



Red Hat build of Apicurio Registry 2.3

OpenShift への Apicurio Registry のインストール とデプロイ

Apicurio Registry 2.3 のインストールとデプロイ

Red Hat build of Apicurio Registry 2.3 OpenShift への Apicurio Registry のインストールとデプロイ

Apicurio Registry 2.3 のインストールとデプロイ

法律上の通知

Copyright © 2023 Red Hat, Inc.

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

本ガイドでは、AMQ Streams または PostgreSQL データベースでレジストリーデータストレージオプションを使用して、OpenShift に Apicurio Registry をインストールおよびデプロイする方法を説明します。以下では、Apicurio Registry の保護、設定、および管理する方法について説明します。また、Apicurio Registry Operator についての参照情報を提供します。

目次

はじめに	3
多様性を受け入れるオープンソースの強化	3
第1章 SERVICE REGISTRY OPERATOR クイックスタート	4
1.1. QUICKSTART SERVICE REGISTRY OPERATOR のインストール	4
1.2. QUICKSTART APICURIO REGISTRY インスタンスのデプロイメント	5
第2章 OPENSIFT に APICURIO REGISTRY をインストールする	7
2.1. OPENSIFT OPERATORHUB からの APICURIO REGISTRY のインストール	7
第3章 AMQ STREAMS に APICURIO REGISTRY ストレージをデプロイする	9
3.1. OPENSIFT OPERATORHUB からの AMQ STREAMS のインストール:	9
3.2. OPENSIFT で KAFKA ストレージを使用して APICURIO REGISTRY を設定する	10
3.3. TLS セキュリティーでの KAFKA ストレージの設定	12
3.4. SCRAM セキュリティーでの KAFKA ストレージの設定	16
3.5. KAFKA ストレージの OAUTH 認証の設定	20
第4章 POSTGRESQL データベースに APICURIO REGISTRY ストレージをデプロイする	21
4.1. OPENSIFT OPERATORHUB からの POSTGRESQL データベースのインストール	21
4.2. OPENSIFT での POSTGRESQL データベースストレージを使用した APICURIO REGISTRY の設定	22
4.3. APICURIO REGISTRY POSTGRESQL ストレージのバックアップ	23
4.4. APICURIO REGISTRY POSTGRESQL ストレージの復元	24
第5章 APICURIO REGISTRY デプロイメントの保護	26
5.1. RED HAT SINGLE SIGN-ON OPERATOR を使用した APICURIO REGISTRY の保護	26
5.2. RED HAT SINGLE SIGN-ON を使用した APICURIO REGISTRY の認証と承認の設定	30
5.3. APICURIO REGISTRY の認証および承認の設定オプション	33
5.4. OPENSIFT クラスター内から APICURIO REGISTRY への HTTPS 接続の設定	38
5.5. OPENSIFT クラスター外から APICURIO REGISTRY への HTTPS 接続の設定	41
第6章 APICURIO REGISTRY デプロイメントの設定と管理	43
6.1. OPENSIFT での APICURIO REGISTRY ヘルスチェックの設定	43
6.2. APICURIO REGISTRY ヘルスチェックの環境変数	44
6.3. APICURIO REGISTRY 環境変数の管理	46
6.4. APICURIO REGISTRY WEB コンソールを設定する	47
6.5. APICURIO REGISTRY ログの設定	48
6.6. APICURIO REGISTRY イベントソーシングの設定	48
第7章 SERVICE REGISTRY OPERATOR の設定リファレンス	51
7.1. APICURIO REGISTRY カスタムリソース	51
7.2. APICURIO REGISTRY CR スペック	52
7.3. APICURIO レジストリーの CR ステータス	55
7.4. APICURIO REGISTRY が管理するリソース	56
7.5. SERVICE REGISTRY OPERATOR ラベル	57
付録A サブスクリプションの使用	58
アカウントへのアクセス	58
サブスクリプションのアクティベート	58
ZIP および TAR ファイルのダウンロード	58
パッケージ用のシステムの登録	58

はじめに

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 SERVICE REGISTRY OPERATOR クイックスタート

カスタムリソース定義 (CRD) を使用して、コマンドラインで Service Registry Operator を迅速にインストールできます。

クイックスタートの例では、SQL データベースにストレージを持つ Apicurio Registry インスタンスをデプロイします。

- [「Quickstart Service Registry Operator のインストール」](#)
- [「Quickstart Apicurio Registry インスタンスのデプロイメント」](#)



注記

実稼働環境で推奨されるインストールオプションは OpenShift OperatorHub です。推奨されるストレージオプションは、パフォーマンス、安定性、およびデータ管理のための SQL データベースです。

1.1. QUICKSTART SERVICE REGISTRY OPERATOR のインストール

Service Registry Operator は、ダウンロードしたインストールファイルとサンプル CRD のセットを使用して、Operator Lifecycle Manager を使用せずにコマンドラインで迅速にインストールおよびデプロイできます。

前提条件

- アクセス権を持つ管理者として OpenShift クラスターにログインしている。
- OpenShift **oc** コマンドラインクライアントがインストールされている。詳細は、[OpenShift CLI のドキュメント](#) を参照してください。

手順

1. [Red Hat Software Downloads](#) に移動し、製品バージョンを選択して、Apicurio Registry CRD **.zip** ファイル内のサンプルをダウンロードします。
2. ダウンロードした CRD **.zip** ファイルを展開して、**apicurio-registry-install-examples** ディレクトリーに移動します。
3. Service Registry Operator インストールの OpenShift プロジェクトを作成します。以下に例を示します。

```
export NAMESPACE="apicurio-registry"
oc new-project "$NAMESPACE"
```

4. 以下のコマンドを実行して、**install/install.yaml** ファイルにサンプル CRD を適用します。

```
cat install/install.yaml | sed "s/apicurio-registry-operator-namespace/$NAMESPACE/g" | oc apply -f -
```

5. **oc get deployment** と入力し、Service Registry Operator の readiness (準備状態) を確認します。たとえば、出力は以下のようになります。

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
apicurio-registry-operator	1/1	1	1	XmYs

1.2. QUICKSTART APICURIO REGISTRY インスタンスのデプロイメント

Apicurio Registry インスタンスのデプロイメントを作成するには、SQL データベースストレージオプションを使用して、既存の PostgreSQL データベースに接続します。

前提条件

- Service Registry Operator がインストールされていることを確認している。
- OpenShift クラスターからアクセス可能な PostgreSQL データベースがある。

手順

1. エディターで **examples/apicurioregistry_sql_cr.yaml** ファイルを開き、**ApicurioRegistry** カスタムリソース (CR) を表示します。

SQL ストレージの CR の例

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-sql
spec:
  configuration:
    persistence: "sql"
    sql:
      dataSource:
        url: "jdbc:postgresql://<service name>.<namespace>.svc:5432/<database name>"
        userName: "postgres"
        password: "<password>" # Optional
```

2. **dataSource** セクションで、設定例をデータベース接続の詳細に置き換えます。以下に例を示します。

```
dataSource:
  url: "jdbc:postgresql://postgresql.apicurio-registry.svc:5432/registry"
  userName: "pgadmin"
  password: "pgpass"
```

3. 次のコマンドを入力して、Apicurio Registry Operator を使用して名前空間に更新された **ApicurioRegistry** CR を適用し、Apicurio Registry インスタンスがデプロイされるのを待ちます。

```
oc project "$NAMESPACE"
oc apply -f ./examples/apicurioregistry_sql_cr.yaml
```

4. **oc getdeployment** と入力して、Apicurio Registry インスタンスの準備ができているかを確認します。たとえば、出力は以下のようになります。

NAME	READY	UP-TO-DATE	AVAILABLE	AGE
example-apicurioregistry-sql-deployment	1/1	1	1	XmYs

5. **oc get routes** と入力して **HOST/PORT** URL を取得し、ブラウザで Apicurio Registry Web コンソールを起動します。以下に例を示します。

