Red Hat AMQ Broker 7.11

OpenShift での AMQ Broker のデプロイ

AMQ Broker 7.11 で使用する場合

Last Updated: 2024-05-02

Red Hat AMQ Broker 7.11 OpenShift での AMQ Broker のデプロイ

AMQ Broker 7.11 で使用する場合

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

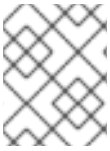
Red Hat AMQ Broker 7.11 は、OpenShift Container Platform (OCP) 4.12、4.13、4.14、または 4.15 で使用するためのコンテナー化されたイメージとして利用できます。

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. サポートされない機能

- 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。代わりに、AMQ Broker は OpenShift Container Platform で提供される HA 機能を使用します。

- 外部クライアントは AMQ Broker によって提供されるトポロジー情報を使用できません。
  AMQ Core Protocol JMS クライアントまたは AMQ JMS クライアントが OpenShift Container Platform クラスター内のブローカーに接続すると、ブローカーはクラスター内の他の各ブローカーの IP アドレスとポート情報をクライアントに送信でき、現在のブローカーへの接続が失われた場合、クライアントのフェイルオーバーリストとして機能します。

  各ブローカーに提供される IP アドレスは内部 IP アドレスであり、OpenShift Container Platform クラスターの外部にあるクライアントはアクセスできません。外部クライアントが内部 IP アドレスを使用してブローカーに接続しようとするのを防ぐには、クライアントが最初にブローカーに接続するために使用する URI に次の設定を設定します。

クライアント	設定
AMQ Core Protocol JMS クライアント	**useTopologyForLoadBalancing=false**
AMQ JMS クライアント	**failover.amqpOpenServerListAction=IGNORE**

1.3. このドキュメントの表記慣例

このドキュメントでは、**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
このドキュメントでは、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 アクセスの管理 を参照してください。

このドキュメントにおけるファイルパスの使用
このドキュメントでは、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。クラスター化されたデプロイメント内のブローカーが、デプロイメントの意図的なスケールダウンによりシャットダウンした場合、この機能により、同じブローカークラスター内でまだ実行されているブローカー Pod にメッセージが移行されます。

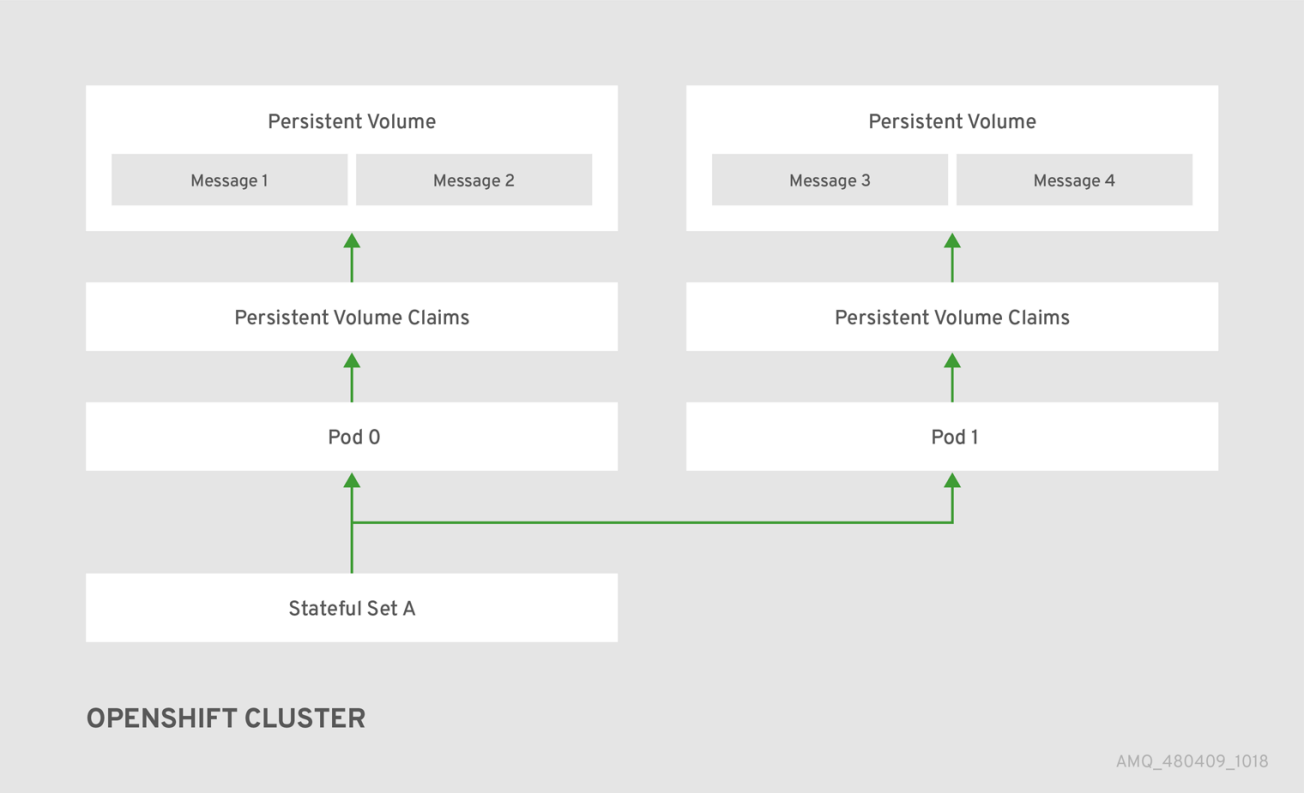
2.1. 高可用性 (HA) の概要

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。OpenShift Container Platform での AMQ Broker の場合、これは、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することを意味します。

AMQ Broker は、OpenShift Container Platform で提供される HA 機能を使用して、Pod の失敗を軽減します。

- AMQ Broker で永続ストレージが有効になっている場合、各ブローカー Pod は、永続ボリューム要求 (PVC) を使用して要求した永続ボリューム (PV) にデータを書き込みます。Pod が削除された後でも PV は引き続き利用可能です。ブローカー Pod が失敗した場合、OpenShift Container Platform は同じ名前で Pod を再起動し、メッセージングデータを含む既存の PV を使用します。
- クラスター内で複数のブローカー Pod を実行し、ノードの障害から保護するために Pod を別個のノードに分散することができます。クラスターでは、各ブローカー Pod はメッセージデータを独自の PV に書き込みます。メッセージデータは、ブローカー Pod が別のノードで再起動された場合に、当該ブローカー Pod で使用できます。

次の図は、クラスター化された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。



関連情報

永続ストレージの使用方法については、「Operator デプロイメントノート」 を参照してください。

ブローカー Pod を別個のノードに分散する方法については、「容認を使用した Pod の配置の制御」 を参照してください。

2.2. 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イスを使用して、OpenShift コマンドラインインターフェイス (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イスについては、「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」

2.3. 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デプロイメントする 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。

ダウンロードおよび展開する Broker Operator アーカイブには、以下に示すように、**deploy/examples/address** および **deploy/examples/artemis** ディレクトリー内のデプロイメント例の CR も含まれています。

address_queue.yaml

異なる名前のアドレスとキューをデプロイします。CR がアンデプロイされるときにキューを削除します。

address_topic.yaml

マルチキャストルーティングタイプのアドレスをデプロイします。CR がアンデプロイされるときにアドレスを削除します。

artemis_address_settings.yaml

特定のアドレス設定を使用してブローカーをデプロイします。

artemis_cluster_persistence.yaml

永続ストレージを備えたクラスター化ブローカーをデプロイします。

artemis_enable_metrics_plugin.yaml

Prometheus メトリックプラグインがメトリックを収集できるようにします。

artemis_resources.yaml

ブローカーの CPU およびメモリーリソースの制限を設定します。

artemis_single.yaml

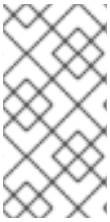
単一のブローカーをデプロイします。

2.4. 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

クラスター上の namespace に以前のバージョンの AMQ Broker Operator がすでにインストールされている場合、Red Hat では、潜在的な競合を避けるために、その namespace を監視するために AMQ Broker Operator 7.11 バージョンをインストールしないことを推奨します。

2.5. OPERATOR がイメージのデプロイに使用する設定を決定する方法

ActiveMQArtemis CR では、次のいずれかの設定を使用してコンテナーイメージをデプロイできます。

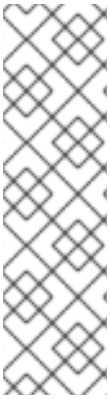
- **spec.version** 属性でバージョン番号を指定して、Operator がそのバージョン番号でデプロイするブローカーおよび init コンテナーイメージを選択できるようにします。
- Operator がデプロイする特定のブローカーと init コンテナーイメージのレジストリー URL を **spec.deploymentPlan.image** 属性と **spec.deploymentPlan.initImage** 属性に指定します。
- **spec.deploymentPlan.image** 属性の値を **placeholder** に設定します。これは、Operator のバージョンが認識している最新のブローカーおよび init コンテナーイメージを Operator が選択することを意味します。

注記

コンテナーイメージのデプロイにこれらの設定を使用しない場合、Operator は Operator のバージョンが認識している最新のブローカーおよび init コンテナーイメージを選択します。

CR を保存した後、Operator は次の検証を実行して、使用する設定を決定します。

- Operator は、CR に **spec.version** 属性が含まれているかどうかを確認します。
  - CR に **spec.version** 属性が含まれていない場合、Operator は CR に **spec.deploymentPlan.image** 属性と **spec.deployment.Plan.initImage** 属性が含まれているかどうかを確認します。
    - CR に **spec.deploymentPlan.image** 属性と **spec.deployment.Plan.initImage** 属性が含まれている場合、Operator はレジストリー URL で識別されるコンテナーイメージをデプロイします。
    - CR に **spec.deploymentPlan.image** 属性と **spec.deployment.Plan.initImage** 属性が含まれていない場合、Operator はデプロイするコンテナーイメージを選択します。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。
  - CR に **spec.version** 属性が含まれている場合、Operator は、指定されたバージョン番号が Operator がサポートする有効なバージョン範囲内にあることを確認します。
    - **spec.version** 属性の値が無効な場合、Operator はデプロイを停止します。
    - **spec.version** 属性の値が有効な場合、Operator は CR に **spec.deploymentPlan.image** 属性と **spec.deployment.Plan.initImage** 属性が含まれているかどうかを確認します。
      - CR に **spec.deploymentPlan.image** 属性と **spec.deployment.Plan.initImage** 属性が含まれている場合、Operator はレジストリー URL で識別されるコンテナーイメージをデプロイします。
      - CR に **spec.deploymentPlan.image** 属性と **spec.deployment.Plan.initImage** 属性が含まれていない場合、Operator はデプロイするコンテナーイメージを選択します。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。



注記

CR に **spec.deploymentPlan.image** 属性と **spec.deployment.Plan.initImage** 属性のいずれか 1 つだけが含まれている場合、Operator は **spec.version** 番号属性を使用して CR にない属性のイメージを選択します。**spec.version** 属性が CR にない場合には、その属性の最新の既知のイメージを選択します。

Red Hat は、異なるバージョンのブローカーおよび init コンテナーイメージがデプロイされることを防ぐために、**spec.deployment.Plan.initImage** 属性と **spec.deploymentPlan.image** 属性のいずれか 1 つのみを指定することは避けることを推奨します。

2.6. OPERATOR によるコンテナーイメージの選択方法

CR に、Operator がデプロイする必要のある特定のコンテナーイメージのレジストリー URL を指定する **spec.deploymentPlan.image** 属性および **spec.deployment.Plan.initImage** 属性が含まれていない場合、Operator は、デプロイ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イスを使用して 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**。CR に **spec.deploymentPlan.initImage** プロパティーを明示的に含めずに値を指定した場合、Operator は、選択した Operator コンテナーイメージのバージョンに一致する組み込み Init コンテナーイメージを選択します。

Operator は、ブローカーおよび init コンテナーイメージを選択するために、まず、必要なイメージの AMQ Broker バージョンを決定します。Operator は、**spec.version** プロパティーの値からバージョンを取得します。**spec.version** プロパティーが設定されていない場合、Operator は AMQ Broker の最新バージョンのイメージを使用します。

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。次のセクションで説明するように、これらの環境変数のセットは、AMQ Broker のさまざまなバージョン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。

2.6.1. ブローカーおよび init コンテナーイメージの環境変数

operator.yaml に含まれる環境変数には、次の命名規則があります。

コンテナープラットフォーム	命名規則
OpenShift Container Platform	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version>**
OpenShift Container Platform on IBM Z	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version>_s390x**
OpenShift Container Platform on IBM Power Systems	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_<AMQ_Broker_version>_ppc64le**

以下は、サポートされている各コンテナープラットフォームのブローカーおよび init コンテナーイメージの環境変数名の例です。

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_7116** **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_7116**
OpenShift Container Platform on IBM Z	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_7116_s390x** **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_s390x_7116**
OpenShift Container Platform on IBM Power Systems	**RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_7116_ppc64le** **RELATED_IMAGE_ActiveMQ_Artemis_Broker_Init_ppc64le_7116**

各環境変数の値は、Red Hat から入手可能なコンテナーイメージのアドレスを指定します。イメージ名は **Secure Hash Algorithm** (SHA) 値で表されます。以下に例を示します。

```
- name: RELATED_IMAGE_ActiveMQ_Artemis_Broker_Kubernetes_7116
  value: registry.redhat.io/amq7/amq-broker-rhel8@sha256:e8fa2a00e576ecb95561ffbdbf87b1c82d479c8791ab2c6ce741dd0d0b496d15
```

したがって、Operator は、AMQ Broker のバージョンとコンテナープラットフォームに基づいて、ブローカーと init コンテナーに適用できる環境変数名を決定します。Operator は、ブローカーコンテナーを開始するときに、対応するイメージ値を使用します。

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。

2.7. 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。通常のユーザーが 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、ロールと権限を 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推奨します。

- 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 つのブローカーで設定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

- CR を初めてデプロイする**前に**、リスト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が必要とする永続ボリューム要求 (PVC) のサイズとストレージクラス

  - 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

- Operator が StatefulSet で動的に更新できない CR のパラメーターを更新すると、Operator は StatefulSet を削除し、更新されたパラメーター値でそれを再作成します。StatefulSet を削除すると、すべての Pod が削除されて再作成されるため、ブローカーが一時的に停止します。Operator が StatefulSet で動的に更新できない CR 更新の例は、**persistenceEnabled=false** を **persistenceEnabled=true** に変更した場合です。

2.8. 既存の OPERATOR によって監視されている名前空間の特定

クラスターにインストール済みの Operators for AMQ Broker がすでに含まれており、新しい Operator がすべてまたは複数の名前空間を監視するようにする場合は、新しい Operator が既存の Operator と同じ名前空間を監視しないようにする必要があります。以下の手順を使用して、既存の Operator によって監視されている名前空間を特定します。

手順

1. OpenShift Container Platform Web コンソールの左ペインで、**Workloads → Deployments** をクリックします。
2. **プロジェクト** ドロップダウンリストで、**All Projects** を選択します。
3. **Filter Name** ボックスに文字列 (**amq** など) を指定して、クラスターにインストールされている Operators for AMQ Broker を表示します。

注記

名前空間 列には、各 Operator が **デプロイ** されている名前空間が表示されます。

4. インストールされた各 Operator for AMQ Broker が **監視** するように設定されている名前空間を確認します。
   a. Operator 名をクリックして Operator の詳細を表示し、**YAML** タブをクリックします。
   b. **WATCH_NAMESPACE** を検索し、Operator が監視する名前空間をメモします。
      - **WATCH_NAMESPACE** セクションに、値が **metadata.namespace** の **fieldPath** フィールドがある場合、Operator はデプロイされている名前空間を監視しています。
      - **WATCH_NAMESPACE** セクションに名前空間のリストを持つ **value** フィールドがある場合、Operator は指定された名前空間を監視しています。以下に例を示します。

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

      - **WATCH_NAMESPACE** セクションに空の **value** フィールドまたはアスタリスクがある場合、Operator はクラスター上のすべての名前空間を監視しています。以下に例を示します。

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

      この場合、新しい Operator をデプロイする前に、既存の Operator をアンインストールするか、特定の名前空間を監視するように再設定する必要があります。

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。手順を完了すると、Operator は個別の Pod で実行され、作成した各ブローカーインスタンスは Operator と同じプロジェクト内の StatefulSet 内の個別の 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、以下で説明するように、最新の **AMQ Broker 7.11.6 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。

このセクションの手順では、OpenShift コマンドラインインターフェイス (CLI) を使用して、最新バージョンの Operator for AMQ Broker 7.11 を特定の OpenShift プロジェクトにインストールおよびデプロイする方法を示します。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イスを使用する 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 のデプロイの準備

CLI を使用して Operator をデプロイする前に、Operator インストールファイルをダウンロードしてデプロイメントを準備する必要があります。

手順

1. Web ブラウザーで、AMQ Broker 7.11.6 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 **7.11.6** 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. 最新の **AMQ Broker 7.11.6 Operator Installation and Example Files**の横にある **Download** をクリックします。
   amq-broker-operator-7.11.6-ocp-install-examples.zip 圧縮アーカイブのダウンロードが自動的に開始されます。
4. アーカイブを選択した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-7.11.6-ocp-install-examples.zip ~/broker/operator
```

5. 選択したディレクトリーで、アーカイブの内容を抽出します。以下に例を示します。

```
$ cd ~/broker/operator
$ unzip amq-broker-operator-7.11.6-ocp-install-examples.zip
```

6. アーカイブのデプロイメント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-7.11.6-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、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. デプロイメントした 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. デフォルトのサービスアカウント名を変更する場合は、**metadata** セクションで **amq-broker-controller-manager** をカスタム名に置き換えます。

   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. デフォルトのロール名を変更する場合は、**metadata** セクションで **amq-broker-operator-role** をカスタム名に置き換えます。

    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。

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

c. 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
```

12. Operator のリーダー選出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リーダー選出ロールにバインドします。

a. Operator のリーダー選出ロールを作成します。

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

b. プロジェクトでリーダー選出ロールバインディングを作成します。

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

13. (オプション) Operator が複数の名前空間を監視するようにする場合は、以下の手順を実行します。

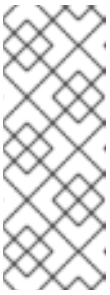
注記

OpenShift Container Platform クラスターに、インストール済みの Operators for AMQ Broker がすでに含まれている場合は、新しい Operator が既存の Operator と同じ名前空間を監視しないようにする必要があります。既存の Operator によって監視されている名前空間を識別する方法については、Identifying namespaces watched by existing Operators を参照してください。

a. 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。

b. Operator がクラスター内のすべての名前空間を監視するようにする場合は、**WATCH_NAMESPACE** セクションで **value** 属性を追加し、値をアスタリスクに設定します。**WATCH_NAMESPACE** セクションの既存の属性をコメントアウトします。以下に例を示します。

