Red Hat AMQ 2021.Q3

OpenShift での AMQ Broker のデプロイ

AMQ Broker 7.9での使用について

Last Updated: 2023-02-23

Red Hat AMQ 2021.Q3 OpenShift での AMQ Broker のデプロイ

AMQ Broker 7.9での使用について

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

法律上の通知

概要

OpenShift Container Platform に AMQ Broker をインストールし、デプロイする方法を説明します。

目次

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージをご覧ください。

第1章 OPENSHIFT CONTAINER PLATFORM での AMQ BROKER について

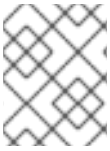
Red Hat AMQ Broker 7.9 は、OpenShift Container Platform(OCP)4.6、4.7、4.8、または 4.9 で使用するためのコンテナー化されたイメージとして利用できます。

AMQ Broker は Apache ActiveMQ Artemis をベースにしています。JMS に準拠するメッセージブローカーを提供します。初期ブローカー Pod を設定した後に、OpenShift Container Platform 機能を使用して重複を迅速にデプロイできます。

1.1. バージョンの互換性とサポート

OpenShift Container Platform イメージのバージョンの互換性についての詳細は、以下を参照してください。

- OpenShift Container Platform 4.x のテスト済みインテグレーション



注記

OpenShift Container Platform での AMQ Broker のすべてのデプロイメントで、RHEL 8 ベースのイメージが使用されるようになりました。

1.2. サポートされない特性

- マスタースレーブベースの高可用性
  マスターとスレーブのペアを設定して実現する高可用性 (HA) はサポートされません。その代わりに、Pod がスケールダウンされると、スケールダウンコントローラーを使用して HA が OpenShift で提供され、メッセージの移行が可能になります。

  OpenShift プロキシーまたはバインドポートを使用して、ブローカーのクラスターに接続する外部クライアントを HA に適切に設定しなければならない場合があります。クラスター化されたシナリオでは、ブローカーによって、ブローカーのホストとポート情報のすべてのアドレスについて特定のクライアントに通知します。これらは内部でのみアクセスできるため、一部のクライアント機能は機能しないか、または無効にする必要があります。

クライアント	設定
Core JMS クライアント	外部 Core Protocol JMS クライアントは HA またはいずれのフェイルオーバーもサポートしないため、接続ファクトリーは **useTopologyForLoadBalancing=false** で設定する必要があります。
AMQP クライアント	AMQP クライアントがフェイルオーバーリストをサポートしません。

- クラスター内の永続サブスクリプション
  永続サブスクリプションが作成されると、これはクライアントが接続したブローカーの永続キューとして表されます。クラスターが OpenShift 内で実行されている場合、クライアントが永続サブスクリプションキューが作成されたブローカーを認識しません。サブスクリプションが永続的であり、クライアントが再接続する方法は現在、ロードバランサーが同じノードに再

接続する方法はありません。このような場合には、クライアントが別のブローカーに接続し、重複したサブスクリプションキューを作成できます。このため、ブローカーのクラスターで永続サブスクリプションを使用することは推奨されていません。

1.3. 本書の表記慣例

本書では、**sudo** コマンド、ファイルパス、および置き換え可能な値について、以下の規則を使用します。

sudo コマンド
本書では、root 権限を必要とするすべてのコマンドに対して **sudo** が使用されています。何らかの変更がシステム全体に影響を与える可能性があるため、**sudo** を使用する場合は、常に注意が必要です。

sudo の使用の詳細は、「**sudo** コマンド」を参照してください。

本書におけるファイルパスの使用
本書では、すべてのファイルパスは Linux、UNIX、および同様のオペレーティングシステムで有効です (例: **/home/...**)。Microsoft Windows を使用している場合は、同等の Microsoft Windows パスを使用する必要があります (例: **C:\Users\...**)。

交換可能な値
このドキュメントでは、お客様の環境に合わせた値に置き換える必要のある置換可能な値を使用している場合があります。置き換え可能な値は小文字で、角括弧(**< >**)で囲まれ、イタリックおよび**monospace**フォントを使用してスタイルされます。単語が複数になる場合は、アンダースコア(_)で区切ります。

たとえば、次のコマンドで、 **<project_name>**を独自のプロジェクト名に置き換えます。

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

第2章 OPENSHIFT CONTAINER PLATFORM での AMQ BROKER のデプロイメントのプランニング

このセクションでは、Operator ベースのデプロイメントを計画する方法について説明します。

Operator は、OpenShift アプリケーションのパッケージ化、デプロイ、および管理を可能にするプログラムです。多くの場合、Operator は共通タスクまたは複雑なタスクを自動化します。通常、Operator は以下を提供することを目的としています。

- 一貫性のある繰り返し可能なインストール
- システムコンポーネントのヘルスチェック
- OTA (Over-the-air) 更新
- 管理アップグレード

Operator は、デプロイメントの設定に使用したカスタムリソース (CR) インスタンスへの変更を常にリッスンしているため、ブローカーインスタンスの実行中に変更を加えることができます。CR に変更を加えると、Operator は既存のブローカーデプロイメントの変更を調整し、変更を反映するためにデプロイメントを更新します。さらに、Operator は、メッセージングデータの整合性を維持するメッセージ移行機能を提供します。デプロイメントの失敗または意図的に縮小してクラスターデプロイメントのブローカーがシャットダウンすると、この機能はメッセージを同じブローカークラスターで実行されているブローカー Pod に移行します。

2.1. AMQ BROKER OPERATOR カスタムリソース定義の概要

通常、カスタムリソース定義 (CRD) は、Operator でデプロイされたカスタム OpenShift オブジェクトのスキーマです。対応するカスタムリソース (CR) インスタンスを作成すると、CRD の設定項目の値を指定できます。Operator 開発者の場合、CRD を使用して公開する内容は基本的に、デプロイされたオブジェクトの設定および使用方法のために API になります。CRD は Kubernetes 経由で自動的に公開されるため、通常の HTTP **curl** コマンドを使用して CRD に直接アクセスできます。

OperatorHub グラフィカルインターフェースを使用して、OpenShift コマンドラインインターフェース (CLI) または Operator Lifecycle Manager を使用して AMQ Broker Operator をインストールできます。いずれの場合も、AMQ Broker Operator に以下で説明されている CRD が含まれます。

メインブローカー CRD

この CRD に基づいて CR インスタンスをデプロイし、ブローカーデプロイメントを作成および設定します。
Operator のインストール方法に基づいて、この CRD は以下になります。

- Operator インストールアーカイブの **crds** ディレクトリーにある **broker_activemqartemis_crd** ファイル (OpenShift CLI インストール方法)
- OpenShift Container Platform Web コンソールの**Custom Resource Definitions** (OperatorHub インストール方法) の **ActiveMQArtemis CRD**

Address CRD

この CRD に基づいて CR インスタンスをデプロイし、ブローカーデプロイメントのアドレスおよびキューを作成します。
Operator のインストール方法に基づいて、この CRD は以下になります。

- Operator インストールアーカイブの **crds** ディレクトリーにある **broker_activemqartemisaddress_crd** ファイル (OpenShift CLI インストール方法)
- OpenShift Container Platform Web コンソールの **Custom Resource Definitions** セクションの **ActiveMQArtemisAddresss** CRD (OperatorHub インストール方法)

セキュリティー CRD

この CRD に基づいて CR インスタンスをデプロイし、ユーザーを作成してそのユーザーをセキュリティーコンテキストに関連付けます。
Operator のインストール方法に基づいて、この CRD は以下になります。

- Operator インストールアーカイブの **crds** ディレクトリーにある **broker_activemqartemissecurity_crd** ファイル (OpenShift CLI インストール方法)
- OpenShift Container Platform Web コンソールの **Custom Resource Definitions** セクションの **ActiveMQArtemisSecurity** CRD (OperatorHub インストール方法)

Scaledown CRD

Operator は、メッセージ移行用にスケールダウンコントローラーをインスタンス化する際に、この CRD に基づいて CR インスタンスを自動的に作成します。
Operator のインストール方法に基づいて、この CRD は以下になります。

- Operator インストールアーカイブの **crds** ディレクトリーにある **broker_activemqartemisscaledown_crd** ファイル (OpenShift CLI インストール方法)
- OpenShift Container Platform Web コンソールの **Custom Resource Definitions** セクションの **ActiveMQArtemisScaledown** CRD (OperatorHub インストール方法)

関連情報

- 以下を使用して AMQ Broker Operator (および含まれるすべての CRD) のインストール方法については、以下を実行します。
  - OpenShift CLI については、「CLI を使用した Operator のインストール」 を参照してください。
  - Operator Lifecycle Manager および OperatorHub グラフィカルインターフェースについては、「OperatorHub を使用した Operator のインストール」 を参照してください。
- メインブローカーおよび CRD に基づいて CR インスタンスの作成時に使用する完全な設定の参照については、以下を参照してください。
  - 「ブローカーカスタムリソース設定リファレンス」
  - 「アドレスのカスタムリソースの設定リファレンス」

2.2. AMQ BROKER OPERATOR サンプルカスタムリソースの概要

インストール時にダウンロードして展開する AMQ Broker Operator アーカイブには、**deploy/crs** ディレクトリーにサンプルカスタムリソース (CR) ファイルが含まれます。以下のサンプル CR ファイルでは、以下が可能になります。

- SSL またはクラスタリングなしで最小ブローカーをデプロイします。

- アドレスを定義します。

ダウンロードするブローカー Operator アーカイブには、以下に一覧表示されているように **deploy/examples** ディレクトリーにデプロイメントの CR が含まれます。

artemis-basic-deployment.yaml
基本ブローカーデプロイメント。

artemis-persistence-deployment.yaml
永続ストレージのあるブローカーデプロイメント。

artemis-cluster-deployment.yaml
クラスター化したブローカーのデプロイメント。

artemis-persistence-cluster-deployment.yaml
永続ストレージのあるクラスターブローカーのデプロイメント。

artemis-ssl-deployment.yaml
SSL セキュリティーを使用したブローカーデプロイメント。

artemis-ssl-persistence-deployment.yaml
SSL セキュリティーおよび永続ストレージを使用したブローカーデプロイメント。

artemis-aio-journal.yaml
ブローカージャーナルで非同期 I/O (AIO) の使用。

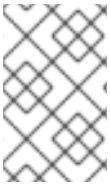
address-queue-create.yaml
アドレスおよびキューの作成。

2.3. CLUSTER OPERATOR デプロイメントの監視オプション

Cluster Operator の実行中に、AMQ Broker カスタムリソース(CR)の更新の**監視**が開始されます。

Cluster Operator をデプロイして、以下の CR を監視するように選択できます。

- 単一の namespace (Operator が含まれる同じ namespace)
- すべての namespace

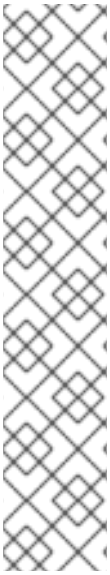


注記

以前のバージョンの AMQBroker Operator をクラスターの名前空間にすでにインストールしている場合には、競合を回避するために、AMQ Broker Operator 7.9 バージョンをインストールしてその名前空間の監視は行わないことを推奨します。

2.4. OPERATOR によるコンテナーイメージの選択方法

少なくとも バージョン 7.9.3-opr-3 の Operator に基づいてブローカーデプロイメントのカスタムリソース(CR)インスタンスを作成する場合、CR でブローカーまた **は Init コンテナーイメージ名を明示的に指定する必要はありません**。デフォルトで、CR をデプロイし、コンテナーイメージの値を明示的に指定しない場合、Operator は使用する適切なコンテナーイメージを自動的に選択します。



注記

OpenShift コマンドラインインターフェースを使用して Operator をインストールする場合、Operator インストールアーカイブには **broker_activemqartemis_cr.yaml** というサンプル CR ファイルが含まれます。サンプル CR では、**spec.deploymentPlan.image** プロパティーが含まれ、**placeholder** のデフォルト値に設定されます。この値は、Operator が CR をデプロイするまでブローカーコンテナーイメージを選択しないことを示します。

Init コンテナーイメージを指定する **spec.deploymentPlan.initImage** プロパティーは、**broker_activemqartemis_cr.yaml** サンプル CR ファイルには**含まれません**。**spec.deploymentPlan.initImage** プロパティーを明示的に CR に追加し、値を指定する場合、Operator は CR のデプロイ時に使用する適切な組み込み Init コンテナーイメージを選択します。

このセクションでは、Operator がこのイメージを選択する仕組みについて説明します。

ブローカーおよび init コンテナーイメージを選択するには、Operator はまず、イメージが対応する AMQ Broker バージョンを判別します。Operator は以下のようにバージョンを判別します。

- メイン CR の **spec.upgrades.enabled** プロパティーがすでに **true** に設定され、**spec.version** プロパティーが **7.7.0**、**7.8.0**、**7.8.1**、または **7.8.2** を指定し、Operator はその指定されたバージョンを使用します。
- **spec.upgrades.enabled** が **true** に設定されて **いない** 場合や、**spec.version** が 7 **.7.0** よりも前の AMQ Broker バージョンに設定されている場合、Operator は **最新** バージョンの AMQ Broker(7. **9.3**)を使用します。

その後、Operator はコンテナープラットフォームを検出します。AMQ Broker Operator は以下のコンテナープラットフォームで実行できます。

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

AMQ Broker およびコンテナープラットフォームのバージョンに基づいて、Operator は **operator.yaml** 設定ファイルで環境変数の 2 セットを参照します。以下のサブセクションで説明されているように、環境変数のセットは、さまざまなバージョンの AMQ Broker のブローカーおよび Init コンテナーイメージを指定します。

2.4.1. ブローカーコンテナーイメージの環境変数

ブローカーコンテナーイメージの **operator.yaml** 設定ファイルに含まれる環境変数には、以下の命名規則があります。

OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>

IBM Z での OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>_s390x

OpenShift Container Platform on IBM Power Systems

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version_identifier>_ppc64le

サポート対象のコンテナープラットフォームと特定の AMQ Broker バージョンの環境変数名を以下の表に示します。

コンテナープラットフォーム	環境変数名
OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_790**
IBM Z 上の OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781_s390x** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_782_s390x** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_790_s390x**
IBM Power Systems 上の OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_781_ppc64le** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_782_ppc64le** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_790_ppc64le**

各環境変数の値は、Red Hat から利用できるブローカーコンテナーイメージを指定します。以下に例を示します。

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

そのため、AMQ Broker のバージョンとコンテナープラットフォームをベースとするため、Operator は該当する環境変数名を決定します。Operator はブローカーコンテナーの起動時に対応するイメージ値を使用します。

注記

operator.yaml ファイルでは、Operator は **Secure Hash Algorithm** (SHA) 値で表されるイメージを使用します。数字記号 (**#**) 記号で始まるコメント行は、SHA 値が特定のコンテナーイメージタグに対応していることを示します。

2.4.2. Init コンテナーイメージの環境変数

Init コンテナーイメージの **operator.yaml** 設定ファイルに含まれる環境変数には、以下の命名規則があります。

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_<AMQ_Broker_version_identifier>

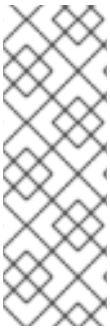
特定の AMQ Broker バージョンの環境変数名を以下に示します。

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

各環境変数の値は、Red Hat で利用できる Init コンテナーイメージを指定します。以下は例になります。

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

したがって、AMQ Broker のバージョンに基づいて、Operator は該当する環境変数名を決定します。Operator は init コンテナーの起動時に対応するイメージ値を使用します。



注記

例のように、Operator は **Secure Hash Algorithm** (SHA) 値で表されるイメージを使用します。数字記号 (**#**) 記号で始まるコメント行は、SHA 値が特定のコンテナーイメージタグに対応していることを示します。対応するコンテナーイメージタグが、0 **.4-17 形式のフローティングタグでは ない ことを確認** します。つまり、Operator で使用されるコンテナーイメージは固定されたままになります。Operator **は、新しい マイクロ イメージ バージョン（0.4 -17- n、n は最新のマイクロバージョン）を自動的にプルし、使用しませ** ん。

Init コンテナーイメージの **operator.yaml** 設定ファイルに含まれる環境変数には、以下の命名規則があります。

OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_<AMQ_Broker_version_identifier>

IBM Z での OpenShift Container Platform

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_<AMQ_Broker_version_identifier>

OpenShift Container Platform on IBM Power Systems

RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_<AMQ_Broker_version_identifier>

サポート対象のコンテナープラットフォームと特定の AMQ Broker バージョンの環境変数名を以下の表に示します。

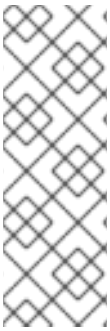
コンテナープラットフォーム	環境変数名

コンテナープラットフォーム	環境変数名
OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_790**
IBM Z 上の OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_790**
IBM Power Systems 上の OpenShift Container Platform	• **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_781** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_782** • **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_790**

各環境変数の値は、Red Hat で利用できる Init コンテナーイメージを指定します。以下に例を示します。

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_790
  #value: registry.redhat.io/amq7/amq-broker-init-rhel8:0.4-17-1
  value: registry.redhat.io/amq7/amq-broker-init-
rhel8@sha256:b74d03ed852a3731467ffda95266ce49f2065972f1c37bf254f3d52b34c11991
```

そのため、AMQ Broker のバージョンとコンテナープラットフォームをベースとするため、Operator は該当する環境変数名を決定します。Operator は init コンテナーの起動時に対応するイメージ値を使用します。



注記

例のように、Operator は **Secure Hash Algorithm** (SHA) 値で表されるイメージを使用します。数字記号 (**#**) 記号で始まるコメント行は、SHA 値が特定のコンテナーイメージタグに対応していることを示します。対応するコンテナーイメージタグが、0 **.4-17 形式のフローティングタグでは ない ことを確認** します。つまり、Operator で使用されるコンテナーイメージは固定されたままになります。Operator **は、新しい マイクロ イメージ バージョン（0.4 -17- n、n は最新のマイクロバージョン）を自動的にプルし、使用しませ** ん。

関連情報

- AMQ Broker Operator を使用してブローカーデプロイメントを作成する方法は、*3章AMQ Broker Operator を使用した OpenShift Container Platform での AMQ Broker のデプロイ* を参照してください。
- Operator が init コンテナーを使用してブローカー設定を生成する方法の詳細は、「Operator によるブローカー設定の生成方法」 を参照してください。
- **カスタム** Init コンテナーイメージをビルドし、指定する方法については、「カスタム Init コンテナーイメージの指定」 を参照してください。

2.5. OPERATOR デプロイメントノート

このセクションでは、Operator ベースのデプロイメントを計画する際の重要な考慮事項について説明します。

- AMQ Broker Operator に付随するカスタムリソース定義 (CRD) をデプロイするには、OpenShift クラスターのクラスター管理者権限が必要です。Operator がデプロイされると、管理者以外のユーザーは対応するカスタムリソース (CR) を使用してブローカーインスタンスを作成できます。通常ユーザーが CR をデプロイできるようにするには、クラスター管理者はまずロールおよびパーミッションを CRD に割り当てる必要があります。詳細は、OpenShift Container Platform ドキュメントの「カスタムリソース定義のクラスターロールの作成」を参照してください。
- 最新の Operator バージョンの CRD を使用してクラスターを更新する場合、今回の更新はクラスターの**すべて**のプロジェクトに影響を与えます。以前のバージョンの Operator からデプロイされたブローカー Pod は、それらのステータスを更新できなくなる可能性があります。OpenShift Container Platform Web コンソールで実行中のブローカー Pod の Logs タブをクリックすると、「UpdatePodStatus」が失敗したことを示すメッセージが表示されます。ただし、そのプロジェクトのブローカー Pod および Operator は予想通りに機能し続けます。影響を受けるプロジェクトに対してこの問題を解決するには、Operator の最新バージョンを使用するようプロジェクトをアップグレードする必要もあります。
- 複数のカスタムリソース(CR)インスタンスをデプロイして、特定の OpenShift プロジェクトに複数のブローカーデプロイメントを作成できますが、通常、プロジェクトに単一のブローカーデプロイメントを作成してから、アドレスに複数の CR インスタンスをデプロイします。Red Hat は、個別のプロジェクトで デプロイメントを作成することをお勧めします。
- 永続ストレージでブローカーをデプロイし、OpenShift クラスターに Container-native ストレージがない場合、永続ボリューム (PV) を手動でプロビジョニングし、それらが Operator で要求できるようにする必要があります。たとえば、永続ストレージ (CR に **persistenceEnabled=true**) を使用して 2 つのブローカーで構成されるクラスターを作成する場合は、永続ボリュームを 2 つ利用可能にしておく必要があります。デフォルトでは、各ブローカーインスタンスには 2 GiB のストレージが必要です。
  CR に **persistenceEnabled=false** を指定した場合、デプロイされたブローカーは **一時** ストレージを使用します。一時ストレージは、ブローカー Pod を再起動するたびに、既存のデータが失われることを意味します。

  OpenShift Container Platform での永続ストレージのプロビジョニングについての詳細は、以下を参照してください。

  - 永続ストレージについて (OpenShift Container Platform 4.5)
- CR を初めてデプロイする**前に**、一覧表示されている項目の設定をメインブローカー CR インスタンスに追加する必要があります。これらのアイテムの設定をすでに実行中のブローカーデプロイメントに追加することは**できません**。

  - 永続ストレージのデプロイメントで各ブローカーに必要な Persistent Volume Claim (永続ボ

リューム要求、PVC) のサイズ

- デプロイメント内の各ブローカーのメモリーおよび CPU の制限および要求

次のセクションの手順では、Operator をインストールし、カスタムリソース (CR) を使用して OpenShift Container Platform でブローカーデプロイメントを作成する方法を説明します。この手順を正常に完了したら、Operator が個別の Pod で実行されます。作成する各ブローカーインスタンスは、Operator と同じプロジェクトの StatefulSet の個別の Pod として実行されます。その後、専用のアドレス CR を使用してブローカーデプロイメントでアドレスを定義する方法を確認できます。

第3章 AMQ BROKER OPERATOR を使用した OPENSHIFT CONTAINER PLATFORM での AMQ BROKER のデプロイ

3.1. 前提条件

- Operator をインストールし、これを使用してブローカーデプロイメントを作成する前に、Operator のデプロイメントについて「「Operator デプロイメントノート」」で参照する必要があります。

3.2. CLI を使用した OPERATOR のインストール

注記

各 Operator リリースでは、以下で説明されているように、最新の **AMQ Broker 7.9.3 Operator インストールおよびサンプルファイルを** ダウンロードする必要があります。

本セクションの手順では、OpenShift コマンドラインインターフェース (CLI) を使用して、指定の OpenShift プロジェクトで AMQ Broker 7.9 の Operator の最新バージョンをインストールし、デプロイする方法を説明します。後続の手順で、この Operator を使用して一部のブローカーインスタンスをデプロイします。

- OperatorHub グラフィカルインターフェースを使用する AMQ Broker Operator の代替方法については、「OperatorHub を使用した Operator のインストール」 を参照してください。
- 既存の Operator ベースのブローカーデプロイメントの**アップグレード**に関する詳細は、6 章*Operator ベースのブローカーデプロイメントのアップグレード* を参照してください。

3.2.1. Operator コードの取得

この手順では、最新バージョンの AMQ Broker 7.9 用 Operator をインストールするのに必要なコードにアクセスし、準備する方法を説明します。

手順

1. Web ブラウザーで、AMQ Broker 7.9.3 リリース の **Software Downloads** ページに移動します。
2. **Version** ドロップダウンリストの値が **7.9.3** に設定され、Releases タブが選択されて **いる** ことを確認します。
3. **AMQ Broker 7.9.3 Operator Installation and Example Files** の横にある **Download** をクリックします。
   amq-broker-operator-7.MYBACKUPDIR-ocp-install-examples.zip 圧縮アーカイブを自動的にダウンロードします。
4. ダウンロードが完了したら、アーカイブを選択したインストールディレクトリーに移動します。以下の例では、アーカイブを ~/**broker/operator** という名前のディレクトリーに移動します。

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

5. 選択したインストールディレクトリーで、アーカイブの内容を展開します。以下に例を示します。

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

6. アーカイブの展開時に作成されたディレクトリーに移動します。以下に例を示します。

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

7. クラスター管理者として OpenShift Container Platform にログインします。以下に例を示します。

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

8. Operator をインストールするプロジェクトを指定します。新規プロジェクトを作成するか、または既存プロジェクトに切り替えることができます。

   a. 新しいプロジェクトを作成します。

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

   b. または、既存のプロジェクトに切り替えます。

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

9. Operator で使用するサービスアカウントを指定します。

   a. 展開した Operator アーカイブの **deploy** ディレクトリーで、**service_account.yaml** ファイルを開きます。

   b. **kind** 要素が **ServiceAccount** に設定されていることを確認します。

   c. **metadata** セクションで、カスタム名をサービスアカウントに割り当てるか、デフォルト名を使用します。デフォルトの名前は **amq-broker-operator** です。

   d. プロジェクトにサービスアカウントを作成します。

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

10. Operator のロール名を指定します。

    a. **role.yaml** ファイルを開きます。このファイルは、Operator が使用できるリソースを指定し、変更します。

    b. **kind** 要素が **Role** に設定されていることを確認します。

    c. **metadata** セクションで、カスタム名をロールに割り当てるか、デフォルト名を使用します。デフォルトの名前は **amq-broker-operator** です。

    d. プロジェクトにロールを作成します。

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

11. Operator のロールバインディングを指定します。ロールバインディングは、指定した名前に基づいて、事前に作成されたサービスアカウントを Operator ロールにバインドします。

    a. **role_binding.yaml** ファイルを開きます。**ServiceAccount** と **Role** の **name** の値が **service_account.yaml** および **role.yaml** ファイルで指定された値と一致していることを確認します。以下に例を示します。

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

    b. プロジェクトでロールバインディングを作成します。

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

以下の手順では、Operator をプロジェクトにデプロイします。

3.2.2. CLI を使用した Operator のデプロイ

本セクションの手順では、OpenShift コマンドラインインターフェース (CLI) を使用して、OpenShift プロジェクトに最新バージョンの Operator for AMQ Broker 7.9 をデプロイする方法を説明します。

前提条件

- Operator デプロイメントのために OpenShift プロジェクトを準備している必要があります。「Operator コードの取得」 を参照してください。
- AMQ Broker 7.3 以降では、新しいバージョンの Red Hat Ecosystem Catalog を使用してコンテナーイメージにアクセスします。この新しいバージョンのレジストリーでは、イメージにアクセスする前に認証されたユーザーである必要があります。本セクションの手順を実行する前に、「Red Hat Container Registry Authentication」 で説明されている手順を完了する必要があります。
- 永続ストレージでブローカーをデプロイし、OpenShift クラスターに Container-native ストレージがない場合、永続ボリューム (PV) を手動でプロビジョニングし、これらが Operator で要求できるようにする必要があります。たとえば、永続ストレージ (Custom Resource に **persistenceEnabled=true** を設定して) とともに 2 つのブローカーで構成されるクラスターを作成する場合は、2 つの PV が利用可能である必要があります。デフォルトでは、各ブローカーインスタンスには 2 GiB のストレージが必要です。
  カスタムリソースで **persistenceEnabled=false** を指定した場合、デプロイされたブローカーは**一時**ストレージを使用します。一時ストレージは、ブローカー Pod を再起動するたびに、既存のデータが失われることを意味します。

  永続ストレージのプロビジョニングの詳細は、以下を参照してください。

  - 永続ストレージについて (OpenShift Container Platform 4.5)

手順

1. OpenShift コマンドラインインターフェース (CLI) で、クラスター管理者として OpenShift にログインします。以下に例を示します。

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

2. Operator デプロイメント用に以前に準備したプロジェクトに切り替えます。以下に例を示します。

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

3. 以前の手順で Operator インストールアーカイブを展開する際に作成されたディレクトリーに移動します。以下に例を示します。

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

4. Operator に含まれる CRD をデプロイします。Operator をデプロイし、起動する前に CRD を OpenShift クラスターにインストールする必要があります。

   a. メインブローカー CRD をデプロイします。

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

   b. アドレス CRD をデプロイします。

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

   c. スケールダウンコントローラー CRD をデプロイします。

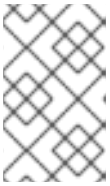
   ```
   $ oc create -f deploy/crds/broker_activemqartemisscaledown_crd.yaml
   ```

5. Red Hat Ecosystem Catalog での認証に使用されるアカウントに関連付けられたプルシークレットを、OpenShift プロジェクトの **default**、**deployer**、および **builder** サービスアカウントにリンクします。