```
example-apicurioregistry-sql.apicurio-registry.router-  
default.apps.mycluster.myorg.mycompany.com
```

第2章 OPENSIFT に APICURIO REGISTRY をインストールする

本章では、OpenShift Container Platform に Apicurio Registry をインストールする方法を説明します。

- [「OpenShift OperatorHub からの Apicurio Registry のインストール」](#)

前提条件

- [Apicurio Registry User Guide の Red Hat ビルドの概要](#) を読む

2.1. OPENSIFT OPERATORHUB からの APICURIO REGISTRY のインストール

OperatorHub から OpenShift クラスターに Apicurio Registry Operator をインストールできます。OperatorHub は OpenShift Container Platform Web コンソールから使用でき、クラスター管理者が Operator を検出およびインストールするためのインターフェイスを提供します。詳細については、[OperatorHub について](#) を参照してください。



注記

環境に応じて、複数の Apicurio Registry インスタンスをインストールできます。インスタンス数は、Apicurio Registry に保存されているアーティファクトの数および種類と、選択したストレージオプションによって異なります。

前提条件

- クラスター管理者として OpenShift クラスターにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. 新しい OpenShift プロジェクトを作成します。
 - a. 左側のナビゲーションメニューで、**Home**、**Project**、**Create Project** と順にクリックします。
 - b. プロジェクト名 (**my-project** など) を入力し、**Create** をクリックします。
3. 左側のナビゲーションメニューで、**Operators** をクリックした後、**OperatorHub** をクリックします。
4. **Filter by keyword** テキストボックスに **registry** を入力し、**Red Hat Integration - Service Registry Operator** を見つけます。
5. Operator に関する情報を読み、**Install** をクリックして Operator サブスクリプションページを表示します。
6. サブスクリプション設定を選択します。以下に例を示します。
 - **Update Channel:** 以下のいずれかを選択します。
 - **2.x:** 2.3.0 や 2.0.3 などのすべてのマイナーおよびパッチの更新が含まれます。たとえば、2.0.x へのインストールは 2.3.x にアップグレードされます。

- **2.0.x:** 2.0.1 や 2.0.2 などのパッチの更新のみが含まれます。たとえば、2.0.x へのインストールは 2.3.x を無視します。
 - **Installation Mode:** 以下のいずれかを選択します。
 - All namespaces on the cluster (default)
 - A specific namespace on the cluster および my-project
 - **Approval Strategy:** Automatic または Manual を選択します。
7. **Install** をクリックし、Operator が使用できるようになるまでしばらく待ちます。

関連情報

- [Operator の OpenShift クラスターへの追加](#)
- [GitHub の Apicurio Registry Operator コミュニティー](#)

第3章 AMQ STREAMS に APICURIO REGISTRY ストレージをデプロイする

この章では、AMQ Streams に Apicurio Registry データストレージをインストールして設定する方法について説明します。

- [「OpenShift OperatorHub からの AMQ Streams のインストール:」](#)
- [「OpenShift で Kafka ストレージを使用して Apicurio Registry を設定する」](#)
- [「TLS セキュリティーでの Kafka ストレージの設定」](#)
- [「SCRAM セキュリティーでの Kafka ストレージの設定」](#)
- [「Kafka ストレージの OAuth 認証の設定」](#)

前提条件

- [2章OpenShift に Apicurio Registry をインストールする](#)

3.1. OPENSIFT OPERATORHUB からの AMQ STREAMS のインストール:

AMQ Streams がインストールされていない場合は、OperatorHub から OpenShift クラスターに AMQ Streams Operator をインストールできます。OperatorHub は OpenShift Container Platform Web コンソールから使用でき、クラスター管理者が Operator を検出およびインストールするためのインターフェイスを提供します。詳細については、[OperatorHub について](#) を参照してください。

前提条件

- クラスター管理者として OpenShift クラスターにアクセスできる必要があります。
- AMQ Streams のインストールの詳細については、[OpenShift での AMQ Streams のデプロイとアップグレード](#) を参照してください。ここでは、OpenShift OperatorHub を使用したインストールの簡単な例を示します。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. AMQ Streams をインストールする OpenShift プロジェクトに切り替えます。たとえば、**Project** ドロップダウンから、**my-project** を選択します。
3. 左側のナビゲーションメニューで、**Operators** をクリックした後、**OperatorHub** をクリックします。
4. **Filter by keyword** テキストボックスに **AMQ Streams** を入力し、**Red Hat Integration - AMQ Streams** を見つけます。
5. Operator に関する情報を読み、**Install** をクリックして Operator サブスクリプションページを表示します。
6. サブスクリプション設定を選択します。以下に例を示します。
 - **Update Channel** と **amq-streams-2.3.x**

- **Installation Mode:** 以下のいずれかを選択します。
 - All namespaces on the cluster (default)
 - A specific namespace on the cluster> my-project
- **Approval Strategy:** Automatic または Manual を選択します。

7. **Install** をクリックし、Operator が使用できるようになるまでしばらく待ちます。

関連情報

- [Operator の OpenShift クラスターへの追加](#)
- [AMQ Streams on OpenShift のデプロイおよびアップグレード](#)

3.2. OPENSIFT で KAFKA ストレージを使用して APICURIO REGISTRY を設定する

このセクションでは、OpenShift で AMQ Streams を使用して Apicurio Registry 用に Kafka ベースのストレージを設定する方法について説明します。**kafkasql** ストレージオプションは、インメモリー H2 データベースを備えた Kafka ストレージを使用します。このストレージオプションは、OpenShift の Kafka クラスターに **persistent** ストレージが設定されている実稼働環境に適しています。

既存の Kafka クラスターに Apicurio Registry をインストールするか、環境に応じて新しい Kafka クラスターを作成できます。

前提条件

- クラスター管理者として OpenShift クラスターにアクセスできる。
- Apicurio Registry がすでにインストールされている必要があります。[2章OpenShift に Apicurio Registry をインストールする](#)を参照してください。
- AMQ Streams がすでにインストールされている。「[OpenShift OperatorHub からの AMQ Streams のインストール](#)」を参照してください。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. Kafka クラスターがまだ設定されていない場合は、AMQ Streams を使用して新しい Kafka クラスターを作成します。たとえば、OpenShift OperatorHub では以下を実行します。
 - a. **Installed Operators** をクリックしてから **Red Hat Integration - AMQ Streams** をクリックします。
 - b. **Provided APIs**、**Kafka** と選択し、**Create Instance** をクリックして新しい Kafka クラスターを作成します。
 - c. 適切にカスタムリソース定義を編集し、**Create** をクリックします。



警告

デフォルトの例では、3 つの Zookeeper ノード、および、**ephemeral** ストレージを持つ 3 つの Kafka ノードを持つクラスターが作成されます。この一時ストレージは開発およびテストにのみ適しており、実稼働には適していません。詳細については、[OpenShift での AMQ ストリームのデプロイとアップグレード](#) を参照してください。

3. クラスターの準備ができたなら、**Provided APIs > Kafka > my-cluster > YAML** をクリックします。
4. **status** ブロックで、**bootstrapServers** 値のコピーを作成します。これは、後で Apicurio Registry をデプロイするために使用します。以下に例を示します。

```
status:
  ...
  conditions:
  ...
  listeners:
    - addresses:
      - host: my-cluster-kafka-bootstrap.my-project.svc
        port: 9092
      bootstrapServers: 'my-cluster-kafka-bootstrap.my-project.svc:9092'
      type: plain
  ...
```

5. **Installed Operators > Red Hat Integration - Service Registry > ApicurioRegistry > Create ApicurioRegistry** をクリックします。
6. 次のカスタムリソース定義を貼り付けますが、前にコピーした **bootstrapServers** 値を使用します。

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql
spec:
  configuration:
    persistence: 'kafkasql'
    kafkasql:
      bootstrapServers: 'my-cluster-kafka-bootstrap.my-project.svc:9092'
```

7. **Create** をクリックし、Apicurio Registry ルートが OpenShift に作成されるのを待ちます。
8. **Networking > Route** をクリックして、Apicurio Registry Web コンソールの新しいルートにアクセスします。以下に例を示します。

```
http://example-apicurioregistry-kafkasql.my-project.my-domain-name.com/
```