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



注記

競合を避けるために、複数の Operator が同じ名前空間を監視しないようにしてください。たとえば、Operator をデプロイしてクラスター上の **すべての** 名前空間を監視する場合は、別の Operator をデプロイして個々の名前空間を監視することはできません。Operator がすでにクラスターにデプロイされている場合、次のステップで説明するように、新しい Operator が監視する名前空間のリストを指定できます。

c. Operator がクラスター上のすべての名前空間ではなく、複数の名前空間を監視するようにする場合は、**WATCH_NAMESPACE** セクションで、名前空間のリストを指定します。既存の Operator によって監視されている名前空間を除外していることを確認してください。以下に例を示します。

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

d. ダウンロードして展開した Operator アーカイブの deploy ディレクトリーで、**cluster_role_binding.yaml** ファイルを開きます。

e. Subjects セクションで、Operator を **デプロイする** OpenShift Container Platform プロジェクトに対応する名前空間を指定します。以下に例を示します。

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

注記

古いバージョンの Operator を使用してブローカーを以前にデプロイし、Operator をデプロイして複数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。

f. プロジェクトにクラスターロールを作成します。

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

g. プロジェクトにクラスターロールバインディングを作成します。

```
$ oc create -f deploy/cluster_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

このセクションの手順では、OpenShift コマンドラインインターフェイス (CLI) を使用して、最新バージョンの Operator for AMQ Broker 7.11 を OpenShift プロジェクトにデプロイする方法を示します。

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る。「Operator のデプロイの準備」 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する。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る。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る。

- 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る。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 つのブローカーで設定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る。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**ストレージを使用する。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、以下を参照すること。

  - 永続ストレージについて

手順

1. OpenShift コマンドラインインターフェイス (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デプロイメント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-7.11.6-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
```

   d. セキュリティー CRD をデプロイします。

```
$ oc create -f deploy/crds/broker_activemqartemissecurity_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。以下に示すように、**spec.containers.image** プロパティーの値が Operator のバージョン 7.11.6-opr-2 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:7.10
            image: registry.redhat.io/amq7/amq-broker-rhel8-
operator@sha256:6a87f1a46da682870c4495bece5d8ef5b2fb62b4ecda9e6826664031757bf984
```

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. 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イスを使用する 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イスです。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。

注記

OperatorHub では、各チャネルで提供される最新の Operator バージョンのみをインストールできます。以前のバージョンの Operator をインストールする場合は、CLI を使用して Operator をインストールする必要があります。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。

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。

- クラスター管理者の権限がある。

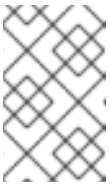
手順

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 7.11 の場合、この Operator の最新マイナーバージョンタグは **7.11.6-opr-2** 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、バージョン 7.11 のみの更新を受信する **7.11.x** チャネルを選択します。**7.11.x** チャネルは長期サポート (LTS) チャネルです。
   OpenShift Container Platform クラスターがインストールされた時期によっては、古いバージョンの AMQ Broker のチャネルが表示される場合もあります。他にサポートされているチャネルは **7.10.x** のみで、これも LTS チャネルです。
   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推奨します。

- AMQ Broker 7.11 で次の項目を設定する場合は、CR を初めてデプロイする **前** に、メインブローカー 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が必要とする永続ボリューム要求 (PVC) のサイズとストレージクラス
  - 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イス (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イスを使用して 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する。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る。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る。

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イスの使用:

      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。

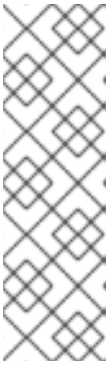
      iv. **Instances** タブをクリックします。

      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。

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

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イスの使用:

      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イスの使用:

      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/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  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。

- 「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 永続ボリュームの解放 を参照してください。

- AMQ Broker 7.11 で次の項目を設定する場合は、CR を初めてデプロイする **前** に、メイン 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が必要とする永続ボリューム要求 (PVC) のサイズとストレージクラス。

  - 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。

3.5. OPERATOR のログレベルの変更

AMQ Broker Operator のデフォルトのログレベルは **info** で、情報とエラーメッセージがログに記録されます。デフォルトのログレベルを変更して、Operator ログに書き込まれる詳細を増減できます。

OpenShift Container Platform コマンドラインインターフェイスを使用して Operator をインストールする場合は、Operator のインストール前または後に、Operator 設定ファイル **operator.yaml** に新しいログレベルを設定できます。Operator Hub を使用する場合は、Operator のインストール後に OpenShift Container Platform Web コンソールを使用して Operator サブスクリプションのログレベルを設定できます。

Operator で使用できるその他のログレベルは次のとおりです。

error

エラーメッセージのみをログに書き込みます。

debug

デバッグメッセージを含むすべてのメッセージをログに書き込みます。

手順

1. OpenShift Container Platform コマンドラインインターフェイスの使用:
   a. クラスター管理者としてログインしている。以下に例を示します。

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

   b. Operator がインストールされていない場合は、次の手順を実行してログレベルを変更します。
      i. 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。
      ii. **zap-log-level** 属性の値を **debug** または **error** に変更します。以下に例を示します。

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

      iii. **operator.yaml** ファイルを保存します。
      iv. Operator をインストールします。
   c. Operator がすでにインストールされている場合は、**sed** コマンドを使用して、**deploy/operator.yaml** ファイルのログレベルを変更し、Operator を再デプロイします。たとえば、次のコマンドはログレベルを **info** から **error** に変更し、Operator を再デプロイします。

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

2. OpenShift Container Platform Web コンソールの使用
   a. OpenShift Container Platform にクラスター管理者としてログインします。
   b. 左側のペインで、**Operators → Installed Operators** をクリックします。
   c. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
   d. **Subscriptions** タブをクリックします。
   e. **Actions** をクリックします。
   f. **Edit Subscription** をクリックします。
   g. **YAML** タブをクリックします。
   コンソール内で YAML エディターが開き、サブスクリプションを編集できるようになります。

h. **config** 要素で、**ARGS** という環境変数を追加し、ログレベルとして **info**、**debug**、または **error** を指定します。次の例では、**debug** ログレベルを指定する **ARGS** 環境変数が Operator コンテナーに渡されます。

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

i. Save をクリックします。

3.6. ブローカーデプロイメントのステータス情報の表示

ブローカーのデプロイメントに関して OpenShift Container Platform によって報告される一連の標準条件のステータスを表示できます。ブローカーデプロイメントのカスタムリソース (CR) で提供される追加のステータス情報を表示することもできます。

手順

1. ブローカーデプロイメントの 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イスの使用:
      i. ブローカーデプロイメントのプロジェクトに CR を表示する権限を持つユーザーとして OpenShift Container Platform にログインします。
      ii. デプロイメントの CR を表示します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
      iv. **ActiveMQ Artemis** タブをクリックします。
      v. ActiveMQ Artemis インスタンスの名前をクリックします。
2. ブローカーデプロイメントの OpenShift Container Platform 条件のステータスを表示します。
   a. OpenShift コマンドラインインターフェイスの使用:
      i. CR の **status** セクションに移動し、**conditions** の詳細を表示します。
   b. OpenShift Container Platform Web コンソールの使用

i. **Details** タブで、**Conditions** セクションまで下にスクロールします。
条件にはステータスとタイプがあります。理由、メッセージ、その他の詳細が含まれている場合もあります。条件のステータス値は、条件が満たされる場合は **True**、条件が満たされない場合は **False**、条件のステータスを判断できない場合は **Unknown** になります。

注記

CR が CR 内の **spec.deploymentPlan.image**、**spec.deploymentPlan.initImage**、および **spec.version** 属性の推奨使用法に準拠していない場合、**Valid** 条件のステータスは **Unknown** になります。詳細は、「自動アップグレードに適用される制限の検証」 を参照してください。

ステータス情報は、次の条件に対して提供されます。

条件名	...のステータスを表示します
Valid	CR の検証。**Valid** 条件のステータスが **False** の場合、False ステータスの原因となった問題を最初に解決するまで、オペレーターは調整を完了せず、StatefulSet を更新しません。
Deployed	StatefulSet、Pod、およびその他のリソースの可用性。
Ready	他のより詳細な条件を集約する最上位の条件。**Ready** 条件のステータスが **True** になるのは、他の条件のステータスが **False** でない場合のみです。
BrokerPropertiesApplied	**brokerProperties** 属性を使用する CR で設定されたプロパティー。**BrokerPropertiesApplied** 条件の詳細は、「カスタムリソース定義で公開されていない項目の設定」 を参照してください。
JaasPropertiesApplied	CR で設定された Java Authentication and Authorization Service (JAAS) ログインモジュール。**JaasPropertiesApplied** 条件の詳細は、「シークレットでの JAAS ログインモジュールの設定」 を参照してください。

3. CR の **status** セクションで、ブローカーデプロイメントの追加のステータス情報を表示します。次の追加のステータス情報が表示されます。

deploymentPlanSize

デプロイメント内のブローカー Pod の数。

podstatus

デプロイメント内の各ブローカー Pod のステータスと名前。

version

ブローカーのバージョン、ブローカーのレジストリー URL、およびデプロイされる初期コンテナーイメージ。

upgrade

オペレータがメジャー、マイナー、パッチ、セキュリティー更新をデプロイメントに適用できるかどうか。これは、CR の **spec.deploymentPlan.image** 属性と **spec.version** 属性の値によって決まります。

- **spec.deploymentPlan.image** 属性でブローカーコンテナーイメージのレジストリー URL が指定されている場合、すべてのアップグレードタイプのステータスは **False** になります。これは、オペレータが既存のコンテナーイメージをアップグレードできないことを意味します。

- **spec.deploymentPlan.image** 属性が CR にない場合、または値が **placeholder** である場合、**spec.version** 属性の設定は次のように **upgrade** ステータスに影響します。

  - **spec.version** 属性が設定されているかどうか、またはその値に関係なく、**securityUpdates** のステータスは **True** です。

  - **spec.version** 属性の値にメジャーバージョンとマイナーバージョン (たとえば 7.10) のみがある場合、**patchUpdates** のステータスは **True** になるため、オペレーターはコンテナーイメージの最新のパッチバージョンにアップグレードできます。

  - **spec.version attribute** の値にメジャーバージョン （例：'7'）がある場合、**minorUpdates** のステータスは **True** になります。これにより、Operator はコンテナーイメージの最新のマイナーバージョンおよびパッチバージョンにアップグレードできます。

  - **spec.version** 属性が CR にない場合、 **majorUpdates** のステータスは **True** になるため、このバージョンが利用可能な場合は、7.xx から 8.xx へのアップグレードを含め、利用可能なアップグレードをデプロイできます。

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、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ている場合、init コンテナーはその設定を処理して XML に変換します。

3. CR の **applyRule** プロパティーの値に基づいて、init コンテナーは、上記のデフォルトのアドレス設定を CR で指定した設定と **マージ** するか、**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 コンテナーとブローカーコンテナーが共有するボリューム上にあります。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イス (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 **ActiveMQArtemisAddress** 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 **ActiveMQArtemis** 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

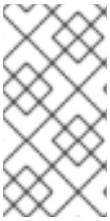
- OpenShift Container Platform ブローカーデプロイメントのアドレスおよびキューの作成と一致する設定の例については、以下を参照してください。
  - OpenShift Container Platform でのブローカーデプロイメントのアドレスおよびキューの作成
  - OpenShift Container Platform でのブローカーデプロイメントに設定されたアドレス設定の一致

- 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、および関連するアドレス設定の設定に関する包括的な情報については、**AMQ Broker の設定** の アドレスとキューの設定 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イスの使用:
      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/v1beta1
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イスの使用:

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。

4.2.3. 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削除

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から削除する方法を説明します。

手順

1. 削除するアドレスとキューの詳細 (**name**、**addressName**、**queueName** など) が記載されたアドレス CR ファイルがあることを確認してください。以下に例を示します。

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

2. アドレス CR の **spec** セクションに、**removeFromBrokerOnDelete** 属性を追加し、値を **true** に設定します。

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

removeFromBrokerOnDelete 属性を **true** に設定すると、アドレス CR を削除するときに、Operator はデプロイメント内のすべてのブローカーのアドレスと関連するメッセージを削除します。

3. 更新されたアドレス CR を適用して、削除するアドレスの **removeFromBrokerOnDelete** 属性を設定します。

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

4. アドレス CR を削除して、デプロイメント内のブローカーからアドレスを削除します。

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

4.2.4. 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

- ブローカーをデプロイするために **ActiveMQArtemis** CR インスタンスを作成している。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 CR インスタンスで指定された設定とマージまたは置換する **デフォルト** のアドレス設定について理解している。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. アドレス CR インスタンスの設定を開始して、デッドレターアドレスを追加し、デプロイメント内の各ブローカーの未配信メッセージを受信するキューを作成します。
   a. OpenShift コマンドラインインターフェイスの使用:
      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/v1beta1
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イスの使用:
      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イスの使用:
      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. CR を編集します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。

iv. **AMQ Broker** タブをクリックします。

v. ActiveMQArtemis インスタンス名をクリックします。

vi. **YAML** タブをクリックします。
コンソールで、YAML エディターが開き、CR インスタンスを編集できるようになります。

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。

5. CR の **spec** セクションに、次に示すように、単一の **addressSetting** セクションを含む新しい **addressSettings** セクションを追加します。

```
spec:
  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:
  deploymentPlan:
    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:
  deploymentPlan:
    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:
  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ハッシュ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:
  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

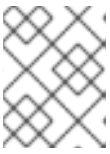
- CR で指定されるアドレス設定**と**、同じアドレスまたはアドレスのセットに一致するデフォルト設定の両方の場合:
  - デフォルト設定で指定されるプロパティー値を CR で指定されたプロパティー値に置き換えます。
  - CR **または**デフォルト設定で一意で指定されるプロパティー値を保持します。これらはそれぞれ最終マージされた設定の組み込みます。
- CR で指定されるアドレス設定**または**特定のアドレスセットに一意になるデフォルト設定の場合は、これらを最終でマージされた設定に含めます。

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. CR インスタンスを保存します。

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イス (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、および関連するアドレス設定の設定に関する包括的な情報については、**AMQ Broker の設定** の アドレスとキューの設定 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。

- OpenShift Container Platform の init コンテナーの詳細は、OpenShift Container Platform ドキュメントの Pod がデプロイされる前に init コンテナーを使用してタスクを実行する を参照してください。

4.3. 認証と承認の設定

デフォルトでは、AMQ Broker は Java Authentication and Authorization Service (JAAS) プロパティーログインモジュールを使用してユーザーを認証および承認します。デフォルトの JAAS ログインモジュールの設定は、各ブローカー Pod の **/home/jboss/amq-broker/etc/login.config** ファイルに保存され、同じディレクトリー内の **artemis-users.properties** および **artemis-roles.properties** ファイルからユーザーおよびロール情報を読み取ります。**ActiveMQArtemisSecurity** カスタムリソース (CR) を更新することで、ユーザーとロールの情報をデフォルトのログインモジュールのプロパティーファイルに追加します。

ActiveMQArtemisSecurity CR を更新してユーザーとロールの情報をデフォルトのプロパティーファイルに追加する代わりに、シークレットで 1 つ以上の JAAS ログインモジュールを設定することもできます。このシークレットは、各ブローカー Pod にファイルとしてマウントされます。JAAS ログインモジュールをシークレットで設定すると、**ActiveMQArtemisSecurity** CR を使用してユーザーとロールの情報を追加する場合に比べて、次の利点があります。

- シークレットでプロパティーログインモジュールを設定すると、プロパティーファイルを更新するたびにブローカーを再始動する必要がありません。たとえば、新しいユーザーをプロパティーファイルに追加してシークレットを更新すると、ブローカーを再起動しなくても変更が有効になります。

- **ActiveMQArtemisSecurity** CRD で定義されていない JAAS ログインモジュールを設定してユーザーを認証できます。たとえば、LDAP ログインモジュールまたはその他の JAAS ログインモジュールを設定できます。

AMQ Broker の認証と承認を設定する両方の方法については、次のセクションで説明します。

4.3.1. シークレットでの JAAS ログインモジュールの設定

AMQ Broker でユーザーを認証するために、シークレットで JAAS ログインモジュールを設定できます。シークレットを作成した後、メインブローカーのカスタムリソース (CR) にシークレットへの参照を追加し、ユーザーに AMQ Broker へのアクセスを許可する権限を CR に設定する必要があります。

手順

1. 新しい JAAS ログインモジュール設定を含むテキストファイルを作成し、そのファイルを **login.config** として保存します。ファイルを **login.config** として保存すると、テキストファイルから作成したシークレットに正しいキーが挿入されます。次に、ログインモジュールの設定例を示します。

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

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

シークレットで JAAS ログインモジュールを設定し、CR でシークレットへの参照を追加すると、デフォルトのログインモジュールは AMQ Broker で使用されなくなります。ただし、デフォルトのログインモジュールで参照される **artemis-users.properties** ファイル内のユーザーは、Operator がブローカーで認証する必要があります。新しい JAAS ログインモジュールを設定した後、Operator がブローカーで認証できることを確認するには、次のいずれかを行う必要があります。

- 上の例に示すように、新しいログインモジュール設定にデフォルトのプロパティーログインモジュールを含めます。この例では、デフォルトのプロパティーログインモジュールは **artemis-users.properties** ファイルと **artemis-roles.properties** ファイルを使用します。新しいログインモジュール設定にデフォルトのログインモジュールを含める場合は、**baseDir** を **/home/jboss/amq-broker/etc** ディレクトリーに設定する必要があります。このディレクトリーには、各ブローカー Pod のデフォルトのプロパティーファイルが含まれています。

- オペレーターがブローカーで認証するために必要なユーザーとロールの情報を、新しいログインモジュール設定で参照されるプロパティーファイルに追加します。この情報は、ブローカー Pod の **/home/jboss/amq-broker/etc directory** にあるデフォルトの **artemis-users.properties** ファイルおよび **artemis-roles.properties** ファイルからコピーできます。

注記

ログインモジュールで参照されるプロパティーファイルは、ブローカーが初めてログインモジュールを呼び出したときにのみロードされます。ブローカーは、ユーザーを認証するためのログインモジュールが見つかるまで、**login.config** ファイルにリストされている順序でログインモジュールを呼び出します。Operator が使用する認証情報を含むログインモジュールを **login.config** ファイルの最後に配置すると、ブローカーが Operator を認証するときに、先行するすべてのログインモジュールが呼び出されます。その結果、プロパティーファイルがブローカー上で表示されないことを示すステータスメッセージはすべてクリアされます。

2. 作成した **login.config** ファイルにプロパティーログインモジュールが含まれている場合は、そ

2. 作成した **login.config** ファイルにプロパティーログインモジュールが含まれている場合は、モジュール内で指定されたユーザーおよびロールファイルにユーザーおよびロールの情報が含まれていることを確認してください。以下に例を示します。

new-users.properties

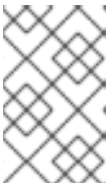
```
ruben=ruben01!
anne=anne01!
rick=rick01!
bob=bob01!
```

new-roles.properties

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

3. **oc create secret** コマンドを使用して、新しいログインモジュール設定で作成したテキストファイルからシークレットを作成します。ログインモジュール設定にプロパティーログインモジュールが含まれている場合は、関連するユーザーとロールファイルもシークレットに含めます。以下に例を示します。

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



注記

シークレットにログインモジュール設定が含まれていることを Operator が認識し、更新を各ブローカー Pod に伝播できるように、シークレット名には接尾辞 **-jaas-config** が必要です。

シークレットの作成方法の詳細については、Kubernetes ドキュメントの シークレット を参照してください。

4. 作成したシークレットをブローカーデプロイメントのカスタムリソース (CR) インスタンスに追加します。

   a. OpenShift コマンドラインインターフェイスの使用:

      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。

      ii. デプロイメントの CR を編集します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。

      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)**Operator をクリックします。

iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。

iv. **AMQ Broker** タブをクリックします。

v. ActiveMQArtemis インスタンス名をクリックします。

vi. **YAML** タブをクリックします。
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。

5. **extraMounts** 要素と **secrets** 要素を作成し、シークレットの名前を追加します。次の例では、**custom-jaas-config** という名前のシークレットを CR に追加します。

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

6. CR で、ブローカー上で設定されているロールに権限を付与します。

a. CR の **spec** セクションで、**brokerProperties** 要素を追加し、権限を追加します。単一のアドレスにロール権限を付与できます。または、**#** 記号を使用してワイルドカード一致を指定し、すべてのアドレスにロールの権限を付与することもできます。以下に例を示します。

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

この例では、group2 ロールにはすべてのアドレスへの **send** 権限が割り当てられ、group1 ロールにはすべてのアドレスへの **consume**、**createAddress**、**createNonDurableQueue** および **browse** 権限が割り当てられます。

7. CR を保存します。
Operator は、**/amq/extra/secrets/secret name** ディレクトリー内のシークレットの **login.config** ファイルを各 Pod にマウントし、デフォルトの **login.config** ファイルの代わりに、マウントされた **login.config** ファイルを読み取るようにブローカー JVM を設定します。**login.config** ファイルにプロパティーログインモジュールが含まれている場合、参照されるユーザーとロールのプロパティーファイルも各 Pod にマウントされます。

8. CR のステータス情報を表示して、デプロイメント内のブローカーが認証にシークレット内の JAAS ログインモジュールを使用していることを確認します。

a. OpenShift コマンドラインインターフェイスの使用:

i. ブローカーの CR でステータス条件を取得します。

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

b. OpenShift Web コンソールの使用

i. CR で、**status** セクションに移動します。

c. ステータス情報に **JaasPropertiesApplied** タイプが存在することを確認します。これは、ブローカーがシークレットで設定された JAAS ログインモジュールを使用していることを示します。以下に例を示します。

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

シークレット内のいずれかのファイルを更新すると、OpenShift Container Platform がシークレット内の最新のファイルを各ブローカー Pod に伝播するまで、**reason** フィールドの値に **OutofSync** が表示されます。たとえば、新しいユーザーを **new-users-properties** ファイルに追加してシークレットを更新すると、更新されたファイルが各 Pod に伝播されるまで、次のステータス情報が表示されます。

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

9. シークレットで参照されるプロパティーファイル内のユーザーまたはロール情報を更新する場合は、**oc set data** コマンドを使用してシークレットを更新します。**login.config** ファイルを含むすべてのファイルをシークレットに再度追加する必要があります。たとえば、この手順の前半で作成した **new-users.properties** ファイルに新しいユーザーを追加する場合は、次のコマンドを使用して、**custom-jaas-config** シークレットを更新します。

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

注記

ブローカー JVM は、起動時のみ、**login.config** ファイル内の設定を読み取ります。たとえば、新しいログインモジュールを追加してシークレットを更新するために、**login.config** ファイルの設定を変更した場合、ブローカーは再起動されるまで新しい設定を使用しません。

関連情報

「JAAS ログインモジュール設定の例」

「例: Red Hat Single Sign-On を使用するように AMQ Broker を設定する」

JAAS ログインモジュールの形式については、JAAS ログイン設定ファイル を参照してください。

4.3.2. セキュリティーカスタムリソース (CR) を使用したデフォルトの JAAS ログインモジュールの設定

ActiveMQArtemisSecurity カスタムリソース (CR) を使用して、デフォルトの JAAS プロパティーログインモジュールでユーザーとロールの情報を設定し、AMQ Broker でユーザーを認証できます。シークレットを使用して AMQ Broker で認証と承認を設定する別の方法については、「シークレットでの JAAS ログインモジュールの設定」 を参照してください。

4.3.2.1. セキュリティーカスタムリソース (CR) を使用したデフォルトの JAAS ログインモジュールの設定

次の手順は、セキュリティーカスタムリソース (CR) を使用してデフォルトの JAAS ログインモジュールを設定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」
- ブローカーの保護 で説明されているブローカーのセキュリティーについて理解している必要があります。

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 をデプロイメントすると、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) インスタンスの設定を開始して、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イスの使用:
      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. デプロイメントの CR を編集します。

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

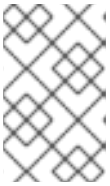
   b. OpenShift Container Platform Web コンソールの使用
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。
      ii. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
      iv. **AMQ Broker** タブをクリックします。

v. ActiveMQArtemis インスタンス名をクリックします。

vi. **YAML** タブをクリックします。
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。以下に例を示します。

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisSecurity
metadata:
  name: ex-prop
spec:
  loginModules:
    propertiesLoginModules:
      - name: "prop-module"
        users:
          - name: "sam"
            password: "samspassword"
            roles:
              - "sender"
          - name: "rob"
            password: "robspassword"
            roles:
              - "receiver"
  securityDomains:
    brokerDomain:
      name: "activemq"
      loginModules:
        - name: "prop-module"
          flag: "sufficient"
  securitySettings:
    broker:
      - match: "#"
        permissions:
          - operationType: "send"
            roles:
              - "sender"
          - operationType: "createAddress"
            roles:
              - "sender"
          - operationType: "createDurableQueue"
            roles:
              - "sender"
          - operationType: "consume"
            roles:
              - "receiver"
              ...
```



