Red Hat AMQ 7.7

OpenShift での AMQ Online の評価

AMQ Online 1.5 で使用する場合

Last Updated: 2023-02-03

Red Hat AMQ 7.7 OpenShift での AMQ Online の評価

AMQ Online 1.5 で使用する場合

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

法律上の通知

Copyright © 2023 | You need to change the HOLDER entity in the en-US/Evaluating_AMQ_Online_on_OpenShift.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

このガイドでは、AMQ Online をインストールおよび管理して、実稼働環境での使用の可能性を評価する方法について説明します。

目次

第1章 はじめに 5
1.1. AMQ ONLINE の概要 5
1.2. サポートされる機能 6
1.3. サポートされる構成 7
1.4. 本書の表記慣例 7
1.4.1. 変数テキスト 8

第2章 スタートガイド 9
2.1. YAML バンドルを使用した AMQ ONLINE のインストール 9
2.1.1. AMQ Online のダウンロード 9
2.1.2. YAML バンドルを使用した AMQ Online のインストール 9
2.2. OPERATOR LIFECYCLE MANAGER を使用した AMQ ONLINE のインストールと設定 10
2.2.1. OpenShift Container Platform コンソールを使用した OperatorHub から AMQ Online へのインストール 11
2.2.2. OpenShift Container Platform コンソールを使用した AMQ Online の設定 12
2.2.2.1. OpenShift Container Platform コンソールを使用したインフラストラクチャー設定のカスタムリソースの作成 12
2.2.2.2. OpenShift Container Platform コンソールを使用した認証サービスのカスタムリソースの作成 13
2.2.2.3. OpenShift Container Platform コンソールを使用したアドレス空間計画のカスタムリソースの作成 13
2.2.2.4. OpenShift Container Platform コンソールを使用したアドレス計画のカスタムリソースの作成 14
2.3. コマンド行を使用したアドレス空間の作成 15
2.4. コマンドラインを使用したアドレスの作成 16
2.5. コマンドラインでのユーザーの作成 16
2.6. メッセージの送受信 17

第3章 AMQ ONLINE のモノのインターネット (IOT) 19
3.1. IOT の使用開始 19
3.1.1. IoT 接続の概念 19
3.1.2. YAML バンドルを使用した AMQ Online のインストール 20
3.1.3. IoT サービスのインストール 21
3.1.4. IoT プロジェクトの作成 22
3.1.5. IoT デバイスの作成 22
3.1.5.1. 新しいデバイスの登録 23
3.1.5.2. デバイスのユーザー名とパスワード認証情報の設定 23
3.1.6. Eclipse Hono コマンドラインクライアントのインストール 24
3.1.7. telemetry コンシューマーの開始 24
3.1.8. HTTP を使用した telemetry の送信 24
3.2. サービス管理者向け IOT 24
3.2.1. IoT モニタリング 25
3.2.2. IoT ロギング 25
3.2.2.1. 設定オプション 25
3.2.2.2. ログレベル 25
3.2.3. IoT トレース 26
3.2.4. デバイスレジストリー 26
3.2.5. IoT サービスのインストール 26
3.2.6. JDBC 外部デバイスレジストリーのデプロイ 28
3.2.6.1. データストレージモデルの選択 28
3.2.6.2. データベースインスタンスの作成 28
3.2.6.3. SQL スキーマのデータベースインスタンスへのデプロイ 28
3.2.6.4. IoT インフラストラクチャーの設定 29
3.2.7. ロギングの設定 29

3.2.7.1. グローバルログレベルの設定 29
3.2.7.2. アプリケーション固有のログレベルの設定 30
3.2.7.3. カスタム logback 固有設定の適用 31
3.2.7.3.1. IoTConfig リソースの使用 31
3.2.7.3.2. サービスの ConfigMap リソースの使用 32
3.2.8. トレースの設定 33
3.2.9. IoT サービスの設定例 34
3.2.9.1. 最小限の IoT 設定の例 34
3.2.9.2. IoT プロトコルアダプターの例のチューニング 35
3.2.9.3. 外部 PostgreSQL を使用した JDBC の設定 36
3.2.10. IoT 固有の指標 36
3.2.10.1. 一般的なタグと指標 36
3.2.10.2. プロトコルアダプター 37
3.2.11. トラブルシューティングガイド 39
3.2.11.1. IoTProject が終了時に動かなくなる問題を修正 39
3.3. プロジェクトオーナー向けの IOT 40
3.3.1. IoT プロジェクトの設定例 40
3.3.1.1. マネージドメッセージングインフラストラクチャーの使用 40
3.3.1.2. 外部メッセージングインフラストラクチャーの使用 41
3.4. デバイスマネージャー向け IOT 42
3.4.1. 認証トークンの取得 42
3.4.1.1. ユーザーの認証トークンの取得 42
3.4.1.2. サービスアカウントの認証トークンの取得 42
3.4.2. IoT デバイスの作成 43
3.4.2.1. 新しいデバイスの登録 44
3.4.2.2. デバイスのユーザー名とパスワード認証情報の設定 44
3.5. ソリューション開発者向け IOT 45
3.5.1. Eclipse Hono コマンドラインクライアントのインストール 45
3.5.2. telemetry コンシューマーの開始 45
3.5.3. イベントコンシューマーの開始 45
3.5.4. コマンド送信者の開始 46
3.6. デバイス開発者向け IOT 46
3.6.1. HTTP デバイス 46
3.6.1.1. HTTP を使用した telemetry の送信 46
3.6.1.2. HTTP を使用したイベントの送信 47
3.6.1.3. HTTP プロトコルを使用したコマンドの受信 47
3.6.2. MQTT デバイス 48
3.6.2.1. MQTT を使用したテレメトリーの送信 48
3.6.2.2. MQTT を使用したイベントの送信 48
3.6.2.3. MQTT プロトコルを使用したコマンドの受信 48
3.6.3. Sigfox デバイスの設定 49
3.6.3.1. Sigfox バックエンドのゲートウェイデバイスとしての登録 49
3.6.3.2. Sigfox デバイスの登録 50
3.6.3.3. Sigfox 接続情報の準備 50
3.6.3.4. Sigfox バックエンドでの新規コールバックの作成 51
3.6.3.5. Sigfox でのコマンドアンドコントロールの有効化 51

第4章 AMQ ONLINE のインストール **53**
4.1. YAML バンドルを使用した AMQ ONLINE のアンインストール 53
4.2. OPENSHIFT CONTAINER PLATFORM 4.X コンソールを使用した AMQ ONLINE OPERATOR のアンインストール 53
4.2.1. Operator Lifecycle Manager を使用した AMQ Online のアンインストール後の残りのリソース削除 54

付録A サブスクリプションの使用 **55**
アカウントへのアクセス 55
サブスクリプションのアクティベート 55
zip および tar ファイルのダウンロード 55
パッケージ用のシステムの登録 55

第1章 はじめに

1.1. AMQ ONLINE の概要

Red Hat AMQ Online は、マネージドサービスとしてメッセージングを配信するための OpenShift ベースのメカニズムです。Red Hat AMQ Online を使用すると、管理者は、クラウドまたはオンプレミスのいずれかで、クラウドネイティブのマルチテナントメッセージングサービスを設定できます。開発者は、Red Hat AMQ コンソールを使用してメッセージングをプロビジョニングできます。複数の開発チームがコンソールからブローカーとキューをプロビジョニングできます。各チームがソフトウェアをインストール、設定、デプロイメント、保守、またはパッチを適用する必要はありません。

AMQ Online は、ユースケースに応じてさまざまな種類のメッセージングをプロビジョニングできます。ユーザーは、Address Space を作成することでメッセージングリソースをリクエストできます。AMQ Online は現在、標準とブローカーの 2 つのアドレス空間タイプをサポートしており、それぞれセマンティクスが異なります。次の図は、各アドレス空間タイプのアーキテクチャーの概要を示しています。

図1.1 標準アドレス空間

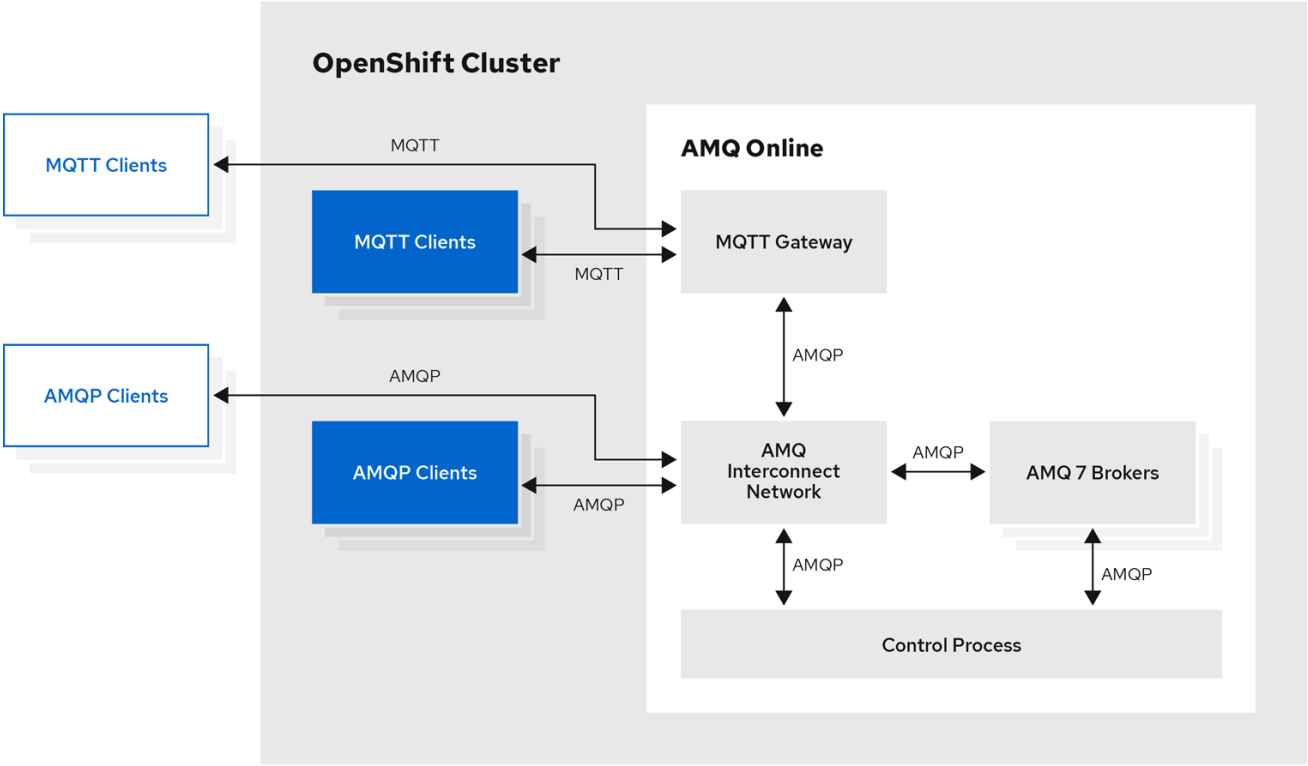
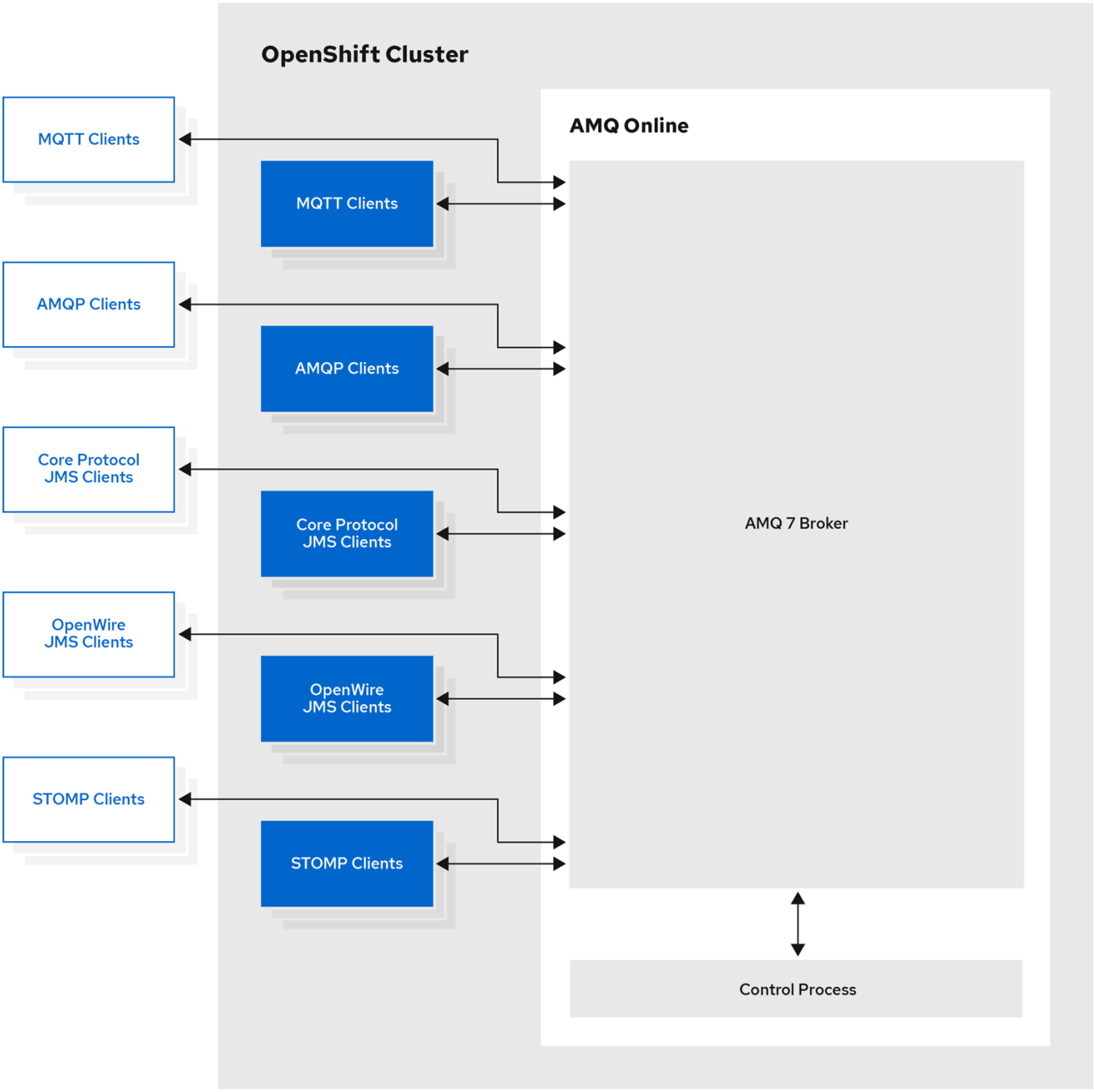