9. Apicurio Registry がデータの保存に使用する Kafka トピックを設定するには、**Installed Operators > Red Hat Integration - AMQ Streams > Provided APIs > Kafka Topic > kafkasql-journal > YAML** をクリックします。以下に例を示します。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: ...
spec:
  partitions: 3
  replicas: 3
  config:
    cleanup.policy: compact
```



警告

Apicurio Registry で使用される Kafka トピック (デフォルトでは **kafkasql-journal** という名前) を圧縮クリーンアップポリシーで設定する必要があります。そうしないと、データ損失が発生する可能性があります。

関連情報

- AMQ Streams を使用した Kafka クラスターとトピックの作成の詳細については、[OpenShift での AMQ Streams のデプロイとアップグレード](#) を参照してください。

3.3. TLS セキュリティでの KAFKA ストレージの設定

AMQ Streams Operator および Service Registry Operator を、暗号化された Transport Layer Security (TLS) 接続を使用するように設定できます。

前提条件

- OperatorHub またはコマンドラインを使用して Service Registry Operator をインストールしている。
- AMQ Streams Operator がインストールされているか、Kafka が OpenShift クラスターからアクセスできる。



注記

ここでは、AMQ Streams Operator が利用可能であることを前提としていますが、任意の Kafka デプロイメントを使用できます。この場合、Service Registry Operator が想定する Openshift シークレットを手動で作成する必要があります。

手順

1. OpenShift Web コンソールで **Installed Operators** をクリックし、**AMQ Streams Operator** の詳細を選択してから、**Kafka** タブをクリックします。
2. **Create Kafka** をクリックし、Apicurio Registry ストレージの新しい Kafka クラスターをプロビジョニングします。
3. Kafka クラスターに TLS 認証を使用するように、**authorization** フィールドと **tls** フィールドを設定します。次に例を示します。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: registry-example-kafkasql-tls
  # Change or remove the explicit namespace
spec:
  kafka:
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      log.message.format.version: '2.7'
      inter.broker.protocol.version: '2.7'
    version: 2.7.0
    storage:
      type: ephemeral
    replicas: 3
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: tls
    authorization:
      type: simple
    entityOperator:
      topicOperator: {}
      userOperator: {}
    zookeeper:
      storage:
        type: ephemeral
      replicas: 3
```

データを保存するために Apicurio Registry によって自動作成されるデフォルトの Kafka トピック名は **kafkasql-journal** です。環境変数を設定することで、この動作またはデフォルトのトピック名をオーバーライドできます。デフォルト値は以下のとおりです。

- **REGISTRY_KAFKASQL_TOPIC_AUTO_CREATE=true**
- **REGISTRY_KAFKASQL_TOPIC=kafkasql-journal**

Kafka トピックを手動で作成しない場合は、次の手順を省略します。

4. **Kafka Topic** タブをクリックし、**Create Kafka Topic** をクリックして、**kafkasql-journal** トピックを作成します。

—

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-tls
spec:
  partitions: 2
  replicas: 1
  config:
    cleanup.policy: compact

```

5. **Kafka User** リソースを作成し、Apicurio Registry ユーザーの認証および承認を設定します。**metadata** セクションでユーザー名を指定するか、デフォルトの **my-user** を使用できません。

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-tls
spec:
  authentication:
    type: tls
  authorization:
    acls:
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: topic
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: cluster
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: transactionalId
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: group
    type: simple

```



注記

このシンプルな例では、admin パーミッションを前提とし、Kafka トピックを自動的に作成します。Apicurio レジストリーが必要とするトピックとリソース専用 **authorization** セクションを設定する必要があります。

次の例は、Kafka トピックを手動で作成する場合に必要な最小設定を示しています。

```
...
authorization:
  acls:
    - operations:
        - Read
        - Write
      resource:
        name: kafkasql-journal
        patternType: literal
        type: topic
    - operations:
        - Read
        - Write
      resource:
        name: apicurio-registry-
        patternType: prefix
        type: group
  type: simple
```

6. **Workloads**、**Secrets** の順にクリックして、Apicurio Registry が Kafka クラスターに接続するために AMQ Streams が作成する 2 つのシークレットを見つけます。

- **my-cluster-cluster-ca-cert** - Kafka クラスターの PKCS12 トラストストアが含まれています
- **my-user** - ユーザーのキーストアが含まれます



注記

シークレットの名前は、クラスターまたはユーザー名によって異なります。

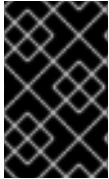
7. シークレットを手動で作成する場合は、以下のキーと値のペアを含める必要があります。

- **my-cluster-ca-cert**
 - **ca.p12** - PKCS12 形式のトラストストア
 - **ca.password** - truststore password
- **my-user**
 - **user.p12** - PKCS12 形式のキーストア
 - **user.password** - keystore password

8. 次の設定例を設定して、Apicurio Registry をデプロイします。

```
apiVersion: registry.apicur.io/v1
```

```
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-tls
spec:
  configuration:
    persistence: "kafkasql"
  kafkasql:
    bootstrapServers: "my-cluster-kafka-bootstrap.registry-example-kafkasql-tls.svc:9093"
    security:
      tls:
        keystoreSecretName: my-user
        truststoreSecretName: my-cluster-cluster-ca-cert
```



重要

プレーンでセキュアでないユースケースとは別の **bootstrapServers** アドレスを使用する必要があります。アドレスは TLS 接続をサポートする必要があり、指定された **Kafka** リソースの **type:tls** フィールドにあります。

3.4. SCRAM セキュリティーでの KAFKA ストレージの設定

Kafka クラスターの Salted Challenge Response Authentication Mechanism (SCRAM-SHA-512) を使用するように AMQ Streams Operator および Service Registry Operator を設定できます。

前提条件

- OperatorHub またはコマンドラインを使用して Service Registry Operator をインストールしている。
- AMQ Streams Operator がインストールされているか、Kafka が OpenShift クラスターからアクセスできる。



注記

ここでは、AMQ Streams Operator が利用可能であることを前提としていますが、任意の Kafka デプロイメントを使用できます。この場合、Service Registry Operator が想定する Openshift シークレットを手動で作成する必要があります。

手順

1. OpenShift Web コンソールで **Installed Operators** をクリックし、**AMQ Streams Operator** の詳細を選択してから、**Kafka** タブをクリックします。
2. **Create Kafka** をクリックし、Apicurio Registry ストレージの新しい Kafka クラスターをプロビジョニングします。
3. Kafka クラスターに SCRAM-SHA-512 認証を使用するように、**authorization** フィールドと **tls** フィールドを設定します。次に例を示します。

```
apiVersion: kafka.strimzi.io/v1beta2
kind: Kafka
metadata:
  name: my-cluster
  namespace: registry-example-kafkasql-scrum
```

```

# Change or remove the explicit namespace
spec:
  kafka:
    config:
      offsets.topic.replication.factor: 3
      transaction.state.log.replication.factor: 3
      transaction.state.log.min.isr: 2
      log.message.format.version: '2.7'
      inter.broker.protocol.version: '2.7'
    version: 2.7.0
    storage:
      type: ephemeral
    replicas: 3
    listeners:
      - name: tls
        port: 9093
        type: internal
        tls: true
        authentication:
          type: scram-sha-512
    authorization:
      type: simple
    entityOperator:
      topicOperator: {}
      userOperator: {}
    zookeeper:
      storage:
        type: ephemeral
      replicas: 3

```

データを保存するために Apicurio Registry によって自動作成されるデフォルトの Kafka トピック名は **kafkasql-journal** です。環境変数を設定することで、この動作またはデフォルトのトピック名をオーバーライドできます。デフォルト値は以下のとおりです。

- **REGISTRY_KAFKASQL_TOPIC_AUTO_CREATE=true**
- **REGISTRY_KAFKASQL_TOPIC=kafkasql-journal**