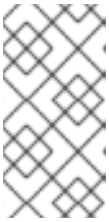
注記

前の例の要素には常に値を指定してください。たとえば、**securityDomains.brokerDomain** の値またはロールの値を指定しないと、設定によって予期しない結果が生じる可能性があります。

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、設定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イスの使用:
      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/security_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」

4.3.2.2. ユーザーパスワードをシークレットに保存する

「セキュリティーカスタムリソース (CR) を使用したデフォルトの JAAS ログインモジュールの設定」の手順では、ユーザーパスワードは **ActiveMQArtemisSecurity** CR にクリアテキストで保存されます。パスワードをクリアテキストで CR に保存したくない場合は、パスワードを CR から除外してシークレットに保存できます。CR を適用すると、Operator はシークレットから各ユーザーのパスワードを取得し、ブローカー Pod の **artemis-users.properties** ファイルに挿入します。

手順

1. **oc create secret** コマンドを使用してシークレットを作成し、各ユーザーの名前とパスワードを追加します。シークレット名は、**security-properties-module name** の命名規則に従う必要があります。ここで、**module name** は、CR で設定されたログインモジュールの名前です。以下に例を示します。

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

2. CR の **spec** セクションで、ロール情報とともにシークレットで指定したユーザー名を追加しますが、各ユーザーのパスワードは含めません。以下に例を示します。

```
apiVersion: broker.amq.io/v1beta1
kind: ActiveMQArtemisSecurity
metadata:
  name: ex-prop
spec:
  loginModules:
    propertiesLoginModules:
      - name: "prop-module"
        users:
          - name: "sam"
            roles:
              - "sender"
          - name: "rob"
            roles:
              - "receiver"
  securityDomains:
    brokerDomain:
      name: "activemq"
      loginModules:
        - name: "prop-module"
          flag: "sufficient"
  securitySettings:
    broker:
      - match: "#"
        permissions:
          - operationType: "send"
            roles:
              - "sender"
          - operationType: "createAddress"
            roles:
              - "sender"
          - operationType: "createDurableQueue"
            roles:
              - "sender"
          - operationType: "consume"
            roles:
              - "receiver"
              ...
```

3. 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イスの使用:

      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

OpenShift Container Platform のシークレットの詳細は、OpenShift Container Platform ドキュメントの Pod への機密データの提供 を参照してください。

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 つのブローカーで設定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。

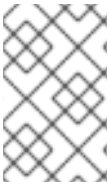
重要

OpenShift Container Platform で PV を手動でプロビジョニングする場合、各 PV の回収ポリシーを **Retain** に設定していることを確認してください。PV の回収ポリシーが **Retain** に設定されておらず、Operator が PV を要求するために使用した PVC が削除されている場合、PV も削除されます。PV を削除すると、ボリューム上のすべてのデータが失われます。回収ポリシーの設定の詳細は、OpenShift Container Platform ドキュメントの 永続ストレージについて を参照してください。

デフォルトでは、PVC は、クラスター用に設定されたデフォルトのストレージクラスから、ブローカーごとに 2 GiB のストレージを取得します。PVC で要求されたデフォルトのサイズとストレージクラスをオーバーライドできますが、CR を初めてデプロイする **前** に、CR で新しい値を設定することによってのみ可能です。

4.4.1. ブローカーのストレージサイズとストレージクラスの設定

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) のサイズとストレージクラスを指定する方法を説明します。



注記

AMQ Broker のデプロイ後に CR でストレージ設定を変更した場合、更新された設定は既存の Pod に遡及的に適用されません。ただし、デプロイメントをスケールアップした場合に作成される新しい Pod には、更新された設定が適用されます。

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。
- 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、OpenShift Container Platform ドキュメントの永続ストレージについて 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イスの使用:
      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。

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

   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. ブローカーのストレージサイズを指定するには、CR の **deploymentPlan** セクションに **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:
  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、バイト表記 (K、M、G など) を使用する単位、または同等の 2 進数 (Ki、Mi、Gi) が含まれている **必要** があります。

3. 各ブローカー Pod が永続ストレージに必要なストレージクラスを指定するには、**storage** セクションで **storageClassName** 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:
  deploymentPlan:
    size: 1
    image: placeholder
    requireLogin: false
    persistenceEnabled: true
    journalType: nio
    messageMigration: true
    storage:
      size: 4Gi
      storageClassName: gp3
```

storage.storageClassName

永続ボリューム要求 (PVC) で要求するストレージクラスの名前。ストレージクラスは、管理者が使用可能なストレージを記述および分類する方法を提供します。たとえば、さまざまなストレージクラスが、特定のサービス品質レベル、バックアップポリシーなどにマッピングされる場合があります。
ストレージクラスを指定しない場合、クラスター用に設定されたデフォルトのストレージクラスを持つ永続ボリュームが PVC によって要求されます。

注記

ストレージクラスを指定すると、ボリュームのストレージクラスが指定されたストレージクラスと一致する場合にのみ、PVC によって永続ボリュームが要求されます。

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イスの使用:
      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

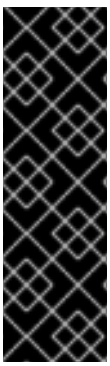
Pod で実行されている各ブローカーコンテナーについて、この値は、コンテナーが要求するホストノードメモリーの量になります。OpenShift スケジューラーは、Pod の配置時にメモリー要求の値を考慮し、ブローカー Pod を十分なコンピュートリソースを持つノードにバインドします。
メモリー要求値は、ブローカーコンテナーの実行に必要なメモリーの**最小**量です。ただし、コンテナーは可能な限り多くのメモリーを使用できます。メモリー制限を指定した場合、ブローカーコンテナーはそのメモリー量を超えることができません。

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イスの使用:

      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。

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

   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:
  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イスの使用:

      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. AMQ 管理コンソールへのアクセスの有効化

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。ブローカーデプロイメントのカスタムリソースインスタンスでコンソールへのアクセスを有効にすることができます。コンソールへのアクセスを有効にすると、コンソールを使用して Web ブラウザーでブローカーを表示および管理できるようになります。

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) インスタンスを編集します。
   a. OpenShift コマンドラインインターフェイスの使用:
      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. デプロイメントの CR を編集します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
      iv. **AMQ Broker** タブをクリックします。
      v. ActiveMQArtemis インスタンス名をクリックします。
      vi. **YAML** タブをクリックします。
      コンソール内で YAML エディターが開き、CR インスタンスを設定できるようになります。
2. CR の **spec** セクションに、**console** セクションを追加します。**console** セクションで、**expose** 属性を追加し、値をに設定します。**true**.以下に例を示します。

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

3. CR を保存します。

関連情報

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。

4.7. ブローカーコンテナーの環境変数の設定

ブローカーデプロイメントのカスタムリソース (CR) インスタンスでは、AMQ Broker コンテナーに渡される環境変数を設定できます。

たとえば、**TZ** などの標準環境変数を使用してタイムゾーンを設定したり、**JDK_JAVA_OPTIONS** を使用して起動時に Java ランチャーによって使用されるコマンドライン引数の先頭に引数を追加したりできます。または、AMQ Broker のカスタム変数 **JAVA_ARGS_APPEND** を使用して、Java ランチャーで使用されるコマンドライン引数にカスタム引数を追加できます。

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) インスタンスを編集します。
   a. OpenShift コマンドラインインターフェイスの使用:
      i. 以下のコマンドを入力します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
      iv. **AMQ Broker** タブをクリックします。
      v. ActiveMQArtemis インスタンス名をクリックします。
      vi. **YAML** タブをクリックします。
      コンソール内で YAML エディターが開き、CR インスタンスを設定できるようになります。
2. CR の **spec** セクションで、**env** 要素を追加し、AMQ Broker コンテナーに設定する環境変数を追加します。以下に例を示します。

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

```
    value: Europe/Vienna
  - name: JAVA_ARGS_APPEND
    value: --Hawtio.realm=console
  - name: JDK_JAVA_OPTIONS
    value: -XshowSettings:system
  ...
```

この例では、CR 設定には次の環境変数が含まれています。

- AMQ Broker コンテナーのタイムゾーンを設定するための **TZ**。
- **JAVA_ARGS_APPEND** は、認証に **console** という名前のレルムを使用するように AMQ 管理コンソールを設定します。
- **JDK_JAVA_OPTIONS** を使用して、Java **-XshowSettings:system** パラメーターを設定します。これは、Java 仮想マシンのシステムプロパティー設定を表示します。

注記

JDK_JAVA_OPTIONS 環境変数を使用して設定された値は、Java ランチャーで使用されるコマンドライン引数の先頭に付加されます。**JAVA_ARGS_APPEND** 環境変数を使用して設定された値は、ランチャーで使用される引数に追加されます。引数が重複した場合は、右端の引数が優先されます。

3. CR を保存します。

注記

AMQ_ 接頭辞を持つ AMQ Broker 環境変数を変更しないこと、また、**POD_NAMESPACE** 変数を変更する場合は、注意することを推奨します。

関連情報

- 環境変数の定義の詳細については、コンテナーの環境変数を定義する を参照してください。

4.8. ブローカーのデフォルトのメモリー制限をオーバーライドする

ブローカーに設定されているデフォルトのメモリー制限をオーバーライドできます。デフォルトでは、ブローカーには、ブローカーの Java 仮想マシンで使用可能な最大メモリーの半分が割り当てられます。次の手順は、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、デフォルトのメモリー制限を上書きする方法を示しています。

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. カスタムリソース (CR) インスタンスの設定を開始して、基本的なブローカーのデプロイメントを作成します。
   a. OpenShift コマンドラインインターフェイスの使用:

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_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. 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. 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。

たとえば、基本的なブローカーデプロイメントの CR は次のようになります。

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

2. CR の **spec** セクションに、**brokerProperties** セクションを追加します。**brokerProperties** セクション内で、**globalMaxSize** プロパティーを追加し、メモリー制限を指定します。以下に例を示します。

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

globalMaxSize プロパティーのデフォルトの単位は bytes です。デフォルトの単位を変更するには、値に m (MB の場合) または g (GB の場合) の接尾辞を追加します。

3. 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イスの使用:

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 の編集が終了したら、**Save** をクリックします。

4. (オプション) **globalMaxSize** プロパティーに設定した新しい値が、ブローカーに割り当てられたデフォルトのメモリー制限をオーバーライドすることを確認します。

a. AMQ 管理コンソールに接続します。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。

b. メニューから **JMX** を選択します。

c. **org.apache.activemq.artemis** を選択します。

d. **global** を検索します。

e. 表示されるテーブルで、**グローバル最大値** 列の値が **globalMaxSize** プロパティーに設定した値と同じであることを確認します。

4.9. 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。特定の状況では、カスタム Init コンテナーの使用が必要になる場合があります。たとえば、追加のランタイム依存関係である **.jar** ファイルをブローカーのインストールディレクトリーに含める場合です。

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-rhel8:7.11
```

- 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

- 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 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イスの使用:
      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 Container Platform にログインします。
      ii. デプロイメントの CR を編集します。

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

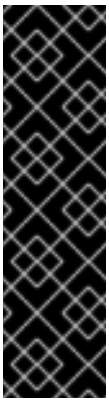
   b. OpenShift Container Platform Web コンソールの使用
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとして OpenShift Container Platform にログインします。
      ii. 左側のペインで、**Administration → Custom Resource Definitions**をクリックします。
      iii. **ActiveMQArtemis** CRD をクリックします。
      iv. **Instances** タブをクリックします。
      v. ブローカーデプロイメントのインスタンスをクリックします。
      vi. **YAML** タブをクリックします。
      コンソールで、YAML エディターが開き、CR インスタンスを編集できるようになります。

2. CR の **deploymentPlan** セクションで、**initImage** 属性を追加し、値をカスタム Init Container イメージの URL に設定します。

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

initImage

カスタム Init Container イメージの完全な URL を指定します。これはコンテナーレジストリーから入手できる必要があります。



重要

CR の **spec.deploymentPlan.initImage** 属性で指定されたカスタム init コンテナーイメージがある場合、Red Hat は、ブローカーイメージの自動アップグレードを防ぐために、**spec.deploymentPlan.image** 属性で対応するブローカーコンテナーイメージの URL も指定することを推奨します。**spec.deploymentPlan.image** 属性で特定のブローカーコンテナーイメージの URL を指定しない場合、ブローカーイメージは自動的にアップグレードされる可能性があります。ブローカーイメージがアップグレードされると、ブローカーとカスタム init コンテナーイメージのバージョンが異なるため、ブローカーが実行できなくなる可能性があります。

カスタム init コンテナーを含む動作するデプロイメントがある場合は、ブローカーコンテナーイメージのそれ以上のアップグレードを防止して、新しいブローカーイメージがカスタム init コンテナーイメージで動作しないリスクを排除できます。ブローカーイメージのアップグレード防止の詳細は、「イメージ URL を使用したイメージの自動アップグレードの制限」 を参照してください。

3. CR を保存します。

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。

4.10. 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

4.10.1. 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。

手順

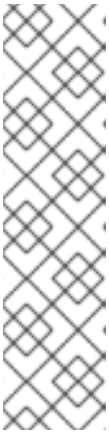
1. 初回インストール時にダウンロードしてデプロイメントした Operator アーカイブの **deploy/crs** ディレクトリーで、**broker_activemqartemis_cr.yaml** カスタムリソース (CR) ファイルを開きます。

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。
- ブローカーとクライアント間で、アクセプターが**相互認証**とも呼ばれる双方向 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。