```
$ 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. ダウンロードした Operator アーカイブの **deploy** ディレクトリーで、**operator.yaml** ファイルを開きます。以下のように、**spec.containers.image** プロパティーの値が Operator のバージョン 7.9.3-opr-3 に対応していることを確認します。

```
spec:
   template:
      spec:
         containers:
            #image: registry.redhat.io/amq7/amq-broker-rhel8-operator:7.9
            image: registry.redhat.io/amq7/amq-broker-rhel8-
operator@sha256:851ae51685e535317486b899eb0f80c3c5236464ae35ef3f9cde740173f7286b
```



注記

operator.yaml ファイルでは、Operator は **Secure Hash Algorithm** (SHA) 値で表されるイメージを使用します。数字記号 (**#**) 記号で始まるコメント行は、SHA 値が特定のコンテナーイメージタグに対応していることを示します。

7. オプションで**operator.yaml**ファイルの**WATCH_NAMESPACE**セクションを編集して、Operator が監視する名前空間を決定します。

- Operator をデプロイしてアクティブな名前空間を監視するには、このセクションは編集しないでください。

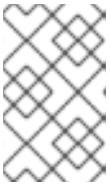
```
- name: WATCH_NAMESPACE
  valueFrom:
    fieldRef:
      fieldPath: metadata.namespace
```

- Operator をデプロイしてすべての名前空間を監視する方法:

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

- Operator をデプロイして複数の名前空間(例: **namespace1** と **namespace2**)を監視する方法:

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



注記

以前のバージョンの Operator を使用してブローカーをデプロイし、Operator をデプロイして多くの名前空間を監視する場合は、アップグレードする前にを参照してください。

8. Operator をデプロイします。

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

OpenShift プロジェクトで、Operator は新規 Pod で起動します。

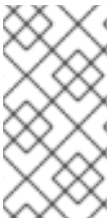
OpenShift Container Platform Web コンソールで、Operator Pod の **Events** タブにある情報により、OpenShift が指定した Operator イメージがデプロイされ、新規コンテナーが OpenShift クラスターのノードに割り当てられ、新規コンテナーが起動されていることを確認します。

さらに、Pod 内の **Logs** タブをクリックしても、出力には、以下のような行が含まれるはずです。

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

上記の出力では、新たにデプロイされた Operator が Kubernetes と通信していること、ブローカーおよびアドレス指定のコントローラーが実行されていることと、これらのコントローラーが一部のワーカーを起動していることを確認します。



注記

所定の OpenShift プロジェクトに AMQ Interconnect Operator の **単一のインスタンス** のみをデプロイすることが推奨されます。Operator デプロイメントの **spec.replicas** プロパティーを **1** より大きい値に設定し、同じプロジェクトで Operator を複数回デプロイしたりすることは推奨**されません**。

関連情報

- OperatorHub グラフィカルインターフェースを使用する AMQ Broker Operator の代替方法については、「OperatorHub を使用した Operator のインストール」 を参照してください。

3.3. OPERATORHUB を使用した OPERATOR のインストール

3.3.1. Operator Lifecycle Manager の概要

OpenShift Container Platform 4.5 以降では、**Operator Lifecycle Manager** (OLM) は、ユーザーがクラスター全体で実行されるすべての Operator やそれらの関連サービスをインストール、更新、およびそのライフサイクルを全般的に管理するのに役立ちます。これは、Kubernetes ネイティブアプリケーション (Operator) を効果的かつ自動化されたスケーラブルな方法で管理するために設計されたオープンソースツールキットの Operator Framework の一部です。

OLM は OpenShift Container Platform 4.5 以降 でデフォルトで実行されます。これは、クラスター管理者がクラスターで実行されている Operator をインストールし、アップグレードし、そのアクセス権限を付与するのに役立ちます。OpenShift Container Platform Web コンソールでは、クラスター管理者が Operator をインストールし、特定のプロジェクトアクセスを付与して、クラスターで利用可能な Operator のカタログを使用するための管理画面を利用できます。

OperatorHub は、OpenShift クラスター管理者が OLM を使用して Operator を検出し、インストールし、アップグレードするために使用するグラフィカルインターフェースです。1 回のクリックで、これらの Operator を OperatorHub からプルし、クラスターにインストールし、OLM で管理して、エンジニアリングチームが開発環境、テスト環境、および本番環境でソフトウェアをセルフサービスで管理できるようにします。

Operator をデプロイしている場合、カスタムリソース (CR) インスタンスを使用してスタンドアロンおよびクラスターブローカーブローカーデプロイメントを作成できます。

3.3.2. OperatorHub からの Operator のデプロイ

この手順では、OperatorHub を使用して、AMQ Broker の Operator の最新バージョンを指定された OpenShift プロジェクトにデプロイする方法について説明します。

重要

OperatorHub を使用した Operator のデプロイには、クラスター管理者権限が必要です。

前提条件

- **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator は Operator Hub で入手できる必要があります。

手順

1. クラスター管理者として OpenShift Container Platform Web コンソールにログインします。
2. 左側のナビゲーションメニューで、**Operators → OperatorHub** をクリックします。
3. **OperatorHub** ページ上部の **Project** ドロップダウンメニューで、Operator をデプロイするプロジェクトを選択します。
4. **OperatorHub** ページで、**Filter by keyword...** ボックスを使用して **Red Hat Integration - AMQ Broker Operator for RHEL 8 (Multiarch)** を見つけます。

注記

OperatorHub では、名前に **AMQ Broker** が含まれているよりも多くの Operator を見つける可能性があります。**Red Hat Integration-AMQ Broker for RHEL 8(Multiarch)** Operator をクリックしてください。この Operator をクリックしたら、開いている情報ペインを確認します。AMQ Broker 7.9 では、この Operator の最新のマイナーバージョンタグは 7. **9.3-opr-3 です**。

5. **Red Hat Integration - AMQ Broker for RHEL 8(Multiarch)** Operator をクリックします。表示されるダイアログボックスで、**Install** をクリックします。
6. **Install Operator** ページで以下を行います。
   a. **Update Channel** で、以下のラジオボタンで **7.x** を選択して、Operator の更新を追跡および受信するために使用されるチャネルを指定します。
      - **7.x** - このチャンネルは利用可能な場合に **7.10** に更新されます。
      - **7.8.x** - Long Term Support (LTS) チャンネルです。
   b. **インストールモード**で、Operator が監視する名前空間を選択します。
      - クラスター上の特定の名前空間: Operator は対象の名前空間にインストールされ、CR に変更がないか、対象の名前空間のみを監視します。
      - すべての名前空間 - Operator は、CR の変更がないか、すべての名前空間を監視します。

注記

以前のバージョンの Operator を使用してブローカーをデプロイし、Operator をデプロイして多くの名前空間を監視する場合は、アップグレードする前にを参照してください。

7. **Installed Namespace** ドロップダウンメニューから、Operator をインストールするプロジェクトを選択します。

8. **Approval Strategy** で、**Automatic** のラジオボタンが選択されていることを確認します。このオプションは、インストールを実行するために Operator への更新を手動で承認する必要がないように指定します。

9. **Install** をクリックします。

Operator のインストールが完了すると、**Installed Operators** ページが開きます。**Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator が指定したプロジェクト名前空間にインストールされていることが確認できるはずです。

関連情報

- AMQ Broker がインストールされているプロジェクトでブローカーデプロイメントを作成する方法は、「基本的なブローカーインスタンスのデプロイ」 を参照してください。

3.4. OPERATOR ベースのブローカーデプロイメントの作成

3.4.1. 基本的なブローカーインスタンスのデプロイ

以下の手順では、カスタムリソース (CR) インスタンスを使用して基本的なブローカーデプロイメントを作成する方法を説明します。

注記

- 複数のカスタムリソース(CR)インスタンスをデプロイして、特定の OpenShift プロジェクトに複数のブローカーデプロイメントを作成できますが、通常、プロジェクトに単一のブローカーデプロイメントを作成してから、アドレスに複数の CR インスタンスをデプロイします。
  Red Hat は、個別のプロジェクトで デプロイメントを作成することをお勧めします。

- AMQ Broker 7.9 では、以下の項目を設定する必要がある場合、最初に CR **をデプロイする前に**、メインのブローカー CR インスタンスに適切な設定を追加する必要があります。

  - 永続ストレージのデプロイメントで各ブローカーに必要な Persistent Volume Claim (永続ボリューム要求、PVC) のサイズ

  - デプロイメント内の各ブローカーのメモリーおよび CPU の制限および要求

前提条件

- AMQ Broker Operator がすでにインストールされている必要があります。

  - OpenShift コマンドラインインターフェース (CLI) を使用して AMQ Broker Operator をインストールするには、「CLI を使用した Operator のインストール」 を参照してください。

  - OperatorHub グラフィカルインターフェースを使用して AMQ Broker Operator をインストールするには、「OperatorHub を使用した Operator のインストール」 を参照してください。

- Operator がブローカーデプロイメントに使用するブローカーコンテナーイメージを選択する方法を理解している必要があります。詳細は、「Operator によるコンテナーイメージの選択方法」 を参照してください。

- AMQ Broker 7.3 以降では、新しいバージョンの Red Hat Ecosystem Catalog を使用してコンテナーイメージにアクセスします。この新しいバージョンのレジストリーでは、イメージにアクセスする前に認証されたユーザーである必要があります。本セクションの手順を実行する前に、「Red Hat Container Registry Authentication」 で説明されている手順を完了する必要があります。

手順

Operator が正常にインストールされると、Operator は実行され、CR に関連する変更をリッスンします。以下の手順では、CR インスタンスを使用して基本的なブローカーをプロジェクトにデプロイする方法を説明します。

1. ブローカーデプロイメントのカスタムリソース (CR) インスタンスの設定を開始します。

   a. OpenShift コマンドラインインターフェースの使用:

      i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemis_cr.yaml** というサンプル CR ファイルを開きます。

   b. OpenShift Container Platform Web コンソールの使用

      i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。

      ii. メインブローカー CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration → Custom Resource Definitions**をクリックします。

      iii. **ActiveMQArtemis** CRD をクリックします。

      iv. **Instances** タブをクリックします。

      v. **Create ActiveMQArtemis** をクリックします。
      コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

   基本的なブローカーデプロイメントの場合、設定が以下のように表示される可能性があります。この設定は、**broker_activemqartemis_cr.yaml** サンプル CR ファイルのデフォルトコンテンツです。

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

```
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
```

broker_activemqartemis_cr.yaml サンプル CR ファイルで、**image** プロパティーが **placeholder** のデフォルト値に設定されていることを確認します。この値はデフォルトで、**image** プロパティーによってデプロイメントに使用するブローカーコンテナーイメージが指定されていないことを示します。Operator が使用する適切なブローカーコンテナーイメージを判別する方法については、「Operator によるコンテナーイメージの選択方法」 を参照してください。

注記

broker_activemqartemis_cr.yaml サンプル CR は、**ex-aao** の命名規則を使用します。この命名規則は、CR が AMQ Broker **Operator** のリソースの**例**になります。AMQ Broker は **ActiveMQ Artemis** プロジェクトをベースにしています。このサンプル CR をデプロイする場合、生成される StatefulSet は **ex-aao-ss** の名前を使用します。さらに、デプロイメントのブローカー Pod は StatefulSet 名に基づいて直接使用されます (例: **ex-aao-ss-0**、**ex-aao-ss-1** など)。CR のアプリケーション名が StatefulSet のラベルとしてデプロイメントに表示されます。このラベルは Pod セレクターで使用できます。

2. **size** プロパティーはデプロイするブローカーの数を指定します。**2** 以上の値は、クラスターブローカーデプロイメントを指定します。ただし、単一のブローカーインスタンスをデプロイするには、値が **1** に設定されていることを確認します。

3. CR インスタンスをデプロイします。

   a. OpenShift コマンドラインインターフェースの使用:

      i. CR ファイルを保存します。

      ii. ブローカーデプロイメントを作成するプロジェクトに切り替えます。

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

      iii. CR インスタンスを作成します。

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

   b. OpenShift Web コンソールの使用

      i. CR の設定が完了したら、**Create** をクリックします。

4. Open Shift Container Platform Web コンソールで、**Workloads → StatefulSets** をクリックします。**ex-aao-ss** という新しい StatefulSet が表示されます。

   a. **ex-aao-ss** StatefulSet をクリックします。CR で定義される単一ブローカーに対応する Pod が 1 つあることが分かります。

   b. StatefulSet 内で **Pods** タブをクリックします。**ex-aao-ss** Pod をクリックします。実行中の Pod の **Events** タブで、ブローカーコンテナーが起動したことを確認できます。**Logs** タブには、ブローカー自体が実行中であることを示します。

5. ブローカーは通常実行されていることをテストするには、ブローカー Pod のシェルにアクセスしてテストメッセージを送信します。

a. OpenShift Container Platform Web コンソールの使用

i. **Workloads → Pods** をクリックします。

ii. **ex-aao-ss** Pod をクリックします。

iii. **Terminal** タブをクリックします。

b. OpenShift コマンドラインインターフェースの使用:

i. プロジェクトの Pod 名および内部 IP アドレスを取得します。

```
$ 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. ブローカー Pod のシェルにアクセスします。

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

6. シェルから **artemis** コマンドを使用して、一部のテストメッセージを送信します。URL にブローカー Pod の内部 IP アドレスを指定します。以下に例を示します。

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

上記のコマンドは、ブローカーに **demoQueue** というキューを自動的に作成し、デフォルトの数量 1000 のメッセージをキューに送信します。

以下のような出力が表示されるはずです。

```
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
```

関連情報

- メインのブローカーカスタムリソース (CR) の完全な設定リファレンスは、「カスタムリソース設定リファレンス」 を参照してください。
- 稼働中のブローカーを AMQ 管理コンソールに接続する方法は、*5章Operator ベースのブローカーデプロイメント用の AMQ 管理コンソール への接続* を参照してください。

3.4.2. クラスター化されたブローカーのデプロイ

2 つ以上のブローカー Pod がプロジェクトで実行されている場合、Pod はブローカークラスターを自動的に形成します。クラスター化の設定により、ブローカーは相互に接続でき、必要に応じてメッセージを再配布できます。

以下の手順では、クラスター化されたブローカーをデプロイする方法を説明します。デフォルトでは、このデプロイメントのブローカーは**オンデマンド**負荷分散を使用します。つまり、ブローカーは一致するコンシューマーを持つ他のブローカーにのみメッセージを転送します。

前提条件

- 基本的なブローカーインスタンスはすでにデプロイされています。「基本的なブローカーインスタンスのデプロイ」 を参照してください。

手順

1. 基本的なブローカーデプロイメントに使用した CR ファイルを開きます。
2. クラスター化したデプロイメントの場合は、**deploymentPlan.size** の値が **2** 以上であることを確認します。以下に例を示します。

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

注記

metadata セクションで、**namespace** プロパティーを追加し、OpenShift Container Platform Web コンソールを使用して CR インスタンスを作成する場合に**のみ**値を指定する必要があります。指定する値は、ブローカーデプロイメントの OpenShift プロジェクトの名前です。

3. 変更された CR ファイルを保存します。
4. 基本的なブローカーデプロイメントを作成したプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

5. 基本的なブローカーデプロイメントを先に作成したプロジェクトに切り替えます。

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

6. コマンドラインで変更を適用します。

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

OpenShift Container Platform Web コンソールで、追加のブローカー Pod は CR で指定される数に基づいてプロジェクトで起動します。デフォルトで、プロジェクトで実行されているブローカーはクラスター化されます。

7. 各 Pod の **Logs** タブを開きます。ログには、OpenShift が各ブローカーでクラスター接続ブリッジが確立されていることが示されています。具体的には、ログ出力には以下のような行が含まれます。

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

3.4.3. ブローカーデプロイメントの実行へのカスタムリソース変更の適用

以下は、ブローカーデプロイメントの実行にカスタムリソース (CR) 変更の適用について留意すべき重要な事項です。

- CR の **persistenceEnabled** 属性を動的に更新することはできません。この属性を変更するには、クラスターをゼロにスケールダウンします。既存の CR を削除します。次に、変更で CR を再作成し、再デプロイします。また、デプロイメントサイズも指定します。

- CR の **deploymentPlan.size** 属性の値は、**oc scale** コマンドによるブローカーデプロイメントのサイズの変更を上書きします。たとえば、**oc scale** を使用してデプロイメントのサイズを **3** つのブローカーから 2 つ変更する場合、CR の **deploymentPlan.size** の値は 3 つになります。この場合、OpenShift はまずデプロイメントを 2 つのブローカーにスケールダウンします。ただし、縮小操作が完了すると、Operator は CR で指定される 3 つのブローカーにデプロイメントを復元します。

- 「CLI を使用した Operator のデプロイ」 で説明されているように、永続ストレージ (CR に **persistenceEnabled=true**) でブローカーデプロイメントを作成する場合、ブローカー Pod について AMQ Broker Operator が要求する永続ボリューム (PV) をプロビジョニングする必要がある場合があります。ブローカーデプロイメントのサイズを縮小する場合、Operator はシャットダウンされるブローカー Pod で以前に要求された PV を解放します。ただし、CR を削除してブローカーデプロイメントを**削除**する場合、AMQ Broker Operator は削除時にデプロイメントにあるブローカー Pod の Persistent Volume Claim (永続ボリューム要求、PVC) を**解放しません**。また、これらのリリースされていない PV はいずれの新規デプロイメントでも利用できません。この場合は、ボリュームを手動で解放する必要があります。詳細は、OpenShift ドキュメントの「Release a persistent volume」を参照してください。

- AMQ Broker 7.9 では、以下の項目を設定する必要がある場合、最初に CR **をデプロイする前に**、メインの CR インスタンスに適切な設定を追加する必要があります。

  - 永続ストレージのデプロイメントで各ブローカーに必要な Persistent Volume Claim (永続ボリューム要求、PVC) のサイズ

  - デプロイメント内の各ブローカーのメモリーおよび CPU の制限および要求

- アクティブなスケーリングイベント時に、さらに適用する変更は Operator によってキューに入れられ、スケーリングが完了した場合にのみ実行されます。たとえば、デプロイメントのサイズを 4 つのブローカーから 1 つにスケールダウンするとします。次に、縮小が行われる間、ブローカー管理者のユーザー名およびパスワードの値も変更します。この場合、Operator は 1 つのアクティブなブローカーでデプロイメントが実行されるまで、ユーザー名とパスワードの変更をキューに入れます。

- すべての CR の変更: デプロイメントのサイズを変更したり、アクセプター、コネクター、またはコンソールの **expose** 属性の値を変更することとは別に、既存のブローカーが再起動されます。デプロイメントに複数のブローカーがある場合は、1 度に 1 つのブローカーのみを再起動します。

第4章 OPERATOR ベースのブローカーデプロイメントの設定

4.1. OPERATOR によるブローカー設定の生成方法

カスタムリソース (CR) インスタンスを使用してブローカーデプロイメントを設定する前に、Operator がブローカー設定を生成する方法を理解する必要があります。

Operator ベースのブローカーのデプロイメントを作成する場合、各ブローカーの Pod は OpenShift プロジェクトの StatefulSet で実行されます。ブローカーのアプリケーションコンテナーは各 Pod 内で実行されます。

Operator は、各 Pod を初期化する際に **Init コンテナー**と呼ばれるコンテナーのタイプを指定します。OpenShift Container Platform では、Init コンテナーはアプリケーションコンテナーの前に実行される特殊なコンテナーです。Init コンテナーには、アプリケーションイメージに存在しないユーティリティーまたはセットアップスクリプトを含めることができます。

デフォルトで、AMQ Broker Operator は組み込み Init コンテナーを使用します。Init コンテナーはデプロイメントのメイン CR インスタンスを使用して、各ブローカーアプリケーションコンテナーで使用される設定を生成します。

CR にアドレス設定を指定した場合、Operator はデフォルト設定を生成し、その設定を CR で指定された設定にマージするか、または置き換えます。このプロセスについては、以下の項で説明します。

4.1.1. Operator によるアドレス設定の生成方法

デプロイメントの主要カスタムリソース (CR) インスタンスにアドレス設定を追加している場合、以下で説明されているように Operator は各ブローカーのアドレス設定を生成します。

1. Operator は、ブローカーのアプリケーションコンテナーの前に Init コンテナーを実行します。Init コンテナーは**デフォルト**のアドレス設定を生成します。デフォルトのアドレス設定を以下に示します。

```
<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. カスタムリソース (CR) インスタンスでアドレス設定も指定した場合、Init Container プロセスがその設定を行い、それを XML に変換します。

3. CR の **applyRule** プロパティーの値に基づき、Init Container が**マージ**するか、上記のデフォルトのアドレス設定を CR で指定した設定に**置き換え**ます。このマージまたは置換の結果は、ブローカーが使用する最終アドレス設定になります。

4. Init コンテナーがブローカー設定の生成が終了すると (アドレス設定を含む)、ブローカーのアプリケーションコンテナーが起動します。起動時に、ブローカーコンテナーは以前に init コンテナーによって使用されたインストールディレクトリーから設定をコピーします。**broker.xml** 設定ファイルでアドレス設定を確認できます。実行中のブローカーの場合、このファイルは **/home/jboss/amq-broker/etc** ディレクトリーにあります。

関連情報

- CR で **applyRule** プロパティーを使用する例については、「Operator ベースのブローカーデプロイメントで設定されたアドレスへのマッチングアドレス設定」を参照してください。

4.1.2. ブローカー Pod のディレクトリー構造

Operator ベースのブローカーのデプロイメントを作成する場合、各ブローカーの Pod は OpenShift プロジェクトの StatefulSet で実行されます。ブローカーのアプリケーションコンテナーは各 Pod 内で実行されます。

Operator は、各 Pod を初期化する際に **Init コンテナー**と呼ばれるコンテナーのタイプを指定します。OpenShift Container Platform では、Init コンテナーはアプリケーションコンテナーの前に実行される特殊なコンテナーです。Init コンテナーには、アプリケーションイメージに存在しないユーティリティーまたはセットアップスクリプトを含めることができます。

ブローカーインスタンスの設定を生成する際に、Init コンテナーはデフォルトのインストールディレクトリーに含まれるファイルを使用します。このインストールディレクトリーは、Operator がブローカー Pod にマウントし、Init Container およびブローカーコンテナーを共有するボリュームにあります。共有ボリュームをマウントするために Init コンテナーが使用するパスは、**CONFIG_INSTANCE_DIR** という環境変数で定義されます。**CONFIG_INSTANCE_DIR** のデフォルト値は **/amq/init/config** です。本書では、このディレクトリーは **<install_dir>** と呼ばれます。

注記

CONFIG_INSTANCE_DIR 環境変数の値を変更することはできません。

デフォルトでは、インストールディレクトリーには以下のサブディレクトリーがあります。

サブディレクトリー	コンテンツ
<install_dir>/bin	ブローカーの実行に必要なバイナリーおよびスクリプト。
<install_dir>/etc	設定ファイル。
<install_dir>/data	ブローカーのジャーナル。
<install_dir>/lib	ブローカーの実行に必要な JAR およびライブラリー。
<install_dir>/log	ブローカーのログファイル。
<install_dir>/tmp	一時的な Web アプリケーションファイル。

Init コンテナーがブローカー設定の生成が終了すると、ブローカーのアプリケーションコンテナーが起動します。起動時に、ブローカーコンテナーは以前に init コンテナーによって使用されたインストールディレクトリーから設定をコピーします。ブローカー Pod が初期化され、実行されている場合、ブローカー設定はブローカーの **/home/jboss/amq-broker** ディレクトリー (およびサブディレクトリー) に置かれます。

関連情報

- Operator がビルトインの Init コンテナーのコンテナーイメージを選択する方法についての詳細は、「Operator によるコンテナーイメージの選択方法」 を参照してください。
- カスタム Init コンテナーイメージをビルドし、指定する方法については、「カスタム Init コンテナーイメージの指定」 を参照してください。

4.2. OPERATOR ベースのブローカーデプロイメントのアドレスおよびキューの設定

Operator ベースのブローカーのデプロイメントの場合、2 つの異なるカスタムリソース (CR) インスタンスを使用してアドレスおよびキューと関連する設定を行います。

- ブローカーでアドレスおよびキューを作成するには、アドレスカスタムリソース定義 (CRD) に基づいて CR インスタンスをデプロイします。
  - OpenShift コマンドラインインターフェース (CLI) を使用して Operator をインストールした場合、アドレス CRD は、ダウンロードした Operator インストールアーカイブの **deploy/crds** に含まれている **broker_activemqartemisaddress_crd.yaml** ファイルです。
  - OperatorHub を使用して Operator をインストールした場合、アドレス CRD は OpenShift Container Platform Web コンソールの**Administration → Custom Resource Definitions**に一覧表示されている **ActiveMQAretmisAddress** CRD になります。
- 特定のアドレスに一致するアドレスおよびキュー設定を設定するには、ブローカーデプロイメントの作成に使用されるメインのカスタムリソース (CR) インスタンスに設定を含めます。

- OpenShift CLI を使用して Operator をインストールした場合、メインのブローカー CRD は、ダウンロードした Operator インストールアーカイブの **deploy/crds** に含まれる **broker_activemqartemis_crd.yaml** ファイルです。
- OperatorHub を使用して Operator をインストールした場合、メインブローカー CRD は OpenShift Container Platform Web コンソールの**Administration → Custom Resource Definitions** に一覧表示されている **ActiveMQAretmis** CRD になります。

通常、OpenShift Container Platform でのブローカーデプロイメントに設定できるアドレスおよびキュー設定は、Linux または Windows のスタンドアロンブローカーデプロイメントのいずれでも**完全に同等**です。ただし、これらの設定についての**違い**に注意してください。これらの違いは、以下のサブセクションで説明します。

4.2.1. OpenShift とスタンドアロンブローカーデプロイメント間のアドレスおよびキュー設定の相違点

- OpenShift Container Platform のブローカーデプロイメントのアドレスおよびキュー設定を設定するには、ブローカーデプロイメントのメインカスタムリソース (CR) インスタンスの **addressSettings** セクションに設定を追加します。これは、Linux または Windows のスタンドアロンデプロイメントとは対照的で、**broker.xml** 設定ファイルの **address-settings** 要素に設定を追加します。
- 設定項目の名前に使用される形式は、OpenShift Container Platform とスタンドアロンブローカーデプロイメントとは異なります。OpenShift Container Platform デプロイメントでは、設定アイテム名は **camel ケース**に置かれます (例: **defaultQueueRoutingType**)。一方、スタンドアロンデプロイメントの設定項目名は小文字にあり、dash (**-**) セパレーターを使用します (例: **default-queue-routing-type**)。
以下の表は、この命名に関する他の例を紹介します。

