
   a. 使用 OpenShift 命令行界面：
      i. 以具有查看代理部署的项目中的 CR 的用户身份登录 OpenShift Container Platform。
      ii. 查看部署的 CR。

```
oc get ActiveMQArtemis <CR instance name> -n <namespace> -o yaml
```

   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，以便在代理部署的项目中部署 CR。
      ii. 在左侧窗格中，点 Operators → Installed Operator。
      iii. 点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)operator。
      iv. 单击 ActiveMQ Artemis 选项卡。
      v. 单击 ActiveMQ Artemis 实例的名称。
2. 查看代理部署的 OpenShift Container Platform 条件的状态。
   a. 使用 OpenShift 命令行界面：
      i. 进入 CR 的 status 部分并查看条件详情。
   b. 使用 OpenShift Container Platform Web 控制台：
      i. 在 Details 选项卡中，向下滚动到 Conditions 部分。
      条件具有状态和类型。它也可能具有原因、消息和其他详情。如果条件满足，则条件的 status 值为 True，如果条件没有满足，则为 False ；如果条件的状态无法确定，则为 Unknown。Valid 条件也可以的状态为 Unknown，以在不影响代理部署的配置中标记 anomaly。如需更多信息，请参阅 第 2.7 节 “在自定义资源(CR)中验证镜像和版本配置”。

      为以下条件提供状态信息：

状况名称	显示... 的状态

状况名称	显示... 的状态
valid	CR 验证。如果 **Valid** 条件的状态为 **False**，Operator 不会完成协调并更新 StatefulSet，直到您首先解决导致 false 状态的问题。
Deployed	StatefulSet、Pod 和其他资源的可用性。
Ready	顶级条件，用于总结了其他详细条件。只有在其他条件没有状态为 **False** 时，**Ready** 条件的状态为 **True**。
BrokerPropertiesApplied	使用 **brokerProperties** 属性在 CR 中配置的属性。有关 **BrokerPropertiesApplied** 条件的更多信息，请参阅 第 4.23 节 “配置不在自定义资源定义中公开的项目”。
JaasPropertiesApplied	在 CR 中配置的 Java 身份验证和授权服务(JAAS)登录模块。有关 **JaasPropertiesApplied** 条件的更多信息，请参阅 第 4.3.1 节 “在 secret 中配置 JAAS 登录模块”。

3. **在 CR 的 status 部分查看代理部署的额外状态信息。此时会显示以下额外状态信息：**

   deploymentPlanSize

   部署中代理 Pod 数量。

   podstatus

   部署中每个代理 pod 的状态和名称。

   version

   代理的版本以及部署的代理和 init 容器镜像的 registry URL。

   Upgrade （升级）

   Operator 对部署应用主要、次版本、补丁和安全更新的功能，由 CR 中的 **spec.deploymentPlan.image 和 spec.version 属性的值决定。**

   - **如果 spec.deploymentPlan.image 属性指定代理容器镜像的 registry URL，则所有升级类型的状态都是 False，这意味着 Operator 无法升级现有容器镜像。**
   - **如果 spec.deploymentPlan.image 属性不在 CR 中，或者具有占位符值，则 spec.version 属性的配置会影响升级状态，如下所示：**
     - **securityUpdates 的状态为 True，无论是否配置了 spec.version 属性还是其值。**
     - **如果 spec.version 属性的值只有一个主版本和次版本，如 '7.10'，则 patchUpdates 的状态为 True，以便 Operator 可以升级到容器镜像的最新补丁版本。**
     - **如果 spec.version 属性的值 只有一个主要版本，如 '7'，则 minorUpdates**

的状态为 **True**，因此 **Operator** 可以升级到容器镜像的最新次版本和补丁版本。

- 如果 **spec.version** 属性不在 **CR** 中，则 **majorUpdates** 的状态为 **True**，因此可以部署任何可用的升级，包括从 **7.x.x** 升级到 **8.x.x**（如果此版本可用）。

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)实例配置代理部署前，您应该了解 Operator 如何生成代理配置。

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

Operator 在初始化每个 Pod 时运行一个名为 *Init Container* 的容器类型。在 OpenShift Container Platform 中，Init Containers 是应用程序容器之前运行的专用容器。Init 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 使用内置的 Init 容器。Init 容器使用部署的主 CR 实例来生成供每个代理应用程序容器使用的配置。

如果您在 CR 中指定地址设置，Operator 会生成一个默认配置，然后将该配置与 CR 中指定的配置合并或替换。下面的部分描述了此过程。

4.1.1. Operator 如何生成地址设置配置

如果您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前运行 Init 容器。Init 容器生成 默认 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. 如果您在自定义资源(CR)实例中也指定了地址设置配置，则初始容器进程将配置并将其转换为 XML。

3. 根据 CR 中的 applyRule 属性的值，Init *容器将* 初始容器 *合并* 或替换为您在 CR 中指定的配置。此合并或替换的结果是代理要使用的最终地址设置配置。

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

其他资源

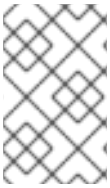
- 有关在 CR 中使用 applyRule 属性的示例，请参阅 第 4.2.4 节 “匹配基于 Operator 的代理部署中配置的地址设置”。

4.1.2. 代理 Pod 的目录结构

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

Operator 在初始化每个 Pod 时运行一个名为 *Init Container* 的容器类型。在 OpenShift Container Platform 中，Init Containers 是应用程序容器之前运行的专用容器。Init 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 容器将使用默认安装目录中所含的文件。此安装目录位于 Operator 挂载到代理 Pod 以及 Init Container 和 broker 容器共享的卷中。Init 容器用来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 JAR 和库。
<install_dir>/log	代理日志文件。
<install_dir>/tmp	临时 Web 应用程序文件。

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

其他资源

- 如需有关 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 2.6 节 “Operator 如何选择容器镜像”。
-

要了解如何构建并指定自定义初始容器镜像，请参阅 第 4.11 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，您可以使用两个独立的自定义资源(CR)实例来配置地址和队列及其关联的设置。

- 要在代理上创建地址和队列，您可以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，则地址 CRD 是您下载并提取的 Operator 安装的 deploy/crds 中的 broker_activemqartemisaddress_crd.yaml 文件。
  - 如果您使用 OperatorHub 安装 Operator，则地址 CRD 是 OpenShift Container Platform Web 控制台的 Administration → Custom Resource Definitions 下列出的 ActiveMQArtemisAddress CRD。
- 要配置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)实例中包括配置。
  - 如果您使用 OpenShift CLI 安装 Operator，主代理 CRD 是您下载和提取的 Operator 安装存档中的 broker_activemqartemis_crd.yaml 文件。
  - 如果您使用 OperatorHub 安装 Operator，则主要代理 CRD 是 OpenShift Container Platform Web 控制台的 Administration → Custom Resource Definitions 下列出的 ActiveMQArtemis CRD。

通常，您可以为 OpenShift Container Platform 上的代理部署配置的地址和队列设置 完全相当于 Linux 或 Windows 上的独立代理部署。但是，您应该注意，这些设置是*如何*配置的一些变化。这些区别在以下子部分进行了描述。

4.2.1. OpenShift 和独立代理部署之间的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，您需要向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。

- 用于配置项名称的格式在 OpenShift Container Platform 和独立代理部署之间有所不同。对于 OpenShift Container Platform 部署，配置项名称在 *camel case* 中，例如 defaultQueueRoutingType。相反，独立部署的配置项名称为小写，并使用短划线(-)分隔符，如 default-queue-routing-type。

  下表显示了一些进一步的命名差异示例。

独立代理部署的配置项目	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

其他资源

- 有关为 OpenShift Container Platform 代理部署创建地址和队列和匹配设置的示例，请参阅：
  - 为 OpenShift Container Platform 上的代理部署创建地址和队列
  - 将地址设置与在 OpenShift Container Platform 上为代理部署配置的地址匹配
- 要了解 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综合信息，*请参阅配置 AMQ Broker* 中的配置 地址和队列。您可以使用此信息为 OpenShift Container Platform 上的代理部署创建对等配置。

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)实例将地址和相关队列添加到基于 Operator 的代理部署中。

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，在每个情况下指定新地址和/或队列名称。另外，每个 CR 实例的 name 属性必须是唯一的。

先决条件

- 您必须已安装了 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 的两个替代方法的详情，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 实例创建基本代理部署。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，以便在代理部署的项目中部署 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和提取的 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，以便在代理部署的项目中部署 CR。

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

iii. 单击 **ActiveMQArtemisAddresss 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，添加行以定义地址、队列和路由类型。例如：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisAddress
metadata:
  name: myAddressDeployment0
  namespace: myProject
spec:
  ...
  addressName: myAddress0
  queueName: myQueue0
  routingType: anycast
  ...
```

上述配置定义了名为 **myAddress0** 的地址，队列名为 **myQueue0** 和 **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 实例时才指定值。您应该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。

4.2.3. 删除基于 Operator 的代理部署的地址和队列

以下流程演示了如何使用自定义资源(CR)实例从基于 Operator 的代理部署中删除地址和相关队列。

流程

1. 确保有一个带有详细信息的地址 CR 文件，例如：您要删除的地址和 队列的名称、addressName 和 queueName。例如：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisAddress
metadata:
  name: myAddressDeployment0
  namespace: myProject
spec:
  ...
  addressName: myAddress0
  queueName: myQueue0
  routingType: anycast
  ...
```

2. 在 address CR 的 spec 部分中，添加 removeFromBrokerOnDelete 属性，并设置为 true 的值。

```
..
spec:
  addressName: myAddress1
  queueName: myQueue1
  routingType: anycast
  removeFromBrokerOnDelete: true
```

将 removeFromBrokerOnDelete 属性设置为 true 会导致 Operator 在删除地址 CR 时删除所有代理的地址和任何关联的消息。

3. 应用更新的地址 CR，为您要删除的地址设置 removeFromBrokerOnDelete 属性。

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

4. 删除地址 CR，从部署中的代理中删除地址。

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

4.2.4. 匹配基于 Operator 的代理部署中配置的地址设置

如果向客户端发送消息失败，您可能不希望代理持续尝试发送消息。要防止无限发送尝试，您可以定义一个 *死信地址* 和一个关联的 *死信队列*。在进行指定数量的发送尝试后，代理会从其原始队列中删除未发送的消息，并将消息发送到配置的死信地址。系统管理员稍后可能会消耗从死信队列中未发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

- 您创建了 ActiveMQArtemis CR 实例来部署代理。如需更多信息，请参阅 第 3.4.1 节 “部署基本代理实例”。
- 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。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. 开始配置地址 CR 实例，以添加死信地址和队列，以接收部署中每个代理的未发送消息。

   a. 使用 OpenShift 命令行界面：

      i. 以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 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和提取的 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，以便在代理部署的项目中部署 CR。

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

      iii. 单击 ActiveMQArtemisAddresss CRD。

      iv. 点 实例 选项卡。

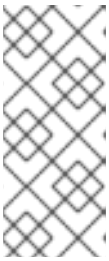
      v. 单击 Create ActiveMQArtemisAddress。

      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，以接收未发送的消息。例如：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisAddress
metadata:
  name: ex-aaoaddress
spec:
  ...
  addressName: myDeadLetterAddress
  queueName: myDeadLetterQueue
  routingType: anycast
  ...
```

以上配置定义了名为 **myDeadLetterAddress** 的死信地址，其死信队列名为 **myDeadLetterQueue** 和 **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** 实例时才指定值。您应该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. **编辑代理部署的主代理 CR 实例。**

   a. **使用 OpenShift 命令行界面：**

      i. **以具有为代理部署的项目中编辑和部署 CR 的用户身份登录 OpenShift。**

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

      ii. **编辑 CR。**

      ```
      oc edit ActiveMQArtemis <CR instance name> -n <namespace>
      ```

   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，以便在代理部署的项目中部署 CR。**

      ii. **在左侧窗格中，点 Operators → Installed Operator。**

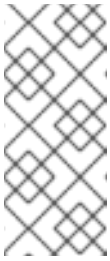
      iii. **点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator。**

      iv. **点 AMQ Broker 选项卡。**

      v. **单击 ActiveMQArtemis 实例名称的名称。**

      vi. **点 YAML 标签。**

      在控制台中，会打开 YAML 编辑器，供您编辑 CR 实例。



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 实例时才指定值。您应该指定的值是代理部署的 OpenShift 项目的名称。

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

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

6. 将 match 属性的单个实例添加到 addressSetting 块。指定地址匹配表达式。例如：

```
spec:
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
  addressSettings:
    addressSetting:
    - match: myAddress
```

match

指定代理应用配置的地址或地址 *集合*。在本例中，match 属性的值对应于一个名为 myAddress 的单个地址。

7. 添加与未发送消息相关的属性并指定值。例如：

```
spec:
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
```

```
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
  addressSettings:
    addressSetting:
    - match: myAddress
      deadLetterAddress: myDeadLetterAddress
      maxDeliveryAttempts: 5
```

deadLetterAddress

代理发送未发送的消息的地址。

maxDeliveryAttempts

代理在将消息移至配置的死信地址前进行的最大发送尝试次数。

在前面的示例中，如果代理有五次失败尝试向以 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）将类似的配置应用到另一个地址或一组地址。例如：

```
spec:
  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 属性的值包含一个哈希通配符。通配符字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，您必须将该值放在单引号中，例如 'myOtherAddresses#'。

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

```
spec:
  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 如何应用添加到 CR 的配置，以进行每个匹配地址或一组地址。您可以指定的值有：

merge_all

- 对于 CR 中指定的地址设置，以及与 相同地址或一组地址匹配的默认配置：
  - 将默认配置中指定的任何属性值替换为 CR 中指定的值。
  - 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括每个。
- 对于在 CR 中指定的地址设置，或者唯一匹配一个特定地址或一组地址的默认配置，将它们包括在最终合并的配置中。

merge_replace

- 对于 CR 中指定的地址设置 和 与相同地址集合匹配的默认配置，请在最终合并的配置中包含 CR 中指定的设置。不要 包括默认配置中指定的任何属性，即使 CR 中没有指定这些属性。

- 对于在 CR 中指定的地址设置，或者唯一匹配一个特定地址或一组地址的默认配置，将它们包括在最终合并的配置中。

replace_all

使用在 CR 中指定的内容替换默认配置中指定的所有地址设置最后，合并的配置与 CR 中指定的配置完全相同。

注意

如果您没有在 CR 中显式包含 applyRule 属性，Operator 将使用默认值 merge_all。

10. 保存 CR 实例。

其他资源

- 要了解 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”。

- 如果您使用 OpenShift 命令行界面(CLI)安装 AMQ Broker Operator，则您下载并提取的安装存档包含配置地址设置的一些额外示例。在安装存档的 deploy/examples 文件夹中，请参阅：

  - 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综合信息，*请参阅配置 AMQ Broker* 中的配置 地址和队列。您可以使用此信息为 OpenShift Container Platform 上的代理部署创建

对等配置。

- 如需有关 OpenShift Container Platform 中初始容器的更多信息，请参阅 OpenShift Container Platform 文档中的 部署 pod 前使用初始容器执行任务。

4.3. 配置身份验证和授权

默认情况下，AMQ Broker 使用 Java 认证和授权服务(JAAS)属性登录模块来验证和授权用户。默认 JAAS 登录模块的配置存储在每个代理 Pod 上的 /home/jboss/amq-broker/etc/login.config 文件中，并从同一目录中的 artemis-users.properties 和 artemis-roles.properties 文件中读取用户和角色信息。您可以通过更新 ActiveMQArtemisSecurity 自定义资源(CR)，将用户和角色信息添加到默认登录模块中的属性文件中。

更新 ActiveMQArtemisSecurity CR 的替代方法是向默认属性文件添加用户和角色信息，方法是在 secret 中配置一个或多个 JAAS 登录模块。此 secret 作为文件挂载到每个代理 Pod 上。与使用 ActiveMQArtemisSecurity CR 来添加用户和角色信息相比，在 secret 中配置 JAAS 登录模块具有以下优点：

- 如果在 secret 中配置属性登录模块，则代理不需要在每次更新属性文件时重启。例如，当您将新用户添加到属性文件中并更新 secret 时，更改会在不需要重启代理的情况下生效。
- 您可以配置 ActiveMQArtemisSecurity CRD 中未定义的 JAAS 登录模块来验证用户。例如，您可以配置 LDAP 登录模块或任何其他 JAAS 登录模块。

以下部分描述了为 AMQ Broker 配置身份验证和授权的方法。

4.3.1. 在 secret 中配置 JAAS 登录模块

您可以在 secret 中配置 JAAS 登录模块，以使用 AMQ Broker 验证用户身份。创建 secret 后，您必须在主代理自定义资源(CR)中添加对 secret 的引用，并在 CR 中配置权限以授予用户对 AMQ Broker 的访问权限。

流程

1. 使用新的 JAAS 登录模块配置创建一个文本文件，并将文件保存为 login.config。将文件保存为 login.config，正确的密钥会插入到您从文本文件创建的 secret 中。以下是登录模块配置示例：

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
    reload=true
    org.apache.activemq.jaas.properties.user="new-users.properties"
    org.apache.activemq.jaas.properties.role="new-roles.properties";

  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
    reload=false
    org.apache.activemq.jaas.properties.user="artemis-users.properties"
    org.apache.activemq.jaas.properties.role="artemis-roles.properties"
    baseDir="/home/jboss/amq-broker/etc";
};
```

在 secret 中配置 JAAS 登录模块并在 CR 中添加对 secret 的引用后，AMQ Broker 不再使用默认的登录模块。但是，Operator 需要在默认登录模块中引用的 artemis-users.properties 文件中的用户与代理进行身份验证。要确保 Operator 可以在配置新的 JAAS 登录模块后与代理进行身份验证，您必须：

- **在新的登录模块配置中包含默认属性登录模块，如上例中所示。在示例中，默认属性登录模块使用 artemis-users.properties 和 artemis-roles.properties 文件。如果您在新的登录模块配置中包含默认登录模块，您必须将 baseDir 设置为 /home/jboss/amq-broker/etc 目录，其中包含每个代理 Pod 上的默认属性文件。**

- **将 Operator 所需的用户和角色信息添加到新登录模块配置中引用的属性文件中。您可以从代理 Pod 上的 /home/jboss/amq-broker/etc 目录中 复制此信息。**

注意

只有在代理首次调用登录模块时，才会加载登录模块中引用的属性文件。代理会按照 login.config 文件中列出的顺序调用登录模块，直到它找到登录模块来验证用户。通过将包含 Operator 使用的凭证的登录模块放在 login.config 文件的末尾，当代理验证 Operator 时，所有前面的登录模块都会被调用。因此，任何状态消息都指出该属性文件在代理中不可见。

2. **如果您创建的 login.config 文件包含属性登录模块，请确保模块中指定的用户和角色文件包含用户和角色信息。例如：**

new-users.properties

```
ruben=ruben01!
anne=anne01!
rick=rick01!
bob=bob01!
```

new-roles.properties

```
admin=ruben, rick
group1=bob
group2=anne
```

3. 使用 oc create secret 命令从使用新登录模块配置创建的文本文件中创建 secret。如果登录模块配置包含属性登录模块，请在机密中包含相关的用户和角色文件。例如：

```
oc create secret generic custom-jaas-config --from-file=login.config --from-file=new-users.properties --from-file=new-roles.properties
```

注意

secret 名称必须具有 -jaas-config 后缀，以便 Operator 可以识别 secret 包含登录模块配置，并将任何更新传播到每个代理 Pod。

有关如何创建 secret 的更多信息，请参阅 Kubernetes 文档中的 Secret。

4. 将您创建的 secret 添加到代理部署的自定义资源(CR)实例中。

a. 使用 OpenShift 命令行界面：

i. 以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。

ii. 编辑部署的 CR。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

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，以便在代理部署的项目中部署 CR。

ii. 在左侧窗格中，点 Operators → Installed Operator。

:::

iii. 点 **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator**。

iv. 点 **AMQ Broker** 选项卡。

v. 单击 **ActiveMQArtemis** 实例名称的名称。

vi. 点 **YAML** 标签。

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

5. 创建一个 **extraMounts** 元素和一个 **secrets** 元素，并添加 **secret** 的名称。以下示例将名为 **custom-jaas-config** 的 **secret** 添加到 **CR** 中。

```
deploymentPlan:
 ...
 extraMounts:
  secrets:
  - "custom-jaas-config"
 ...
```

6. 在 **CR** 中，为代理配置的角色授予权限。

a. 在 **CR** 的 **spec** 部分，添加一个 **brokerProperties** 元素并添加权限。您可以为单个地址授予角色权限。或者，您可以使用 **#** 符号指定通配符匹配，为所有地址授予角色权限。例如：

```
spec:
  ...
  brokerProperties:
  - securityRoles.#.group2.send=true
  - securityRoles.#.group1.consume=true
  - securityRoles.#.group1.createAddress=true
  - securityRoles.#.group1.createNonDurableQueue=true
  - securityRoles.#.group1.browse=true
  ...
```

在示例中，**group2** 角色分配为所有地址 发送 权限，并且分配 **group1** 角色 消耗、**createAddress**、**createNonDurableQueue** 并将权限浏览到 所有地址。

7. 保存 CR。

Operator 在每个 Pod 上的 /amq/extra/secrets/*secret 名称* 目录中挂载 login.config 文件，并将代理 JVM 配置为读取挂载的 login.config 文件，而不是默认的 login.config 文件。如果 login.config 文件包含属性登录模块，则引用的用户和角色属性文件也会挂载到每个 Pod 上。

8. 查看 CR 中的状态信息，以验证部署中的代理是否使用 secret 中的 JAAS 登录模块进行身份验证。

a. 使用 OpenShift 命令行界面：

i. 获取代理 CR 中的状态条件。

```
$ oc get activemqartemis -o yaml
```

b. 使用 OpenShift Web 控制台：

i. 在 CR 中，进入到 status 部分。

c. 在状态信息中，验证存在 JaasPropertiesApplied 类型，这表示代理正在使用 secret 中配置的 JAAS 登录模块。例如：

```
- lastTransitionTime: "2023-02-06T20:50:01Z"
  message: ""
  reason: Applied
  status: "True"
  type: JaasPropertiesApplied
```

当您更新 secret 中的任何文件时，reason 字段的值会显示 OutofSync，直到 OpenShift Container Platform 将 secret 中的最新文件传播到每个代理 Pod。例如，如果您在 new-users-properties 文件中添加新用户并更新 secret，您会看到以下状态信息，直到更新的文件被传播到每个 Pod：

```
- lastTransitionTime: "2023-02-06T20:55:20Z"
  message: 'new-users.properties status out of sync, expected: 287641156, current:
2177044732'
  reason: OutOfSync
  status: "False"
  type: JaasPropertiesApplied
```

9. 当您在 secret 中引用的属性文件中更新用户或组信息时，请使用 oc set data 命令更新 secret。您必须再次读取到 secret 的所有文件，包括 login.config 文件。例如，如果您向此流程前面创建的 new-users.properties 文件中添加新用户，请使用以下命令更新 custom-jaas-config secret：

```
oc set data secret/custom-jaas-config --from-file=login.config=login.config --from-file=new-users.properties=new-users.properties --from-file=new-roles.properties=new-roles.properties
```

注意

代理 JVM 仅在 login.config 文件中读取配置。如果您更改了 login.config 文件中的配置，例如要添加新的登录模块并更新 secret，则代理不会在代理重启前使用新配置。

其它资源

第 8.2 节 “JAAS 登录模块配置示例”

第 8.3 节 “示例：将 AMQ Broker 配置为使用 Red Hat Single Sign-On”

有关 JAAS 登录模块格式的详情，请参考 JAAS 登录配置文件。

4.3.2. 使用安全自定义资源(CR)配置默认的 JAAS 登录模块

您可以使用 ActiveMQArtemisSecurity 自定义资源(CR)在默认的 JAAS 属性登录模块中配置用户和角色信息，以使用 AMQ Broker 验证用户的身份。有关使用 secret 在 AMQ Broker 中配置身份验证和授权的替代方法，请参考 第 4.3.1 节 “在 secret 中配置 JAAS 登录模块”。

4.3.2.1. 使用安全自定义资源(CR)配置默认的 JAAS 登录模块

以下流程演示了如何使用安全自定义资源(CR)配置默认的 JAAS 登录模块。

先决条件

- 您必须已安装了 AMQ Broker Operator。有关安装 Operator 的两个替代方法的详情，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性，如安全 代理中所述

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，则会重启代理 pod 以接受新配置。

1. 开始配置自定义资源(CR)实例，以定义用户以及代理部署的关联安全配置。

a. 使用 OpenShift 命令行界面：

i. 以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。

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

ii. 编辑部署的 CR。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

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，以便在代理部署的项目中部署 CR。

ii. 在左侧窗格中，点 Operators → Installed Operator。

iii. 点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator。

iv. 点 **AMQ Broker** 选项卡。

v. 单击 **ActiveMQArtemis** 实例名称的名称

vi. 点 **YAML** 标签。

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，添加行以定义用户和角色。例如：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisSecurity
metadata:
  name: ex-prop
spec:
  loginModules:
    propertiesLoginModules:
      - name: "prop-module"
        users:
          - name: "sam"
            password: "samspassword"
            roles:
              - "sender"
          - name: "rob"
            password: "robspassword"
            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"
       ...
```

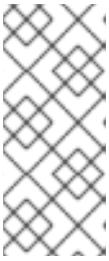
注意

始终为上例中的元素指定值。例如，如果您没有为 **securityDomains.brokerDomain** 或角色值指定值，则生成的配置可能会导致意外的结果。

前面的配置定义了两个用户：

- 名为 **prop-module** 的属性LoginModule，用于定义名为 **sam** 的用户，角色名为 **sender**。
- 名为 **prop-module** 的属性LoginModule，定义名为 **rob** 的用户，角色名为 **receiver**。

这些角色的属性在 **securityDomains** 的 **brokerDomain** 和 **broker** 部分中定义。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，配置适用于当前命名空间中 CR 定义的所有部署代理。要将配置限制到特定的代理部署，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 实例时才指定值。您应该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/security_custom_resource_instance>.yaml
```

b. 使用 OpenShift Web 控制台：

i. 配置完 CR 后，点 Create。

其他资源

- 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”

4.3.2.2. 将用户密码存储在 secret 中

在 第 4.3.2.1 节 “使用安全自定义资源(CR)配置默认的 JAAS 登录模块” 流程中，用户密码以明文形式存储在 ActiveMQArtemisSecurity CR 中。如果您不想在 CR 中以明文形式存储密码，您可以从 CR 中排除密码并将其存储在 secret 中。应用 CR 时，Operator 从 secret 检索每个用户的密码，并将其插入到代理 pod 上的 artemis-users.properties 文件中。

流程

1. 使用 oc create secret 命令创建 secret，并添加每个用户名和密码。secret 名称必须遵循 security-properties-*module name* 的命名约定，其中 *module name* 是 CR 中配置的登录模块的名称。例如：

```
oc create secret generic security-properties-prop-module \
  --from-literal=sam=samspassword \
  --from-literal=rob=robspassword
```

2. 在 CR 的 spec 部分，添加您在 secret 中指定的用户名以及角色信息，但不包含每个用户的密码。例如：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisSecurity
```

```
metadata:
  name: ex-prop
spec:
  loginModules:
    propertiesLoginModules:
      - name: "prop-module"
        users:
          - name: "sam"
            roles:
              - "sender"
          - name: "rob"
            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"
              ...
```

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。

其他资源

如需有关 OpenShift Container Platform 中 secret 的更多信息，请参阅 OpenShift Container Platform 文档中的 向 pod 提供敏感数据。

4.4. 添加第三方 JAR 文件

您可以在运行时将第三方 JAR 文件提供给 AMQ Broker。例如，如果您希望代理将消息存储在 JDBC 数据库中，您可以将代理配置为加载数据库所需的第三方 JAR 文件。

您必须配置 Operator，使第三方 JAR 文件在每个代理 pod 上的挂载卷上可用，并将 JAR 文件的卷路径添加到代理的 Java 类路径中。

如果 JAR 文件的大小小于 1 MB，您可以将 JAR 文件添加到 secret 或 configmap 中，并将 Operator 配置为将 JAR 文件挂载到每个代理 pod 上。如果 JAR 文件大于 secret 和 configmap 的 1 MB 限制，您可以将 Operator 配置为在每个代理 pod 上挂载共享卷，并将 JAR 文件下载到该卷。

4.4.1. 使用 secret 或配置映射在代理 pod 上挂载 JAR 文件

如果 JAR 文件小于 1 MB，您可以使用 secret 或配置映射来在每个代理 pod 上挂载第三方 JAR 文件。您还必须修改代理的 Java 类路径，以便从挂载的位置加载 JAR 文件。

以下流程假定您使用 secret 来挂载 JAR 文件。

流程

1. 使用 oc create secret 命令创建一个包含要添加的第三方 JAR 文件的 secret。例如：