4.10.2. 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。

注記

次の手順では、自己署名証明書を使用して一方向および双方向 TLS を設定する方法について説明します。自己署名証明書が Java 仮想マシン (JVM) トラストストア内に信頼できる証明書としてリストされている場合、JVM は証明書の有効期限を検証しません。実稼働環境では、Red Hat は認証局によって署名された証明書を使用することを推奨します。

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。

4.10.2.1. 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、コ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**個別にリスト表示**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,..."
```

4.10.2.2. 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=~/client.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
...
```

4.10.2.3. 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、フローカーとクライアントの両方が証明書を表示します。フ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
```

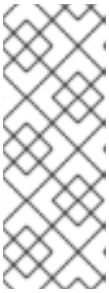
8. ブローカーのデプロイメントが含まれるプロジェクトに切り替えます。以下に例を示します。

```
$ 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
...
```

4.10.3. 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、内部ブローカークラスタリングに使用されるポート 61616 へのアクセスを提供します。

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。

4.10.4. 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。

4.10.4.1. 内部クライアントからのブローカーへの接続

内部クライアントをブローカーに接続するには、クライアント接続の詳細で、ブローカー Pod の DNS 解決可能な名前を指定します。以下に例を示します。

```
$ tcp://ex–aao-ss-0:<port>
```

内部クライアントがコアプロトコルを使用していて、**useTopologyForLoadBalancing=false** キーが接続 URL に設定されていない場合、クライアントがブローカーに初めて接続した後、ブローカーはクラスター内のすべてのブローカーのアドレスをクライアントに通知することができます。その後、クライアントはすべてのブローカー間で接続の負荷を分散できます。

ブローカーに永続的なサブスクリプションキューまたはリクエスト/応答キューがある場合は、クライアント接続の負荷が分散されているときにこれらのキューを使用する場合の注意事項に注意してください。詳細は、「永続的なサブスクリプションキューまたはリクエスト/要求キューがある場合のクライアント接続の負荷分散に関する警告」 を参照してください。

4.10.4.2. 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。

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
```

ブローカー Pod のルートの完全なホスト名は、OpenShift ルーターをホストしているノードに解決される必要があります。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。

外部クライアントでクラスター内のブローカー間で接続の負荷を分散する場合は、次のようにします。

- 各ブローカー Pod の OpenShift ルートで **haproxy.router.openshift.io/balance** ラウンドロビンオプションを設定して、ロードバランシングを有効にします。
- 外部クライアントがコアプロトコルを使用する場合は、クライアントの接続 URL に **useTopologyForLoadBalancing=false** キーを設定します。**useTopologyForLoadBalancing=false** キーを設定すると、クライアントは、ブローカーによって提供されるクラスタートポロジー情報に含まれる AMQ Broker Pod の DNS 名を使用できなくなります。Pod DNS 名は、外部クライアントがアクセスできない内部 IP アドレスに解決されます。

ブローカーに永続的なサブスクリプションキューまたはリクエスト/応答キューがある場合は、クライアント接続の負荷を分散するときにこれらのキューを使用する際の注意事項に注意してください。詳細は、「永続的なサブスクリプションキューまたはリクエスト/要求キューがある場合のクライアント接続の負荷分散に関する警告」 を参照してください。

外部クライアントがクラスター内のブローカー間で接続の負荷を分散しないようにする場合は、次のようにします。

- 各クライアントの接続 URL で、各フローカー Pod のルートの完全なホスト名を指定します。クライアントは、接続 URL の最初のホスト名に接続しようとします。ただし、最初のホスト名が使用できない場合、クライアントは接続 URL の次のホスト名に自動的に接続します。

- 外部クライアントがコアプロトコルを使用する場合は、クライアントの接続 URL に **useTopologyForLoadBalancing=false** キーを設定して、ブローカーが提供するクラスタートポロジー情報をクライアントが使用できないようにします。

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トコル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>
```

4.10.4.3. NodePort を使用したブローカーへの接続

Route 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。

4.10.4.4. 永続的なサブスクリプションキューまたはリクエスト/要求キューがある場合のクライアント接続の負荷分散に関する警告

永続サブスクリプション

永続サブスクリプションはブローカー上のキューとして表され、永続サブスクライバーが最初にブローカーに接続したときに作成されます。このキューは存在し、クライアントがサブスクライブを解除するまでメッセージを受信します。クライアントが別のブローカーに再接続すると、そのブローカーに別の永続サブスクリプションキューが作成されます。これにより、次の問題が発生する可能性があります。

問題	軽減策
メッセージは元のサブスクリプションキューで取り残される可能性があります。	メッセージの再配布が有効になっていることを確認します。詳細については、メッセージの再配布の有効化 を参照してください。
他のメッセージがまだルーティングされているときにメッセージの再配布中にウィンドウが表示されるため、メッセージが間違った順序で受信される可能性があります。	なし。

問題	軽減策
クライアントがサブスクライブを解除すると、最後に接続したブローカーでのみキューが削除されます。これは、他のキューがまだ存在してメッセージを受信できることを意味します。	サブスクライブを解除したクライアントに存在する可能性のある他の空のキューを削除するには、次の両方のプロパティーを設定します。 **auto-delete-queues-message-count** プロパティーを **0** に設定して、キューにメッセージがない場合にのみキューを削除できるようにします。**auto-delete-queues-delay** プロパティーを設定して、指定されたミリ秒数の間使用されなかった後にメッセージがないキューを削除します。 詳細については、アドレスとキューの自動作成と削除の設定 を参照してください。

リクエスト/応答キュー

JMS プロデューサーが一時応答キューを作成すると、ブローカー上にキューが作成されます。作業キューから消費して一時キューに応答しているクライアントが別のブローカーに接続すると、次の問題が発生する可能性があります。

問題	軽減策
クライアントが接続しているブローカーに応答キューが存在しないため、クライアントがエラーを生成する可能性があります。	**auto-create-queues** プロパティーが **true** に設定されていることを確認します。詳細については、アドレスとキューの自動作成と削除の設定 を参照してください。
ワークキューに送信されたメッセージは配信されない場合があります。	**message-load-balancing** プロパティーを **ON-DEMAND** に設定して、メッセージがオンデマンドで負荷分散されるようにします。また、メッセージの再配布が有効になっていることを確認してください。詳細については、メッセージの再配布の有効化 を参照してください。

関連情報

- クラスターで実行されているサービスを使用して 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。
  - OpenShift Container Platform ドキュメントの イングレスクラスタートラフィックの設定の概要。

4.11. 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。

注記

AMQP プロトコルのブローカー設定でのみ、大きなメッセージサイズの制限を設定できます。AMQ Core および Openwire プロトコルの場合、クライアント接続設定で大きなメッセージサイズの制限を設定できます。詳細は、Red Hat AMQ Clients のドキュメント を参照してください。

4.11.1. 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イスの使用:

   ```
   $ 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 namespace 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。

4.12. ブローカーヘルスチェックの設定

AMQ Broker でヘルスチェックを設定するには、startup プローブ、liveness プローブ、および readiness プローブを使用します。

- startup プローブは、コンテナー内のアプリケーションが起動しているかどうかを示します。
- liveness プローブは、コンテナーが実行中かどうかを判別します。
- readiness プローブは、コンテナーがサービス要求を受け入れることができるかどうかを判別します。

Pod の startup プローブまたは liveness プローブのチェックが失敗した場合、プローブは Pod を再起動します。

AMQ Broker には、デフォルトの readiness プローブと liveness プローブが含まれています。デフォルトの liveness プローブは、ブローカーの HTTP ポートに ping を実行して、ブローカーが実行されているかどうかを確認します。デフォルトの readiness プローブは、ブローカー用に設定された各アクセプターポートへの接続を開くことにより、ブローカーがネットワークトラフィックを受け入れることができるかどうかを確認します。

デフォルトの liveness プローブと readiness プローブの使用に際しての限界は、ブローカーのファイルシステムの問題など、根本的な問題を特定できないことです。より包括的なヘルスチェックを実行するには、ブローカーのコマンドラインユーティリティー **artemis** を使用するカスタムの liveness プローブと readiness プローブを作成します。

AMQ Broker にはデフォルトの startup プローブが含まれていません。**ActiveMQArtemis** カスタムリソース (CR) で startup プローブを設定できます。

4.12.1. startup プローブの設定

startup プローブを設定して、ブローカーコンテナー内の AMQ Broker アプリケーションが起動したかどうかを確認できます。

手順

1. ブローカーデプロイメントの 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イスの使用:
      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 Container Platform にログインします。
      ii. デプロイメントの CR を編集します。

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

   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 OpenShift Container Platform 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. ブローカーデプロイメントのインスタンスをクリックします。
      vi. **YAML** タブをクリックします。
      コンソールで、YAML エディターが開き、CR インスタンスを編集できるようになります。
2. CR の **deploymentPlan** セクションに、**startupProbe** セクションを追加します。以下に例を示します。

```
spec:
  deploymentPlan:
    startupProbe:
      exec:
        command:
          - /bin/bash
          - '-c'
          - /opt/amq/bin/artemis
          - 'check'
          - 'node'
          - '--up'
          - '--url'
          - 'tcp://$HOSTNAME:61616'
      initialDelaySeconds: 5
      periodSeconds: 10
      timeoutSeconds: 3
      failureThreshold: 30
```

command

コンテナー内で実行する startup プローブコマンド。この例では、startup プローブは **artemis check node** コマンドを使用して、AMQ Broker がブローカー Pod のコンテナー内で起動していることを確認します。

initialDelaySeconds

コンテナーの起動後にプローブが実行されるまでの遅延 (秒単位)。デフォルトは **0** です。

periodSeconds

プローブが実行される間隔 (秒単位)。デフォルトは **10** です。

timeoutSeconds

startup プローブコマンドがブローカーからの応答を待つ時間 (秒単位)。コマンドに対する応答が受信されない場合、コマンドは終了します。デフォルト値は **1** です。

failureThreshold

startup プローブが失敗したとみなされるまでの、startup プローブの連続失敗回数の最小値 (タイムアウトを含む)。プローブが失敗したとみなされると、Pod が再起動されます。デフォルト値は **3** です。
クラスターのリソースとブローカージャーナルのサイズによっては、ブローカーが起動してプローブチェックに合格するのに十分な時間を確保するために、失敗しきい値を引き上げる必要がある場合があります。そうしなければ、ブローカーはループ状態に入って繰り返し失敗しきい値に達し、そのたびに startup プローブによって再起動されることになります。たとえば、**failureThreshold** を **30** に設定し、プローブがデフォルトの 10 秒間隔で実行される場合、ブローカーが起動してプローブチェックに合格するまでに 300 秒が確保されます。

3. CR を保存します。

関連情報

OpenShift Container Platform の liveness プローブと readiness プローブの詳細については、OpenShift Container Platform ドキュメントの ヘルスチェックを使用したアプリケーションの健全性の監視 を参照してください。

4.12.2. liveness および readiness プローブの設定

次の例は、liveness および readiness プローブを使用して可用性チェックを実行するように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) インスタンスを設定する方法を示しています。

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 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イスの使用:

      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 Container Platform にログインします。

      ii. デプロイメントの CR を編集します。

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

   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 OpenShift Container Platform 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. ブローカーデプロイメントのインスタンスをクリックします。

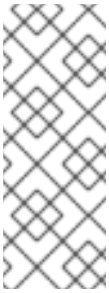
      vi. **YAML** タブをクリックします。

2. liveness プローブを設定するには、CR の **deploymentPlan** セクションに **livenessProbe** セクションを追加します。以下に例を示します。

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

initialDelaySeconds

コンテナーの起動後にプローブが実行されるまでの遅延 (秒単位)。デフォルトは **5** です。



注記

デプロイメントに startup プローブも設定されている場合は、liveness プローブと readiness プローブの両方で遅延を 0 に設定できます。これらのプローブは両方とも、startup プローブに合格した後にのみ実行されます。startup プローブにすでに合格している場合は、ブローカーが正常に起動したことが確認できるため、liveness プローブと readiness プローブの実行を遅らせる必要はありません。

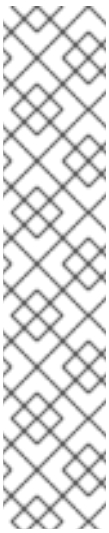
periodSeconds

プローブが実行される間隔 (秒単位)。デフォルトは **5** です。

failureThreshold

プローブが失敗したことを示す、liveness プローブの連続失敗回数の最小値 (タイムアウトを含む)。プローブが失敗すると、Pod が再起動されます。デフォルト値は 3 です。
デプロイメントに、liveness プローブの実行前にブローカーアプリケーションが起動していることを確認する startup プローブが設定されていない場合は、ブローカーが起動して liveness プローブのチェックに合格するのに十分な時間を確保するために、失敗しきい値を引き上げる必要がある場合があります。そうしなければ、ブローカーがループ状態に入って繰り返し失敗しきい値に達し、そのたびにブローカー Pod が liveness プローブによって再起動される可能性があります。

ブローカーが起動して liveness プローブのチェックに合格するまでに必要な時間は、クラスターのリソースとブローカージャーナルのサイズによって異なります。たとえば、**failureThreshold** を 30 に設定し、プローブがデフォルトの 5 秒間隔で実行される場合、ブローカーが起動して liveness プローブのチェックに合格するまでに 150 秒が確保されます。



注記

liveness プローブを設定していない場合、または設定されたプローブからハンドラーが欠落している場合、AMQ Broker Operator は次の設定を持つデフォルトの TCP プローブを作成します。デフォルトの TCP プローブは、指定されたポートでブローカーコンテナーへのソケットを開こうとします。

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

3. readiness プローブを設定するには、CR の **deploymentPlan** セクションに、**readinessProbe** セクションを追加します。以下に例を示します。

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

readiness プローブを設定しない場合、ビルトイン スクリプト は、すべてのアクセプターが接続を受け入れることができるかどうかをチェックします。

4. より包括的な可用性チェックを設定する場合は、**artemis check** コマンドラインユーティリティーを liveness または readiness プローブの設定に追加します。

   a. ブローカーへの完全なクライアント接続を作成する可用性チェックを設定する場合は、**livenessProbe** または **readinessProbe** セクションに **exec** セクションを追加します。**exec** セクションに、**command** セクションを追加します。**command** セクションで、**artemis check node** コマンド構文を追加します。以下に例を示します。

```
spec:
  deploymentPlan:
    readinessProbe:
      exec:
        command:
          - bash
          - '-c'
          - /home/jboss/amq-broker/bin/artemis
          - check
          - node
          - '--silent'
          - '--acceptor'
          - <acceptor name>
          - '--user'
          - $AMQ_USER
          - '--password'
          - $AMQ_PASSWORD
      initialDelaySeconds: 30
      timeoutSeconds: 5
```

デフォルトでは、**artemis check node** コマンドは **artemis** と呼ばれるアクセプターの URI を使用します。ブローカーに **artemis** というアクセプターがある場合は、コマンドから **--acceptor <acceptor name>** オプションを除外できます。

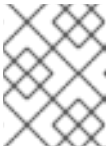
注記

$AMQ_USER および **$AMQ_PASSWORD** は、AMQ Operator によって設定される環境変数です。

b. メッセージを生成および消費する可用性チェックを設定し、ブローカーのファイルシステムの可用性も検証する場合は、**livenessProbe** または **readinessProbe** セクションに **exec** セクションを追加します。**exec** セクションに、**command** セクションを追加します。**command** セクションで、**artemis check queue** コマンド構文を追加します。以下に例を示します。

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

```
      - $AMQ_PASSWORD
    initialDelaySeconds: 30
    timeoutSeconds: 5
```



注記

指定するキュー名は、ブローカーで設定され、**anycast** の **routingType** を持っている必要があります。以下に例を示します。

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

5. CR を保存します。

関連情報

OpenShift Container Platform の liveness プローブと readiness プローブの詳細については、OpenShift Container Platform ドキュメントの ヘルスチェックを使用したアプリケーションの健全性の監視 を参照してください。

4.13. クラスターのスケールダウンをサポートするためのメッセージ移行の有効化

クラスター内のブローカーの数をスケールダウンして、クラスター内の残りの Pod にメッセージを移行できるようにするには、メッセージ移行を有効にする必要があります。

メッセージ移行が有効になっているクラスターをスケールダウンすると、スケールダウンコントローラーがメッセージ移行プロセスを管理します。

4.13.1. メッセージ移行プロセスの手順

メッセージ移行プロセスは、次の手順を実行します。

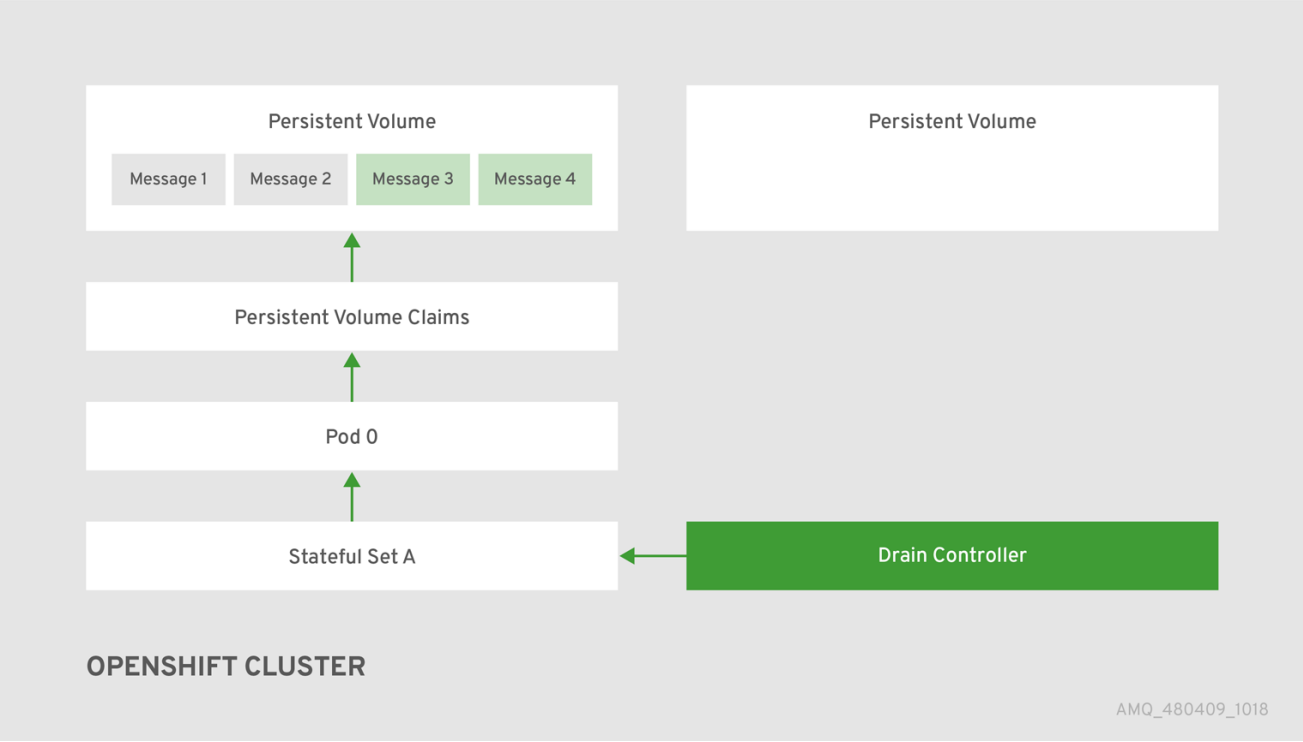
1. デプロイメントの意図的なスケールダウンにより、デプロイメント内のブローカー Pod がシャットダウンすると、Operator は自動的にスケールダウンカスタムリソースをデプロイして、メッセージ移行の準備をします。