Kafka トピックを手動で作成しない場合は、次の手順を省略します。

4. **Kafka Topic** タブをクリックし、**Create Kafka Topic** をクリックして、**kafkasql-journal** トピックを作成します。

```

apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaTopic
metadata:
  name: kafkasql-journal
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-scram
spec:
  partitions: 2
  replicas: 1
  config:
    cleanup.policy: compact

```

5. **Kafka User** リソースを作成し、Apicurio Registry ユーザーの SCRAM 認証および承認を設定します。 **metadata** セクションでユーザー名を指定するか、デフォルトの **my-user** を使用できます。

```
apiVersion: kafka.strimzi.io/v1beta1
kind: KafkaUser
metadata:
  name: my-user
  labels:
    strimzi.io/cluster: my-cluster
  namespace: registry-example-kafkasql-scam
spec:
  authentication:
    type: scram-sha-512
  authorization:
    acls:
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: topic
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: cluster
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: transactionalId
      - operation: All
        resource:
          name: '*'
          patternType: literal
          type: group
    type: simple
```



注記

このシンプルな例では、admin パーミッションを前提とし、Kafka トピックを自動的に作成します。Apicurio レジストリーが必要とするトピックとリソース専用 **authorization** セクションを設定する必要があります。

次の例は、Kafka トピックを手動で作成する場合に必要な最小設定を示しています。

```
...
authorization:
  acls:
    - operations:
        - Read
        - Write
      resource:
        name: kafkasql-journal
        patternType: literal
```

```

    type: topic
  - operations:
    - Read
    - Write
  resource:
    name: apicurio-registry-
    patternType: prefix
    type: group
  type: simple

```

6. **Workloads**、**Secrets** の順にクリックして、Apicurio Registry が Kafka クラスターに接続するために AMQ Streams が作成する 2 つのシークレットを見つけます。

- **my-cluster-cluster-ca-cert** - Kafka クラスターの PKCS12 トラストストアが含まれています
- **my-user** - ユーザーのキーストアが含まれます



注記

シークレットの名前は、クラスターまたはユーザー名によって異なります。

7. シークレットを手動で作成する場合は、以下のキーと値のペアを含める必要があります。

- **my-cluster-ca-cert**
 - **ca.p12** - PKCS12 形式のトラストストア
 - **ca.password** - truststore password
- **my-user**
 - **パスワード** - ユーザーパスワード

8. Apicurio Registry をデプロイするように、以下の設定例を設定します。

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-scam
spec:
  configuration:
    persistence: "kafkasql"
    kafkasql:
      bootstrapServers: "my-cluster-kafka-bootstrap.registry-example-kafkasql-
scram.svc:9093"
    security:
      scram:
        truststoreSecretName: my-cluster-cluster-ca-cert
        user: my-user
        passwordSecretName: my-user

```



重要

プレーンでセキュアでないユースケースとは別の **bootstrapServers** アドレスを使用する必要があります。アドレスは TLS 接続をサポートする必要があり、指定された Kafka リソースの **type:tls** フィールドにあります。

3.5. KAFKA ストレージの OAUTH 認証の設定

AMQ Streams で Kafka-based のストレージを使用する場合、Service Registry は OAuth 認証を必要とする Kafka クラスターへのアクセスをサポートします。このサポートを有効にするには、Apicurio Registry デプロイメントでいくつかの環境変数を設定する必要があります。

これらの環境変数を設定すると、Apicurio Registry の Kafka プロデューサーおよびコンシューマーアプリケーションはこの設定を使用して、OAuth を介して Kafka クラスターに対して認証します。

前提条件

- AMQ Streams で Apicurio Registry データの Kafka ベースのストレージをすでに設定している必要があります。[「OpenShift で Kafka ストレージを使用して Apicurio Registry を設定する」](#)を参照してください。

手順

- Apicurio Registry デプロイメントで次の環境変数を設定します。

環境変数	説明	デフォルト値
ENABLE_KAFKA_SASL	Kafka の Apicurio Registry ストレージの SASL OAuth 認証を有効にします。他の変数を有効にするには、この変数を true に設定する必要があります。	false
CLIENT_ID	Kafka への認証に使用されるクライアント ID。	-
CLIENT_SECRET	Kafka への認証に使用されるクライアントシークレット。	-
OAUTH_TOKEN_ENDPOINT_URI	OAuth ID サーバーの URL。	http://localhost:8090

関連情報

- OpenShift で Apicurio Registry 環境変数を設定する方法の例については、次を参照してください。[「OpenShift での Apicurio Registry ヘルスチェックの設定」](#)

第4章 POSTGRESQL データベースに APICURIO REGISTRY ストレージをデプロイする

本章では、PostgreSQL データベースで Apicurio Registry データストレージをインストール、設定、および管理する方法を説明します。

- [「OpenShift OperatorHub からの PostgreSQL データベースのインストール」](#)
- [「OpenShift での PostgreSQL データベースストレージを使用した Apicurio Registry の設定」](#)
- [「Apicurio Registry PostgreSQL ストレージのバックアップ」](#)
- [「Apicurio Registry PostgreSQL ストレージの復元」](#)

前提条件

- [2章OpenShift に Apicurio Registry をインストールする](#)

4.1. OPENSIFT OPERATORHUB からの POSTGRESQL データベースのインストール

PostgreSQL データベース Operator がインストールされていない場合は、OperatorHub から OpenShift クラスターに PostgreSQL Operator をインストールできます。OperatorHub は OpenShift Container Platform Web コンソールから使用でき、クラスター管理者が Operator を検出およびインストールするためのインターフェイスを提供します。詳細については、[OperatorHub について](#) を参照してください。

前提条件

- クラスター管理者として OpenShift クラスターにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. PostgreSQL Operator をインストールする OpenShift プロジェクトに切り替えます。たとえば、**Project** ドロップダウンから、**my-project** を選択します。
3. 左側のナビゲーションメニューで、**Operators** をクリックした後、**OperatorHub** をクリックします。
4. **Filter by keyword** テキストボックスに **PostgreSQL** と入力して、環境に適した Operator を見つけます (例: **Crunchy PostgreSQL for OpenShift**)。
5. Operator に関する情報を読み、**Install** をクリックして Operator サブスクリプションページを表示します。
6. サブスクリプション設定を選択します。以下に例を示します。
 - **Update Channel:** **stable**
 - **Installation Mode:** **A specific namespace on the cluster** および **my-project**
 - **Approval Strategy:** **Automatic** または **Manual** を選択します。

7. **Install** をクリックし、Operator が使用できるようになるまでしばらく待ちます。



重要

データベースの作成と管理方法の詳細については、選択した **PostgreSQL Operator** のドキュメントを読む必要があります。

関連情報

- [Operator の OpenShift クラスターへの追加](#)
- [Crunchy PostgreSQL Operator QuickStart](#)

4.2. OPENSIFT での POSTGRESQL データベースストレージを使用した APICURIO REGISTRY の設定

本セクションでは、PostgreSQL データベース Operator を使用して、OpenShift の Apicurio Registry のストレージを設定する方法を説明します。既存のデータベースに Apicurio Registry をインストールするか、環境に応じて新規データベースを作成することができます。本セクションでは、Dev4Ddevs.com による PostgreSQL Operator を使用する簡単な例を紹介します。

前提条件

- クラスター管理者として OpenShift クラスターにアクセスできる。
- Apicurio Registry がすでにインストールされている必要があります。[2章 OpenShift に Apicurio Registry をインストールする](#)を参照してください。
- OpenShift に PostgreSQL Operator がすでにインストールされている。たとえば、「[OpenShift OperatorHub からの PostgreSQL データベースのインストール](#)」を参照してください。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. Apicurio Registry および PostgreSQL Operator がインストールされている OpenShift プロジェクトに切り替えます。たとえば、**Project** ドロップダウンから、**my-project** を選択します。
3. Apicurio Registry ストレージの PostgreSQL データベースを作成します。たとえば、**Installed Operators**、**PostgreSQL Operator by Dev4Ddevs.com** の順にクリックした後、**Create database** をクリックします。
4. **YAML** をクリックし、以下のようにデータベース設定を編集します。
 - **name:** 値を **registry** に変更します
 - **image:** 値を **centos/postgresql-12-centos7** に変更します
5. 実際の環境に応じて、必要に応じてその他のデータベース設定を編集します。以下に例を示します。