図1.2 ブローカーアドレス空間



1.2. サポートされる機能

次の表は、AMQ Online 1.5 でサポートされている機能を示しています。

表1.1 サポート対象機能の参照表

機能		ブローカーアドレス空間	標準アドレス空間
アドレスの種類	Queue	はい	はい
	トピック	はい	はい

機能		ブローカーアドレス空間	標準アドレス空間
	マルチキャスト	いいえ	はい
	anycast	いいえ	はい
	Subscription	いいえ	はい
メッセージングプロトコル	AMQP	はい	はい
	MQTT	はい	テクノロジープレビューとしてのみ提供
	CORE	はい	いいえ
	OpenWire	はい	いいえ
	STOMP	はい	いいえ
トランスポート	TCP	はい	はい
	WebSocket	はい	はい
永続サブスクリプション	JMS 永続サブスクリプション	はい	いいえ
	名前付き永続サブスクリプション	いいえ	はい
JMS	トランザクションサポート	はい	いいえ
	キューのセレクター	はい	いいえ
	メッセージ順序の保証 (優先順位付けを含む)	はい	いいえ
スケーラビリティー	スケーラブルな分散キューとトピック	いいえ	はい

1.3. サポートされる構成

AMQ Online でサポートされる設定の詳細は、Red Hat AMQ 7 でのサポート対象設定 を参照してください。

1.4. 本書の表記慣例

1.4.1. 変数テキスト

本書では、変数を含むコードブロックが紹介されていますが、これは、お客様のシステム環境に固有の値に置き換える必要があります。このドキュメントでは、そのようなテキストはイタリックモノスペースのスタイルで指定しています。

たとえば、次のコードブロックで、**my-namespace** はインストールで使用されている namespace に置き換えます。

```
sed -i 's/amq-online-infra/my-namespace/' install/bundles/enmasse-with-standard-authservice/*.yaml
```

第2章 スタートガイド

このガイドでは、メッセージを送受信するクライアントを使用して OpenShift で AMQ Online を設定し、実稼働環境での使用の可能性を評価するプロセスについて説明します。

前提条件

- Debezium をインストールするには、OpenShift Container Platform コマンドラインインターフェイス (CLI) が必要です。
  - OpenShift 3.x の CLI のインストール方法については、OpenShift Container Platform 3.11 のドキュメント を参照してください。
  - OpenShift 4.1 の CLI のインストール方法については、OpenShift Container Platform 4.1 のドキュメント を参照してください。
- OpenShift クラスターが必要です。
- 必要なクラスターロールおよび API サービスを設定するための **cluster-admin** 権限を持つ OpenShift クラスターのユーザーが必要です。

2.1. YAML バンドルを使用した AMQ ONLINE のインストール

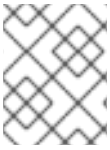
ダウンロードとインストールの手順が完了したら、次のことを行う必要があります。

- アドレス空間の作成
- アドレスの作成
- メッセージングユーザーの作成

2.1.1. AMQ Online のダウンロード

手順

- AMQ Online ダウンロードサイト から **amq-online-install.zip** ファイルをダウンロードして展開します。



注記

AMQ Online のコンテナーイメージは Red Hat Container Catalog で使用できますが、この代わりに提供される YAML ファイルを使用することが推奨されます。

2.1.2. YAML バンドルを使用した AMQ Online のインストール

AMQ Online をインストールする最も簡単な方法は、定義済みの YAML バンドルを使用することです。

手順

1. **cluster-admin** 権限を持つユーザーとしてログインしてます。

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

2. (オプション) **amq-online-infra** 以外のプロジェクトにデプロイする場合は、次のコマンドを実行し、後続の手順で **amq-online-infra** を置き換える必要があります。

```
sed -i 's/amq-online-infra/my-project/' install/bundles/amq-online/*.yaml
```

3. AMQ Online をデプロイするプロジェクトを作成します。

```
oc new-project amq-online-infra
```

4. ダウンロードしたリリースファイルの場所にディレクトリーを変更します。

5. **amq-online** バンドルを使用してデプロイします。

```
oc apply -f install/bundles/amq-online
```

6. (オプション) サンプルプランとインフラストラクチャー設定をインストールします。

```
oc apply -f install/components/example-plans
```

7. (オプション) サンプルのロールをインストールします。

```
oc apply -f install/components/example-roles
```

8. (オプション) **標準** 認証サービスをインストールします。

```
oc apply -f install/components/example-authservices/standard-authservice.yaml
```

9. (オプション) Service Catalog 統合をインストールします。

```
oc apply -f install/components/service-broker
oc apply -f install/components/cluster-service-broker
```

2.2. OPERATOR LIFECYCLE MANAGER を使用した AMQ ONLINE のインストールと設定

Operator Lifecycle Manager を使用して、AMQ Online の評価インスタンスをインストールおよび設定できます。

OpenShift Container Platform 4.1 では、Operator Lifecycle Manager (OLM) を使用することにより、ユーザーはすべての Operator およびクラスター全体で実行される関連サービスをインストールし、更新し、管理することができます。これは、Kubernetes のネイティブアプリケーション (Operator) を効率的に自動化された拡張可能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。

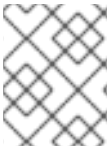
OLM は OpenShift Container Platform 4.1 でデフォルトで実行されます。OpenShift Container Platform コンソールでは、クラスター管理者が Operator をインストールし、特定のプロジェクトアクセスを付与して、クラスターで利用可能な Operator のカタログを使用するための管理画面を利用できます。

OperatorHub は、OpenShift クラスター管理者が Operator を検出、インストール、および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。

2.2.1. OpenShift Container Platform コンソールを使用した OperatorHub から AMQ Online へのインストール

OpenShift Container Platform コンソールの OperatorHub を使用して、OpenShift Container Platform 4.1 クラスターに AMQ Online Operator をインストールします。



注記

AMQ Online Operator を **openshift-operator** プロジェクトにインストールしてデプロイする必要があります。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform 4.1 クラスターにアクセスできる。

手順

1. OpenShift Container Platform コンソールで、**cluster-admin** 権限を持つアカウントを使用してログインします。
2. **Operators** > **OperatorHub** の順にクリックします。
3. **Filter by keyword** ボックスに **AMQ Online** と入力して、AMQ Online Operator を見つけます。
4. AMQ Online Operator をクリックします。operator に関する情報が表示されます。
5. Operator についての情報を確認してから、**Install** をクリックします。Create Operator Subscription のページが表示されます。
6. **Create Operator Subscription** ページで、すべてのデフォルト設定を受け入れ **Subscribe** をクリックします。

   注記

   クラスター上のすべての namespace (デフォルト) は、デフォルトの **openshift-operators** プロジェクトに Operator をインストールし、クラスター内のすべてのプロジェクトで Operator を使用できるようにします。

   amq-online ページが表示され、AMQ Online Operator サブスクリプションのインストールの進行状況を監視できます。
7. サブスクリプションのアップグレードステータスが **Up to date** と表示されたら、**Operators** > **Installed Operators** をクリックして、**AMQ Online** ClusterServiceVersion (CSV) が表示され、その **ステータス** が最終的に **openshift-operators** プロジェクトで **InstallSucceeded** に解決されることを確認します。

注記

All namespaces... インストールモードの場合、ステータスは **openshift-operators** プロジェクトで **InstallSucceeded** に解決されますが、他のプロジェクトを表示すると、ステータスは **Copied** になります。

トラブルシューティングに関する情報は、OpenShift Container Platform のドキュメント を参照してください。

次のステップ

- OpenShift Container Platform コンソールを使用した AMQ Online の設定

2.2.2. OpenShift Container Platform コンソールを使用した AMQ Online の設定

OpenShift Container Platform コンソールを使用して OperatorHub から AMQ Online をインストールした後、**openshift-operators** プロジェクト内の以下のアイテムに対してカスタムリソースの新しいインスタンスを作成します。

- アドレス空間タイプのサービスインフラストラクチャー設定 (例では標準のアドレス空間タイプを使用)
- 認証サービス
- アドレス空間計画
- アドレス計画

カスタムリソースの新しいインスタンスを作成したら、次に以下を実行します。

- アドレス空間の作成
- アドレスの作成
- メッセージングユーザーの作成

以下の手順では、OpenShift Container Platform コンソールの使用時に提供されるサンプルデータを使用します。

2.2.2.1. OpenShift Container Platform コンソールを使用したインフラストラクチャー設定のカスタムリソースの作成

AMQ Online を使用するには、インフラストラクチャー設定のカスタムリソースを作成する必要があります。この例では、標準アドレス空間に **StandardInfraConfig** を使用します。

手順

1. 右上にある **プラス** アイコン (+) をクリックします。YAML のインポートウィンドウが開きます。
2. 左上のドロップダウンメニューから、**amq-online-infra** プロジェクトを選択します。
3. 次のコードをコピーします。

```
apiVersion: admin.enmasse.io/v1beta1
```

```
kind: StandardInfraConfig
metadata:
  name: default
```

4. YAML のインポートウィンドウで、コピーしたコードを貼り付けて **作成** をクリックします。StandardInfraConfig の概要ページが表示されます。

5. **Operators > Installed Operators**の順にクリックします。

6. AMQ Online Operator をクリックし、**Standard Infra Config** タブをクリックして、**Status** が **Active** と表示されていることを確認します。

次のステップ

- OpenShift Container Platform コンソールを使用した認証サービスのカスタムリソースの作成

2.2.2.2. OpenShift Container Platform コンソールを使用した認証サービスのカスタムリソースの作成

AMQ Online を使用するには、認証サービス用のカスタムリソースを作成する必要があります。この例では、標準の認証サービスを使用しています。

手順

1. 右上にある **プラス** アイコン (+) をクリックします。YAML のインポートウィンドウが開きます。

2. 左上のドロップダウンメニューから、**amq-online-infra** プロジェクトを選択します。

3. 次のコードをコピーします。

```
apiVersion: admin.enmasse.io/v1beta1
kind: AuthenticationService
metadata:
  name: standard-authservice
spec:
  type: standard
```