2. 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。

3. 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、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。

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。



メッセージを動作中のブローカー 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

PV の回収ポリシーが **retain** に設定されている場合、PV を削除して再作成するまで、その PV を別の Pod で使用することはできません。たとえば、クラスターをスケールダウンした後にスケールアップした場合、PV を削除して再作成するまで、起動した Pod で PV を使用することはできません。

関連情報

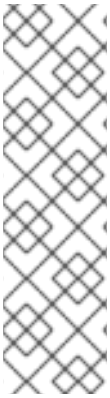
- ブローカーのデプロイメントをスケールダウンする際のメッセージ移行の例については、「メッセージ移行の有効化」 を参照してください。

4.13.2. メッセージ移行の有効化

ActiveMQArtemis カスタムリソース (CR) でメッセージ移行を有効にすることができます。

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、「メッセージ移行プロセスの手順」 を参照してください。



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。

手順

1. ブローカーデプロイメントの 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イスの使用:
      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 Container Platform にログインします。
      ii. デプロイメントの CR を編集します。

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

   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 OpenShift Container Platform 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. ブローカーデプロイメントのインスタンスをクリックします。
      vi. **YAML** タブをクリックします。
      コンソールで、YAML エディターが開き、CR インスタンスを編集できるようになります。
2. CR の **deploymentPlan** セクションで、**messageMigration** 属性を追加し、**true** に設定します。設定されていない場合は、**persistenceEnabled** 属性を追加し、**true** に設定します。以下に例を示します。

```
spec:
  deploymentPlan:
    messageMigration: true
```

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

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. CR を保存します。

4. (オプション) 以下の手順を実行してクラスターをスケールダウンし、メッセージ移行プロセスを表示します。

   a. 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。

   b. 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。

      i. 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/bin/artemis producer --url tcp://172.17.0.6:61616 --user admin --password admin
      ```

   c. 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/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。

   d. 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。

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

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

      iii. 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**)。

e. 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。

4.14. OPENSHIFT CONTAINER PLATFORM ノードでのブローカー POD の配置の制御

ノードセレクター、容認、またはアフィニティーおよび非アフィニティールールを使用して、OpenShift Container Platform ノード上の AMQ Broker Pod の配置を制御できます。

ノードセレクター

ノードセレクターを使用すると、特定のノードでブローカー Pod をスケジュールできます。

Tolerations

容認がノードに設定されたテイントと一致する場合、容認によりノードでブローカー Pod をスケジュールできます。Pod 容認が一致しない場合、テイントにより、ノードは Pod の受け入れを拒否できます。

アフィニティー/非アフィニティー

ノードアフィニティールールは、ノードのラベルに基づいて Pod をスケジュールできるノードを制御します。Pod のアフィニティールールと非アフィニティールールは、そのノードですでに実行されている Pod に基づいて、Pod をスケジュールできるノードを制御します。

4.14.1. ノードセレクターの使用による特定ノードへの Pod の配置

ノードセレクターは、ノードラベルに一致するキーと値のペアを持つノードでブローカー Pod をスケジュールする必要があるキーと値のペアを指定します。

次の例は、特定のノードでブローカー Pod をスケジュールするようにノードセレクターを設定する方法を示しています。

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。
- ブローカー Pod をスケジュールする OpenShift Container Platform ノードにラベルを追加します。ノードラベルの追加の詳細は、OpenShift Container Platform ドキュメントの ノードセレクターを使用して Pod の配置を制御する を参照してください。

手順

1. メインブローカー CRD に基づいてカスタムリソース (CR) インスタンスを作成します。

   a. OpenShift コマンドラインインターフェイスの使用:

      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_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. 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. 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。

2. CR の **deploymentPlan** セクションで、**nodeSelector** セクションを追加し、Pod のノードを選択するために一致させたいノードラベルを追加します。以下に例を示します。

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

この例では、ブローカー Pod は **app: broker1** ラベルを持つノードでスケジュールされます。

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イスの使用:

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

OpenShift Container Platform のノードセレクターの詳細は、OpenShift Container Platform ドキュメントの ノードセレクターを使用した特定のノードへの Pod の配置 を参照してください。

4.14.2. 容認を使用した Pod の配置の制御

テイントと容認は、特定のノードで Pod をスケジュールできるかできないかを制御します。テイントにより、Pod に一致する容認がない限り、ノードは Pod のスケジュールを拒否できます。テイントを使用すると、ノードから Pod を除外して、ブローカー Pod など、一致する容認を持つ特定の Pod 用に

ノードを予約することができます。

一致する容認を持つことは、ブローカー Pod をノード上にスケジュールすることを許可しますが、Pod がそのノード上にスケジュールされることを保証するものではありません。テイントが設定されているノードでブローカー Pod が確実にスケジュールされるようにするために、アフィニティールールを設定できます。詳細は、「アフィニティールールと非アフィニティールールを使用した Pod の配置の制御」 を参照してください。

次の例は、ノードで設定されているテイントに一致する容認を設定する方法を示しています。

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。
- ブローカー Pod をスケジュールするために予約するノードにテイントを適用します。テイントは、key、value、および effect で構成されています。テイント effect は、以下を決定します。
  - ノード上の既存の Pod が削除されるかどうか
  - 既存の Pod をノードに残すことができるかどうか (ただし、新しい Pod は容認が一致しない限り、スケジュールすることはできない)
  - 必要に応じてノードで新しい Pod をスケジュールできるかどうか (ただし、ノードで新しい Pod をスケジュールしないことが優先される)

テイントの適用の詳細は、OpenShift Container Platform ドキュメントの ノードテイントを使用した Pod 配置の制御 を参照してください。

手順

1. メインブローカー CRD に基づいてカスタムリソース (CR) インスタンスを作成します。
   a. OpenShift コマンドラインインターフェイスの使用:
      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_activemqartemis_cr.yaml** というサンプル CR ファイルを開きます。

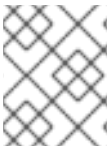
   b. OpenShift Container Platform Web コンソールの使用
      i. ブローカーデプロイメントのプロジェクトに CR をデプロイする権限を持つユーザーとしてコンソールにログインします。
      ii. メインブローカー CRD に基づいて新規 CR インスタンスを起動します。左側のペインで、**Administration** → **Custom Resource Definitions** をクリックします。
      iii. **ActiveMQArtemis** CRD をクリックします。
      iv. **Instances** タ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。

2. CR の **deploymentPlan** セクションに、**tolerations** セクションを追加します。**tolerations** セクションで、一致させたいノードテイントの容認を追加します。以下に例を示します。

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

この例では、容認は **app=amq-broker:NoSchedule** のノードテイントと一致するため、このテイントが設定されているノードで Pod をスケジュールできます。



注記

ブローカー Pod が正しくスケジュールされるようにするには、CR の **tolerations** セクションで **tolerationsSeconds** 属性を指定しないでください。

1. 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イスの使用:
      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

OpenShift Container Platform のテイントと容認の詳細は、OpenShift Container Platform ドキュメントの ノードテイントを使用した Pod 配置の制御 を参照してください。

4.14.3. アフィニティールールと非アフィニティールールを使用した Pod の配置の制御

ノードアフィニティールール、Pod アフィニティールール、または Pod 非アフィニティールールを使用して、Pod の配置を制御できます。ノードアフィニティーにより、Pod はターゲットノードのグループに対するアフィニティーを指定できます。Pod のアフィニティーと非アフィニティーを使用すると、ノードですでに実行されている他の Pod に対して、Pod をどのように相対的にスケジュールできるか、またはできないかについてのルールを指定することができます。

4.14.3.1. ノードアフィニティールールを使用した Pod の配置の制御

ノードアフィニティーは、ブローカー Pod が配置可能なノードのグループに対するアフィニティーを指定することができます。ブローカー Pod は、Pod 用に作成したアフィニティールールと同じキーと値のペアを持つラベルを持つ任意のノードでスケジュールできます。

次の例は、ノードアフィニティールールを使用して Pod の配置を制御するようにブローカーを設定する方法を示しています。

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。
- ブローカー Pod をスケジュールできる OpenShift Container Platform クラスター内のノードに共通のラベルを割り当てます (例: **zone: emea**)。

手順

1. メインブローカー CRD に基づいてカスタムリソース (CR) インスタンスを作成します。
   a. OpenShift コマンドラインインターフェイスの使用:
      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_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. 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. 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。
2. CR の **deploymentPlan** セクション
   に、**affinity**、**nodeAffinity**、**requiredDuringSchedulingIgnoredDuringExecution**、および **nodeSelectorTerms** の各セクションを追加します。**nodeSelectorTerms** セクションで、**-matchExpressions** パラメーターを追加し、一致させるノードラベルのキーと値の文字列を指定します。以下に例を示します。

   ```
   spec:
     deploymentPlan:
       affinity:
         nodeAffinity:
           requiredDuringSchedulingIgnoredDuringExecution:
             nodeSelectorTerms:
   ```

```
      - matchExpressions:
        - key: zone
          operator: In
          values:
          - emea
```

この例では、アフィニティールールにより、キーが **zone** で値が **emea** のラベルを持つ任意のノードで Pod をスケジュールできます。

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イスの使用:

      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

OpenShift Container Platform のアフィニティールールの詳細は、OpenShift Container Platform ドキュメントの ノードアフィニティールールを使用したノード上の Pod 配置の制御 を参照してください。

4.14.3.2. 非アフィニティールールを使用して Pod を他の Pod に相対的に配置する

非アフィニティールールを使用すると、そのノードですでに実行されている Pod のラベルに基づいて、ブローカー Pod をスケジュールできるノードを制限することができます。

非アフィニティールールのユースケースとしては、クラスター内の複数のブローカー Pod が同じノードにスケジュールされないようにすることで、単一障害点を発生させないようにすることが挙げられます。Pod の配置を制御しない場合、クラスター内の 2 つ以上のブローカー Pod を同じノードでスケジュールすることができます。

次の例は、非アフィニティールールを設定して、クラスター内の 2 つのブローカー Pod が同じノードでスケジュールされないようにする方法を示しています。

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. メインブローカーの CRD に基づいて、クラスター内の最初のブローカーの CR インスタンスを作成します。

a. OpenShift コマンドラインインターフェイスの使用:

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_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. 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. 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。

2. CR の **deploymentPlan** セクションに、**labels** セクションを追加します。最初のブローカー Pod の識別ラベルを作成して、2 番目のブローカー Pod に非アフィニティールールを作成し、両方の Pod が同じノードでスケジュールされないようにします。以下に例を示します。

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

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イスの使用:

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. メインブローカーの CRD に基づいて、クラスター内の 2 番目のブローカーの CR インスタンスを作成します。

a. CR の **deploymentPlan** セクションに、**affinity**、**podAntiAffinity**、**requiredDuringSchedulingIgnoredDuringExecution**、および **labelSelector** の各セクションを追加します。**labelSelector** セクションで、**matchExpressions** パラメーターを追加し、一致するブローカー Pod ラベルのキーと値の文字列を指定して、この Pod が同じノードでスケジュールされないようにします。

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

この例では、Pod 非アフィニティールールにより、キーが **name** で値が **broker1** のラベル (クラスター内の最初のブローカーに割り当てられたラベル) を持つ Pod と同じノードに Pod が配置されないようにします。

5. 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イスの使用:

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

OpenShift Container Platform のアフィニティールールの詳細は、OpenShift Container Platform ドキュメントの ノードアフィニティールールを使用したノード上の Pod 配置の制御 を参照してください。

4.15. ブローカーのログの設定

AMQ Broker は、Log4j 2 ログユーティリティーを使用してメッセージログを提供します。ブローカーをデプロイすると、デフォルトの Log4j 2 設定が使用されます。デフォルト設定を変更する場合は、シークレットまたは ConfigMap で新しい Log4j 2 設定を作成する必要があります。シークレットまたは ConfigMap の名前をメインブローカーのカスタムリソース (CR) に追加すると、Operator は新しいログ設定を使用するように各ブローカーを設定します。このログ設定は、Operator が各 Pod にマウントするファイルに保存されます。

前提条件

- Log4j 2 設定オプションについて理解している。

手順

1. AMQ Broker で使用する Log4j 2 設定を含むファイルを準備します。
   ブローカーによって使用されるデフォルトの Log4j 2 設定ファイルは、各ブローカー Pod の **/home/jboss/amq-broker/etc/log4j2.properties** ファイルにあります。デフォルト設定ファイルの内容をベースとして、シークレットまたは ConfigMap 内に新しい Log4j 2 設定を作成できます。デフォルトの Log4j 2 設定ファイルの内容を取得するには、次の手順を実行します。

   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。

      iv. **cat** コマンドを使用して、ブローカー Pod 上の **/home/jboss/amq-broker/etc/log4j2.properties** ファイルの内容を表示し、その内容をコピーします。

      v. OpenShift Container Platform CLI がインストールされているローカルファイルに内容を貼り付け、ファイルを **logging.properties** として保存します。

   b. OpenShift コマンドラインインターフェイスの使用:

      i. デプロイメント内の Pod の名前を取得します。

```
$ 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. **oc cp** コマンドを使用して、ログ設定ファイルを Pod からローカルディレクトリーにコピーします。

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

      ここで、コンテナー名の <name> 部分は、Pod 名の **-ss** 文字列の前の接頭辞です。以下に例を示します。

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

注記

ファイルから ConfigMap またはシークレットを作成する場合、ConfigMap またはシークレット内のキーはデフォルトでファイル名になり、値はデフォルトでファイルの内容になります。**logging.properties** という名前のファイルからシークレットを作成すると、新しいログ設定に必要なキーがシークレットまたは ConfigMap に挿入されます。

2. **logging.properties** ファイルを編集し、AMQ Broker で使用する Log4j 2 設定を作成します。たとえば、デフォルト設定では、AMQ Broker はコンソールのみにメッセージを記録します。AMQ Broker がメッセージをディスクにも記録するように設定を更新することもできます。

3. 更新された Log4j 2 設定をシークレットまたは ConfigMap に追加します。

   a. ブローカーデプロイメントのプロジェクトでシークレットまたは ConfigMap を作成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>
   ```

   b. シークレットでログ設定を行う場合は、**oc create secret** コマンドを使用します。以下に例を示します。

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

   c. ConfigMap でログ設定を行う場合は、**oc create ConfigMap** コマンドを使用します。以下に例を示します。

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

   ConfigMap またはシークレットの名前には、Operator がシークレットに新しいログ設定が含まれていることを認識できるように、**-logging-config** という接尾辞が必要です。

4. シークレットまたは ConfigMap をブローカーデプロイメントのカスタムリソース (CR) インスタンスに追加します。

   a. OpenShift コマンドラインインターフェイスの使用:

      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. CR を編集します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。

      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。

      iv. **AMQ Broker** タブをクリックします。

      v. ActiveMQArtemis インスタンス名をクリックします。

      vi. **YAML** タブをクリックします。
      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。

c. Log4j 2 ログ設定を含むシークレットまたは ConfigMap を CR に追加します。次の例は、CR に追加されたシークレットと ConfigMap を示しています。

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

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

5. CR を保存します。

各ブローカー Pod で、Operator は、作成したシークレットまたは ConfigMap 内のログ設定を含む **logging.properties** ファイルをマウントします。さらに、Operator は、デフォルトのログ設定ファイルの代わりにマウントされたログ設定ファイルを使用するように各ブローカーを設定します。

注記

ConfigMap またはシークレットでログ設定を更新すると、各ブローカーは更新されたログ設定を自動的に使用します。

4.16. POD の DISRUPTION BUDGET の設定

Pod の Disruption Budget は、メンテナンス期間などの自主的な中断中に同時に使用可能にする必要があるクラスター内の Pod の最小数を指定します。

手順

1. ブローカーデプロイメントの 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イスの使用:

      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 Container Platform にログインします。

      ii. デプロイメントの CR を編集します。

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

-

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 OpenShift Container Platform 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. ブローカーデプロイメントのインスタンスをクリックします。

vi. **YAML** タブをクリックします。
コンソールで、YAML エディターが開き、CR インスタンスを編集できるようになります。

2. CR の **spec** セクションで、**podDisruptionBudget** 要素を追加し、自発的な中断中に使用できるデプロイメント内の Pod の最小数を指定します。次の例では、少なくとも 1 つの Pod が使用可能である必要があります。

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

3. CR を保存します。

関連情報

Pod の Disruption Budget の詳細は、OpenShift Container Platform ドキュメントの Pod の Disruption Budget (停止状態の予算) を使用して起動している Pod の数を指定する方法 を参照してください。

4.17. カスタムリソース定義で公開されていない項目の設定

カスタムリソース定義 (CRD) は、AMQ Broker 用に変更できる設定項目のスキーマです。対応するカスタムリソース (CR) インスタンスの CRD にある設定項目の値を指定できます。Operator は、CR インスタンスから各ブローカーコンテナーの設定を生成します。

CRD で公開されていない設定項目を CR に含めるには、項目を **brokerProperties** 属性に追加します。**brokerProperties** 属性に含まれる項目はシークレットに保存され、ブローカー Pod 上にプロパティーファイルとしてマウントされます。起動時に、XML 設定が適用された後、プロパティーファイルが内部 Java 設定 Bean に適用されます。

次の例では、単一のプロパティーが設定 Bean に適用されます。

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

次の例では、複数のプロパティーが設定 Bean のネストされたコレクションに適用され、別のブローカーとメッセージをミラーリングする **target** という名前のブローカー接続が作成されます。

```
spec:
  ...
  brokerProperties
  - "AMQPConnections.target.uri=tcp://<hostname>:<port>"
  - "AMQPConnections.target.connectionElements.mirror.type=MIRROR"
  - "AMQPConnections.target.connectionElements.mirror.messageAcknowledgements=true"
  - "AMQPConnections.target.connectionElements.mirror.queueCreation=true"
  - "AMQPConnections.target.connectionElements.mirror.queueRemoval=true"
  ...
```

重要

brokerProperties 属性を使用すると、他の方法では OpenShift Container Platform 上の AMQ Broker に設定できない多くの設定項目にアクセスできるようになります。一部のプロパティーは、誤って使用すると、デプロイメントに重大な影響を与える可能性があります。この方法を使用してプロパティーを設定する場合は、常に注意してください。

手順

1. デプロイメントの CR を編集します。
   a. OpenShift Web コンソールの使用
      i. 以下のコマンドを入力します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
      iv. **AMQ Broker** タブをクリックします。
      v. ActiveMQArtemis インスタンス名をクリックします。
      vi. **YAML** タブをクリックします。
      コンソールで、YAML エディターが開き、CR インスタンスを編集できるようになります。
2. CR の **spec** セクションに、**brokerProperties** 要素を追加し、キャメルケース形式でプロパティーのリストを追加します。以下に例を示します。

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

```
    - globalMaxSize=500m
    - maxDiskUsage=85
    ...
```