スタンドアロンブローカーデプロイメントの設定アイテム	OpenShift ブローカーデプロイメントの設定アイテム
address-full-policy	addressFullPolicy
auto-create-queues	autoCreateQueues
default-queue-routing-type	defaultQueueRoutingType
last-value-queue	lastValueQueue

関連情報

- OpenShift Container Platform ブローカーデプロイメントのアドレスおよびキューの作成と一致する設定の例については、以下を参照してください。
  - OpenShift Container Platform でのブローカーデプロイメントのアドレスおよびキューの作成
  - OpenShift Container Platform でのブローカーデプロイメントに設定されたアドレス設定の一致

- OpenShift Container Platform フローカーデプロイメントのアドレス、キュー、およびアドレス設定のすべての設定オプションについては、「カスタムリソース設定リファレンス」 を参照してください。
- **スタンドアロン**ブローカーデプロイメントのアドレス、キュー、および関連アドレス設定に関する包括的な情報は、『AMQ Broker の設定』の「**アドレス、キュー、およびトピック**」を参照してください。この情報を使用して、OpenShift Container Platform のブローカーデプロイメントの同等の設定を作成できます。

4.2.2. Operator ベースのブローカーデプロイメントのアドレスおよびキューの作成

以下の手順では、カスタムリソース (CR) インスタンスを使用してアドレスおよび関連付けられたキューを Operator ベースのブローカーデプロイメントに追加する方法を説明します。

注記

ブローカーデプロイメントに複数のアドレスやキューを作成するには、個別の CR ファイルを作成してそれらを個別にデプロイし、それぞれのケースに新しいアドレスやキュー名を指定する必要があります。さらに、各 CR インスタンスの **name** 属性は一意である必要があります。

前提条件

- ブローカーでアドレスおよびキューを作成するために必要な専用のカスタムリソース定義 (CRD) を含む AMQ Broker Operator がすでにインストールされている必要があります。Operator のインストール方法の 2 つの代替方法については、以下を参照してください。
  - 「CLI を使用した Operator のインストール」.
  - 「OperatorHub を使用した Operator のインストール」.
- CR インスタンスを使用して基本的なブローカーデプロイメントを作成する方法を理解する必要があります。詳細は、「基本的なブローカーインスタンスのデプロイ」 を参照してください。

手順

1. カスタムリソース (CR) インスタンスの設定を開始し、ブローカーデプロイメントのアドレスおよびキューを定義します。
   a. OpenShift コマンドラインインターフェースの使用:
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemisaddress_cr.yaml** というサンプル CR ファイルを開きます。
   b. OpenShift Container Platform Web コンソールの使用
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。

ii. アドレス CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration → Custom Resource Definitions**をクリックします。

iii. **ActiveMQArtemisAddresss** CRD をクリックします。

iv. **Instances** タブをクリックします。

v. **Create ActiveMQArtemisAddress**をクリックします。
コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

2. CR の **spec** セクションで、行を追加してアドレス、キュー、およびルーティングタイプを定義します。以下に例を示します。

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

上記の設定では、**myQueue0** という名前のキューと **anycast** ルーティングタイプを持つ **myAddress0** という名前のアドレスが定義されます。

注記

metadata セクションで、**namespace** プロパティーを追加し、OpenShift Container Platform Web コンソールを使用して CR インスタンスを作成する場合に**のみ**値を指定する必要があります。指定する値は、ブローカーデプロイメントの OpenShift プロジェクトの名前です。

3. CR インスタンスをデプロイします。

a. OpenShift コマンドラインインターフェースの使用:

i. CR ファイルを保存します。

ii. ブローカーデプロイメントのプロジェクトに切り替えます。

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

iii. CR インスタンスを作成します。

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

b. OpenShift Web コンソールの使用

i. CR の設定が完了したら、**Create** をクリックします。

4. (オプション) CR インスタンスを使用して以前にデプロイメントに追加されたアドレスおよびキューを削除するには、以下のコマンドを使用します。

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

4.2.3. Operator ベースのブローカーデプロイメントで設定されたアドレスへのマッチングアドレス設定

クライアントにメッセージの配信に失敗した場合は、ブローカーがメッセージの配信を継続しようとしない場合があります。無限配信を試行するのを防ぐために、**デッドレターアドレス**と関連する**デッドレターキュー**を定義できます。指定の数の配信試行後、ブローカーは元のキューから未配信メッセージを削除し、そのメッセージを設定済みのデッドレターアドレスに送信します。システム管理者は、デッド文字キューから未配信メッセージを後で消費してメッセージを検査できます。

以下の例は、Operator ベースのブローカーデプロイメントのデッドレターアドレスおよびキューを設定する方法を示しています。この例では、以下の方法を示しています。

- メインのブローカーカスタムリソース (CR) インスタンスの **addressSetting** セクションを使用して、アドレスを設定します。
- これらのアドレス設定をブローカーデプロイメントのアドレスに一致させます。

前提条件

- Operator for AMQ Broker 7.9 の最新バージョン（バージョン 7.9.3-opr-3）を使用している必要があります。Operator を最新バージョンにアップグレードする方法については、6 章*Operator ベースのブローカーデプロイメントのアップグレード* を参照してください。
- CR インスタンスを使用して基本的なブローカーデプロイメントを作成する方法を理解する必要があります。詳細は、「基本的なブローカーインスタンスのデプロイ」 を参照してください。
- Operator がマージするか、または CR インスタンスで指定された設定に置き換える**デフォルト**のアドレス設定について理解している必要があります。詳細は、「Operator によるアドレス設定の生成方法」 を参照してください。

手順

1. CR インスタンスを設定して、デッドレターアドレスとキューを追加して、デプロイメント内の各ブローカーの配信されていないメッセージを受信します。
   a. OpenShift コマンドラインインターフェースの使用:
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemisaddress_cr.yaml** というサンプル CR ファイルを開きます。
   b. OpenShift Container Platform Web コンソールの使用
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。
      ii. アドレス CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration → Custom Resource Definitions**をクリックします。

iii. **ActiveMQArtemisAddresss** CRD をクリックします。

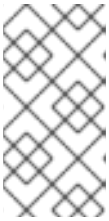
iv. **Instances** タブをクリックします。

v. **Create ActiveMQArtemisAddress** をクリックします。
コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

2. CR の **spec** セクションで、未配信のメッセージを受信するデッドレターアドレスおよびキューを指定する行を追加します。以下に例を示します。

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

上記の設定では、**myDeadLetterQueue** という名前のデッドレターキューと **anycast** ルーティングタイプを持つ **myDeadLetterAddress** という名前のデッドレターアドレスを定義します。



注記

metadata セクションで、**namespace** プロパティーを追加し、OpenShift Container Platform Web コンソールを使用して CR インスタンスを作成する場合に**のみ**値を指定する必要があります。指定する値は、ブローカーデプロイメントの OpenShift プロジェクトの名前です。

3. アドレス CR インスタンスをデプロイします。

a. OpenShift コマンドラインインターフェースの使用:

i. CR ファイルを保存します。

ii. ブローカーデプロイメントのプロジェクトに切り替えます。

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

iii. アドレス CR を作成します。

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

b. OpenShift Web コンソールの使用

i. CR の設定が完了したら、**Create** をクリックします。

4. ブローカーデプロイメントのカスタムリソース (CR) インスタンスの設定を開始します。

a. CR ファイルのサンプルの場合:

i. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemis_cr.yaml** というサンプル CR ファイルを開きます。

b. OpenShift Container Platform Web コンソールの使用

i. メインブローカー CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration** → **Custom Resource Definitions**をクリックします。

ii. **ActiveMQArtemis** CRD をクリックします。

iii. **Instances** タブをクリックします。

iv. **Create ActiveMQArtemis** をクリックします。
コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

基本的なブローカーデプロイメントの場合、設定が以下のように表示される可能性があります。この設定は、**broker_activemqartemis_cr.yaml** サンプル CR ファイルのデフォルトコンテンツです。

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

broker_activemqartemis_cr.yaml サンプル CR ファイルで、**image** プロパティーが **placeholder** のデフォルト値に設定されていることを確認します。この値はデフォルトで、**image** プロパティーによってデプロイメントに使用するブローカーコンテナーイメージが指定されていないことを示します。Operator が使用する適切なブローカーコンテナーイメージを判別する方法については、「Operator によるコンテナーイメージの選択方法」 を参照してください。

注記

metadata セクションで、**namespace** プロパティーを追加し、OpenShift Container Platform Web コンソールを使用して CR インスタンスを作成する場合に**のみ**値を指定する必要があります。指定する値は、ブローカーデプロイメントの OpenShift プロジェクトの名前です。

5. CR の **deploymentPlan** セクションで、以下に示すように単一の **addressSetting** セクションが含まれる新規 **addressSettings** セクションを追加します。

```
spec:
    version: 7.9.3
    deploymentPlan:
        size: 1
        image: placeholder
        requireLogin: false
        persistenceEnabled: true
```

```
      journalType: nio
      messageMigration: true
      addressSettings:
        addressSetting:
```

6. **addressSetting** ブロックに **match** プロパティーのインスタンスを 1 つ追加します。アドレス一致式を指定します。以下に例を示します。

```
spec:
  version: 7.9.3
  deployment Plan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    addressSettings:
      addressSetting:
       - match: myAddress
```

match

ブローカーが以下の設定を適用するアドレスまたはアドレスの**セット**を指定します。この例では、**match** プロパティーの値は **myAddress** と呼ばれる単一のアドレスに対応します。

7. 未配信メッセージに関連するプロパティーを追加し、値を指定します。以下に例を示します。

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

deadLetterAddress

ブローカーが未達のメッセージを送信するアドレス。

maxDeliveryAttempts

メッセージを設定済みのデッドレターアドレスに移動する前にブローカーが行う最大配信試行数。
上記の例では、ブローカーによって、**myAddress** で始まるアドレスにメッセージの配信が 5 回失敗する場合、ブローカーはメッセージを指定の dead letter address (**myDeadLetterAddress**) に移動します。

8. (オプション) 別のアドレスまたはアドレスセットに同様の設定を適用します。以下に例を示します。

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

この例では、2 つ目の **match** プロパティーの値にはアスタリスクワイルドカード文字が含まれます。ワイルドカード文字では、上記の設定が文字列 **myOtherAddresses** で始まる**任意**のアドレスに適用されることを意味します。

注記

ワイルドカード式を **match** プロパティーの値として使用する場合には、値を単一引用符で囲む必要があります (例: **'myOtherAddresses*'**)。

9. **addressSettings** セクションの最初に **applyRule** プロパティーを追加し、値を指定します。以下に例を示します。

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

applyRule プロパティーは、Operator を一致するアドレスまたはアドレスのセットごとに CR に追加する設定を適用する方法を指定します。指定できる値は次のとおりです。

merge_all

- CR で指定されるアドレス設定**と**、同じアドレスまたはアドレスのセットに一致するデフォルト設定の両方の場合:
  - デフォルト設定で指定されるプロパティー値を CR で指定されたプロパティー値に置き換えます。
  - CR **または**デフォルト設定で一意で指定されるプロパティー値を保持します。これらはそれぞれ最終マージされた設定の組み込みます。
- CR で指定されるアドレス設定**または**特定のアドレスセットに一意になるデフォルト設定の場合は、これらを最終でマージされた設定に含めます。

merge_replace

- CR に指定されたアドレス設定**と**、同じアドレスまたはアドレスセットに一致するデフォルト設定について、最終的なマージされた設定の **CR** に指定された設定を含めます。それらのプロパティーが CR で指定されていない場合でも、デフォルト設定に指定されたプロパティーを**含めないで**ください。
- CR で指定されるアドレス設定**または**特定のアドレスセットに一意になるデフォルト設定の場合は、これらを最終でマージされた設定に含めます。

replace_all

デフォルト設定に指定された**すべて**のアドレス設定を CR で指定されたアドレス設定に置き換えます。最後にマージされた設定は、CR で指定したものと完全に対応します。

注記

CR に **applyRule** プロパティーを明示的に含ない場合、Operator は **merge_all** のデフォルト値を使用します。

10. ブローカー CR インスタンスをデプロイします。
    a. OpenShift コマンドラインインターフェースの使用:
       i. CR ファイルを保存します。
       ii. CR インスタンスを作成します。

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

    b. OpenShift Web コンソールの使用
       i. CR の設定が完了したら、**Create** をクリックします。

関連情報

- OpenShift Container Platform ブローカーデプロイメントのアドレス、キュー、およびアドレス設定のすべての設定オプションについては、「カスタムリソース設定リファレンス」 を参照してください。
- OpenShift コマンドラインインターフェース (CLI) を使用して AMQ Broker Operator をインストールしている場合、ダウンロードしたインストールアーカイブおよび抽出したインストールアーカイブには、アドレス設定に関する追加例が含まれています。インストールアーカイブの **deploy/examples** ディレクトリーで、以下を参照してください。

- **artemis-basic-address-settings-deployment.yaml**
- **artemis-merge-replace-address-settings-deployment.yaml**
- **artemis-replace-address-settings-deployment.yaml**

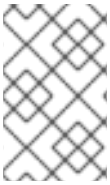
- **スタンドアロン**ブローカーデプロイメントのアドレス、キュー、および関連アドレス設定に関する包括的な情報は、『AMQ Broker の設定』の「**アドレス、キュー、およびトピック**」を参照してください。この情報を使用して、OpenShift Container Platform のブローカーデプロイメントの同等の設定を作成できます。
- Open Shift Container Platform の Init Containers の詳細は、Pod をデプロイする前に Init コンテナーを使用したタスクの実行を参照してください。

4.3. OPERATOR ベースのブローカーデプロイメント用のセキュリティー構成の作成

次の手順は、カスタムリソース(CR)インスタンスを使用して、ユーザーおよび関連するセキュリティー構成を　Operator ベースのブローカーデプロイメントに追加する方法を示しています。

前提条件

- AMQ Broker Operator がすでにインストールされている必要があります。Operator のインストール方法の 2 つの代替方法については、以下を参照してください。
  - 「CLI を使用した Operator のインストール」.
  - 「OperatorHub を使用した Operator のインストール」.
- ブローカーのセキュリティー保護で説明されているように、ブローカーのセキュリティーについて詳しく理解している必要があります
- CR インスタンスを使用して基本的なブローカーデプロイメントを作成する方法を理解する必要があります。詳細は、「基本的なブローカーインスタンスのデプロイ」 を参照してください。



手順

ブローカーデプロイメントを作成する前または後に、セキュリティー CR をデプロイできます。ただし、ブローカーデプロイメントの作成後にセキュリティー CR を展開すると、新しい設定を適用するために、ブローカー Pod が再起動されます。

1. カスタムリソース(CR)インスタンスの構成を開始して、ブローカーデプロイメントのユーザーと関連するセキュリティー設定を定義します。
   a. OpenShift コマンドラインインターフェースの使用:
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemissecurity_cr.yaml** というサンプル CR ファイルを開きます。

b. OpenShift Container Platform Web コンソールの使用

i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。

ii. アドレス CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration → Custom Resource Definitions**をクリックします。

iii. **Active MQArtemis Security CRD**をクリックします。

iv. **Instances** タブをクリックします。

v. **ActiveMQArtemisSecurity の作成**をクリックします。
コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

2. CR の**Spec**セクションで、ユーザーとロールを定義する行を追加します。以下は例になります。

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

```
        roles:
          - "receiver"
          ...
```

上記の設定では、ユーザーを2つ定義しています。

- **sender**のロールが割り当てられた **sam**という名前のユーザーを定義する**prop-module**という **propertiesLoginModule**。
- **receiver**のロールが割り当てられた **rob**という名前のユーザーを定義する**prop-module**という **propertiesLoginModule**。

これらのロールのプロパティーは、 **securityDomains**セクションの**brokerDomain**セクションと**broker**セクションで定義されます。たとえば、**send**ロールは、そのロールが割り当てられたユーザーが任意のアドレスに永続キューを作成できるように定義されています。デフォルトでは、構成は現在のネームスペースの CR によって定義されたすべてのデプロイ済みブローカーに適用されます。特定のブローカーに設定を限定する場合は、「セキュリティーのカスタムリソースの設定リファレンス」 に記載の **applyToCrNames** オプションを使用します。

注記

metadata セクションで、**namespace** プロパティーを追加し、OpenShift Container Platform Web コンソールを使用して CR インスタンスを作成する場合に**のみ**値を指定する必要があります。指定する値は、ブローカーデプロイメントの OpenShift プロジェクトの名前です。

3. CR インスタンスをデプロイします。
   a. OpenShift コマンドラインインターフェースの使用:
      i. CR ファイルを保存します。
      ii. ブローカーデプロイメントのプロジェクトに切り替えます。

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

      iii. CR インスタンスを作成します。

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

   b. OpenShift Web コンソールの使用
      i. CR の設定が完了したら、**Create** をクリックします。

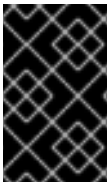
関連情報

- 「セキュリティーのカスタムリソースの設定リファレンス」
- 「基本的なブローカーインスタンスのデプロイ」

4.4. ブローカーのストレージ要件の設定

Operator ベースのブローカーデプロイメントで永続ストレージを使用するには、デプロイメントの作成に使用されるカスタムリソース (CR) インスタンスで **persistenceEnabled** を **true** に設定します。OpenShift クラスターに Container-native ストレージがない場合、永続ボリューム (PV) を手動でプロ

ビジョニングし、それらは Persistent Volume Claim (永続ボリューム要求、PVC) を使用して Operator で要求できるようにする必要があります。たとえば、永続ストレージを持つ 2 つのブローカーで構成されるクラスターを作成する場合は、2 つの PV が利用可能である必要があります。デフォルトでは、デプロイメントの各ブローカーには 2 GiB のストレージが必要です。ただし、ブローカーデプロイメントの CR を、各ブローカーに必要な PVC のサイズを指定するように設定できます。

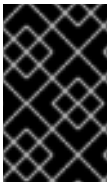


重要

初めて CR をデプロイする**前に**、ブローカーストレージサイズの設定をブローカーデプロイメントのメイン CR に追加する必要があります。すでに実行中のブローカーデプロイメントに設定を追加**できません**。

4.4.1. ブローカーのストレージサイズの設定

以下の手順では、ブローカーデプロイメントのカスタムリソース (CR) インスタンスを設定し、永続メッセージストレージ用に各ブローカーに必要な Persistent Volume Claim (永続ボリューム要求、PVC) のサイズを指定する方法を説明します。



重要

初めて CR をデプロイする**前に**、ブローカーストレージサイズの設定をブローカーデプロイメントのメイン CR に追加する必要があります。すでに実行中のブローカーデプロイメントに設定を追加**できません**。

前提条件

- **最低でも** AMQ Broker 7.7 の Operator の最新バージョン (バージョン 0.17) を使用する必要があります。Operator を AMQ Broker 7.9 の最新バージョンにアップグレードする方法については、*6章Operator ベースのブローカーデプロイメントのアップグレード* を参照してください。
- CR インスタンスを使用して基本的なブローカーデプロイメントを作成する方法を理解する必要があります。「基本的なブローカーインスタンスのデプロイ」 を参照してください。
- 永続ボリューム (PV) がすでにプロビジョニングされ、それらを Operator で要求できるように利用可能にする必要があります。たとえば、永続ストレージを持つ 2 つのブローカーのクラスターを作成する場合は、2 つの PV が利用可能である必要があります。
  永続ストレージのプロビジョニングの詳細は、以下を参照してください。
  - 永続ストレージについて (OpenShift Container Platform 4.5)

手順

1. ブローカーデプロイメントのカスタムリソース (CR) インスタンスの設定を開始します。
   a. OpenShift コマンドラインインターフェースの使用:
      i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemis_cr.yaml** というサンプル CR ファイルを開きます。

b. OpenShift Container Platform Web コンソールの使用

i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。

ii. メインブローカー CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration** → **Custom Resource Definitions** をクリックします。

iii. **ActiveMQArtemis** CRD をクリックします。

iv. **Instances** タブをクリックします。

v. **Create ActiveMQArtemis** をクリックします。
コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

基本的なブローカーデプロイメントの場合、設定が以下のように表示される可能性があります。この設定は、**broker_activemqartemis_cr.yaml** サンプル CR ファイルのデフォルトコンテンツです。

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

broker_activemqartemis_cr.yaml サンプル CR ファイルで、**image** プロパティーが **placeholder** のデフォルト値に設定されていることを確認します。この値はデフォルトで、**image** プロパティーによってデプロイメントに使用するブローカーコンテナーイメージが指定されていないことを示します。Operator が使用する適切なブローカーコンテナーイメージを判別する方法については、「Operator によるコンテナーイメージの選択方法」 を参照してください。

2. ブローカーストレージ要件を指定するには、CR の **deploymentPlan** セクションで、**Storage** セクションを追加します。**size** プロパティーを追加し、値を指定します。以下に例を示します。

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

storage.size

各ブローカー Pod が永続ストレージに必要な Persistent Volume Claim (永続ボリューム要求、PVC) のサイズ (バイト単位)。このプロパティーは、**persistenceEnabled** が **true** に設定されている場合にのみ適用されます。指定する値には単位が含まれている**必要があります**。バイト表記 (K、M、G など)、または同等のバイナリー (Ki、Mi、Gi) をサポートします。

3. CR インスタンスをデプロイします。

a. OpenShift コマンドラインインターフェースの使用:

i. CR ファイルを保存します。

ii. ブローカーデプロイメントを作成するプロジェクトに切り替えます。

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

iii. CR インスタンスを作成します。

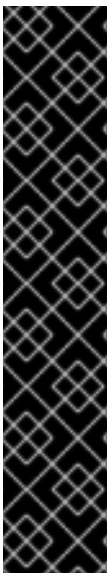
```
$ oc create -f <path/to/custom_resource_instance>.yaml
```

b. OpenShift Web コンソールの使用

i. CR の設定が完了したら、**Create** をクリックします。

4.5. OPERATOR ベースのブローカーデプロイメントのリソース制限および要求の設定

Operator ベースのブローカーデプロイメントを作成する場合、デプロイメントのブローカー Pod は OpenShift クラスターのノードの StatefulSet で実行されます。デプロイメントのカスタムリソース (CR) インスタンスを設定し、各 Pod で実行されるブローカーコンテナーによって使用される host-node コンピュートリソースを指定できます。CPU およびメモリー (RAM) の制限および要求値を指定することで、ブローカー Pod のパフォーマンスを確保できます。



重要

- ブローカーデプロイメントの CR を初めてデプロイする**前に**、制限および要求を CR インスタンスに追加する必要があります。すでに実行中のブローカーデプロイメントに設定を追加**できません**。
- これらの値は特定のメッセージングシステムのユースケースと実装したアーキテクチャーをベースとするため、Red Hat は制限およびリクエストの値を推奨できません。ただし、実稼働環境で設定する前に、これらの値をテストして開発環境で調整することが**推奨されます**。
- Operator は、各ブローカー Pod を初期化する際に **Init コンテナー**と呼ばれるコンテナーのタイプを指定します。各ブローカーコンテナーについて設定するリソース制限および要求は、各 Init コンテナーにも適用されます。ブローカーデプロイメントで Init コンテナーを使用する方法についての詳細は、「Operator によるブローカー設定の生成方法」 を参照してください。

以下の制限および要求値を指定できます。

CPU limit

Pod で実行されている各ブローカーコンテナーの場合、この値は、コンテナーが消費できるホストノード CPU の最大量になります。ブローカーコンテナーが指定の CPU 制限を超えると、OpenShift スロットルでコンテナーを調整します。これにより、ノードで実行中の Pod の数にかかわらず、コンテナーがパフォーマンスに一貫性を持たせることができます。

Memory limit

Pod で実行されている各ブローカーコンテナーの場合、この値は、コンテナーが消費できるホストノードメモリーの最大量です。ブローカーコンテナーが指定のメモリー制限を超過しようとすると、OpenShift はコンテナーを終了します。ブローカー Pod を再起動します。

CPU request

Pod で実行される各ブローカーコンテナーの場合、この値は、コンテナーが要求するホストノード CPU の量になります。OpenShift スケジューラーは、Pod の配置時に CPU 要求の値を考慮し、ブローカー Pod を十分なコンピュートリソースを持つノードにバインドします。
CPU request の値は、ブローカーコンテナーの実行に必要な CPU の**最小**量です。ただし、ノード上の CPU の競合がない場合、コンテナーは利用可能なすべての CPU を使用できます。CPU 制限を指定する場合には、コンテナーは CPU 使用量を超過することはできません。ノードに CPU の競合がある場合、CPU 要求の値により、OpenShift がすべてのコンテナーにおいて CPU 使用率を重み付けすることができます。

メモリー要求

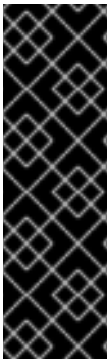
Pod で実行されている各ブローカーコンテナーについて、この値は、コンテナーが要求するホストノードメモリーの量になります。OpenShift スケジューラーは、Pod の配置時にメモリー要求の値を考慮し、ブローカー Pod を十分なコンピュートリソースを持つノードにバインドします。
メモリー要求値は、ブローカーコンテナーの実行に必要なメモリーの**最小**量です。ただし、コンテナーは可能な限り多くのメモリーを使用できます。メモリー制限を指定した場合、ブローカーコンテナーはそのメモリー量を超えることができません。

CPU は millicore という単位で測定されます。OpenShift クラスターの各ノードは、オペレーティングシステムを検査して、ノード上の CPU コア数を判別します。次に、ノードはその値を 1000 で乗算して、合計容量を表します。たとえば、ノードに 2 つのコアがある場合に、ノードの CPU 容量は **2000m** として表現されます。したがって、1 つのコアの連結を使用する場合は、**100m** の値を指定します。

メモリーはバイト単位で測定されます。バイト表記 (E、P、T、G、M、K) または同等のバイナリー (Ei、Pi、Ti、Gi、Mi、Ki) を使用して値を指定できます。指定する値には単位が含まれている必要があります。

4.5.1. ブローカーリソース制限および要求の設定

以下の例は、デプロイメントデプロイメントの主なカスタムリソース (CR) インスタンスを設定し、デプロイメントの Pod で実行される各ブローカーコンテナーの CPU およびメモリーの制限および要求を設定する方法を示しています。



重要

- ブローカーデプロイメントの CR を初めてデプロイする**前に**、制限および要求を CR インスタンスに追加する必要があります。すでに実行中のブローカーデプロイメントに設定を追加**できません**。
- これらの値は特定のメッセージングシステムのユースケースと実装したアーキテクチャーをベースとするため、Red Hat は制限およびリクエストの値を推奨できません。ただし、実稼働環境で設定する前に、これらの値をテストして開発環境で調整することが**推奨されます**。

前提条件

- CR インスタンスを使用して基本的なブローカーデプロイメントを作成する方法を理解する必要があります。「基本的なブローカーインスタンスのデプロイ」 を参照してください。

手順

1. ブローカーデプロイメントのカスタムリソース (CR) インスタンスの設定を開始します。

   a. OpenShift コマンドラインインターフェースの使用:

      i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemis_cr.yaml** というサンプル CR ファイルを開きます。

   b. OpenShift Container Platform Web コンソールの使用

      i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。

      ii. メインブローカー CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration → Custom Resource Definitions**をクリックします。

      iii. **ActiveMQArtemis** CRD をクリックします。

      iv. **Instances** タブをクリックします。

      v. **Create ActiveMQArtemis** をクリックします。
      コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