```
oc create secret generic log4j-template --from-file=log4j-layout-template-json-2.22.1.jar
```

有关如何创建 secret 的更多信息，请参阅 Kubernetes 文档中的 Secret。

2. 编辑代理部署的 CR，并将 Operator 配置为在每个代理 pod 上挂载包含第三方 JAR 文件的 secret。例如，以下配置挂载名为 log4j-template 的 secret。

```
deploymentPlan:
  ...
  extraMounts:
    secrets:
    - "log4j-template"
  ...
```

JAR 文件挂载到每个代理 pod 上的 /amq/extra/secrets/*secret 名称* 目录中。例如，/amq/extra/secrets/postgresql-driver/log4j-template.jar。

3. 创建 ARTEMIS_EXTRA_LIBS 环境变量来扩展代理的 Java 类路径，以便代理从每个 pod 上挂载的目录加载 JAR 文件。例如：

```
spec:
  ...
  env:
  - name: ARTEMIS_EXTRA_LIBS
    value: /amq/extra/secrets/log4j-template
```

4. 保存 CR。

4.4.2. 将 JAR 文件下载到每个代理 pod 上的卷

如果 JAR 文件大于 1 MB，则无法使用 secret 或配置映射将 JAR 文件挂载到每个代理 pod 上。相反，您可以将 Operator 配置为将 JAR 文件下载到 Operator 在每个代理 pod 上挂载的持久性卷中。

先决条件

持久性卷可用于挂载到每个代理 pod。

流程

1. 编辑用于代理部署的 ActiveMQArtemis CR。

2. 在 broker CR 中，使用 extraMounts 和 extraVolumeMounts 属性添加持久性卷并在每个代理 pod 上挂载卷。例如：

```
deploymentPlan:
  ...
  extraMounts:
  - name: extra-volume
    persistentVolumeClaim:
      claimName: extra-jars
  extraVolumeMounts:
  - name: extra-volume
    mountPath: /opt/extra-lib
  ...
```

3. 使用 resourceTemplates 属性为部署自定义 StatefulSet 资源。在自定义中，使用 init 容器挂载您在每个 pod 上创建的 extra-volume 卷，并将 JAR 文件下载到卷。例如：

```
spec:
  ...
  resourceTemplates:
  - selector:
      kind: StatefulSet
    patch:
      kind: StatefulSet
      spec:
        template:
          spec:
            initContainers:
            - name: mysql-jdbc-driver-init
              volumeMounts:
              - mountPath: /opt/extra-lib
                name: extra-volume
              image: curlimages/curl:8.6.0
              command:
              - /bin/sh
              args:
              - -c
              - "if ! [ -f /opt/extra-lib/mysql-connector.jar ]; then curl
https://repo1.maven.org/maven2/mysql/mysql-connector-java/8.0.23/mysql-connector-
java-8.0.23.jar --output /opt/extra-lib/mysql-connector.jar ; fi"
```

在示例中，如果该文件还没有位于卷中，则使用 curl 镜像将 mysql-connector.jar 文件下载到卷的挂载路径 /opt/extra-lib。

4. 创建 ARTEMIS_EXTRA_LIBS 环境变量来扩展代理的 Java 类路径，以便代理从共享卷加载

JAR 文件。例如：

```
spec:
  ...
  env:
  - name: ARTEMIS_EXTRA_LIBS
    value: /opt/extra-lib
```

5. 保存 CR。

4.5. 配置消息持久性

默认情况下，AMQ Broker 不会保留（即存储）消息数据。AMQ Broker 有两个选项用于持久性消息数据：

- 在日志中持久保留消息。如果您启用持久性，这是保留消息的默认方法。基于日志的持久性是一种高性能选项，可将消息写入文件系统上的日志。
- 在数据库中持久保留消息。此选项使用 Java 数据库连接(JDBC)连接将消息保留至您选择的数据库。

注意

有关 AMQ Broker 支持哪些数据库和网络文件系统的当前信息，请参阅红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

4.5.1. 配置基于日志的持久性

当您启用持久性时，默认情况下消息会在日志文件中保留。

流程

1. 编辑代理部署的 ActiveMQArtemis 自定义资源(CR)。
2. 将 persistenceEnabled 属性设置为 true。例如：

```
spec:
```

```
...
deploymentPlan:
  persistenceEnabled: true
...
```

3. 保存 CR。

4.5.2. 配置数据库持久性

您可以使用 Java 数据库连接(JDBC)连接将 AMQ Broker 配置为保留数据库中的信息。

当您在数据库中持久保留消息数据时，代理使用 Java 数据库连接(JDBC)连接将消息和绑定数据存储在数据库表中。表中的数据使用 AMQ Broker 日志编码进行编码。有关支持的数据库的详情，请参考红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

重要

管理员可以选择根据组织更广泛的 IT 基础架构要求将消息数据存储在数据库中。但是，使用数据库可能会对消息传递系统性能造成负面影响。具体来说，通过 JDBC 将消息传递数据写入数据库表可为代理创建显著的性能开销。

前提条件

- 用于 AMQ Broker 的专用数据库。
- 在运行时代理可以使用所需的 JDBC 驱动程序 JAR 文件。有关如何在运行时将 JAR 文件提供给代理的详情，请参考 第 4.4 节 “添加第三方 JAR 文件”。
- 部署只有一个代理实例。为确保部署只有一个代理实例，请确保 deployment.size 属性不在 ActiveMQArtemis 自定义资源(CR)中。当 CR 省略 deployment.size 属性时，会部署一个代理实例。

流程

1. 编辑代理部署的 ActiveMQArtemis 自定义资源(CR)。
2. 使用 brokerProperties 属性启用 JDBC 数据库持久性。例如：

```
spec:
  ...
  brokerProperties:
  - storeConfiguration=DATABASE
  - storeConfiguration.jdbcDriverClassName=<class name>
  - storeConfiguration.jdbcConnectionUrl=jdbc:<URL>
  - HAPolicyConfiguration=SHARED_STORE_PRIMARY
  ...
```

storeConfiguration

指定 DATABASE 值，以将消息持久保留到 JDBC 数据库。

storeConfiguration.jdbcDriverClassName

JDBC 数据库驱动程序的完全限定类名称。例如，org.postgresql.Driver。

有关支持的数据库的详情，请参考红帽客户门户网站中的 Red Hat AMQ 7 支持的配置。

storeConfiguration.jdbcConnectionUrl

您的数据库服务器的完整 JDBC 连接 URL，包括数据库名称和所有配置参数。例如：

```
jdbc:postgresql://postgresql-service.default.svc.cluster.local:5432/postgres?user=postgres&password=postgres
```

在示例中，数据库名称是 postgres。

HAPolicyConfiguration

设置为 SHARED_STORE_PRIMARY，以确保代理使用 JDBC 租期锁定来保护数据库表不受多个代理的并发访问。如果第二个代理实例被意外部署，则租期锁定会阻止第二个代理写入数据库。

3. （可选）根据需要更改以下属性的默认值：

storeConfiguration.jdbcNetworkTimeout

JDBC 网络连接超时，以毫秒为单位。默认值为 20000 毫秒。

storeConfiguration.jdbcLockRenewPeriod

当前 JDBC 锁的续订周期的长度，以毫秒为单位。当这个时间过后，代理可以续订锁定。设置一个比 storeConfiguration.jdbcLockExpiration 值小于 storeConfiguration.jdbcLockExpiration 的值的值，以便代理有足够的时间扩展租期，同时还为代理提供在发生连接问题时尝试续订锁定的时间。默认值为 2000 毫秒。

storeConfiguration.jdbcLockExpiration

以毫秒为单位，当前 JDBC 锁定被视为拥有的时间（即获取或续订），即使 storeConfiguration.jdbcLockRenewPeriod 的值已过。代理定期尝试根据 storeConfiguration.jdbcLockRenewPeriod 的值续订其拥有的锁定。如果代理无法续订锁定，例如，因为连接问题，代理会不断尝试续订锁定，直到 storeConfiguration.jdbcLockExpiration 的值自上次成功获取或续订后才会传递。上述续订行为的例外是另一个代理获取锁定时。如果数据库管理系统(DBMS)和代理之间有时间不对齐，或者对于垃圾回收有长时间暂停，则会出现这种情况。在这种情况下，最初拥有的锁定的代理会考虑丢失锁定，且不会尝试续订它。如果当前拥有过期时间后的代理没有续订 JDBC 锁定，则另一个代理可以建立 JDBC 锁定。默认值为 20000 毫秒。

storeConfiguration.jdbcJournalSyncPeriod

持续时间，以毫秒为单位，代理日志与 JDBC 同步。默认值为 5 毫秒。

storeConfiguration.jdbcMaxPageSizeBytes

当 AMQ Broker 将消息保留到 JDBC 数据库时，每个页文件的最大大小（以字节为单位）。默认值为 102400，其值为 100KB。您指定的值也支持字节表示法，如 "K" "MB" 和 "GB"。

4. 保存 CR。

4.6. 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存储，则需要手动置备持久性卷(PV)，并确保这些卷可以使用持久性卷声明(PVC)来声明它们。如果要创建带有持久性存储的两个代理集群，例如，您需要有两个 PV 可用。

重要

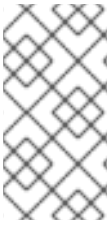
当您在 OpenShift Container Platform 中手动置备 PV 时，请确保将每个 PV 的重新声明策略设置为 Retain。如果 PV 的 reclaim 策略没有设置为 Retain，且 Operator 用于声明 PV 的 PVC 被删除，则 PV 也会被删除。删除 PV 会导致卷中的任何数据丢失。如需更多信息，请参阅 OpenShift Container Platform 文档中的了解持久性存储。

默认情况下，PVC 从为集群配置的默认存储类获取每个代理的 2 GiB 存储。您可以覆盖 PVC 中请求的

默认大小和存储类，但只有在首次部署 CR 前 在 CR 中配置新值。

4.6.1. 配置代理存储大小和存储类

以下流程演示了如何为代理部署配置自定义资源(CR)实例，以指定每个代理用于持久性消息存储所需的持久性卷声明(PVC)的大小和存储类。



注意

如果在部署 AMQ Broker 后更改 CR 中的存储配置，则不会将更新的配置应用到现有 Pod。但是，更新后的配置将应用到扩展部署时创建的新 Pod。

先决条件

- 您应该熟悉如何使用 CR 实例创建基本代理部署。请参阅 第 3.4.1 节 “部署基本代理实例”。
- 您必须已经置备了持久性卷(PV)，并可供 Operator 声明。例如，如果要创建带有持久性存储的两个代理集群，则需要有两个 PV 可用。

  如需有关置备持久性存储的更多信息，请参阅 OpenShift Container Platform 文档中的了解持久性存储。

流程

1. 开始为代理部署配置自定义资源(CR)实例。
   a. 使用 OpenShift 命令行界面：
      i. 以具有在您要创建部署的项目中部署 CR 的用户身份登录 OpenShift。

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

      ii. 打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 Operator 安装的 deploy/crs 目录中。

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有在您要创建部署的项目中部署 CR 的用户身份登录控制台。

ii. 根据主代理 CRD 启动新的 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。

对于基本代理部署，配置可能类似如下。

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

观察 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性被设置为 占位符 的默认值。此值表示，默认情况下 image 属性没有指定用于部署的代理容器镜像。要了解 Operator 如何确定要使用的适当代理容器镜像，请参阅 第 2.6 节 “Operator 如何选择容器镜像”。

2. 要指定代理存储大小，在 CR 的 deploymentPlan 部分中，添加一个 storage 部分。添加 size 属性并指定值。例如：

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

storage.size

每个代理 Pod 需要的持久性卷声明(PVC)的大小，以字节为单位。此属性仅在 persistenceEnabled 设为 true 时才适用。您指定的值必须使用字节表示法（如 K、M、G）或二进制等效的二进制(Ki、Mi、Gi)包含单元。

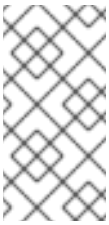
3. 要指定每个代理 Pod 需要的存储类，在 storage 部分添加 storageClassName 属性并指定一个值。例如：

```
spec:
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    storage:
      size: 4Gi
      storageClassName: gp3
```

storage.storageClassName

在持久性卷声明(PVC)中请求的存储类的名称。存储类为管理员提供了一种描述和分类可用存储的方法。例如，不同的存储类可能会映射到特定的服务质量级别、备份策略等。

如果没有指定存储类，PVC 会声明一个带有为集群配置默认存储类的持久性卷。



注意

如果您指定了存储类，只有在卷的存储类与指定存储类匹配时，才会由 PVC 声明持久性卷。

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。

4.7. 为基于 OPERATOR 的代理部署配置资源限值和请求

当您创建基于 Operator 的代理部署时，部署中的代理 Pod 在 OpenShift 集群的节点中在 StatefulSet 中运行。您可以为部署配置自定义资源(CR)实例，以指定每个 Pod 中运行的代理容器使用的 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 满意的性能。

重要

- 在首次部署 CR 之前，您必须为代理部署的 CR 实例添加限制和请求配置。您无法将 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，因为它们取决于您的特定消息传递系统用例和您实施的结果架构。但是，*建议您* 在开发环境中测试和调整这些值，然后再为生产环境配置这些值。
- Operator 在初始化每个代理 Pod 时运行一个名为 *Init Container* 的容器类型。每个代理容器配置的任何资源限值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，这个值是容器可以消耗的主机节点 CPU 的最大数量。如果代理容器尝试超过指定的 CPU 限制，OpenShift 会节流容器。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 会终止容器。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，将代理 Pod 绑定到具有足够计算资源的节点。

CPU 请求值是代理容器需要运行 *的最小* CPU 量。但是，如果节点上没有 CPU 争用，则容器可以使用所有可用的 CPU。如果指定了 CPU 限制，容器不能超过那个 CPU 用量。如果节点上有 CPU 争用，则 CPU 请求值为 OpenShift 提供了所有容器的 CPU 使用量权衡方法。

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，将代理 Pod 绑定到具有足够计算资源的节点。

内存请求值是代理容器需要运行的 *最小内存量*。但是，容器可以尽可能消耗可用内存。如果指定

了内存限制，则代理容器不能超过该内存用量。

CPU 以名为 millicores 的单位来衡量。OpenShift 集群中的每个节点检查操作系统，以确定节点上的 CPU 内核数。然后，节点将该值乘以 1000 以表示总容量。例如，如果某个节点有两个内核，节点的 CPU 容量表示为 2000m。因此，如果您想要使用单核中的一个，请指定 100m 值。

内存以字节为单位。您可以使用字节表示法(E、P、T、G、M、K)或二进制等效的二进制(Ei、Ti、Ti、Gi、Mi、Ki)来指定值。您指定的值必须包含一个单元。

4.7.1. 配置代理资源限制和请求

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

重要

- 在首次部署 CR 之前，您必须为代理部署的 CR 实例添加限制和请求配置。您无法将 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，因为它们取决于您的特定消息传递系统用例和您实施的结果架构。但是，*建议您* 在开发环境中测试和调整这些值，然后再为生产环境配置这些值。

先决条件

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

流程

1. 开始为代理部署配置自定义资源(CR)实例。
   a. 使用 OpenShift 命令行界面：
      i. 以具有在您要创建部署的项目中部署 CR 的用户身份登录 OpenShift。

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

ii. 打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 Operator 安装的 deploy/crs 目录中。

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有在您要创建部署的项目中部署 CR 的用户身份登录控制台。

ii. 根据主代理 CRD 启动新的 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。

对于基本代理部署，配置可能类似如下。

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

观察 broker_activemqartemis_cr.yaml 示例 CR 文件中，image 属性被设置为 占位符 的默认值。此值表示，默认情况下 image 属性没有指定用于部署的代理容器镜像。要了解

Operator 如何确定要使用的适当代理容器镜像，请参阅 第 2.6 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 limits 和 requests 子项。在每个子部分中，添加一个 cpu 和 memory 属性并指定值。例如：

```
spec:
  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 的每个代理容器不能超过此主机节点 CPU 用量。

limits.memory

在部署中运行的 Pod 的每个代理容器不能超过此主机节点内存用量。

requests.cpu

在部署中运行的 Pod 的每个代理容器会请求此数量的主机节点 CPU。这个值是 *代理容器* 运行所需的最小 CPU 量。

requests.memory

在部署中运行的 Pod 的每个代理容器会请求此数量的主机节点内存。这个值是代理容器运行所需的 *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。

4.8. 启用对 AMQ 管理控制台的访问

基于 Operator 的部署中的每个代理 pod 都会在端口 8161 中托管自己的 AMQ 管理控制台实例。您可以为代理部署在自定义资源(CR)实例中启用对控制台的访问。在启用对控制台的访问后，您可以使用控制台在网页浏览器中查看和管理代理。

流程

1. 编辑代理部署的 ActiveMQArtemis (CR)实例。

2. 在 CR 的 spec 部分中，添加一个 console 属性。在 console 部分中，添加 expose 属性，并将值设为 true。

```
spec:
  ..
  console:
    expose: true
```

当您公开控制台时，Operator 会自动为部署中的每个代理 pod 上的控制台创建一个专用服务和 Openshift 路由。

3. 如果要自定义控制台公开的路由的主机名，以匹配 Openshift 集群中的内部路由配置，您可

以执行以下操作之一或两者：

- 使用 ingressHost 属性将默认主机名替换为控制台路由的自定义主机名。

- 使用 ingressDomain 属性将自定义域附加到主机名中。自定义域也应用于由 CR 配置公开的所有其他路由，如 acceptors 的路由。

a. 要设置专用于控制台路由的自定义主机名，请添加 ingressHost 属性并指定主机字符串。例如：

```
spec:
  ..
  console:
    expose: true
    ingressHost: my-console-production.my-subdomain.com
  ..
```

注意

ingressHost 值在 Openshift 集群中必须是唯一的。如果您的代理集群有多个代理 pod，您可以通过在值中包含 $(BROKER_ORDINAL)变量来使 ingressHost 值是唯一的。Operator 会将每个代理 pod 创建的路由中的此变量替换为分配给 pod 的 StatefulSet 编号。例如，在第一个 pod 上，my-console-$(BROKER_ORDINAL)-production.my -subdomain.com 的 ingressHost 值将路由的主机名设置为 my-console-0-production.my-subdomain.com 。

您可以在自定义主机字符串中包含以下变量：

Name	描述
$(CR_NAME)	CR 中的 **metadata.name** 属性的值。
$(CR_NAMESPACE)	自定义资源的命名空间。
$(BROKER_ORDINAL)	StatefulSet 分配给代理 pod 的普通数量。
$(ITEM_NAME)	接受者的名称。
$(RES_TYPE)	资源类型。路由具有 **rte** 资源类型。入口的资源类型为 **ing**。

Name	描述
$(INGRESS_DOMAIN)	如果在 CR 中配置，**spec.ingressDomain** 属性的值。

b. 要将自定义域附加到路由中的主机名中，请添加 **spec.ingressDomain** 属性并指定自定义字符串。例如：

```
spec:
  ...
  ingressDomain: my.domain.com
```

4. 如果机构的网络策略需要您使用 **ingress** 而不是路由公开控制台，请完成以下步骤：

a. 添加 **exposeMode** 属性，并将值设为 **ingress**。

```
spec:
  ..
  console:
    expose: true
    exposeMode: ingress
  ..
```

b. 如果要自定义控制台公开的入口主机名，以匹配 Openshift 集群中的内部路由配置，您可以执行以下操作之一或两者：

- 使用 **ingressHost** 属性将默认主机名替换为自定义主机名。
- 使用 **ingressDomain** 属性将自定义域附加到主机名中。自定义域也应用于由 CR 配置公开的所有其他入口，如 **acceptors** 的 **ingress**。

i. 要为控制台创建的入口设置自定义主机名，请添加 **ingressHost** 属性并指定主机字符串。例如：

```
spec:
  ..
  console:
    expose: true
    exposeMode: ingress
    expose: true
```

```
    exposeMode: ingress
    ingressHost: my-console-production.my-subdomain.com
  ...
```

您可以包括相同的变量来自定义入口主机作为路由主机，如此流程前面所述。

ii. 要将自定义域附加到 ingresses 中的主机名，请添加 spec.ingressDomain 属性并指定自定义字符串。

```
spec:
  ...
  ingressDomain: my.domain.com
```

对于控制台，ingress 的默认主机名采用 < cr-name>-wconsj-<ordinal>-svc-ing-<namespace> 格式。例如，如果您在 amqbroker 命名空间中有一个名为 production 的 CR，则 ingressDomain 值 mydomain.com 会为 Pod 0 上创建的 ingress 提供主机值 production-wconsj-0-svc-ing-mynamespace.amqbroker.com。

有关 spec.ingressDomain 属性的更多信息，请参阅 第 8.1 节 “自定义资源配置参考”。

5. 如果要从 OpenShift 集群以外的客户端启用到控制台的安全连接，请完成以下步骤：

a. 添加 sslEnabled 属性，并将值设为 true。

```
spec:
  ..
  console:
    expose: true
    exposeMode: ingress
    sslEnabled: true
  ..
```

b. 添加 sslSecret 属性，并指定包含证书的 secret 名称，以保护控制台。例如：

```
spec:
  ..
  console:
    expose: true
    exposeMode: ingress
```

```
    sslEnabled: true
    sslSecret: console-tls-secret
  ..
```

c. 使用 spec.env 属性添加一个环境变量，该变量将控制台配置为在每次证书续订时自动加载新证书。例如：

```
spec:
  ..
  env:
  - name: JAVA_ARGS_APPEND
    value: -Dwebconfig.bindings.artemis.sslAutoReload=true
  ..
```

6. 保存 CR。

其他资源

有关如何连接到 AMQ 管理控制台的详情，请参考 *第 5 章 连接到基于 Operator 的代理部署的 AMQ 管理控制台*

4.9. 为代理容器设置环境变量

在代理部署的自定义资源(CR)实例中，您可以设置传递给 AMQ Broker 容器的环境变量。

例如，您可以使用 TZ 等标准环境变量设置时区或 JDK_JAVA_OPTIONS，将参数添加到 Java 启动程序启动时使用的命令行参数。或者，您可以使用 AMQ Broker 的自定义变量 JAVA_ARGS_APPEND 将自定义参数附加到 Java 启动程序使用的命令行参数中。

流程

1. 编辑代理部署的自定义资源(CR)实例。

a. 使用 OpenShift 命令行界面：

i. 输入以下命令：

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

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，以便在代理部署的项目中部署 CR。

ii. 在左侧窗格中，点 Operators → Installed Operator。

iii. 点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator。

iv. 点 AMQ Broker 选项卡。

v. 单击 ActiveMQArtemis 实例名称的名称。

vi. 点 YAML 标签。

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，添加一个 env 元素，并添加您要为 AMQ Broker 容器设置的环境变量。例如：

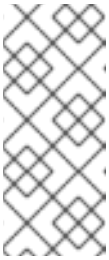
```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  ...
  env:
  - name: TZ
    value: Europe/Vienna
  - name: JAVA_ARGS_APPEND
    value: --Hawtio.realm=console
  - name: JDK_JAVA_OPTIONS
    value: -XshowSettings:system
  ...
```

在示例中，CR 配置包括以下环境变量：

- TZ 用来设置 AMQ Broker 容器的时区。

- JAVA_ARGS_APPEND 将 AMQ 管理控制台配置为使用名为 console 的域进行身份验证。

- JDK_JAVA_OPTIONS 设置 Java -XshowSettings:system 参数，该参数显示 Java 虚拟机的系统属性设置。



注意

使用 JDK_JAVA_OPTIONS 环境变量配置的值会添加到 Java 启动程序使用的命令行参数前面。使用 JAVA_ARGS_APPEND 环境变量配置的值会附加到启动程序使用的参数中。如果一个参数重复，则最右参数会优先使用。

3. 保存 CR。

注意

红帽建议不要更改具有 AMQ_ 前缀的 AMQ Broker 环境变量，如果想要更改 POD_NAMESPACE 变量，请小心谨慎。

其他资源

- 有关定义环境变量的更多信息，请参阅为容器定义环境变量。

4.10. 覆盖代理的默认内存限值

您可以覆盖为代理设置的默认内存限值。默认情况下，代理被分配为代理 Java 虚拟机可用的最大内存的一半。以下流程演示了如何为代理部署配置自定义资源(CR)实例，以覆盖默认内存限值。

先决条件

- 您应该熟悉如何使用 CR 实例创建基本代理部署。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)实例以创建基本代理部署。**

   a. **使用 OpenShift 命令行界面：**

      i. **以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。**

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

      ii. **打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 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控制台，以便在代理部署的项目中部署 CR。**

      ii. **根据主代理 CRD 启动新的 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。

      例如，基本代理部署的 CR 可能类似以下：

      ```
      apiVersion: broker.amq.io/v1beta1
      kind: ActiveMQArtemis
      metadata:
        name: ex-aao
      spec:
      ```

```
deploymentPlan:
  size: 1
  image: placeholder
  requireLogin: false
  persistenceEnabled: true
  journalType: nio
  messageMigration: true
```

2. 在 CR 的 spec 部分，添加一个 brokerProperties 部分。在 brokerProperties 部分中，添加一个 globalMaxSize 属性并指定内存限制。例如：

```
spec:
  ...
  brokerProperties:
  - globalMaxSize=500m
  ...
```

globalMaxSize 属性的默认单位是字节。要更改默认单元，请将后缀 m （用于 MB）或 g（用于 GB）添加到值。

3. 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 Save。

4. （可选）验证您为 globalMaxSize 属性设置的新值会覆盖分配给代理的默认内存限值。

a. 连接到 AMQ 管理控制台。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*。

b. 从菜单中，选择 JMX。

c. 选择 org.apache.activemq.artemis。

d. 搜索 全局。

e. 在显示的表中，确认 Global max 列中的值与您为 globalMaxSize 属性配置的值相同。

4.11. 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 容器使用主自定义资源(CR)实例进行部署。在某些情况下，您可能需要使用自定义初始容器。例如，如果您要在代理安装目录中包含额外的运行时依赖项 .jar 文件。