4. YAML のインポートウィンドウで、コピーしたコードを貼り付けて **作成** をクリックします。AuthenticationService の概要ページが表示されます。

5. **Workloads > Pods**をクリックします。カスタムリソースがデプロイされると、**Readiness** 列の Pod ステータスは **Ready** になります。

次のステップ

- OpenShift Container Platform コンソールを使用したアドレス空間計画のカスタムリソースの作成

2.2.2.3. OpenShift Container Platform コンソールを使用したアドレス空間計画のカスタムリソースの作成

AMQ Online を使用するには、アドレス空間計画のカスタムリソースを作成する必要があります。この手順では、OpenShift Container Platform コンソールの使用時に提供されるサンプルデータを使用します。

手順

1. 右上にある **プラス** アイコン (+) をクリックします。YAML のインポートウィンドウが開きます。
2. 左上のドロップダウンメニューから、**amq-online-infra** プロジェクトを選択します。
3. 次のコードをコピーします。

```
apiVersion: admin.enmasse.io/v1beta2
kind: AddressSpacePlan
metadata:
  name: standard-small
spec:
  addressSpaceType: standard
  infraConfigRef: default
  addressPlans:
    - standard-small-queue
  resourceLimits:
    router: 2.0
    broker: 3.0
    aggregate: 4.0
```

4. YAML のインポートウィンドウで、コピーしたコードを貼り付けて **作成** をクリックします。AddressSpacePlan の概要ページが表示されます。
5. **Operators** > **Installed Operators**の順にクリックします。
6. AMQ Online Operator をクリックし、**Address Space Plan** タブをクリックして、**Status** が **Active** と表示されていることを確認します。

次のステップ

- OpenShift Container Platform コンソールを使用したアドレス計画のカスタムリソースの作成

2.2.2.4. OpenShift Container Platform コンソールを使用したアドレス計画のカスタムリソースの作成

AMQ Online を使用するには、アドレス計画のカスタムリソースを作成する必要があります。この手順では、OpenShift Container Platform コンソールの使用時に提供されるサンプルデータを使用します。

手順

1. 右上にある **プラス** アイコン (+) をクリックします。YAML のインポートウィンドウが開きます。
2. 左上のドロップダウンメニューから、**amq-online-infra** プロジェクトを選択します。
3. 次のコードをコピーします。

```
apiVersion: admin.enmasse.io/v1beta2
```

```
kind: AddressPlan
metadata:
  name: standard-small-queue
spec:
  addressType: queue
  resources:
    router: 0.01
    broker: 0.1
```

4. YAML のインポートウィンドウで、コピーしたコードを貼り付けて **作成** をクリックします。AddressPlan の概要ページが表示されます。

5. **Operators > Installed Operators**の順にクリックします。

6. AMQ Online Operator をクリックし、**Address Plan** タブをクリックして、**Status** が **Active** と表示されていることを確認します。

次のステップ

- アドレス空間の作成
- アドレスの作成
- メッセージングユーザーの作成

2.3. コマンド行を使用したアドレス空間の作成

AMQ Online では、標準のコマンドラインツールを使用してアドレス空間を作成します。

手順

1. メッセージングテナントとしてログインします。

```
oc login -u developer
```

2. メッセージングアプリケーションのプロジェクトを作成します。

```
oc new-project myapp
```

3. アドレス空間定義を作成します。

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

4. アドレス空間を作成します。

```
oc create -f standard-address-space.yaml
```

5. アドレス空間のステータスを確認します。

```
oc get addressspace myspace -o jsonpath={.status.isReady}
```

前のコマンドが **true** を出力すると、アドレス空間を使用する準備が整います。

2.4. コマンドラインを使用したアドレスの作成

コマンドラインを使用してアドレスを作成できます。

手順

1. アドレス定義を作成します。

```
apiVersion: enmasse.io/v1beta1
kind: Address
metadata:
    name: myspace.myqueue
spec:
    address: myqueue
    type: queue
    plan: standard-small-queue
```

注記

異なるアドレス空間からのアドレスが競合しないようにするには、名前の前にアドレス空間名を付ける必要があります。

2. アドレスを作成します。

```
oc create -f standard-small-queue.yaml
```

3. アドレスを一覧表示します。

```
oc get addresses -o yaml
```

2.5. コマンドラインでのユーザーの作成

AMQ Online では、標準のコマンドラインツールを使用してユーザーを作成できます。

前提条件

- アドレス空間 を作成しておく。

手順

1. ユーザー定義ファイルのパスワードを正しく base64 エンコードするには、次のコマンドを実行します。

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

注記

このコマンドを実行するときは、必ず **-n** パラメーターを使用してください。このパラメーターを指定しないと、パスワードが正しくコード化されず、ログインの問題が発生します。

2. ユーザー定義をファイルに保存します。

```
apiVersion: user.enmasse.io/v1beta1
kind: MessagingUser
metadata:
  name: myspace.user1
spec:
  username: user1
  authentication:
    type: password
    password: cGFzc3dvcmQ= # Base64 encoded
  authorization:
    - addresses: ["myqueue", "queue1", "queue2", "topic*"]
      operations: ["send", "recv"]
    - addresses: ["anycast1"]
      operations: ["send"]
```

3. ユーザーと関連するユーザー権限を作成します。

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

4. ユーザーが作成されたことを確認します。

```
oc get messagingusers
```

2.6. メッセージの送受信

前提条件

- Apache Qpid Proton Python バインディングをインストールしている。
- **myspace** という名前のアドレス空間を作成しておく。
- **myqueue** という名前のアドレスを作成しておく。
- パスワード **password** を持つ **user1** という名前のユーザーを作成しておく。

手順

1. Python クライアントの例をファイルに保存します。

```
from __future__ import print_function, unicode_literals
import optparse
from proton import Message
from proton.handlers import MessagingHandler
from proton.reactor import Container
```

```
class HelloWorld(MessagingHandler):
    def __init__(self, url):
        super(HelloWorld, self).__init__()
        self.url = url

    def on_start(self, event):
        event.container.create_receiver(self.url)
        event.container.create_sender(self.url)

    def on_sendable(self, event):
        event.sender.send(Message(body="Hello World!"))
        event.sender.close()

    def on_message(self, event):
        print("Received: " + event.message.body)
        event.connection.close()

parser = optparse.OptionParser(usage="usage: %prog [options]")
parser.add_option("-u", "--url", default="amqps://localhost:5672/myqueue",
                  help="url to use for sending and receiving messages")
opts, args = parser.parse_args()

try:
    Container(HelloWorld(opts.url)).run()
except KeyboardInterrupt: pass
```

2. アドレス空間のメッセージングエンドポイントのホスト名を取得します。

```
oc get addressspace myspace -o 'jsonpath={.status.endpointStatuses[?
(@.name=="messaging")].externalHost}'
```

次の手順で、出力をホスト名として使用します。

3. クライアントを実行します。

```
python client-example1.py -u
amqps://user1:password@messaging.example1.com:443/myqueue
```

第3章 AMQ ONLINE のモノのインターネット (IOT)

重要

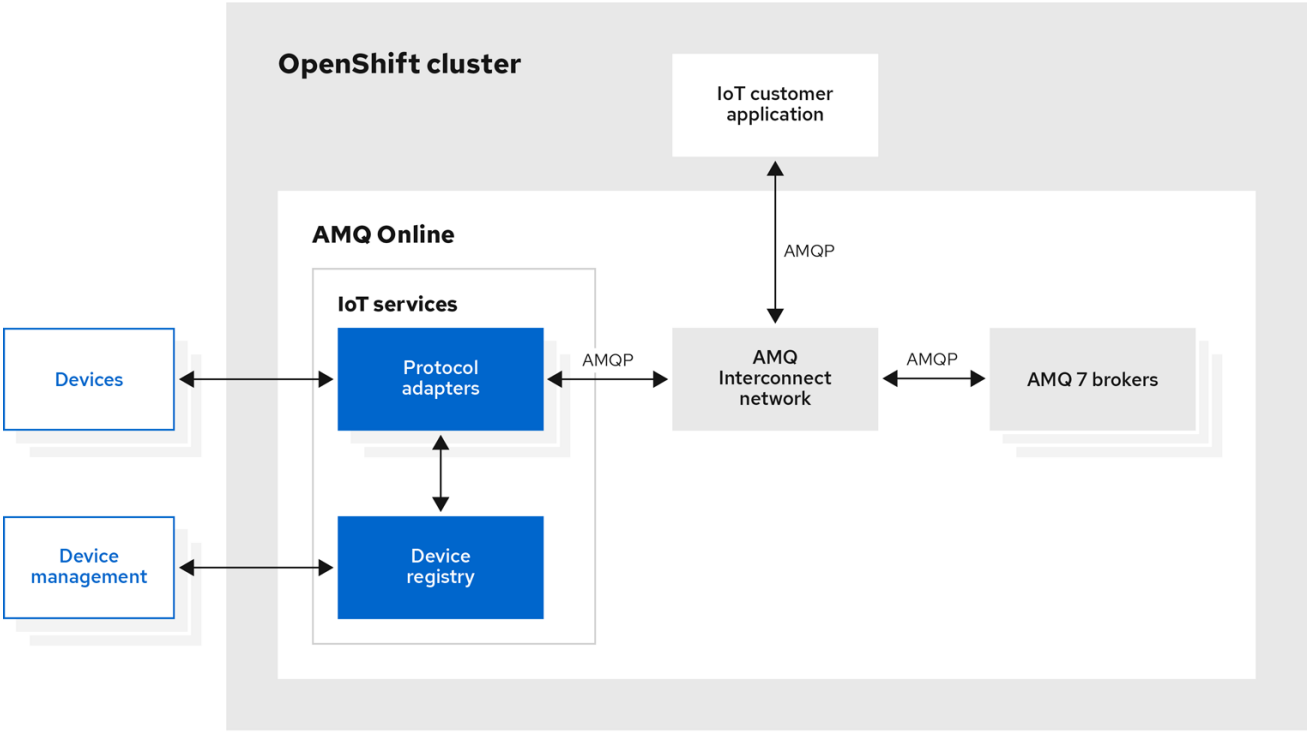
AMQ Online のモノのインターネット (IoT) 機能は、テクノロジープレビュー機能のみです。テクノロジープレビュー機能は、Red Hat の実稼働環境のサービスレベルアグリーメント (SLA) ではサポートされません。また、機能的に完全ではない可能性があるため、Red Hat はテクノロジープレビュー機能を実稼働環境に実装することは推奨しません。テクノロジープレビューの機能は、最新の技術をいち早く提供して、開発段階で機能のテストやフィードバックの収集を可能にするために提供されます。サポート範囲の詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

3.1. IOT の使用開始

次の情報では、AMQ Online IoT 機能を設定および管理する方法について説明します。

3.1.1. IoT 接続の概念

モノのインターネット (IoT) 接続機能により、AMQ Online を使用してデバイスを管理し、バックエンドアプリケーションと接続できます。一般的な IoT アプリケーションでは、デバイスには通常のメッセージングアプリケーションとは異なる要件があります。開発者は、通常利用可能な任意のアドレスとセキュリティー設定を使用する代わりに、IoT サービスを使用して、デバイスの ID とセキュリティー設定を明示的に処理し、IoT 空間でよく使用される複数のプロトコルをサポートして、一貫性を持ったサポートを提供して、必要なデバイス通信パターンをサポートします。



重要な概念の 1 つは **デバイスレジストリー** です。これは、開発者がデバイスを登録して認証情報を提供するために使用します。これらの認証情報を使用すると、デバイスは、サポートされているプロトコルの 1 つ (HTTP、MQTT、LoRaWAN、および SigFox) を使用して **プロトコルアダプター** に接続できます。接続すると、デバイスは、次のメッセージングセマンティクスのいずれかを使用して、バックエンドアプリケーションからメッセージを送受信できます。

- Telemetry: デバイスが非永続データをバックエンドアプリケーションに送信できるようにするため、メッセージは **multicast** アドレスタイプを使用して送信されます。このオプションは、重要でないセンサーの読み取り値を送信する場合に最適です。
- Events: デバイスが永続的なデータをバックエンドアプリケーションに送信できるようにするため、**queue** アドレスタイプを使用してメッセージが送信されます。このオプションは、アラートや通知など、より重要なデバイスデータを送信する場合に最適です。