3. CR を保存します。
4. (オプション) 設定のステータスを確認します。
   a. OpenShift コマンドラインインターフェイスの使用:
      i. ブローカーのステータス条件を取得します。

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

   b. OpenShift Web コンソールの使用
      i. ブローカーデプロイメントの CR の status セクションに移動します。
   c. **BrokerPropertiesApplied** ステータス情報の **reason** フィールドの値を確認してください。以下に例を示します。

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

以下の値を使用できます。

Applied

OpenShift Container Platform は、更新されたシークレットを各ブローカー Pod のプロパティーファイルに伝播しました。

AppliedWithError

OpenShift Container Platform は、更新されたシークレットを各ブローカー Pod のプロパティーファイルに伝播しました。ただし、**brokerProperties** 設定でエラーが見つかりました。CR の **status** セクションで、**message** フィールドを確認して無効なプロパティーを特定し、CR で修正します。

OutOfSync

OpenShift Container Platform は、更新されたシークレットを各ブローカー Pod のプロパティーファイルにまだ伝播していません。OpenShift Container Platform が更新されたシークレットを各 Pod に伝播すると、ステータスが更新されます。

注記

ブローカーは、Pod にマウントされているプロパティーファイルの更新などの設定変更を定期的にチェックし、変更を検出した場合は設定をリロードします。ただし、ブローカーの起動時に読み取り専用となるプロパティーの更新 (JVM 設定など) は、ブローカーを再起動するまで再ロードされません。どのプロパティーが再ロードされるかの詳細については、**AMQ Broker の設定** の 設定更新の再ロード を参照してください。

追加情報

CR の **brokerProperties** 要素で設定できるプロパティーのリストについては、**AMQ Broker の設定** の ブローカーのプロパティー を参照してください。

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。

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ました。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。
- デプロイメント内のブローカーに対して AMQ 管理コンソールへのアクセスを有効にしました。AMQ 管理コンソールへのアクセスを有効にする方法の詳細は、「AMQ 管理コンソールへのアクセスの有効化」 を参照してください。

5.1. AMQ 管理コンソールへの接続

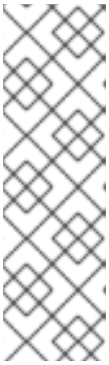
ブローカーデプロイメントのカスタムリソース (CR) インスタンスで AMQ 管理コンソールへのアクセスを有効にすると、オペレーターは各ブローカー Pod に専用のサービスとルートを自動的に作成し、AMQ 管理コンソールへのアクセスを提供します。

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。



注記

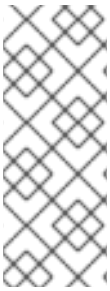
CR の **requireLogin** プロパティーが **true** に設定されている場合に **のみ**、AMQ 管理コンソールにログインするために認証情報が必要です。このプロパティーは、ブローカー **および** AMQ 管理コンソールへのログインにログイン認証情報が必要かどうかを指定します。デフォルトでは、**requireLogin** プロパティーは **false** に設定されます。**requireLogin** が **false** に設定されている場合、ユーザー名とパスワードの入力を求められたら任意のテキストを入力することで、有効なユーザー名とパスワードを入力しなくても AMQ 管理コンソールにログインできます。

4. **requireLogin** プロパティーが **true** に設定されている場合は、ユーザー名とパスワードを入力します。
ブローカーおよび AMQ 管理コンソールへの接続に使用できる、事前設定されたユーザーの認証情報を入力できます。これらの認証情報は、カスタムリソース (CR) インスタンスで設定されている場合、**adminUser** プロパティーと **adminPassword** プロパティーで見つけることができます。これらのプロパティーが CR で設定されていない場合、Operator は認証情報を自動的に生成します。自動的に生成された認証情報を取得するには、「AMQ Management Console のログインクレデンシャルへのアクセス」 を参照してください。

   他のユーザーとしてログインする場合、AMQ 管理コンソールへのログインに必要な権限を得るには、ユーザーは **hawtio.role** システムプロパティーに指定されたセキュリティーロールに属している必要があることに注意してください。**hawtio.role** システムプロパティーのデフォルトのロールは **admin** で、事前設定されたユーザーはこれに属します。

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リスト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てブローカーを表示および管理する方法の詳細については、**AMQ Broker の管理** の 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。

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。

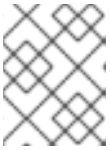
- OpenShift コマンドラインインターフェイス (CLI) と OperatorHub の両方を使用した AMQ Broker Operator バージョン
- Operator ベースのブローカーデプロイメント用のブローカーコンテナーイメージ

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 コマンドラインインターフェイス (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」
- **redeliveryDelayMultiplier** および **redeliveryCollisionAvoidanceFactor** 属性が 7.8.x または 7.9.x デプロイメントのメインブローカー CR で設定されている場合、7.10.x にアップグレードした後、新しい Operator は CR を調整できません。両方の属性のデータ型が 7.10.x で float から string に変更されたため、調整は失敗します。
  この問題を回避するには、**spec.deploymentPlan.addressSettings.addressSetting** 属性から **redeliveryDelayMultiplier** および **redeliveryCollisionAvoidanceFactor** 属性を削除します。次に、**brokerProperties** 属性の下に属性を設定します。以下に例を示します。

```
spec:
  ...
  brokerProperties:
  - "addressSettings.#.redeliveryMultiplier=2.1"
  - "addressSettings.#.redeliveryCollisionAvoidanceFactor=1.2"
```



注記

brokerProperties 属性で、削除した **redeliveryDelayMultiplier** 属性名の代わりに **redeliveryMultiplier** 属性名を使用します。

6.2. CLI を使用した OPERATOR のアップグレード

このセクションの手順では、OpenShift コマンドラインインターフェイス (CLI) を使用して、さまざまなバージョンの Operator を AMQ Broker 7.11 で利用可能な最新バージョンにアップグレードする方法を示します。

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. CLI を使用した Operator のアップグレード

OpenShift コマンドラインインターフェイス (CLI) を使用して、Operator を AMQ Broker 7.11 の最新バージョンにアップグレードできます。

手順

1. Web ブラウザーで、AMQ Broker 7.11.6 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 **7.11.6** 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 7.11.6 Operator Installation and Example Files**の横にある **Download** をクリックします。
amq-broker-operator-7.11.6-ocp-install-examples.zip 圧縮アーカイブのダウンロードが自動的に開始されます。
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-7.11.6-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デプロイメントします。以下に例を示します。

```
$ cd ~/broker/operator
$ unzip amq-broker-operator-operator-7.11.6-ocp-install-examples.zip
```

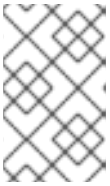
6. 既存の Operator デプロイメントが含まれるプロジェクトの管理者として OpenShift Container Platform にログインします。

```
$ 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** ファイルでは、Operator の名前はデフォルトで **amq-broker-controller-manager** です。以前のデプロイメントの Operator の名前が **amq-broker-controller-manager** ではない場合は、**amq-broker-controller-manager** のすべてのインスタンスを以前の Operator 名に置き換えます。以下に例を示します。

```
spec:
  ...
  selector
    matchLabels
      name: amq-broker-operator
  ...
```

11. **新しい operator.yaml** ファイルでは、Operator のサービスアカウントの名前は **amq-broker-controller-manager** です。以前のバージョンでは、Operator のサービスアカウントの名前は **amq-broker-operator** でした。

    a. 以前のデプロイメントのサービスアカウント名を使用する場合は、**新しい operator.yaml** ファイル内のサービスアカウントの名前を、以前のデプロイメントで使用されていた名前に置き換えます。以下に例を示します。

    ```
    spec:
      ...
      serviceAccountName: amq-broker-operator
      ...
    ```

    b. Operator に新しいサービスアカウント名 **amq-broker-controller-manager** を使用する場合は、プロジェクト内のサービスアカウント、ロール、ロールバインディングを更新します。

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

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

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

12. Operator に含まれる CRD を更新します。

    a. メインブローカー CRD を更新します。

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

    b. アドレス CRD を更新します。

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

c. スケールダウンコントローラー CRD を更新します。

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

d. セキュリティー CRD を更新します。

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

13. AMQ Broker Operator 7.10.0 からのアップグレードのみの場合は、Operator と StatefulSet を削除します。
デフォルトでは、新しい Operator は StatefulSet を削除して、7.10.0 で Operator により StatefulSet セレクターに誤って追加されたカスタムと Operator メータリングラベルを削除します。Operator が StatefulSet を削除すると、既存のブローカー Pod も削除されるため、一時的なブローカーの停止が発生します。停止を回避するには、以下の手順を実行して、ブローカー Pod を削除せずに Operator と StatefulSet を削除します。

a. Operator を削除します。

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

b. **--cascade=orphan** オプションを指定して StatefulSet を削除し、ブローカー Pod を孤立させます。孤立したブローカー Pod は、StatefulSet が削除された後も引き続き実行されます。

```
$ oc delete statefulset <statefulset-name> --cascade=orphan
```

14. AMQ Broker Operator 7.10.0 または 7.10.1 からアップグレードする場合は、メインブローカー CR に **application** または **ActiveMQArtemis** というラベルが **deploymentPlan.labels** 属性で設定されているか確認します。
これらのラベルは、Operator が Pod にラベルを割り当てるために予約されており、7.10.1 以降ではカスタムラベルとして許可されていません。これらのカスタムラベルがメインブローカー CR で設定されていた場合、Operator が割り当てた Pod のラベルはカスタムラベルによって上書きされました。これらのカスタムラベルのいずれかがメインブローカー CR で設定されている場合は、次の手順を実行して Pod で正しいラベルを復元し、CR からラベルを削除します。

a. 7.10.0 からアップグレードする場合は、前の手順で Operator を削除しています。7.10.1 からアップグレードする場合は、Operator を削除します。

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

b. 次のコマンドを実行して、正しい Pod ラベルを復元します。次の例では、ex-aao はデプロイされた StatefulSet の名前です。

```
$ for pod in $(oc get pods | grep -o '^ex-aao[^ ]*'); do oc label --overwrite pods $pod
ActiveMQArtemis=ex-aao application=ex-aao-app; done
```

c. CR の **deploymentPlan.labels** 属性から、**application** ラベルと **ActiveMQArtemis** ラベルを削除します。

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_activemqartemis_cr.yaml** というサンプル CR ファイルを開きます。

iii. CR の **deploymentPlan.labels** 属性で、**application** または **ActiveMQArtemis** というカスタムラベルをすべて削除します。

iv. CR ファイルを保存します。

v. CR インスタンスをデプロイします。

A. 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>
```

B. CR を適用します。

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

d. 以前の Operator を削除した場合は、新しい Operator をデプロイします。

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

15. 更新された Operator 設定を適用します。

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

16. 新しい Operator は、以前のブローカーのデプロイメントを認識して管理できます。CR の **image** または **version** フィールドに値を設定すると、Operator の調整プロセスは、Operator の起動時に、ブローカー Pod を対応するイメージにアップグレードします。詳細は、「ブローカーコンテナーイメージの自動アップグレードの制限」 を参照してください。それ以外の場合、Operator は各ブローカー Pod を最新のコンテナーイメージにアップグレードします。

注記

調整プロセスが開始されない場合は、デプロイをスケーリングすることでプロセスを開始できます。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。

17. 必要に応じて、アップグレードされたブローカーで利用可能な新機能の CR に属性を追加します。

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

このセクションでは、OperatorHub を使用して Operator for AMQ Broker をアップグレードする方法について説明します。

6.3.1. 前提条件

- 最初に OperatorHub を使用して Operator を **インストール** した場合のみ、OperatorHub を使用して Operator をアップグレードします (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** 下に表示されます)。一方、OpenShift コマンドラインインターフェイス (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。既存の CRD を削除すると、すべての CR とブローカーインスタンスも削除されます。

- 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。

- 従うべき手順は、アップグレード元およびアップグレード先の Operator のバージョンによって異なります。現在のバージョンのアップグレード手順に従っていることを確認してください。

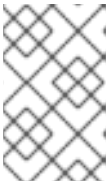
6.3.3. Operator を 7.10.0 より前から 7.11.x にアップグレードする

7.10.0 より前から 7.11.x にアップグレードするには、Operator をアンインストールおよび再インストールする必要があります。

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. 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。
3. 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。
4. 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。
5. 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。
6. 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。
7. 確認ダイアログボックスで、**Uninstall** をクリックします。
8. OperatorHub を使用して、Operator for AMQ Broker 7.11 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。

新しい Operator は、以前のブローカーのデプロイメントを認識して管理できます。CR の **image** または **version** フィールドに値を設定すると、Operator の調整プロセスは、Operator の起動時に、ブローカー Pod を対応するコンテナーイメージにアップグレードします。詳細は、「ブローカーコンテナーイメージの自動アップグレードの制限」 を参照してください。それ以外の場合、Operator は各ブローカー Pod を最新のコンテナーイメージにアップグレードします。



注記

調整プロセスが開始されない場合は、デプロイをスケーリングすることでプロセスを開始できます。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。

6.3.4. Operator を 7.10.0 から 7.11.x にアップグレードする

7.10.0 から 7.11.x にアップグレードするには、Operator をアンインストールおよび再インストールする必要があります。

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. 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。
3. 7.10.0 Operator をアップグレードすると、新しい Operator は StatefulSet を削除して、7.10.0 で Operator により StatefulSet セレクターに誤って追加されたカスタムと Operator メータリングラベルを削除します。Operator が StatefulSet を削除すると、既存のブローカー Pod も削除されるため、一時的なブローカーの停止が発生します。停止を回避したい場合は、次の手順を実行して StatefulSet を削除し、ブローカー Pod を孤立させて実行を継続できるようにします。
   i. 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 CLI にログインします。

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

   ii. 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>
   ```

   iii. **--cascade=orphan** オプションを指定して StatefulSet を削除し、ブローカー Pod を孤立させます。孤立したブローカー Pod は、StatefulSet が削除された後も引き続き実行されます。

```
$ oc delete statefulset <statefulset-name> --cascade=orphan
```

4. メインブローカー CR に **application** または **ActiveMQArtemis** というラベルが **deploymentPlan.labels** 属性で設定されているか確認します。
7.10.0 では、CR でこれらのカスタムラベルを設定できました。これらのラベルは、Operator が Pod にラベルを割り当てるために予約されており、7.10.0 以降ではカスタムラベルとして追加できません。これらのカスタムラベルが 7.10.0 のメインブローカー CR で設定されていた場合、Operator が割り当てた Pod のラベルはカスタムラベルによって上書きされました。CR にこれらのラベルのいずれかがある場合は、次の手順を実行して Pod で正しいラベルを復元し、CR からラベルを削除します。

   a. OpenShift コマンドラインインターフェイス (CLI) で、次のコマンドを実行して正しい Pod ラベルを復元します。次の例では、ex-aao はデプロイされた StatefulSet の名前です。

   ```
   $ for pod in $(oc get pods | grep -o '^ex-aao[^ ]*'); do oc label --overwrite pods $pod
   ActiveMQArtemis=ex-aao application=ex-aao-app; done
   ```

   b. CR の **deploymentPlan.labels** 属性から、**application** ラベルと **ActiveMQArtemis** ラベルを削除します。

      i. OpenShift コマンドラインインターフェイスの使用:

         A. 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>
         ```

         B. デプロイメントの CR を編集します。

         ```
         oc edit ActiveMQArtemis <statefulset name> -n <namespace>
         ```

         C. CR の **deploymentPlan.labels** 要素で、**application** または **ActiveMQArtemis** という名前のカスタムラベルをすべて削除します。

         D. CR を保存します。

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

         A. 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。

         B. 左側のペインで、**Administration** → **Custom Resource Definitions**をクリックします。

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

         D. **Instances** タブをクリックします。

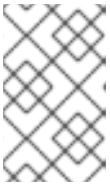
         E. ブローカーデプロイメントのインスタンスをクリックします。

         F. **YAML** タブをクリックします。
         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。

G. CR の **deploymentPlan.labels** 要素で、**application** または **ActiveMQArtemis** という名前のカスタムラベルをすべて削除します。

H. **Save** をクリックします。

5. OperatorHub を使用して、Operator for AMQ Broker 7.11 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。
新しい Operator は、以前のブローカーのデプロイメントを認識して管理できます。CR の **image** または **version** フィールドに値を設定すると、Operator の調整プロセスは、Operator の起動時に、ブローカー Pod を対応するイメージにアップグレードします。詳細は、「ブローカーコンテナーイメージの自動アップグレードの制限」 を参照してください。それ以外の場合、Operator は各ブローカー Pod を最新のコンテナーイメージにアップグレードします。



注記

調整プロセスが開始されない場合は、デプロイをスケーリングすることでプロセスを開始できます。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。

6. 必要に応じて、アップグレードされたブローカーで利用可能な新機能の CR に属性を追加します。

6.3.5. Operator を 7.10.1 から 7.11.x にアップグレードする

7.10.1 から 7.11.x にアップグレードするには、Operator をアンインストールおよび再インストールする必要があります。

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. メインブローカー CR に **application** または **ActiveMQArtemis** というラベルが **deploymentPlan.labels** 属性で設定されているか確認します。
これらのラベルは、Operator が Pod にラベルを割り当てるために予約されており、7.10.1 以降では使用できません。これらのカスタムラベルがメインブローカー CR で設定されていた場合、Operator が割り当てた Pod のラベルはカスタムラベルによって上書きされました。

3. これらのカスタムラベルがメインブローカー CR で設定されていない場合は、OperatorHub を使用して、Operator for AMQ Broker 7.11 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。

4. これらのカスタムラベルのいずれかがメインブローカー CR で設定されている場合、新しい Operator をインストールする前に、以下の手順を実行して既存の Operator をアンインストールし、正しい Pod ラベルを復元して CR からラベルを削除します。

注記

Operator をアンインストールすると、Operator が StatefulSet を削除しなくてもカスタムラベルを削除できます。これにより、既存のブローカー Pod も削除され、ブローカーが一時的に停止します。

a. 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。

i. 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。

ii. 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。

iii. 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。

iv. 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。

v. 確認ダイアログボックスで、**Uninstall** をクリックします。

b. OpenShift コマンドラインインターフェイス (CLI) で、次のコマンドを実行して正しい Pod ラベルを復元します。次の例では、ex-aao はデプロイされた StatefulSet の名前です。

```
$ for pod in $(oc get pods | grep -o '^ex-aao[^ ]*'); do oc label --overwrite pods $pod
ActiveMQArtemis=ex-aao application=ex-aao-app; done
```

c. CR の **deploymentPlan.labels** 属性から、**application** ラベルと **ActiveMQArtemis** ラベルを削除します。

i. OpenShift コマンドラインインターフェイスの使用:

A. 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>
```