   基本的なブローカーデプロイメントの場合、設定が以下のように表示される可能性があります。この設定は、**broker_activemqartemis_cr.yaml** サンプル CR ファイルのデフォルトコンテンツです。

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

   broker_activemqartemis_cr.yaml サンプル CR ファイルで、**image** プロパティーが **placeholder** のデフォルト値に設定されていることを確認します。この値はデフォルト

で、**image** プロパティーによってデプロイメントに使用するブローカーコンテナーイメージが指定されていないことを示します。Operator が使用する適切なブローカーコンテナーイメージを判別する方法については、「Operator によるコンテナーイメージの選択方法」 を参照してください。

2. CR の **deploymentPlan** セクションで、**resources** セクションを追加します。**limits** および **requests** サブセクションを追加します。各サブセクションで **cpu** および **memory** プロパティーを追加し、値を指定します。以下に例を示します。

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

limits.cpu

デプロイメントで Pod で実行される各ブローカーコンテナーは、このホストノードの CPU 使用率を超過することはできません。

limits.memory

デプロイメントで Pod で実行される各ブローカーコンテナーは、このホストノードのメモリー使用率を超過することはできません。

requests.cpu

デプロイメントで Pod で実行される各ブローカーコンテナーはこのホストノード CPU の量を要求します。この値は、ブローカーコンテナーの実行に必要な CPU の**最小**量です。

requests.memory

デプロイメントで Pod で実行される各ブローカーコンテナーはこのホストノードメモリーを要求します。この値は、ブローカーコンテナーの実行に必要なメモリーの**最小**量です。

3. CR インスタンスをデプロイします。

   a. OpenShift コマンドラインインターフェースの使用:

      i. CR ファイルを保存します。

      ii. ブローカーデプロイメントを作成するプロジェクトに切り替えます。

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

      iii. CR インスタンスを作成します。

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

b. OpenShift Web コンソールの使用

i. CR の設定が完了したら、**Create** をクリックします。

4.6. カスタム INIT コンテナーイメージの指定

「Operator によるブローカー設定の生成方法」 で説明されているように、AMQ Broker Operator はデフォルトの組み込み Init コンテナーを使用してブローカー設定を生成します。設定を生成するために、Init コンテナーはデプロイメント用にメインのカスタムリソース (CR) インスタンスを使用します。CR に指定できる**唯一の**項目は、メインのブローカーカスタムリソース定義 (CRD) で公開される項目です。

ただし、CRD で公開**されない**設定を含める必要があるかもしれません。この場合、メイン CR インスタンスで**カスタム** Init コンテナーを指定できます。カスタム Init コンテナーは Operator によってすでに作成された設定を修正したり、追加したりできます。たとえば、カスタムの Init コンテナーを使用して、ブローカーのロギング設定を変更することができます。または、カスタムの Init コンテナーを使用して、ブローカーのインストールディレクトリーに追加のランタイム依存関係 (**.jar** ファイル) を含めることができます。

カスタムの Init コンテナーイメージを構築する場合は、以下の重要なガイドラインに従う必要があります。

- カスタムイメージ用に作成するビルドスクリプト (Docker Dockerfile または Podman Containerfile など) では、**FROM** 命令は最新バージョンの AMQ Broker Operator ビルトインの Init コンテナーイメージをベースイメージとして指定する必要があります。スクリプトに以下の行を追加します。

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

- カスタムイメージには、**/amq/scripts** というディレクトリーに追加する **post-config.sh** というスクリプトが含まれている必要があります。**post-config.sh** スクリプトは、Operator が生成する初期設定を変更または追加できます。カスタム Init コンテナーを指定する場合、Operator は **post-config.sh** スクリプトを実行します。これは、CR インスタンスを使用して設定を生成した**後**ですが、ブローカーアプリケーションコンテナーを起動する**前**に実行します。

- 「ブローカー Pod のディレクトリー構造」 で説明されているように、Init コンテナーによって使用されるインストールディレクトリーへのパスは、**CONFIG_INSTANCE_DIR** という環境変数で定義されます。**post-config.sh** スクリプトは、インストールディレクトリーを参照する際に、この環境変数名 (例: **${CONFIG_INSTANCE_DIR}/lib**) を使用し、この変数の値 (例: **/amq/init/config/lib**) **ではなく**、この環境変数名を使用する必要があります。

- カスタムブローカー設定に追加のリソース (**.xml** または **.jar** ファイルなど) を含める場合は、これらがカスタムイメージに含まれ、**post-config.sh** スクリプトからアクセスできることを確認する必要があります。

以下の手順では、カスタムの Init コンテナーイメージを指定する方法を説明します。

前提条件

- Operator のバージョン 7.9.3-opr-3 **以上** を使用する必要があります。最新の Operator バージョンにアップグレードする方法は、*6章Operator ベースのブローカーデプロイメントのアップグレード* を参照してください。

- 上記のガイドラインを満たす、カスタムの Init コンテナーイメージを構築する必要があります。ArtemisCloud Operator のカスタム Init コンテナーイメージをビルドし、指定する完全な例

については、「JDBC ベースの永続性のカスタム Init コンテナーイメージ」を参照してください。

- AMQ Broker Operator のカスタム Init コンテナーイメージを提供するには、Quay コンテナーレジストリーなどのコンテナーレジストリーのリポジトリーにイメージを追加できる必要があります。
- Operator による Init コンテナーの使用方法を理解し、ブローカー設定を生成する必要があります。詳細は、「Operator によるブローカー設定の生成方法」を参照してください。
- CR を使用してブローカーデプロイメントを作成する方法を理解している。詳細は、「Operator ベースのブローカーデプロイメントの作成」を参照してください。

手順

1. ブローカーデプロイメントのカスタムリソース (CR) インスタンスの設定を開始します。
   a. OpenShift コマンドラインインターフェースの使用:
      i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. ダウンロードした Operator インストールアーカイブの **deploy/crs** ディレクトリーに含まれる **broker_activemqartemis_cr.yaml** というサンプル CR ファイルを開きます。

   b. OpenShift Container Platform Web コンソールの使用
      i. デプロイメントを作成するプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。
      ii. メインブローカー CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration → Custom Resource Definitions** をクリックします。
      iii. **ActiveMQArtemis** CRD をクリックします。
      iv. **Instances** タブをクリックします。
      v. **Create ActiveMQArtemis** をクリックします。
      コンソールで、YAML エディターが開き、CR インスタンスを設定できます。

基本的なブローカーデプロイメントの場合、設定が以下のように表示される可能性があります。この設定は、**broker_activemqartemis_cr.yaml** サンプル CR ファイルのデフォルトコンテンツです。

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

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

broker_activemqartemis_cr.yaml サンプル CR ファイルで、**image** プロパティーが **placeholder** のデフォルト値に設定されていることを確認します。この値はデフォルトで、**image** プロパティーによってデプロイメントに使用するブローカーコンテナーイメージが指定されていないことを示します。Operator が使用する適切なブローカーコンテナーイメージを判別する方法については、「Operator によるコンテナーイメージの選択方法」 を参照してください。

2. CR の **deploymentPlan** セクションで、**initImage** プロパティーを追加します。

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

3. **initImage** プロパティーの値をカスタム Init コンテナーイメージの URL に設定します。

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

initImage

カスタムの Init コンテナーイメージの完全な URL を指定します。この URL をコンテナーレジストリーのリポジトリーに追加しておく必要があります。

4. CR インスタンスをデプロイします。

a. OpenShift コマンドラインインターフェースの使用:

i. CR ファイルを保存します。

ii. ブローカーデプロイメントを作成するプロジェクトに切り替えます。

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

iii. CR インスタンスを作成します。

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

b. OpenShift Web コンソールの使用

i. CR の設定が完了したら、**Create** をクリックします。

関連情報

- ArtemisCloud Operator のカスタム Init コンテナーイメージをビルドし、指定する完全な例については、「JDBC ベースの永続性のカスタム Init コンテナーイメージ」を参照してください。

4.7. クライアント接続用の OPERATOR ベースのブローカーデプロイメントの設定

4.7.1. アクセプターの設定

OpenShift デプロイメントでブローカー Pod へのクライアント接続を有効にするには、デプロイメントの**アクセプター**を定義します。アクセプターは、ブローカー Pod が接続を受け入れる方法を定義します。ブローカーのデプロイメントに使用されるメインのカスタムリソース (CR) でアクセプターを定義します。アクセプターを作成する場合は、アクセプターを有効にするメッセージングプロトコルや、これらのプロトコルに使用するブローカー Pod のポートなどの情報を指定します。

以下の手順は、ブローカーデプロイメントの CR で新規アクセプターを定義する方法を示しています。

前提条件

- アクセプターを設定するには、ブローカーのデプロイメントは AMQ Broker Operator のバージョン 0.9 以上をベースとする必要があります。Operator の最新バージョンのインストールについての詳細は、「CLI を使用した Operator のインストール」 を参照してください。

手順

1. 初回インストール時にダウンロードして展開した Operator アーカイブの **deploy/crs** ディレクトリーで、**broker_activemqartemis_cr.yaml** カスタムリソース (CR) ファイルを開きます。

2. **acceptors** 要素に名前付きアクセプターを追加します。**protocols** および **port** パラメーターを追加します。値を設定して、アクセプターおよび各ブローカー Pod のポートによってこれらのプロトコル用に公開されるメッセージングプロトコルを指定します。以下に例を示します。

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

```
    protocols: amqp
    port: 5672
...
```

設定されたアクセプターはポート 5672 を AMQP クライアントに公開します。**protocols** パラメーターに指定できる値の完全なセットが表に表示されます。

プロトコル	値
Core Protocol	**core**
AMQP	**amqp**
OpenWire	**openwire**
MQTT	**mqtt**
STOMP	**stomp**
すべてのサポート対象プロトコル	**all**

注記

- デプロイメントの各ブローカー Pod に対して、Operator はポート 61616 を使用するデフォルトのアクセプターも作成します。このデフォルトのアクセプターはブローカークラスタリングに必要ですが、Core Protocol は有効になっています。
- デフォルトでは、AMQ Broker 管理コンソールはブローカー Pod で 8161 ポートを使用します。デプロイメントの各ブローカー Pod には、コンソールへのアクセスを提供する専用のサービスがあります。詳細は、5 章*Operator ベースのブローカーデプロイメント用の AMQ 管理コンソールへの接続* を参照してください。

3. 同じアクセプターで別のプロトコルを使用するには、**protocol** パラメーターを変更します。プロトコルのコンマ区切りリストを指定します。以下に例を示します。

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

設定されたアクセプターはポート 5672 を AMQP および OpenWire クライアントに公開するようになりました。

4. アクセプターが許可する同時クライアント接続の数を指定するには、**connectionAllowed** パラメーターを追加して値を設定します。以下に例を示します。

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

5. デフォルトでは、アクセプターはブローカーデプロイメントと同じ OpenShift クラスターのクライアントにのみ公開されます。アクセプターを OpenShift 外部のクライアントに公開するには、**expose** パラメーターを追加し、値を **true** に設定します。

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

OpenShift 外部にあるクライアントにアクセプターを公開すると、Operator はデプロイメント内のブローカー Pod ごとに専用のサービスとルートを自動作成します。

6. OpenShift 外部のクライアントからアクセプターへのセキュアな接続を有効にするには、**sslEnabled** パラメーターを追加し、値を **true** に設定します。

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

アクセプター (またはコネクター) で SSL (Secure Sockets Layer) セキュリティーを有効にすると、以下のような関連する設定を追加できます。

- OpenShift クラスターに認証情報を保存するために使用されるシークレット名。アクセプターで SSL を有効にする場合は、シークレットが**必要**です。このシークレットの生成に関する詳細は、「ブローカークライアント接続のセキュリティー保護」 を参照してください。

- セキュアなネットワーク通信に使用する TLS (Transport Layer Security) プロトコル。TLS は、よりセキュアな SSL バージョンで更新されています。**enabledProtocols** パラメーターで TLS プロトコルを指定します。

- フローカーとクライアント間で、アクセプターが**相互認証**とも呼はれる双方向 TLS を使用するかどうか。これは、**needClientAuth** パラメーターの値を **true** に設定して指定します。

関連情報

- 認証情報を保存するシークレットの生成など、ブローカークライアント接続をセキュアにするように TLS を設定する方法は、「ブローカークライアント接続のセキュリティー保護」 を参照してください。
- アクセプターおよびコネクターの設定を含む完全なカスタムリソース参照については、「カスタムリソース設定リファレンス」 を参照してください。

4.7.2. ブローカークライアント接続のセキュリティー保護

アクセプターまたはコネクター (**sslEnabled** を **true** に設定) でセキュリティーを有効にしている場合、ブローカーとクライアント間での証明書ベースの認証を許可するように Transport Layer Security (TLS) を設定する必要があります。TLS は、よりセキュアな SSL バージョンで更新されています。2 つの主要な TLS 設定があります。

一方向 TLS

ブローカーのみが証明書を表示します。証明書はクライアントによってブローカーを認証するために使用されます。これが最も一般的な設定です。

双方向 TLS

ブローカーとクライアントの両方が証明書を提示します。これは**相互認証**と呼ばれることもあります。

これ以降のセクションで以下を説明します。

- 一方向および双方向 TLS が使用するブローカー証明書の設定要件
- 一方向 TLS の設定方法
- 双方向 TLS の設定方法

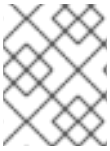
一方向と双方向 TLS の両方の場合、ブローカーとクライアント間の正常な TLS ハンドシェイクに必要な認証情報を保存するシークレットを生成して、設定を完了します。これは、セキュアなアクセプターまたはコネクターの **sslSecret** パラメーターに指定する必要のあるシークレット名です。シークレットには、Base64 でエンコードされたブローカーキーストア (一方向と双方向 TLS の両方)、Base64 でエンコードされたブローカートラストストア (two-way TLS のみ)、およびこれらのファイルに対応するパスワード (Base64 エンコード) が含まれる必要があります。一方向および双方向 TLS の設定手順では、このシークレットの生成方法を説明します。

注記

セキュアなアクセプターまたはコネクターの **sslSecret** パラメーターにシークレット名を明示的に指定しないと、アクセプターまたはコネクターはデフォルトのシークレット名を想定します。デフォルトのシークレット名は、 **<custom_resource_name>-<acceptor_name>-secret**または**<custom_resource_name>- <connector_name>-secret**の形式を使用します。例: **my-broker-deployment-my-acceptor-secret**

アクセプターまたはコネクターがデフォルトのシークレット名を想定している場合でも、このシークレットを独自に生成する必要があります。これは自動的に作成されません。

4.7.2.1. ホスト名検証用のブローカー証明書の設定



注記

本セクションでは、一方向または双方向 TLS の設定時に生成する必要のあるブローカー証明書の要件をいくつか説明します。

クライアントがデプロイメントでブローカー Pod への接続を試行する場合、クライアント接続 URL の **verifyHost** オプションはクライアントによって、ブローカーの証明書の Common Name (CN) をホスト名に比較するかどうかを判別し、一致することを確認します。クライアントが、クライアント接続 URL に **verifyHost=true** や同様の場合、クライアントはこの検証を実行します。

たとえば、ブローカーが分離されたネットワークの OpenShift クラスターにデプロイされる場合など、接続のセキュリティーに懸念がない場合、この検証を省略する場合があります。セキュアな接続では、クライアントがこの検証を実行することが推奨されます。この場合、ブローカーキーストア証明書の正しい設定は、クライアント接続を成功させるために不可欠です。

通常、クライアントがホストの検証を使用している場合、ブローカー証明書の生成時に指定する CN はクライアントが接続しているブローカー Pod の Route の完全なホスト名と一致する必要があります。たとえば、単一のブローカー Pod を持つデプロイメントがある場合、CN は以下のようになります。

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

CN が複数のブローカーを持つデプロイメントの**任意の**ブローカー Pod に解決するようにするには、ブローカー Pod の ordinal の場所でアスタリスク (*) ワイルドカード文字を指定できます。以下に例を示します。

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

前述の例に記載されている CN は、**my-broker-deployment** デプロイメントのブローカー Pod に正常に解決します。

さらに、ブローカー証明書の生成時に指定する SAN (Subject Alternative Name) は、カンマ区切りのリストとして、デプロイメント内のすべてのブローカー Pod を**個別に一覧表示**する必要があります。以下に例を示します。

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

4.7.2.2. 一方向 TLS の設定

本セクションの手順では、broker-client 接続のセキュリティーを保護するために一方向トランスポート層セキュリティー (TLS) を設定する方法を説明します。

一方向 TLS では、証明書を表示するブローカーのみが表示されます。この証明書は、クライアントがブローカーを認証するために使用されます。

前提条件

- クライアントがホスト名の検証を使用する場合のブローカー証明書の生成の要件を理解する必要があります。詳細は、「ホスト名検証用のブローカー証明書の設定」 を参照してください。

手順

1. ブローカーキーストアの自己署名証明書を生成します。

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

2. ブローカーキーストアから証明書をエクスポートし、クライアントと共有できるようにします。Base64 エンコードの **.pem** 形式の証明書をエクスポートします。以下に例を示します。

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

3. クライアントで、ブローカー証明書をインポートするクライアントトラストストアを作成します。

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

4. 管理者として OpenShift Container Platform にログインします。以下に例を示します。

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

5. ブローカーのデプロイメントが含まれるプロジェクトに切り替えます。以下に例を示します。

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

6. TLS 認証情報を保存するためのシークレットを作成します。以下に例を示します。

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

注記

シークレットを生成する際に、OpenShift ではキーストアとトラストストアの両方を指定する必要があります。トラストストアキーは、基本的に **client.ts** という名前です。ブローカーとクライアント間の一方向 TLS では、トラストストアは実際には必要ありません。ただし、シークレットを正常に生成するには、一部の有効なストアファイルを **client.ts** の値として指定する必要があります。前述の手順では、以前に生成されたブローカーキーストアファイルを再利用することで、**client.ts** の「dummy」値を指定します。これは、一方向 TLS に必要なすべての認証情報でシークレットを生成するには十分です。

7. シークレットを Operator のインストール時に作成したサービスアカウントにリンクします。以下に例を示します。

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

8. セキュアなアクセプターまたはコネクターの **sslSecret** パラメーターにシークレット名を指定します。以下に例を示します。

```
spec:
...
  acceptors:
```

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

4.7.2.3. 双方向 TLS の設定

本セクションの手順では、broker-client 接続のセキュリティーを保護するために双方向トランスポート層セキュリティー (TLS) を設定する方法を説明します。

双方向 TLS では、ブローカーとクライアントの両方が証明書を表示します。ブローカーおよびクライアントはこれらの証明書を使用して**相互認証**と呼ばれることもあります。

前提条件

- クライアントがホスト名の検証を使用する場合のブローカー証明書の生成の要件を理解する必要があります。詳細は、「ホスト名検証用のブローカー証明書の設定」 を参照してください。

手順

1. ブローカーキーストアの自己署名証明書を生成します。

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

2. ブローカーキーストアから証明書をエクスポートし、クライアントと共有できるようにします。Base64 エンコードの **.pem** 形式の証明書をエクスポートします。以下に例を示します。

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

3. クライアントで、ブローカー証明書をインポートするクライアントトラストストアを作成します。

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

4. クライアントで、クライアントキーストアの自己署名証明書を生成します。

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

5. クライアントで、クライアントキーストアから証明書をエクスポートし、ブローカーと共有できるようにします。Base64 エンコードの **.pem** 形式の証明書をエクスポートします。以下に例を示します。

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

6. クライアント証明書をインポートするブローカートラストストアを作成します。

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

7. 管理者として OpenShift Container Platform にログインします。以下に例を示します。

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

8. ブローカーのデプロイメントが含まれるプロジェクトに切り替えます。以下に例を示します。

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

9. TLS 認証情報を保存するためのシークレットを作成します。以下に例を示します。

```
$ 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>
```

注記

シークレットを生成する際に、OpenShift ではキーストアとトラストストアの両方を指定する必要があります。トラストストアキーは、基本的に **client.ts** という名前です。ブローカーとクライアント間の双方向 TLS の場合は、クライアント証明書を保持するため、ブローカートラストストアを含むシークレットを生成する必要があります。そのため、前の手順では、**client.ts** キーに指定した値は実際に**ブローカー**のトラストストアファイルになります。

10. シークレットを Operator のインストール時に作成したサービスアカウントにリンクします。以下に例を示します。

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

11. セキュアなアクセプターまたはコネクターの **sslSecret** パラメーターにシークレット名を指定します。以下に例を示します。

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

4.7.3. ブローカーデプロイメントのネットワークサービス

ブローカーデプロイメントの OpenShift Container Platform Web コンソールの **Networking** ペインで、2 つの実行中のサービスがあり、**ヘッドレス** サービスと **ping** サービスが 2 つあります。ヘッドレスサービスのデフォルト名は、**<custom_resource_name>-hdls-svc** の形式を使用します (例: **my-broker-deployment-hdls-svc**)。ping サービスのデフォルト名は、**<custom_resource_name>-ping-svc** の形式を使用します (例: **'my-broker-deployment-ping-svc**)。

ヘッドレスサービスは、各ブローカー Pod でポート 8161 および 61616 へのアクセスを提供します。ポート 8161 はブローカー管理コンソールに使用され、ポート 61616 はブローカーのクラスタリングに

使用されます。ヘッドレスサービスを使用して、内部クライアントからブローカー Pod に接続することもできます (つまり、ブローカーデプロイメントと同じ OpenShift クラスター内のクライアント)。

ping サービスは検出のブローカーによって使用されます。また、ブローカーは OpenShift 環境内でクラスターを形成できるようにします。内部的には、このサービスはポート 8888 を公開します。

関連情報

- ヘッドレスサービスを使用して内部クライアントからブローカー Pod に接続する方法については、「内部クライアントからのブローカーへの接続」 を参照してください。

4.7.4. 内部および外部クライアントからのブローカーへの接続

このセクションの例では、内部クライアント (つまりブローカーデプロイメントと同じ OpenShift クラスターのクライアント) および外部クライアント (OpenShift クラスター外のクライアント) からブローカーに接続する方法を示しています。

4.7.4.1. 内部クライアントからのブローカーへの接続

内部クライアントは、ブローカーデプロイメント用に実行されている **ヘッドレス** サービスを使用してブローカー Pod に接続できます。

ヘッドレスサービスを使用してブローカー Pod に接続するには、アドレスを **<Protocol>://<PodName>.<HeadlessServiceName>.<ProjectName>.svc.cluster.local** 形式で指定します。以下に例を示します。

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

OpenShift DNS は、Operator ベースのブローカーデプロイメントによって作成された StatefulSets は安定した Pod 名を提供するため、この形式でアドレスが正常に解決されました。

関連情報

- ブローカーデプロイメントのデフォルトで実行するヘッドレスサービスの詳細は、「ブローカーデプロイメントのネットワークサービス」 を参照してください。

4.7.4.2. 外部クライアントからのブローカーへの接続

外部クライアントにアクセプターを公開する場合 (つまり **expose** パラメーターの値を **true** に設定して)、Operator により、デプロイメントの各ブローカー Pod に専用のサービスと Route が自動的に作成されます。指定のブローカー Pod で設定された Routes を表示するには、OpenShift Container Platform Web コンソールの Pod を選択し、**Routes** タブをクリックします。

外部クライアントはブローカー Pod 用に作成される Route の完全なホスト名を指定して、ブローカーに接続できます。基本的な **curl** コマンドを使用して、この完全なホスト名への外部アクセスをテストできます。以下に例を示します。

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

Route の完全なホスト名は、OpenShift ルーターをホストするノードに解決する必要があります。OpenShift ルーターは、ホスト名を使用して、OpenShift 内部ネットワーク内のトラフィックを送信する場所を判別します。

デフォルトでは、OpenShift ルーターは、セキュアでないトラフィック (SSL 以外) トラフィックとポー

ト 443 (SSL で暗号化した) トラフィックに対してポート 80 をリッスンします。HTTP 接続の場合、ルーターはセキュアな接続 URL (**https**) を指定する場合 (**https**) またはポート 80 を指定する場合は、トラフィックをポート 443 に自動的に転送します。

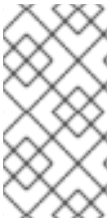
HTTP 以外の接続の場合:

- クライアントは、接続 URL の一部としてポート番号 (ポート 443 など) を明示的に指定する必要があります。
- 一方向 TLS では、クライアントは接続 URL の一部としてトラストストアと対応するパスワードへのパスを指定する必要があります。
- 双方向 TLS の場合、クライアントは接続 URL の一部としてその**キー**ストアと対応するパスワードへのパス**も**指定する必要があります。

以下は、サポートされるメッセージングプローブ用のクライアント接続 URL の例は次のとおりです。

一方向 TLS を使用する外部 Core クライアント

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



注記

外部コアクライアントはブローカーによって返されるトポロジー情報を使用できないため、**useTopologyForLoadBalancing** キーは接続 URL で **false** に明示的に設定されます。このキーが **true** に設定されているか、値を指定しないと、DEBUG ログメッセージが作成されます。

双方向 TLS を使用する外部 Core クライアント

```
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>
```

一方向 TLS を使用する外部 OpenWire クライアント

```
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>
```

双方向 TLS を使用する外部 OpenWire クライアント

```
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>
```

一方向 TLS を使用する外部 AMQP クライアント

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

双方向 TLS を使用する外部 AMQP クライアント

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

4.7.4.3. NodePort を使用したブローカーへの接続

ルートを使用する代わりに、OpenShift 管理者は NodePort を OpenShift 外部のクライアントからブローカー Pod に接続するように設定できます。NodePort は、ブローカーに設定されたアクセプターによって指定されるプロトコル固有のポートのいずれかにマップする必要があります。

デフォルトで、NodePort は 30000 から 32767 の範囲に置かれます。つまり、NodePort はブローカー Pod の意図されるポートとは一致しません。

NodePort 経由で OpenShift 外のクライアントからブローカーに接続するには、**<protocol>://<ocp_node_ip>:<node_port_number>** の形式で URL を指定します。

関連情報

- クラスターで実行されているサービスを使って OpenShift クラスター外からの通信を行うために Routes および NodePort などの方法についての詳細は、以下を参照してください。
  - ingress クラスタートラフィックの設定の概要 (OpenShift Container Platform 4.5 以降)

4.8. AMQP メッセージに対する大きなメッセージ処理の設定

クライアントは、ブローカーの内部バッファーのサイズを超える大きな AMQP メッセージを送信する可能性があり、予期せぬエラーが発生する可能性があります。この状態を回避するには、メッセージが指定の最小値よりも大きい場合にメッセージをファイルとして保存するようにブローカーを設定できます。このように大きなメッセージを処理すると、ブローカーはメモリー内にメッセージを保持しません。代わりに、ブローカーはメッセージを大きなメッセージファイルを保存するために使用される専用ディレクトリーに保存します。

OpenShift Container Platform でのブローカーデプロイメントでは、大きなメッセージディレクトリーは、メッセージストレージ用にブローカーが使用する永続ボリューム (PV) の **/opt/<custom_resource_name>/data/large-messages** です。ブローカーがメッセージを大きなメッセージとして保存すると、キューは大きなメッセージディレクトリーのファイルへの参照を保持します。

重要

AMQ Broker 7.9 の Operator ベースのブローカーデプロイメントでは、AMQP プロトコルでのみ大きなメッセージ処理を利用することができます。