バックエンドアプリケーションは、**コマンド** メッセージをデバイスに送信することもできます。コマンドを使用して、デバイスでアクションをトリガーできます。例としては、設定プロパティーの更新、ソフトウェアコンポーネントのインストール、アクチュエーターの状態の切り替えなどがあります。

3.1.2. YAML バンドルを使用した AMQ Online のインストール

AMQ Online をインストールする最も簡単な方法は、定義済みの YAML バンドルを使用することです。

手順

1. **cluster-admin** 権限を持つユーザーとしてログインしてます。

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

2. (オプション) **amq-online-infra** 以外のプロジェクトにデプロイする場合は、次のコマンドを実行し、後続の手順で **amq-online-infra** を置き換える必要があります。

```
sed -i 's/amq-online-infra/my-project/' install/bundles/amq-online/*.yaml
```

3. AMQ Online をデプロイするプロジェクトを作成します。

```
oc new-project amq-online-infra
```

4. ダウンロードしたリリースファイルの場所にディレクトリーを変更します。
5. **amq-online** バンドルを使用してデプロイします。

```
oc apply -f install/bundles/amq-online
```

6. (オプション) サンプルプランとインフラストラクチャー設定をインストールします。

```
oc apply -f install/components/example-plans
```

7. (オプション) サンプルのロールをインストールします。

```
oc apply -f install/components/example-roles
```

8. (オプション) **標準** 認証サービスをインストールします。

```
oc apply -f install/components/example-authservices/standard-authservice.yaml
```

9. (オプション) Service Catalog 統合をインストールします。

```
oc apply -f install/components/service-broker
oc apply -f install/components/cluster-service-broker
```

3.1.3. IoT サービスのインストール

AMQ Online で IoT 機能の使用を開始するには、まず IoT サービスをインストールする必要があります。

手順

1. (オプション) **amq-online-infra** 以外のプロジェクトにデプロイする場合は、次のコマンドを実行し、後続の手順で **amq-online-infra** を置き換える必要があります。

```
sed -i 's/amq-online-infra/my-project/' install/preview-bundles/iot/*.yaml
```

2. IoT バンドルをデプロイします。

```
oc apply -f install/preview-bundles/iot
```

3. MQTT プロトコルアダプターの証明書を作成します。テスト目的で、自己署名証明書を作成できます。

```
./install/components/iot/examples/k8s-tls/create
oc create secret tls iot-mqtt-adapter-tls --key=install/components/iot/examples/k8s-tls/build/iot-mqtt-adapter-key.pem --cert=install/components/iot/examples/k8s-tls/build/iot-mqtt-adapter-fullchain.pem
```

スクリプトを呼び出すときに環境変数 **NAMESPACE** を設定して、**deploy** スクリプトがキーと証明書をインストールする namespace をオーバーライドできます。以下に例を示します。

```
NAMESPACE=my-namespace ./install/components/iot/examples/k8s-tls/deploy
```

注記

クラスターが **localhost** で実行されていない場合には、外部クライアント (MQTT など) が適切なサービスに適切に接続できるように、証明書を作成するときにクラスターのホスト名を指定する必要があります。以下に例を示します。

```
CLUSTER=x.x.x.x.nip.io install/components/iot/examples/k8s-tls/create
```

4. (オプション) PostgreSQL サーバーをインストールし、データベースを作成します。

```
oc apply -f install/components/iot/examples/postgresql/deploy
```

すでに PostgreSQL インスタンスがあり、それにアクセスするユーザーでデータベースを作成した場合は、この手順を省略できます。

5. データベーススキーマを適用します。
作成したデータベースインスタンスで次の SQL ファイルを実行する必要があります。設定によっては、データベース管理者権限が必要になる場合があります。

- **install/components/iot/examples/postgresql/create.sql**
- **install/components/iot/examples/postgresql/create.devcon.sql**

データベースに接続された **psql** コマンドを使用して、SQL ファイルを実行できます。以下は、前の手順で説明したように PostgreSQL をインストールしたときに、コンテナー内から **psql** を実行する方法の例を示しています。

```
oc exec -ti deployment/postgresql -- bash -c "PGPASSWORD=user12 psql device-registry
registry" < install/components/iot/examples/postgresql/create.sql
oc exec -ti deployment/postgresql -- bash -c "PGPASSWORD=user12 psql device-registry
registry" < install/components/iot/examples/postgresql/create.devcon.sql
```

6. IoT インフラストラクチャー設定の例をインストールします。

```
oc apply -f install/components/iot/examples/iot-config.yaml
```

3.1.4. IoT プロジェクトの作成

IoT サービスをインストールしたら、IoT プロジェクトを作成します。

前提条件

- IoT サービスをインストールしている。

手順

1. メッセージングテナントとしてログインします。

```
oc login -u developer
```

2. **管理対象** IoT プロジェクトを作成します。

```
oc new-project myapp
oc create -f install/components/iot/examples/iot-project-managed.yaml
```

3. リソースの準備が整うまで待ちます。

```
oc get addressspace iot
oc get iotproject iot
```

注記

Phase フィールドに両方のリソースの **Ready** ステータスが表示されていることを確認します。

4. メッセージングコンシューマーユーザーを作成します。

```
oc create -f install/components/iot/examples/iot-user.yaml
```

3.1.5. IoT デバイスの作成

IoT サービスをインストールして IoT プロジェクトを作成したら、監視するデバイス用の IoT デバイスを作成できます。

前提条件

前提条件

- IoT サービスをインストールしている。
- IoT プロジェクトを作成している。

3.1.5.1. 新しいデバイスの登録

新しいデバイスを作成するには、最初にデバイスを登録する必要があります。

手順

1. デバイスレジストリーホストをエクスポートします。

```
export REGISTRY_HOST=$(oc -n amq-online-infra get routes device-registry --template='{{
.spec.host }}')
```

2. デバイスレジストリーアクセストークンをエクスポートします。

```
export TOKEN=$(oc whoami --show-token)
```

このトークンは、デバイスレジストリー管理 API に対する認証に使用されます。

3. 定義された ID でデバイスを登録します (この例では **4711** を使用します)。

```
curl --insecure -X POST -i -H 'Content-Type: application/json' -H "Authorization: Bearer
${TOKEN}" https://$REGISTRY_HOST/v1/devices/myapp.iot/4711
```

4. (オプション) 追加の登録情報を指定する必要がある場合は、次のように行います。

```
curl --insecure -X POST -i -H 'Content-Type: application/json' -H "Authorization: Bearer
${TOKEN}" --data-binary '{
 "via": ["gateway1"]
}' https://$REGISTRY_HOST/v1/devices/myapp.iot/4711
```

3.1.5.2. デバイスのユーザー名とパスワード認証情報の設定

新しいデバイスを登録したら、デバイスのユーザー名とパスワード認証情報を設定する必要があります。

手順

1. デバイスの認証情報を追加します。

```
curl --insecure -X PUT -i -H 'Content-Type: application/json' -H "Authorization: Bearer
${TOKEN}" --data-binary '[{
 "type": "hashed-password",
 "auth-id": "sensor1",
 "secrets": [{
  "pwd-plain":"'hono-secret'"
 }]
}]' https://$REGISTRY_HOST/v1/credentials/myapp.iot/4711
```

3.1.6. Eclipse Hono コマンドラインクライアントのインストール

前提条件

- IoT サービスをインストールしている。
- IoT プロジェクトを作成している。
- IoT デバイスを作成している。

手順

1. Eclipse Hono コマンドラインクライアント をダウンロードします。
2. メッセージングエンドポイント証明書を取得します。

```
oc -n myapp get addressspace iot -o jsonpath={.status.caCert} | base64 --decode > tls.crt
```

3. メッセージングエンドポイントのホストとポートをエクスポートします。

```
export MESSAGING_HOST=$(oc get -n myapp addressspace iot -o
jsonpath='{.status.endpointStatuses[?(@.name=="messaging")].externalHost}')
export MESSAGING_PORT=443
```

3.1.7. telemetry コンシューマーの開始

たとえば、センサーの読み取り値などの telemetry データをデバイスからクラウドに送信できます。これを行うには、最初に顧客アプリケーションを実行して telemetry コンシューマーを開始する必要があります。

手順

1. 顧客アプリケーションを実行して、telemetry を受信します。

```
java -jar hono-cli-*-exec.jar --hono.client.host=$MESSAGING_HOST --
hono.client.port=$MESSAGING_PORT --hono.client.username=consumer --
hono.client.password=foobar --tenant.id=myapp.iot --hono.client.trustStorePath=tls.crt --
message.type=telemetry
```

3.1.8. HTTP を使用した telemetry の送信

HTTP プロトコルを使用して、デバイスからクラウドに telemetry を送信できます。

手順

1. HTTP プロトコルを使用して telemetry を送信します。

```
curl --insecure -X POST -i -u sensor1@myapp.iot:hono-secret -H 'Content-Type:
application/json' --data-binary '{"temp": 5}' https://$(oc -n amq-online-infra get routes iot-http-
adapter --template='{{ .spec.host }}')/telemetry
```

3.2. サービス管理者向け IOT

通常、サービス管理者は、AMQ Online のモノのインターネット (IoT) サービスをインストールして設定します。

3.2.1. IoT モニタリング

AMQ Online 監視をデプロイメントすると、AMQ Online の IoT 関連コンポーネントを監視することもできます。追加の設定は必要ありません。

AMQ Online 監視の詳細は、AMQ Online の監視 を参照してください。

IoT に固有の使用可能なメトリックの詳細は、IoT 固有のメトリック を参照してください。

3.2.2. IoT ロギング

デフォルトでは、AMQ Online の IoT コンポーネントは、インフラストラクチャーサービスの保守的なログ設定を使用して、通常の操作用のリソースを保持します。

AMQ Online では、より詳細なトレースシナリオ向けに、Jaeger を使用した操作レベルのトレースも可能です。

3.2.2.1. 設定オプション

IoT コンポーネントの場合に、次のアプリケーションレベルでログ出力を増やす必要がある場合があります。レベルは、優先度が低いものから高いものの順にリストされています。

- すべての IoT コンポーネントのデフォルトのログレベル
- 特定のログチャネルのデフォルト設定
- サービス固有のデフォルトのログレベル
- 特定のログチャネルのサービス固有の設定
- サービス固有のカスタムログ設定ファイル

すべての設定は **IoTConfig** インスタンスの一部であり、AMQ Online Operator は設定をサービスに適用します。唯一の例外は、サービス固有のログ設定ファイルです。これは、**IoTConfig** リソースを使用するか、またはサービス固有の **ConfigMap** リソースにエントリーを作成することによって提供できます。

3.2.2.2. ログレベル

次のログレベルが利用可能で、重大度が最も高いものから順にリストされています。

error

エラーのある状況。システムの安定性に影響を与える可能性のある予期しない状態を示します。エラーメッセージのみを表示します。

warn

警告のある状況。システムの現在の動作または安定性に影響を与える可能性のある、予期できる状態を示します。警告メッセージとエラーメッセージのみを表示します。

info

情報メッセージ。現在の操作に影響を与える可能性のある、予期できるイベントを示します。情報、警告、およびエラーメッセージのみを表示します。

debug

上記のすべてに加えて、デバッグメッセージを表示します。

trace

すべてのメッセージを表示します。

3.2.3. IoT トレース

AMQ Online では、Jaeger を使用して IoT インフラストラクチャーでアプリケーションレベルのトレースを行うことができます。この機能により、サービス管理者は IoT サービスの内部の仕組みを把握して、システムのパフォーマンスと問題を分析できます。

デフォルトでは、トレースサポートは有効になっていないため、手動で有効にする必要があります。詳しくは、トレースの設定 を参照してください。

Jaeger トレースの詳細は、https://www.jaegertracing.io/ を参照してください。

3.2.4. デバイスレジストリー

AMQ Online の IoT コンポーネントは、すべてのデバイス関連情報をデバイスレジストリーと呼ばれるサービスに保存します。これにより、デバイスレジストリーは IoT 機能全体の重要なコンポーネントになるため、デバイスレジストリーの設定を微調整する必要がある場合があります。

警告

デバイスレジストリーストレージバックエンドはさまざまな方法で設定できますが、このような設定を行い、IoT テナントを作成したら、ストレージ設定は変更できません。そうしないと、データの損失、データの不整合、またはその他の予期しない動作が発生する可能性があります。