B. デプロイメントの CR を編集します。

```
oc edit ActiveMQArtemis <statefulset name> -n <namespace>
```

C. CR の **deploymentPlan.labels** 属性で、**application** または **ActiveMQArtemis** というカスタムラベルをすべて削除します。

D. CR ファイルを保存します。

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

A. 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。

B. 左側のペインで、**Administration** → **Custom Resource Definitions**をクリックします。

C. **ActiveMQArtemis** CRD をクリックします。

D. **Instances** タブをクリックします。

E. ブローカーデプロイメントのインスタンスをクリックします。

F. **YAML** タブをクリックします。
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。

G. CR の **deploymentPlan.labels** 属性で、**application** または **ActiveMQArtemis** というカスタムラベルをすべて削除します。

H. **Save** をクリックします。

5. OperatorHub を使用して、Operator for AMQ Broker 7.11 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。
新しい Operator は、以前のブローカーのデプロイメントを認識して管理できます。CR の **image** または **version** フィールドに値を設定すると、Operator の調整プロセスは、Operator の起動時に、ブローカー Pod を対応するイメージにアップグレードします。詳細は、「ブローカーコンテナーイメージの自動アップグレードの制限」 を参照してください。それ以外の場合、Operator は各ブローカー Pod を最新のコンテナーイメージにアップグレードします。

注記

調整プロセスが開始されない場合は、デプロイをスケーリングすることでプロセスを開始できます。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。

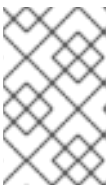
6. 必要に応じて、アップグレードされたブローカーで利用可能な新機能の CR に属性を追加します。

6.3.6. Operator を 7.10.2 以降から 7.11.x にアップグレードする

7.10.2 以降から 7.11.x にアップグレードするには、Operator をアンインストールおよび再インストールする必要があります。

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. 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。
3. 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。
4. 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。
5. 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。
6. 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。
7. 確認ダイアログボックスで、**Uninstall** をクリックします。
8. OperatorHub を使用して、Operator for AMQ Broker 7.11 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。
新しい Operator は、以前のブローカーのデプロイメントを認識して管理できます。CR の **image** または **version** フィールドに値を設定すると、Operator の調整プロセスは、Operator の起動時に、ブローカー Pod を対応するイメージにアップグレードします。詳細は、「ブローカーコンテナーイメージの自動アップグレードの制限」 を参照してください。それ以外の場合、Operator は各ブローカー Pod を最新のコンテナーイメージにアップグレードします。



注記

調整プロセスが開始されない場合は、デプロイをスケーリングすることでプロセスを開始できます。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。

6.4. ブローカーコンテナーイメージの自動アップグレードの制限

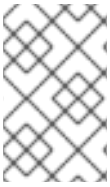
デフォルトでは、Operator は、利用可能な最新のコンテナーイメージを使用するように、デプロイメント内の各ブローカーを自動的にアップグレードします。デプロイメントのカスタムリソース (CR) では、特定のコンテナーイメージのバージョン番号または URL を指定することで、Operator がイメージをアップグレードする機能を制限できます。

注記

ブローカーコンテナーイメージの自動アップグレードを制限する場合は、バージョン番号と、ブローカーと init コンテナーイメージの結合 URL との **いずれか** が CR に含まれていることを確認してください。

6.4.1. バージョン番号を使用したイメージの自動アップグレードの制限

新しいバージョンが利用可能になったときにブローカーが自動的にアップグレードされる、コンテナーイメージのバージョンを制限できます。



注記

バージョン番号に基づいてアップグレードを制限すると、Operator は、デプロイされたバージョンのセキュリティー修正を含む新しいイメージを使用するために、ブローカーの自動アップグレードを継続します。

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イスの使用:
      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. CR を編集します。

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

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
      iv. **AMQ Broker** タブをクリックします。
      v. ActiveMQArtemis インスタンス名をクリックします。
      vi. **YAML** タブをクリックします。

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

注記

CR の **status** セクションの **.status.version.brokerVersion** フィールドには、現在デプロイされている AMQ Broker のバージョンが表示されます。

2. **spec.version** 属性で、Operator がデプロイメント内のブローカーおよび init コンテナーイメージをアップグレードできるバージョンを指定します。指定できる値の例を次に示します。

例

次の例では、Operator がデプロイメント内の現在のコンテナーイメージを 7.11.0 にアップグレードします。

```
spec:
  version: '7.11.0'
   ...
```

次の例では、Operator は、デプロイメント内の現在のコンテナーイメージを利用可能な最新の 7.10.x イメージにアップグレードします。たとえば、デプロイメントで 7.10.1 コンテナーイメージを使用している場合、Operator はイメージを 7.11.6 ではなく 7.10.2 に自動的にアップグレードします。

```
spec:
   version: '7.10'
   ...
```

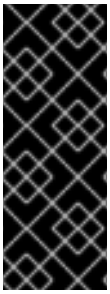
次の例では、Operator がデプロイメント内の現在のコンテナーイメージを最新の 7.x.x イメージにアップグレードします。たとえば、デプロイメントで 7.10.2 イメージを使用している場合、Operator はイメージを 7.11.6 に自動的にアップグレードします。

```
spec:
   version: '7'
   ...
```

注記

コンテナーイメージのマイナーバージョン間で (たとえば、7.10.x から 7.11.x に) アップグレードするには、新しいコンテナーイメージと同じマイナーバージョンを持つ Operator が必要です。たとえば、7.10.2 から 7.11.6 にアップグレードするには、7.11.x Operator をインストールする必要があります。

3. CR を保存します。



重要

CR に、**spec.version** 属性に加えて **spec.deploymentPlan.image** 属性または **spec.deploymentPlan.initImage** 属性が含まれていないことを確認してください。これらの属性はどちらも **spec.version** 属性をオーバーライドします。CR に **spec.version** 属性に加えてこれらの属性のいずれかが含まれている場合、デプロイされているブローカーと init イメージのバージョンが異なる可能性があり、ブローカーが実行できなくなる可能性があります。

CR を保存すると、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既存のデプロイメントで利用可能であることを検証します。アップグレードする AMQ Broker の無効なバージョン (まだ利用できないバージョンなど) を指定した場合、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は実行しません。

ただし、指定されたバージョンへのアップグレードが利用可能 **である** 場合、Operator はデプロイメント内の各ブローカーをアップグレードして、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します。

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_7116** は AMQ Broker 7.11.6 に対応しています。

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。
- デプロイメントのステータスを表示するには、「自動アップグレードに適用される制限の検証」 を参照してください。

6.4.2. イメージ URL を使用したイメージの自動アップグレードの制限

特定のコンテナーイメージを使用するようにデプロイメント内のブローカーをアップグレードする場合は、CR でイメージのレジストリー URL を指定できます。Operator がブローカーを指定したコンテナーイメージにアップグレードした後は、CR 内のイメージ URL を置き換えるまで、それ以上のアップグレードは行われません。たとえば、Operator は、デプロイされたイメージのセキュリティー修正を含む新しいイメージを使用するようにブローカーを自動的にアップグレードしません。

重要

イメージ URL を使用して自動アップグレードを制限する場合は、CR で **spec.deploymentPlan.image** 属性と **spec.deploymentPlan.initImage** 属性の両方の URL を指定して、ブローカーイメージと初期コンテナーイメージが一致することを確認します。1 つのコンテナーイメージの URL のみを指定すると、ブローカーと初期コンテナーイメージが分岐し、ブローカーが実行できなくなる可能性があります。

注記

CR に **spec.deploymentPlan.image** 属性および **spec.deploymentPlan.initImage** 属性に加えて **spec.version** 属性がある場合、Operator は **spec.version** 属性を無視します。

手順

1. Operator が現在のイメージをアップグレードできるブローカーおよび init コンテナーイメージの URL を取得します。
   a. Red Hat カタログで、ブローカーコンテナーコンポーネントページ AMQ Broker for RHEL 8 (Multiarch) を開きます。
   b. **Architecture** ドロップダウンで、アーキテクチャーを選択します。
   c. **Tag** ドロップダウンで、インストールするイメージに対応するタグを選択します。タグはリリース日に基づいて時系列に表示されます。タグは、リリースバージョンと割り当てられたタグで設定されます。
   d. **Get this image** タブを開きます。
   e. **Manifest** フィールドで、**Copy** アイコンをクリックします。
   f. URL をテキストファイルに貼り付けます。
   g. Red Hat カタログで、init コンテナーコンポーネントページ AMQ Broker Init for RHEL 8 (Multiarch) を開きます。
   h. init コンテナーイメージの URL を取得するには、ブローカーコンテナーイメージの URL を取得するために実行した手順を繰り返します。
2. 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イスの使用:
      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. CR を編集します。

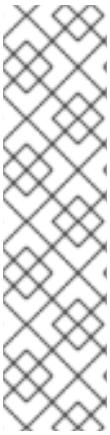
```
oc edit ActiveMQArtemis <CR instance name> -n <namespace>
```

   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. 左側のペインで、**Operators → Installed Operator** をクリックします。
      iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。
      iv. **AMQ Broker** タブをクリックします。
      v. ActiveMQArtemis インスタンス名をクリックします。
      vi. **YAML** タブをクリックします。
      コンソール内で YAML エディターが開き、CR インスタンスを設定できるようになります。

c. テキストファイルに記録したブローカーと init コンテナーイメージの URL をコピーし、CR の **spec.deploymentPlan.image** フィールドと **spec.deploymentPlan.initImage** フィールドに挿入します。以下に例を示します。

```
spec:
  ...
  deploymentPlan:
    image: registry.redhat.io/amq7/amq-broker-rhel8@e8fa2a00e576ecb95561ffbdbf87b1c82d479c8791ab2c6ce741dd0d0b496d15
    initImage: registry.redhat.io/amq7/amq-broker-init-rhel8@f8c1fc7a4298ec691855c5067ed92b4859a92cea7b7ed9af1bba04469b5a315f
  ...
```

3. CR を保存します。
CR を保存すると、Operator は新しいイメージを使用するようにブローカーをアップグレードし、**spec.deploymentPlan.image** 属性および **spec.deploymentPlan.initImage** 属性の値が再度更新されるまで、これらのイメージを使用します。



注記

イメージ URL を設定せずに AMQ Broker をすでにデプロイしている場合は、イメージ URL を遡及的に設定して、Operator が現在デプロイされているイメージをアップグレードしないようにすることができます。デプロイされたイメージのレジストリー URL は、CR の **status** セクションの **.status.version.image** 属性と **.status.version.initImage** 属性にあります。

.status.version.image 属性と **.status.version.initImage** 属性からイメージ URL をコピーし、それぞれ **spec.deploymentPlan.image** 属性と **spec.deploymentPlan.initImage** 属性に挿入した場合、Operator は現在デプロイされているイメージをアップグレードしません。

関連情報

- デプロイメントのステータスを表示するには、「自動アップグレードに適用される制限の検証」 を参照してください。

6.4.3. 自動アップグレードに適用される制限の検証

CR を保存した後、Operator は CR に次のいずれも含まれていないことを検証します。

- **spec.deploymentPlan.initImage** 属性のない **spec.deploymentPlan.image** 属性、またはその逆も同様です。
- **spec.deploymentPlan.image** 属性と **spec.deploymentPlan.initImage** 属性のいずれか、またはその両方を持つ **spec.version** 属性。

これらの設定のいずれも、アップグレード後にブローカーと初期コンテナーイメージのバージョンが異なる可能性があり、ブローカーが起動できなくなる可能性があります。CR にこれらの設定のいずれかが含まれている場合、オペレーターは警告として **Valid** 条件のステータスを **Unknown** 設定します。たとえば、CR に **spec.deploymentPlan.initImage** 属性のない **spec.deploymentPlan.image** 属性がある場合、またはその逆の場合、オペレータは CR の **Valid** 条件に関する次のステータス情報を表示します。

```
status:
  conditions:
```

```
- lastTransitionTime: "2023-05-18T15:17:22Z"
  message: Init image and broker image must both be configured as an interdependent pair
  observedGeneration: 1
  reason: InitImageMustBePairedWithBrokerImage
  status: "Unknown"
  type: Valid
```

ステータス値が **Unknown** である **Valid** 条件でも、Operator による StatefulSet の更新は妨げられません。ただし、Red Hat では、CR で **spec.deploymentPlan.image `and `spec.deploymentPlan.initImage** 属性の組み合わせ、または **spec.version attribute** (両方ではなく) を指定して、**Valid** 条件のステータスを修正することを推奨します。

注記

CR に **spec.version** 属性がある場合、Operator は、バージョン形式が正しいこと、およびバージョンが Operator がサポートする有効な範囲内にあることも検証します。

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。

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 Operator ベースのデプロイメントを作成する方法については、「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/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.11
        ...
```

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/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.11
```

```
...
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 に承認が必要なくなります。**managementRBACEnabled** が **false** に設定されている間は、ブローカー上のすべての管理操作が不正に使用される可能性があるため、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方法については、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 メトリック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 メトリックプラグインを使用すると、ブローカーメトリックを 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設定、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、設定、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. メトリックの概要

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ー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ック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ックを以下に説明します。

ブローカーメトリック

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ーを、**global-max-size** パラメーター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。

アドレスメトリック

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。

キューメトリック

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

指定のキューに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ックについては、下位レベルのメトリック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** メトリックを集約できます。

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) のメトリックも 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 メトリック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ックを収集して 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/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.11
    ...
```

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/v1beta1
kind: ActiveMQArtemis
metadata:
  name: ex-aao
spec:
  deploymentPlan:
    size: 4
    image: registry.redhat.io/amq7/amq-broker-rhel8:7.11
    ...
    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 メトリックへのアクセス

以下の手順では、実行中のブローカー Pod の Prometheus メトリック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. メトリック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。メトリック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。

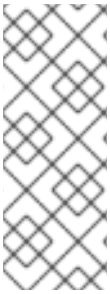
a. **Networking → Routes** をクリックします。

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 メトリック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推奨します。

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=%22amq-
broker%22,component=addresses,address=%22TESTQUEUE%22,subcomponent=queues,ro
uting-type=%22anycast%22,queue=%22TESTQUEUE%22/MaxConsumers

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

第8章 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、Operator とともにデプロイされるカスタム 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:e8fa2a00e576ecb95561ffbdbf87b1c82d479c8791ab2c6ce741dd0d0b496d15 **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**: 1

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認証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 コンテナーイメージ。 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 init コンテナーイメージを選択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。 **カスタム** init コンテナーイメージを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。 **型**: String **例**: registry.redhat.io/amq7/amq-broker-init-rhel8@sha256:f8c1fc7a4298ec691855c5067ed92b4859a92cea7b7ed9af1bba04469b5a315f **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意図的なスケールダウンによりブローカー Pod がシャットダウンした場合、ブローカークラスター内でまだ実行されている別のブローカー 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。

エントリー	サブエントリー	説明と使用法
	resources.limits.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。
	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。

エントリー	サブエントリー	説明と使用法
	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。
	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

エントリー	サブエントリー	説明と使用法
	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
	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
	affinity	Pod のスケジューリングの制約を指定します。アフィニティープロパティーに関する詳細は、OpenShift Container Platform ドキュメントの properties を参照してください。
	tolerations	Pod の容認を指定します。容認のプロパティーに関する詳細は、OpenShift Container Platform ドキュメントの properties 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
	nodeSelector	そのノードでスケジュールされる Pod のノードのラベルと一致するラベルを指定します。
	storageClassName	永続ボリューム要求 (PVC) に使用するストレージクラスの名前を指定します。ストレージクラスは、管理者が使用可能なストレージを記述および分類する方法を提供します。たとえば、ストレージクラスには、特定のサービス品質レベル、バックアップポリシー、またはその他の管理ポリシーが関連付けられている場合があります。 **型**: String **例**:gp3 **デフォルト値**: 指定なし
	startupProbe	startup プローブを設定して、ブローカーコンテナー内の AMQ Broker アプリケーションが起動したかどうかを確認します。startup プローブのプロパティーについては、OpenShift Container Platform ドキュメントのプロパティー を参照してください。
	livenessProbe	実行中のブローカーコンテナーで定期的な可用性チェックを設定して、ブローカーが実行中であることを確認します。liveness プローブのプロパティーについては、OpenShift Container Platform ドキュメントの プロパティー 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
	readinessProbe	実行中のブローカーコンテナーで定期的な可用性チェックを設定して、ブローカーがネットワークトラフィックを受け入れていることを確認します。readiness プローブのプロパティーについては、OpenShift Container Platform ドキュメントのプロパティー を参照してください。
	extraMounts	設定情報を含むシークレットまたは ConfigMap をブローカー Pod 上のファイルとしてマウントします。たとえば、AMQ Broker のカスタマイズされたログ設定を含むシークレットをマウントできます。 **型**: object **例**: 「ブローカーのログの設定」 を参照 **デフォルト値**: 指定なし
	labels	ブローカー Pod にラベルを割り当てます。 **型**: String **例**: 場所: 実稼働 **デフォルト値**: 指定なし
	podSecurity.serviceAccountName	ブローカー Pod のサービスアカウント名を指定します。 **型**: String **例**: amq-broker-controller-manager **デフォルト値**: デフォルト

エントリー	サブエントリー	説明と使用法
	podSecurityContext	次の Pod レベルのセキュリティー属性と一般的なコンテナー設定を指定します。 * fsGroup * fsGroupChangePolicy * runAsGroup * runAsUser * runAsNonRoot * seLinuxOptions * seccompProfile * supplementalGroups * sysctls * windowsOptions **podSecurityContext** プロパティーについては、OpenShift Container Platform ドキュメントのプロパティー を参照してください。
console		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。このプロパティーは、**sslEnabled** プロパティーが **true** に設定されている場合のみ適用されます。 **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承認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。

エントリー	サブエントリー	説明と使用法
	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
	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 に必要な認証情報を探します。 **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 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。どの暗号スイートを指定すればよいかわからない場合は、まずクライアントをデバッグモードで実行してブローカーとクライアント間の接続を確立し、ブローカーとクライアントの両方に共通する暗号スイートを確認します。次に、ブローカーで **enabledCipherSuites** を設定します。 利用可能な暗号スイートは、ブローカーとクライアントによって使用される TLS プロトコルバージョンによって異なります。ブローカーをアップグレードした後にデフォルトの TLS プロトコルバージョンが変更された場合は、ブローカーとクライアントが共通の暗号スイートを使用できるようにするために、以前の TLS プロトコルバージョンを選択する必要がある場合があります。詳細は、**enabledProtocols** を参照してください。 **型**: String **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 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 TLS プロトコルのバージョンを指定しない場合、ブローカーは JVM のデフォルトバージョンを使用します。ブローカーが JVM のデフォルトの TLS プロトコルバージョンを使用しており、ブローカーのアップグレード後にそのバージョンが変更された場合、ブローカーとクライアントが使用する TLS プロトコルバージョンに互換性がない可能性があります。新しい TLS プロトコルバージョンを使用することを推奨しますが、新しい TLS プロトコルバージョンをサポートしていないクライアントと相互運用するために、**enabledProtocols** で以前のバージョンを指定することもできます。
	keyStoreProvider	ブローカーが使用するキーストアのプロバイダーの名前。 **型**: String **例**:SunJCE **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
	trustStoreProvider	ブローカーが使用するトラストストアのプロバイダーの名前。 **型**: String **例**:SunJCE **デフォルト値**: 指定なし
	trustStoreType	ブローカーが使用するトラストストアのタイプ。 **型**: String **例**:JCEKS **デフォルト値**:JKS
	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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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接頭辞。 **型**: String **例**: jms.queue **デフォルト値**: 指定なし
	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接頭辞。 **型**: String **例**: /topic / **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
	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)
	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)