4.8.1. 大規模なメッセージ処理のための AMQP アクセプターの設定

以下の手順は、指定したサイズよりも大きい AMQP メッセージを処理するようにアクセプターを設定する方法を説明します。

前提条件

- Operator ベースのブローカーデプロイメントのアクセプターの設定方法を理解する必要があります。「アクセプターの設定」 を参照してください。
- 大規模な AMQP メッセージを専用の大きなメッセージディレクトリーに保存するには、ブローカーデプロイメントは永続ストレージ (つまり、**persistenceEnabled** はデプロイメントの作成に使用するカスタムリソース (CR) インスタンスで **true** に設定する必要があります)。永続ストレージの設定についての詳細は、以下のドキュメントを参照してください。
  - 「Operator デプロイメントノート」
  - 「カスタムリソース設定リファレンス」

手順

1. AMQP アクセプターを定義したカスタムリソース (CR) インスタンスを開きます。
   a. OpenShift コマンドラインインターフェースの使用:

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

   b. OpenShift Container Platform Web コンソールの使用
      i. 左側のナビゲーションメニューで、**Administration → Custom Resource Definitions** をクリックします。
      ii. **ActiveMQArtemis CRD** をクリックします。
      iii. **Instances** タブをクリックします。
      iv. プロジェクトの名前空間に対応する CR インスタンスを見つけます。

   以前に設定された AMQP アクセプターは、以下のようになります。

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

2. ブローカーが大きいメッセージとして処理する AMQP メッセージの最小サイズをバイト単位で指定します。以下に例を示します。

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

```
      protocols: amqp
      port: 5672
      connectionsAllowed: 5
      expose: true
      sslEnabled: true
      amqpMinLargeMessageSize: 204800
      ...
...
```

上記の例では、ブローカーはポート 5672 で AMQP メッセージを受け入れるように設定されます。**amqpMinLargeMessageSize** の値に基づいて、アクセプターが 204800 バイトよりも大きい AMQP メッセージ (200 キロバイト以上) を受信する場合、ブローカーはメッセージを大きなメッセージとして格納します。

ブローカーはメッセージを、メッセージストレージ用にブローカーが使用する永続ボリューム (PV)の永続ボリューム (デフォルトでは **/opt/<custom_resource_name>/data/large-messages**) にメッセージを保存します。

amqpMinLargeMessageSize プロパティーの値を明示的に指定しないと、ブローカーは 102400 (つまり 100 キロバイト) のデフォルト値を使用します。

amqpMinLargeMessageSize を **-1** に設定すると、AMQP メッセージに対する大きなメッセージ処理が無効になります。

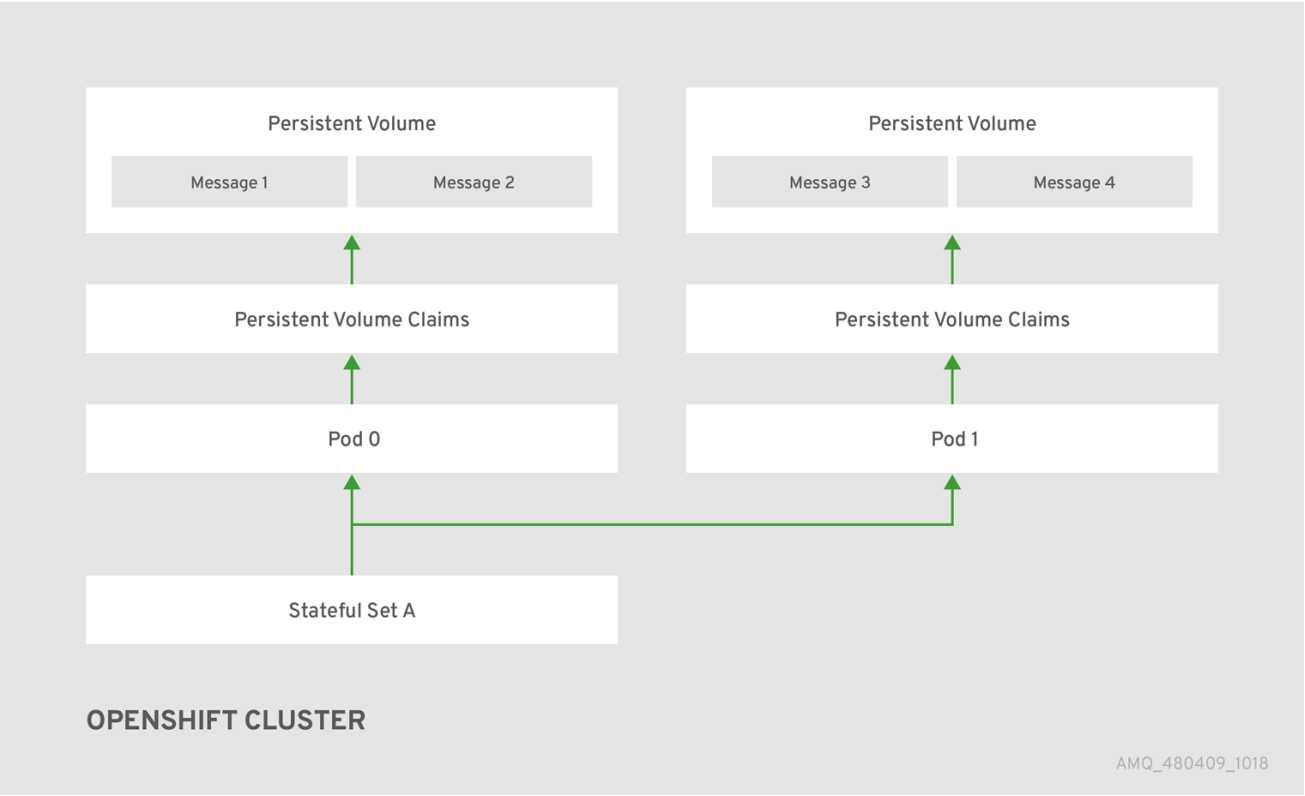
4.9. 高可用性およびメッセージの移行

4.9.1. 高可用性

高可用性という用語は、そのシステムの一部に障害が発生したりシャットダウンしている場合でも、稼働を継続できるシステムを指します。AMQ Broker on OpenShift Container Platform の場合、ブローカー Pod が失敗した場合にメッセージングデータの整合性と可用性を確保したり、デプロイメントを意図的にスケールダウンしてシャットダウンします。

OpenShift Container Platform の AMQ Broker の高可用性を確保するには、ブローカークラスターで複数のブローカー Pod を実行します。各ブローカー Pod は、永続ボリューム要求 (PVC) で使用するために要求する利用可能な永続ボリューム (PV) にメッセージデータを書き込みます。ブローカー Pod が失敗するか、またはシャットダウンされた場合、PV に保存されているメッセージデータはブローカークラスターの別の利用可能なブローカー Pod に移行されます。他のブローカー Pod はメッセージデータを独自の PV に保存します。

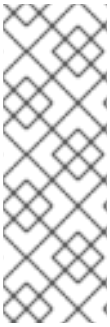
以下の図は、StatefulSet ベースのブローカーのデプロイメントを示しています。この場合、ブローカークラスターの 2 つのブローカー Pod は引き続き実行されます。



ブローカー Pod がシャットダウンすると、AMQ Broker Operator は、ブローカークラスターで実行中の別のブローカー Pod へのメッセージ移行を実行する**スケールダウンコントローラー**を自動的に開始します。このメッセージの移行プロセスは、**Pod のドレイン**としても知られています。以降のセクションでは、メッセージの移行について説明します。

4.9.2. メッセージの移行

デプロイメントの失敗や意図的なスケールダウンによってクラスターデプロイメントのブローカーがシャットダウンする場合に、メッセージの移行はメッセージングデータの整合性を確保することです。このプロセスは、**Pod のドレイン**としても、シャットダウンしたブローカー Pod からのメッセージの削除および再分配を指します。



注記

- メッセージ移行を実行するスケールダウンコントローラーは、単一の OpenShift プロジェクト内でのみ動作します。コントローラーは、別のプロジェクトのブローカー間でメッセージを移行できません。
- メッセージの移行を使用するには、デプロイメントに 2 つ以上のブローカーが必要です。デフォルトでは 2 つ以上のブローカーを持つブローカーはクラスター化されます。

Operator ベースのブローカーのデプロイメントでは、デプロイメントのメインブローカーカスタムリソースで **messageMigration** を **true** に設定して、メッセージの移行を有効にします。

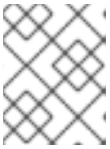
メッセージ移行プロセスは、次の手順を実行します。

1. デプロイメントのブローカー Pod がデプロイメントの失敗または意図的にスケールダウンによってシャットダウンすると、Operator はスケールダウンコントローラーを自動的に起動してメッセージ移行の準備を行います。縮小コントローラーは、ブローカークラスターと同じ OpenShift プロジェクト名で実行されます。

2. 縮小コントローラーは、それ自体を登録し、プロジェクトの Persistent Volume Claim (永続ボリューム要求、PVC) に関連する Kubernetes イベントをリッスンします。

3. 孤立した永続ボリューム (PV) の有無を確認するには、縮小コントローラーはボリューム要求上の序数を探します。コントローラーは、ボリューム要求の序数を、プロジェクトの StatefulSet (ブローカークラスター) で実行されているブローカー Pod と比較します。
ボリューム要求の序数がブローカー Pod の序数よりも高くなる場合、スケールダウンコントローラーは、その序数のブローカー Pod がシャットダウンされ、メッセージングデータが別のブローカー Pod に移行する必要があるかどうかを判断します。

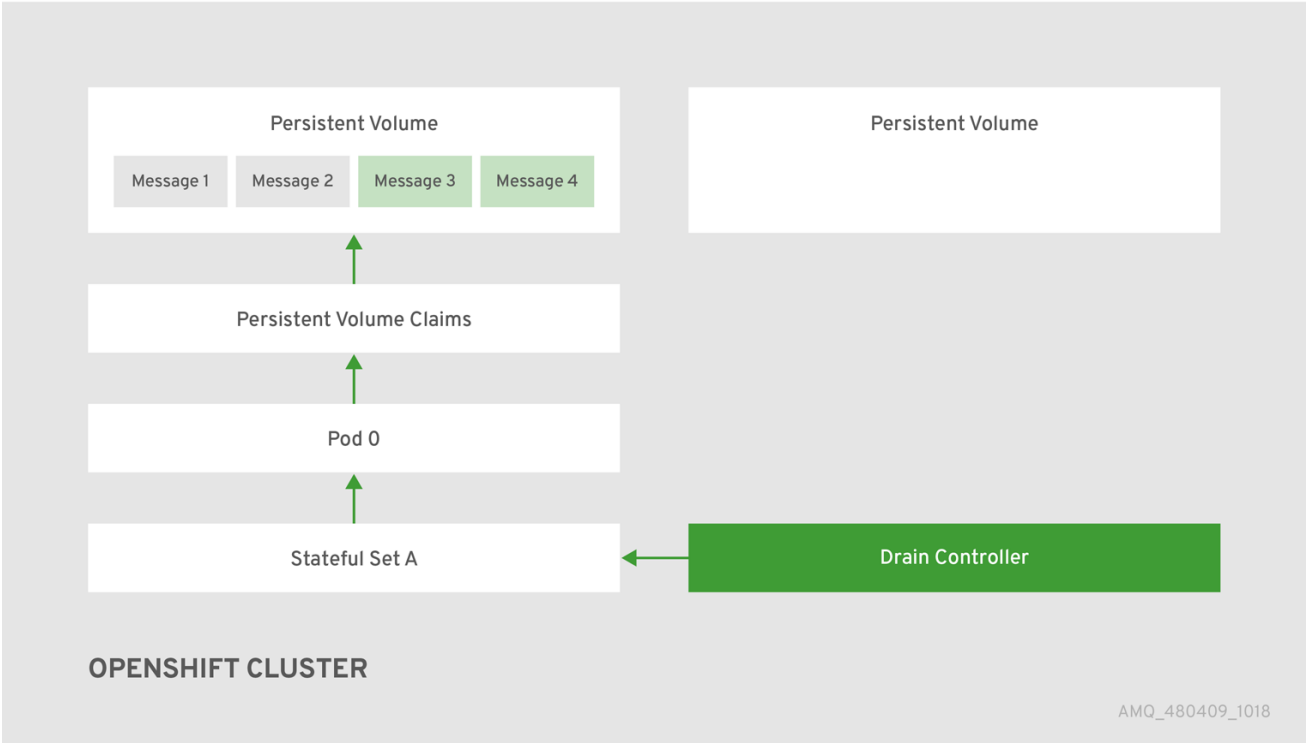
4. 縮小コントローラーはドレイン Pod を起動します。ドレイン Pod はブローカーを実行し、メッセージ移行を実行します。次に、ドレイン Pod は孤立したメッセージを移行する代替ブローカー Pod を識別します。



注記

メッセージの移行を可能にするには、少なくとも 1 つ以上のブローカー Pod がデプロイメントで実行されている必要があります。

以下の図は、スケールダウンコントローラー (**ドレインコントローラーとしても知られる**) がメッセージを稼働中のブローカー Pod に移行する方法を示しています。



メッセージを運用ブローカー Pod に正常に移行した後、ドレイン Pod はシャットダウンし、スケールダウンコントローラーは孤立した PV の PVC を削除します。PV は「Released」の状態に戻ります。

注記

ブローカーデプロイメントを 0 (ゼロ) にスケールダウンする場合、メッセージングデータを移行できる稼働中のブローカー Pod がないため、メッセージ移行は行われません。ただし、デプロイメントをゼロにスケールダウンしてから、元のデプロイメントよりも小さいサイズに再び戻すと、シャットダウンされたブローカーについてのドレイン Pod が起動します。

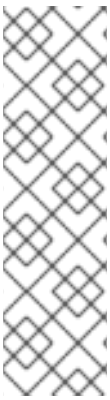
関連情報

- ブローカーのデプロイメントをスケールダウンする際のメッセージ移行の例については、「縮小後のメッセージの移行」を参照してください。

4.9.3. スケールダウン時のメッセージの移行

ブローカーデプロイメントの縮小時にメッセージを移行するには、メインブローカーカスタムリソース (CR) を使用してメッセージの移行を有効にします。AMQ Broker Operator は、クラスター化されたブローカーデプロイメントをスケールダウンする際に、専用のスケールダウンコントローラーを自動的に実行し、メッセージ移行を実行します。

メッセージ移行を有効にすると、Operator 内のスケールダウンコントローラーがブローカー Pod のシャットダウンを検出し、ドレイン Pod を開始し、メッセージ移行を実行します。ドレイン Pod は、クラスター内の他のライブブローカー Pod の 1 つに接続し、メッセージをそのライブブローカー Pod に移行します。移行が完了すると、スケールダウンコントローラーがシャットダウンします。



注記

- 縮小コントローラーは、単一の OpenShift プロジェクト内でのみ機能します。コントローラーは、別のプロジェクトのブローカー間でメッセージを移行できません。
- ブローカーデプロイメントを 0 (ゼロ) にスケールダウンする場合、メッセージングデータを移行できる稼働中のブローカー Pod がないため、メッセージ移行は行われません。ただし、デプロイメントをゼロブローカーにスケールダウンし、元のデプロイメントに含まれていた一部のブローカーのみに戻っても、シャットダウンされたブローカーのドレイン Pod が起動します。

以下の手順の例は、スケールダウンコントローラーの動作を示しています。

前提条件

- 基本的なブローカーデプロイメントがすでにある。「基本的なブローカーインスタンスのデプロイ」 を参照してください。
- メッセージの移行の仕組みを理解している。詳細は、「メッセージの移行」 を参照してください。

手順

1. 当初ダウンロードおよび抽出した Operator リポジトリーの **deploy/crs** ディレクトリーで、メインブローカー CR の **broker_activemqartemis_cr.yaml** を開きます。
2. メインブローカー CR では、**messageMigration** および **persistenceEnabled** を **true** に設定します。
   これらの設定は、クラスターブローカーデプロイメントのサイズを後でスケールダウンすると、Operator はスケールダウンコントローラーが自動的に起動し、メッセージを実行中のブローカー Pod に移行することができます。
3. 既存のブローカーデプロイメントで、実行中の Pod を確認します。

   ```
   $ oc get pods
   ```

   以下のような出力が表示されます。

```
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
```

上記の出力では、3 つの Pod が実行されていることが示されています。1 つはブローカー Operator 自体用で、デプロイメントの各ブローカーに個別の Pod が実行されていることを示しています。

4. 各 Pod にログインし、各ブローカーにメッセージを送信します。

   a. Pod **ex-aao-ss-0** にクラスター IP アドレスが **172.17.0.6** である場合は、以下のコマンドを実行します。

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

   b. Pod **ex-aao-ss-1** にクラスター IP アドレスが **172.17.0.7** である場合は、以下のコマンドを実行します。

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

   前述のコマンドは、各ブローカーに **TEST** というキューを作成し、各キューに 1000 個のメッセージを追加します。

5. クラスターを 2 つのブローカーにスケールダウンします。

   a. メインブローカー CR **broker_activemqartemis_cr.yaml** を開きます。

   b. CR で、**deploymentPlan.size** を **1** に設定します。

   c. コマンドラインで変更を適用します。

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

   Pod **ex-aao-ss-1** がシャットダウンを開始したことを確認します。縮小コントローラーは、同じ名前の新しいドレイン Pod を起動します。このドレイン Pod は、ブローカー Pod **ex-aao-ss-1** からクラスター内の他のブローカー Pod にすべてのメッセージを移行した後にシャットダウンします (**ex-aao-ss-0**)。

6. ドレイン Pod がシャットダウンされたら、ブローカー Pod **ex-aao-ss-0** の **TEST** キューのメッセージ数を確認します。キューのメッセージ数が 2000 であることを確認できます。これは、ドレイン Pod がシャットダウンするブローカー Pod から 1000 個のメッセージを正常に移行しました。

第5章 OPERATOR ベースのブローカーデプロイメント用の AMQ 管理コンソール への接続

Operator ベースのデプロイメント内の各ブローカー Pod は、ポート 8161 で AMQ 管理コンソールの独自のインスタンスをホストします。各ブローカーのコンソールへのアクセスを提供するには、ブローカーデプロイメントのカスタムリソース (CR) インスタンスを設定し、Operator に対して各ブローカー Pod に専用のサービスとルートを自動的に作成するように指示できます。

以下の手順では、デプロイされたブローカーの AMQ 管理コンソールに接続する方法を説明します。

前提条件

- AMQ Broker Operator を使用してブローカーデプロイメントを作成している必要があります。たとえば、サンプル CR を使用して基本的なブローカーデプロイメントを作成する方法は、「基本的なブローカーインスタンスのデプロイ」 を参照してください。
- Operator に対して、コンソールアクセスのためにデプロイメントで各ブローカー Pod のサービスおよびルートを自動的に作成するように指示するには、デプロイメントの作成に使用されるカスタムリソース (CR) インスタンスで **console.expose** プロパティーの値を **true** に設定する必要があります。このプロパティーのデフォルト値は **false** です。CR の **console** セクションの設定を含む完全なカスタムリソース設定の参照については、「カスタムリソース設定リファレンス」 を参照してください。

5.1. AMQ 管理コンソールへの接続

ブローカーデプロイメントの作成に使用されるカスタムリソース (CR) インスタンスで **console.expose** プロパティーの値を **true** に設定すると、Operator は各ブローカー Pod に専用のサービスとルートを自動的に作成し、AMQ 管理コンソールへのアクセスを提供します。

自動作成されたサービスのデフォルト名は **<custom-resource-name>-wconsj-<broker-pod-ordinal>-svc** の形式です。例: **my-broker-deployment-wconsj-0-svc**自動作成されたルートのデフォルト名は **<custom-resource-name>-wconsj-<broker-pod-ordinal>-svc-rte** 形式になります。例: **my-broker-deployment-wconsj-0-svc-rte**

この手順では、稼働中のブローカー Pod のコンソールにアクセスする方法を説明します。

手順

1. OpenShift Container Platform Web コンソールで、 **Networking → Routes** をクリックします。**Routes** ページで、指定のブローカー Pod の **wconsj** Route を特定します。例: **my-broker-deployment-wconsj-0-svc-rte**
2. **場所**で、ルートに対応するリンクをクリックします。
   Web ブラウザーで新しいタブが開きます。
3. **管理コンソール** リンクをクリックします。
   AMQ Management Console のログインページが開きます。
4. コンソールにログインするには、ブローカーデプロイメントの作成に使用されるカスタムリソース (CR) インスタンスの **adminUser** および **adminPassword** プロパティーに指定された値を入力します。
   CR の **adminUser** および **adminPassword** に値が明示的に指定されていない場合は、「AMQ Management Console のログインクレデンシャルへのアクセス」 の手順にしたがって、コンソールにログインするのに必要な認証情報を取得します。

注記

adminUser および **adminPassword** の値は、CR の **requireLogin** プロパティーが **true** に設定されている場合**のみ**コンソールにログインする必要があります。このプロパティーは、ブローカー**および**コンソールにログインするためにログイン認証情報が必要であるかどうかを指定します。**require Login**が**false**に設定されている場合には、ユーザー名とパスワードの入力を求められた時に任意のテキストを入力することで、有効なユーザー名パスワードを入力せずにコンソールにログインできます。

5.2. AMQ MANAGEMENT CONSOLE のログインクレデンシャルへのアクセス

ブローカーデプロイメントに使用するカスタムリソース (CR) インスタンスに **adminUser** および **adminPassword** の値を指定しない場合、Operator はこれらの認証情報を自動的に生成し、それらをシークレットに保存します。デフォルトのシークレット名は **<custom-resource-name>-credentials-secret** の形式を取ります (例: **my-broker-deployment-credentials-secret**)。

注記

adminUser および **adminPassword** の値は、CR の **requireLogin** パラメーターが **true** に設定されている場合に**のみ**管理コンソールにログインする必要があります。

require Loginが**false**に設定されている場合には、ユーザー名とパスワードの入力を求められた時に任意のテキストを入力することで、有効なユーザー名パスワードを入力せずにコンソールにログインできます。

以下の手順では、ログイン認証情報にアクセスする方法を説明します。

手順

1. OpenShift プロジェクトのシークレットの詳細な一覧を参照してください。
   a. OpenShift Container Platform Web コンソールから、**Workload → Secrets** をクリックします。
   b. コマンドラインで以下を行います。

```
$ oc get secrets
```

2. 適切なシークレットを開き、Base64 でエンコードされたコンソールログイン認証情報を表示します。
   a. OpenShift Container Platform Web コンソールから、名前にブローカーカスタムリソースインスタンスが含まれるシークレットをクリックします。**YAML** タブをクリックします。
   b. コマンドラインで以下を行います。

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

3. シークレットの値をデコードするには、以下のようなコマンドを実行します。

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

関連情報

- AMQ 管理コンソールを使用したブローカーの表示および管理に関する詳細は、『Managing AMQ Broker』の「Managing brokers using AMQ Management Console」を参照してください。

第6章 OPERATOR ベースのブローカーデプロイメントのアップグレード

本セクションの手順では、アップグレードする方法を説明します。

- OpenShift コマンドラインインターフェース (CLI) と OperatorHub の両方を使用した AMQ Broker Operator バージョン
- Operator ベースのブローカーデプロイメント用のブローカーコンテナーイメージ

6.1. 作業を開始する前に

このセクションでは、Operator ベースのブローカーデプロイメントの Operator およびブローカーコンテナーイメージをアップグレードする前に、いくつかの重要な考慮事項について説明します。

- OpenShift Container Platform 3.11 で稼働している Operator ベースのブローカーデプロイメントをアップグレードして OpenShift Container Platform 4.5 以降で実行するには、まず OpenShift Container Platform インストールをアップグレードする必要があります。次に、既存のデプロイメントに一致する新規の Operator ベースのブローカーデプロイメントを作成する必要があります。新規 Operator ベースのブローカーデプロイメントの作成方法は、*3章AMQ Broker Operator を使用した OpenShift Container Platform での AMQ Broker のデプロイ* を参照してください。
- OpenShift コマンドラインインターフェース(CLI)または OperatorHub のいずれかを使用して Operator をアップグレードするには、OpenShift クラスターのクラスター管理者権限が必要です。
- CLI を使用して Operator を**インストール**した場合、CLI を使用して Operator を**アップグレード**する必要もあります。OperatorHub を使用して Operator をインストールします (つまり、OpenShift Container Platform Web コンソールのプロジェクトの **Operators → Installed Operators** の下に表示される)、OperatorHub を使用して Operator をアップグレードする必要もあります。これらのアップグレード方法の詳細については、以下を参照してください。
  - 「CLI を使用した Operator のアップグレード」
  - 「OperatorHub を使用した Operator のアップグレード」
- たとえば、Operator をデプロイして多くの名前空間を監視して、すべての名前空間を監視する場合などは、次のことを行う必要があります。
  1. クラスター内のブローカーのデプロイメントに関連するすべての CR をバックアップしたことを確認してください。
  2. 既存の Operator をアンインストールします。
  3. 7.9 Operator をデプロイして、必要な名前空間を監視します。
  4. すべての展開を確認し、必要に応じて再作成します。

6.2. CLI を使用した OPERATOR のアップグレード

本セクションの手順では、OpenShift コマンドラインインターフェース (CLI) を使用して、異なるバージョンの Operator を AMQ Broker 7.9 で利用可能な最新バージョンに更新する方法を説明します。

6.2.1. 前提条件

- CLI を使用して最初に CLI を使用して Operator を**インストール**した場合のみ Operator をアップグレードする必要があります。OperatorHub を使用して Operator をインストールします (つまり、Operator は OpenShift Container Platform Web コンソールのプロジェクトの **Operators → Installed Operators** に表示されます)、OperatorHub を使用して Operator をアップグレードする必要があります。OperatorHub を使用して Operator をアップグレードする方法については、「OperatorHub を使用した Operator のアップグレード」 を参照してください。

6.2.2. Operator のバージョン 7.8.x のアップグレード

この手順では、OpenShift コマンドラインインターフェース (CLI) を使用して Operator のバージョン 7.8.x を AMQ Broker 7.9 の最新バージョンにアップグレードする方法を説明します。

手順

1. Web ブラウザーで、AMQ Broker 7.9.3 パッチの **Software Downloads** ページに移動します。
2. **Version** ドロップダウンリストの値が **7.9.3** に設定され、**Patches** タブが選択されていることを確認します。
3. **AMQ Broker 7.9.3 Operator Installation and Example Files** の横にある **Download** をクリックします。
amq-broker-operator-7.MYBACKUPDIR-ocp-install-examples.zip 圧縮アーカイブを自動的にダウンロードします。
4. ダウンロードが完了したら、アーカイブを選択したインストールディレクトリーに移動します。以下の例では、アーカイブを **~/broker/operator** という名前のディレクトリーに移動します。

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

5. 選択したインストールディレクトリーで、アーカイブの内容を展開します。以下に例を示します。

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

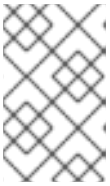
6. 既存の Operator デプロイメントが含まれるプロジェクトの管理者として OpenShift Container Platform にログインします。

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

7. Operator バージョンをアップグレードする OpenShift プロジェクトに切り替えます。

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

8. ダウンロードした最新の Operator アーカイブの **deploy** ディレクトリーで、**operator.yaml** ファイルを開きます。



注記

operator.yaml ファイルでは、Operator は **Secure Hash Algorithm** (SHA) 値で表されるイメージを使用します。数字記号 (**#**) 記号で始まるコメント行は、SHA 値が特定のコンテナーイメージタグに対応していることを示します。

9. **以前**の Operator デプロイメントの **operator.yaml** ファイルを開きます。以前の設定で指定したデフォルト以外の値が**新しい operator.yaml** 設定ファイルに複製されていることを確認します。