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（例如，Docker Dockerfile 或 Podman Containerfile）中，FROM 指令必须指定 AMQ Broker Operator 内置 Init 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-rhel8:7.12
```

- 自定义镜像必须包含一个名为 post-config.sh 的脚本，该脚本会包含在名为 /amq/scripts 的目录中。post-config.sh 脚本是您可以修改或添加到 Operator 生成的初始配置的位置。当您指定自定义 Init Container 时，Operator 在使用 CR 实例生成配置后，但在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，初始容器使用的安装目录的路径在名为 CONFIG_INSTANCE_DIR 的环境变量中定义。post-config.sh 脚本在引用安装目录时应使用此

环境变量名称（例如 ：${CONFIG_INSTANCE_DIR}/lib），而不是 此变量的实际值（例如 /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），您必须确保它们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

- 您必须构建了一个自定义 Init 容器镜像，它满足上述准则。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 基于 JDBC 的持久性自定义初始容器镜像。
- 要为 AMQ Broker Operator 提供自定义初始容器镜像，您需要将镜像上传到容器 registry 中的存储库，如 Quay 容器 registry。
- 您应该了解 Operator 如何使用初始容器生成代理配置。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。更多信息请参阅 第 3.4 节 “创建基于 Operator 的代理部署”。

流程

1. 编辑代理部署的 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 Container Platform，以便在代理部署的项目中部署 CR。
      ii. 编辑部署的 CR。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有特权的用户身份登录 OpenShift Container Platform，以便在代理部署的项目中部署 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. 点代理部署的实例。

vi. 点 YAML 标签。

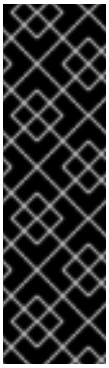
在控制台中，会打开 YAML 编辑器，供您编辑 CR 实例。

2. 在 CR 的 deploymentPlan 部分中，添加一个 initImage 属性，并将值设置为自定义 Init 容器镜像的 URL。

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  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 Init 容器镜像的完整 URL，该镜像必须可从容器 registry 获得。



重要

如果 CR 在 spec.deploymentPlan.initImage 属性中指定自定义 init 容器镜像，红帽建议您在 spec.deploymentPlan.image 属性中指定相应代理容器镜像的 URL，以防止自动升级代理镜像。如果您没有在 spec.deploymentPlan.image 属性中指定特定代理容器镜像的 URL，则会自动升级代理镜像。升级代理镜像后，代理和自定义 init 容器镜像的版本不同，这可能会阻止代理运行。

如果您有一个具有自定义 init 容器的工作部署，您可以防止进一步升级代理容器镜像，以消除较新的代理镜像无法使用自定义 init 容器镜像的风险。有关防止升级到代理镜像的更多信息，请参阅 第 6.4.2 节 “使用镜像 URL 限制镜像自动升级”。

3. 保存 CR。

其他资源

- 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 基于 JDBC 的持久性自定义初始容器镜像。

4.12. 为客户端连接配置基于 OPERATOR 的代理部署

4.12.1. 配置接收器

要在 OpenShift 部署中启用到代理 pod 的客户端连接，您可以为部署定义 *acceptors*。acceptors 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 acceptors。当您创建 acceptor 时，您可以指定要在接收器上启用的消息传递协议等信息，以及代理 pod 上用于这些协议的端口。

流程

1. 编辑代理部署的 ActiveMQArtemis 自定义资源(CR)。

2. 在 acceptors 属性中，添加 named acceptor。添加 protocols 和 port 属性。设置值，以指定接受者和每个代理 pod 上的端口用于这些协议的消息传递协议。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。下表中显示了您可以为 protocol 属性指定的完整值集。

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受器，并启用了核心协议。
- 默认情况下，AMQ Broker 管理控制台使用代理 pod 上的端口 8161。部署中的每个代理 pod 都有一个专用的服务，它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. 要在同一个接收器中使用另一个协议，请修改 protocol 属性。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. 要指定接受者允许的并发客户端连接数量，请添加 connectionsAllowed 属性并设置值。例如：

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

5. 默认情况下，acceptor 仅公开给与代理部署相同的 OpenShift 集群中的客户端。要同时将 acceptor 公开给 OpenShift 之外的客户端，请将 expose 属性和 sslEnabled 属性设置为 true。

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

当您在接受器（或连接器）上启用 SSL （安全套接字层）安全性时，您可以添加相关的配置，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 acceptor 上启用 SSL 时，需要一个 secret。
- 用于保护网络通信的传输层安全性(TLS)协议。TLS 是一个更新的、更加安全的 SSL 版本。您可以在 enabledProtocols 属性中指定 TLS 协议。
- acceptor 是否在代理和客户端之间使用 mTLS （也称为 mutual 身份验证）。您可以通过将 needClientAuth 属性的值设置为 true 来指定此值。

有关这些任务的详情，请参考 第 4.12.2 节 “保护 broker-client 连接”。

当您向 OpenShift 外部的客户端公开接受者时，Operator 会自动为部署中的每个代理 pod 上的接受器创建一个专用服务和 Openshift 路由。

6. 如果要自定义每个 pod 上为接收器公开的路由的主机名，以匹配 Openshift 集群中的内部路由配置，您可以执行以下操作之一或两者：

- 使用 ingressHost 属性将默认主机名替换为特定接收器的自定义主机名。
- 使用 ingressDomain 属性将自定义域附加到主机名中。自定义域也应用于由 CR 配置公开的所有其他路由，如其他接受器和控制台的路由。

a. 要为 acceptor 路由设置自定义主机名，请添加 ingressHost 属性并指定主机字符串。例如：

```
spec:
  ...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
    connectionsAllowed: 5
    expose: true
    ingressHost: my-acceptor-production.my-subdomain.com
  ...
```

注意

ingressHost 值在 Openshift 集群中必须是唯一的。如果您的代理集群有多个代理 pod，您可以通过在值中包含 $(BROKER_ORDINAL) 变量来使 ingressHost 值是唯一的。Operator 将每个代理 pod 上的此变量替换为分配给 pod 的 StatefulSet 编号。例如，在第一个 pod 上，my-acceptor-$(BROKER_ORDINAL)-production.my-subdomain.com 的 ingressHost 值将路由的主机名设置为 my-acceptor-0-production.my-subdomain 。

您可以在自定义主机字符串中包含以下变量：

Name	描述
$(CR_NAME)	CR 中的 **metadata.name** 属性的值。
$(CR_NAMESPACE)	自定义资源的命名空间。
$(BROKER_ORDINAL)	StatefulSet 分配给代理 pod 的普通数量。
$(ITEM_NAME)	接受者的名称。
$(RES_TYPE)	资源类型。路由具有 **rte** 资源类型。入口的资源类型为 **ing**。
$(INGRESS_DOMAIN)	如果在 CR 中配置，**spec.ingressDomain** 属性的值。

b. **要将自定义域附加到路由中的主机名中，请添加 spec.ingressDomain 属性并指定自定义字符串。例如：**

```
spec:
  ...
  ingressDomain: my.domain.com
```

7. **如果机构的网络策略要求您使用 ingress 而不是路由公开接收器，请完成以下步骤：**

a. **添加 exposeMode 属性，并将值设为 ingress。**

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

b. **如果要自定义为接受者公开的入口主机名，以匹配 Openshift 集群中的内部路由配置，您可以执行以下操作之一或两者：**

- **使用 ingressHost 属性将默认主机名替换为自定义主机名。**

- 使用 ingressDomain 属性将自定义域附加到主机名中。自定义域也应用于 CR 配置公开的所有其他入口（如其他接收器和控制台的 ingress）。

  i. 要为 acceptor 的 ingresses 设置自定义主机名，请添加 ingressHost 属性并指定主机字符串。例如：

```
spec:
  ...
  acceptors:
  - name: my-acceptor
    protocols: amqp,openwire
    port: 5672
    connectionsAllowed: 5
    expose: true
    exposeMode: ingress
    ingressHost: my-acceptor-production.my-subdomain.com
  ...
```

  您可以包括相同的变量来自定义入口主机作为路由主机，如此流程前面所述。

  ii. 要将自定义域附加到 ingresses 中的主机名，请添加 spec.ingressDomain 属性并指定自定义字符串。例如：

```
spec:
  ...
  ingressDomain: my-subdomain.domain.com
```

  对于 acceptors，ingress 的默认主机名采用 < *cr-name*>-<*acceptor name*>-<*ordinal*>-svc-ing-<*namespace*> 的格式。例如，如果您在 amqbroker 命名空间中有一个名为 production 的 CR，则 ingressDomain 值 mydomain.com 会为 Pod 0 上创建的 ingress 提供主机值 production-wconsj-0-svc-ing-mynamespace.amqbroker.com。

其他资源

- 要了解如何配置 TLS 来保护 broker-client 连接，包括生成用于存储身份验证凭证的 secret，请参阅 第 4.12.2 节 “保护 broker-client 连接”。

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

4.12.2. 保护 broker-client 连接

如果您在接收器或连接器上启用了安全性（即，将 sslEnabled 设置为 true），您必须配置传输层安全(TLS)以允许代理和客户端之间的基于证书的身份验证。TLS 是一个更新的、更加安全的 SSL 版本。主要 TLS 配置有两个：

TLS

只有代理会显示证书。客户端使用证书来验证代理。这是最常见的配置。

mTLS

代理和客户端都提供证书。这有时被称为 *mutual 身份验证*。

您可以使用各种方法生成 TLS 证书。

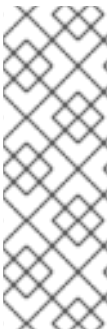
如果代理和客户端在同一 Openshift 集群上运行，您可以使用 Openshift 为代理生成服务证书。

如果代理和客户端没有在同一 Openshift 集群上运行，则必须使用允许您自定义证书的方法生成证书。本节论述了可以用来生成自定义证书的两种方法：

- 用于 Openshift 的 cert-manager Operator
- Java Keytool 工具.

4.12.2.1. 使用 Openshift 服务服务证书

如果要保护同一 Openshift 集群中的代理和客户端之间的内部连接，您可以在 acceptor 服务中添加注解来请求 Openshift 生成服务证书。生成的证书和密钥采用 PEM 格式，分别存储在所创建 secret 的 tls.crt 和 tls.key 中。



注意

用于发布服务证书的服务 CA 证书在 26 个月内有效，并在有效期少于 13 个月时进行自动轮转。轮转后，以前的服务 CA 配置仍会被信任直到其过期为止。这将为所有受影响的服务建立一个宽限期，以在过期前刷新其密钥内容。如果没有在这个宽限期内对集群进行升级（升级会重启服务并刷新其密钥），您可能需要手动重启服务以避免在上一个服务 CA 过期后出现故障。

流程

1. 编辑用于代理部署的 ActiveMQArtemis 自定义资源(CR)。

2. 使用 resourceTemplates 属性来注解为接收器创建的服务。例如：

```
spec:
 ...
 resourceTemplates:
  - selector:
      kind: Service
      name: amq-broker-myacceptor-0-svc
    annotations:
      service.beta.openshift.io/serving-cert-secret-name: myacceptor-ptls
 ...
```

resourceTemplates.selector.kind

指定自定义应用到的资源类型是 Service。

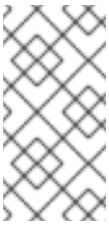
resourceTemplates.selector.name

指定要应用注解的服务名称。acceptor 服务的名称格式为：*< CR name><acceptor name><ordinal>*-svc，其中：

- *<CR name>* name 是 CR 中的 metadata.name 属性的值。
- *<acceptor name>* 是接受者的名称。示例假定接受者的名称为 myacceptor。
- *<ordinal>* 是 StatefulSet 分配给代理 pod 的普通号。

resourceTemplates.annotations

指定 service.beta.openshift.io/serving-cert-secret-name: *<secret>* 的注解，其中 *< secret >* 是 Openshift 为服务创建的 secret 的名称。



注意

secret 名称必须与 acceptor 名称匹配，并且具有 -ptls 后缀。需要特定的后缀，以允许 Operator 在创建 secret 前部署 CR。

3. 在 CR 中的 sslSecret' 属性中，指定包含代理证书的 secret。例如：

```
spec:
 acceptors:
  - name: myacceptor
    protocols: CORE
    port: 61626
    sslEnabled: true
    sslSecret: myacceptor-ptls
```

4. 在 brokerProperties 属性中，将代理配置为在每次在 Openshift 中续订证书时自动加载新证书。例如：

```
spec:
 ...
 brokerProperties
 - "acceptorConfigurations.myacceptor.params.sslAutoReload=true"
  ...
```

5. 将服务用证书的公钥添加到每个客户端的信任存储中。

6. 如果要在代理和客户端之间配置 mTLS 身份验证，请完成以下步骤。

   a. 创建一个信任捆绑包，其中包含您希望代理信任的每个客户端证书，并将信任捆绑包添加到 secret 中，例如trusted-clients-bundle。

   b. 在代理 CR 中配置的 acceptors 中，添加 needClientAuth 属性，并设置为 true 以要求客户端身份验证。例如：

```
spec:
  ..
 acceptors:
  - name: myacceptor
    protocols: all
    port: 62666
    sslEnabled: true
    sslSecret: myacceptor-ptls
    needClientAuth: true
  ..
```

   c. 在每个 acceptor 的 trustSecret 属性中，指定包含客户端证书信任捆绑包的 secret。例如：

```
spec:
 ..
 acceptors:
  - name: new-acceptor
    protocols: all
    port: 62666
    sslEnabled: true
    sslSecret: myacceptor-ptls
    needClientAuth: true
    trustSecret: trusted-clients-bundle
 ..
```

7. 保存 CR。

其他资源

使用服务提供的证书 secret 保护服务流量。

4.12.2.2. 为 Openshift 使用 cert-manager Operator

OpenShift 的 cert-manager Operator 是一个集群范围的服务，提供应用程序证书生命周期管理。cert-manager 会自动管理，并从各种证书颁发机构隔离 TLS 证书。

以下示例演示了如何使用自签名证书配置传输层安全(TLS)。如果您的策略需要由可识别的证书管理器签名的证书，您可以使用 OpenShift 的 cert-manager Operator 请求证书。

先决条件

- 已安装 cert-manager Operator for Red Hat OpenShift。

  如需更多信息，请参阅 OpenShift Container Platform 文档中的 为 Red Hat OpenShift 安装 cert-manager Operator。

- 如果要在代理和客户端之间配置 mTLS，则安装 Kubernetes 的信任管理器。

  如需更多信息，请参阅安装 trust-manager。

流程

1. 创建一个 YAML 文件，如 self-signed-issuer.yaml，该文件定义 root 自签名签发者。签发者(issuer)是一个 Openshift 资源，它代表证书颁发机构(CA)，它可以通过遵循证书签名请求来生成签名证书。

   以下示例 yaml 创建一个自签名签发者，然后使用该签发者创建证书颁发机构(CA)证书。您的 CA 证书可由 cert-manager Operator 管理。

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: root-issuer
spec:
  selfSigned: {}
```

2. 创建一个 YAML 文件，如 root-ca.yaml，用于定义 root CA 证书。

   在 issuerRef.name 字段中，指定您创建的自签名签发者(root-issuer)的名称。例如：

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: root-ca
  namespace: cert-manager
spec:
  isCA: true
  commonName: "amq.io.root"
  secretName: root-ca-secret
  subject:
    organizations:
    - "www.amq.io"
  issuerRef:
    name: root-issuer
    kind: ClusterIssuer
```

   证书以 Privacy Enhanced Mail (PEM)格式创建，格式为 root-ca-secret。

3. 创建一个 YAML 文件，如 root-ca-issuer.yaml，该文件定义了发布由 root CA 签名的证书的 CA 签发者。例如：

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: root-ca-issuer
```

```
spec:
  ca:
    secretName: root-ca-secret
```

4. 创建定义代理证书的 YAML 文件，如 broker-cert.yaml。

在 issuerRef.Name 字段中，指定您创建用来发布由 root CA 签名的证书的签发者名称 root-ca-issuer。例如：

```
apiVersion: cert-manager.io/v1
kind: Certificate
Metadata:
 name: broker-cert
spec:
 isCA: false
 commonName: “amq.io”
 dnsNames:
   - “amq-broker-ss-0.amq-broker-svc-rte-default.cluster.local
   - “amq-broker-ss-1.amq-broker-svc-rte-default.cluster.local
 secretName: broker-cert-secret
 subject:
   organizations:
   - “www.amq.io”
 issuerRef:
   name: root-ca-issuer
   kind: ClusterIssuer
```

5. 部署您在 YAML 文件中为签发者和证书定义的自定义资源，以创建对应的 OpenShift 对象。例如：

```
$ oc create -f  self-signed-issuer.yaml
$ oc create -f  root-ca.yaml
$ oc create -f  root-ca-issuer.yaml
$ oc create -f  broker-cert.yaml
```

6. 编辑用于代理部署的 ActiveMQArtemis CR。

7. 指定在您要保护的每个接受者的 sslSecret 属性中包含代理证书的 secret。例如：

```
spec:
  ..
  acceptors:
    - name: new-acceptor
      protocols: all
      port: 62666
      sslEnabled: true
```

```
    needClientAuth: false
    sslSecret: broker-cert-secret
  ..
```

8. 在 brokerProperties 属性中，将代理配置为在每次为 Openshift 的 cert-manager Operator 更新证书时自动加载接受者的新代理证书。例如：

```
spec:
  ...
  brokerProperties
  - "acceptorConfigurations.new-acceptor.params.sslAutoReload=true"
   ...
```

9. 向每个客户端的信任存储中添加签名代理证书的 root CA 证书，该证书在本示例过程中创建一个名为 root-ca-secret secret 的 secret 中，以便客户端可以信任代理。

10. 如果要在代理和客户端之间配置 mTLS 身份验证，请完成以下步骤。

    a. 使用 Trust Manager for Kubernetes 创建一个信任捆绑包，其中包含您希望代理信任的每个客户端证书，并将信任捆绑包添加到 secret 中，如trusted-clients-bundle。有关如何创建信任捆绑包的详情，请参考 trust-manager 文档。

    b. 在代理 CR 中配置的 acceptors 中，添加 needClientAuth 属性，并设置为 true 以要求客户端身份验证。例如：

```
spec:
  ..
  acceptors:
    - name: new-acceptor
      protocols: all
      port: 62666
      sslEnabled: true
      sslSecret: broker-cert-secret
      needClientAuth: true
  ..
```

    c. 在每个 acceptor 的 trustSecret 属性中，指定包含客户端证书信任捆绑包的 secret。例如：

```
spec:
  ..
  acceptors:
    - name: new-acceptor
```

```
      protocols: all
      port: 62666
      sslEnabled: true
      sslSecret: broker-cert-secret
      needClientAuth: true
      trustSecret: trusted-clients-bundle
    ..
```

11. 保存 CR。

其他资源

- cert-manager Operator for Red Hat OpenShift 文档.

4.12.2.3. 使用 Java keytool 工具

keytool 是 Java 中包含的证书管理实用程序。

4.12.2.3.1. 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 broker-client 连接。

在单向 TLS 中，只有代理会显示证书。此证书供客户端用来验证代理。

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。更多信息请参阅 第 4.12.2.4 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，以便它可以与客户端共享。以 Base64 编码的 .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 Container Platform。例如：

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

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=~/client.ks \
--from-literal=keyStorePassword=<password> \
--from-literal=trustStorePassword=<password>
```

注意

在生成 secret 时，OpenShift 需要同时指定密钥存储和信任存储。信任存储密钥通常被命名为 client.ts。对于代理和客户端之间的单向 TLS，实际上不需要信任存储。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。上一步为 client.ts 提供"dummy"值，方法是重新使用之前生成的代理密钥存储文件。这足以生成一个 secret，其中包含单向 TLS 所需的所有凭证。

7. 将 secret 链接到安装 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
...
```

4.12.2.3.2. 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 broker-client 连接。

在双向 TLS 中，代理和客户端都提供证书。代理和客户端使用这些证书在进程中相互验证，有时被称为 *mutual 身份验证*。

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。更多信息请参阅 第 4.12.2.4 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，以便它可以与客户端共享。以 Base64 编码的 .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 编码的 .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. 以管理员身份登录 OpenShift Container Platform。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 secret 时，OpenShift 需要同时指定密钥存储和信任存储。信任存储密钥通常被命名为 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，因为这会包含客户端证书。因此，在前面的步骤中，您为 client.ts 键指定的值实际上是 代理 信任存储文件。

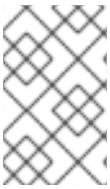
10. 将 secret 链接到安装 Operator 时创建的服务帐户。例如：

```
$ 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
...
```

4.12.2.4. 为主机名验证配置代理证书



注意

本节论述了在配置单向或双向 TLS 时必须生成的代理证书的一些要求。

当客户端尝试连接到部署中的代理 Pod 时，客户端连接 URL 中的 verifyHost 选项决定客户端是否将代理证书的通用名称(CN)与主机名进行比较，以验证它们是否匹配。如果您指定了 verifyHost=true，或者在客户端连接 URL 中类似，客户端会执行此验证。

在罕见的情况下，您可能会省略这个验证，当您对连接的安全性无关，例如，如果在隔离的网络中的 OpenShift 集群上部署代理时，可能会省略这个验证。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成功客户端连接所必需的。

通常，当客户端使用主机验证时，您在生成代理证书时指定的 CN 必须与客户端连接到的代理 Pod 上 Route 的完整主机名匹配。例如，如果您使用单个代理 Pod 部署，则 CN 可能类似如下：

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

为确保 CN 可以解析带有多个代理的的代理 Pod 中的任何代理，您可以指定一个星号(*)通配符字符来代替普通的代理 Pod。例如：

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

上例中显示的 CN 可以成功解析为 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，作为逗号分隔的列表。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,..."
```

4.12.3. 代理部署中的网络服务

在代理部署的 OpenShift Container Platform Web 控制台的 Networking 窗格中，有两个运行的服务：*无头服务* 和 *ping* 服务。无头服务的默认名称使用 < *custom_resource_name*> -hdls-svc 的格式，如 my-broker-deployment-hdls-svc。ping 服务的默认名称使用 < *custom_resource_name*> -ping-svc 的格式，例如 'my-broker-deployment-ping-svc。

无头服务提供对端口 61616 的访问，用于内部代理集群。

ping 服务供代理用于发现，并允许代理在 OpenShift 环境中组成集群。在内部，该服务会公开端口 8888。

4.12.4. 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 集群以外的客户端）连接到代理。

4.12.4.1. 从内部客户端连接到代理

要将内部客户端连接到代理，请在客户端连接详情中指定代理 pod 的 DNS 可解析名称。例如：

```
$ tcp://ex–aao-ss-0:<port>
```

如果内部客户端使用 Core 协议，且连接 URL 中没有设置 useTopologyForLoadBalancing=false 键，在客户端第一次连接到代理后，代理可以告知客户端集群中的所有代理的地址。然后，客户端可以在所有代理间负载均衡连接。

如果您的代理具有持久订阅队列或请求/回复队列，请注意在客户端连接负载均衡时使用这些队列的相关注意事项。更多信息请参阅 第 4.12.4.4 节 “具有持久订阅队列或回复/请求队列时负载平衡客户端连接的注意事项”。

4.12.4.2. 从外部客户端连接到代理

当您向外部客户端公开接受者（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 创建一个专用服务和路由。

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 命令测试对这个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
```

代理 pod 路由的完整主机名必须解析为托管 OpenShift 路由器的节点。OpenShift 路由器使用主机名来确定在 OpenShift 内部网络内发送流量的位置。默认情况下，OpenShift 路由器侦听非安全（即非 SSL）流量和端口 443 的端口 80，用于安全（即 SSL 加密）流量。对于 HTTP 连接，如果指定了安全连接 URL （即 https），路由器会自动将流量定向到端口 443 （即 http）。

如果您希望外部客户端在集群中的代理间负载均衡连接：

- 通过在每个代理 pod 的 OpenShift 路由上配置 haproxy.router.openshift.io/balance roundrobin 选项来启用负载均衡。
- 如果外部客户端使用 Core 协议，请在客户端连接 URL 中设置 useTopologyForLoadBalancing=false 键。

  设置 useTopologyForLoadBalancing=false 键可防止客户端使用代理提供的集群拓扑信息中的 AMQ Broker Pod DNS 名称。Pod DNS 名称解析为内部 IP 地址，外部客户端无法访问。

如果您的代理具有持久订阅队列或请求/回复队列，请注意在负载平衡客户端连接时使用这些队列的相关注意事项。更多信息请参阅 第 4.12.4.4 节 “具有持久订阅队列或回复/请求队列时负载平衡客户端连接的注意事项”。

如果您不希望外部客户端在集群中的不同代理间进行负载均衡连接：

- 在每个客户端的连接 URL 中，为每个代理 pod 指定路由的完整主机名。客户端尝试在连接 URL 中连接到第一个主机名。但是，如果第一个主机名不可用，客户端会在连接 URL 中自动连接到下一个主机名，以此类推。
- 如果外部客户端使用 Core 协议，在客户端的连接 URL 中设置 useTopologyForLoadBalancing=false 键，以防止客户端使用代理提供的集群拓扑信息。

对于非 HTTP 连接：

- 客户端必须明确指定端口号（例如，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。

以下是一些用于支持的消息传递协议的客户端连接 URL 示例。

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，useTopologyForLoadBalancing 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，它会生成 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>
```

外部 AMQP 客户端，使用双向 TLS

```
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>
```

4.12.4.3. 使用 NodePort 连接到 Broker

作为使用路由的替代选择，OpenShift 管理员可以配置 NodePort，以从 OpenShift 外部的客户端连接到代理 pod。NodePort 应该映射到为代理配置的接收器指定的特定协议端口之一。

默认情况下，NodePort 在 30000 到 32767 范围内，这意味着 NodePort 通常与代理 Pod 上预期的端口不匹配。

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

4.12.4.4. 具有持久订阅队列或回复/请求队列时负载平衡客户端连接的注意事项

持久化订阅

持久订阅表示为代理上的队列，在持久订阅者首次连接到代理时创建。此队列存在并接收信息，直到客户端取消订阅为止。如果客户端重新连接到不同的代理，则在那个代理上创建另一个持久订阅队列。这可能导致以下问题。

问题	缓解方案
消息可以在原始订阅队列中进行控制。	确保启用了消息 redistribution。如需更多信息，请参阅启用消息重新发布。
在仍路由其他消息时，可能会以错误的顺序接收消息，因为消息重新发布期间有一个窗口。	无。

问题	缓解方案
当客户端取消订阅时，它只会删除它最后一次连接的代理上的队列。这意味着其他队列仍然可以存在并接收消息。	要删除订阅的客户端可能存在的其他空队列，请配置以下两个属性： 将 **auto-delete-queues-message-count** 属性设置为 0，因此只有队列中没有消息时才能删除队列。**设置 auto-delete-queues-delay 属性，以删除在没有用于指定毫秒数的队列后没有消息的队列。** **如需更多信息，请参阅配置自动创建和删除地址和队列。**

请求/重新队列

当 JMS Producer 创建临时回复队列时，会在代理上创建队列。如果客户端从工作队列使用并回复临时队列连接到不同的代理，则可能会出现以下问题。

问题	缓解方案
由于客户端连接到的代理上不存在回复队列，因此客户端可能会生成错误。	确保 **auto-create-queues** 属性设为 **true**。如需更多信息，请参阅配置自动创建和删除地址和队列。
发送到工作队列的消息可能无法分发。	通过将 **message-load-balancing** 属性设置为 **ON-DEMAND**，确保消息按需进行负载均衡。另外，请确保启用了消息 redistribution。如需更多信息，请参阅启用消息重新发布。

其他资源

- **有关使用 Routes 和 NodePort 等方法与集群中运行的服务从 OpenShift 集群外部进行通信的更多信息，请参阅：**

- OpenShift Container Platform 文档中的配置集群入口流量概述。

4.13. 保护集群连接

集群中的代理之间的内部连接使用内部连接器和接收器，它们都名为 artemis。您可以使用传输层安全(TLS)协议启用 SSL 来保护集群中代理之间的连接。

在启用了 SSL 的 acceptor 上，您可以指定一个 secret，其中包含集群中的所有代理的通用 TLS 证书。在启用了 SSL 的连接器中，您可以指定一个信任存储，其中包含 TLS 证书的公钥。每个代理的信任存储中都需要公钥，因此代理可以在建立 TLS 连接时信任集群中的其他代理。

以下示例演示了如何使用自签名证书保护集群中代理之间的内部连接。

流程

1. 生成自签名 TLS 证书，并将其添加到密钥存储文件中。

- 在证书的 Subject Alternative Name (SAN)字段中，指定一个通配符 DNS 名称以匹配集群中的所有代理，如下例所示。这个示例基于使用在 test 命名空间中部署的名为 ex-aao 的 CR。

```
$ keytool -storetype jks -keystore server-keystore.jks -storepass artemis -keypass artemis -alias server -genkey -keyalg "RSA" -keysize 2048 -dname "CN=AMQ Server, OU=Artemis, O=ActiveMQ, L=AMQ, S=AMQ, C=AMQ" -validity 365 -ext bc=ca:false -ext eku=sA -ext san=dns:*.ex-aao-hdls-svc.test.svc.cluster.local
```

- 如果证书不支持使用通配符 DNS 名称，您可以在集群中所有代理 pod 的 SAN 字段中包含以逗号分隔的 DNS 名称列表。例如：