```
apiVersion: postgresql.dev4devs.com/v1alpha1
kind: Database
```

```

metadata:
  name: registry
  namespace: my-project
spec:
  databaseCpu: 30m
  databaseCpuLimit: 60m
  databaseMemoryLimit: 512Mi
  databaseMemoryRequest: 128Mi
  databaseName: example
  databaseNameKeyEnvVar: POSTGRESQL_DATABASE
  databasePassword: postgres
  databasePasswordKeyEnvVar: POSTGRESQL_PASSWORD
  databaseStorageRequest: 1Gi
  databaseUser: postgres
  databaseUserKeyEnvVar: POSTGRESQL_USER
  image: centos/postgresql-12-centos7
  size: 1

```

6. **Create** をクリックし、データベースが作成されるまで待ちます。
7. **Installed Operators > Red Hat Integration - Service Registry > ApicurioRegistry > Create ApicurioRegistry** をクリックします。
8. 以下のカスタムリソース定義に貼り付け、データベース **url** およびクレデンシャルの値を編集して環境と一致するようにします。

```

apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-sql
spec:
  configuration:
    persistence: 'sql'
  sql:
    dataSource:
      url: 'jdbc:postgresql://<service name>.<namespace>.svc:5432/<database name>'
      # e.g. url: 'jdbc:postgresql://acid-minimal-cluster.my-project.svc:5432/registry'
      userName: 'postgres'
      password: '<password>' # Optional

```

9. **Create** をクリックし、Apicurio Registry ルートが OpenShift に作成されるのを待ちます。
10. **Networking > Route** をクリックして、Apicurio Registry Web コンソールの新しいルートにアクセスします。以下に例を示します。

```
http://example-apicurioregistry-sql.my-project.my-domain-name.com/
```

関連情報

- [Crunchy PostgreSQL Operator QuickStart](#)
- [Apicurio Registry Operator QuickStart](#)

4.3. APICURIO REGISTRY POSTGRESQL ストレージのバックアップ

PostgreSQL データベースでストレージを使用する場合は、Apicurio Registry に保存されているデータを定期的にバックアップする必要があります。

[SQL Dump](#) は、どのような PostgreSQL インストールでも動作するシンプルな手順です。これは [pg_dump](#) ユーティリティを使用して、ダンプ時と同じ状態でデータベースを再作成するために使用できる SQL コマンドでファイルを生成します。

pg_dump は通常の PostgreSQL クライアントアプリケーションで、データベースにアクセスできる任意のリモートホストから実行することができます。他のクライアントと同様に、実行できる操作はユーザーの権限によって制限されます。

手順

- **pg_dump** コマンドを使用して、出力をファイルにリダイレクトします。

```
$ pg_dump dbname > dumpfile
```

-hhost および **-pport** オプションを使用して、**pg_dump** が接続するデータベースサーバーを指定できます。

- gzip などの圧縮ツールを使用して大きなダンプファイルを減らすことができます。以下に例を示します。

```
$ pg_dump dbname | gzip > filename.gz
```

関連情報

- クライアント認証の詳細は、[PostgreSQL のドキュメント](#) を参照してください。
- レジストリーコンテンツのインポートおよびエクスポートに関する詳細は、[Managing Apicurio Registry content using the REST API](#) を参照してください。

4.4. APICURIO REGISTRY POSTGRESQL ストレージの復元

psql ユーティリティを使用して、**pg_dump** によって作成された SQL Dump ファイルを復元できます。

前提条件

- **pg_dump** を使用して、PostgreSQL データベースをすでにバックアップしている。「[Apicurio Registry PostgreSQL ストレージのバックアップ](#)」を参照してください。
- オブジェクトを所有するユーザー、またはダンプされたデータベースのオブジェクトに対する権限があるユーザーがすべて存在している。

手順

1. 以下のコマンドを入力して、データベースを作成します。

```
$ createdb -T template0 dbname
```

2. 以下のコマンドを入力して SQL ダンプを復元します

```
$ psql dbname < dumpfile
```

-
- 3. クエリーオプティマイザーが便利な統計を持つように、各データベースで [ANALYZE](#) を実行します。

第5章 APICURIO REGISTRY デプロイメントの保護

本章では、OpenShift での Apicurio Registry デプロイメントのセキュリティー設定について説明します。

- [「Red Hat Single Sign-On Operator を使用した Apicurio Registry の保護」](#)
- [「Red Hat Single Sign-On を使用した Apicurio Registry の認証と承認の設定」](#)
- [「Apicurio Registry の認証および承認の設定オプション」](#)
- [「OpenShift クラスター内から Apicurio Registry への HTTPS 接続の設定」](#)
- [「OpenShift クラスター外から Apicurio Registry への HTTPS 接続の設定」](#)

Apicurio Registry は、OpenID Connect (OIDC) または HTTP Basic に基づいて Red Hat Single Sign-On を使用して認証および承認を行います。Red Hat Single Sign-On Operator を使用して必要な設定を自動的に設定するか、Red Hat Single Sign-On および Apicurio Registry で手動で設定する必要があります。

Apicurio Registry は、Red Hat Single Sign-On を使用して、Apicurio Registry Web コンソールとコア REST API にロールベースの認証と許可を提供します。Apicurio Registry は、アーティファクト作成者のみが書き込みアクセスを持つスキーマまたは API レベルでコンテンツベースの承認も提供します。OpenShift クラスターの内部または外部から Apicurio Registry への HTTPS 接続を設定することもできます。

関連情報

- Java クライアントアプリケーションのセキュリティー設定の詳細は、以下を参照してください。
 - [Apicurio Registry Java クライアント設定](#)
 - [Apicurio Registry シリアライザー/デシリアライザーの設定](#)

5.1. RED HAT SINGLE SIGN-ON OPERATOR を使用した APICURIO REGISTRY の保護

次の手順は、Red Hat Single Sign-On によって保護されるように Apicurio Registry REST API と Web コンソールを設定する方法を示しています。

Apicurio Registry は、次のユーザーロールをサポートしています。

表5.1 Apicurio Registry ユーザーのロール

名前	機能
sr-admin	完全なアクセス。制限はありません。
sr-developer	アーティファクトを作成し、アーティファクトルールを設定します。グローバルルールの変更、インポート/エクスポートの実行、 /admin REST API エンドポイントの使用はできません。

名前	機能
sr-readonly	表示と検索のみ。アーテファクトやルールの変更、インポート/エクスポートの実行、 /admin REST API エンドポイントの使用はできません。



注記

ApicurioRegistry CRD には、Web コンソールを読み取り専用モードに設定するために使用できる関連する設定オプションがあります。ただし、この設定は REST API には影響しません。

前提条件

- Service Registry Operator がインストールされている。
- Red Hat Single Sign-On Operator をインストールするか、または OpenShift クラスターからアクセスできる Red Hat Single Sign-On が必要です。



重要

この手順の設定例は、開発およびテストのみを目的としています。手順を単純にするために、実稼働環境で推奨される HTTPS やその他のセキュリティは使用しません。詳細は、Red Hat Single Sign-On のドキュメントを参照してください。

手順

1. OpenShift Web コンソールで、**Installed Operators** および **Red Hat Single Sign-On Operator** をクリックし、**Keycloak** タブをクリックします。
2. **Create Keycloak** をクリックし、Apicurio Registry デプロイメントのセキュリティを保護するために、新しい Red Hat Single Sign-On インスタンスをプロビジョニングします。デフォルト値を使用できます。以下に例を示します。