10. **新しい operator.yaml** ファイルへの更新を行った場合は、ファイルを保存します。

11. 更新された Operator 設定を適用します。

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

OpenShift は、最新の Operator バージョンを使用するようにプロジェクトを更新します。

12. 直前のブローカーデプロイメントを再作成するには、元の CR の目的に一致するように新規 CR yaml ファイルを作成し、これを適用します。「基本的なブローカーインスタンスのデプロイ」では、Operator インストールアーカイブに **deploy/crs/broker_activemqartemis_cr.yaml** ファイルを適用する方法を記述します。このファイルは新規 CR yaml ファイルのベースとして使用することができます。

6.3. OPERATORHUB を使用した OPERATOR のアップグレード

本セクションでは、OperatorHub を使用して、異なるバージョンの Operator を AMQ Broker 7.9 で利用可能な最新バージョンに更新する方法を説明します。

6.3.1. 前提条件

- OperatorHub を使用して Operator を**インストールし** (つまり、OpenShift Container Platform Web コンソールのプロジェクトの **Operators → Installed Operators** の下に表示される) OperatorHub を使用して Operator をアップグレードする必要があります。一方、OpenShift コマンドラインインターフェース (CLI) を使用して Operator をインストールした場合、CLI を使用して Operator をアップグレードする必要もあります。CLI を使用して Operator をアップグレードする方法については、「CLI を使用した Operator のアップグレード」 を参照してください。

- OperatorHub を使用して AMQ Broker Operator をアップグレードするには、OpenShift クラスターのクラスター管理者権限が必要です。

6.3.2. 作業を開始する前に

本セクションでは、OperatorHub を使用して AMQ Broker Operator のインスタンスをアップグレードする前に、いくつかの重要な考慮事項について説明します。

- Operator Lifecycle Manager は、OperatorHub から最新の Operator バージョンをインストールする際に、OpenShift クラスターの CRD を**自動的**に更新します。既存の CRD を削除する必要はありません。

- 最新の Operator バージョンの CRD を使用してクラスターを更新する場合、今回の更新はクラスターの**すべて**のプロジェクトに影響を与えます。以前のバージョンの Operator からデプロイされたブローカー Pod は、OpenShift Container Platform Web コンソールでそれらのステータスを更新できなくなる可能性があります。稼働中のブローカー Pod の **Logs** タブをクリックし

たら、「UpdatePodStatus」が失敗したことを示すメッセージが表示されます。ただし、そのプロジェクトのブローカー Pod および Operator は予想通りに機能し続けます。影響を受けるプロジェクトに対してこの問題を解決するには、Operator の最新バージョンを使用するようプロジェクトをアップグレードする必要もあります。

6.3.3. OperatorHub を使用した Operator のアップグレード

この手順では、OperatorHub を使用して AMQ Broker Operator のインスタンスをアップグレードする方法を説明します。

手順

1. クラスター管理者として OpenShift Container Platform Web コンソールにログインします。
2. プロジェクトのブローカーデプロイメントのメインカスタムリソース (CR) インスタンスを削除します。このアクションにより、ブローカーデプロイメントが削除されます。
   a. 左側のナビゲーションメニューで、**Administration → Custom Resource Definitions**をクリックします。
   b. **Custom Resource Definitions**ページで、**ActiveMQArtemis CRD** をクリックします。
   c. **Instances** タブをクリックします。
   d. プロジェクトの名前空間に対応する CR インスタンスを見つけます。
   e. CR インスタンスの場合は、右側の **More Options** アイコン (3 つの点) をクリックします。**Delete ActiveMQArtemis** を選択します。
3. プロジェクトから既存の AMQ Broker Operator をアンインストールします。
   a. 左側のナビゲーションメニューで、**Operators → Installed Operators** をクリックします。
   b. ページ上部の **Project** ドロップダウンメニューから、Operator をアンインストールするプロジェクトを選択します。
   c. アンインストールする **Red Hat Integration - AMQ Broker**インスタンスを見つけます。
   d. Operator インスタンスの場合は、右側の **More Options** アイコン (3 つの点) をクリックします。**Uninstall Operator** を選択します。
   e. 確認ダイアログボックスで、**Uninstall** をクリックします。
4. OperatorHub を使用して、AMQ Broker 7.9 の Operator の最新バージョンをインストールします。詳細は、「OperatorHub からの Operator のデプロイ」 を参照してください。
5. 直前のブローカーデプロイメントを再作成するには、元の CR の目的に一致するように新規 CR yaml ファイルを作成し、これを適用します。「基本的なブローカーインスタンスのデプロイ」では、Operator インストールアーカイブに **deploy/crs/broker_activemqartemis_cr.yaml** ファイルを適用する方法を記述します。このファイルは新規 CR yaml ファイルのベースとして使用することができます。

6.4. AMQ BROKER バージョンの指定によるブローカーコンテナーイメージのアップグレード

以下の手順では、AMQ Broker バージョンを指定して、Operator ベースのブローカーデプロイメントの

ブローカーコンテナーイメージをアップグレードする方法を説明します。たとえば、Operator を AMQ Broker 7.9.3 の最新バージョンにアップグレードするものの、CR の **spec.upgrades.enabled** プロパティーがすでに **true** に設定され、**spec.version** プロパティーが 7.8.0 を指定します。ブローカーコンテナーイメージを **アップグレード** するには、新しい AMQ Broker バージョンを手動で指定する必要があります（例： **7.9.3**）。新しいバージョンの AMQ Broker を指定する場合、Operator はこのバージョンに対応するブローカーコンテナーイメージを自動的に選択します。

前提条件

- 7.9.3 には、Operator の最新バージョンを使用する必要があります。Operator を最新バージョンにアップグレードする方法については、以下を参照してください。
  - 「CLI を使用した Operator のアップグレード」
  - 「OperatorHub を使用した Operator のアップグレード」.
- 「Operator によるコンテナーイメージの選択方法」 で説明されているように、CR をデプロイし、ブローカーコンテナーイメージを明示的に指定しない場合、Operator は使用する適切なコンテナーイメージを自動的に選択します。このセクションで説明されているアップグレードプロセスを使用するには、このデフォルトの動作を使用する**必要があります**。CR でブローカーコンテナーイメージを直接指定し、デフォルト動作を上書きする場合、Operator は以下で説明されているように、ブローカーコンテナーイメージを自動的に AMQ Broker バージョンに対応するようにアップグレードすることは**できません**。

手順

1. ブローカーデプロイメントのメインブローカー CR インスタンスを編集します。
   a. OpenShift コマンドラインインターフェースの使用:
      i. ブローカーデプロイメントのプロジェクトで CR を編集およびデプロイする権限を持つユーザーとして OpenShift にログインします。

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

      ii. テキストエディターで、ブローカーデプロイメントに使用した CR ファイルを開きます。たとえば、これは以前にダウンロードおよび抽出した Operator インストールアーカイブの **deploy/crs** ディレクトリーにある **broker_activemqartemis_cr.yaml** ファイルである可能性があります。
   b. OpenShift Container Platform Web コンソールの使用
      i. ブローカーデプロイメントのプロジェクトに CR を編集およびデプロイする権限を持つユーザーとしてコンソールにログインします。
      ii. 左側のペインで、**Administration** → **Custom Resource Definitions**をクリックします。
      iii. **ActiveMQArtemis** CRD をクリックします。
      iv. **Instances** タブをクリックします。
      v. プロジェクトの名前空間に対応する CR インスタンスを見つけます。
      vi. CR インスタンスの場合は、右側の **More Options** アイコン (3 つの点) をクリックします。**Edit ActiveMQArtemis** を選択します。

コンソールで、YAML エディターが開き、CR インスタンスを編集できるようになります。

2. ブローカーコンテナーイメージをアップグレードする AMQ Broker のバージョンを指定するには、CR の **spec.version** プロパティーの値を設定します。以下に例を示します。

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

3. CR の **spec** セクションで、**upgrades** セクションを見つけます。このセクションが CR に含まれていない場合は、これを追加します。

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

4. **upgrades** セクションに、**enabled** および **minor** プロパティーが含まれていることを確認します。

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

5. 指定されたバージョンの AMQ Broker に基づくブローカーコンテナーイメージのアップグレードを有効にするには、**enabled** プロパティーの値を **true** に設定します。

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

6. ブローカーのアップグレード動作を定義するには、**minor** プロパティーの値を設定します。

   a. AMQ Broker の**マイナー**バージョン間のアップグレードを許可するには、**minor** の値を **true** に設定します。

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

   たとえば、現在のブローカーコンテナーイメージが **7.8.0** に対応し、**spec.version** に指定された **7.9.0** バージョンに対応する新しいイメージが利用できるとします。この場合、Operator は **7.8** マイナーバージョンと **7.9** マイナーバージョン間の利用可能なアップグ

レードがあることを判別します。マイナーバージョン間のアップグレードを可能にする前述の設定に基づいて、Operator によってブローカーのコンテナーイメージがアップグレードされます。

反対に、現在のブローカーコンテナーイメージが**7.9.0** に対応し、**spec.version** に **7.9.1** と**新しい** 値を指定するとします。**7.9.1**に対応するイメージが存在する場合、Operator は、**7.9.0** から **7.9.1**のマイクロバージョンの間で、利用可能なアップグレードがあると判断します。マイナーバージョン間のアップグレードのみを許可する前述の設定に基づいて、Operator はブローカーのコンテナーイメージをアップグレード**しません**。

b. **マイクロ** AMQ Broker バージョン間のアップグレードを許可するには、 **minor** の値を **false** に設定します。

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

たとえば、現在のブローカーコンテナーイメージが **7.8.0** に対応し、**spec.version** に指定された **7.9.0** バージョンに対応する新しいイメージが利用できるとします。この場合、Operator は **7.8** マイナーバージョンと **7.9** マイナーバージョン間の利用可能なアップグレードがあることを判別します。前述の設定に基づいて、マイナーバージョン間のアップグレードを許可しない (マイクロバージョン間のアップグレードのみ)、Operator はブローカーのコンテナーイメージをアップグレード**しません**。

反対に、現在のブローカーコンテナーイメージが**7.9.0** に対応し、**spec.version** に **7.9.1** と**新しい** 値を指定するとします。**7.9.1**に対応するイメージが存在する場合、Operator は、**7.9.0** から **7.9.1**のマイクロバージョンの間で、利用可能なアップグレードがあると判断します。マイクロバージョン間のアップグレードを可能にする前述の設定に基づいて、Operator によってブローカーのコンテナーイメージがアップグレードされます。

7. 変更を CR に適用します。

a. OpenShift コマンドラインインターフェースの使用:

i. CR ファイルを保存します。

ii. ブローカーデプロイメントのプロジェクトに切り替えます。

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

iii. CR を適用します。

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

b. OpenShift Web コンソールの使用

i. CR の編集が完了したら、**Save** をクリックします。

CR の変更を適用する際に、Operator はまず **spec.version** に指定された AMQ Broker バージョンへのアップグレードが利用可能であることを検証します。アップグレードする無効なバージョンの AMQ Broker を指定している場合 (たとえば、まだ利用できないバージョンなど)、Operator は警告メッセージをログに記録し、それ以上のアクションを取ることができません。

ただし、指定されたバージョンにアップグレードでき、**upgrade.enabled** および **upgrades.minor** に指定される値を指定すると、デプロイメントの各ブローカーが、新しい AMQ Broker バージョンに対応するブローカーコンテナーイメージを使用するようになります。

Operator が使用するブローカーコンテナーイメージは、Operator デプロイメントの **operator.yaml** 設定ファイルの環境変数で定義されます。環境変数名には、AMQ Broker バージョンの ID が含まれます。たとえば、環境変数 **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_791** は AMQ Broker 7.9.1 に一致します。

Operator が CR の変更を適用すると、デプロイメントで各ブローカー Pod が再起動し、各 Pod が指定されたイメージバージョンを使用するようにします。デプロイメントに複数のブローカーがある場合、1 つのブローカー Pod のみがシャットダウンし、一度に再起動します。

関連情報

- Operator が環境変数を使用してブローカーコンテナーイメージを選択する方法の詳細は、「Operator によるコンテナーイメージの選択方法」 を参照してください。

第7章 ブローカーの監視

7.1. FUSE CONSOLE でのブローカーの表示

Operator ベースのブローカーのデプロイメントを、AMQ Management Console ではなく OpenShift に Fuse Console を使用するように設定できます。ブローカーのデプロイメントを適切に設定すると、Fuse Console はブローカーを検出し、専用の **Artemis** タブに表示されます。AMQ 管理コンソールで行うのと同じブローカーランタイムデータを表示できます。アドレスやキューの作成など、同じ基本的な管理操作を実行することもできます。

以下の手順では、ブローカーデプロイメントのカスタムリソース(CR)インスタンスを設定して、Fuse Console for Open Shift がデプロイメント内のブローカーを検出して表示できるようにする方法について説明します。

重要

Fuse Console からのブローカーの表示は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、開発プロセスの中でお客様に機能性のテストとフィードバックをしていただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、「テクノロジプレビュー機能のサポート範囲」を参照してください。

前提条件

- Fuse Console for Open Shift は、OCP クラスター、またはそのクラスター上の特定の名前空間にデプロイする必要があります。コンソールを特定の名前空間にデプロイした場合に、コンソールがブローカーを検出できるようにするには、ブローカーのデプロイメントを**同じ**名前空間に配置する必要があります。それ以外の場合は、Fuse Console とブローカーを同じ OCP クラスターにデプロイするだけで十分です。OCP への Fuse Online のインストールの詳細はFuse Online on OpenShift Container Platform のインストールと操作を参照してください。
- ブローカーデプロイメントをすでに作成している必要があります。たとえば、カスタムリソース(CR)インスタンスを使用して、基本的なオペレーターベースのデプロイメントを作成する方法は、「基本的なブローカーインスタンスのデプロイ」を参照してください。

手順

1. ブローカーのデプロイメントに使用した CR インスタンスを開きます。たとえば、基本的なデプロイメントの CR は次のようになります。

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

2. 次に示すように、 **deployment Plan**セクションで、 **jolokia Agent Enabled**プロパティーと**management RBACEnabled**プロパティーを追加し、値を指定します。

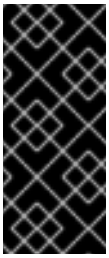
```
apiVersion: broker.amq.io/v2alpha4
kind: ActiveMQArtemis
metadata:
  name: ex-aao
  application: ex-aao-app
spec:
    version: 7.9.3
    deploymentPlan:
        size: 4
        image: registry.redhat.io/amq7/amq-broker-rhel8:7.9
        ...
        jolokiaAgentEnabled: true
        managementRBACEnabled: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

jolokiaAgentEnabled

Fuse Console がデプロイメント内のブローカーのランタイムデータを検出して表示できるかどうかを指定します。Fuse Console を使用するには、値を**true**に設定します。

managementRBACEnabled

デプロイメント内のブローカーに対してロールベースのアクセス制御 (RBAC) を有効にするかどうかを指定します。Fuse Console は独自のロールベースのアクセス制御を使用するため、Fuse Console を使用するには値を**false**に設定する**必要**があります。



重要

managementRBACEnabledの値を**false**に設定して、Fuse Console の使用を有効にすると、ブローカーの管理 MBean では認可が必要なくなります。**management RBACEnabled**が**false**に設定されている間は、ブローカーでの全管理操作が不正に使用される可能性があるので、AMQ 管理コンソールを使用**しないでください**。

3. CR インスタンスを保存します。

4. ブローカーデプロイメントを先に作成したプロジェクトに切り替えます。

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

5. コマンドラインで変更を適用します。

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

6. Fuse Console で、Fuse アプリケーションを表示するには、**オンライン** タブをクリックします。実行中のブローカーを表示するには、左側のナビゲーションメニューで **Artemis** をクリックします。

関連情報

- OpenShift での Fuse Console の使用の詳細は、OpenShift でのRed Hat Fuse アプリケーションの監視と管理 を参照してください。

- Fuse Console と同じ方法で AMQ 管理コンソールを使用してブローカーを表示および管理する場合は、AMQ 管理コンソールを使用したブローカーの管理を参照してください。

7.2. PROMETHEUS を使用したブローカーのランタイムメトリックの監視

以下のセクションでは、OpenShift Container Platform で AMQ Broker の Prometheus メトリクスプラグインを設定する方法について説明します。プラグインを使用して、ブローカーのランタイムメトリックを監視および保存できます。Grafana などのグラフィカルツールを使用して、Prometheus プラグインが収集するデータをさらに詳細にわたり視覚化する設定や、ダッシュボードの設定も行うことができます。

注記

Prometheus メトリクスプラグインを使用すると、ブローカーメトリクスを Prometheus**形式**で収集およびエクスポートできます。ただし、Red Hat では、Prometheus 自体のインストールや構成、または Grafana などの視覚化ツールは、**サポートしていません**。Prometheus または Grafana のインストール、構成、または実行に関するサポートが必要な場合は、製品の Web サイトにアクセスして、コミュニティーのサポートやドキュメントなどのリソースを入手してください。

7.2.1. メトリクスの概要

AMQ Broker の Prometheus プラグインを使用し、ブローカーのランタイムメトリックを監視および保存して、ブローカーインスタンスの正常性とパフォーマンスを監視できます。AMQ Broker Prometheus プラグインは、ブローカーのランタイムメトリックを Prometheus 形式にエクスポートし、Prometheus 自体を使用してデータのクエリを視覚化および実行できるようにします。

Grafana などのグラフィカルツールを使用して、Prometheus プラグインが収集するメトリクスをさらに詳細にわたり視覚化する設定や、ダッシュボードの設定も行うことができます。

プラグインが Prometheus 形式にエクスポートするメトリクスを以下に説明します。

ブローカーのメトリクス

artemis_address_memory_usage

メモリーメッセージ向けに、このブローカの全アドレスにより使用されるバイト数。

artemis_address_memory_usage_percentage

このブローカ上のすべてのアドレスで使用されるメモリを、**global-max-size**パラメータの割合で示したもの。

artemis_connection_count

このブローカーに接続されているクライアントの数。

artemis_total_connection_count

開始してから、このブローカーに接続しているクライアントの数。

アドレスメトリクス

artemis_routed_message_count

1 つ以上のキューバインディングにルーティングされたメッセージの数。

artemis_unrouted_message_count

キューバインディングにルーティングされ**なかった**メッセージの数。

キューメトリクス

artemis_consumer_count

特定のキューからのメッセージを消費しているクライアントの数。

artemis_delivering_durable_message_count

特定のキューが現在コンシューマーに配信している永続メッセージの数。

artemis_delivering_durable_persistent_size

特定のキューが現在コンシューマーに配信している永続メッセージの永続サイズ。

artemis_delivering_message_count

特定のキューが現在コンシューマーに配信しているメッセージの数。

artemis_delivering_persistent_size

特定のキューが現在コンシューマーに配信しているメッセージの永続サイズ。

artemis_durable_message_count

特定のキューに現存する永続メッセージの数。これには、スケジュールされたメッセージ、ページングされたメッセージ、および配信中のメッセージが含まれます。

artemis_durable_persistent_size

現在特定のキューにある永続メッセージの永続サイズ。これには、スケジュールされたメッセージ、ページングされたメッセージ、および配信中のメッセージが含まれます。

artemis_messages_acknowledged

キューが作成されてから、特定のキューから確認応答されたメッセージの数。

artemis_messages_added

キューが作成されてから特定のキューに追加されたメッセージの数。

artemis_message_count

特定のキューに現在あるメッセージの数。これには、スケジュールされたメッセージ、ページングされたメッセージ、および配信中のメッセージが含まれます。

artemis_messages_killed

キューが作成されてからその特定のキューから削除されたメッセージの数。メッセージが設定済みの最大配信試行回数を超えると、ブローカはメッセージを強制終了します。

artemis_messages_expired

キューが作成されてから、その特定のキューから期限切れになったメッセージの数。

artemis_persistent_size

現在特定のキューにある全メッセージ(永続および一時)の永続サイズ。これには、スケジュールされたメッセージ、ページングされたメッセージ、および配信中のメッセージが含まれます。

artemis_scheduled_durable_message_count

特定のキューにあるスケジュールされた永続メッセージの数。

artemis_scheduled_durable_persistent_size

特定のキューにあるスケジュールされた永続メッセージの永続サイズ。

artemis_scheduled_message_count

特定のキューでスケジュールされたメッセージの数。

artemis_scheduled_persistent_size

特定のキューでスケジュールされたメッセージの永続サイズ。

上記にリストされていない上位レベルのブローカーメトリクスについては、下位レベルのメトリクスを集計することで算出できます。たとえば、メッセージの合計数を算出するには、ブローカーデプロイメントのすべてのキューから **artemis_message_count** メトリクスを集約できます。

AMQ Broker のオンプレミスデプロイメントの場合には、ブローカーをホストする Java 仮想マシン (JVM)のメトリクスも Prometheus 形式にエクスポートされます。これは、OpenShift Container Platform での AMQ Broker のデプロイには適用されません。

7.2.2. CR を使用した Prometheus プラグインの有効化

AMQ Broker をインストールすると、Prometheus メトリクスプラグインがインストールに含まれます。有効にすると、プラグインはブローカーのランタイムメトリクスを収集して Prometheus 形式でエクスポートします。

次の手順は、CR を使用して AMQ Broker の Prometheus プラグインを有効にする方法を示しています。この手順は、AMQ Broker7.9 以降の新規および既存のデプロイメントをサポートします。

実行中のブローカーの別の手順は、「環境変数を使用した実行中のブローカーデプロイメントに対する Prometheus プラグインの有効化」 を参照してください。

手順

1. ブローカーのデプロイメントに使用する CR インスタンスを開きます。たとえば、基本的なデプロイメントの CR は次のようになります。

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

2. 次に示すように、 **deployment Plan**セクションで、 **enable Metrics Plugin**プロパティーを追加し、値を**true**に設定します。

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

enableMetricsPlugin

デプロイメント内のブローカーに対して Prometheus プラグインを有効にするかどうかを指定します。

3. CR インスタンスを保存します。

4. ブローカーデプロイメントを先に作成したプロジェクトに切り替えます。

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

5. コマンドラインで変更を適用します。

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

メトリックプラグインは、Prometheus 形式でブローカーランタイムメトリックの収集を開始します。

関連情報

- 実行中のブローカーの更新については、「ブローカーデプロイメントの実行へのカスタムリソース変更の適用」を参照してください。

7.2.3. 環境変数を使用した実行中のブローカーデプロイメントに対する Prometheus プラグインの有効化

次の手順は、環境変数を使用して AMQ Broker の Prometheus プラグインを有効にする方法を示しています。別の手順は、「CR を使用した Prometheus プラグインの有効化」 を参照してください。

前提条件

- AMQ Broker Operator で作成されたブローカー Podの Prometheus プラグインを有効にできます。ただし、デプロイされたブローカーは、AMQ Broker 7.7 以降のブローカーコンテナーイメージを使用する必要があります。

手順

1. ブローカーのデプロイメントなどのプロジェクトに対する管理者権限で、OpenShift Container Platform Web コンソールにログインします。
2. Web コンソールで、**Home → Projects** をクリックします。ブローカーのデプロイメントが含まれるプロジェクトを選択します。
3. プロジェクトの StatefulSets または DeploymentConfigs を表示するには、**Workloads → StatefulSets** または **Workloads → DeploymentConfigs** をクリックします。
4. ブローカーのデプロイメントに対応する StatefulSet または DeploymentConfig をクリックします。
5. ブローカーデプロイメントの環境変数にアクセスするには、**環境** タブをクリックします。
6. 新しい環境変数**AMQ_ENABLE_METRICS_PLUGIN**を追加します。変数の値を**true**に設定します。
   AMQ_ENABLE_METRICS_PLUGIN環境変数を設定すると、OpenShift は StatefulSet または DeploymentConfig で各ブローカー Pod を再起動します。デプロイメントに複数の Pod がある場合、OpenShift は各 Pod を順番に再起動します。各ブローカー Pod が再起動すると、そのブローカーの Prometheus プラグインがブローカーのランタイムメトリックの収集を開始します。

7.2.4. 実行中のブローカー Pod の Prometheus メトリクスへのアクセス

以下の手順では、実行中のブローカー Pod の Prometheus メトリクスにアクセスする方法を説明します。

前提条件

- ブローカー Pod の Prometheus プラグインを有効にしておく必要があります。「環境変数を使用した実行中のブローカーデプロイメントに対する Prometheus プラグインの有効化」 を参照してください。

手順

1. メトリクスのアクセス先のブローカー Pod では、以前に Pod への接続用に作成したルートを特定して、AMQ Broker 管理コンソールに接続する必要があります。メトリクスへのアクセスに必要な URL の一部に、ルート名が含まれます。

   a. **Networking → Routes** をクリックします。

   b. 選択したブローカー Pod で、AMQ Broker 管理コンソールへのPod の接続用に作成されたルートを特定します。**ホスト名**に表示される完全な URL をメモします。以下に例を示します。

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

2. Prometheus メトリクスにアクセスするには、Web ブラウザーで、先程メモをしたルート名に /**metrics**が付けて入力します。以下に例を示します。

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

注記

コンソール設定で SSL を使用しない場合は、URL に**http**を指定してください。この場合、ホスト名の DNS が解決されて、トラフィックは OpenShift ルーターのポート 80 に転送されます。コンソール設定で SSL を使用する場合は、URL に**https**を指定します。この場合、ブラウザーはデフォルトで OpenShift ルーターのポート 443 になります。この設定により、OpenShift ルーターが SSL トラフィックにポート 443 も使用する場合には、コンソールに正常に接続できます (これは、ルーターのデフォルト設定)。

7.3. JMX を使用したブローカーランタイムデータの監視

この例では、JMX への Jolokia REST インターフェイスを使用してブローカーをモニターする方法を説明します。

前提条件

- 基本的なブローカーのデプロイ を完了しておくことをお勧めします。

手順

1. 実行中の Pod のリストを取得します。

   ```
   $ oc get pods