```
keytool -storetype jks -keystore server-keystore.jks -storepass artemis -keypass artemis -alias server -genkey -keyalg "RSA" -keysize 2048 -dname "CN=AMQ Server, OU=Artemis, O=ActiveMQ, L=AMQ, S=AMQ, C=AMQ" -validity 365 -ext bc=ca:false -ext eku=sA -ext san=dns:ex-aao-ss-0.ex-aao-hdls-svc.test.svc.cluster.local,dns:ex-aao-ss-1.ex-aao-hdls-svc.test.svc.cluster.local
```

- 如果 TLS 证书不支持使用 DNS 名称，则必须在 ActiveMQArtemis CR 中禁用主机验证，如下所述。

2. 从密钥存储文件导出 TLS 证书的公钥，以便将其导入到信任存储文件中。例如：

```
$ keytool -storetype jks -keystore server-keystore.jks -storepass artemis -alias server -exportcert -rfc > server.crt
```

3. 将 TLS 证书的公钥导入到信任存储文件中，以便集群中的其他代理可以信任证书。例如：

```
$ keytool -storetype jks -keystore server-truststore.jks -storepass artemis -keypass artemis -importcert -alias server -file server.crt -noprompt
```

4. 创建一个 secret，以存储密钥存储和信任存储文件及其关联的密码。例如：

```
oc create secret generic artemis-ssl-secret --namespace test --from-file=broker.ks=server-keystore.jks --from-file=client.ts=server-truststore.jks --from-literal=keyStorePassword=artemis --from-literal=trustStorePassword=artemis
```

5. 编辑代理部署的 ActiveMQArtemis CR，并添加名为 artemis 的内部接受者。在 artemis acceptor 中，将 sslEnabled 属性设置为 true，并指定您在 sslSecret 属性中创建的 secret 名称。例如：

```
spec:
  ..
  deploymentPlan:
    size: 2
  acceptors:
  - name: artemis
    port: 61616
    sslEnabled: true
    sslSecret: artemis-ssl-secret
  ..
```

6. 为 artemis 连接器启用 SSL，供集群中的每个代理用来连接到集群中的其他代理。使用 brokerProperties 属性启用 SSL，并指定包含 TLS 证书公钥的信任存储文件的路径和凭证。

```
spec:
  ..
  deploymentPlan:
    size: 2
  acceptors:
  - name: artemis
    port: 61616
    sslEnabled: true
    sslSecret: artemis-ssl-secret
  brokerProperties:
  - 'connectorConfigurations.artemis.params.sslEnabled=true'
```

```
 - 'connectorConfigurations.artemis.params.trustStorePath=/etc/artemis-ssl-secret-
volume/client.ts'
 - 'connectorConfigurations.artemis.params.trustStorePassword=artemis'
 ..
```

connectorConfigurations.artemis.params.trustStorePath

这个值必须与代理 pod 上的 truststore 文件 client.ts 匹配。secret 中的 truststore 文件以及附带的密码文件挂载到每个代理 pod 上的 /etc/<*secret name*>-volume 目录中。前面的示例指定信任存储的位置，该存储位于名为 artemis-ssl-secret 的 secret 中。

7. 如果 TLS 证书不支持使用 DNS 名称，请使用 brokerProperties 属性来禁用主机验证。例如：

```
spec:
 ..
 brokerProperties:
 ..
 - 'connectorConfigurations.artemis.params.verifyHost=false'
 ..
```

8. 保存 CR。

4.14. 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当消息大于指定最小值时将消息存储为文件。以这种方式处理大型消息意味着代理不会在内存中保存消息。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，大型消息目录为 /opt/<*custom_resource_name*> /data/large-messages，代理用于消息存储的持久性卷(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。

注意

您只能在 AMQP 协议的代理配置中配置大型消息大小限制。对于 AMQ Core 和 Openwire 协议，您可以在客户端连接配置中配置大量消息大小限制。如需更多信息，请参阅 Red Hat AMQ 客户端文档。

4.14.1. 为大型消息处理配置 AMQP 接受器

以下流程演示了如何配置接受者来处理大于指定大小的 AMQP 消息作为大消息。

先决条件

- 您应该熟悉如何为基于 Operator 的代理部署配置 acceptors。请参阅 第 4.12.1 节 “配置接收器”。
- 要将大型 AMQP 消息存储在专用的大型消息目录中，您的代理部署必须使用持久性存储（即 persistenceEnabled 在用于创建部署的自定义资源(CR)实例中设为 true ）。有关配置持久性存储的更多信息，请参阅：
  - 第 2.8 节 “Operator 部署备注”
  - 第 8.1 节 “自定义资源配置参考”

流程

1. 打开您之前在其中定义 AMQP 接受器的自定义资源(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 Administration → Custom Resource Definitions
      ii. 单击 ActiveMQArtemis CRD。
      iii. 点 Instances 选项卡。
      iv. 找到与项目命名空间对应的 CR 实例。

之前配置的 AMQP 接受者可能类似如下：

```
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
    ...
...
```

在上例中，代理配置为接受端口 5672 上的 AMQP 消息。根据 amqpMinLargeMessageSize 的值，如果接受者收到一个大于或等于 204800 字节的正文的 AMQP 消息（即 200 KB），代理将消息存储为大消息。

代理将消息存储在大型消息目录中(/opt/ &*lt;custom_resource_name*> /data/large-messages，默认为代理用于消息存储的持久性卷(PV))。

如果您没有为 amqpMinLargeMessageSize 属性显式指定值，代理会使用默认值 102400（即 100 KB）。

如果将 amqpMinLargeMessageSize 设置为 -1，则 AMQP 消息的大量消息处理被禁用。

4.15. 配置代理健康检查

您可以使用启动、存活度和就绪度探测在 AMQ Broker 上配置健康检查。

- 启动探测指示容器内的应用程序是否启动。
- 存活度探测决定容器是否仍在运行。
- 就绪度探测(Readiness probe)决定容器是否准备好接受服务请求

如果启动探测或存活度探测检查失败，探测会重启 Pod。

AMQ Broker 包括默认的就绪度和存活度探测。默认存活度探测通过 ping 代理的 HTTP 端口来检查代理是否在运行。默认就绪度探测通过打开到为代理配置的每个接受端口的连接来检查代理是否可以接受网络流量。

使用默认存活度和就绪度探测的限制是它们无法识别底层问题，例如，代理的文件系统出现问题。您可以创建自定义存活度和就绪度探测，以使用代理的命令行工具 artemis 运行更全面的健康检查。

AMQ Broker 不包括默认的启动探测。您可以在 ActiveMQArtemis 自定义资源(CR)中配置启动探测。

4.15.1. 配置启动探测

您可以配置启动探测来检查代理容器中的 AMQ Broker 应用程序是否已启动。

流程

1. 编辑代理部署的 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 Container Platform，以便在代理部署的项目中部署 CR。

ii. 编辑部署的 CR。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有特权的用户身份登录 OpenShift Container Platform，以便在代理部署的项目中部署 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. 点代理部署的实例。

vi. 点 YAML 标签。

在控制台中，会打开 YAML 编辑器，供您编辑 CR 实例。

2. 在 CR 的 deploymentPlan 部分中，添加一个 startupProbe 部分。例如：

```
spec:
 deploymentPlan:
  startupProbe:
   exec:
    command:
     - /bin/bash
     - '-c'
     - /opt/amq/bin/artemis
     - 'check'
     - 'node'
     - '--up'
     - '--url'
     - 'tcp://$HOSTNAME:61616'
   initialDelaySeconds: 5
```

```
periodSeconds: 10
timeoutSeconds: 3
failureThreshold: 30
```

命令

在容器内运行的启动探测命令。在示例中，启动探测使用 artemis check node 命令来验证是否在容器中为代理 Pod 启动 AMQ Broker。

initialDelaySeconds

探测在容器启动后运行前的延迟（以秒为单位）。默认值为 0。

periodSeconds

探测运行的时间间隔（以秒为单位）。默认值为 10。

timeoutSeconds

启动探测命令等待代理回复的时间（以秒为单位）。如果没有收到对命令的响应，命令将被终止。默认值为 1。

failureThreshold

连续失败（包括探测被认为已失败）的启动探测的最小失败。当探测被视为失败时，它会重启 Pod。默认值为 3。

根据集群的资源和代理日志的大小，您可能需要增加故障阈值，以允许代理有足够的时间启动并传递探测检查。否则，代理会输入一个循环条件，其中会重复达到失败阈值，代理每次由启动探测重启。例如，如果您将 failureThreshold 设置为 30，且探测的默认间隔为 10 秒，则代理有 300 秒才能启动并传递探测检查。

3. 保存 CR。

其他资源

如需有关 OpenShift Container Platform 中存活度和就绪度探测的更多信息，请参阅 OpenShift Container Platform 文档中的使用 健康检查来监控应用程序健康状况。

4.15.2. 配置存活度和就绪度探测

以下示例演示了如何为代理部署配置主自定义资源(CR)实例，以使用存活度和就绪度探测运行健康检查。

先决条件

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

流程

1. 编辑代理部署的 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 Container Platform，以便在代理部署的项目中部署 CR。

      ii. 编辑部署的 CR。

      ```
      oc edit ActiveMQArtemis <CR instance name> -n <namespace>
      ```

   b. 使用 OpenShift Container Platform Web 控制台：

      i. 以具有特权的用户身份登录 OpenShift Container Platform，以便在代理部署的项目中部署 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. 点代理部署的实例。

      vi. 点 **YAML** 标签。

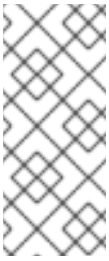
2.

要配置存活度探测，请在 CR 的 deploymentPlan 部分中添加一个 livenessProbe 部分。例如：

```
spec:
  deploymentPlan:
    livenessProbe:
      initialDelaySeconds: 5
      periodSeconds: 5
      failureThreshold: 30
```

initialDelaySeconds

探测在容器启动后运行前的延迟（以秒为单位）。默认值为 5。



注意

如果部署也配置了启动探测，则可以将存活度和就绪度探测的延迟设置为 0。这两个探测仅在启动探测通过后运行。如果启动探测已经通过，它会确认代理已成功启动，因此不需要延迟运行存活度和就绪度探测。

periodSeconds

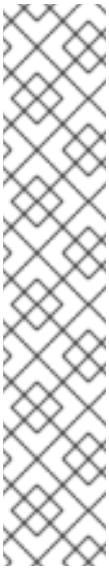
探测运行的时间间隔（以秒为单位）。默认值为 5。

failureThreshold

连续最小失败，包括表示探测失败的存活度探测的超时。当探测失败时，它会重启 Pod。默认值为 3。

如果您的部署没有配置启动探测，它会验证代理应用程序是否在存活度探测运行前启动，您可能需要增加故障阈值，以允许代理有足够的时间启动并传递存活度探测检查。否则，代理可能会输入一个循环条件，其中不再达到失败阈值，代理 Pod 每次由存活度探测重启。

代理启动和传递存活度探测检查所需的时间取决于集群的资源和代理日志的大小。例如，如果您将 failureThreshold 设置为 30，且探测在默认间隔 5 秒运行，代理需要 150 秒才能启动并传递存活度探测检查。



注意

如果您没有配置存活度探测，或者没有配置的探测，AMQ Broker Operator 会创建一个具有以下配置的默认 TCP 探测。默认 TCP 探测尝试为指定端口上的代理容器打开一个套接字。

```
spec:
  deploymentPlan:
    livenessProbe:
      tcpSocket:
        port: 8181
      initialDelaySeconds: 30
      timeoutSeconds: 5
```

3. 要配置就绪度探测，请在 CR 的 deploymentPlan 部分中添加一个 readinessProbe 部分。例如：

```
spec:
  deploymentPlan:
    readinessProbe:
      initialDelaySeconds: 5
      periodSeconds: 5
```

如果您没有配置就绪度探测，则内置 脚本将检查 所有接收器是否可以接受连接。

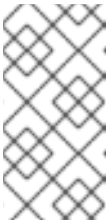
4. 如果要配置更全面的健康检查，请将 artemis check 命令行工具添加到存活度或就绪度探测配置中。

   a. 如果要配置健康检查，以创建与代理的完整客户端连接，在 livenessProbe 或 readinessProbe 部分中，添加一个 exec 部分。在 exec 部分中，添加一个 command 部分。在 command 部分中，添加 artemis check node 命令语法。例如：

```
spec:
  deploymentPlan:
    readinessProbe:
      exec:
        command:
          - bash
          - '-c'
          - /home/jboss/amq-broker/bin/artemis
          - check
          - node
          - '--silent'
          - '--acceptor'
```

```
      - <acceptor name>
      - '--user'
      - $AMQ_USER
      - '--password'
      - $AMQ_PASSWORD
    initialDelaySeconds: 30
    timeoutSeconds: 5
```

默认情况下，artemis check node 命令使用名为 artemis 的 acceptor 的 URI。如果代理有一个名为 artemis，您可以在命令中排除 --acceptor <acceptor name& gt; 选项。



注意

$AMQ_USER 和 $AMQ_PASSWORD 是 AMQ Operator 配置的环境变量。

b.

如果要配置生成和消耗消息的健康检查，它也在 livenessProbe 或 readinessProbe 部分中验证代理文件系统的健康状态，请添加 exec 部分。在 exec 部分中，添加一个 command 部分。在 command 部分中，添加 artemis check queue 命令语法。例如：

```
spec:
  deploymentPlan:
    readinessProbe:
      exec:
        command:
          - bash
          - '-c'
          - /home/jboss/amq-broker/bin/artemis
          - check
          - queue
          - '--name'
          - livenessqueue
          - '--produce'
          - "1"
          - '--consume'
          - "1"
          - '--silent'
          - '--user'
          - $AMQ_USER
          - '--password'
          - $AMQ_PASSWORD
      initialDelaySeconds: 30
      timeoutSeconds: 5
```

注意

您指定的队列名称必须在代理上配置，且 anycast 为 routingType。例如：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisAddress
metadata:
  name: livenessqueue
  namespace: activemq-artemis-operator
spec:
  addressName: livenessqueue
  queueConfiguration:
    purgeOnNoConsumers: false
    maxConsumers: -1
    durable: true
    enabled: true
  queueName: livenessqueue
  routingType: anycast
```

5. 保存 CR。

其他资源

如需有关 OpenShift Container Platform 中存活度和就绪度探测的更多信息，请参阅 OpenShift Container Platform 文档中的使用 健康检查来监控应用程序健康状况。

4.16. 启用消息迁移来支持集群缩减

如果要缩减集群中的代理数量，并将信息迁移到集群中剩余的 Pod，您必须启用消息迁移。

当您缩减启用了消息迁移的集群时，缩减控制器会管理消息迁移过程。

4.16.1. 消息迁移过程中的步骤

消息迁移过程遵循以下步骤：

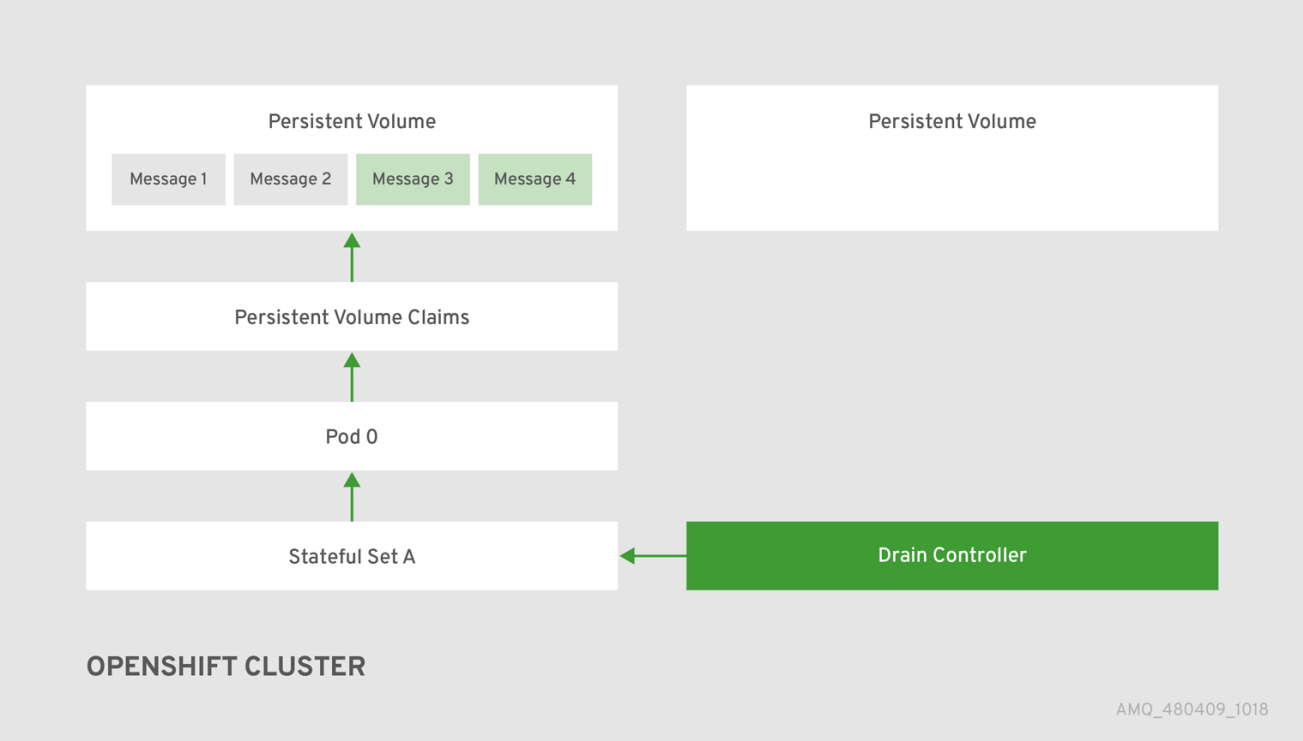
1. 当部署中的代理 Pod 因部署的意图缩减而关闭时，Operator 会自动部署 scaledown 自定义资源以准备消息迁移。

2. 要检查已孤立的持久性卷(PV)，缩减控制器将查看卷声明上的异常。控制器将卷声明上的情况与仍在 StatefulSet （即代理集群）中运行的代理 Pod 进行比较。

   如果卷声明上的情况高于代理集群中仍在运行的任何代理 Pod 上的情况，则 scaledown 控制器决定代理 Pod 已关闭，且消息传递数据必须迁移到另一个代理 Pod。

3. scaledown 控制器启动一个 drainer Pod。drainer Pod 连接到集群中的其他实时代理 Pod，并将信息迁移到该 live 代理 Pod。

下图演示了 scaledown 控制器（也称为 *drain controller*）如何将消息迁移到正在运行的代理 Pod。



当消息成功迁移到可正常工作的代理 Pod 后，排空器 Pod 会关闭，缩减控制器会删除孤立 PV 的 PVC。PV 会被返回到 "Released" 状态。

注意

如果将 PV 的 reclaim 策略设置为 保留，则 PV 无法供另一个 Pod 使用，直到您删除并重新创建 PV。例如，如果您在缩减后扩展集群，则 PV 将无法供 Pod 启动，直到删除并重新创建 PV。

其他资源

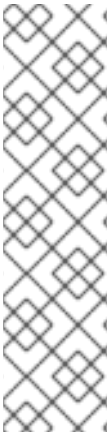
- 有关缩减代理部署时的消息迁移示例，请参阅 第 4.16.2 节 “启用消息迁移”。

4.16.2. 启用消息迁移

您可以在 ActiveMQArtemis 自定义资源(CR)中启用消息迁移。

先决条件

- 您已有一个基本的代理部署。请参阅 第 3.4.1 节 “部署基本代理实例”。
- 您了解消息迁移的工作原理。更多信息请参阅 第 4.16.1 节 “消息迁移过程中的步骤”。



注意

- 缩减控制器仅在单一 OpenShift 项目中运行。控制器无法在不同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），则不会发生消息迁移，因为没有运行代理 Pod 可以迁移到哪些消息传递数据。但是，如果您将部署缩减为零，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。

流程

1. 编辑代理部署的 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 Container Platform，以便在代理部署的项目中部署 CR。

      ii. 编辑部署的 CR。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

.

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有特权的用户身份登录 OpenShift Container Platform，以便在代理部署的项目中部署 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. 点代理部署的实例。

vi. 点 YAML 标签。

在控制台中，会打开 YAML 编辑器，供您编辑 CR 实例。

2. 在 CR 的 deploymentPlan 部分中，添加一个 messageMigration 属性，并设置为 true。如果尚未配置，请添加 persistenceEnabled 属性，并设置为 true。例如：

```
spec:
  deploymentPlan:
    messageMigration: true
    persistenceEnabled: 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 会自动启动一个 scaledown 控制器，并将信息迁移到仍然运行的代理 Pod 中。

3. 保存 CR。

4. （可选）完成以下步骤来缩减集群并查看消息迁移过程。

a. 在现有代理部署中，验证哪些 Pod 正在运行。

```
$ oc get pods
```

您会看到类似如下的输出。

```
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；一个用于代理 Operator 本身，以及部署中每个代理有一个单独的 Pod。

b. 登录到每个 Pod，并将一些信息发送到每个代理。

i. Pod ex-aao-ss-0 有集群 IP 地址 172.17.0.6，运行以下命令：

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

c. Pod ex-aao-ss-1 有集群 IP 地址 172.17.0.7，运行以下命令：

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

前面的命令在每个代理上创建一个名为 TEST 的队列，并将 1000 个消息添加到每个队列中。

d. 将集群从两个代理缩减为一个。

i. 打开主代理 CR，broker_activemqartemis_cr.yaml。

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

iii. 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 控制器启动相同名称的新 drainer Pod。这个 drainer Pod 也会在将信息从代理 Pod ex-ao-s-1 中迁移到集群（ex-aao-ss-0）中的其他代理 Pod 中后关闭。

e. 当 drainer Pod 关闭时，检查代理 Pod ex-aao-ss-0 的 TEST 队列上的消息计数。您会看到队列中的消息数量是 2000，这表示 drainer Pod 已成功从关闭的代理 Pod 中迁移 1000 个信息。

4.17. 控制 OPENSHIFT CONTAINER PLATFORM 节点上的代理 POD 放置

您可以使用节点选择器、容限或关联性和反关联性规则来控制 OpenShift Container Platform 节点上的 AMQ Broker pod 放置。

节点选择器

节点选择器允许您将代理 pod 调度到特定的节点上。

容限（Tolerations）

如果容限与为节点配置的污点匹配，则容限使代理 pod 能够调度到节点上。如果没有匹配的 pod 容限，污点允许节点拒绝接受 pod。

关联性/关联性

节点关联性规则根据节点标签控制 pod 可以调度到哪些节点。Pod 关联性和反关联性规则根据已在该节点上运行的 pod 控制 pod 可以调度到哪些节点。

4.17.1. 使用节点选择器将 pod 放置到特定节点

节点选择器指定一个键值对，它要求将代理 pod 调度到在节点标签中匹配键值对的节点上。

以下示例演示了如何配置节点选择器来在特定节点上调度代理 pod。

先决条件

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

- 为要在其上调度代理 pod 的 OpenShift Container Platform 节点添加标签。有关添加节点标签的更多信息，请参阅 OpenShift Container Platform 文档中的 使用节点选择器来控制 pod 放置。

流程

1. 根据主代理 CRD 创建自定义资源(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，以便在代理部署的项目中部署 CR。

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

      ii. 打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 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控制台，以便在代理部署的项目中部署 CR。

      ii. 根据主代理 CRD 启动新的 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。

2. 在 CR 的 deploymentPlan 部分中，添加一个 nodeSelector 部分，并添加您要匹配的节点标签，以便为 pod 选择节点。例如：

```
spec:
  deploymentPlan:
    nodeSelector:
      app: broker1
```

在本例中，代理 pod 调度到具有 app: broker1 标签的节点上。

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。

其他资源

如需有关 OpenShift Container Platform 中的节点选择器的更多信息，请参阅 OpenShift Container Platform 文档中的 使用节点选择器将 pod 放置到特定的节点上。

4.17.2. 使用容限控制 pod 放置

污点和容限控制 pod 是否可以调度到特定的节点上。通过使用污点(taint)，节点可以拒绝调度 pod，除非 pod 具有匹配的容限。您可以使用污点从节点中排除 pod，以便节点为特定 pod 保留，如代理 pod，具有匹配的容限。

具有匹配的容限允许将代理 pod 调度到节点上，但不保证 pod 调度到该节点上。为确保代理 pod 调度到配置了污点的节点上，您可以配置关联性规则。如需更多信息，请参阅 第 4.17.3 节 “使用关联性和反关联性规则控制 pod 放置”。

以下示例演示了如何配置容限以匹配节点上配置的污点。

先决条件

- 您应该熟悉如何使用 CR 实例创建基本代理部署。请参阅 第 3.4.1 节 “部署基本代理实例”。
- 将污点应用到您要为调度代理 pod 保留的节点。污点由 key、value 和 effect 组成。污点效果决定是否：
  - 节点上的现有 pod 会被驱除
  - 允许将现有 pod 保留在节点上，但无法调度新 pod，除非有匹配的容限(toleration)
  - 如果需要，可以将新 pod 调度到节点上，但最好不要在节点上调度新 pod。

如需有关应用污点的更多信息，请参阅 OpenShift Container Platform 文档中的使用节点污点控制 pod 放置。

流程

1. 根据主代理 CRD 创建自定义资源(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，以便在代理部署的项目中部署 CR。

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

ii. 打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 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控制台，以便在代理部署的项目中部署 CR。

ii. 根据主代理 CRD 启动新的 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。

2. 在 CR 的 deploymentPlan 部分中，添加一个 tolerations 部分。在 tolerations 部分中，为您要匹配的节点污点添加容限。例如：

```
spec:
  deploymentPlan:
    tolerations:
    - key: "app"
      value: "amq-broker"
      effect: "NoSchedule"
```

在本例中，容限与 app=amq-broker:NoSchedule 的节点污点匹配，因此可将 pod 调度到配置了此污点的节点。

注意

为确保代理 pod 被正确调度，请不要在 CR 的 tolerations 部分指定 tolerationsSeconds 属性。

1. 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。

其他资源

如需有关 OpenShift Container Platform 中污点和容限的更多信息，请参阅 OpenShift Container Platform 文档中的使用节点污点控制 pod 放置。

4.17.3. 使用关联性和反关联性规则控制 pod 放置

您可以使用节点关联性、pod 关联性或 pod 反关联性规则来控制 pod 放置。节点关联性允许 pod 指定与一组目标节点的关联性。通过 pod 关联性和反关联性，您可以指定有关 pod 如何能够或无法相对于节点上运行的其他 pod 的规则。

4.17.3.1. 使用节点关联性规则控制 pod 放置

节点关联性允许代理 pod 指定与可以放置它的一组节点的关联性。代理 pod 可以调度到具有与您为 pod 创建的关联性规则相同的键值对的任何节点上。

以下示例演示了如何使用节点关联性规则将代理配置为控制 pod 放置。

先决条件

- 您应该熟悉如何使用 CR 实例创建基本代理部署。请参阅 第 3.4.1 节 “部署基本代理实例”。
- 为 OpenShift Container Platform 集群中的节点分配一个通用标签，它可以调度代理 pod，如 zone: emea。

流程

1. 根据主代理 CRD 创建自定义资源(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，以便在代理部署的项目中部署 CR。

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

      ii. 打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 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控制台，以便在代理部署的项目中部署 CR。

      ii. 根据主代理 CRD 启动新的 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。

2. **在 CR 的 deploymentPlan 部分中，添加以下部分：affinity,nodeAffinity,requiredDuringSchedulingIgnoredDuringExecution, 和 nodeSelectorTerms。在 nodeSelectorTerms 部分中，添加 - matchExpressions 参数，并指定要匹配的节点标签的键值字符串。例如：**

```
spec:
  deploymentPlan:
    affinity:
      nodeAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          nodeSelectorTerms:
          - matchExpressions:
            - key: zone
              operator: In
              values:
              - emea
```

在本例中，关联性规则允许将 pod 调度到具有键为 zone 且值为 emea 的任何节点上。

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。

其他资源

如需有关 OpenShift Container Platform 中的关联性规则的更多信息，请参阅 OpenShift Container Platform 文档中的使用节点关联性规则控制节点上的 pod 放置。