デバイスレジストリーの設定は、グローバルな **IoTConfig** カスタムリソースオブジェクトを編集することで変更できます。このカスタムリソースに加えられた変更は、AMQ Online Operator によって適用されます。

JDBC という名前のデータベースでバックアップされたデバイスレジストリーは、既存の外部データベースを使用するように設定できます。

デフォルトでは、PostgreSQL のみがサポートされています。ただし、カスタム JDBC ドライバーとカスタム SQL ステートメントを設定に指定して、インストールを拡張できます。これにより、PostgreSQL 以外のデータベースと統合できます。

3.2.5. IoT サービスのインストール

AMQ Online で IoT 機能の使用を開始するには、まず IoT サービスをインストールする必要があります。

手順

1. (オプション) **amq-online-infra** 以外のプロジェクトにデプロイする場合は、次のコマンドを実行し、後続の手順で **amq-online-infra** を置き換える必要があります。

```
sed -i 's/amq-online-infra/my-project/' install/preview-bundles/iot/*.yaml
```

2. IoT バンドルをデプロイします。

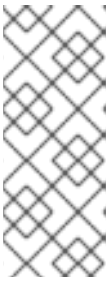
```
oc apply -f install/preview-bundles/iot
```

3. MQTT プロトコルアダプターの証明書を作成します。テスト目的で、自己署名証明書を作成できます。

```
./install/components/iot/examples/k8s-tls/create
oc create secret tls iot-mqtt-adapter-tls --key=install/components/iot/examples/k8s-tls/build/iot-mqtt-adapter-key.pem --cert=install/components/iot/examples/k8s-tls/build/iot-mqtt-adapter-fullchain.pem
```

スクリプトを呼び出すときに環境変数 **NAMESPACE** を設定して、**deploy** スクリプトがキーと証明書をインストールする namespace をオーバーライドできます。以下に例を示します。

```
NAMESPACE=my-namespace ./install/components/iot/examples/k8s-tls/deploy
```



注記

クラスターが **localhost** で実行されていない場合には、外部クライアント (MQTT など) が適切なサービスに適切に接続できるように、証明書を作成するときにクラスターのホスト名を指定する必要があります。以下に例を示します。

```
CLUSTER=x.x.x.x.nip.io install/components/iot/examples/k8s-tls/create
```

4. (オプション) PostgreSQL サーバーをインストールし、データベースを作成します。

```
oc apply -f install/components/iot/examples/postgresql/deploy
```

すでに PostgreSQL インスタンスがあり、それにアクセスするユーザーでデータベースを作成した場合は、この手順を省略できます。

5. データベーススキーマを適用します。
作成したデータベースインスタンスで次の SQL ファイルを実行する必要があります。設定によっては、データベース管理者権限が必要になる場合があります。

- **install/components/iot/examples/postgresql/create.sql**
- **install/components/iot/examples/postgresql/create.devcon.sql**

データベースに接続された **psql** コマンドを使用して、SQL ファイルを実行できます。以下は、前の手順で説明したように PostgreSQL をインストールしたときに、コンテナー内から **psql** を実行する方法の例を示しています。

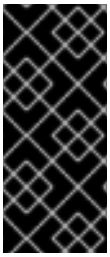
```
oc exec -ti deployment/postgresql -- bash -c "PGPASSWORD=user12 psql device-registry registry" < install/components/iot/examples/postgresql/create.sql
oc exec -ti deployment/postgresql -- bash -c "PGPASSWORD=user12 psql device-registry registry" < install/components/iot/examples/postgresql/create.devcon.sql
```

6. IoT インフラストラクチャー設定の例をインストールします。

```
oc apply -f install/components/iot/examples/iot-config.yaml
```

3.2.6. JDBC 外部デバイスレジストリーのデプロイ

外部データベースを使用するには、データベースインスタンスを作成し、必要なテーブルとインデックスを作成する必要があります。



重要

技術的には、JDBC ドライバーを提供し、SQL をサポートするデータベースであればどれでも使用できるはずですが、現時点では、PostgreSQL のみが AMQ Online でサポートされています。明示的に言及されていない限り、このドキュメントは PostgreSQL を使用していることを前提としています。そうでない場合は、指定のコマンドと SQL ステートメントを調整する必要がある場合があります。

JDBC ベースのデバイスレジストリーを設定するには、次の手順を実行する必要があります。

3.2.6.1. データストレージモデルの選択

JDBC ベースのデバイスレジストリーは、次のデータモデルをサポートしています。

- フラットな JSON
- 階層的な JSON
- プレーンテーブル

JSON ベースのデータモデルには、ロックや外部キーは必要ありません。ただし、JSONB の PostgreSQL サポートに依存しています。フラットな JSON モデルは、さまざまな種類の認証情報を格納する場合により柔軟です。階層的な JSON モデルは、フラットな JSON モデルよりも優れたパフォーマンスを発揮しますが、このパフォーマンスを実現するには、認証情報の種類ごとに専用のインデックスが必要です。

プレーンテーブルモデルは、JSON 固有のデータベースサポートを必要としませんが、外部キーでリンクされた複数のテーブルを必要とし、変更時にロックのサポートが必要になります。反対に、ほとんどの場合、読み取りパフォーマンスが向上します。

デフォルトの選択は、階層的な JSON モデルです。

注記

後でデータモデルを変更すると、すべてのデータが失われるか、データを手動で移行する必要があります。

3.2.6.2. データベースインスタンスの作成

まず、データベースインスタンスを作成する必要があります。また、少なくとも 2 種類のユーザーを作成することをお勧めします。1 つはデータベースを管理するためのもので、もう 1 つはデバイスレジストリー固有のテーブルにアクセスするためのものです。以下のセクションでは、前者のユーザーを admin、後者を registry とします。

3.2.6.3. SQL スキーマのデータベースインスタンスへのデプロイ

前提条件

- データベースインスタンスを作成している。
- admin データベースユーザーのアクセス認証情報がある。

手順

1. 管理者ユーザーを使用してデータベースインスタンスに接続します。
2. SQL スキーマ **templates/iot/examples/postgresql/create.sql** を確認してデプロイします。

3.2.6.4. IoT インフラストラクチャーの設定

外部 JDBC デバイスレジストリーの実装を有効にするには、**.spec.services.deviceRegistry.jdbc.server.external** セクションを設定し、選択したデータモデル、データベース接続情報、およびアクセス認証情報を提供する必要があります。

例については 外部 PostgreSQL を使用した JDBC の設定 を参照してください。

3.2.7. ロギングの設定

デフォルトのロギング設定が十分でない場合、次のセクションでは、ロギングシステム設定の各種方法について説明します。

3.2.7.1. グローバルログレベルの設定

グローバルロギング設定は、明示的なロギング設定がないすべてのサービスに適用されます。

デフォルトでは、グローバルログレベルは **info** です。

前提条件

- IoT サービスがインストールされている。

手順

1. **default** という名前の **IoTConfig** インスタンスを編集します。

```
oc edit iotconfig default
```

2. ロギングオプションを設定し、保存してエディターを終了します。

```
apiVersion: iot.enmasse.io/v1alpha1
kind: IoTConfig
metadata:
  namespace: enmasse-infra
  name: default
spec:
  logging:
    level: info (1)
    loggers: (2)
      io.enmasse: debug (3)
      io.netty: error (4)
```

1. デフォルトのグローバルログレベル。省略した場合は、**info** が使用されます。
2. ログチャネル固有のエントリーのセクション。
3. チャネル **io.enmasse** 内のメッセージのフィルタリングを **debug** レベルに下げます。
4. チャネル **io.netty** 内のメッセージのフィルタリングを **error** レベルに上げます。

3. Operator はロギング設定を適用し、必要なすべてのコンポーネントを再デプロイします。

上記の例では、以下のようになります。

- ロガーが明示的な設定と一致せず、グローバルなデフォルトが **info** であるため、このロガー **org.eclipse.hono** の **Info** メッセージはログに記録されます。
- **io.enmasse** の設定が **debug** であり、**info** メッセージの重大度が高いため、このロガー **io.enmasse** の **info** メッセージはログに記録されます。
- **io.netty** の設定が **error** メッセージのみを表示するように設定されているため、ロガー **io.netty** の **warn** メッセージは削除されます。

3.2.7.2. アプリケーション固有のログレベルの設定

グローバルな既定値をオーバーライドするために、IoT サービス専用のログを設定できます。

前提条件

- IoT サービスがインストールされている。

手順

1. **default** という名前の **IoTConfig** インスタンスを編集します。

```
oc edit iotconfig default
```

2. ロギングオプションを設定し、保存してエディターを終了します。

```
apiVersion: iot.enmasse.io/v1alpha1
kind: IoTConfig
metadata:
  namespace: enmasse-infra
  name: default
spec:
  adapters:
    mqtt:
      containers:
        adapter:
          logback:
            level: info (1)
            loggers: (2)
              io.enmasse: debug (3)
              io.netty: error (4)
```

1. アプリケーションのグローバルログレベル。省略した場合には、デフォルトのグローバル

レベルが使用されます。

2. ログチャネル固有のエントリーのセクション。省略され、アプリケーションのグローバルログレベルも省略された場合には、インフラストラクチャーの既定のログチャネル設定が使用されます。アプリケーショングローバルログレベルが設定されている場合には、それは空のセットと見なされ、ログチャネル固有の設定は適用されません。

3. チャネル **io.enmasse** 内のメッセージのフィルタリングを **debug** レベルに下げます。

4. チャネル **io.netty** 内のメッセージのフィルタリングを **error** レベルに上げます。

3. Operator はロギング設定を適用し、必要なすべてのコンポーネントを再デプロイします。

3.2.7.3. カスタム logback 固有設定の適用

Logback ロギングの実装を使用してアプリケーションを実行しているコンテナーの場合には、カスタムの XML ベースの logback 設定ファイルを提供できます。これにより、システム内の他のログ設定が上書きされます。

> **警告**
>
> ロギング設定は、AMQ Online でチェックされません。不適切な設定を指定すると、パフォーマンスや安定性が失われたり、システム全体の障害につながる可能性があります。

Logback の設定の詳細は、http://logback.qos.ch/manual/configuration.html を参照してください。

前提条件

- IoT サービスがインストールされている。
- XML ベースの logback 設定ファイルのコンテンツを作成します。

3.2.7.3.1. IoTConfig リソースの使用

IoTConfig リソースを使用して設定を適用できます。

手順

1. **default** という名前の **IoTConfig** インスタンスを編集します。

```
oc edit iotconfig default
```

2. ロギングオプションを設定し、保存してエディターを終了します。

```
apiVersion: iot.enmasse.io/v1alpha1
kind: IoTConfig
metadata:
  namespace: enmasse-infra
```

```
  name: default
spec:
  adapters:
    mqtt:
      containers:
        adapter:
          logback:
            logback: | (1)
              <configuration>
                <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
                  <encoder>
                    <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} -
%msg%n</pattern>
                  </encoder>
                </appender>
                <root level="debug">
                  <appender-ref ref="STDOUT" />
                </root>
              </configuration>
```

(1) 完全な XML ベースの logback 設定。

3. Operator はロギング設定を適用し、必要なすべてのコンポーネントを再デプロイします。

3.2.7.3.2. サービスの ConfigMap リソースの使用

IoTConfig を使用してカスタム設定を提供するだけでなく、カスタムログ設定をサービスの **ConfigMap** ソースに配置することもできます。

手順

1. サービスの **ConfigMap** インスタンスを編集します。たとえば、HTTP プロトコルアダプターの場合は **iot-http-adapter-config** です。

```
oc edit cm iot-http-adapter-config
```

2. キー **logback-custom.xml** を使用して、データセクションに XML ベースの logback 設定を追加します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  namespace: enmasse-infra
  name: iot-http-adapter-config
data:
  application.yaml: … (1)
  logback-spring.xml: … (2)
  logback-custom.xml: | (3)
    <configuration>
      <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
          <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
      </appender>
```