```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: example-keycloak
  labels:
    app: sso
spec:
  instances: 1
  externalAccess:
    enabled: True
  podDisruptionBudget:
    enabled: True
```

3. インスタンスが作成されるまで待ち、**Networking** をクリックした後に **Routes** をクリックし、**keycloak** インスタンスの新規ルートにアクセスします。
4. **Location URL** をクリックし、表示された **../auth** URL 値をコピーして、後で Apicurio Registry のデプロイ時に使用します。

5. **Installed Operators** および **Red Hat Single Sign-On Operator** をクリックし、**Keycloak Realm** タブをクリックした後、**Create Keycloak Realm** をクリックして **registry** のサンプルレムを作成します。

```

apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  name: registry-keycloakrealm
  labels:
    app: registry
spec:
  instanceSelector:
    matchLabels:
      app: sso
  realm:
    displayName: Registry
    enabled: true
    id: registry
    realm: registry
    sslRequired: none
    roles:
      realm:
        - name: sr-admin
        - name: sr-developer
        - name: sr-readonly
  clients:
    - clientId: registry-client-ui
      implicitFlowEnabled: true
      redirectUris:
        - '*'
      standardFlowEnabled: true
      webOrigins:
        - '*'
      publicClient: true
    - clientId: registry-client-api
      implicitFlowEnabled: true
      redirectUris:
        - '*'
      standardFlowEnabled: true
      webOrigins:
        - '*'
      publicClient: true
  users:
    - credentials:
        - temporary: false
          type: password
          value: changeme
      enabled: true
      realmRoles:
        - sr-admin
      username: registry-admin
    - credentials:
        - temporary: false
          type: password
          value: changeme
      enabled: true

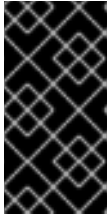
```



```

realmRoles:
  - sr-developer
username: registry-developer
- credentials:
  - temporary: false
    type: password
    value: changeme
enabled: true
realmRoles:
  - sr-readonly
username: registry-user

```



重要

実稼働環境にデプロイする場合は、ご使用の環境に適した値でこの **KeycloakRealm** リソースをカスタマイズする必要があります。Red Hat Single Sign-On Web コンソールを使用してレルムを作成および管理することもできます。

6. クラスターに有効な HTTPS 証明書が設定されていない場合は、一時的な回避策として次の HTTP **Service** および **Ingress** リソースを作成できます。
 - a. **Networking** をクリックしてから **Services** をクリックし、以下の例を使用して **Create Service** をクリックします。

```

apiVersion: v1
kind: Service
metadata:
  name: keycloak-http
labels:
  app: keycloak
spec:
  ports:
    - name: keycloak-http
      protocol: TCP
      port: 8080
      targetPort: 8080
  selector:
    app: keycloak
    component: keycloak
  type: ClusterIP
  sessionAffinity: None
status:
  loadBalancer: {}

```

- b. **Networking** をクリックしてから **Ingresses** をクリックし、以下の例を使用して **Create Ingress** をクリックします。

```

apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: keycloak-http
labels:
  app: keycloak
spec:

```

```
rules:
  - host: KEYCLOAK_HTTP_HOST
    http:
      paths:
        - path: /
          pathType: ImplementationSpecific
      backend:
        service:
          name: keycloak-http
          port:
            number: 8080
```

host の値を変更して、Apicurio Registry ユーザーがアクセスできるルートを作成し、Red Hat Single Sign-On Operator によって作成された HTTPS ルートの代わりにこれを使用します。

7. **Service Registry Operator** をクリックし、以下の例のように **ApicurioRegistry** タブをクリックして **Create ApicurioRegistry** をクリックしますが、**keycloak** セクションの値を置き換えます。

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry-kafkasql-keycloak
spec:
  configuration:
    security:
      keycloak:
        url: "http://keycloak-http-<namespace>.apps.<cluster host>/auth"
        # ^ Required
        # Keycloak server URL, must end with `auth`.
        # Use an HTTP URL in development.
        realm: "registry"
        # apiClientId: "registry-client-api"
        # ^ Optional (default value)
        # uiClientId: "registry-client-ui"
        # ^ Optional (default value)
      persistence: 'kafkasql'
      kafkasql:
        bootstrapServers: '<my-cluster>-kafka-bootstrap.<my-namespace>.svc:9092'
```

5.2. RED HAT SINGLE SIGN-ON を使用した APICURIO REGISTRY の認証と承認の設定

本セクションでは、Red Hat Single Sign-On を使用して Apicurio Registry の認証および承認オプションを手作業で設定する方法を説明します。



注記

また、これらの設定を自動的に設定する方法の詳細は、[「Red Hat Single Sign-On Operator を使用した Apicurio Registry の保護」](#) を参照してください。

OpenID Connect (OIDC) 使用して OAuth をベースとした Red Hat Single Sign-On を使用して、Apicurio Registry Web コンソールおよびコア REST API の認証を有効にすることができます。同じ Red

Hat Single Sign-On レalmとユーザーは OpenID Connect を使用して Apicurio Registry Web コンソールとコア REST API で連携されるため、必要なクレデンシャルは1セットのみです。

Apicurio Registry は、デフォルトの admin、write、および read-only ユーザーロールのロールベースの承認を提供します。Apicurio Registry は、スキーマまたは API レベルでコンテンツベースの承認を提供し、レジストリーアーティファクトの作成者のみが更新または削除できます。Apicurio Registry の認証および承認設定は、デフォルトでは無効になっています。

前提条件

- Red Hat Single Sign-On がインストールされ、実行中である。詳細は、[Red Hat Single Sign-On のユーザードキュメント](#) を参照してください。
- Apicurio Registry がインストールされ、実行されています。

手順

1. Red Hat Single Sign-On 管理コンソールで、Apicurio Registry 用の Red Hat Single Sign-On レalmを作成します。デフォルトでは、Apicurio Registry は **registry** のレalm名を想定しています。レalmの作成に関する詳細は、[Red Hat Single Sign-On のユーザードキュメント](#) を参照してください。
2. Apicurio Registry API 用の Red Hat Single Sign-On クライアントを作成します。デフォルトでは、Apicurio Registry は次の設定を想定しています。

- **Client ID:** **registry-api**
- **Client Protocol:** **openid-connect**
- **Access Type:** **bearer-only**
他のクライアント設定にはデフォルト値を使用できます。



注記

Red Hat Single Sign-On サービスアカウントを使用している場合、クライアントの **Access Type** は、**bearer-only** ではなく **confidential** である必要があります。

3. Apicurio Registry Web コンソール用の Red Hat Single Sign-On クライアントを作成します。デフォルトでは、Apicurio Registry は次の設定を想定しています。
- **Client ID:** **apicurio-registry**
 - **Client Protocol:** **openid-connect**
 - **Access Type:** **public**
 - **Valid Redirect URLs:** **http://my-registry-url:8080/***
 - **Web Origins:** **+**
他のクライアント設定にはデフォルト値を使用できます。
4. OpenShift 上の Apicurio Registry デプロイメントで、次の Apicurio Registry 環境変数を設定して、Red Hat Single Sign-On を使用した認証を設定します。

表5.2 Apicurio Registry 認証の設定

環境変数	説明	型	デフォルト
AUTH_ENABLED	true に設定すると、以下の環境変数が必要です。	文字列	false
KEYCLOAK_URL	使用する Red Hat Single Sign-On 認証サーバーの URL。/auth で終わる必要があります。	文字列	なし
KEYCLOAK_REALM	認証に使用する Red Hat Single Sign-On レalm。	文字列	レジストリー
KEYCLOAK_API_CLIENT_ID	Apicurio Registry REST API のクライアント ID。	文字列	registry-api
KEYCLOAK_UI_CLIENT_ID	Apicurio Registry Web コンソールのクライアント ID。	文字列	apicurio-registry

ヒント

OpenShift に環境変数を設定する例については、「[OpenShift での Apicurio Registry ヘルスチェックの設定](#)」を参照してください。

- 以下のオプションを **true** に設定して、Red Hat Single Sign-On で Apicurio Registry ユーザーロールを有効にします。

表5.3 Apicurio Registry ロールベース認証の設定

環境変数	Java システムプロパティ	型	デフォルト値
ROLE_BASED_AUTHZ_ENABLED	registry.auth.role-based-authorization	Boolean	false