エントリー	サブエントリー	説明と使用法
	BindToAllInterfaces	true に設定すると、Pod の内部 IP アドレスではなく 0.0.0.0 IP アドレスを使用してブローカーアクセプターを設定します。ブローカーアクセプターが 0.0.0.0 IP アドレスを持っている場合、Pod 用に設定されたすべてのインターフェイスにバインドし、クライアントは OpenShift Container Platform ポート転送を使用してトラフィックをブローカーに転送できます。通常、この設定を使用してサービスをデバッグします。ポート転送の詳細は、OpenShift Container Platform ドキュメントのポート転送を使用してコンテナー内のアプリケーションにアクセスする を参照してください。 **注記** ポート転送が正しく使用されていない場合、環境にセキュリティーリスクが生じる可能性があります。可能な場合、Red Hat は、実稼働環境ではポート転送を使用しないことを推奨します。 **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
connectors.connector		単一のコネクター設定インスタンス。
	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
	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 に必要な認証情報を探します。 **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 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**ことを推奨します。 **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
	keyStoreProvider	ブローカーが使用するキーストアのプロバイダーの名前。 **型**: String **例**:SunJCE **デフォルト値**: 指定なし
	trustStoreProvider	ブローカーが使用するトラストストアのプロバイダーの名前。 **型**: String **例**:SunJCE **デフォルト値**: 指定なし
	trustStoreType	ブローカーが使用するトラストストアのタイプ。 **型**: String **例**:JCEKS **デフォルト値**:JKS
	enabledProtocols	TLS 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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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**

エントリー	サブエントリー	説明と使用法
		CR に指定された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 **または** 特定のアドレスあるいはアドレスセットに一意に一致するデフォルト設定のいずれかに指定されるアドレス設定の場合は、これらを最終的に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 **デフォルト値**: merge_all
addressSettings.addressSetting		一致するアドレスまたはアドレスの **セット** の設定。

エントリー	サブエントリー	説明と使用法
	addressFullPolicy	**maxSizeBytes** で設定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 dead letter キューの名前に適用される接頭辞。 **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 dead letter キューに適用される接尾辞。 **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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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)
	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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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
	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ック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ック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

エントリー	サブエントリー	説明と使用法
	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)
	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接頭辞。 **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接尾辞。 **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、ブローカーで設定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
	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
	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

エントリー	サブエントリー	説明と使用法
	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
	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)
env		ブローカーの環境変数を設定します。
	<property name>=<value>	ブローカー用に設定するプロパティー名と値のリスト。 **型**: array

エントリー	サブエントリー	説明と使用法
`name:<**variable name**>` value:<**variable value**>	ブローカーに設定する環境変数の名前と値のリスト。 **型**: array **例**: 名前: TZ 値: Europe/Vienna **デフォルト値**: 該当なし	**brokerProperties**
	ブローカーのカスタムリソース定義 (CRD) で公開されていないブローカープロパティーを設定します。それ以外の場合は、カスタムリソース (CR) で設定できません。	
<property name>=<value>	ブローカー用に設定するプロパティー名と値のリスト。現在、1 つのプロパティー **globalMaxSize** は、**brokerProperties** セクションで設定できます。**globalMaxSize** プロパティーを設定すると、ブローカーに割り当てられたデフォルトのメモリー量がオーバーライドされます。デフォルトでは、ブローカーには、ブローカーの Java 仮想マシンで使用可能な最大メモリーの半分が割り当てられます。 **globalMaxSize** プロパティーのデフォルトの単位は bytes です。デフォルトの単位を変更するには、値に m (MB の場合) または g (GB の場合) の接尾辞を追加します。 **型**: String **例**: globalMaxSize=512m **デフォルト値**: 該当なし	**version**

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。

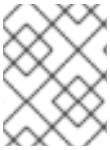
エントリー	説明と使用法
addressName*	ブローカーで作成されるアドレス名。 **型**: String **例**: address0 **デフォルト値**: 指定なし
queueName	ブローカーで作成されるキュー名。**queueName** が指定されていない場合、CR はアドレスのみを作成します。 **型**: String **例**: queue0 **デフォルト値**: 指定なし
removeFromBrokerOnDelete*	デプロイメントのアドレス CR インスタンスを削除するときに、Operator がデプロイメント内のすべてのブローカーの既存のアドレスを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設定を定義できます。これには、次のものが含まれます。

- ユーザーとロール

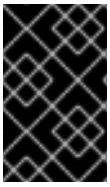
- **Properties Login Module**、**guest Login Module**、**keycloak Login Module** などのログインモジュール
- ロールベースのアクセス制御
- コンソールアクセス制御



注記

オプションの多くでは、ブローカーの保護 で説明されているブローカーのセキュリティー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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
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
	configuration	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 Open IDConnect 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。

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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。
	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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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
	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 クエリー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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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	デバッグ
	brokerDomain.loginModules.reload	Reload
	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設定。
	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	デバッグ **型**: Boolean **例**: false

エントリー	サブエントリー	説明と使用法
	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。一致パターン構文の詳細については、AMQ Broker のワイルドカード構文 を参照してください。
	broker.permissions.operationType	セキュリティー設定の操作タイプ。権限の設定 で説明されています。 **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** を設定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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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。
	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。設定ファイル内のパスワードの暗号化 で説明されている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

エントリー	サブエントリー	説明と使用法
	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
	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

エントリー	サブエントリー	説明と使用法
	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. JAAS ログインモジュール設定の例

次の例は、プロパティーログインモジュールと LDAP ログインモジュールの両方が設定された JAAS ログインモジュール設定を示しています。プロパティーログインモジュールは、Operator がブローカーで認証するために使用する認証情報を含むデフォルトのログインモジュールを参照します。

```
activemq {
  org.apache.activemq.artemis.spi.core.security.jaas.LDAPLoginModule required
    debug=true
   initialContextFactory=com.sun.jndi.ldap.LdapCtxFactory
   connectionURL="LDAP://localhost:389"
   connectionUsername="CN=Administrator,CN=Users,OU=System,DC=example,DC=com"
   connectionPassword=redhat.123
   connectionProtocol=s
   connectionTimeout="5000"
   authentication=simple
    userBase="dc=example,dc=com"
   userSearchMatching="(CN={0})"
   userSearchSubtree=true
   readTimeout="5000"
    roleBase="dc=example,dc=com"
    roleName=cn
    roleSearchMatching="(member={0})"
    roleSearchSubtree=true;

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

次の例は、別々のレルムに 2 つのプロパティーログインモジュールがある JAAS ログインモジュール設定を示しています。

- デフォルトのプロパティーログインモジュールは、**console** という名前のレルム内にあり、Operator と AMQ 管理コンソールがブローカーで認証するために使用するプロパティーファイルを持っています。

- **activemq** レルムのログインモジュールには新しいプロパティーファイルがあり、たとえば、メッセージングのためにユーザーを認証するための認証情報を含めることができます。

たとえば、Operator がブローカーで認証するために使用するログインモジュールを含むレルムに特定のセキュリティー制御を適用するために、別のレルムを作成できます。

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

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

注記

デフォルトでは、AMQ 管理コンソールは、認証に **activemq** レルムのデフォルトのプロパティーログインモジュールを使用します。例のように、デフォルトのプロパティーログインモジュールが別のレルムで設定されている場合は、ブローカー CR で環境変数を設定して、そのレルムを使用するように AMQ 管理コンソールを設定する必要があります。以下に例を示します。

```
spec:
  ...
  env:
  - name: JAVA_ARGS_APPEND
    value: --Hawtio.realm=console
  ...
```

CR での環境変数の設定の詳細については、「ブローカーコンテナーの環境変数の設定」 を参照してください。

8.3. 例: RED HAT SINGLE SIGN-ON を使用するように AMQ BROKER を設定する

この例では、JAAS ログインモジュールを使用して認証と認可に Red Hat Single Sign-On を使用するように AMQ Broker を設定する方法を示します。

前提条件

- LDAP ディレクトリーと統合された Red Hat Single Sign-On インスタンス。
  - LDAP ディレクトリーには、AMQ Broker のユーザーとロール情報が設定されます。
  - Red Hat Single Sign-On は、LDAP サーバーからユーザーをフェデレーションするように設定されています。

- Red Hat Single Sign-On は、role-ldap-mapper を使用してロール情報を LDAP から Red Hat Single Sign-On にマッピングするように設定されています。

- 以下を備えた Red Hat Single Sign-On レルム:
  - oAuth プロトコルを使用してトークンを取得できる AMQ 管理コンソールなどのアプリケーションに対して次の設定を使用して設定されたクライアント:
    認証フロー: 標準フロー

    有効なリダイレクト URI: AMQ 管理コンソールの OpenShift Container Platform ルート。
    例: http://artemis-wconsj-0-svc-rte-kc-ldap-tests-0eae49.apps.redhat-412t.broker.app-services-dev.net/console/*

  - oAuth プロトコルを使用してトークンを取得できないメッセージングクライアントアプリケーションがある場合は、次の設定で設定された別のクライアント:
    認証フロー: ダイレクトアクセス許可

    有効なリダレクト URIs: *

注記

Red Hat Single Sign-On の各レルムには、**Broker** という名前のクライアントが含まれます。このクライアントは AMQ Broker に関連しません。

手順

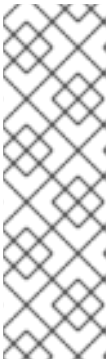
1. **login.config** という名前のテキストファイルを作成し、AMQ Broker を Red Hat Single Sign-On に接続するための JAAS ログインモジュール設定を追加します。以下に例を示します。

```
console {
   // ensure the operator can connect to the broker by referencing the existing properties
config
   org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule sufficient
      org.apache.activemq.jaas.properties.user="artemis-users.properties"
      org.apache.activemq.jaas.properties.role="artemis-roles.properties"
      baseDir="/home/jboss/amq-broker/etc";

  org.keycloak.adapters.jaas.BearerTokenLoginModule sufficient
      keycloak-config-file="/amq/extra/secrets/sso-jaas-config/_keycloak-bearer-token.json"
      role-principal-class=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal;
};
activemq {
   org.keycloak.adapters.jaas.BearerTokenLoginModule sufficient
      keycloak-config-file="/amq/extra/secrets/sso-jaas-config/_keycloak-bearer-token.json"
      role-principal-class=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal;

   org.keycloak.adapters.jaas.DirectAccessGrantsLoginModule sufficient
      keycloak-config-file="/amq/extra/secrets/sso-jaas-config/_keycloak-direct-access.json"
      role-principal-class=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal;

   org.apache.activemq.artemis.spi.core.security.jaas.PrincipalConversionLoginModule
required
      principalClassList=org.keycloak.KeycloakPrincipal;
};
```

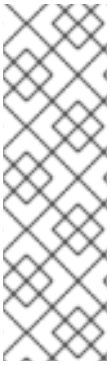


注記

- **.json** 設定ファイルへのパスは、**/amq/extra/secrets/name-jaas-config** の形式である必要があります。**name** には文字列値を指定します。この手順の後半で作成するシークレットに名前を付けるには、同じ文字列値と **-jaas-config** 接尾辞を使用する必要があります。
- この **login.config** ファイルの例では、AMQ 管理コンソールユーザーを認証するために **console** という名前のレルムが使用され、メッセージングクライアントを認証するために **activemq** という名前のレルムが使用されます。

この **login.config** ファイルの例では、次のログインモジュールが設定されています。

ログインモジュール	説明と使用法
org.apache.activemq.artemis.spi.core.security.jaas.PropertiesLoginModule	これはデフォルトのログインモジュールであり、**artemis-users.properties** ファイルが含まれています。このファイルには、Operator がブローカーで認証するために必要なデフォルトユーザーが含まれています。
org.keycloak.adapters.jaas.BearerTokenLoginModule	このログインモジュールは、oAuth プロトコルを使用してトークンを取得できるアプリケーション (AMQ 管理コンソールなど) 用です。ユーザーがブラウザーウィンドウで AMQ 管理コンソールを開くと、Red Hat Single Sign-On コンソールにリダイレクトされ、ログインしてベアラートークンを取得します。
org.keycloak.adapters.jaas.DirectAccessGrantsLoginModule	このログインモジュールは、oAuth プロトコルを使用できないメッセージングクライアントなどの非 HTTP アプリケーションに必要です。このログインモジュールを使用すると、ブローカーはまず Red Hat Single Sign-On で設定されたシークレットを使用してクライアントを認証し、次にクライアントに代わってトークンを取得します。
org.apache.activemq.artemis.spi.core.security.jaas.PrincipalConversionLoginModule	このログインモジュールは、受信した Keycloak プリンシパルを AMQ Broker で使用できる JAAS プリンシパルに変換するために必要です。



注記

login.config ファイルの例では、各 **.json** プロパティーファイル名にはアンダースコアの接頭辞が付いています。Operator は、**JaasPropertiesApplied** 条件のステータスを報告するときに、先頭にアンダースコアが付いたファイルを無視します。ファイル名にアンダースコア接頭辞が付いていないと、ブローカーはサードパーティーのログインモジュールによって使用されるプロパティーファイルを認識しないため、**JaasPropertiesApplied** 条件のステータスには永続的に **OutofSync** が表示されます。ステータスレポートの詳細は、「セキュリティーカスタムリソース (CR) を使用したデフォルトの JAAS ログインモジュールの設定」 を参照してください。

1. ログインモジュールで参照される各 **.json** プロパティーファイルのテキストファイルを作成し、AMQ ブローカーを Red Hat Single Sign-On に接続するために必要な詳細を設定します。以下に例を示します。

_keycloak-bearer-token.json

```
{
    "realm": "amq-broker-ldap",
    "resource": "amq-console",
    "auth-server-url": "https://keycloak-svc-rte-kc-ldap-tests-0eae49.apps.412t.broker.app-services-dev.net",
    "principal-attribute": "preferred_username",
    "use-resource-role-mappings": false,
    "ssl-required": "external",
    "confidential-port": 0
}
```

_keycloak-direct-access.json

```
{
    "realm": "amq-broker-ldap",
    "resource": "amq-broker",
    "auth-server-url": "https://keycloak-svc-rte-kc-ldap-tests-0eae49.apps.412t.broker.app-services-dev.net",
    "principal-attribute": "preferred_username",
    "use-resource-role-mappings": false,
    "ssl-required": "external",
    "credentials": {
        "secret": "Lfk6g1ZKlGzNT6eRkz0d1scM4M29Ohmn"
    }
}
```

realm

Red Hat Single Sign-On で AMQ ブローカーのアプリケーションとサービスを認証するように設定されたレルム。

resource

Red Hat Single Sign-On で設定されているクライアントのクライアント ID。

auth-server-url

Red Hat Single Sign-On サーバーのベース URL。

principal-attribute

UserPrincipal 名を設定するためのトークン属性。

use-resource-role-mappings

true に設定すると、Red Hat Single Sign-On はトークン内でユーザーのアプリケーションレベルのロールマッピングを調べます。false の場合、レルムレベルでユーザーロールマッピングを調べます。デフォルト値は false です。

ssl-required

Red Hat Single Sign-On サーバーとの間のすべての通信が HTTPS を介して行われるようにします。デフォルト値は **external** です。これは、外部リクエストにはデフォルトで HTTPS が必要であることを意味します。

credentials

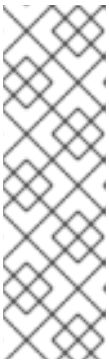
Red Hat Single Sign-On に設定されたシークレット。ブローカーが Red Hat Single Sign-On にログインし、クライアントに代わってトークンを取得するために使用します。

2. **_keycloak-js-client.json** という名前のテキストファイルを作成し、AMQ 管理コンソールに必要な設定を追加して、ユーザーを Red Hat Single Sign-On 管理コンソールの URL にリダイレクトし、そこで認証情報を入力します。以下に例を示します。

```
{
  "realm": "amq-broker-ldap",
  "clientId": "amq-console",
  "url": "https://keycloak-svc-rte-kc-ldap-tests-0eae49.apps.412t.broker.app-services-dev.net"
}
```

3. **oc create secret** コマンドを使用して、ログインモジュール設定で参照されるファイルを含む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 sso-jaas-config --from-file=login.config --from-file=artemis-users.properties --from-file=artemis-roles.properties --from-file=_keycloak-bearer-token.json --from-file=_keycloak-direct-access.json --from-file=_keycloak-js-client.json
```



注記

- シークレットにログインモジュール設定が含まれていることを Operator が認識し、更新を各ブローカー Pod に伝播できるように、シークレット名には接尾辞 **-jaas-config** が必要です。
- シークレット名は、**login.config** ファイルで指定した **.json** 設定ファイルへのパスの末尾にあるディレクトリー名と一致する必要があります。たとえば、設定ファイルへのパスが **/amq/extra/secrets/sso-jaas-config** の場合、シークレット名として **sso-jaas-config** を指定する必要があります。

シークレットの作成方法の詳細については、Kubernetes ドキュメントの シークレット を参照してください。

4. 作成したシークレットを、ブローカーデプロイメントの ActiveMQArtemis カスタムリソース (CR) インスタンスに追加します。

   a. OpenShift コマンドラインインターフェイスの使用:

      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。

      ii. デプロイメントの CR を編集します。

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

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. 左側のペインで、**Operators → Installed Operator** をクリックします。

iii. **Red Hat Integration - AMQ Broker for RHEL 8 (Multiarch)** Operator をクリックします。

iv. **AMQ Broker** タブをクリックします。

v. ActiveMQArtemis インスタンス名をクリックします。

vi. **YAML** タブをクリックします。
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。

5. **extraMounts** 属性と **Secrets** 属性を作成し、シークレットの名前を追加します。次の例では、**custom-jaas-config** という名前のシークレットを CR に追加します。

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

6. **ActiveMQArtemis** CR で、認証に Red Hat Single Sign-On を使用するために AMQ 管理コンソールで必要な hawtio 設定を含む環境変数を作成します。環境変数の内容は、ブローカーをホストする JVM の起動時に Java アプリケーションランチャーに引数として渡されます。以下に例を示します。

```
env:
- name: JAVA_ARGS_APPEND
  value: -
Dhawtio.rolePrincipalClasses=org.apache.activemq.artemis.spi.core.security.jaas.RolePrincipal

      -Dhawtio.keycloakEnabled=true -Dhawtio.keycloakClientConfig=/amq/extra/secrets/sso-
jaas-config/_keycloak-js-client.json
      -Dhawtio.authenticationEnabled=true -Dhawtio.realm=console
```

hawtio 設定の詳細は、hawtio のドキュメント を参照してください。

7. **ActiveMQArtemis** CR の **spec** セクションで、**brokerProperties** 属性を追加し、LDAP ディレクトリーで設定されたロールの権限を追加します。単一のアドレスにロール権限を付与できます。または、**#** 記号を使用してワイルドカード一致を指定し、すべてのアドレスにロールの権限を付与することもできます。以下に例を示します。

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

```
- securityRoles.#.producers.send=true
- securityRoles.#.consumers.consume=true
...
```

8. CR を保存します。
   Operator は、**/amq/extra/secrets/secret name** ディレクトリー内のシークレットのファイルを各 Pod にマウントし、デフォルトの **login.config** ファイルの代わりに、マウントされた **login.config** ファイルを読み取るようにブローカー JVM を設定します。このファイルには、SSO 設定が含まれます。

8.4. 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: 2024-04-20