```
  <root level="debug">
    <appender-ref ref="STDOUT" />
  </root>
</configuration>
```

1. アプリケーション固有の設定ファイル。Operator はこのファイルを生成し、変更を上書きします。
2. システムによって適用される有効な logback 設定。これは Operator によって上書きされるため、変更しないでください。
3. 完全な XML ベースの logback 設定。

3. Operator は **ConfigMap** リソースの変更を検出し、ロギング設定を適用して、必要なすべてのコンポーネントを再デプロイします。

3.2.8. トレースの設定

前提条件

- IoT サービスがインストールされている。
- OperatorHub の Jaeger Operator は AMQ Online namespace にインストールされます。
- Jaeger のインスタンスは、Operator を使用して AMQ Online namespace にデプロイされます。
- Jaeger インスタンスがサイドカーまたはデーモンセットエージェントを使用してデプロイされたかどうかを確認します。

手順

1. **default** という名前の **IoTConfig** インスタンスを編集します。

```
oc edit iotconfig default
```

2. Jaeger エージェントの設定に従って設定を変更します。

   a. Jaeger インスタンスがサイドカーエージェントでデプロイされている場合は、次の設定を追加します。

```
apiVersion: iot.enmasse.io/v1alpha1
kind: IoTConfig
metadata:
  namespace: amq-online-infra
  name: default
spec:
  tracing:
    strategy: (1)
      sidecar: {} (2)
```

1. ストラテジーを選択するフィールド。設定する必要があるストラテジーは 1 つだけです。

2. 「サイドカー」ストラテジーを有効にします。空のオブジェクト (**{}**) は意図的に使用しています。

b. Jaeger インスタンスが daemonset エージェントでデプロイされている場合は、次の設定を追加します。

```
apiVersion: iot.enmasse.io/v1alpha1
kind: IoTConfig
metadata:
  namespace: amq-online-infra
  name: default
spec:
  tracing:
    strategy: (1)
      daemonset: {} (2)
```

1. ストラテジーを選択するフィールド。設定する必要があるストラテジーは 1 つだけです。
2. daemonset ストラテジーを有効にします。空のオブジェクト (**{}**) は意図的に使用しています。

3. エディターを保存し、終了します。
4. Operator はトレース設定を適用し、必要なすべてのコンポーネントを再デプロイします。

3.2.9. IoT サービスの設定例

3.2.9.1. 最小限の IoT 設定の例

この IoT 設定の例では、**iotConfig** を作成するために必要なオプションのみを示しています。

```
kind: IoTConfig
apiVersion: iot.enmasse.io/v1alpha1
metadata:
  name: default
spec:
  services:
    deviceRegistry:
      infinispan:
        server:
          external: (1)
            host: infinispan
            port: 11222
            username: app
            password: test12
            saslServerName: hotrod
            saslRealm: ApplicationRealm
  adapters:
    mqtt:
      endpoint:
        secretNameStrategy:
          secretName: iot-mqtt-adapter-tls
```

-

1. Red Hat Data Grid サービスを提供する必要があります。

3.2.9.2. IoT プロトコルアダプターの例のチューニング

この IoT 設定の例は、プロトコルアダプターを個別に調整する方法を示しています。

```
kind: IoTConfig
apiVersion: iot.enmasse.io/v1alpha1
metadata:
  name: default
spec:
  services:
    deviceRegistry:
      infinispan:
        server:
          external:
            host: infinispan
            port: 11222
            username: app
            password: test12
            saslServerName: hotrod
            saslRealm: ApplicationRealm
  adapters:
    mqtt:
      enabled: true (1)
      replicas: 1
      options:
        tenantIdleTimeout: 30m (2)
        maxPayloadSize: 2048
    http:
      enabled: true
      replicas: 1 (3)
      options:
        tenantIdleTimeout: 30m
        maxPayloadSize: 2048 (4)
      containers:
        adapter:
          resources: (5)
            limits:
              memory: 128Mi
              cpu: 500m
    lorawan:
      enabled: false
    sigfox:
      enabled: false
```

1. 必要に応じて、プロトコルアダプターを無効にすることができます。デフォルト値は **true** です。
2. クライアント接続を維持する期間を指定します。
4. 受信メッセージの最大許容サイズをバイト単位で指定します。
3. 5. コンテナーリソースとインスタンスは、必要に応じて調整できます。

3.2.9.3. 外部 PostgreSQL を使用した JDBC の設定

```
kind: IoTConfig
apiVersion: iot.enmasse.io/v1alpha1
metadata:
  name: default
spec:
  services:
    deviceRegistry:
      jdbc:
        server:
          external:
            url: jdbc://postgresql.namespace.svc:5432/database-name (1)
            username: app (2)
            password: test12 (3)
```

1. PostgreSQL データベースへの JDBC URL。これには、ホスト名、ポート、およびデータベース名が含まれます。詳細は、https://jdbc.postgresql.org/documentation/head/connect.html も参照してください。
2. PostgreSQL サーバーへの接続に使用するユーザー名
3. PostgreSQL サーバーへの接続に使用するパスワード

3.2.10. IoT 固有の指標

AMQ Online の IoT 固有のコンポーネントは、このセクションで説明するメトリックを提供します。

3.2.10.1. 一般的なタグと指標

次のタグは、IoT 関連のすべてのコンポーネントで使用できます。

タグ	値	説明
host	**string**	メトリックを報告するコンポーネントが実行されているホストの名前を指定します。
component-type	**adapter**、**service**	メトリックを報告するコンポーネントのタイプを指定します。
component-name	**string**	メトリックを報告するコンポーネントの名前。コンポーネントのリストについては、次の表を参照してください。

表3.1 コンポーネント名

コンポーネント	component-name
HTTP プロトコルアダプター	**hono-http**
MQTT プロトコルアダプター	**hono-mqtt**
LoRaWAN プロトコルアダプター	**hono-lora**
Sigfox プロトコルアダプター	**hono-sigfox**

3.2.10.2. プロトコルアダプター

タイプ **adapter** のコンポーネントであるプロトコルアダプターには、いくつかの追加タグがあります。プロトコルアダプタータグ

名前	値	説明
direction	**one-way**、**request**、**response**	コマンド & コントロールメッセージが送信される方向を指定します。**one-way** は、アプリケーションに送信しても応答を受け取る必要のないデバイスに送信するコマンドを、**request** は、デバイスに送信されるコマンド要求メッセージを、**response** は、デバイスから受信したコマンド応答を示します。
qos	**0**、**1**、**unknown**	テレメトリーまたはイベントメッセージに使用されるサービスの品質を示します。**0** は **最大 1 回**、**1** は **少なくとも 1 回** を、**none** は配信セマンティクスが不明であることを示します。
status	**forwarded**、**unprocessable**、**undeliverable**	メッセージの処理状況を示します。**forwarded** は、メッセージがダウンストリームコンシューマーに転送されたことを示します。**unprocessable** は、メッセージの形式が正しくないなどの理由で、メッセージが処理または転送されなかったことを示します。**undeliverable** は、メッセージを転送できなかったことを示します。たとえば、ダウンストリームコンシューマーが存在しないか、インフラストラクチャーの問題が原因です。

名前	値	説明
tenant	string	メトリックが報告されているテナントの識別子を指定します。
ttd	**command**、**expired**、**none**	デバイスから受信したメッセージに含まれる TTD 値を処理した結果のステータスを示します。**command** は、メッセージをアップロードするためのデバイスの要求に対する応答に、デバイスのコマンドが含まれていることを、**expired** は、コマンドのない応答がデバイスに送信されたことを、**none** は、デバイスによって TTD 値が指定されていないか、プロトコルアダプターが TTD 値をサポートしていないことを示します。
type	**telemetry**、**event**	メトリックの (ダウンストリーム) メッセージのタイプを示します。

表3.2 プロトコルアダプターメトリック

メトリクス	タイプ	タグ	説明
hono.commands.received	Timer	host、component-type、component-name、tenant、type、status、direction	コマンドまたはコマンドへの応答を伝えるメッセージの処理にかかった時間を示します。
hono.commands.payload	DistributionSummary	host、component-type、component-name、tenant、type、status、direction	コマンドメッセージのペイロードで伝達されるバイト数を示します。
hono.connections.authenticated	ゲージ	host, component-type, component-name, tenant	接続済みかつ認証済みデバイスの現在の数。 注: このメトリックは、認証されたデバイスとの **接続状態** を維持するプロトコルアダプターによってのみサポートされます。特に、この HTTP アダプターはこのメトリックをサポートしていません。

メトリクス	タイプ	タグ	説明
hono.connections.unauthenticated	ゲージ	**host**, **component-type**, **component-name**	現在接続済みで、認証されていないデバイスの数。 注: このメトリックは、認証されたデバイスとの **接続状態** を維持するプロトコルアダプターによってのみサポートされます。特に、この HTTP アダプターはこのメトリックをサポートしていません。
hono.messages.received	Timer	**host**、**component-type**、**component-name**、**tenant**、**type**、**status**, **qos**、**ttd**	テレメトリーまたはイベントメッセージを伝達するメッセージの処理にかかった時間を示します。
hono.messages.payload	DistributionSummary	**host**, **component-type**, **component-name**, **tenant**, **type**, **status**	テレメトリーまたはイベントメッセージのペイロードで伝達されるバイト数を示します。

3.2.11. トラブルシューティングガイド

3.2.11.1. IoTProject が終了時に動かなくなる問題を修正

IoTProject インスタンスが削除されても、リソースはすぐには削除されません。削除のみがマークされ、必要なクリーンアップ操作はバックグラウンドで実行されます。クリーンアップが正常に実行されると、リソースは自動的に削除されます。

状況によっては、インフラストラクチャーの問題が原因で、この時点でクリーンアップ操作を実行できない場合があります。**IoTProject** は引き続き保持され、Operator は定期的に再試行してリソースをクリーンアップします。インフラストラクチャーが稼働状態に戻ると、クリーンアップは成功します。

インフラストラクチャーが二度と機能しないことが予想される場合は、**IoTProject** リソースを強制的に破棄することが望ましい場合があります。

警告

リソースクリーンアップファイナライザーを手動で削除すると、クリーンアッププロセスがスキップされ、システムが適切にクリーンアップされなくなります。

手順

1. **oc** ツールを使用して、プロジェクトが終了状態でスタックしているかどうかを評価します。

```
oc get iotproject iot -n myapp
NAME   IOT TENANT  DOWNSTREAM HOST                    DOWNSTREAM PORT   TLS
PHASE
iot    myapp.iot   messaging-be482a6.enmasse-infra.svc   5671            true   Terminating
```

出力には、終了中の状態のプロジェクトが表示されます。さらに、クリーンアップファイナライザーがまだ存在することを確認します。

```
oc get iotproject iot -n myapp -ojsonpath='{range .metadata.finalizers[*]}{..}{"\n"}{end}'
iot.enmasse.io/deviceRegistryCleanup
```

リストに **iot.enmasse.io/deviceRegistryCleanup** のエントリーが含まれている場合には、リソースのクリーンアッププロセスはまだ保留中です。

2. ファイナライザーのリストからファイナライザー **iot.enmasse.io/deviceRegistryCleanup** を手動で削除します。

```
oc edit iotproject iot -n myapp
```

これにより、リソースのコンテンツを含むテキストエディターが開きます。

```
apiVersion: iot.enmasse.io/v1alpha1
kind: IoTProject
metadata:
  creationTimestamp: "2019-12-09T15:00:00Z"
  deletionTimestamp: "2019-12-09T16:00:00Z"
  finalizers:
  - iot.enmasse.io/deviceRegistryCleanup (1)
  name: iot
  namespace: myapp
```

(1) 削除するファイナライザーのある行

ファイナライザーの行を削除します。エディターを保存し、終了します。これにより、サーバーで自動的に更新がトリガーされ、システムは引き続き **IoTProject** リソースを削除します。

3. ファイナライザーを削除すると、リソースは削除され、システムから表示されなくなっているはずです。

3.3. プロジェクトオーナー向けの IOT

3.3.1. IoT プロジェクトの設定例

IoT プロジェクトは、IoT テナントが消費できるメッセージングリソースを定義します。

3.3.1.1. マネージドメッセージングインフラストラクチャーの使用

この IoT プロジェクトの設定例は、AMQ Online に依存して、IoT トラフィックで使用されるメッセージングインフラストラクチャーを管理します。AMQ Online の **standard** アドレス空間とアドレス計画が使用されます。