4.17.3.2. 使用反关联性规则相对于其他 pod 放置 pod

通过反关联性规则，您可以根据已在该节点上运行的 pod 标签限制代理 pod 可以调度到哪些节点。

使用反关联性规则的用例是确保集群中的多个代理 pod 没有调度到同一节点上，从而造成单点故障。如果您不控制 pod 的放置，则集群中的 2 个或更多代理 pod 可以调度到同一节点上。

以下示例演示了如何配置反关联性规则，以防止将集群中的 2 个代理 pod 调度到同一节点上。

先决条件

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

流程

1. 根据主代理 CRD，为集群中的第一个代理创建一个 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，以便在代理部署的项目中部署 CR。

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

ii. 打开一个名为 **broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 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控制台，以便在代理部署的项目中部署 CR。**

ii. **根据主代理 CRD 启动新的 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。

2. **在 CR 的 deploymentPlan 部分中，添加一个 labels 部分。为第一个代理 pod 创建标识标签，以便您可以在第二个代理 pod 上创建反关联性规则，以防止两个 pod 调度到同一节点上。例如：**

```
spec:
  deploymentPlan:
    labels:
      name: broker1
```

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. 根据主代理 CRD，为集群中的第二个代理创建一个 CR 实例。

a. 在 CR 的 deploymentPlan 部分中，添加以下部分：affinity,podAntiAffinity,requiredDuringSchedulingIgnoredDuringExecution, 和 labelSelector。在 labelSelector 部分中，添加 - matchExpressions 参数，并指定代理 pod 标签的键值字符串，因此此 pod 不会调度到同一节点上。

```
spec:
  deploymentPlan:
    affinity:
      podAntiAffinity:
        requiredDuringSchedulingIgnoredDuringExecution:
          labelSelector:
            - matchExpressions:
            - key: name
              operator: In
              values:
                - broker1
          topologyKey: topology.kubernetes.io/zone
```

在本例中，pod 反关联性规则可防止 pod 与具有键为 name 和值 broker1 的 pod 相同的节点上放置，该标签是分配给集群中第一个代理的标签。

5. 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。

其他资源

如需有关 OpenShift Container Platform 中的关联性规则的更多信息，请参阅 OpenShift Container Platform 文档中的使用节点关联性规则控制节点上的 pod 放置。

4.18. 为代理配置日志记录

AMQ Broker 使用 Log4j 2 logging 工具提供消息日志记录。当您部署代理时，它会使用默认的 Log4j 2 配置。如果要更改默认配置，则必须在 secret 或 configMap 中创建一个新的 Log4j 2 配置。将 secret 或 configMap 的名称添加到主代理自定义资源(CR)后，Operator 会将每个代理配置为使用新的日志记录配置，该配置存储在 Operator 挂载到每个 Pod 的文件中。

前提条件

- 熟悉 Log4j 2 配置选项。

流程

1. 准备包含您要与 AMQ Broker 搭配使用的 log4j 2 配置的文件。

代理使用的默认 Log4j 2 配置文件位于每个代理 Pod 上的 /home/jboss/amq-broker/etc/log4j2.properties 文件中。您可以使用默认配置文件的内容作为在 secret 或 configMap 中创建新的 Log4j 2 配置的基础。要获取默认 Log4j 2 配置文件的内容，请完成以下步骤。

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

i. 单击 **Workloads → Pods**。

ii. **点 ex-aao-ss Pod。**

iii. **点击 Terminal 选项卡。**

iv. **使用 cat 命令显示代理 Pod 上 /home/jboss/amq-broker/etc/log4j2.properties 文件的内容并复制其内容。**

v. **将内容粘贴到安装 OpenShift Container Platform CLI 的本地文件中，并将文件保存为 logging.properties。**

b. **使用 OpenShift 命令行界面：**

i. **获取部署中的 Pod 的名称。**

```
$ 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. **使用 oc cp 命令将日志配置文件从 Pod 复制到您的本地目录。**

```
$ oc cp <pod name>:/home/jboss/amq-broker/etc/log4j2.properties
logging.properties -c <name>-container
```

其中，容器名称的 *<name>* 部分是 Pod 名称的 **-ss** 字符串前的前缀。例如：

```
$ oc cp ex-aao-ss-0:/home/jboss/amq-broker/etc/log4j2.properties logging.properties
-c ex-aao-container
```

注意

从文件创建 **configMap** 或 **secret** 时，**configMap** 或 **secret** 中的键默认为文件名，值默认为文件内容。从名为 **logging.properties** 的文件创建 **secret**，新日志记录配置所需的密钥会插入到 **secret** 或 **configMap** 中。

2. 编辑 **logging.properties** 文件，并创建您要与 **AMQ Broker** 搭配使用的 **Log4j 2** 配置。

例如，在使用默认配置时，**AMQ Broker** 只会将信息记录到控制台。您可能想要更新配置，以便 **AMQ Broker** 也会将信息记录到磁盘。

3. 将更新的 **Log4j 2** 配置添加到 **secret** 或 **ConfigMap** 中。

a. 以具有在项目中为代理部署创建 **secret** 或 **ConfigMap** 的用户身份登录 **OpenShift**。

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

b. 如果要在 **secret** 中配置日志设置，请使用 **oc create secret** 命令。例如：

```
oc create secret generic newlog4j-logging-config --from-file=logging.properties
```

c. 如果要在 **ConfigMap** 中配置日志设置，请使用 **oc create configmap** 命令。例如：

```
oc create configmap newlog4j-logging-config --from-file=logging.properties
```

configMap 或 **secret** 名称必须具有 **-logging-config** 的后缀，因此 **Operator** 可以识别 **secret** 包含新的日志记录配置。

4. 将 **secret** 或 **ConfigMap** 添加到代理部署的自定义资源**(CR)**实例中。

a. **使用 OpenShift 命令行界面：**

i. **以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。**

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

ii. **编辑 CR。**

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

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，以便在代理部署的项目中部署 CR。**

ii. **在左侧窗格中，点 Operators → Installed Operator。**

iii. **点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator。**

iv. **点 AMQ Broker 选项卡。**

v. **单击 ActiveMQArtemis 实例名称的名称**

vi. **点 YAML 标签。**

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

c. **将包含 Log4j 2 日志记录配置的 secret 或 configMap 添加到 CR。以下示例显示了一个 secret，并在 CR 中添加 configMap。**

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
```

```
spec:
  deploymentPlan:
    ...
    extraMounts:
      secrets:
      - "newlog4j-logging-config"
    ...
```

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    ...
    extraMounts:
      configMaps:
      - "newlog4j-logging-config"
    ...
```

5. 保存 CR。

在每个代理 Pod 中，Operator 挂载一个 logging.properties 文件，该文件包含您创建的 secret 或 configMap 中的日志记录配置。另外，Operator 将每个代理配置为使用挂载的日志文件，而不是默认的日志文件。

注意

如果在 configMap 或 secret 中更新日志配置，每个代理会自动使用更新的日志记录配置。

4.19. 配置 POD 中断预算

Pod 中断预算指定集群中在自愿中断期间必须同时可用的最少 Pod 数量，如维护窗口。

流程

1. 编辑代理部署的 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 Container Platform，以便在代理部署的

项目中部署 CR。

ii. 编辑部署的 CR。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

b. 使用 OpenShift Container Platform Web 控制台：

i. 以具有特权的用户身份登录 OpenShift Container Platform，以便在代理部署的项目中部署 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. 点代理部署的实例。

vi. 点 YAML 标签。

在控制台中，会打开 YAML 编辑器，供您编辑 CR 实例。

2. 在 CR 的 spec 部分，添加一个 podDisruptionBudget 元素，并在部署中指定在自愿中断期间必须可用的最小 Pod 数量。在以下示例中，最少有一个 Pod 可用：

```
spec:
  ...
  podDisruptionBudget:
    minAvailable: 1
  ...
```

3. 保存 CR。

其他资源

如需有关 Pod 中断预算的更多信息，请参阅 了解如何使用 pod 中断预算来指定 OpenShift Container Platform 文档中必须在线的 pod 数量。

4.20. 为管理操作配置基于角色的访问控制

基于角色的访问控制(RBAC)用于限制对 MBeans 属性和方法的访问。Mbeans 是 AMQ Broker 公开管理 API 的方式，以支持管理操作。在以前的版本中，您可以通过在 ActiveMQArtemisSecurity 自定义资源(CR)中设置 RBAC 配置并重启代理以使更改生效来限制对 MBeans 的访问。从 7.12 开始，您可以限制对 ActiveMQArtemis CR 中的 MBeans 的访问，并且不需要代理重启才能使更改生效。

流程

1. 编辑用于代理部署的 ActiveMQArtemis CR。

2. 添加以下环境变量，将代理配置为使用您在 ActiveMQArtemis CR 中指定的 RBAC 配置。

```
spec:
  ..
  env:
  - name: JAVA_ARGS_APPEND
    value: "-Dhawtio.role=-
Djavax.management.builder.initial=org.apache.activemq.artemis.core.server.managem
ent.ArtemisRbacMBeanServerBuilder"
  ..
```

3. 在 brokerProperties 属性中，为管理操作添加基于角色的访问控制配置。

管理操作的匹配地址的格式是：

mops.<*资源类型 >*.<*resource name*>.<*operation*>

例如，以下配置向 activemq.management 地址授予 manager 角色 视图 和 编辑权限。操作位置中的星号 packagemanifests 授予对所有操作的访问权限。

```
spec:
  ..
  brokerProperties:
```

```
 - securityRoles."mops.address.activemq.management.*".manager.view=true
 - securityRoles."mops.address.activemq.management.*".manager.edit=true
```

在以下示例中，mops 前缀后的数字符号(#)将 amq 角色视图授予 amq 角色 视图，并将权限编辑 到所有 MBeans。

```
spec:
 ..
 brokerProperties:
 - securityRoles."mops.#".amq.view=true
 - securityRoles."mops.#".amq.edit=true
 ..
```

4. 保存 CR。

4.21. 自定义 OPERATOR 创建的 OPENSHIFT 资源

AMQ Broker 部署创建 OpenShift 资源，如部署、Pod、有状态集和服务资源。这些资源由 AMQ Broker Operator 管理。只有负责管理特定 OpenShift 资源的操作器才能更改该资源。

如果要执行某些任务，自定义 Operator 管理的 OpenShift 资源会很有用，例如：

- 添加自定义注解，以控制资源如何被其他服务处理。
- 修改没有在代理自定义资源中公开的属性。

您可以使用 resourceTemplates 属性来自定义 AMQ Broker Operator 创建的资源。如果要向资源添加注解或标签，请将 resourceTemplates 属性配置为包含 annotations 或 labels 属性。在以下示例中，annotations 属性用于为 Operator 管理的所有服务添加注解。

```
spec:
 ..
 resourceTemplates:
  - selector:
      kind: "Service"
    annotations:
      name: "amq-operator-managed"
 ..
```

注意

selector' 属性用于过滤应用自定义的资源。如果 selector 属性为空，则自定义将应用到 Operator 管理的所有资源。

如果要自定义资源中的注解或标签以外的属性，则必须使用 patch 属性配置 resourceTemplates 属性。当您指定 patch 属性时，Operator 会使用策略合并来更新指定的资源属性。在以下示例中，patch 属性用于更改 StatefulSet 资源中的 minReadySeconds 属性的默认值。

```
spec:
  ..
  resourceTemplates:
  - selector:
      kind: "StatefulSet"
    patch:
      kind: "StatefulSet"
      spec:
       template:
        spec:
          minReadySeconds: 10
  ..
```

其他资源

有关战略合并的详情，请参考 使用战略合并补丁来更新部署。

4.22. 使用 AMQ BROKER 注册插件

您可以通过在 CR 中的 brokerProperties 属性中注册插件来扩展 AMQ Broker 的功能。

流程

1. 编辑代理部署的自定义资源(CR)。

2. 在 brokerProperties 属性中，指定插件的类名称，并包含以逗号分隔的 *<key>=<value>* 对字符串，用于定义插件的属性。

   在以下示例中，通过 AMQ Broker 提供的 LoggingActiveMQServerPlugin 插件已被注册。

   ```
   spec:
     ...
   ```

```
 brokerProperties:
 -
brokerPlugins.\"org.apache.activemq.artemis.core.server.plugin.impl.LoggingActiveM
QServerPlugin.class\".init=LOG_CONNECTION_EVENTS=true,LOG_SESSION_EVENTS
=true,LOG_CONSUMER_EVENTS=true
 ...
```

3. 保存 CR。

创建插件实例后，init 方法会传递一个字符串，其中包含 *<key>=<value>* pair，用于为插件设置属性。

注意

如果创建自定义插件，请确保插件类的 JAR 文件位于代理的 Java 类路径中。如需更多信息，请参阅 第 4.4 节 “添加第三方 JAR 文件”。

4.23. 配置不在自定义资源定义中公开的项目

自定义资源定义(CRD)是您可以配置项目的模式，您可以修改 AMQ Broker。您可以为对应自定义资源(CR)实例中的 CRD 中的配置项指定值。Operator 从 CR 实例为每个代理容器生成配置。

您可以通过在 brokerProperties 属性中添加项，在 CRD 中包含没有公开的配置项。brokerProperties 属性中包含的项目存储在 secret 中，该 secret 作为代理 Pod 的属性文件挂载。在启动时，属性文件会在应用 XML 配置后应用到内部 java 配置 bean。

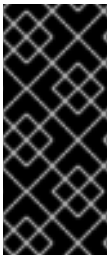
在以下示例中，单个属性应用到配置 bean。

```
spec:
  ...
  brokerProperties:
  - globalMaxSize=500m
  ...
```

在以下示例中，将多个属性应用到嵌套配置 Bean 集合，以创建一个名为 target 的代理连接，该代理将消息与另一个代理一起镜像。

```
spec:
  ...
  brokerProperties
  - "AMQPConnections.target.uri=tcp://<hostname>:<port>"
```

```
- "AMQPConnections.target.connectionElements.mirror.type=MIRROR"
- "AMQPConnections.target.connectionElements.mirror.messageAcknowledgements=true"
- "AMQPConnections.target.connectionElements.mirror.queueCreation=true"
- "AMQPConnections.target.connectionElements.mirror.queueRemoval=true"
...
```



重要

使用 brokerProperties 属性提供对 OpenShift Container Platform 上 AMQ Broker 无法配置的许多配置项目的访问。如果使用不正确，一些属性可能会对您的部署造成严重后果。使用此方法配置属性时始终谨慎。

流程

1. **编辑部署的 CR。**

   a. **使用 OpenShift Web 控制台：**

      i. **输入以下命令：**

      ```
      oc edit ActiveMQArtemis <CR instance name> -n <namespace>
      ```

   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，以便在代理部署的项目中部署 CR。**

      ii. **在左侧窗格中，点 Operators → Installed Operator。**

      iii. **点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator。**

      iv. **点 AMQ Broker 选项卡。**

      v. **单击 ActiveMQArtemis 实例名称的名称。**

vi. 点 YAML 标签。

在控制台中，会打开 YAML 编辑器，供您编辑 CR 实例。

2. 在 CR 的 spec 部分，添加一个 brokerProperties 元素，并以 camel-case 格式添加属性列表。例如：

```
spec:
 ...
 brokerProperties:
 - globalMaxSize=500m
 - maxDiskUsage=85
 ...
```

3. 保存 CR。

4. （可选）检查配置的状态。

a. 使用 OpenShift 命令行界面：

i. 获取代理状态条件。

```
$ oc get activemqartemis -o yaml
```

b. 使用 OpenShift Web 控制台：

i. 导航到代理部署的 CR 的 status 部分。

c. 检查 BrokerPropertiesApplied 状态信息中的 reason 字段的值。例如：

```
- lastTransitionTime: "2023-02-06T20:50:01Z"
  message: ""
  reason: Applied
  status: "True"
  type: BrokerPropertiesApplied
```

可能的值有：

已应用

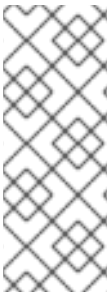
OpenShift Container Platform 将更新的 secret 传播到每个代理 Pod 的属性文件中。

AppliedWithError

OpenShift Container Platform 将更新的 secret 传播到每个代理 Pod 的属性文件中。但是，在 brokerProperties 配置中发现了一个错误。在 CR 的 status 部分中，检查 message 字段以识别无效属性并在 CR 中更正它。

OutOfSync

OpenShift Container Platform 还没有将更新的 secret 传播到每个代理 Pod 的属性文件中。当 OpenShift Container Platform 将更新的 secret 传播到每个 Pod 时，其状态会更新。



注意

代理定期检查配置更改，包括对 Pod 上挂载的属性文件的更新，并在检测到任何更改时重新载入配置。但是，在重启代理前，对仅读取的属性的更新（如 JVM 设置）不会被重新载入。有关重新加载哪些属性的更多信息，*请参阅配置 AMQ Broker* 中的 Reloading 配置更新。

其它信息

有关在 CR 中的 brokerProperties 元素中配置的属性列表，*请参阅配置 AMQ Broker 中的 Broker 属性*。

4.23.1. 隔离 brokerProperties 配置

如果您的 CR 包含 brokerProperties 部分，并且 CR 处于最大大小限制 1MB，您可以在一个或多个 Java 属性文件中隔离 brokerProperties 配置。您可能还想在单独的文件中隔离 brokerProperties 配置，以逻辑方式对 brokerProperties 项进行分组，以便更轻松地维护。

流程

1. 以 Java 属性格式创建一个文件，其中包含您要应用到代理的 brokerProperties 配置。在属性文件中的单独行中添加每个属性。例如：

```
securityRoles.address1.group2.send=true
securityRoles.address2.group1.consume=true
securityRoles.address2.group2.createAddress=true
```

2. 使用 .properties 扩展保存文件，如 securityRoles.properties。

3. 创建包含您创建的 .properties 文件的 secret 或 configMap。此流程中的剩余步骤假定您创建 secret。

```
oc create secret generic address-settings-bp --from-file=securityRoles.properties
```

注意

secret 名称必须具有 -bp 后缀。当 secret 具有 -bp 后缀时，Operator 会将代理配置为搜索在代理 pod 上挂载 secret 的目录中的属性文件。

4. 在 extraMounts 属性中添加对 secret 的引用，以便 Operator 在每个代理 pod 上挂载 secret 中的属性文件：

```
deploymentPlan:
  ...
  extraMounts:
    secrets:
    - "address-settings-bp"
  ...
```

Operator 在每个代理 pod 上的 /amq/extra/secrets/ <secret*name*> *目录中挂载 secret* 中的 .properties 文件。

在启动时，代理会在每个挂载的目录中搜索具有 .properties 扩展名的文件，按字母顺序对文件进行排序，并在文件中应用配置。在属性文件中，代理会根据列出的顺序应用属性。

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 中托管自己的 AMQ 管理控制台实例。

以下流程描述了如何连接到已部署代理的 AMQ 管理控制台。

先决条件

- 已使用 AMQ Broker Operator 创建代理部署。例如，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”。
- 您为部署中的代理启用对 AMQ 管理控制台的访问。有关启用对 AMQ 管理控制台的访问的更多信息，请参阅 第 4.8 节 “启用对 AMQ 管理控制台的访问”。

5.1. 连接到 AMQ 管理控制台

当您为代理部署启用对 AMQ 管理控制台的访问时，Operator 会自动为每个代理 Pod 创建一个专用服务和路由，以提供对 AMQ 管理控制台的访问。

自动创建服务的默认名称格式为 < *custom-resource-name*> -wconsj- <*broker-pod-ordinal*>-svc。例如，my-broker-deployment-wconsj-0-svc。自动创建的 Route 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，识别给定代理 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。

在您的 Web 浏览器中打开一个新标签页。

3. 单击 **Management Console** 链接。

此时会打开 AMQ Management Console 登录页面。

注意

只有在 CR 的 requireLogin 属性设置为 true 时，才需要凭证登录到 AMQ 管理控制台。此属性指定是否需要登录凭证才能登录到 代理和 AMQ 管理控制台。默认情况下，requireLogin 属性设为 false。如果将 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 AMQ 管理控制台，而无需提供有效的用户名和密码。

4. 如果 requireLogin 属性设置为 true，请输入用户名和密码。

您可以为预配置的用户输入凭证，可用于连接到代理和 AMQ 管理控制台。如果在自定义资源 (CR)实例中配置了这些属性，您可以在 adminUser 和 adminPassword 属性中找到这些凭证。它不会在 CR 中配置这些属性，Operator 会自动生成凭证。要获取自动生成的凭证，请参阅 第 5.2 节 “访问 AMQ 管理控制台登录凭证”。

如果要以任何其他用户身份登录，请注意，用户必须属于为 hawtio.role 系统属性指定的安全角色，才能具有登录 AMQ 管理控制台所需的权限。hawtio.role 系统属性的默认角色是 admin，预配置的用户属于这个属性。

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 名称格式为 < *custom-resource-name&gt*; -credentials-secret，如 my-broker-deployment-credentials-secret。

注意

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

如果将 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据。

流程

1. 请参阅 OpenShift 项目中的 secret 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. 打开适当的机密，以显示 Base64 编码的控制台登录凭据。

   a. 在 OpenShift Container Platform Web 控制台中，点击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，请使用以下命令：

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

其他资源

- 要了解更多有关使用 AMQ 管理控制台查看和管理代理的信息，请参阅管理 AMQ *Broker* 中的使用 *AMQ 管理控制台* 管理代理。

第 6 章 升级基于 OPERATOR 的代理部署

本节中的步骤演示了如何升级：

- AMQ Broker Operator 版本，使用 OpenShift 命令行界面(CLI)和 OperatorHub
- 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事项。

- 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 *升级* Operator。如果您最初使用 OperatorHub 安装 Operator （即，它出现在 OpenShift Container Platform Web 控制台中项目的 Operators → Installed Operators 下），您还应使用 OperatorHub 升级 Operator。有关这些升级方法的更多信息，请参阅：
  - 第 6.2 节 “使用 CLI 升级 Operator”
  - 第 6.3 节 “使用 OperatorHub 升级 Operator”
- 如果 redeliveryDelayMultiplier 和 redeliveryCollisionAvoidanceFactor 属性在 7.8.x 或 7.9.x 部署的主代理 CR 中配置，则新 Operator 在升级到 7.10.x 或更高版本后无法协调任何 CR。协调失败，因为这两个属性的数据类型从 float 改为 7.10.x 中的字符串。

  您可以通过从 spec.deploymentPlan.addressSettings.addressSetting 属性中删除 redeliveryDelayMultiplier 和 redeliveryCollisionAvoidanceFactor 属性来解决这个问题。然后，在 brokerProperties 属性下配置属性。例如：

```
spec:
  ...
  brokerProperties:
```

```
- "addressSettings.#.redeliveryMultiplier=2.1"
- "addressSettings.#.redeliveryCollisionAvoidanceFactor=1.2"
```

注意

在 brokerProperties 属性下，使用 redeliveryMultiplier 属性名称，而不是您删除的 redeliveryDelayMultiplier 属性名称。

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)将不同版本 Operator 升级到 AMQ Broker 7.12 可用版本。

6.2.1. 先决条件

- 只有在最初使用 CLI *安装* Operator 时，才使用 CLI 升级 Operator。如果您最初使用 OperatorHub 安装 Operator （即，Operator 出现在 OpenShift Container Platform Web 控制台的项目的 Operators 下），请使用 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. 使用 CLI 升级 Operator

您可以使用 OpenShift 命令行界面(CLI)将 Operator 升级到 AMQ Broker 7.12 的最新版本。

流程

1. 在 Web 浏览器中，导航到 AMQ Broker 7.12.0 的 Software Downloads 页面。
2. 确保 Version 下拉列表的值设为 7.12.0，并且选择了 Releases 选项卡。
3. 在 AMQ Broker 7.12.0 Operator Installation and Example Files 旁边，点 Download。

   下载 amq-broker-operator-7.12.0-ocp-install-examples.zip 压缩存档会自动开始。
4. 下载完成后，将存档移到您选择的安装目录中。以下示例将存档移到名为 ~/broker/operator 的目录中。

```
$ mkdir ~/broker/operator
$ mv amq-broker-operator-7.12.0-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-operator-7.12.0-ocp-install-examples.zip
```

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

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

7. 切换到要升级 Operator 版本的 OpenShift 项目。

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

8. 在您下载和提取的最新 Operator 归档的 deploy 目录中，打开 operator.yaml 文件。

注意

在 operator.yaml 文件中，Operator 使用一个由 *Secure Hash Algorithm* (SHA) 值代表的镜像。注释行以数字符号 (#) 符号开头，注明 SHA 值对应于特定的容器镜像标签。

9. 为 以前的 Operator 部署打开 operator.yaml 文件。检查您在之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. 在新的 operator.yaml 文件中，Operator 默认名为 amq-broker-controller-manager。如果上一个部署中的 Operator 名称不是 amq-broker-controller-manager，请将 amq-broker-controller-manager 的所有实例替换为以前的 Operator 名称。例如：

```
spec:
  ...
  selector
    matchLabels
      name: amq-broker-operator
  ...
```

11.

在新的 operator.yaml 文件中，Operator 的服务帐户名为 amq-broker-controller-manager。在以前的版本中，Operator 的服务帐户名为 amq-broker-operator。

a. 如果要在以前的部署中使用服务帐户名称，请将 新 operator.yaml 文件中的服务帐户名称替换为之前部署中使用的名称。例如：

```
spec:
  ...
  serviceAccountName: amq-broker-operator
  ...
```

b. 如果要使用新服务帐户名称 amq-broker-controller-manager 用于 Operator，更新项目中的服务帐户、角色和角色绑定。

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

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

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

12. 更新 Operator 中包含的 CRD。

a. 更新主代理 CRD。

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

b. 更新地址 CRD。

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

c. 更新 scaledown 控制器 CRD。

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

d. 更新安全 CRD。

```
$ oc apply -f deploy/crds/broker_activemqartemissecurity_crd.yaml
```

13. 如果您只从 AMQ Broker Operator 7.10.0 升级，请删除 Operator 和 StatefulSet。

默认情况下，新 Operator 删除 StatefulSet 以删除自定义和 Operator metering 标签，这些标签由 7.10.0 中的 Operator 错误地添加到 StatefulSet 选择器中。当 Operator 删除 StatefulSet 时，它还会删除现有代理 pod，这会导致临时代理中断。如果要避免停机，请完成以下步骤删除 Operator 和 StatefulSet，而不删除代理 pod。

a. 删除 Operator。

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

b. 使用 --cascade=orphan 选项删除 StatefulSet，以孤立代理 pod。在 StatefulSet 被删除后，孤立的代理 pod 会继续运行。

```
$ oc delete statefulset <statefulset-name> --cascade=orphan
```

14. 如果您要从 AMQ Broker Operator 7.10.0 或 7.10.1 升级，请检查您的主代理 CR 是否在 deploymentPlan.labels 属性中配置了名为 application 或 ActiveMQArtemis 的标签。

这些标签供 Operator 为 pod 分配标签，在 7.10.1 后不允许作为自定义标签。如果在主代理 CR 中配置了这些自定义标签，则 pod 上的 Operator 分配标签会被自定义标签覆盖。如果在主代理 CR 中配置了这些自定义标签，请完成以下步骤来恢复 pod 上的正确标签并从 CR 中删除标签。

a. 如果要从 7.10.0 升级，请删除上一步中的 Operator。如果要从 7.10.1 升级，请删除 Operator。

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

b. 运行以下命令来恢复正确的 pod 标签。在以下示例中，'ex-aao' 是部署的 StatefulSet 的名称。