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

2. **oclogs**コマンドを実行します。

```
$ 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. クエリーを実行して、ブローカーの**Max Consumers**を監視します。

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

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

第8章 リファレンス

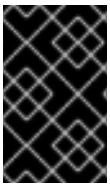
8.1. カスタムリソース設定リファレンス

カスタムリソース定義(CRD)は、オペレーターでデプロイされたカスタム OpenShift オブジェクトの設定項目のスキーマです。対応するカスタムリソース(CR)インスタンスをデプロイして、CRD に表示される設定アイテムの値を指定します。

次のサブセクションでは、メインブローカー CRD に基づいてカスタムリソースインスタンスで設定できる設定項目について詳説します。

8.1.1. ブローカーカスタムリソース設定リファレンス

メインブローカー CRD に基づく CR インスタンスを使用すると、ブローカーを設定して OpenShift プロジェクトにデプロイできます。次の表に、CR インスタンスで設定できる項目を示します。



重要

アスタリスク(*)でマークされた設定アイテムは、該当するカスタムリソース(CR)でデプロイに必要です。不要なアイテムの値を明示的に指定しない場合には、設定にデフォルト値が使用されます。

エントリー	サブエントリー	説明と使用法
adminUser*		ブローカーおよび管理コンソールの接続に必要な管理者ユーザー名。 値を指定しない場合、値は自動的に生成され、シークレットに保存されます。デフォルトのシークレット名の形式は、**<custom_resource_name>-credentials-secret**です。例: **my-broker-deployment-credentials-secret**。 **型**: String **例**: my-user **デフォルト値**: 無作為に、自動生成された値

エントリー	サブエントリー	説明と使用法
adminPassword*		ブローカーおよび管理コンソールへの接続に必要な管理者パスワード。 値を指定しない場合、値は自動的に生成され、シークレットに保存されます。デフォルトのシークレット名の形式は、**<custom_resource_name>-credentials-secret**です。例: **my-broker-deployment-credentials-secret**。 **型**: String **例**: my-password **デフォルト値**: 無作為に、自動生成された値
deploymentPlan*		ブローカーのデプロイメント設定

エントリー	サブエントリー	説明と使用法
	image*	デプロイメントの各ブローカーに使用されるブローカーコンテナーイメージの完全パス。 CR で**image**の値を明示的に指定する必要はありません。**プレースホルダー**のデフォルト値は、Operator が使用する適切なイメージをまだ決定していないことを示します。 Operator が使用するブローカーコンテナーイメージを選択する方法は、「Operator によるコンテナーイメージの選択方法」を参照してください。 **型**: String **例**: registry.redhat.io/amq7/amq-broker-rhel8@sha256:979b59325aa0f34eb05625201beba53fccbb83bd5eb80a89dcb5261ae358138f **デフォルト値**: placeholder
	size*	デプロイメントで作成するブローカー Pod の数。 2 以上の値を指定すると、ブローカーのデプロイメントはデフォルトでクラスター化されます。クラスターのユーザー名とパスワードは自動的に生成され、同じシークレットに保存されます (デフォルトで**admin User**および**admin Password**)。 **型**: int **例**: 1 **デフォルト値**: 2

エントリー	サブエントリー	説明と使用法
	requireLogin	ブローカーへの接続にログイン資格情報が必要かどうかを指定します。 **型**: Boolean **例**: false **デフォルト値**: true
	persistenceEnabled	デプロイメントでブローカー Pod ごとにジャーナルストレージを使用するかどうかを指定します。**true**に設定されている場合には、各ブローカー Pod には、Operator が永続ボリューム要求(PVC)を使用して要求できる永続ボリューム(PV) に空きがなければなりません。 **型**: Boolean **例**: false **デフォルト値**: true

エントリー	サブエントリー	説明と使用法
	initImage	ブローカーの設定に使用される Init Container イメージ。 カスタムイメージを提供する場合を除いて、CR で**init Image**の値を明示的に指定する必要はありません。 Operator が使用する組み込みの Init Container イメージの選択方法は、「Operator によるコンテナーイメージの選択方法」を参照してください。 **カスタム** の Init Container イメージを指定する方法は、「カスタム Init コンテナーイメージの指定」 を参照してください。 **型**: String **例**: registry.redhat.io/amq7/amq-broker-init-rhel8@sha256:b74d03ed852a3731467ffda95266ce49f2065972f1c37bf254f3d52b34c11991 **デフォルト値**: 指定なし
	journalType	非同期 I/O (AIO) と非ブロッキング I/O(NIO)のどちらを使用するかを指定します。 **型**: String **例**: aio **デフォルト値**: nio

エントリー	サブエントリー	説明と使用法
	messageMigration	ブローカーデプロイメントの失敗や、意図的にスケールダウンしたことでブローカー Pod がシャットダウンした場合には、ブローカークラスターでまだ実行中の別のブローカー Pod にメッセージを移行するかどうかを指定します。 **型**: Boolean **例**: false **デフォルト値**: true
	resources.limits.cpu	デプロイメントの Pod で実行されている各ブローカーコンテナーが消費できるホストノード CPU の最大量 (ミリコア単位)。 **型**: String **Example**: "500m" **デフォルト値**: お使いのバージョンの OpenShift Container Platform と同じデフォルト値を使用します。クラスター管理者に相談してください。
	resources.limits.memory	デプロイメント内のポッドで実行されている各ブローカーコンテナーが消費できるホストノードメモリーの最大量 (バイト単位)。バイト表記 (K、M、G など)、または同等のバイナリー (Ki、Mi、Gi) をサポートします。 **型**: String **Example**: "1024M" **デフォルト値**: お使いのバージョンの OpenShift Container Platform と同じデフォルト値を使用します。クラスター管理者に相談してください。

エントリー	サブエントリー	説明と使用法
	resources.requests.cpu	デプロイメント内の Pod で実行されている各ブローカーコンテナーが明示的に要求するホストノードの CPU 量(ミリコア単位)。 **型**: String **Example**: "250m" **デフォルト値**: お使いのバージョンの OpenShift Container Platform と同じデフォルト値を使用します。クラスター管理者に相談してください。
	resources.requests.memory	デプロイメント内の Pod で実行されている各ブローカーコンテナーが明示的に要求するホストノードメモリーの量(バイト単位)。バイト表記 (K、M、G など)、または同等のバイナリー (Ki、Mi、Gi) をサポートします。 **型**: String **Example**: "512M" **デフォルト値**: お使いのバージョンの OpenShift Container Platform と同じデフォルト値を使用します。クラスター管理者に相談してください。

エントリー	サブエントリー	説明と使用法
	storage.size	デプロイメントにある各ブローカーが永続ストレージに必要な Persistent Volume Claim (永続ボリューム要求、PVC) のサイズ (バイト単位)。このプロパティーは、**persistenceEnabled** が **true** に設定されている場合にのみ適用されます。指定する値には単位が含まれている**必要があります**。バイト表記 (K、M、G など)、または同等のバイナリー (Ki、Mi、Gi) をサポートします。 **型**: String **例**: 4Gi **デフォルト値**: 2Gi
	jolokiaAgentEnabled	デプロイメント内のブローカーに対して Jolokia JVM エージェントを有効にするかどうかを指定します。このプロパティーの値が**true**に設定されている場合には、Fuse Console はブローカーのランタイムデータを検出して表示できます。 **型**: Boolean **例**: True **デフォルト値**: false

エントリー	サブエントリー	説明と使用法
	managementRBACEnabled	デプロイメント内のブローカーに対してロールベースのアクセス制御 (RBAC) を有効にするかどうかを指定します。Fuse Console を使用するには、値を**false**に設定する**必要**があります。これは、Fuse Console が独自のロールベースのアクセス制御を使用しているためです。 **型**: Boolean **例**: false **デフォルト値**: true
コンソール		ブローカー管理コンソールの設定。
	expose	デプロイメントの各ブローカーの管理コンソールポートを公開するかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	sslEnabled	管理コンソールポートで SSL を使用するかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false

エントリー	サブエントリー	説明と使用法
	sslSecret	ブローカーキーストア、トラストストア、および対応するパスワード (すべて Base64 でエンコードされたもの) が保存されるシークレット。**ssl Secret**の値を指定しない場合には、コンソールはデフォルトのシークレット名を使用します。デフォルトのシークレット名は、**<custom_resource_name> -console-secret**の形式です。 **型**: String **例**: my-broker-deployment-console-secret **デフォルト値**: 指定なし
	useClientAuth	管理コンソールにクライアント認証が必要かどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
acceptors.acceptor		単一のアクセプターの設定インスタンス。
	name*	アクセプターの名前。 **型**: String **例**: my-acceptor **デフォルト値**: 該当なし

エントリー	サブエントリー	説明と使用法
	ポート	アクセプターインスタンスに使用するポート番号。 **型**: int **例**: 5672 **デフォルト値**: 61626 (定義する最初のアクセプター)。その後、デフォルト値は、定義する後続のアクセプターごとに 10 ずつ増えます。
	protocols	アクセプターインスタンスで有効にするメッセージングプロトコル。 **型**: String **例**: amqp、core **デフォルト値**: all
	sslEnabled	アクセプターポートで SSL を有効にするかどうかを指定します。**true** に設定されている場合は、 **ssl Secret**で指定されているシークレット名を調べて、TLS/SSL に必要な資格情報を探します。 **型**: Boolean **例**: True **デフォルト値**: false

エントリー	サブエントリー	説明と使用法
	sslSecret	ブローカーキーストア、トラストストア、および対応するパスワード (すべて Base64 でエンコードされたもの) が保存されるシークレット。 **ssl Secret** にカスタムシークレット名を指定しない場合には、アクセプターはデフォルトのシークレット名を想定します。デフォルトのシークレット名の形式は、**<custom_resource_name>-<acceptor_name>-secret**です。 アクセプターでデフォルト名が必要であっても、常にこのシークレットを自分で作成する必要があります。 **型**: String **例**: my-broker-deployment-my-acceptor-secret **Default value**: **<custom_resource_name>-<acceptor_name>**-secret

エントリー	サブエントリー	説明と使用法
	enabledCipherSuites	TLS/SSL 通信に使用する暗号スイートのコンマ区切りリスト。 クライアントアプリケーションでサポートする最も安全な暗号スイートを指定します。コンマ区切りのリストを使用して、ブローカーとクライアントの両方に共通の暗号スイートのセットを指定する場合、または暗号スイートを指定しない場合には、ブローカーとクライアントは、使用する暗号スイートについて相互に交渉します。指定する暗号スイートがわからない場合は、最初にデバッグモードで実行されているクライアントとのブローカークライアント接続を確立して、ブローカーとクライアントの両方に共通の暗号スイートを確認することをお勧めします。次に、ブローカーで **enabledCipherSuites**を設定します。 **型**: String **デフォルト値**: 指定なし
	enabledProtocols	TLS/SSL 通信に使用するプロトコルのコンマ区切りリスト。 **型**: String **例**: TLSv1、TLSv1.1、TLSv1.2 **デフォルト値**: 指定なし

エントリー	サブエントリー	説明と使用法
	needClientAuth	ブローカーがアクセプターで双方向 TLS が必要であることをクライアントに通知するかどうかを指定します。このプロパティーは、**wantClientAuth**をオーバーライドします。 **型**: Boolean **例**: True **デフォルト値**: 指定なし
	wantClientAuth	アクセプターで双方向 TLS が**要求**されていることを通知するかどうかを指定します。ただし、必須ではありません。このプロパティーは**needClientAuth**にオーバーライドされます。 **型**: Boolean **例**: True **デフォルト値**: 指定なし
	verifyHost	クライアントの証明書のコモンネーム (CN) をホスト名と比較して一致することを確認するかどうかを指定します。このオプションは、双方向 TLS が使用されている場合にのみ適用されます。 **型**: Boolean **例**: True **デフォルト値**: 指定なし
	sslProvider	SSL プロバイダーが JDK であるか OPENSSL であるかを指定します。 **型**: String **例**: OPENSSL **デフォルト値**: JDK

エントリー	サブエントリー	説明と使用法
	sniHost	受信接続の **server_name** の拡張子と照合するための正規表現。名前が一致しない場合には、アクセプターへの接続は拒否されます。 **型**: String **例**: some_regular_expression **デフォルト値**: 指定なし
	expose	Open Shift Container Platform の外部のクライアントにアクセプターを公開するかどうかを指定します。 OpenShift 外部にあるクライアントにアクセプターを公開すると、Operator はデプロイメント内のブローカー Pod ごとに専用のサービスとルートを自動作成します。 **型**: Boolean **例**: True **デフォルト値**: false
	anycastPrefix	**anycast** ルーティングタイプを使用することを指定するためにクライアントが使用するプレフィックス。 **型**: String **例**: jms.queue **デフォルト値**: 指定なし

エントリー	サブエントリー	説明と使用法
	multicastPrefix	**multicast** ルーティングタイプを使用する必要があることを指定するためにクライアントが使用するプレフィックス。 **型**: String **例**: /topic / **デフォルト値**: 指定なし
	connectionsAllowed	アクセプターで許可されている接続の数。この制限に達すると、DEBUG メッセージがログに出力され、接続は拒否されました。使用中のクライアントのタイプによって、接続が拒否されたときに何が起こるかが決まります。 **型**: integer **例**: 2 **デフォルト値**: 0 (無制限の接続)

エントリー	サブエントリー	説明と使用法
	amqpMinLargeMessageSize	ブローカーが AMQP メッセージを大きなメッセージとして処理するために必要な最小メッセージサイズ (バイト単位)。AMQ メッセージのサイズがこの値以上の場合は、ブローカーはメッセージを、メッセージストレージ用にブローカーが使用する永続ボリューム (PV)の永続ボリューム (デフォルトでは **/opt/<custom_resource_name>/data/large-messages**) にメッセージを保存します。値を**-1**に設定すると、AMQP メッセージの大きなメッセージ処理が無効になります。 **型**: integer **例**: 204800 **デフォルト値**: 102400(100 KB)
connectors.connector		単一のコネクター構成インスタンス。
	name*	コネクターの名前。 **型**: String **例**: my-connector **デフォルト値**: 該当なし
	type	作成するコネクターのタイプ。 **tcp**または**vm**。 **型**: String **例**: vm **デフォルト値**: tcp

エントリー	サブエントリー	説明と使用法
	host*	接続するホスト名または IP アドレス。 **型**: String **例**: 192.168.0.58 **デフォルト値**: 指定なし
	port*	コネクターインスタンスに使用されるポート番号。 **型**: int **例**: 22222 **デフォルト値**: 指定なし
	sslEnabled	コネクターポートで SSL を有効にするかどうかを指定します。**true** に設定されている場合は、 **ssl Secret**で指定されているシークレット名を調べて、TLS/SSL に必要な資格情報を探します。 **型**: Boolean **例**: True **デフォルト値**: false

エントリー	サブエントリー	説明と使用法
	sslSecret	ブローカーキーストア、トラストストア、および対応するパスワード (すべて Base64 でエンコードされたもの) が保存されるシークレット。 **ssl Secret** にカスタムシークレット名を指定しない場合には、コネクターはデフォルトのシークレット名を想定します。デフォルトのシークレット名の形式は、**<custom_resource_name>-<connector_name>-secret**です。 コネクターでデフォルト名が必要であっても、このシークレットは常に自分で作成する必要があります。 **型**: String **例**: my-broker-deployment-my-connector-secret **Default value**: **<custom_resource_name>-<connector_name>**-secret
	enabledCipherSuites	TLS/SSL 通信に使用する暗号スイートのコンマ区切りリスト。 **型**: String **注**: コネクターの場合、暗号スイートのリストを指定**しない**ことをお勧めします。 **デフォルト値**: 指定なし

エントリー	サブエントリー	説明と使用法
	enabledProtocols	TLS/SSL 通信に使用するプロトコルのコンマ区切りリスト。 **型**: String **例**: TLSv1、TLSv1.1、TLSv1.2 **デフォルト値**: 指定なし
	needClientAuth	コネクターに双方向 TLS が必要であることをブローカがクライアントに通知するかどうかを指定します。このプロパティーは、**wantClientAuth**をオーバーライドします。 **型**: Boolean **例**: True **デフォルト値**: 指定なし
	wantClientAuth	コネクターで双方向 TLS が**要求**されていることを通知するかどうかを指定します。ただし、必須ではありません。このプロパティーは**needClientAuth**にオーバーライドされます。 **型**: Boolean **例**: True **デフォルト値**: 指定なし
	verifyHost	クライアントの証明書のコモンネーム (CN) をホスト名と比較して一致することを確認するかどうかを指定します。このオプションは、双方向 TLS が使用されている場合にのみ適用されます。 **型**: Boolean **例**: True **デフォルト値**: 指定なし

エントリー	サブエントリー	説明と使用法
	sslProvider	SSL プロバイダーが**JDK**であるか**OPENSSL**であるかを指定します。 **型**: String **例**: OPENSSL **デフォルト値**: JDK
	sniHost	送信接続の**server_name**拡張子と照合するための正規表現。名前が一致しない場合には、コネクター接続は拒否されます。 **型**: String **例**: some_regular_expression **デフォルト値**: 指定なし
	expose	OpenShift Container Platform 外のクライアントにコネクターを公開するかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
addressSettings.applyRule		Operator を一致するアドレスまたはアドレスのセットごとに CR に追加する設定を適用する方法を指定します。 指定できる値は次のとおりです。 **merge_all** CR で指定されるアドレス設定**と**、同じアドレスまたはアドレスのセットに一致するデフォルト設定の両方の場合: • デフォルト設定で指定され

エントリー	サブエントリー	説明と使用法
		るプロパティー値を CR で指定されたプロパティー値に置き換えます。 ● CR **または**デフォルト設定で一意で指定されるプロパティー値を保持します。これらはそれぞれ最終マージされた設定の組み込みます。 CR で指定されるアドレス設定**または**特定のアドレスセットに一意になるデフォルト設定の場合は、これらを最終でマージされた設定に含めます。 **merge_replace** CR に指定されたアドレス設定**と**、同じアドレスまたはアドレスセットに一致するデフォルト設定について、最終的なマージされた設定の **CR** に指定された設定を含めます。それらのプロパティーが CR で指定されていない場合でも、デフォルト設定に指定されたプロパティーを**含めないで**ください。 CR で指定されるアドレス設定**または**特定のアドレスセットに一意になるデフォルト設定の場合は、これらを最終でマージされた設定に含めます。 **replace_all** デフォルト設定に指定された**すべて**のアドレス設定を CR で指定されたアドレス設定に置き換えます。最後にマージされた設定は、CR で指定したものと完全に対応します。 **型**: String **例**: replace_all

エントリー	サブエントリー	デフォルト値: merge_all 説明と使用法
addressSettings.addressSetting		一致するアドレスまたはアドレスの **セット** の設定。
	addressFullPolicy	**maxSizeBytes**で構成されたアドレスがいっぱいになったときにどうなるかを指定します。利用可能なポリシーは以下のとおりです。 **PAGE** 完全アドレスに送信されたメッセージはディスクにページングされます。 **DROP** 完全アドレスに送信されたメッセージは通知なしに破棄されます。 **FAIL** 完全アドレスに送信されたメッセージはドロップされ、メッセージプロデューサーでは例外が発生します。 **BLOCK** メッセージプロデューサーは、それ以上メッセージを送信しようとするとブロックします。 BLOCK ポリシーは、AMQP、OpenWire、およびコアプロトコルでのみ機能します。これらのプロトコルはフロー制御をサポートしているためです。 **型**: String **例**: DROP **デフォルト値**: PAGE

エントリー	サブエントリー	説明と使用法
	autoCreateAddresses	クライアントが、存在しないアドレスにバインドされているキューにメッセージを送信するとき、またはキューからメッセージを消費しようとするときに、ブローカーが自動的にアドレスを作成するかどうかを指定します。 **型**: Boolean **例**: false **デフォルト値**: true
	autoCreateDeadLetterResources	ブローカーがデッドレターアドレスおよびキューを自動的に作成し、未配信メッセージを受信するかどうかを指定します。 パラメーターが **true** に設定されている場合には、ブローカはデッドレターアドレスと関連するデッドレターキューを自動的に作成します。自動的に作成されたアドレスの名前は、**dead Letter Address**に指定した値と一致します。 **型**: Boolean **例**: True **デフォルト値**: false

エントリー	サブエントリー	説明と使用法
	autoCreateExpiryResources	期限切れのメッセージを受信するため、ブローカーがアドレスとキューを自動的に作成するかどうかを指定します。 パラメーターが**true**に設定されている場合、ブローカーは自動的に有効期限アドレスと関連する有効期限キューを作成します。自動的に作成されたアドレスの名前は、 **expiry Address**に指定した値と一致します。 **型**: Boolean **例**: True **デフォルト値**: false
	autoCreateJmsQueues	このプロパティーは非推奨にされています。代わりに**autoCreateQueues**を使用してください。
	autoCreateJmsTopics	このプロパティーは非推奨にされています。代わりに**autoCreateQueues**を使用してください。
	autoCreateQueues	クライアントがまだ存在していないキューにメッセージを送信するとき、またはキューからメッセージを消費しようとするときに、ブローカーが自動的にキューを作成するかどうかを指定します。 **型**: Boolean **例**: false **デフォルト値**: true

エントリー	サブエントリー	説明と使用法
	autoDeleteAddresses	ブローカーにキューがなくなったときに、ブローカーが自動的に作成されたアドレスを自動的に削除するかどうかを指定します。 **型**: Boolean **例**: false **デフォルト値**: true
	autoDeleteAddressDelay	アドレスにキューがない場合に、ブローカーが自動作成されたアドレスを自動削除するまで待機する時間 (ミリ秒単位)。 **型**: integer **例**: 100 **デフォルト値**: 0
	autoDeleteJmsQueues	このプロパティーは非推奨にされています。代わりに**autoDeleteQueues**を使用してください。
	autoDeleteJmsTopics	このプロパティーは非推奨にされています。代わりに**autoDeleteQueues**を使用してください。
	autoDeleteQueues	キューにコンシューマーとメッセージがない場合に、ブローカーが自動作成されたキューを自動削除するかどうかを指定します。 **型**: Boolean **例**: false **デフォルト値**: true

エントリー	サブエントリー	説明と使用法
	autoDeleteCreatedQueues	キューにコンシューマーとメッセージがない場合に、ブローカーが手動で作成されたキューを自動削除するかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	autoDeleteQueuesDelay	キューにコンシューマーがない場合に、ブローカーが自動作成されたキューを自動削除するまで待機する時間(ミリ秒単位)。 **型**: integer **例**: 10 **デフォルト値**: 0
	autoDeleteQueuesMessageCount	ブローカーがキューを自動的に削除できるかどうかを評価する前に、キューに入れることができるメッセージの最大数。 **型**: integer **例**: 5 **デフォルト値**: 0

エントリー	サブエントリー	説明と使用法
	configDeleteAddresses	設定ファイルを再読み込みすると、このパラメーターで、設定ファイルから削除されたアドレス（とそのキュー）を処理する方法を指定します。以下の値を指定できます。 **OFF** 設定ファイルの再読み込み時には、ブローカーはアドレスを削除しません。 **FORCE** ブローカは、設定ファイルの再読み込み時にアドレスとそのキューを削除します。キューにメッセージがある場合は、それらも削除されます。 **型**: String **例**: FORCE **デフォルト値**: OFF
	configDeleteQueues	設定ファイルを再読み込みすると、この設定は、ブローカーが設定ファイルから削除されたキューを処理する方法を指定します。以下の値を指定できます。 **OFF** 設定ファイルの再読み込み時には、ブローカーはキューを削除しません。 **FORCE** 設定ファイルの再読み込み時には、ブローカーはキューを削除します。キューにメッセージがある場合は、それらも削除されます。 **型**: String **例**: FORCE **デフォルト値**: OFF

エントリー	サブエントリー	説明と使用法
	deadLetterAddress	ブローカーが未達の (**未配信**) メッセージを送信するアドレス。 **型**: String **例**: DLA **デフォルト値**: None
	deadLetterQueuePrefix	ブローカーにより、自動作成されたデッドレターキューの名前に適用されるプレフィックス。 **型**: String **例**: my DLQ。 **デフォルト値**: DLQ.
	deadLetterQueueSuffix	ブローカーにより、自動作成されたデッドレターキューに適用されるサフィックス。 **型**: String **例**: .DLQ **デフォルト値**: None
	defaultAddressRoutingType	自動作成されたアドレスで使用されるルーティングタイプ。 **型**: String **例**: ANYCAST **デフォルト値**: MULTICAST
	defaultConsumersBeforeDispatch	アドレスのキューに対してメッセージディスパッチを開始する前に必要なコンシューマーの数。 **型**: integer **例**: 5 **デフォルト値**: 0

エントリー	サブエントリー	説明と使用法
	defaultConsumerWindowSize	コンシューマーのデフォルトのウィンドウサイズ (バイト単位)。 **型**: integer **例**: 300000 **デフォルト値**: 1048576 (1024*1024)
	defaultDelayBeforeDispatch	**defaultConsumersBeforeDispatch**に指定された値に達していない場合に、ブローカーがメッセージをディスパッチするまで待機するデフォルトの時間(ミリ秒単位)。 **型**: integer **例**: 5 **デフォルト値**: -1(遅延なし)
	defaultExclusiveQueue	アドレス上のすべてのキューがデフォルトで独占キューであるかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	defaultGroupBuckets	メッセージのグループ化に使用するバケットの数。 **型**: integer **例**: 0(メッセージのグループ化は無効) **デフォルト値**: -1(制限なし)

エントリー	サブエントリー	説明と使用法
	defaultGroupFirstKey	グループ内のどのメッセージが最初であるかをコンシューマーに示すために使用されるキー。 **型**: String **Example**: firstMessageKey **デフォルト値**: None
	defaultGroupRebalance	新しいコンシューマーがブローカーに接続するときにグループのリバランスするかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	defaultGroupRebalancePauseDispatch	ブローカーがグループのリバランスをしている間、メッセージのディスパッチを一時停止するかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	defaultLastValueQueue	アドレス上のすべてのキューがデフォルトで最後の値のキューであるかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	defaultLastValueKey	最後の値のキューに使用するデフォルトのキー。 **型**: String **例**: stock_ticker **デフォルト値**: None

エントリー	サブエントリー	説明と使用法
	defaultMaxConsumers	任意のタイミングでキューで許可されるコンシューマーの最大数。 **型**: integer **例**: 100 **デフォルト値**: -1(制限なし)
	defaultNonDestructive	アドレス上のすべてのキューがデフォルトで non-destructive であるかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	defaultPurgeOnNoConsumers	コンシューマーがなくなったときにブローカーがキューの内容をパージするかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	defaultQueueRoutingType	自動作成されたキューで使用されるルーティングタイプ。デフォルト値は **MULTICAST** です。 **型**: String **例**: ANYCAST **デフォルト値**: MULTICAST

エントリー	サブエントリー	説明と使用法
	defaultRingSize	リングサイズが明示的に設定されていない、一致するキューのデフォルトのリングサイズ。 **型**: integer **例**: 3 **デフォルト値**: -1(サイズ制限なし)
	enableMetrics	Prometheus プラグインなどの設定されたメトリクスプラグインが一致するアドレスまたはアドレスのセットのメトリクスを収集するかどうかを指定します。 **型**: Boolean **例**: false **デフォルト値**: true
	expiryAddress	期限切れのメッセージを受信するアドレス。 **型**: String **Example**: myExpiryAddress **デフォルト値**: None
	expiryDelay	デフォルトの有効期限を使用しているメッセージに適用される有効期限(ミリ秒単位)。 **型**: integer **例**: 100 **デフォルト値**: -1(有効期限は適用されません)

エントリー	サブエントリー	説明と使用法
	expiryQueuePrefix	ブローカーが自動作成された期限切れキューの名前に適用されるプレフィックス。 **型**: String **Example**: myExp. **デフォルト値**: EXP.
	expiryQueueSuffix	ブローカーが自動作成された期限切れキューの名前に適用されるサフィックス。 **型**: String **例**: .EXP **デフォルト値**: None
	lastValueQueue	キューが最後の値のみを使用するかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	managementBrowsePageSize	管理リソースが参照できるメッセージの数を指定します。 **型**: integer **例**: 100 **デフォルト値**: 200

エントリー	サブエントリー	説明と使用法
	match*	アドレス設定と、ブローカーで構成されたアドレスとを照合する文字列。正確なアドレス名を指定するか、ワイルドカード式を使用してアドレス設定をアドレスの**セット**と照合できます。 ワイルドカード式を **match** プロパティーの値として使用する場合には、値を単一引用符で囲む必要があります (例: **'myOAddresses*'**)。 **型**: String **Example**: 'myAddresses*' **デフォルト値**: None