```
kind: IoTProject
apiVersion: iot.enmasse.io/v1alpha1
metadata:
  name: user-1
spec:
  downstreamStrategy:
    managedStrategy: (1)
      addressSpace:
        name: iot-user-1
        plan: standard-unlimited (2)
        type: standard (3)
      addresses:
        telemetry:
          plan: standard-small-anycast (4)
          type: standard (5)
        event:
          plan: standard-small-queue (6)
        command:
          plan: standard-small-anycast (7)
```

1. **managedStrategy** 値は、AMQ Online によって管理されるメッセージングインフラストラクチャーを参照します。
2. アドレス空間のリソース使用量を定義して、アドレス空間計画を指定します。各 IoT テナントには、独自のアドレス空間が必要です。
4. 6. 7. 各アドレスは、有効なアドレス計画に関連付けられている必要があります。
3. アドレス空間のタイプを指定します。デフォルト値は **standard** です。

   詳しくは、アドレス空間の管理 を参照してください。
5. アドレスのタイプを指定します。

   詳しくは、アドレス空間の管理 を参照してください。

3.3.1.2. 外部メッセージングインフラストラクチャーの使用

この IoT 設定の例は、外部メッセージングインフラストラクチャーを設定する方法を示しています。

```
kind: IoTProject
apiVersion: iot.enmasse.io/v1alpha1
metadata:
  name: user-1
spec:
  downstreamStrategy:
    externalStrategy:
      host: messaging-hono-default.enmasse-infra.svc
      port: 5672
```

```
username: http
tls: true
password: http-secret
```

3.4. デバイスマネージャー向け IOT

通常、デバイスマネージャーは、システム内のデバイス ID と認証情報の作成と管理を担当します。

3.4.1. 認証トークンの取得

デバイス管理 API にアクセスするには、API に対して自分自身を認証するためのトークンを取得する必要があります。

IoT テナントのデバイスへのアクセスは、**IoTProject** リソースへのアクセス権に基づいて、デバイスレジストリーによりマップされます。アカウントに **IoTProject** への **読み取り** アクセス権がある場合に、このアカウントは、この IoT テナントのデバイスレジストリーに対して **読み取り** 操作を実行することもできます。

トークンは、HTTP ヘッダー値 **Authorization: Bearer <token>** を追加して、**ベアラートークン** として API に提示する必要があります。詳細は RFC 6750 を参照してください。

次の設定例では、**${TOKEN}** を実際のトークンに置き換えます。

3.4.1.1. ユーザーの認証トークンの取得

現在の OpenShift のトークンを使用する場合は、トークンを抽出できます。

前提条件

- トークンをサポートするユーザーとして OpenShift インスタンスにログインする必要があります。

手順

1. 現在のユーザーのトークンを抽出します。

```
oc whoami -t
```

注記

ユーザートークンの有効期間は限られているため、有効期限が切れた後にトークンを更新する必要がある場合があります。

3.4.1.2. サービスアカウントの認証トークンの取得

次の手順を実行して、新しいサービスアカウントを作成し、トークンを抽出します。

前提条件

- 新しいサービスアカウント、ロール、およびロールバインディングを作成するには、OpenShift インスタンスにログインする必要があります。

手順

1. 新規のサービスアカウントを作成します。

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-device-manager-account (1)
```

(1) サービスアカウントの名前

2. **IoTProject** へのアクセスを許可する新しいロールを作成します。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: device-manager-role (1)
rules: (2)
- apiGroups: ["iot.enmasse.io"]
  resources: ["iotprojects"]
  verbs: ["create", "update", "get", "list", "delete"]
```

(1) ロールの名前。

(2) **IoTProject** への CRUD アクセスを許可する必要があるアクセスルール。

この例では、namespace 内のすべての **IoTProjects** へのアクセスを許可します。アクセスをさらに制限するには、より具体的なルールを使用します。

3. 新しいロールバインディングを作成し、ロールをサービスアカウントに割り当てます。

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: my-device-manager-account-role-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: device-manager-role (1)
subjects:
- kind: ServiceAccount
  name: my-device-manager-account (2)
```

(1) ロールの名前。

(2) サービスアカウントの名前

4. サービスアカウントからトークンを取得します。

```
oc serviceaccounts get-token my-device-manager-account
```

3.4.2. IoT デバイスの作成

IoT サービスをインストールして IoT プロジェクトを作成したら、監視するデバイス用の IoT デバイスを作成できます。

前提条件

- IoT サービスをインストールしている。
- IoT プロジェクトを作成している。

3.4.2.1. 新しいデバイスの登録

新しいデバイスを作成するには、最初にデバイスを登録する必要があります。

手順

1. デバイスレジストリーホストをエクスポートします。

```
export REGISTRY_HOST=$(oc -n amq-online-infra get routes device-registry --template='{{
.spec.host }}')
```

2. デバイスレジストリーアクセストークンをエクスポートします。

```
export TOKEN=$(oc whoami --show-token)
```

このトークンは、デバイスレジストリー管理 API に対する認証に使用されます。

3. 定義された ID でデバイスを登録します (この例では **4711** を使用します)。

```
curl --insecure -X POST -i -H 'Content-Type: application/json' -H "Authorization: Bearer
${TOKEN}" https://$REGISTRY_HOST/v1/devices/myapp.iot/4711
```

4. (オプション) 追加の登録情報を指定する必要がある場合は、次のように行います。

```
curl --insecure -X POST -i -H 'Content-Type: application/json' -H "Authorization: Bearer
${TOKEN}" --data-binary '{
 "via": ["gateway1"]
}' https://$REGISTRY_HOST/v1/devices/myapp.iot/4711
```

3.4.2.2. デバイスのユーザー名とパスワード認証情報の設定

新しいデバイスを登録したら、デバイスのユーザー名とパスワード認証情報を設定する必要があります。

手順

1. デバイスの認証情報を追加します。

```
curl --insecure -X PUT -i -H 'Content-Type: application/json' -H "Authorization: Bearer
${TOKEN}" --data-binary '[{
 "type": "hashed-password",
 "auth-id": "sensor1",
 "secrets": [{
```

```
  "pwd-plain":"'hono-secret'"
 }]
}]' https://$REGISTRY_HOST/v1/credentials/myapp.iot/4711
```

3.5. ソリューション開発者向け IOT

通常、IoT ソリューションの開発者は、IoT クラウドアプリケーションの作成を担当します。

3.5.1. Eclipse Hono コマンドラインクライアントのインストール

前提条件

- IoT サービスをインストールしている。
- IoT プロジェクトを作成している。
- IoT デバイスを作成している。

手順

1. Eclipse Hono コマンドラインクライアント をダウンロードします。
2. メッセージングエンドポイント証明書を取得します。

```
oc -n myapp get addressspace iot -o jsonpath={.status.caCert} | base64 --decode > tls.crt
```

3. メッセージングエンドポイントのホストとポートをエクスポートします。

```
export MESSAGING_HOST=$(oc get -n myapp addressspace iot -o
jsonpath='{.status.endpointStatuses[?(@.name=="messaging")].externalHost}')
export MESSAGING_PORT=443
```

3.5.2. telemetry コンシューマーの開始

たとえば、センサーの読み取り値などの telemetry データをデバイスからクラウドに送信できます。これを行うには、最初に顧客アプリケーションを実行して telemetry コンシューマーを開始する必要があります。

手順

1. 顧客アプリケーションを実行して、telemetry を受信します。

```
java -jar hono-cli-*-exec.jar --hono.client.host=$MESSAGING_HOST --
hono.client.port=$MESSAGING_PORT --hono.client.username=consumer --
hono.client.password=foobar --tenant.id=myapp.iot --hono.client.trustStorePath=tls.crt --
message.type=telemetry
```

3.5.3. イベントコンシューマーの開始

アラートやその他の重要なデータなどのイベントをデバイスからコンシューマーアプリケーションに送信できます。これを行うには、最初に顧客アプリケーションを実行してイベントコンシューマーを開始する必要があります。

手順

1. 顧客アプリケーションを実行してイベントを受信します。

```
java -jar hono-cli-*-exec.jar --hono.client.host=$MESSAGING_HOST --
hono.client.port=$MESSAGING_PORT --hono.client.username=consumer --
hono.client.password=foobar --tenant.id=myapp.iot --hono.client.trustStorePath=tls.crt --
message.type=events
```

3.5.4. コマンド送信者の開始

クラウドからデバイスにコマンドを送信できます。これを行うには、顧客アプリケーションを実行してコマンド送信者を開始する必要があります。

手順

1. カスタマーアプリケーションを実行して、ID が **4711** のデバイスにコマンドを送信します。

```
java -jar hono-cli-*-exec.jar --hono.client.host=$MESSAGING_HOST --
hono.client.port=$MESSAGING_PORT --hono.client.username=consumer --
hono.client.password=foobar --tenant.id=myapp.iot --hono.client.trustStorePath=tls.crt --
device.id=4711 --spring.profiles.active=command
```

2. コマンドの名前、ペイロード、およびコンテンツタイプを入力するための指示に従います。以下に例を示します。

```
>>>>>>>>> Enter name of command for device [4711] in tenant [myapp.iot] (prefix with 'ow:'
to send one-way command):
ow:setVolume
>>>>>>>>> Enter command payload:
{"level": 50}
>>>>>>>>> Enter content type:
application/json

INFO  org.eclipse.hono.cli.app.Commander - Command sent to device
```

3.6. デバイス開発者向け IOT

通常、デバイス開発者は、既存のデバイスをクラウドプラットフォームに接続するか、デバイス用のソフトウェア作成を担当します。以下の情報において、サポートされているプロトコルのいずれかを使用してデバイスを接続する方法について説明します。

3.6.1. HTTP デバイス

3.6.1.1. HTTP を使用した telemetry の送信

HTTP プロトコルを使用して、デバイスからクラウドに telemetry を送信できます。

手順

1. HTTP プロトコルを使用して telemetry を送信します。

```
curl --insecure -X POST -i -u sensor1@myapp.iot:hono-secret -H 'Content-Type:
application/json' --data-binary '{"temp": 5}' https://$(oc -n amq-online-infra get routes iot-http-
adapter --template='{{ .spec.host }}')/telemetry
```

3.6.1.2. HTTP を使用したイベントの送信

HTTP プロトコルを使用して、顧客アプリケーションからデバイスにイベントメッセージを送信できます。

手順

1. HTTP プロトコルを使用してイベントを送信します。

```
curl --insecure -X POST -i -u sensor1@myapp.iot:hono-secret -H 'Content-Type:
application/json' --data-binary '{"temp": 5}' https://$(oc -n amq-online-infra get routes iot-http-
adapter --template='{{ .spec.host }}')/events
```

3.6.1.3. HTTP プロトコルを使用したコマンドの受信

HTTP プロトコルを使用して、クラウドからデバイスにコマンドを送信できます。

手順

1. クライアントがコマンドを待機する時間を示す **hono-ttd** パラメーターを指定し、HTTP プロトコルを使用して Telemetry メッセージを送信します。

```
curl --insecure -X POST -i -u sensor1@myapp.iot:hono-secret -H 'Content-Type:
application/json' --data-binary '{"temp": 5}' https://$(oc -n amq-online-infra get routes iot-http-
adapter --template='{{ .spec.host }}')/telemetry?hono-ttd=600
```

2. カスタマーアプリケーションを実行して、ID が **4711** のデバイスにコマンドを送信します。

```
java -jar hono-cli-*-exec.jar --hono.client.host=$MESSAGING_HOST --
hono.client.port=$MESSAGING_PORT --hono.client.username=consumer --
hono.client.password=foobar --tenant.id=myapp.iot --hono.client.trustStorePath=tls.crt --
device.id=4711 --spring.profiles.active=command
```

3. コマンドの名前、ペイロード、およびコンテンツタイプを入力するための指示に従います。以下に例を示します。

```
>>>>>>>>> Enter name of command for device [4711] in tenant [myapp.iot] (prefix with 'ow:'
to send one-way command):
ow:setVolume
>>>>>>>>> Enter command payload:
{"level": 50}
>>>>>>>>> Enter content type:
application/json

INFO  org.eclipse.hono.cli.app.Commander - Command sent to device
```

クライアントは HTTP 応答でコマンドを受け取ります。

```
HTTP/1.1 200 OK
```

```
hono-command: setVolume
content-type: application/json
content-length: 13

{"level": 50}
```

3.6.2. MQTT デバイス

3.6.2.1. MQTT を使用したテレメトリーの送信

MQTT プロトコルを使用して、デバイスからクラウドにテレメトリーを送信できます。

手順

1. MQTT プロトコルを使用してテレメトリーを送信します。

```
mosquitto_pub -d -h $(oc -n amq-online-infra get routes iot-mqtt-adapter --template='{{
.spec.host }}') -p 443 -u 'sensor1@myapp.iot' -P hono-secret -t telemetry -m '{"temp": 5}' -i
4711 --cafile install/components/iot/examples/k8s-tls/build/iot-mqtt-adapter-fullchain.pem
```