- Apicurio Registry ユーザーロールが有効になっている場合、Apicurio Registry ユーザーを、Red Hat Single Sign-On レalm内の以下のデフォルトユーザーロールの少なくとも1つに割り当てる必要があります。

表5.4 レジストリーの認証および承認のデフォルトユーザーロール

ロール	アーティファクトの読み取り	アーティファクトの書き込み	グローバルルール	概要
sr-admin	○	○	○	すべての作成、読み取り、更新、および削除操作へのフルアクセス。

ロール	アーティファクトの読み取り	アーティファクトの書き込み	グローバルルール	概要
sr-developer	○	○	×	グローバルルールの設定を除く、作成、読み取り、更新、および削除操作へのアクセス。このロールはアーティファクトルールを設定できません。
sr-readonly	○	×	×	読み取りおよび検索操作のみへのアクセス。このロールはルールを設定できません。

7. 以下を **true** に設定して、Apicurio Registry のスキーマおよび API アーティファクトの更新に対して所有者のみの承認を有効にします。

表5.5 所有者のみの承認の設定

環境変数	Java システムプロパティ	型	デフォルト値
REGISTRY_AUTH_OBAC_ENABLED	registry.auth.owner-only-authorization	Boolean	false

関連情報

- デフォルト以外のユーザーロール名の設定の詳細は、[「Apicurio Registry の認証および承認の設定オプション」](#) を参照してください。
- オープンソースのアプリケーションおよび Keycloak レalmについては、[Docker Compose-based example of using Keycloak with Apicurio Registry](#) を参照してください。
- 実稼働環境で Red Hat Single Sign-On を使用する方法の詳細は、[Red Hat Single Sign-On のドキュメント](#) を参照してください。
- カスタムセキュリティの設定に関する詳細は、[Quarkus Open ID Connect のドキュメント](#) を参照してください。

5.3. APICURIO REGISTRY の認証および承認の設定オプション

Apicurio Registry は、Red Hat Single Sign-On または HTTP 基本認証を使用した OpenID Connect の認証オプションを提供します。

Apicurio Registry には、ロールベースおよびコンテンツベースのアプローチの承認オプションが用意されています。

- デフォルトの管理者、書き込み、および読み取り専用のユーザーロールに対するロールベースの承認。
- アーティファクトまたはアーティファクトグループの所有者のみがアーティファクトを更新または削除できるスキーマまたは API アーティファクトのコンテンツベースの承認。



注記

Apicurio Registry の認証および承認オプションは、デフォルトで無効になっています。

この章では、次の設定オプションについて詳しく説明します。

- [OpenID Connect と Red Hat Single Sign-On を使用した Apicurio レジストリー認証](#)
- [HTTP Basic を使用した Apicurio レジストリー認証](#)
- [Apicurio Registry のロールベースの承認](#)
- [Apicurio Registry 所有者のみの許可](#)
- [Apicurio Registry の認証済み読み取りアクセス](#)
- [Apicurio Registry の匿名の読み取り専用アクセス](#)

OpenID Connect と Red Hat Single Sign-On を使用した Apicurio レジストリー認証

以下の環境変数を設定して、Red Hat Single Sign-On を使用して Apicurio Registry Web コンソールおよび API の認証を設定できます。

表5.6 Apicurio Registry 認証オプションの設定

環境変数	説明	型	デフォルト
AUTH_ENABLED	Apicurio Registry での認証を有効または無効にします。 true に設定すると、以下の環境変数が必要です。	文字列	false
KEYCLOAK_URL	使用する Red Hat Single Sign-On 認証サーバーの URL。/auth で終わる必要があります。	文字列	-
KEYCLOAK_REALM	認証に使用する Red Hat Single Sign-On レalm。	文字列	-
KEYCLOAK_API_CLIENT_ID	Apicurio Registry REST API のクライアント ID。	文字列	registry-api
KEYCLOAK_UI_CLIENT_ID	Apicurio Registry Web コンソールのクライアント ID。	文字列	apicurio-registry

HTTP Basic を使用した Apicurio レジストリー認証



注記

デフォルトでは、Apicurio Registry は OpenID Connect を使用した認証をサポートしています。ユーザー (または API クライアント) は、Apicurio Registry REST API への認証済み呼び出しを行うためにアクセストークンを取得する必要があります。ただし、一部のツールは OpenID Connect をサポートしていないため、次の設定オプションを **true** に設定することで、HTTP 基本認証をサポートするように Apicurio Registry を設定することもできます。

表5.7 Apicurio Registry HTTP 基本認証の設定

環境変数	Java システムプロパティ	型	デフォルト値
CLIENT_CREDENTIALS_BASIC_AUTH_ENABLED	registry.auth.basic-auth-client-credentials.enabled	Boolean	false

Apicurio Registry のロールベースの承認

次のオプションを **true** に設定して、Apicurio Registry でロールベースの承認を有効にすることができます。

表5.8 Apicurio Registry ロールベース認証の設定

環境変数	Java システムプロパティ	型	デフォルト値
ROLE_BASED_AUTHZ_ENABLED	registry.auth.role-based-authorization	Boolean	false

次に、ユーザーの認証トークンに含まれるロール (Red Hat Single Sign-On を使用した認証時に付与されるロールなど) を使用するか、Apicurio Registry によって内部的に管理されるロールマッピングを使用するように、ロールベースの承認を設定できます。

Red Hat Single Sign-On で割り当てられたロールを使用する

Red Hat Single Sign-On によって割り当てられたロールを使用できるようにするには、以下の環境変数を設定します。

表5.9 Red Hat Single Sign-On を使用した Apicurio Registry ロールベース認証の設定

環境変数	説明	型	デフォルト
ROLE_BASED_AUTHZ_SOURCE	token に設定すると、ユーザーのロールは認証トークンから取得されます。	文字列	トークン (token)
REGISTRY_AUTH_ROLES_ADMIN	ユーザーが管理者であることを示すロールの名前。	文字列	sr-admin
REGISTRY_AUTH_ROLES_DEVELOPER	ユーザーが開発者であることを示すロールの名前。	文字列	sr-developer
REGISTRY_AUTH_ROLES_READONLY	ユーザーが読み取り専用アクセス権を持っていることを示すロールの名前。	文字列	sr-readonly

Apicurio Registry が Red Hat Single Sign-On のロールを使用するように設定されている場合は、Apicurio Registry ユーザーを Red Hat Single Sign-On の以下のユーザーロールの少なくとも1つに割り当てる必要があります。ただし、[表5.9「Red Hat Single Sign-On を使用した Apicurio Registry ロールベース認証の設定」](#)の環境変数を使用して別のユーザーロール名を設定できます。

表5.10 認証および承認のための Apicurio Registry ロール

ロール名	アーティファクトの読み取り	アーティファクトの書き込み	グローバルルール	説明
sr-admin	○	○	○	すべての作成、読み取り、更新、および削除操作へのフルアクセス。
sr-developer	○	○	×	グローバルルールの設定およびインポート/エクスポートを除く、操作の作成、読み取り、更新、および削除へのアクセス。このロールはアーティファクトルールのみを設定できます。
sr-readonly	○	×	×	読み取りおよび検索操作のみへのアクセス。このロールはルールを設定できません。

Apicurio Registry でロールを直接管理する

Apicurio Registry によって内部的に管理されるロールの使用を有効にするには、次の環境変数を設定します。

表5.11 内部ロールマッピングを使用した Apicurio Registry ロールベースの承認の設定

環境変数	説明	型	デフォルト
ROLE_BASED_AUTHZ_SOURCE	application に設定すると、ユーザーロールは Apicurio Registry によって内部的に管理されます。	文字列	トークン (token)

内部でkに管理されたロールマッピングを使用する場合は、Apicurio Registry REST API の **/admin/roleMappings** エンドポイントを使用して、ユーザーにロールを割り当てることができます。詳細は、[Apicurio Registry REST API のドキュメント](#) を参照してください。