```
$ for pod in $(oc get pods | grep -o '^ex-aao[^ ]*'); do oc label --overwrite pods
$pod ActiveMQArtemis=ex-aao application=ex-aao-app; done
```

c. 从 CR 中的 deploymentPlan.labels 属性中删除 application 和 ActiveMQArtemis 标签。

i. 以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。

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

ii. 打开一个名为 broker_activemqartemis_cr.yaml 的示例 CR 文件，该文件包含在您下载和提取的 Operator 安装的 deploy/crs 目录中。

iii. 在 CR 中的 deploymentPlan.labels 属性中，删除名为 application 或 ActiveMQArtemis 的任何自定义标签。

iv. 保存 CR 文件。

v. 部署 CR 实例。

A. 切换到代理部署的项目。

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

B. 应用 CR。

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

d. 如果删除了前面的 Operator，请部署新的 Operator。

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

15. 应用更新的 Operator 配置。

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

16. 新的 Operator 可以识别和管理之前代理部署。如果您在 CR 中的 image 或 version 字段中设置了值，Operator 的协调过程会在 Operator 启动时将代理 pod 升级到对应的镜像。如需更多信息，请参阅 第 6.4 节 “限制代理容器镜像的自动升级”。否则，Operator 会将每个代理 pod 升级到最新的容器镜像。

注意

如果协调过程没有启动，您可以通过扩展部署来启动该过程。更多信息请参阅 第 3.4.1 节 “部署基本代理实例”。

17. 根据需要，为升级代理中提供的新功能向 CR 添加属性。

6.3. 使用 OPERATORHUB 升级 OPERATOR

只有在最初使用 OperatorHub *安装* Operator （即，Operator 出现在 OpenShift Container Platform Web 控制台的项目的 Operators 下）时，使用 OperatorHub 升级 Operator。如果您最初使用 OpenShift 命令行界面(CLI)安装 Operator，请使用 CLI 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”。

注意

如果您要从 7.10.0 或 7.10.1 升级，请参阅描述如何完成 Operator 版本升级的特定部分。

6.3.1. 开始前

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

- 当您从 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 CRD。如果您删除现有 CRD，则所有 CR 和代理实例也会被删除。
- 当使用最新 Operator 版本的 CRD 更新集群时，这个更新会影响 集群中的所有项目。从 Operator 之前的版本部署的任何代理 pod 可能无法在 OpenShift Container Platform Web 控制台中更新其状态。当您点正在运行的代理 pod 的 Logs 选项卡时，您会看到指示 'UpdatepodStatus' 失败的消息。但是，该项目中的代理 pod 和 Operator 将继续按预期工作。要为受影响的项目修复此问题，还必须升级该项目以使用最新版本的 Operator。
- 如果您要从 7.10.0 或 7.10.1 升级，请参阅描述如何完成 Operator 版本升级的特定部分。升级这些版本需要额外的步骤来防止 Operator 升级在部署中重启代理 pod。

  第 6.3.3 节 “从 7.10.0 升级 Operator”

第 6.3.4 节 “从 7.10.1 升级 Operator”

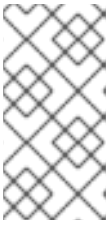
6.3.2. 升级 Operator

您必须卸载当前的 Operator 并安装新的 Operator 来完成升级。

流程

1. 以集群管理员身份登录 OpenShift Container Platform Web 控制台。
2. 从项目中卸载现有的 AMQ Broker Operator。
3. 在左侧导航菜单中，点 InstalledOperators 。
4. 在页面顶部的 Project 下拉菜单中选择您要卸载 Operator 的项目。
5. 找到您要卸载的 Red Hat Integration - AMQ Broker 实例。
6. 对于 Operator 实例，点右侧的 More Options 图标（三个垂直点）。点击 Uninstall Operator。
7. 在确认对话框中，点 Uninstall。
8. 使用 OperatorHub 为 AMQ Broker 7.12 安装最新版本的 Operator。更多信息请参阅 第 3.3.2 节 “从 OperatorHub 部署 Operator”。

新的 Operator 可以识别和管理之前代理部署。如果在 CR 中的 image 或 version 字段中设置了值，Operator 的协调过程会在 Operator 启动时将代理 pod 升级到对应的容器镜像。如需更多信息，请参阅 第 6.4 节 “限制代理容器镜像的自动升级”。否则，Operator 会将每个代理 pod 升级到最新的容器镜像。



注意

如果协调过程没有启动，您可以通过扩展部署来启动该过程。更多信息请参阅 第 3.4.1 节 “部署基本代理实例”。

6.3.3. 从 7.10.0 升级 Operator

您必须卸载 7.10.0 Operator 并安装新的 Operator 来完成升级。此流程包括防止新 Operator 在部署中重启代理 pod 的额外步骤，这会导致停机。

流程

1. 以集群管理员身份登录 OpenShift Container Platform Web 控制台。
2. 从项目中卸载现有的 AMQ Broker Operator。
   a. 在左侧导航菜单中，点 InstalledOperators 。
   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 Uninstall。
3. 当您升级 7.10.0 Operator 时，新 Operator 会删除 StatefulSet 以删除自定义和 Operator metering 标签，这些标签被 Operator 在 7.10.0 中错误地添加到 StatefulSet 选择器中。当 Operator 删除 StatefulSet 时，它还会删除现有代理 pod，这会导致临时代理中断。如果要避免中断，请完成以下步骤删除 StatefulSet 和孤立代理 pod，以便它们继续运行。
   i. 作为包含现有 Operator 部署的项目的管理员登录到 OpenShift Container Platform CLI：
   -

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

ii. 切换到要升级 Operator 版本的 OpenShift 项目。

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

iii. 使用 --cascade=orphan 选项删除 StatefulSet，以孤立代理 pod。在 StatefulSet 被删除后，孤立的代理 pod 会继续运行。

```
$ oc delete statefulset <statefulset-name> --cascade=orphan
```

4. 检查您的主代理 CR 是否在 deploymentPlan.labels 属性中配置名为 application 或 ActiveMQArtemis 的标签。

在 7.10.0 中，可以在 CR 中配置这些自定义标签。这些标签是为 Operator 分配标签分配给 pod 的保留，在 7.10.0 后无法添加为自定义标签。如果在 7.10.0 中的主代理 CR 中配置了这些自定义标签，则 pod 上的 Operator 分配标签会被自定义标签覆盖。如果 CR 具有这些标签，请完成以下步骤来恢复 pod 上的正确标签，并从 CR 中删除标签。

a. 在 OpenShift 命令行界面(CLI)中，运行以下命令来恢复正确的 pod 标签。在以下示例中，'ex-aao' 是部署的 StatefulSet 的名称。

```
$ for pod in $(oc get pods | grep -o '^ex-aao[^ ]*'); do oc label --overwrite pods
$pod ActiveMQArtemis=ex-aao application=ex-aao-app; done
```

b. 从 CR 中的 deploymentPlan.labels 属性中删除 application 和 ActiveMQArtemis 标签。

i. 使用 OpenShift 命令行界面：

A. 以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。

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

B. 编辑部署的 CR。

```
oc edit ActiveMQArtemis <statefulset name> -n <namespace>
```

C. 在 CR 中的 deploymentPlan.labels 元素中，删除名为 application 或 ActiveMQArtemis 的任何自定义标签。

D. 保存 CR。

ii. 使用 OpenShift Container Platform Web 控制台：

A. 以具有特权的用户身份登录控制台，以便在代理部署的项目中部署 CR。

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

C. 单击 ActiveMQArtemis CRD。

D. 点 实例 选项卡。

E. 点代理部署的实例。

F. 点 YAML 标签。

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

G. 在 CR 中的 deploymentPlan.labels 元素中，删除名为 application 或 ActiveMQArtemis 的任何自定义标签。

H. 点击 Save。

5. 使用 OperatorHub 为 AMQ Broker 7.12 安装最新版本的 Operator。更多信息请参阅 第 3.3.2 节 “从 OperatorHub 部署 Operator”。

新的 Operator 可以识别和管理之前代理部署。如果您在 CR 中的 image 或 version 字段中设置了值，Operator 的协调过程会在 Operator 启动时将代理 pod 升级到对应的镜像。如需更多信息，请参阅 第 6.4 节 “限制代理容器镜像的自动升级”。否则，Operator 会将每个代理 pod 升级到最新的容器镜像。

注意

如果协调过程没有启动，您可以通过扩展部署来启动该过程。更多信息请参阅 第 3.4.1 节 “部署基本代理实例”。

6. 根据需要，为升级代理中提供的新功能向 CR 添加属性。

6.3.4. 从 7.10.1 升级 Operator

您必须卸载 7.10.1 Operator，并安装新的 Operator 来完成升级。此流程包括您可能需要完成的额外步骤，具体取决于您的配置，以防止新的 Operator 重启代理 pod，从而导致停机。

流程

1. 以集群管理员身份登录 OpenShift Container Platform Web 控制台。

2. 检查您的主代理 CR 是否在 deploymentPlan.labels 属性中配置名为 application 或 ActiveMQArtemis 的标签。

   这些标签为 Operator 保留，用于将标签分配给 pod，且在 7.10.1 后无法使用。如果在主代理 CR 中配置了这些自定义标签，则 pod 上的 Operator 分配标签会被自定义标签覆盖。

3. 如果在主代理 CR 中没有配置这些自定义标签，请使用 OperatorHub 为 AMQ Broker 7.12 安装最新版本的 Operator。更多信息请参阅 第 3.3.2 节 “从 OperatorHub 部署 Operator”。

4. 如果在主代理 CR 中配置了这些自定义标签，请按照以下步骤卸载现有 Operator，在安装新 Operator 前恢复正确的 pod 标签并从 CR 中删除标签。

注意

通过卸载 Operator，您可以在没有 Operator 删除 StatefulSet 的情况下删除自定义标签，这也会删除现有代理 pod 并导致临时代理中断。

a. 从项目中卸载现有的 AMQ Broker Operator。

i. 在左侧导航菜单中，点 InstalledOperators 。

ii. 在页面顶部的 Project 下拉菜单中选择您要卸载 Operator 的项目。

iii. 找到您要卸载的 Red Hat Integration - AMQ Broker 实例。

iv. 对于 Operator 实例，点右侧的 More Options 图标（三个垂直点）。点击 Uninstall Operator。

v. 在确认对话框中，点 Uninstall。

b. 在 OpenShift 命令行界面(CLI)中，运行以下命令来恢复正确的 pod 标签。在以下示例中，'ex-aao' 是部署的 StatefulSet 的名称。

```
$ for pod in $(oc get pods | grep -o '^ex-aao[^ ]*'); do oc label --overwrite pods
$pod ActiveMQArtemis=ex-aao application=ex-aao-app; done
```

c. 从 CR 中的 deploymentPlan.labels 属性中删除 application 和 ActiveMQArtemis 标签。

i. 使用 OpenShift 命令行界面：

A. 以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。

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

B. **编辑部署的 CR。**

```
oc edit ActiveMQArtemis <statefulset name> -n <namespace>
```

C. **在 CR 中的 deploymentPlan.labels 属性中，删除名为 application 或 ActiveMQArtemis 的任何自定义标签。**

D. **保存 CR 文件。**

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

A. **以具有特权的用户身份登录控制台，以便在代理部署的项目中部署 CR。**

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

C. **单击 ActiveMQArtemis CRD。**

D. **点 实例 选项卡。**

E. **点代理部署的实例。**

F. **点 YAML 标签。**

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

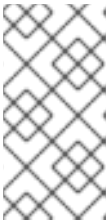
G. **在 CR 中的 deploymentPlan.labels 属性中，删除名为 application 或 ActiveMQArtemis 的任何自定义标签。**

H. **点击 Save。**

5. **使用 OperatorHub 为 AMQ Broker 7.12 安装最新版本的 Operator。更多信息请参阅**

第 3.3.2 节 “从 OperatorHub 部署 Operator”。

新的 Operator 可以识别和管理之前代理部署。如果您在 CR 中的 image 或 version 字段中设置了值，Operator 的协调过程会在 Operator 启动时将代理 pod 升级到对应的镜像。如需更多信息，请参阅 第 6.4 节 “限制代理容器镜像的自动升级”。否则，Operator 会将每个代理 pod 升级到最新的容器镜像。



注意

如果协调过程没有启动，您可以通过扩展部署来启动该过程。更多信息请参阅 第 3.4.1 节 “部署基本代理实例”。

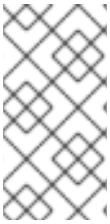
6. 根据需要，为升级代理中提供的新功能向 CR 添加属性。

6.4. 限制代理容器镜像的自动升级

默认情况下，Operator 会自动升级部署中的每个代理，以使用最新可用的容器镜像。在部署的自定义资源(CR)中，您可以通过指定版本号或特定容器镜像的 URL 来限制 Operator 升级镜像的能力。

6.4.1. 使用版本号限制镜像的自动升级

您可以在新版本可用时，限制代理自动升级的容器镜像版本。



注意

当您根据版本号限制升级时，Operator 继续自动升级代理以使用包含部署的安全修复的任何新镜像。

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. 以具有为代理部署的项目中编辑和部署 CR 的用户身份登录 OpenShift。

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

ii. 编辑 **CR**。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

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，以便在代理部署的项目中部署 **CR**。

ii. 在左侧窗格中，点 **Operators** → **Installed Operator**。

iii. 点 **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator**。

iv. 点 **AMQ Broker** 选项卡。

v. 单击 **ActiveMQArtemis** 实例名称的名称。

vi. 点 **YAML** 标签。

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

注意

在 **CR** 的 **status** 部分中，**.status.version.brokerVersion** 字段显示当前部署的 **AMQ Broker** 版本。

2. 在 **spec.version** 属性中，指定 **Operator** 可以在部署中升级代理和 **init** 容器镜像的版本。以下是您可以指定的值示例。

例子

在以下示例中，**Operator** 将部署中的当前容器镜像升级到 **7.12.0**。

```
spec:
  version: '7.12.0'
   ...
```

在以下示例中，Operator 将部署中的当前容器镜像升级到最新可用的 7.11.x 镜像。例如，如果您的部署使用 7.11.1 容器镜像，Operator 会自动将镜像升级到 7.11.6，而不是 7.12.0。

```
spec:
  version: '7.11'
   ...
```

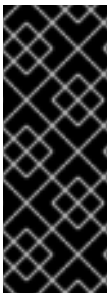
在以下示例中，Operator 将部署中的当前容器镜像升级到最新的 7.x.x 镜像。例如，如果您的部署使用 7.11.6 镜像，Operator 会自动将镜像升级到 7.12.0。

```
spec:
  version: '7'
   ...
```

注意

要在容器镜像的次版本间（例如从 7.11.x 升级到 7.12.x），您需要一个与新容器镜像相同的次版本的 Operator。例如，若要从 7.11.6 升级到 7.12.0，必须安装 7.12.x Operator。

3. 保存 CR。



重要

如果您使用 CR 中的 spec.version 属性来限制代理容器镜像的自动升级，请确保 CR 不包含 spec.deploymentPlan.image 或 spec.deploymentPlan.initImage 属性。这两个属性都覆盖 spec.version 属性。如果 CR 具有这些属性之一以及 spec.version 属性，则部署的代理和 init 镜像版本可能会被分离，这可能会阻止代理运行。

保存 CR 时，Operator 首先验证为 spec.version 指定的 AMQ Broker 版本是否可用于现有部署。如果您指定了要升级到的 AMQ Broker 的无效版本，例如：一个不可用的版本，Operator 会记录警告信息，且不会执行进一步的操作。

但是，如果对指定版本的升级可用，Operator 会 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_7120 对应于 AMQ Broker 7.12.0。

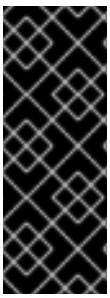
当 Operator 应用 CR 更改时，它会重启部署中的每个代理 pod，以便每个 pod 使用指定的镜像版本。如果部署中有多个代理，则只有一个代理 pod 会一次关闭并重启。

其他资源

- 要了解 Operator 如何使用环境变量来选择代理容器镜像，请参阅 第 2.6 节 “Operator 如何选择容器镜像”。
- 要查看部署的状态，请参阅 第 2.7 节 “在自定义资源(CR)中验证镜像和版本配置”

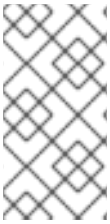
6.4.2. 使用镜像 URL 限制镜像自动升级

如果要升级部署中的代理以使用特定的容器镜像，您可以在 CR 中指定镜像的 registry URL。在 Operator 将代理升级到指定的容器镜像后，在替换 CR 中的镜像 URL 前，不会进一步升级。例如，Operator 不会自动升级代理以使用包含部署的镜像的安全修复的较新的镜像。



重要

如果要使用镜像 URL 限制自动升级，请为 CR 中的 spec.deploymentPlan.image 和 spec.deploymentPlan.initImage 属性指定 URL，以确保代理和 init 容器镜像匹配。如果您只指定了一个容器镜像的 URL，则代理和 init 容器镜像可以分离，这可能会阻止代理运行。



注意

如果 CR 在 spec.deploymentPlan.image 和 spec.deploymentPlan.initImage 属性之外有一个 spec.version 属性，Operator 会忽略 spec.version 属性。

流程

1. 获取 Operator 可以升级当前镜像的代理和 init 容器镜像的 URL。

a. 在 Red Hat Catalog 中，打开代理容器组件页面： 适用于 RHEL 8 的 AMQ Broker (Multiarch)。

b. 在 Architecture 下拉菜单中，选择您的架构。

c. 在 Tag 下拉菜单中，选择与您要安装的镜像对应的标签。标签根据发行日期按时间顺序显示。标签由发行版本和分配的标签组成。

d. 打开 Get this image 选项卡。

e. 在 Manifest 字段中，单击 Copy 图标。

f. 将 URL 粘贴到文本文件中。

g. 在 Red Hat Catalog 中，打开 init 容器组件页面： 适用于 RHEL 8 的 AMQ Broker Init （多架构）

h. 要获取 init 容器镜像的 URL，请重复上述步骤，以获取代理容器镜像的 URL。

2. 编辑代理部署的主代理 CR 实例。

a. 使用 OpenShift 命令行界面：

i. 以具有为代理部署的项目中编辑和部署 CR 的用户身份登录 OpenShift。

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

ii. 编辑 CR。

```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

.

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，以便在代理部署的项目中部署 CR。

ii. 在左侧窗格中，点 Operators → Installed Operator。

iii. 点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator。

iv. 点 AMQ Broker 选项卡。

v. 单击 ActiveMQArtemis 实例名称的名称。

vi. 点 YAML 标签。

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例

c. 复制您在文本文件中记录的代理和 init 容器镜像的 URL，并将它们插入到 CR 中的 spec.deploymentPlan.image 和 spec.deploymentPlan.initImage 字段中。例如：

```
spec:
  ...
  deploymentPlan:
    image: registry.redhat.io/amq7/amq-broker-
rhel8@3cc58e98c905d427d6be214a8df3c2687170f74abeacc33366f91bfc18e69a3a
    initImage: registry.redhat.io/amq7/amq-broker-init-
rhel8@227fbe6e428848a0edb8086a00f8d9d993b184392e1f4161febff949ce943cbf
  ...
```

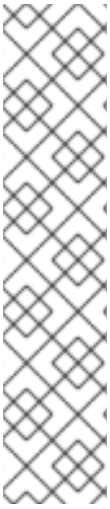
3. 保存 CR。

保存 CR 时，Operator 会升级代理以使用新镜像，并使用这些镜像，直到您再次更新 spec.deploymentPlan.image 和 spec.deploymentPlan.initImage 属性的值。

4. 如果要防止将来的 Operator 升级来重启部署中的代理，请编辑 CR 并指定在 spec.version 属性中部署的代理的版本号。

如果 CR 中没有配置 spec.version 属性，则后续 Operator 升级会导致代理 pod 重启。pod 重启是必需的，因为新 Operator 将最新的支持的代理版本添加到 StatefulSet 中的标签，除非 spec.version 属性中明确设置了版本号。

您可以在代理启动后在 CR 的 status 部分找到为 spec.version 属性指定的版本号值。如需更多信息，请参阅查看代理部署的状态信息。



注意

如果您在没有设置镜像 URL 的情况下部署了 AMQ Broker，您可以原样设置镜像 URL，以防止 Operator 升级部署的当前镜像。您可以找到 .status.version.image 和 .status.version.initImage 属性中部署的镜像的 registry URL，它们位于 CR 的 status 部分。

如果您从 .status. version.image 和 .status.version.initImage 属性复制镜像 URL，并将它们分别插入到 spec.deploymentPlan.image 和 spec.deploymentPlan.initImage 属性中，Operator 不会升级当前部署的镜像。

其它资源

- 要查看部署的状态，请参阅 第 2.7 节 “在自定义资源(CR)中验证镜像和版本配置”。

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，以使用 Fuse 控制台进行 OpenShift 而不是 AMQ Management 控制台。当您正确配置了代理部署时，Fuse Console 会发现代理并在专用 Artemis 选项卡中显示它们。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，以便为 OpenShift 启用 Fuse 控制台来发现和显示部署中的代理。

先决条件

- 用于 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。如果您将控制台部署到特定命名空间中，您的代理部署必须 位于同一命名空间中，以便控制台能够发现代理。否则，Fuse Console 和要部署到同一 OCP 集群中的代理就足够了。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，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 可能类似以下：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.12
        ...
```

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/v1beta1
kind: ActiveMQArtemis
```

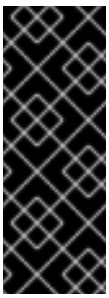
```
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.12
    ...
    jolokiaAgentEnabled: true
    managementRBACEnabled: false
```

jolokiaAgentEnabled

指定 Fuse 控制台是否可以发现和显示部署中代理的运行时数据。要使用 Fuse 控制台，请将值设为 **true**。

managementRBACEnabled

指定是否为部署中的代理启用基于角色的访问控制(RBAC)。您必须将 值设为 **false** 以使用 Fuse 控制台，因为 Fuse 控制台使用自己的基于角色的访问控制。



重要

如果将 **managementRBACEnabled** 的值设置为 **false** 来启用使用 Fuse 控制台，则代理的管理 MBeans 不再需要授权。当 **managementRBACEnabled** 被设置为 **false** 时，您不应该使用 AMQ 管理控制台，因为这可能会公开代理上的所有管理操作到未授权使用。

3. 保存 CR 实例。

4. 切换到之前创建的代理部署的项目。

```
$ 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 Artemis。

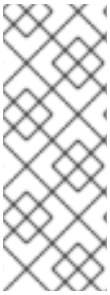
其他资源

- 有关将 Fuse 控制台用于 OpenShift 的更多信息，请参阅在 OpenShift 中监控和管理红帽 Fuse 应用程序。

- 要了解使用 AMQ 管理控制台以与 Fuse 控制台中相同的方式查看和管理代理的信息，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

以下章节描述了如何在 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 等图形工具来配置更多高级视觉化和 Prometheus 插件收集的数据的仪表板。



注意

Prometheus metrics 插件可让您以 Prometheus 格式 收集和导出代理指标。但是，红帽不提供对 安装或配置 Prometheus 本身的支持，也不提供 Grafana 等视觉化工具的支持。如果您需要支持安装、配置或运行 Prometheus 或 Grafana，请访问产品网站以了解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，可让您使用 Prometheus 本身视觉化并在数据上运行查询。

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

此代理上的所有地址用于内存中消息的字节数。

artemis_address_memory_usage_percentage

此代理上的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量。

artemis_total_connection_count

自此代理启动以来已连接到此代理的客户端数量。

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

*没有*路由到任何队列绑定的消息数。

队列指标

artemis_consumer_count

使用来自给定队列的消息的客户端数量。

artemis_delivering_durable_message_count

指定队列当前提供给消费者的持久消息数量。

artemis_delivering_durable_persistent_size

给定队列当前提供给消费者的持久消息大小。

artemis_delivering_message_count

指定队列当前提供给消费者的消息数。

artemis_delivering_persistent_size

指定队列当前提供给消费者的持久消息大小。

artemis_durable_message_count

当前在给定队列中的持久消息数量。这包括已调度、页面和传送的消息。

artemis_durable_persistent_size

当前在给定队列中的持久性消息大小。这包括已调度、页面和传送的消息。

artemis_messages_acknowledged

自创建队列以来，从给定队列确认的消息数。

artemis_messages_added

由于队列创建，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。这包括已调度、页面和传送的消息。

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次数时，代理会终止一条消息。

artemis_messages_expired

自创建队列后，从给定队列已过期的消息数。

artemis_persistent_size

当前在给定队列中所有消息的持久大小（包括持久和非持久）。这包括已调度、页面和传送的消息。

artemis_scheduled_durable_message_count

在给定队列中的持久调度消息数量。

artemis_scheduled_durable_persistent_size

持久大小，在给定队列中调度的消息。

artemis_scheduled_message_count

指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。

对于上面未列出的高级别代理指标，您可以通过聚合较低级别指标来计算这些指标。例如，要计算总

消息数，您可以聚合代理部署中的所有队列的 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 metrics 插件。启用后，插件会为代理收集运行时指标，并将其导出为 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 或更高版本的新部署。

有关使用运行代理的替代流程，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 可能类似以下：

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.12
  ...
```

2. 在 deploymentPlan 部分中，添加 enableMetricsPlugin 属性，并将值设为 true，如下所示。

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
```

```
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.12
    ...
    enableMetricsPlugin: true
```

enableMetricsPlugin

指定是否为部署中的代理启用 Prometheus 插件。

3. 保存 CR 实例。

4. 切换到之前创建的代理部署的项目。

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

5. 在命令行中应用更改。

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

metrics 插件开始以 Prometheus 格式收集代理运行时指标。

其他资源

- 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正在运行的代理部署”。

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。有关替代流程，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，您部署的代理必须使用 AMQ Broker 7.7 或更高版本的代理容器镜像。

流程

1. 使用包含代理部署的项目的管理员特权登录到 OpenShift Container Platform Web 控制

台。

2. 在 Web 控制台中，点 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 Workloads → StatefulSets → DeploymentConfigs。

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 Environment 选项卡。

6. 添加新的环境变量 AMQ_ENABLE_METRICS_PLUGIN。将 变量的值设置为 true。

当您设置 AMQ_ENABLE_METRICS_PLUGIN 环境变量时，OpenShift 会在 StatefulSet 或 DeploymentConfig 中重启每个代理 Pod。当部署中有多个 Pod 时，OpenShift 会依次重启每个 Pod。当每个代理 Pod 重启时，该代理的 Prometheus 插件将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

- 您必须已经为代理 Pod 启用 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. 对于您要访问其指标的代理 Pod，您需要识别之前创建的 Route，以将 Pod 连接到 AMQ Broker 管理控制台。Route 名称表单是访问指标所需的 URL 的一部分。

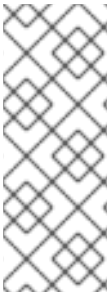
   a. 单击 Networking → Routes。

   b. 对于您选择的代理 Pod，识别为将 Pod 连接到 AMQ Broker 管理控制台而创建的路由。在 Hostname 下，记录显示的完整 URL。例如：

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

2. 要访问 Prometheus 指标，请在网页浏览器中输入之前记录的 Route 名称，并附带 "/metrics "。例如：

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



注意

如果您的控制台配置没有使用 SSL，请在 URL 中指定 http。在本例中，主机名的 DNS 解析会将流量定向到 OpenShift 路由器的端口 80。如果您的控制台配置使用 SSL，在 URL 中指定 https。在这种情况下，您的浏览器默认为 OpenShift 路由器的端口 443。如果 OpenShift 路由器也为路由器默认使用端口 443，则启用与控制台的连接。

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 接口来监控代理。

先决条件

- 建议完成部署基本代理。

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. **运行查询来监控您的代理 MaxConsumers ：**

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

{"request":{"mbean":"org.apache.activemq.artemis:address=\"TESTQUEUE\",broker=\"amq-
```

broker\",component=addresses,queue=\"TESTQUEUE\",routing-type=\"anycast\",subcomponent=queues","attribute":"MaxConsumers","type":"read"},"value":-1,"timestamp":1528297825,"status":200}

第 8 章 参考

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 对象的配置项目的模式。通过部署对应的自定义资源(CR)实例，您可以为 CRD 中显示的配置项指定值。