3.6.2.2. MQTT を使用したイベントの送信

MQTT プロトコルを使用して、顧客アプリケーションからデバイスにイベントメッセージを送信できます。

手順

1. MQTT プロトコルを使用してイベントを送信します。

```
mosquitto_pub -d -h $(oc -n amq-online-infra get routes iot-mqtt-adapter --template='{{
.spec.host }}') -p 443 -u 'sensor1@myapp.iot' -P hono-secret -t events -m '{"temp": 5}' -i 4711
--cafile install/components/iot/examples/k8s-tls/build/iot-mqtt-adapter-fullchain.pem
```

3.6.2.3. MQTT プロトコルを使用したコマンドの受信

MQTT プロトコルを使用して、クラウドからデバイスにコマンドを送信できます。

手順

1. MQTT クライアントを使用して、コマンドを受信するために MQTT トピックにサブスクライブします。

```
mosquitto_sub -v -d -h $(oc -n amq-online-infra get routes iot-mqtt-adapter --template='{{
.spec.host }}') -p 443 -u 'sensor1@myapp.iot' -P hono-secret -t command/+/+/req/# -i 4711 --
cafile install/components/iot/examples/k8s-tls/build/iot-mqtt-adapter-fullchain.pem
```

2. カスタマーアプリケーションを実行して、ID が **4711** のデバイスにコマンドを送信します。

```
java -jar hono-cli-*-exec.jar --hono.client.host=$MESSAGING_HOST --
hono.client.port=$MESSAGING_PORT --hono.client.username=consumer --
hono.client.password=foobar --tenant.id=myapp.iot --hono.client.trustStorePath=tls.crt --
device.id=4711 --spring.profiles.active=command
```

3. コマンドの名前、ペイロード、およびコンテンツタイプを入力するための指示に従います。以下に例を示します。

```
>>>>>>>>> Enter name of command for device [4711] in tenant [myapp.iot] (prefix with 'ow:'
to send one-way command):
ow:setVolume
>>>>>>>>> Enter command payload:
{"level": 50}
>>>>>>>>> Enter content type:
application/json

INFO  org.eclipse.hono.cli.app.Commander - Command sent to device
```

クライアントは MQTT メッセージでコマンドを受け取ります。

```
Client 4711 received PUBLISH (d0, q0, r0, m0, 'command///req//setVolume', ... (13 bytes))
command///req//setVolume {"level": 50}
```

3.6.3. Sigfox デバイスの設定

IoT サービスをインストールして IoT プロジェクトを作成したら、Sigfox バックエンド統合を設定できます。

前提条件

- IoT サービスをインストールしている。
- IoT プロジェクトを作成している。
- TLS がクラスターに適切にデプロイされている。Sigfox プロトコルアダプターエンドポイントは、TLS で適切に検証しておく。
- https://backend.sigfox.com でアカウントを設定し、Sigfox デバイスを登録しておく。
- Sigfox バックエンドに精通している。Sigfox バックエンドにログインした後の詳細は Sigfox のドキュメント を参照してください。

3.6.3.1. Sigfox バックエンドのゲートウェイデバイスとしての登録

前提条件

- Sigfox バックエンドを、AMQ Online でゲートウェイデバイスとして設定している。
- このデバイスに割り当てられた認証情報は、Sigfox バックエンドでコールバック設定に必要である。
- 実際のデバイスが、このゲートウェイデバイスをトランスポートとして使用するように設定されている。

手順

1. 新しいデバイスを登録します。
   この手順のステップ 3 で、**sigfox-backend** を ID として指定します。

2. このデバイスの パスワード認証情報を設定します (例: **sigfox-user/sigfox-password**)。

3.6.3.2. Sigfox デバイスの登録

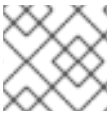
前提条件

- Sigfox バックエンドをゲートウェイデバイスとして登録している。

手順

1. 登録する Sigfox デバイスの **デバイス ID** を見つけます。この ID は、Sigfox バックエンドでの登録プロセスの一部として取得します。

2. 新しいデバイスを登録します。
   デバイス ID を名前として指定し (例: **1AB2C3**)、登録情報の一部として **via** フィールドにゲートウェイデバイスの名前を指定します (例 **: {"via": "sigfox-backend"}**)。



注記

このデバイスにはパスワードを設定しないでください。

3.6.3.3. Sigfox 接続情報の準備

Sigfox バックエンド手順での新しいコールバックの作成 で使用される次の接続情報を準備します。

IoT テナント名

IoT テナントの名前は、OpenShift namespace と IoT プロジェクトリソースの名前 (例: **namespace.iotproject**) で設定されます。

HTTP 認証ヘッダー

Sigfox バックエンドの認証には、ゲートウェイデバイスのユーザー名とパスワードの組み合わせを HTTP の基本認証ヘッダーに変換する必要があります。**authentication id @ IoT tenant name** という構文を使用して、完全なユーザー名を指定してください。
例: **sigfox-user@namespace.iotproject**

基本認証ヘッダー値は、次のコマンドを使用してコマンドラインで生成できます。

```
echo "Basic $(echo -n "sigfox-user@namespace.iotproject:password" | base64)"
```

URL パターン

URL パターンは、Sigfox プロトコルアダプターへの URL と Sigfox 固有のクエリーパラメーターで設定されます。

```
https://<ADAPTER URL>/data/telemetry/<TENANT>?device={device}&data={data}
```

次のコマンドを実行して、プロトコルアダプターの URL を取得します。

```
echo "https://$(oc -n amq-online-infra get routes iot-sigfox-adapter --template='{{ .spec.host }}')"
```

パスセグメント **/data/telemetry** は、メッセージを **telemetry** データとして処理することをプロトコルアダプターに示します。代わりに **/data/event** を使用して、メッセージを **イベント** として処理できます。

注記

{device} と **{data}** はリテラル値ですので、置き換えないようにしてください。

3.6.3.4. Sigfox バックエンドでの新規コールバックの作成

前提条件

- Sigfox 接続情報を作成している。

手順

1. https://backend.sigfox.com にログインします。
2. **Device Type** でタイプを開いて編集し、**Callbacks** セクションに切り替えます。
3. 次の設定で、新しいカスタムコールバックを作成します。

   タイプ

   DATA – UPLINK

   チャネル

   URL

   URL パターン

   URL パターン。例: **https://iot-sigfox-adapter.my.cluster/data/telemetry/<TENANT>?device={device}&data={data}**

   HTTP メソッドの使用

   GET

   ヘッダー

   Authorization – Basic...

   SNI の送信

   ☑ (有効)

3.6.3.5. Sigfox でのコマンドアンドコントロールの有効化

前提条件

- Sigfox バックエンドでコールバックを作成している。

手順

1. https://backend.sigfox.com にログインします。
2. **Device Type** で、タイプを開いて編集し、**Callbacks** セクションに切り替えます。
3. コマンドアンドコントロールを有効にするコールバック設定を編集します。

   タイプ

   DATA – BIDIR への切り替え

   URL パターン

ack パラメーターを追加します。例: **https://iot-sigfox-adapter.my.cluster/data/telemetry/<TENANT>?device={device}&data={data} &ack={ack}**

第4章 AMQ ONLINE のインストール

AMQ Online のインストールに使用したのと同じ方法を使用して、AMQ Online をアンインストールする必要があります。

4.1. YAML バンドルを使用した AMQ ONLINE のアンインストール

このメソッドは、YAML バンドルを使用してインストールされた AMQ Online をアンインストールします。

手順

1. **cluster-admin** 権限を持つユーザーとしてログインしてます。

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

2. クラスターレベルのリソースを削除します。

```
oc delete crd -l app=enmasse,enmasse-component=iot
oc delete crd -l app=enmasse --timeout=600s
oc delete clusterrolebindings -l app=enmasse
oc delete clusterroles -l app=enmasse
oc delete apiservices -l app=enmasse
oc delete oauthclients -l app=enmasse
```

3. (OpenShift 4) コンソール統合を削除します。

```
oc delete consolelinks -l app=enmasse
```

4. (オプション) サービスカタログ統合を削除します。

```
oc delete clusterservicebrokers -l app=enmasse
```

5. AMQ Online がデプロイされているプロジェクトを削除します。

```
oc delete project amq-online-infra
```

4.2. OPENSHIFT CONTAINER PLATFORM 4.X コンソールを使用した AMQ ONLINE OPERATOR のアンインストール

OpenShift Container Platform コンソールで、OpenShift Container Platform 4.1 クラスターの AMQ Online Operator をアンインストールできます。

前提条件

- AMQ Online Operator を OpenShift Container Platform 4.1 クラスターにインストールしておく。

手順

1. プロジェクトリストから、**openshift-operators** プロジェクトを選択します。

2. **Catalog → Operator Management** をクリックします。Operator 管理ページが開きます。

3. **Operator Subscriptions** タブをクリックします。

4. アンインストールする AMQ Online Operator を見つけます。右端の列で、縦の省略記号アイコンをクリックし、**Remove Subscription** を選択します。

5. Remove Subscription ウィンドウでプロンプトが表示されたら、Operator に関連するすべてのコンポーネントを削除する場合は、**Also completely remove the AMQ Online Operator from the selected namespace** チェックボックスを選択してください。

6. **Remove** をクリックします。AMQ Online Operator は実行を停止し、更新を受信しなくなります。

次のステップ

- 残りのすべてのリソースを完全に削除するには Operator Lifecycle Manager を使用した AMQ Online のアンインストール後の残りのリソース削除 を参照してください。

4.2.1. Operator Lifecycle Manager を使用した AMQ Online のアンインストール後の残りのリソース削除

ENTMQMAAS-1281 により、Operator Lifecycle Manager を使用して AMQ Online をアンインストールすると、一部のリソースが残ります。この手順により、残りのリソースが削除され、AMQ Online が完全にアンインストールされます。

前提条件

- OpenShift Container Platform 4.x コンソールを使用して AMQ Online Operator をアンインストールしている

手順

1. コマンドラインで、**openshift-operators** プロジェクトにコマンドを実行する権限を持つユーザーとしてログインします。

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

2. **openshift-operators** プロジェクトに移動します。

```
oc project openshift-operators
```

3. 次のコマンドを実行して、残りのリソースを削除します。

```
oc delete all -l app=enmasse
oc delete crd -l app=enmasse
oc delete apiservices -l app=enmasse
oc delete cm -l app=enmasse
oc delete secret -l app=enmasse
```

付録A サブスクリプションの使用

AMQ Online は、ソフトウェアサブスクリプションを通じて提供されます。サブスクリプションを管理するには、Red Hat カスタマーポータルでアカウントにアクセスします。

アカウントへのアクセス

1. access.redhat.com に移動します。
2. アカウントがない場合は、作成します。
3. アカウントにログインします。

サブスクリプションのアクティベート

1. access.redhat.com に移動します。
2. **サブスクリプション** に移動します。
3. **Activate a subscription** に移動し、16 桁のアクティベーション番号を入力します。

zip および tar ファイルのダウンロード

zip または tar ファイルにアクセスするには、Red Hat カスタマーポータルを使用して、ダウンロードする関連ファイルを検索します。RPM パッケージを使用している場合は、この手順は必要ありません。

1. ブラウザーを開き、access.redhat.com/downloads で Red Hat カスタマーポータルの **Product Downloads** ページにログインします。
2. **JBOSS INTEGRATION AND AUTOMATION** カテゴリーの **Red Hat AMQ Online** エントリーを見つけます。
3. 目的の AMQ Online 製品を選択します。Software Downloads ページが開きます。
4. コンポーネントの **ダウンロード** リンクをクリックします。

パッケージ用のシステムの登録

RPM パッケージを Red Hat Enterprise Linux にインストールするには、システムが登録されている必要があります。zip または tar ファイルを使用している場合、この手順は必要ありません。

1. access.redhat.com に移動します。
2. **Registration Assistant** に移動します。
3. ご使用の OS バージョンを選択し、次のページに進みます。
4. システムターミナルで listed コマンドを使用して、登録を完了します。

詳細は、How to Register and Subscribe a System to the Red Hat Customer Portal を参照してください。

改訂日時：2023-01-28 12:18:03 +1000