	maxDeliveryAttempts	設定されたデッドレターアドレスにメッセージを送信するまでに、ブローカがメッセージの配信を試行する回数を指定します。 **型**: integer **例**: 20 **デフォルト値**: 10
	maxExpiryDelay	この値より大きい有効期限を使用しているメッセージに適用される有効期限(ミリ秒単位)。 **型**: integer **例**: 20 **デフォルト値**: -1(最大有効期限は適用されません)

エントリー	サブエントリー	説明と使用法
	maxRedeliveryDelay	ブローカーによるメッセージの再配信試行間の最大値(ミリ秒単位)。 **型**: integer **例**: 100 **デフォルト値**: デフォルト値は、**redelivery Delay**の値の 10 倍です。デフォルト値は**0**。
	maxSizeBytes	アドレスの最大メモリーサイズ(バイト単位)。**address Full Policy**が**PAGING**、**BLOCK**、または**FAIL**に設定されている場合に使用されます。K、Mb、GB などのバイト表記もサポートしています。 **型**: String **例**: 10Mb **デフォルト値**: -1(制限なし)
	maxSizeBytesRejectThreshold	ブローカーがメッセージを拒否する前にアドレスが到達できる最大サイズ（バイト単位）。**address-full-policy**が**BLOCK**に設定されている場合に使用されます。AMQP プロトコルの場合のみ**maxSizeBytes**と組み合わせて動作します。 **型**: integer **例**: 500 **デフォルト値**: -1(最大サイズなし)

エントリー	サブエントリー	説明と使用法
	messageCounterHistoryDayLimit	ブローカーがアドレスのメッセージカウンター履歴を保持する日数。 **型**: integer **例**: 5 **デフォルト値**: 0
	minExpiryDelay	この値よりも短い有効期限を使用しているメッセージに適用される有効期限(ミリ秒単位)。 **型**: integer **例**: 20 **デフォルト値**: -1(最小有効期限は適用されません)
	pageMaxCacheSize	ページングナビゲーション中に I/O を最適化するためにメモリー内に保持するページファイルの数。 **型**: integer **例**: 10 **デフォルト値**: 5
	pageSizeBytes	ページングサイズ (バイト単位)。**K**、**Mb**、**GB**などのバイト表記もサポートします。 **型**: String **例**: 20971520 **デフォルト値**: 10485760(約 10.5 MB)

エントリー	サブエントリー	説明と使用法
	redeliveryDelay	キャンセルされたメッセージを再配信する前にブローカーが待機する時間(ミリ秒単位)。 **型**: integer **例**: 100 **デフォルト値**: 0
	redeliveryDelayMultiplier	**redelivery Delay**の値に適用する倍率。 **型**: number **例**: 5 **デフォルト値**: 1
	redeliveryCollisionAvoidanceFactor	競合を回避するために**redelivery Delay**の値に適用する倍率。 **型**: number **例**: 1.1 **デフォルト値**: 0
	redistributionDelay	キューの最後のコンシューマーが閉じられてから残りのメッセージを再分配するまでブローカーが待機する時間 (ミリ秒単位) を定義します。 **型**: integer **例**: 100 **デフォルト値**: -1(未設定)
	retroactiveMessageCount	アドレスに今後作成されるキューに対して保持するメッセージの数。 **型**: integer **例**: 100 **デフォルト値**: 0

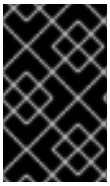
エントリー	サブエントリー	説明と使用法
	sendToDlaOnNoRoute	キューにルーティングされないメッセージは、設定済みのデッドレターアドレスアドレスに送信されます。 **型**: Boolean **例**: True **デフォルト値**: false
	slowConsumerCheckPeriod	コンシューマーが遅いかどうかをブローカーがチェックする頻度 (**秒単位**)。 **型**: integer **例**: 15 **デフォルト値**: 5
	slowConsumerPolicy	低速なコンシューマーが特定されたときにどうするのかを指定します。有効なオプションは **KILL** または **NOTIFY** です。**KILL**はコンシューマーの接続を強制終了します。これは、同じ接続を使用するすべてのクライアントスレッドに影響を与えます。**NOTIFY** は **CONSUMER_SLOW** 管理通知をクライアントに送信します。 **型**: String **例**: KILL **デフォルト値**: NOTIFY

エントリー	サブエントリー	説明と使用法
	slowConsumerThreshold	最小限許可されるメッセージ消費速度 (秒単位)。この値を下回るとコンシューマーは遅いと見なされます。 **型**: integer **例**: 100 **デフォルト値**: -1(未設定)
upgrades		
	enabled	**version** の値を更新して、AMQ Broker の新しいターゲットを指定する場合は、Operator が **deploymentPlan.image**の値をそのバージョンの AMQ Broker に対応するブローカーコンテナーイメージに自動更新できるようにするかどうかを指定します。 **型**: Boolean **例**: True **デフォルト値**: false
	minor	バージョンの値を AMQ Broker のあるマイナーバージョンから別の **マイナー バージョン** に更新する際に、Operator が **deploymentPlan.image** 値を自動的に更新することを許可するかどうかを指定します（例： 7 **.8.0** から 7. **9.3** ）。 **型**: Boolean **例**: True **デフォルト値**: false

エントリー	サブエントリー	説明と使用法
version		対応するブローカーコンテナーイメージを使用するように Operator が CR を自動更新する先の AMQ Broker の**マイナーバージョン**を指定します。たとえば、**version**の値を**7.6.0**から**7.7.0**に変更した場合 (および**upgrades.enabled**と**upgrades.minor**の両方が**true**に設定されている場合)、Operator は**deployment Plan.image**を**registry.redhat.io/amq7/amq-broker-rhel8:7.8-x** の形式のブローカーイメージに更新します **型**: String **例**: 7.7.0 **デフォルト値**: AMQ Broker の現在のバージョン

8.1.2. アドレスのカスタムリソースの設定リファレンス

アドレス CRD に基づく CR インスタンスを使用して、デプロイメント内のブローカーのアドレスとキューを定義できます。次の表で、設定できる項目の詳細を示します。



重要

アスタリスク(*)でマークされた設定アイテムは、該当するカスタムリソース(CR)でデプロイに必要です。不要なアイテムの値を明示的に指定しない場合には、設定にデフォルト値が使用されます。

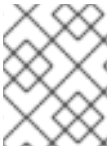
エントリー	説明と使用法
addressName*	ブローカーで作成されるアドレス名。 **型**: String **例**: address0 **デフォルト値**: 指定なし

エントリー	説明と使用法
queueName	ブローカーで作成されるキュー名。**queueName** が指定されていない場合、CR はアドレスのみを作成します。 **型**: String **例**: queue0 **デフォルト値**: 指定なし
removeFromBrokerOnDelete*	デプロイメントのアドレス CR インスタンスを削除するときに、Operator がデプロイメント内のすべてのブローカーの既存のアドレスを削除するかどうかを指定します。デフォルト値は**false**です。これは、CR を削除するときに Operator が既存のアドレスを削除しないことを意味します。 **型**: Boolean **例**: True **デフォルト値**: false
routingType*	使用するルーティングタイプ。**anycast** または **multicast**。 **型**: String **例**: anycast **デフォルト値**: multicast

8.1.3. セキュリティーのカスタムリソースの設定リファレンス

セキュリティー CRD に基づく CR インスタンスを使用して、デプロイメント内のブローカーのセキュリティー構成を定義できます。これには、次のものが含まれます。

- ユーザーとロール
- **Properties Login Module**、**guest Login Module**、**keycloak Login Module** などのログインモジュール
- ロールベースのアクセス制御
- コンソールアクセス制御



注記

オプションの多くは、ブローカーのセキュリティー保護 で説明されているブローカーのセキュリティーの概念を理解する必要があります

次の表で、設定できる項目の詳細を示します。

重要

アスタリスク(*)でマークされた設定アイテムは、該当するカスタムリソース(CR)でデプロイに必要です。不要なアイテムの値を明示的に指定しない場合には、設定にデフォルト値が使用されます。

エントリー	サブエントリー	説明と使用法
loginModules		1 つ以上のログインモジュール設定。 ログインモジュールは、次のいずれかのタイプになります。 ● **Properties Login Module**: ブローカーユーザーを直接定義できます。 ● **guestLoginModule** - ログイン認証情報がないユーザーや、認証情報が認証に失敗するユーザーの場合は、ゲストアカウントを使用してブローカーへの制限されたアクセスを付与できます。 ● **keycloak Login Module** - Red Hat シングルサインオンを使用してブローカーを保護できます。
propertiesLoginModule	name*	ログインモジュールの名前。 **型**: String **例**: my-login **デフォルト値**: 該当なし
	users.name*	ユーザーの名前。 **型**: String **例**: jdoe **デフォルト値**: 該当なし
	users.password*	ユーザーのパスワード。 **型**: String **例**: パスワード **デフォルト値**: 該当なし

エントリー	サブエントリー	説明と使用法
	users.roles	ロールの名前。 **型**: String **例**: viewer **デフォルト値**: 該当なし
guestLoginModule	name*	ゲストログインモジュールの名前。 **型**: String **例**: guest-login **デフォルト値**: 該当なし
	guestUser	ゲストユーザーの名前。 **型**: String **例**: myguest **デフォルト値**: 該当なし
	guestRole	ゲストユーザーのロールの名前。 **型**: String **例**: guest **デフォルト値**: 該当なし
keycloakLoginModule	name	KeycloakLoginModule の名前 **型**: String **例**: sso **デフォルト値**: 該当なし
	moduleType	KeycloakLoginModule のタイプ (directAccess または bearerToken) **型**: String **例**: bearerToken **デフォルト値**: 該当なし

エントリー	サブエントリー	説明と使用法
	設定	以下の設定項目は Red Hat シングルサインオンに関連しており、詳細情報はOpenIDConnectのドキュメントから入手できます。
	configuration.realm*	KeycloakLoginModule のレルム **型**: String **例**: myrealm **デフォルト値**: 該当なし
	configuration.realmPublicKey	レルムの公開鍵 **型**: String **デフォルト値**: 該当なし
	configuration.authServerUrl*	keycloak 認証サーバーの URL **型**: String **デフォルト値**: 該当なし
	configuration.sslRequired	SSL が必要かどうかを指定します **型**: String 有効な値は、「all」、「external」、および「none」です。
	configuration.resource*	リソース名 アプリケーションの client-id各アプリケーションには、アプリケーションを識別するために使用される client-id があります。
	configuration.publicClient	パブリッククライアントかどうかを指定します。 **型**: Boolean **デフォルト値**: false **例**: false

エントリー	サブエントリー	説明と使用法
	configuration.credentials.key	認証情報キーを指定します。 **型**: String **デフォルト値**: 該当なし **型**: String **デフォルト値**: 該当なし
	configuration.credentials.value	認証情報の値を指定します **型**: String **デフォルト値**: 該当なし
	configuration.useResourceRoleMappings	リソースロールマッピングを使用するかどうかを指定します **型**: Boolean **例**: false
	configuration.enableCors	CORS(Cross-Origin Resource Sharing)を有効にするかどうかを指定します。 CORS プリフライトリクエストを処理します。また、アクセストークンを調べて、有効な発信元を判別します。 **型**: Boolean **デフォルト値**: false
	configuration.corsMaxAge	CORS 最大有効期限 CORS が有効な場合は、Access-Control-Max-Age ヘッダーの値が設定されます。
	configuration.corsAllowedMethods	CORS で利用可能なメソッド CORS が有効な場合は、Access-Control-Allow-Methods ヘッダーの値が設定されます。これはコンマ区切りの文字列である必要があります。
	configuration.corsAllowedHeaders	CORS で利用可能なヘッダー CORS が有効な場合は、Access-Control-Allow-Headers ヘッダーの値を設定します。これはコンマ区切りの文字列である必要があります。

エントリー	サブエントリー	説明と使用法
	configuration.corsExposedHeaders	CORS 公開ヘッダー CORS が有効な場合は、Access-Control-Expose-Headers ヘッダーの値を設定します。これはコンマ区切りの文字列である必要があります。
	configuration.exposeToken	アクセストークンを公開するかどうかを指定します **型**: Boolean **デフォルト値**: false
	configuration.bearerOnly	ベアラートークンを検証するかどうかを指定します **型**: Boolean **デフォルト値**: false
	configuration.autoDetectBearerOnly	ベアラートークンのみを自動検出するかどうかを指定します **型**: Boolean **デフォルト値**: false
	configuration.connectionPoolSize	接続プールのサイズ **型**: Integer **デフォルト値**: 20
	configuration.allowAnyHostName	ホスト名を許可するかどうかを指定します **型**: Boolean **デフォルト値**: false
	configuration.disableTrustManager	トラストマネージャーを無効にするかどうかを指定します **型**: Boolean **デフォルト値**: false

エントリー	サブエントリー	説明と使用法
	configuration.trustStore*	トラストストアのパス これは、ssl-required が none、または disable-trust-manager が true でない限り、必須です。
	configuration.trustStorePassword*	トラストストアのパスワード これは、トラストストアが設定され、トラストストアにパスワードが必要な場合は必須です。
	configuration.clientKeyStore	クライアントキーストアのパス **型**: String **デフォルト値**: 該当なし
	configuration.clientKeyStorePassword	クライアントのキーストアパスワード **型**: String **デフォルト値**: 該当なし
	configuration.clientKeyPassword	クライアントキーのパスワード **型**: String **デフォルト値**: 該当なし
	configuration.alwaysRefreshToken	トークンを常に更新するかどうかを指定します **型**: Boolean **例**: false
	configuration.registerNodeAtStartup	起動時にノードを登録するかどうかを指定します **型**: Boolean **例**: false
	configuration.registerNodePeriod	ノードの再登録期間 **型**: String **デフォルト値**: 該当なし

エントリー	サブエントリー	説明と使用法
	configuration.tokenStore	トークンストアのタイプ (session または Cookie) **型**: String **デフォルト値**: 該当なし
	configuration.tokenCookiePath	クッキーストアのクッキーパス **型**: String **デフォルト値**: 該当なし
	configuration.principalAttribute	UserPrincipal 名を入力する OpenID Connect ID Token 属性。 UserPrincipal 名を入力する OpenID Connect ID Token 属性。トークン属性が null の場合、デフォルトは sub に設定されます。使用できる値は sub、preferred_username、email、name、nickname、given_name、family_name です。
	configuration.proxyUrl	プロキシー URL
	configuration.turnOffChangeSessionIdOnLogin	ログインに成功したときにセッション ID を変更するかどうかを指定します **型**: Boolean **例**: false
	configuration.tokenMinimumTimeToLive	アクティブなアクセストークンを更新するまでの最小時間 **型**: Integer **デフォルト値**: 0
	configuration.minTimeBetweenJwksRequests	新しい公開鍵取得の Keycloak への要求 2 件の間で最小限あける間隔 **型**: Integer **デフォルト値**: 10

エントリー	サブエントリー	説明と使用法
	configuration.publicKeyCacheTtl	新しい公開鍵取得の Keycloak への要求 2 件の間で最大あけることのできる間隔 **型**: Integer **デフォルト値**: 86400
	configuration.ignoreOauthQueryParameter	ベアラートークン処理の access_token クエリパラメーターの処理をオフにするかどうか **型**: Boolean **例**: false
	configuration.verifyTokenAudience	トークンにオーディエンスとしてこのクライアント名(リソース)が含まれているかどうかを検証します **型**: Boolean **例**: false
	configuration.enableBasicAuth	Basic 認証をサポートするかどうか **型**: Boolean **デフォルト値**: false
	configuration.confidentialPort	SSL/TLS を介した安全な接続で Keycloak サーバーが使用する機密ポート **型**: Integer **例**: 8443
	configuration.redirectRewriteRules.key	リダイレクト URI の照合に使用される正規表現。 **型**: String **デフォルト値**: 該当なし
	configuration.redirectRewriteRules.value	置換文字列 **型**: String **デフォルト値**: 該当なし

エントリー	サブエントリー	説明と使用法
	configuration.scope	Direct Access Grants Login Module の OAuth2 スコープパラメーター **型**: String **デフォルト値**: 該当なし
securityDomains		ブローカーのセキュリティードメイン
	brokerDomain.name	ブローカードメイン名 **型**: String **例**: activemq **デフォルト値**: 該当なし
	brokerDomain.loginModules	1 つ以上のログインモジュール。各エントリーは、上記の **login Modules** セクションで事前に定義されている必要があります。
	brokerDomain.loginModules.name	ログインモジュールの名前 **型**: String **例**: prop-module **デフォルト値**: 該当なし
	brokerDomain.loginModules.flag	PropertiesLoginModuleと同じように、**required**、**requisite**、**sufficient** および **optional** が有効な値です。 **型**: String **例**: sufficient **デフォルト値**: 該当なし
	brokerDomain.loginModules.debug	Debug
	brokerDomain.loginModules.reload	Reload

エントリー	サブエントリー	説明と使用法
	consoleDomain.name	ブローカードメイン名 **型**: String **例**: activemq **デフォルト値**: 該当なし
	consoleDomain.loginModules	シングルログインモジュール構成。
	consoleDomain.loginModules.name	ログインモジュールの名前 **型**: String **例**: prop-module **デフォルト値**: 該当なし
	consoleDomain.loginModules.flag	PropertiesLoginModuleと同じように、**required**、**requisite**、**sufficient** および **optional** が有効な値です。 **型**: String **例**: sufficient **デフォルト値**: 該当なし
	consoleDomain.loginModules.debug	Debug **型**: Boolean **例**: false
	consoleDomain.loginModules.reload	Reload **型**: Boolean **例**: True **デフォルト**: false
securitySettings		**broker.xml** または **management.xml** に追加する別のセキュリティー設定
	broker.match	セキュリティー設定セクションのアドレス一致パターン。一致パターンの構文の詳細は、 AMQ Broker のワイルドカード構文を参照してください。

エントリー	サブエントリー	説明と使用法
	broker.permissions.operationType	権限の設定 で説明されている、セキュリティー設定の操作タイプ。 **型**: String **例**: createAddress **デフォルト値**: 該当なし
	broker.permissions.roles	権限の設定 で説明されているように、セキュリティー設定はこれらのロールに適用されます。 **型**: String **例**: root **デフォルト値**: 該当なし
securitySettings.management		**management.xml**を構成するためのオプション。
	hawtioRoles	ブローカーコンソールへのログインを許可されたロール。 **型**: String **例**: root **デフォルト値**: 該当なし
	connector.host	管理 API に接続するためのコネクターホスト。 **型**: String **例**: myhost **デフォルト値**: localhost
	connector.port	管理 API に接続するためのコネクターポート。 **型**: integer **例**: 1099 **デフォルト値**: 1099

エントリー	サブエントリー	説明と使用法
	connector.jmxRealm	管理 API の JMX レルム。 **型**: String **例**: activemq **デフォルト値**: activemq
	connector.objectName	管理 API の JMX オブジェクト名。 **型**: String **例**: connector:name = rmi **デフォルト**: connector:name = rmi
	connector.authenticatorType	管理 API 認証タイプ。 **型**: String **例**: パスワード **デフォルト**: password
	connector.secured	管理 API 接続が保護されているかどうか。 **型**: Boolean **例**: True **デフォルト値**: false
	connector.keyStoreProvider	管理コネクターのキーストアプロバイダー。Connector.secured = "true" を設定した場合に必要です。デフォルト値は JKS です。
	connector.keyStorePath	キーストアの場所。Connector.secured = "true" を設定した場合に必要です。
	connector.keyStorePassword	管理コネクターのキーストアパスワード。Connector.secured = "true" を設定した場合に必要です。

エントリー	サブエントリー	説明と使用法
	connector.trustStoreProvider	管理コネクターのトラストストアプロバイダー connector.secured = "true"を設定した場合に必要です。 **型**: String **例**: JKS **デフォルト**: JKS
	connector.trustStorePath	管理コネクターのトラストストアの場所。Connector.secured = "true" を設定した場合に必要です。 **型**: String **デフォルト値**: 該当なし
	connector.trustStorePassword	管理コネクターのトラストストアパスワード。Connector.secured = "true" を設定した場合に必要です。 **型**: String **デフォルト値**: 該当なし
	connector.passwordCodec	管理コネクターのパスワードコーデック。構成ファイルでのパスワードの暗号化 で説明されているように使用するパスワードコーデックの完全修飾クラス名。
	authorisation.allowedList.domain	allowedList のドメイン **型**: String **デフォルト値**: 該当なし
	authorisation.allowedList.key	allowedList のキー **型**: String **デフォルト値**: 該当なし
	authorisation.defaultAccess.method	defaultAccessList のメソッド **型**: String **デフォルト値**: 該当なし

エントリー	サブエントリー	説明と使用法
	authorisation.defaultAccess.roles	defaultAccess リストのロール **型**: String **デフォルト値**: 該当なし
	authorisation.roleAccess.domain	roleAccess リストのドメイン **型**: String **デフォルト値**: 該当なし
	authorisation.roleAccess.key	roleAccess リストのキー **型**: String **デフォルト値**: 該当なし
	authorisation.roleAccess.accessList.method	roleAccess リストの方法 **型**: String **デフォルト値**: 該当なし
	authorisation.roleAccess.accessList.roles	roleAccess リストのロール **型**: String **デフォルト値**: 該当なし
	applyToCrNames	このセキュリティー設定を、現在の名前空間の指定された CR で定義したブローカーに適用します。* または空の文字列の値は、すべてのブローカーに適用することを意味します。 **型**: String **例**: my-broker **デフォルト値**: 現在の名前空間の CR で定義したすべてのブローカー。

8.2. アプリケーションテンプレートパラメーター

OpenShift Container Platform イメージでの AMQ Broker の設定は、アプリケーションテンプレートパラメーターの値を指定して実行されます。次のパラメーターを設定できます。

表8.1 アプリケーションテンプレートパラメーター

パラメーター	説明
AMQ_ADDRESSES	起動時にブローカーでデフォルトで使用可能なアドレスを、コンマ区切りのリストで指定します。
AMQ_ANYCAST_PREFIX	多重化プロトコルポート 61616 および 61617 に適用される anycast プレフィックスを指定します。
AMQ_CLUSTERED	クラスタリングを有効にします。
AMQ_CLUSTER_PASSWORD	クラスタリングに使用するパスワードを指定します。AMQ Broker アプリケーションテンプレートは、AMQ_CREDENTIAL_SECRET で指定されたシークレットに格納されているこのパラメーターの値を使用します。
AMQ_CLUSTER_USER	クラスタリングに使用するクラスターユーザーを指定します。AMQ Broker アプリケーションテンプレートは、AMQ_CREDENTIAL_SECRET で指定されたシークレットに格納されているこのパラメーターの値を使用します。
AMQ_CREDENTIAL_SECRET	ブローカーのユーザー名/パスワード、クラスターのユーザー名/パスワード、トラストストアとキーストアのパスワードなどの機密性の高い資格情報が保存されるシークレットを指定します。
AMQ_DATA_DIR	データのディレクトリーを指定します。Stateful Sets で使用されます。
AMQ_DATA_DIR_LOGGING	データディレクトリーロギング用のディレクトリーを指定します。
AMQ_EXTRA_ARGS	**artemiscreate** に渡す追加の引数を指定します。
GLOBAL_MAX_SIZE	メッセージデータが使用可能な最大メモリー量を指定します。値が指定されていない場合は、システムのメモリーの半分が割り当てられます。
AMQ_KEYSTORE	SSL キーストアファイル名を指定します。値が指定されていない場合に、パスワードが無作為に生成されますが、SSL は構成されません。
AMQ_KEYSTORE_PASSWORD	(オプション) SSL キーストアの復号化に使用するパスワードを指定します。AMQ Broker アプリケーションテンプレートは、AMQ_CREDENTIAL_SECRET で指定されたシークレットに格納されているこのパラメーターの値を使用します。

パラメーター	説明
AMQ_KEYSTORE_TRUSTSTORE_DIR	シークレットがマウントされるディレクトリーを指定します。デフォルト値は **/etc/amq-secret-volume** です。
AMQ_MAX_CONNECTIONS	SSL の場合のみ、アクセプターが受け入れる接続の最大数を指定します。
AMQ_MULTICAST_PREFIX	多重化プロトコルポート 61616 および 61617 に適用される multicast プレフィックスを指定します。
AMQ_NAME	ブローカーインスタンスの名前を指定します。デフォルト値は**amq-broker**です。
AMQ_PASSWORD	ブローカーへの認証に使用するパスワードを指定します。AMQ Broker アプリケーションテンプレートは、AMQ_CREDENTIAL_SECRET で指定されたシークレットに格納されているこのパラメーターの値を使用します。
AMQ_PROTOCOL	ブローカーが使用するメッセージングプロトコルをコンマ区切りのリストで指定します。使用可能なオプションは、 **amqp**、**mqtt**、**openwire**、**stomp**、および**hornetq**です。何も指定されていない場合は、全プロトコルを使用できます。イメージを Red Hat JBoss Enterprise Application Platform と統合するには、OpenWire プロトコルを指定する必要がありますが、他のプロトコルもオプションで指定できます。
AMQ_QUEUES	起動時にブローカーでデフォルトで使用可能なキューを、コンマ区切りのリストで指定します。
AMQ_REQUIRE_LOGIN	**true**に設定すると、ログインが必要になります。指定しない場合、または**false**に設定した場合、匿名アクセスを使用できます。デフォルトでは、このパラメーターの値は指定されていません。
AMQ_ROLE	作成されたロールの名前を指定します。デフォルト値は **amq** です。
AMQ_TRUSTSTORE	SSL トラストストアのファイル名を指定します。値が指定されていない場合に、パスワードが無作為に生成されますが、SSL は構成されません。

パラメーター	説明
AMQ_TRUSTSTORE_PASSWORD	(オプション) SSL トラストストアの復号化に使用されるパスワードを指定します。AMQ Broker アプリケーションテンプレートは、AMQ_CREDENTIAL_SECRET で指定されたシークレットに格納されているこのパラメーターの値を使用します。
AMQ_USER	ブローカーへの認証に使用されるユーザー名を指定します。AMQ Broker アプリケーションテンプレートは、AMQ_CREDENTIAL_SECRET で指定されたシークレットに格納されているこのパラメーターの値を使用します。
APPLICATION_NAME	OpenShift 内で使用されるアプリケーションの名前を指定します。これは、アプリケーション内のサービス、Pod、およびその他のオブジェクトの名前で使用されます。
IMAGE	イメージを指定します。**永続性**、 **persistent-ssl**、および**statefulset-clustered**テンプレートで使用されます。
IMAGE_STREAM_NAMESPACE	イメージストリームの名前空間を指定します。**ssl** および **basic**テンプレートで使用されます。
OPENSHIFT_DNS_PING_SERVICE_PORT	OpenShift DNSping サービスのポート番号を指定します。
PING_SVC_NAME	OpenShift DNSping サービスの名前を指定します。**APPLICATION_NAME**の値を指定した場合、デフォルト値は**$APPLICATION_NAME-ping**です。そうでない場合には、デフォルト値は **ping** です。**PING_SVC_NAME**にカスタム値を指定すると、この値がデフォルト値を上書きします。テンプレートを使用して同じ OpenShift プロジェクト namespace に複数のブローカークラスターをデプロイする場合は、 **PING_SVC_NAME**がデプロイメントごとに一意の値が指定されていることを確認する必要があります。
VOLUME_CAPACITY	データベースボリュームの永続ストレージのサイズを指定します。

注記

カスタム設定に**broker.xml**を使用する場合に、そのファイルで次のパラメーターに指定された値は、アプリケーションテンプレートで同じパラメーターに指定された値をオーバーライドします。

- AMQ_NAME
- AMQ_ROLE
- AMQ_CLUSTER_USER
- AMQ_CLUSTER_PASSWORD

8.3. ロギング

OpenShift ログの表示に加えて、コンテナーのコンソールに出力される AMQ ログを表示することにより、OpenShift Container Platform イメージで実行中の AMQ Broker のトラブルシューティングを行うことができます。

手順

- コマンドラインで、次のコマンドを実行します。

```
$ oc logs -f <pass:quotes[<pod-name>]> <pass:quotes[<container-name>]>
```

改訂日時： 2022-04-09 22:49:00 +1000