以下子部分详细介绍了您可以根据主代理 CRD 在自定义资源实例中设置的配置项目。

8.1.1. 代理自定义资源配置参考

基于主代理 CRD 的 CR 实例允许您为 OpenShift 项目中的部署配置代理。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)中需要标有星号(*)的配置项目。如果没有为非必需项目显式指定值，则配置将使用默认值。

entry	sub-entry	描述和用法
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，则会自动生成并存储在 secret 中的值。默认 secret 名称的格式是 <***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

entry	sub-entry	描述和用法
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，则会自动生成并存储在 secret 中的值。默认 secret 名称的格式是 <***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
ingressDomain		将自定义域附加到路由中的主机名，以及为接收器、连接器和管理控制台创建的入口。 类型: 字符串 **示例** ： mydomain.com
deploymentPlan*		代理部署配置

entry	sub-entry	描述和用法
	image*	用于部署中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 默认值表示 Operator 尚未确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 2.6 节 “Operator 如何选择容器镜像”。** 类型: **字符串** **示例 ： registry.redhat.io/amq7/amq-broker-rhel8@sha256:3cc58e98c905d427d6be214a8df3c2687170f74abeacc33366f91bfc18e69a3a** **默认值 ：占位符**

entry	sub-entry	描述和用法
	size*	要在部署中创建的代理 Pod 数量。 如果您指定了 2 或更高值，则代理部署默认是集群的。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 **adminPassword** 相同的 secret 中。 **键入**: int **示例** ：1 **默认值** ：1
	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**: true
	persistenceEnabled	指定是否为部署中的每个代理 Pod 使用日志存储。如果设置为 **true**，则每个代理 Pod 都需要一个可用的持久性卷(PV)，Operator 可以使用持久性卷声明(PVC)进行声明。 **类型**: 布尔值 **示例** ：false **默认值**: true

entry	sub-entry	描述和用法
	initImage	用于配置代理的 init 容器镜像。 除非您要提供自定义镜像，否则不需要为 CR 中明确指定 **initImage** 值。 要了解 Operator 如何选择要使用的内置初始容器镜像，请参阅 第 2.6 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 4.11 节 “指定自定义初始容器镜像”。 **类型**: 字符串 **示例** ：registry.redhat.io/amq7/amq-broker-init-rhel8@sha256:227fbe6e428848a0edb8086a00f8d9d993b184392e1f4161febff949ce943cbf **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** ：nio
	messageMigration	当代理 Pod 因代理部署有意缩减而关闭时，指定是否将消息迁移到仍然在代理集群中运行的另一个代理 Pod。 **类型**: 布尔值 **示例** ：false **默认值**: true

entry	sub-entry	描述和用法
	resources.limits.cpu	部署中运行的每个代理容器都可以消耗的最大主机节点 CPU 数量（以 millicores 为单位）。 **类型**: 字符串 **示例**: "500m" **默认值** ：使用与 OpenShift Container Platform 版本相同的默认值。查阅集群管理员。
	resources.limits.memory	在部署中运行的每个代理容器都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（例如，K、M、G）或二进制等效的(Ki、Mi、Gi)。 **类型**: 字符串 **示例**: "1024M" **默认值** ：使用与 OpenShift Container Platform 版本相同的默认值。查阅集群管理员。
	resources.requests.cpu	主机节点 CPU 的数量，单位为 millicores，即部署中 Pod 中运行的每个代理容器，明确请求。 **类型**: 字符串 **示例**: "250m" **默认值** ：使用与 OpenShift Container Platform 版本相同的默认值。查阅集群管理员。

entry	sub-entry	描述和用法
	resources.requests.memory	主机节点内存量（以字节为单位），每个代理容器在部署中的 Pod 中运行明确请求。支持字节表示法（例如，K、M、G）或二进制等效的(Ki、Mi、Gi)。 **类型**: 字符串 **示例** ："512M" **默认值** ：使用与 OpenShift Container Platform 版本相同的默认值。查阅集群管理员。
	storage.size	部署中的每个代理需要的持久性卷声明(PVC)的大小，以字节为单位。此属性仅在 **persistenceEnabled** 设为 **true** 时才适用。您指定的值 **必须包含一个** 单元。支持字节表示法（例如，K、M、G）或二进制等效的(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 代理。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**: false

entry	sub-entry	描述和用法
	managementRBACEnabled	指定是否为部署中的代理启用基于角色的访问控制 (RBAC)。要使用 Fuse 控制台，**您必须** 将值设为 **false**，因为 Fuse 控制台使用自己的基于角色的访问控制。 **类型**: 布尔值 **示例** ：false **默认值**: true
	关联性	指定 pod 的调度限制。如需有关关联性属性的信息，请参阅 OpenShift Container Platform 文档中的 属性。
	容限（tolerations）	指定 pod 的容限。如需有关容限属性的信息，请参阅 OpenShift Container Platform 文档中的 属性。
	nodeSelector	指定与要调度到该节点上的 pod 的节点标签匹配的标签。
	storageClassName	指定用于持久性卷声明 (PVC)的存储类的名称。存储类为管理员提供了一种描述和分类可用存储的方法。例如，存储类可能具有特定的服务质量级别、备份策略或其他与之关联的管理策略。 **类型**: 字符串 **示例** ： gp3 **默认值** ：未指定

entry	sub-entry	描述和用法
	startupProbe	配置启动探测来检查代理容器中的 AMQ Broker 应用程序是否已启动。有关启动探测属性的详情，请查看 OpenShift Container Platform 文档中的 属性。
	livenessProbe	在运行的代理容器上配置定期健康检查，以检查代理是否正在运行。如需有关存活度探测属性的信息，请参阅 OpenShift Container Platform 文档中的 属性。
	readinessProbe	在运行的代理容器上配置定期健康检查，以检查代理是否接受网络流量。有关就绪度探测属性的详情，请查看 OpenShift Container Platform 文档中的 属性。
	extraMounts	挂载包含配置信息的 secret 或 configMAP，作为代理 Pod 上的文件。例如，您可以挂载包含 AMQ Broker 的自定义日志记录配置的 secret。 **类型** ：对象 **示例** see 第 4.18 节 “为代理配置日志记录” **默认值** ：未指定
	labels	为代理 pod 分配标签。 **类型**: 字符串 **示例** ：位置："production" **默认值** ：未指定

entry	sub-entry	描述和用法
	podSecurityContext	定义用于运行代理 pod 的安全选项。以下默认安全值允许代理 Pod 在 OpenShift Container Platform 受限安全上下文约束(SCC)上运行： **runAsNonRoot: true** **SeccompProfile:type:RuntimeDefault** 如果您希望代理在自定义 SCC 上运行，您可以在 CR 中配置以下 **podSecurityContext** 选项。如果在 CR 中配置任何 **podSecurityContext** 选项，则不应用默认值，因此您必须配置在自定义 SCC 下运行所需的所有选项。 • **fsGroup** • **fsGroupChangePolicy** • **runAsGroup** • **runAsUser** • **runAsNonRoot** • **seLinuxOptions** • **seccompProfile** • **supplementalGroups** • **sysctls** • **windowsOptions** 如需有关 **podSecurityContext** 选项的信息，请参阅 OpenShift Container Platform 文档中的 属性。
	containerSecurityContext	定义用于在 pod 中运行代理容器的安全选项。使用以下默认值，容器在

entry	sub-entry	描述和用法
		OpenShift Container Platform 受限安全上下文约束(SCC)上运行： • **allowPrivilegeEscalation:false** • **capabilities:drop:ALL** • **runAsNonRoot:true** • **SeccompProfile:type:RuntimeDefault** 如果您希望代理在自定义 SCC 上运行，您可以在 CR 中配置以下 **containerSecurityContext** 选项。如果在 CR 中配置任何 **containerSecurityContext** 选项，则不应用默认值，因此您必须配置在自定义 SCC 下运行所需的所有选项。 • **allowPrivilegeEscalation** • **功能** • **privileged** • **procMount** • **readOnlyRootFilesystem** • **runAsGroup** • **runAsNonRoot** • **runAsUser** • **seLinuxOptions** • **seccompProfile** • **windowsOptions** 如需有关 **containerSecurityContext** 选项的信息，请参阅 OpenShift Container Platform 文档中的 属性。

entry	sub-entry	描述和用法
	podSecurity.serviceAccountName	为代理 pod 指定服务帐户名称。 **类型**: 字符串 **示例** ： amq-broker-controller-manager **默认值** ：default
控制台		配置代理管理控制台。
	expose	指定是否将管理控制台公开给 OpenShift Container Platform 之外的客户端。 **类型**: 布尔值 **示例**: true **默认值**: false
	exposeMode	指定是否使用路由或入口公开管理控制台。默认情况下，管理控制台仅使用路由公开。 **类型** ：字符串 **示例** ： ingress **默认值** ：route 如果使用 ingress 公开控制台，则必须在 CR 中指定 **ingressHost** 或 **ingressDomain** 值。

entry	sub-entry	描述和用法
	ingressHost	为路由指定自定义主机值，并为管理控制台公开入口。您可以在 host 值中包含以下变量： * $(CR_NAME)- CR 中的 **metadata.name** 属性的值。 * $(CR_NAMESPACE)- 自定义资源的命名空间。 * $(BROKER_ORDINAL)- StatefulSet 分配给代理 pod 的普通号。 * $(ITEM_NAME)- 控制台名称。默认名称为 **wconsj** * $(RES_TYPE)- 资源类型。路由具有 **rte** 资源类型。入口的资源类型为 **ing**。 * $(INGRESS_DOMAIN)- 如果在 CR 中配置，**spec.ingressDomain** 属性的值。 **类型**: 字符串 **示例** ：console-$(CR_NAME)-$(ITEM_NAME)-$(BROKER_ORDINAL).mydomain.com
	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**: false

entry	sub-entry	描述和用法
	sslSecret	存储代理密钥存储、信任存储及其对应密码的 secret（所有 Base64 编码）。如果没有为 **sslSecret** 指定值，控制台将使用默认 secret 名称。默认 secret 名称是 <***custom_resource_name***> **-console-secret** 的形式。此属性仅在 **sslEnabled** 属性设置为 **true** 时才应用。 **类型**: 字符串 **示例** ： my-broker-deployment-console-secret **默认值** ：未指定
	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**: false
acceptors.acceptor		单个接收器配置实例。
	name*	接受者的名称。 **类型**: 字符串 **示例** ： my-acceptor **默认值** ：不适用
	port	用于 acceptor 实例的端口号。 **键入**: int **示例** ：5672 **您定义的第一个接受者的默认值**: 61626。然后，对于您定义的每个后续接受者，默认值会按 10 递增。

entry	sub-entry	描述和用法
	协议	要在 acceptor 实例上启用的消息传递协议。 **类型**: 字符串 **示例**: amqp,core **默认值** ：all
	sslEnabled	指定在接受器端口上是否启用了 SSL。如果设置为 **true**，请查找 TLS/SSL 所需的凭证的 **sslSecret** 中指定的 secret 名称。 **类型**: 布尔值 **示例**: true **默认值**: false
	sslSecret	存储代理密钥存储、信任存储及其对应密码的 secret（所有 Base64 编码）。 如果没有为 **sslSecret** 指定自定义 secret 名称，则接受者会假定默认 secret 名称。默认 secret 名称的格式是 < ***custom_resource_name> - <acceptor_name>-*** **secret**。 您必须始终自己创建此 secret，即使接受者假设默认名称。 **类型**: 字符串 **示例** ： my-broker-deployment-my-acceptor-secret **默认值** : <*custom_resource_name*> - <*acceptor_name*> - secret

entry	sub-entry	描述和用法
	enabledCipherSuites	用于 TLS 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。如果您指定了代理和客户端通用的、以逗号分隔的密码套件列表，或者您没有指定任何密码套件、代理和客户端相互协商要使用的密码套件。如果您不知道要指定哪个密码套件，您可以首先与以 debug 模式运行的客户端建立 broker-client 连接，以验证代理和客户端通用的密码套件。然后，在代理上配置 **enabledCipherSuites**。 可用的密码套件取决于代理和客户端使用的 TLS 协议版本。如果在升级代理后默认 TLS 协议版本有变化，您可能需要选择早期的 TLS 协议版本，以确保代理和客户端可以使用通用密码套件。如需更多信息，请参阅 **enabledProtocols**。 类**型**: 字符串 **默**认值 ：未指定

entry	sub-entry	描述和用法
	enabledProtocols	用于 TLS 通信的、以逗号分隔的协议列表。 **类型**: 字符串 **示例** ：TLSv1,TLSv1.1,TLSv1.2 **默认值** ：未指定 如果没有指定 TLS 协议版本，代理将使用 JVM 的默认版本。如果代理使用 JVM 的默认 TLS 协议版本，且升级代理后该版本会改变，代理和客户端使用的 TLS 协议版本可能会不兼容。虽然建议您使用更新的 TLS 协议版本，但您可以在 **enabledProtocols** 中指定较早的版本，以便与不支持较新的 TLS 协议版本的客户端进行交互。
	keyStoreProvider	代理使用的密钥存储的供应商名称。 **类型**: 字符串 **示例** ：SunJCE **默认值** ：未指定
	trustStoreProvider	代理使用的信任存储的供应商名称。 **类型**: 字符串 **示例** ：SunJCE **默认值** ：未指定
	trustStoreType	代理使用的信任存储类型。 **类型**: 字符串 **示例** ：JCEKS **默认值** ：JKS

entry	sub-entry	描述和用法
	needClientAuth	指定代理是否通知客户端在接受者上需要双向 TLS。此属性覆盖 **wantClientAuth**。 **类型**: 布尔值 **示例**: true **默认值** ：未指定
	wantClientAuth	指定代理是否通知客户端是否在接受者上 *请求* 双向 TLS，但不需要。此属性被 **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通用名称(CN)与其主机名进行比较，以验证它们是否匹配。这个选项仅在使用双向 TLS 时适用。 **类型**: 布尔值 **示例**: true **默认值** ：未指定
	sslProvider	指定 SSL 供应商是 JDK 还是 OPENSSL。 **类型**: 字符串 **示例** ： OPENSSL **默认值** ： JDK

entry	sub-entry	描述和用法
	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	指定是否向 OpenShift Container Platform 之外的客户端公开接受者。 类型: 布尔值 **示例**: true **默认值**: false
	exposeMode	指定是否使用路由或入口公开接受者。默认情况下，接收器只使用路由公开。 类型 ：字符串 **示例** ： ingress **默认值** ：route 如果使用 ingress 公开连接器，则必须在 CR 中包含 **ingressHost** 或 **ingressDomain** 属性。

entry	sub-entry	描述和用法
	ingressHost	为 acceptor 公开的路由和入口指定自定义主机值。您可以为主机包含以下变量： * $(CR_NAME)- CR 中的 **metadata.name** 属性的值。 * $(CR_NAMESPACE)- 自定义资源的命名空间。 * $(BROKER_ORDINAL)- StatefulSet 分配给代理 pod 的普通号。 * $(ITEM_NAME)- 接受者的名称。 * $(RES_TYPE)- 资源类型。路由具有 **rte** 资源类型。入口的资源类型为 **ing**。 * $(INGRESS_DOMAIN)- 如果在 CR 中配置，**spec.ingressDomain** 属性的值。 **类型**: 字符串 **示例** ：my-acceptor-$(CR_NAME)-$(ITEM_NAME)-$(BROKER_ORDINAL).mydomain.com
	anycastPrefix	客户端用来指定应使用 **anycast** 路由类型的前缀。 **类型**: 字符串 **示例** ：jms.queue **默认值** ：未指定
	multicastPrefix	客户端用来指定应使用 **多播** 路由类型的前缀。 **类型**: 字符串 **示例** ： /topic/ **默认值** ：未指定

entry	sub-entry	描述和用法
	connectionsAllowed	接受者上允许的连接数。当达到这个限制时，会向日志发出 DEBUG 消息，连接被拒绝。使用的客户端类型决定了连接被拒绝时会发生什么。 **类型** ：整数 **示例** ： 2 **默认值** ：0 （无限连接）
	amqpMinLargeMessageSize	代理所需的最小消息大小（以字节为单位）以处理 AMQP 消息作为大消息。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消息目录中 (**/opt/*<custom_resource_name>*/data/large-messages**，默认为代理用于消息存储的持久性卷 (PV))。将值设为 **-1** 可禁用对 AMQP 消息的大型消息处理。 **类型** ：整数 **示例**: 204800 **默认值**: 102400 (100 KB)

entry	sub-entry	描述和用法
	BindToAllInterfaces	如果设置为 true，请使用 0.0.0.0 IP 地址而不是 pod 的内部 IP 地址配置代理 acceptors。当代理接受器具有 0.0.0.0 IP 地址时，它们绑定到为 pod 配置的所有接口，客户端可以使用 OpenShift Container Platform 端口转发将流量定向到代理。通常，您可以使用此配置调试服务。如需有关端口转发的更多信息，请参阅 OpenShift Container Platform 文档中的使用 端口转发访问容器中的应用程序。 **注意** 如果错误地使用了端口转发，则可能会给您的环境创建一个安全风险。在可能的情况下，不要在生产环境中使用端口转发。 **类型**: 布尔值 **示例**: true **默认值**: false
connectors.connector		单个连接器配置实例。

entry	sub-entry	描述和用法
	name*	连接器的名称。 **类型**: 字符串 **示例** ： my-connector **默认值** ：不适用
	type	要创建的连接器类型； **tcp** 或 **vm**。 **类型**: 字符串 **示例** ：vm **默认值** ：tcp
	host*	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。 **键入**: int **示例** ： 22222 **默认值** ：未指定
	sslEnabled	指定在连接器端口上是否启用了 SSL。如果设置为 **true**，请查找 TLS/SSL 所需的凭证的 **sslSecret** 中指定的 secret 名称。 **类型**: 布尔值 **示例**: true **默认值**: false

entry	sub-entry	描述和用法
	sslSecret	存储代理密钥存储、信任存储及其对应密码的 secret（所有 Base64 编码）。 如果没有为 **sslSecret** 指定自定义 secret 名称，连接器会假定默认 secret 名称。默认 secret 名称的格式是 <***custom_resource_name> - <connector_name>-*** **secret**。 您必须始终自己创建此 secret，即使连接器假设默认名称。 **类型**: 字符串 **示例** ： my-broker-deployment-my-connector-secret **默认值 :** *<custom_resource_name> - <connector_name>* - secret
	enabledCipherSuites	用于 TLS 通信的密码套件的逗号分隔列表。 **类型**: 字符串 **注意** ：对于连接器，**建议您不要** 指定密码套件列表。 **默认值** ：未指定
	keyStoreProvider	代理使用的密钥存储的供应商名称。 **类型**: 字符串 **示例** ：SunJCE **默认值** ：未指定

entry	sub-entry	描述和用法
	trustStoreProvider	代理使用的信任存储的供应商名称。 **类型**: 字符串 **示例** ：SunJCE **默认值** ：未指定
	trustStoreType	代理使用的信任存储类型。 **类型**: 字符串 **示例** ：JCEKS **默认值** ：JKS
	enabledProtocols	用于 TLS 通信的、以逗号分隔的协议列表。 **类型**: 字符串 **示例** ：TLSv1,TLSv1.1,TLSv1.2 **默认值** ：未指定
	needClientAuth	指定代理是否通知客户端在连接器中是否需要双向 TLS。此属性覆盖 **wantClientAuth**。 **类型**: 布尔值 **示例**: true **默认值** ：未指定
	wantClientAuth	指定代理是否通知客户端是否在连接器上 *请求* 双向 TLS，但不需要。此属性被 **needClientAuth** 覆盖。 **类型**: 布尔值 **示例**: true **默认值** ：未指定

entry	sub-entry	描述和用法
	verifyHost	指定是否将客户端证书的通用名称(CN)与主机名进行比较，以验证它们是否匹配。这个选项仅在使用双向 TLS 时适用。 **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
	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	指定是否将连接器公开给 OpenShift Container Platform 之外的客户端。 **类型**: 布尔值 **示例**: true **默认值**: false

entry	sub-entry	描述和用法
	exposeMode	指定是否使用路由或入口公开连接器。默认情况下，连接器只使用路由公开。 类型: 字符串 **示例** ： ingress **默认值** ：route 如果使用 ingress 公开连接器，则必须在 CR 中包含 **ingressHost** 或 **ingressDomain** 属性。

entry	sub-entry	描述和用法
	ingressHost	为连接器公开的路由和入口指定自定义主机值。您可以在 host 值中包含以下变量： * $(CR_NAME)- CR 中的 **metadata.name** 属性的值。 * $(CR_NAMESPACE)- 自定义资源的命名空间。 * $(BROKER_ORDINAL)- StatefulSet 分配给代理 pod 的普通号。 * $(ITEM_NAME)- 连接器的名称。 * $(RES_TYPE)- 资源类型。路由具有 **rte** 资源类型。入口的资源类型为 **ing**。 * $(INGRESS_DOMAIN)- 如果在 CR 中配置，**spec.ingressDomain** 属性的值。 类型: 字符串 **示例** ：my-connector-$(CR_NAME)-$(ITEM_NAME)-$(BROKER_ORDINAL).$(INGRESS_DOMAIN).mydomain.com
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 CR 中指定的地址设置，**以及与** 相同地址或一组地址匹配的默认配置： • 将默认配置中指定的任何属性值替换为

entry	sub-entry	描述和用法
		CR 中指定的值。 • 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括每个。 对于在 CR 中指定的地址设置，**或者**唯一匹配一个特定地址或一组地址的默认配置，将它们包括在最终合并的配置中。 **merge_replace** 对于 CR 中指定的地址设置 **和** 与相同地址集合匹配的默认配置，请在最终合并的配置中包含 **CR** 中指定的设置。**不要** 包括默认配置中指定的任何属性，即使 CR 中没有指定这些属性。+ 对于在 CR 中指定的地址设置，**或者**唯一匹配一个特定地址或一组地址的默认配置，将它们包括在最终合并的配置中。 **replace_all** 使用在 CR 中指定的内容替换默认配置中指定的**所有**地址设置最终的 megred 配置与 CR 中指定的配置完全相同。 **类型**: 字符串 **示例**: replace_all **默认值** ：merge_all
addressSettings.addressSetting		匹配地址 *或一组* 地址的地址设置。

entry	sub-entry	描述和用法
	addressFullPolicy	指定使用 **maxSizeBytes** 配置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传给磁盘。 **DROP** 发送到完整地址的消息会被静默丢弃。 **FAIL** 发送到完整地址的消息将被丢弃，消息制作者收到异常。 **BLOCK** 当消息制作者试图发送任何进一步消息时，会阻断。 BLOCK 策略仅适用于 AMQP、OpenWire 和核心协议，因为这些协议支持流控制。 **类型**: 字符串 **示例** ： DROP **默认值** ：PAGE
	autoCreateAddresses	指定代理在客户端向发送消息时自动创建地址，或者尝试从使用消息，一个绑定到不存在的地址的队列。 **类型**: 布尔值 **示例** ：false **默认值**: true

entry	sub-entry	描述和用法
	autoCreateDeadLetterResources	指定代理是否自动创建死信地址和队列来接收未发送的消息。 如果参数设为 **true**，则代理会自动创建死信地址和关联的死信队列。自动创建的地址名称与您为 **deadLetterAddress** 指定的值匹配。 **类型**: 布尔值 **示例**: true **默认值**: false
	autoCreateExpiryResources	指定代理是否自动创建地址和队列来接收过期的信息。 如果参数设为 **true**，则代理会自动创建到期地址和相关到期队列。自动创建的地址名称与您为 **expiryAddress** 指定的值匹配。 **类型**: 布尔值 **示例**: true **默认值**: false
	autoCreateJmsQueues	此属性已弃用。改为使用 **autoCreateQueues**。
	autoCreateJmsTopics	此属性已弃用。改为使用 **autoCreateQueues**。
	autoCreateQueues	指定代理在客户端向发送消息时自动创建队列，或者尝试从 中使用消息，即不存在的队列。 **类型**: 布尔值 **示例** ：false **默认值**: true

entry	sub-entry	描述和用法
	autoDeleteAddresses	指定代理是否在代理不再有任何队列时自动删除自动创建的地址。 **类型**: 布尔值 **示例** ：false **默认值**: true
	autoDeleteAddressDelay	时间（以毫秒为单位），代理会在地址没有队列时自动删除自动创建的地址前等待。 **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	指定代理在队列没有消费者且没有消息时自动删除自动创建的队列。 **类型**: 布尔值 **示例** ：false **默认值**: true
	autoDeleteCreatedQueues	指定代理在队列没有消费者且没有消息时自动删除手动创建的队列。 **类型**: 布尔值 **示例**: true **默认值**: false

entry	sub-entry	描述和用法
	autoDeleteQueuesDelay	时间（以毫秒为单位），代理会在队列没有消费者时自动删除自动创建队列前等待。 **类型** ：整数 **示例** ： 10 **默认值** ：0
	autoDeleteQueuesMessageCount	代理在代理评估是否自动删除前可以处于队列中的最大消息数。 **类型** ：整数 **示例** ：5 **默认值** ：0
	configDeleteAddresses	重新加载配置文件后，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** 重新加载配置文件时，代理不会删除地址。 **FORCE** 当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

entry	sub-entry	描述和用法
	configDeleteQueues	重新加载配置文件时，此设置指定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** 重新加载配置文件时，代理不会删除队列。 **FORCE** 当重新载入配置文件时，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	代理发送死地址（即 *未传输）消息*的地址。 类型: 字符串 **示例** ： DLA **默认值**: None
	deadLetterQueuePrefix	代理应用到自动创建的死信队列的名称的前缀。 类型: 字符串 **示例** ：myDLQ. **默认值** ：DLQ。
	deadLetterQueueSuffix	代理应用于自动创建的死信队列的后缀。 类型: 字符串 **示例** ：.DLQ **默认值**: None

entry	sub-entry	描述和用法
	defaultAddressRoutingType	在自动创建的地址上使用的路由类型。 **类型**: 字符串 **示例** ： ANYCAST **默认值** ：MULTICAST
	defaultConsumersBeforeDispatch	消息发送之前所需的消费者数量可以开始用于地址上的队列。 **类型** ：整数 **示例** ：5 **默认值** ：0
	defaultConsumerWindowSize	消费者的默认窗口大小（以字节为单位）。 **类型** ：整数 **示例** ： 300000 **默认值**: 1048576 (1024*1024)
	defaultDelayBeforeDispatch	如果没有为 **defaultConsumersBeforeDispatch** 的值指定，则代理会在分配消息前等待默认时间，以毫秒为单位。 **类型** ：整数 **示例** ：5 **默认值**: -1 （无延迟）
	defaultExclusiveQueue	指定地址上的所有队列是否默认为专用队列。 **类型**: 布尔值 **示例**: true **默认值**: false

entry	sub-entry	描述和用法
	defaultGroupBuckets	用于消息分组的存储桶数量。 **类型** ：整数 **示例** ： 0 （禁用消息分组） **默认值**: -1 （无限制）
	defaultGroupFirstKey	用于指示组中消息的使用者的关键。 **类型**: 字符串 **示例**: firstMessageKey **默认值**: None
	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**: false
	defaultGroupRebalancePauseDispatch	指定在代理重新平衡组时是否暂停消息发送。 **类型**: 布尔值 **示例**: true **默认值**: false
	defaultLastValueQueue	指定地址上的所有队列是否默认为最后值队列。 **类型**: 布尔值 **示例**: true **默认值**: false
	defaultLastValueKey	用于最后值队列的默认键。 **类型**: 字符串 **示例** ： stock_ticker **默认值**: None

entry	sub-entry	描述和用法
	defaultMaxConsumers	队列中允许的最大消费者数量。 **类型** ：整数 **示例** ：100 **默认值**: -1 （无限制）
	defaultNonDestructive	指定地址上的所有队列是否默认为非破坏性。 **类型**: 布尔值 **示例**: true **默认值**: false
	defaultPurgeOnNoConsumers	指定代理是否在没有消费者后清除队列的内容。 **类型**: 布尔值 **示例**: true **默认值**: false
	defaultQueueRoutingType	自动创建队列上使用的路由类型。默认值为 **MULTICAST**。 **类型**: 字符串 **示例** ： ANYCAST **默认值** ：MULTICAST
	defaultRingSize	没有明确设置环大小的匹配队列的默认环大小。 **类型** ：整数 **示例** ： 3 **默认值**: -1 （无大小限制）

entry	sub-entry	描述和用法
	enableMetrics	指定配置的指标插件，如 Prometheus 插件为匹配地址或一组地址收集指标。 **类型**: 布尔值 **示例** ：false **默认值**: true
	expiryAddress	接收过期消息的地址。 **类型**: 字符串 **示例** ： myExpiryAddress **默认值**: None
	expiryDelay	过期时间（以毫秒为单位）应用于使用默认过期时间的消息。 **类型** ：整数 **示例** ：100 **默认值** ：-1（没有应用过期时间）
	expiryQueuePrefix	代理应用到自动创建的过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过期队列的后缀。 **类型**: 字符串 **示例** ： .EXP **默认值**: None

entry	sub-entry	描述和用法
	lastValueQueue	指定队列是否只使用最后的值。 **类型**: 布尔值 **示例**: true **默认值**: false
	managementBrowsePageSize	指定管理资源可以浏览的消息数量。 **类型** ：整数 **示例** ：100 **默认值** ：200
	match*	将地址设置与代理中配置的地址匹配的字符串。您可以指定一个准确的地址名称，或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，您必须将该值放在单引号中，例如 **'myAddresses*'**。 **类型**: 字符串 **示例**: 'myAddresses*' **默认值**: None
	maxDeliveryAttempts	指定代理在将消息发送到配置的死信地址前尝试发送消息的次数。 **类型** ：整数 **示例** ：20 **默认值** ：10

entry	sub-entry	描述和用法
	maxExpiryDelay	过期时间（以毫秒为单位）应用于使用大于这个值的过期时间的信息。 **类型** ：整数 **示例** ：20 **默认值**: -1 （没有应用最大过期时间）
	maxRedeliveryDelay	代理发出的消息重新发送尝试之间的最大值（以毫秒为单位）。 **类型** ：整数 **示例** ：100 **默认值** ：**redeliveryDelay** 值的 10 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 ,**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**: -1 （无限制）
	maxSizeBytesRejectThreshold	地址在代理开始拒绝消息前可以访问的最大大小（以字节为单位）。当 **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** ：-1（无最大大小）

entry	sub-entry	描述和用法
	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）应用于使用低于这个值的消息。 **类型** ：整数 **示例** ：20 **默认值**: -1 （没有应用最小过期时间）
	pageMaxCacheSize	在内存中保留在内存中的页面文件数，以便在分页导航期间优化 I/O。 **类型** ：整数 **示例** ： 10 **默认值** ：5
	pageSizeBytes	分页大小（以字节为单位）。还支持字节表示法，如 **K**、**Mb** 和 **GB**。 **类型**: 字符串 **示例** ：20971520 **默认值** ： 10485760 （大约 10.5 MB）
	redeliveryDelay	时间（以毫秒为单位），代理会在重新设计取消消息前等待的时间。 **类型** ：整数 **示例** ：100 **默认值** ：0

entry	sub-entry	描述和用法
	redistributionDelay	以毫秒为单位，代理会在重新发送任何剩余的消息前在队列关闭时等待。 **类型** ：整数 **示例** ：100 **默认值** ：-1（未设置）
	retroactiveMessageCount	为在地址上创建未来的队列保留的消息数量。 **类型** ：整数 **示例** ：100 **默认值** ：0
	sendToDlaOnNoRoute	指定在无法路由到任何队列时是否将消息发送到配置的死信地址。 **类型**: 布尔值 **示例**: true **默认值**: false
	slowConsumerCheckPeriod	代理检查速度较慢的用户的频率(**以秒为单位**)。 **类型** ：整数 **示例** ：15 **默认值** ：5

entry	sub-entry	描述和用法
	slowConsumerPolicy	指定识别速度较慢的消费者时会发生什么。有效选项为 **KILL** 或 **NOTIFY**。**KILL** 终止消费者的连接，这会影响到使用该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** ： NOTIFY
	slowConsumerThreshold	在消费者考虑速度较慢前，消息消耗的最小消息率（每秒消息数）。 **类型** ：整数 **示例** ：100 **默认值** ：-1（未设置）
env	***<variable name>=<value>***	为代理设置环境变量。 **类型** ：数组 **示例** ： Name: TZ value: Europe/Vienna **默认值** ：不适用
brokerProperties		配置没有在代理的自定义资源定义(CRD)中公开的代理属性，否则无法在自定义资源(CR)中配置。
	<property name>=<value>	为代理配置的属性名称和值列表。 **类型**: 字符串 **示例** ： globalMaxSize=512m **默认值** ：不适用

entry	sub-entry	描述和用法
version		指定您希望 Operator 部署的 AMQ Broker 容器镜像版本。例如，如果您将 **version** 的值从 **7.11.1** 更改为 **7.12.0**，Operator 会将代理镜像升级到 7.12.0。 您可以从版本号省略 micro 和 minor 数字，以自动升级到可用于最新微版本或次版本的代理镜像。例如，如果您指定了 **7.11** 版本，Operator 会升级到最新的 7.1.x 版本的镜像。或者，如果您指定了 **7** 版本，Operator 会升级到最新的 7.x.x 版本的镜像。 **类型**: 字符串 **示例** ： 7.12.0 **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 实例允许您为部署中的代理定义地址和队列。下表详细介绍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)中需要标有星号(*)的配置项目。如果没有为非必需项目显式指定值，则配置将使用默认值。

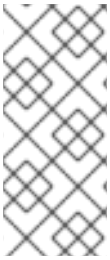
entry	描述和用法
addressName*	在代理上创建的地址名称。 **类型**: 字符串 **示例** ： address0 **默认值** ：未指定

entry	描述和用法
queueName	要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*	指定当您删除该部署的地址 CR 实例时，Operator 是否应该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**: false
routingType*	要使用的路由类型： **anycast** 或 **multicast**。 **类型**: 字符串 **示例**: anycast **默认值** ： multicast

8.1.3. 安全自定义资源配置参考

基于安全 CRD 的 CR 实例允许您为部署中的代理定义安全配置，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/zh-cn/red_hat_amq_broker/7.12/html-single/configuring_amq_broker/#assembly-br-securing-brokers_configuring

下表详细介绍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)中需要标有星号(*)的配置项目。如果没有为非必需项目显式指定值，则配置将使用默认值。

entry	sub-entry	描述和用法
loginModules		一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** - 对于没有登录凭据或其凭证失败的身份验证的用户，您可以使用客户机帐户授予代理的有限访问权限。 • **keycloakLoginModule**. - 允许您使用红帽单点登录保护代理。
propertiesLoginModule	name*	登录模块的名称。 **类型**: 字符串 **示例** ： my-login **默认值** ：不适用
	users.name*	用户名称。 **类型**: 字符串 **示例** ：jdoe **默认值** ：不适用

entry	sub-entry	描述和用法
	users.password*	用户的密码。 **类型**: 字符串 **示例** ：password **默认值** ：不适用
	users.roles	角色的名称。 **类型**: 字符串 **示例** ： viewer **默认值** ：不适用
guestLoginModule	name*	客户机登录模块的名称。 **类型**: 字符串 **示例** ：guest-login **默认值** ：不适用
	guestUser	客户机用户的名称。 **类型**: 字符串 **示例** ： myguest **默认值** ：不适用
	guestRole	guest 用户的角色名称。 **类型**: 字符串 **示例** ：guest **默认值** ：不适用
keycloakLoginModule	name	KeycloakLoginModule 的名称 **类型**: 字符串 **示例** ：sso **默认值** ：不适用

entry	sub-entry	描述和用法
	moduleType	KeycloakLoginModule 类型(directAccess 或 bearerToken) **类型**: 字符串 **示例**: bearerToken **默认值** ：不适用
	配置	以下配置项目与红帽单点登录相关，详细信息可通过 OpenID Connect 文档获取。
	configuration.realm*	KeycloakLoginModule 的域 **类型**: 字符串 **示例** ： myrealm **默认值** ：不适用
	configuration.realmPublicKey	域的公钥 **类型**: 字符串 **默认值** ：不适用
	configuration.authServerUrl*	keycloak 身份验证服务器的 URL **类型**: 字符串 **默认值** ：不适用
	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 应用程序的客户端 ID。每个应用都有一个客户端 ID，用于识别应用。
	configuration.publicClient	指定它是公共客户端。 **类型**: 布尔值 **默认值**: false **示例** ：false

entry	sub-entry	描述和用法
	configuration.credentials.key	指定凭证密钥。 类型: 字符串 默认值 ：不适用 类型: 字符串 默认值 ：不适用
	configuration.credentials.value	指定凭证值 类型: 字符串 默认值 ：不适用
	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 Cross-Origin Resource Sharing (CORS) 它将处理 CORS preflight 请求。它还将查看访问令牌来确定有效的来源。 类型: 布尔值 默认值: false
	configuration.corsMaxAge	CORS 最长期限 如果启用了 CORS，这将设置 Access-Control-Max-Age 标头的值。
	configuration.corsAllowedMethods	CORS 允许的方法 如果启用了 CORS，这将设置 Access-Control-Allow-Methods 标头的值。这应该是一个用逗号分开的字符串。
	configuration.corsAllowedHeaders	CORS 允许的标头 如果启用了 CORS，这将设置 Access-Control-Allow-Headers 标头的值。这应该是一个用逗号分开的字符串。

entry	sub-entry	描述和用法
	configuration.corsExposedHeaders	CORS 公开的标头 如果启用了 CORS，这将设置 Access-Control-Expose-Headers 标头的值。这应该是一个用逗号分开的字符串。
	configuration.exposeToken	指定是否公开访问令牌 **类型**: 布尔值 **默认值**: false
	configuration.bearerOnly	指定是否验证 bearer 令牌 **类型**: 布尔值 **默认值**: false
	configuration.autoDetectBearerOnly	指定是否只自动探测 bearer 令牌 **类型**: 布尔值 **默认值**: 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**: false
	configuration.disableTrustManager	指定是否禁用信任管理器 **类型**: 布尔值 **默认值**: 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。
	configuration.trustStorePassword*	truststore 密码 如果设置了 truststore 且信任存储需要密码，则这是 REQUIRED。

entry	sub-entry	描述和用法
	configuration.clientKeyStore	客户端密钥存储的路径 类型: 字符串 默认值 ：不适用
	configuration.clientKeyStorePassword	客户端密钥存储密码 类型: 字符串 默认值 ：不适用
	configuration.clientKeyPassword	客户端密钥密码 类型: 字符串 默认值 ：不适用
	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 ：不适用
	configuration.tokenStore	令牌存储类型（会话或 Cookie） 类型: 字符串 默认值 ：不适用
	configuration.tokenCookiePath	Cookie 存储的 Cookie 路径 类型: 字符串 默认值 ：不适用

entry	sub-entry	描述和用法
	configuration.principalAttribute	使用 OpenID Connect ID Token 属性填充 UserPrincipal 名称 使用 OpenID Connect ID Token 属性填充 UserPrincipal 名称。如果 token 属性为空，则默认为 sub。可能的值有 sub, preferred_username, email, name, nickname, given_name, family_name。
	configuration.proxyUrl	代理 URL
	configuration.turnOffChangeSessionIdOnLogin	指定是否成功登录时更改会话 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	是否为 bearer 令牌处理关闭对 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 audience **类型**: 布尔值 **示例** ：false

entry	sub-entry	描述和用法
	configuration.enableBasicAuth	是否支持基本身份验证 **类型**: 布尔值 **默认值**: false
	configuration.confidentialPort	Keycloak 服务器用来通过 SSL/TLS 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** ：不适用
	configuration.redirectRewriteRules.value	替换字符串 **类型**: 字符串 **默认值** ：不适用
	configuration.scope	DirectAccessGrantsLoginModule 的 OAuth2 scope 参数 **类型**: 字符串 **默认值** ：不适用
securityDomains		代理安全域
	brokerDomain.name	代理域名 **类型**: 字符串 **示例** ：activemq **默认值** ：不适用
	brokerDomain.loginModules	一个或多个登录模块。每个条目之前必须在上面的 **loginModules** 部分中定义。

entry	sub-entry	描述和用法
	brokerDomain.loginModules.name	登录模块的名称 类型: 字符串 示例 ：prop-module 默认值 ：不适用
	brokerDomain.loginModules.flag	与 propertiesLoginModule 相同，必需、requisite、sufficient 和 optional 是有效的值。 类型: 字符串 示例 ： sufficient 默认值 ：不适用
	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 ：不适用
	consoleDomain.loginModules	单个登录模块配置。
	consoleDomain.loginModules.name	登录模块的名称 类型: 字符串 示例 ：prop-module 默认值 ：不适用
	consoleDomain.loginModules.flag	与 propertiesLoginModule 相同，必需、requisite、sufficient 和 optional 是有效的值。 类型: 字符串 示例 ： sufficient 默认值 ：不适用

entry	sub-entry	描述和用法
	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**: createAddress **默认值** ：不适用
	broker.permissions.roles	安全设置应用于这些角色，如 设置权限 中所述。 **类型**: 字符串 **示例** ：root **默认值** ：不适用
securitySettings.management		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** ：不适用

entry	sub-entry	描述和用法
	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 ：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 连接是否安全。 类型: 布尔值 示例: true 默认值: false
	connector.keyStoreProvider	管理连接器的密钥存储供应商。如果您设置了 connector.secured="true"，则需要此项。默认值为 JKS。

entry	sub-entry	描述和用法
	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"，则需要管理连接器的 truststore 供应商。 **类型** ：字符串 **示例** ：JKS **默认** ：JKS
	connector.trustStorePath	管理连接器的信任存储的位置。如果您设置了 connector.secured="true"，则需要此项。 **类型**: 字符串 **默认值** ：不适用
	connector.trustStorePassword	管理连接器的 truststore 密码。如果您设置了 connector.secured="true"，则需要此项。 **类型**: 字符串 **默认值** ：不适用
	connector.passwordCodec	管理连接器的密码 codec 是要使用的密码 codec 的完全限定类名称，如 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** ：不适用
	authorisation.allowedList.key	allowedList 的密钥 **类型**: 字符串 **默认值** ：不适用

entry	sub-entry	描述和用法
	authorisation.defaultAccess.method	defaultAccess List 的方法 类型: 字符串 默认值 ：不适用
	authorisation.defaultAccess.roles	defaultAccess List 的角色 类型: 字符串 默认值 ：不适用
	authorisation.roleAccess.domain	roleAccess List 的域 类型: 字符串 默认值 ：不适用
	authorisation.roleAccess.key	roleAccess List 的键 类型: 字符串 默认值 ：不适用
	authorisation.roleAccess.accessList.method	roleAccess List 的方法 类型: 字符串 默认值 ：不适用
	authorisation.roleAccess.accessList.roles	roleAccess List 的角色 类型: 字符串 默认值 ：不适用
	applyToCrNames	将此安全配置应用到当前命名空间中命名 CR 定义的代理。值为 * 或空字符串表示适用于所有代理。 类型: 字符串 示例 ： my-broker 默认值 ：当前命名空间中的 CR 定义的所有代理。

8.2. JAAS 登录模块配置示例

以下示例显示了配置了属性登录模块和 LDAP 登录模块的 JAAS 登录模块配置。properties 登录模块引用包含 Operator 用来与代理进行身份验证的凭证的默认登录模块。

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
    debug=true
    initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
    connectionURL="LDAP://localhost:389"
    connectionUsername="CN=Administrator,CN=Users,OU=System,DC=example,DC=com"
    connectionPassword=redhat.123
    connectionProtocol=s
    connectionTimeout="5000"
    authentication=simple
    userBase="dc=example,dc=com"
    userSearchMatching="(CN={0})"
    userSearchSubtree=true
    readTimeout="5000"
    roleBase="dc=example,dc=com"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=true;

  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule
   reload=true
   org.apache.activemq.jaas.properties.user="artemis-users.properties"
   org.apache.activemq.jaas.properties.role="artemis-roles.properties"
   baseDir="/home/jboss/amq-broker/etc";
};
```

以下示例显示了一个 JAAS 登录模块配置，它在单独的域中有两个属性登录模块。

- 默认属性登录模块位于名为 console 的域中，并且具有 Operator 和 AMQ 管理控制台用来与代理进行身份验证的属性文件。
- activemq 域中的登录模块具有新的属性文件，例如，这些文件可以包含用于验证消息传递用户的凭据。

您可能想要创建单独的域，例如，将特定的安全控制应用到包含 Operator 用来与代理进行身份验证的登录模块的域。

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule
    reload=true
  org.apache.activemq.jaas.properties.user="new-users.properties"
  org.apache.activemq.jaas.properties.role="new-roles.properties"
};

console {
org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule
```

```
  reload=true
   org.apache.activemq.jaas.properties.user="artemis-users.properties"
  org.apache.activemq.jaas.properties.role="artemis-roles.properties"
  baseDir="/home/jboss/amq-broker/etc";
};
```

注意

默认情况下，AMQ Management Console 使用 activemq realm 中的默认属性登录模块进行身份验证。如果在另一个域中配置了默认属性登录模块，如示例所示，您必须在 broker CR 中设置环境变量，将 AMQ 管理控制台配置为使用该域。例如：

```
spec:
  ...
  env:
  - name: JAVA_ARGS_APPEND
    value: --Hawtio.realm=console
  ...
```

有关在 CR 中设置环境变量的更多信息，请参阅 第 4.9 节 “为代理容器设置环境变量”。

8.3. 示例：将 AMQ BROKER 配置为使用 RED HAT SINGLE SIGN-ON

本例演示了如何将 AMQ Broker 配置为使用 Red Hat Single Sign-On 来使用 JAAS 登录模块来身份验证和授权。

先决条件

- 与 LDAP 目录集成的 Red Hat Single Sign-On 实例。
  - LDAP 目录填充 AMQ Broker 的用户和角色信息。
  - Red Hat Single Sign-On 配置为联合来自 LDAP 服务器的用户。
  - Red Hat Single Sign-On 被配置为使用 role-ldap-mapper 将角色信息从 LDAP 映射到红帽单点登录。
- 一个 Red Hat Single Sign-On 域，它有：

- 使用以下设置配置的应用程序（如 AMQ 管理控制台）的客户端，可以使用 oAuth 协议获取令牌：

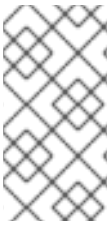
  身份验证流程：标准流

  有效的 Redirect URI：AMQ Management Console 的 OpenShift Container Platform 路由。例如 ：http://artemis-wconsj-0-svc-rte-kc-ldap-tests-0eae49.apps.redhat-412t.broker.app-services-dev.net/console/*

- 如果您有无法使用 oAuth 协议获取令牌的消息客户端应用程序，使用以下设置配置单独的客户端：

  身份验证流程：直接访问授予

  有效的 Redirect URI: *



注意

Red Hat Single Sign-On 中的每个域都包括一个名为 Broker 的客户端。这个客户端与 AMQ Broker 无关。

流程

1. 创建名为 login.config 的文本文件，并添加 JAAS 登录模块配置，以将 AMQ Broker 与 Red Hat Single Sign-On 连接。例如：

```
console {
   // ensure the operator can connect to the broker by referencing the existing properties
config
   org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
      org.apache.activemq.jaas.properties.user="artemis-users.properties"
      org.apache.activemq.jaas.properties.role="artemis-roles.properties"
      baseDir="/home/jboss/amq-broker/etc";

  org.keycloak.adapters.jaas.BearerTokenLoginModule sufficient
      keycloak-config-file="/amq/extra/secrets/sso-jaas-config/_keycloak-bearer-token.json"
      role-principal-class=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal;
};
activemq {
   org.keycloak.adapters.jaas.BearerTokenLoginModule sufficient
      keycloak-config-file="/amq/extra/secrets/sso-jaas-config/_keycloak-bearer-token.json"
```

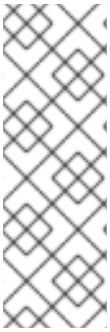
```
        role-principal-class=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal;

    org.keycloak.adapters.jaas.DirectAccessGrantsLoginModule sufficient
        keycloak-config-file="/amq/extra/secrets/sso-jaas-config/_keycloak-direct-access.json"
        role-principal-class=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal;

    org.apache.activemq.artemis.spi.core.security.jaas.PrincipalConversionLoginModule
required
        principalClassList=org.keycloak.KeycloakPrincipal;
};
```

注意

- **.json 配置文件的路径必须采用 /amq/extra/secrets/*name*-jaas-config 的格式。对于 *名称*，请指定字符串值。您必须使用相同的字符串值和 -jaas-config 后缀来命名稍后在此流程中创建的 secret。**
- **在示例 login.config 文件中，名为 console 的域用于验证 AMQ 管理控制台用户以及名为 activemq 的域来验证消息传递客户端。**

以下登录模块在示例 login.config 文件中配置。

登录模块	描述和用法
org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule	这是默认登录模块，包含 **artemis-users.properties** 文件，其中包含 Operator 向代理进行身份验证的默认用户。
org.keycloak.adapters.jaas.BearerTokenLoginModule	此登录模块用于应用程序，如 AMQ 管理控制台，可以使用 oAuth 协议获取令牌。当用户在浏览器窗口中打开 AMQ 管理控制台时，它们会被重定向到 Red Hat Single Sign-On 控制台，以登录来获取 bearer 令牌。
org.keycloak.adapters.jaas.DirectAccessGrantsLoginModule	非 HTTP 应用程序（如消息传递客户端）需要这个登录模块，它们无法使用 oAuth 协议。使用这个登录模块，代理首先使用 Red Hat Single Sign-On 中配置的 secret 验证客户端，然后代表客户端获取令牌。
org.apache.activemq.artemis.spi.core.security.jaas.PrincipalConversionLoginModule	需要此登录模块才能将接收的 Keycloak 主体转换为 AMQ Broker 使用的 JAAS 主体。



注意

在 login.config 文件示例中，每个 .json 属性文件名都有一个下划线前缀。当报告 JaasPropertiesApplied 条件的状态时，Operator 会忽略前缀为下划线的文件。如果文件名没有下划线前缀，JaasPropertiesApplied 条件的状态会永久显示 OutofSync，因为代理无法识别第三方登录模块使用的属性文件。有关状态报告的更多信息，请参阅 第 4.3.2.1 节 “使用安全自定义资源(CR)配置默认的 JAAS 登录模块”。

1. 为登录模块中引用的每个 .json 属性文件创建文本文件，并将 AMQ Broker 连接到 Red Hat Single Sign-On 所需的详细信息。例如：

_keycloak-bearer-token.json

```
{
   "realm": "amq-broker-ldap",
   "resource": "amq-console",
   "auth-server-url": "https://keycloak-svc-rte-kc-ldap-tests-0eae49.apps.412t.broker.app-services-dev.net",
   "principal-attribute": "preferred_username",
   "use-resource-role-mappings": false,
   "ssl-required": "external",
   "confidential-port": 0
}
```

_keycloak-direct-access.json

```
{
   "realm": "amq-broker-ldap",
   "resource": "amq-broker",
   "auth-server-url": "https://keycloak-svc-rte-kc-ldap-tests-0eae49.apps.412t.broker.app-services-dev.net",
   "principal-attribute": "preferred_username",
   "use-resource-role-mappings": false,
   "ssl-required": "external",
   "credentials": {
      "secret": "Lfk6g1ZKlGzNT6eRkz0d1scM4M29Ohmn"
   }
}
```

realm

配置为在 Red Hat Single Sign-On 中验证 AMQ Broker 应用程序和服务的域。

resource

在 Red Hat Single Sign-On 中配置的客户端的客户端 ID。

auth-server-url

Red Hat Single Sign-On 服务器的基本 URL。

principal-attribute

用于填充 UserPrincipal 名称的 token 属性。

use-resource-role-mappings

如果设置为 true，Red Hat Single Sign-On 会在令牌中查找用户的应用程序级别角色映射。如果为 false，它会查看用户角色映射的域级别。默认值为 false。

ssl-required

确保与 Red Hat Single Sign-On 服务器与和来自 Red Hat Single Sign-On 服务器的所有通信都是通过 HTTPS。默认值为 外部，这意味着外部请求默认需要 HTTPS。

credentials

在 Red Hat Single Sign-On 中配置的 secret，代理用来登录到 Red Hat Single Sign-On，并代表客户端获取令牌。

2. 创建名为 _keycloak-js-client.json 的文本文件，并添加 AMQ 管理控制台所需的配置，将用户重定向到 Red Hat Single Sign-On Admin Console 的 URL，其中输入其凭证。例如：

```
{
  "realm": "amq-broker-ldap",
  "clientId": "amq-console",
  "url": "https://keycloak-svc-rte-kc-ldap-tests-0eae49.apps.412t.broker.app-services-dev.net"
}
```

3. 使用 oc create secret 命令创建包含登录模块配置中引用的文件的 secret。例如：

```
oc create secret generic sso-jaas-config --from-file=login.config --from-file=artemis-
users.properties --from-file=artemis-roles.properties --from-file=_keycloak-bearer-token.json
--from-file=_keycloak-direct-access.json --from-file=_keycloak-js-client.json
```

注意

- **secret 名称必须具有 -jaas-config 后缀，以便 Operator 可以识别 secret 包含登录模块配置，并将任何更新传播到每个代理 Pod。**
- **secret 名称必须与您在 login.config 文件中指定的 .json 配置文件路径中的最后一个目录名称匹配。例如，如果配置文件的路径为 /amq/extra/secrets/sso-jaas-config，您必须指定一个 secret 名称 sso-jaas-config。**

有关如何创建 secret 的更多信息，请参阅 Kubernetes 文档中的 Secret。

4. **将您创建的 secret 添加到代理部署的 ActiveMQArtemis 自定义资源(CR)实例中。**

   a. **使用 OpenShift 命令行界面：**

      i. **以具有特权的用户身份登录 OpenShift，以便在代理部署的项目中部署 CR。**

      ii. **编辑部署的 CR。**

      ```
      oc edit ActiveMQArtemis <CR instance name> -n <namespace>
      ```

   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，以便在代理部署的项目中部署 CR。**

      ii. **在左侧窗格中，点 Operators → Installed Operator。**

      iii. **点 Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch) operator。**

      iv. **点 AMQ Broker 选项卡。**

v. 单击 ActiveMQArtemis 实例名称的名称。

vi. 点 YAML 标签。

在控制台中，会打开 YAML 编辑器，供您配置 CR 实例。

5. 创建一个 extraMounts 属性和 secrets 属性，并添加 secret 的名称。以下示例将名为 custom-jaas-config 的 secret 添加到 CR 中。

```
deploymentPlan:
  ...
  extraMounts:
    secrets:
    - "sso-jaas-config"
  ...
```

6. 在 ActiveMQArtemis CR 中，创建一个环境变量，其中包含 AMQ 管理控制台使用 Red Hat Single Sign-On 进行身份验证所需的 hawtio 设置。当托管代理的 JVM 时，环境变量的内容作为参数传递到 Java 应用程序启动程序。例如：

```
env:
- name: JAVA_ARGS_APPEND
  value: -
Dhawtio.rolePrincipalClasses=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal
      -Dhawtio.keycloakEnabled=true -
Dhawtio.keycloakClientConfig=/amq/extra/secrets/sso-jaas-config/_keycloak-js-client.json
      -Dhawtio.authenticationEnabled=true -Dhawtio.realm=console
```

有关 hawtio 设置的更多信息，请参阅 hawtio 文档。

7. 在 ActiveMQArtemis CR 的 spec 部分中，添加一个 brokerProperties 属性，并为 LDAP 目录中配置的角色添加权限。您可以为单个地址授予角色权限。或者，您可以使用 # 符号指定通配符匹配，为所有地址授予角色权限。例如：

```
spec:
  ...
  brokerProperties:
  - securityRoles.#.producers.send=true
  - securityRoles.#.consumers.consume=true
  ...
```

8. **保存 CR。**

Operator 在每个 Pod 上的 /amq/extra/secrets/*secret 名称目录中挂载 secret* 中的文件，并将代理 JVM 配置为读取挂载的 login.config 文件，该文件包含 SSO 配置，而不是默认的 login.config 文件。

8.4. 日志记录

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。

流程

- **在命令行中运行以下命令：**

```
$ oc logs -f <pass:quotes[<pod-name>]> <pass:quotes[<container-name>]>
```

更新于 2024-05-21