ユーザーに付与できるロールは、**ADMIN**、**DEVELOPER**、または **READ_ONLY** のいずれかだけです。管理者権限を持つユーザーのみが、他のユーザーにアクセス権を付与できます。

Apicurio Registry の管理者オーバーライド設定

Apicurio Registry にはデフォルトの管理者ユーザーがないため、通常、ユーザーが管理者として識別されるように別の方法を設定すると便利です。次の環境変数を使用して、この管理オーバーライド機能を設定できます。

表5.12 Apicurio Registry admin-override の設定

環境変数	説明	型	デフォルト
REGISTRY_AUTH_ADMIN_OVERRIDE_ENABLED	管理オーバーライド機能を有効にします。	文字列	false

環境変数	説明	型	デフォルト
REGISTRY_AUTH_ADMIN_OVERRIDE_FROM	管理オーバーライド情報を探す場所。現在 token のみがサポートされています。	文字列	token
REGISTRY_AUTH_ADMIN_OVERRIDE_TYPE	ユーザーが管理者かどうかを判断するために使用される情報の種類。値は FROM 変数の値によって異なります。たとえば、FROM が token の場合は role または claim です。	文字列	role
REGISTRY_AUTH_ADMIN_OVERRIDE_ROLE	ユーザーが管理者であることを示すロールの名前。	文字列	sr-admin
REGISTRY_AUTH_ADMIN_OVERRIDE_CLAIM	管理オーバーライドを決定するために使用する JWT トークンクレームの名前。	文字列	org-admin
REGISTRY_AUTH_ADMIN_OVERRIDE_CLAIM_VALUE	CLAIM 変数によって示される JWT トークンクレームの値は、ユーザーが管理オーバーライドを付与されるためのものでなければなりません。	文字列	true

たとえば、この管理オーバーライド機能を使用して、**sr-admin** ロールを Red Hat Single Sign-On の 1 人のユーザーに割り当て、そのユーザーに admin ロールを付与できます。そのユーザーは、**/admin/roleMappings** REST API (または関連する UI) を使用して、追加のユーザー (追加の管理者を含む) にロールを付与できます。

Apicurio Registry 所有者のみの許可

以下のオプションを **true** に設定して、Apicurio Registry 内のアーティファクトまたはアーティファクトグループの更新に対して所有者のみの許可を有効にすることができます。

表5.13 所有者のみの承認の設定

環境変数	Java システムプロパティ	型	デフォルト値
REGISTRY_AUTH_OBAC_ENABLED	registry.auth.owner-only-authorization	Boolean	false
REGISTRY_AUTH_OBAC_LIMIT_GROUP_ACCESS	registry.auth.owner-only-authorization.limit-group-access	Boolean	false

所有者のみの許可が有効になっている場合は、アーティファクトを作成したユーザーのみがそのアーティファクトを変更または削除できます。

所有者のみの許可とグループの所有者のみの許可の両方が有効になっている場合は、アーティファクトグループを作成したユーザーのみが、そのアーティファクトグループへの書き込みアクセス権 (そのグループのアーティファクトを追加または削除するなど) を持ちます。

Apicurio Registry の認証済み読み取りアクセス

認証済み読み取りアクセスオプションが有効になっている場合、Apicurio Registry は、ユーザーロールに関係なく、同じ組織内の認証済みユーザーからのリクエストに対して、少なくとも読み取り専用アクセスを許可します。

認証された読み取りアクセスを有効にするには、最初にロールベースの承認を有効にしてから、次のオプションを **true** に設定する必要があります。

表5.14 認証済み読み取りアクセスの設定

環境変数	Java システムプロパティ	型	デフォルト値
REGISTRY_AUTH_AUTHENTICATED_READS_ENABLED	registry.auth.authenticated-read-access.enabled	Boolean	false

詳細は、「[Apicurio Registry のロールベースの承認](#)」を参照してください。

Apicurio Registry の匿名の読み取り専用アクセス

2つの主要な認証タイプ(ロールベースの認証と所有者ベースの認証)に加えて、Apicurio Registry は匿名の読み取り専用アクセスオプションをサポートしています。

認証のクレデンシャルを持たない REST API 呼び出しなどの匿名ユーザーが REST API への読み取り専用呼び出しを行うことを許可するには、次のオプションを **true** に設定します。

表5.15 匿名の読み取り専用アクセスの設定

環境変数	Java システムプロパティ	型	デフォルト値
REGISTRY_AUTH_ANONYMOUS_READ_ACCESS_ENABLED	registry.auth.anonymous-read-access.enabled	Boolean	false

関連情報

- OpenShift の Apicurio Registry デプロイメントで環境変数を設定する方法の例については、次を参照してください。「[OpenShift での Apicurio Registry ヘルスチェックの設定](#)」
- Apicurio Registry のカスタム認証の設定に関する詳細は、[Quarkus Open ID Connect のドキュメント](#)を参照してください。

5.4. OPENSIFT クラスター内から APICURIO REGISTRY への HTTPS 接続の設定

以下の手順では、OpenShift クラスター内から HTTPS 接続のポートを公開するように Apicurio Registry デプロイメントを設定する方法を説明します。



警告

このような接続は、クラスター外部で直接利用できません。ルーティングはホスト名に基づいており、HTTPS 接続の場合はエンコードされます。そのため、エッジターミネーションまたはその他の設定は必要です。[「OpenShift クラスター外から Apicurio Registry への HTTPS 接続の設定」](#) を参照してください。

前提条件

- Service Registry Operator がインストールされている。

手順

1. 自己署名証明書を使用して **keystore** を生成します。独自の証明書を使用している場合は、この手順を省略できます。

```
keytool -genkey -trustcacerts -keyalg RSA -keystore registry-keystore.jks -storepass password
```

2. キーストアおよびキーストアのパスワードを保持する新しいシークレットを作成します。
 - a. OpenShift Web コンソールの左側のナビゲーションメニューで、**Workloads > Secrets > Create Key/Value Secret** とクリックします。
 - b. 次の値を使用します。

Name: **registry-keystore**

Key 1: **keystore.jks**

Value 1: **registry-keystore.jks** (アップロードされたファイル)

Key 2: **password**

Value 2: **password**



注記

java.io.IOException: Invalid keystore format が発生した場合、バイナリーファイルのアップロードは正しく機能しませんでした。別の方法として、**cat registry-keystore.jks | base64 -w0 > data.txt** を使用してファイルを base64 文字列にエンコードし、yaml として **Secret** リソースを編集してエンコードされたファイルを手動で追加してください。

3. Apicurio Registry インスタンスの **Deployment** リソースを編集します。Service Registry Operator の status フィールドで正しい名前を見つけることができます。
 - a. キーストアシークレットをボリュームとして追加します。

```
template:
  spec:
    volumes:
      - name: registry-keystore-secret-volume
        secret:
          secretName: registry-keystore
```

- b. ボリュームマウントを追加します。

```
volumeMounts:
  - name: registry-keystore-secret-volume
    mountPath: /etc/registry-keystore
    readOnly: true
```

- c. **JAVA_OPTIONS** および **KEYSTORE_PASSWORD** 環境変数を追加します。

```
- name: KEYSTORE_PASSWORD
  valueFrom:
    secretKeyRef:
      name: registry-keystore
      key: password
- name: JAVA_OPTIONS
  value: >-
    -Dquarkus.http.ssl.certificate.key-store-file=/etc/registry-keystore/keystore.jks
    -Dquarkus.http.ssl.certificate.key-store-file-type=jks
    -Dquarkus.http.ssl.certificate.key-store-password=$(KEYSTORE_PASSWORD)
```



注記

順序は、文字列の補間を使用する場合に重要です。

- d. HTTPS ポートを有効にします。

```
ports:
  - containerPort: 8080
    protocol: TCP
  - containerPort: 8443
    protocol: TCP
```

4. Apicurio Registry インスタンスの **Service** リソースを編集します。Service Registry Operator の status フィールドで正しい名前を見つけることができます。

```
ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 8080
  - name: https
    protocol: TCP
    port: 8443
    targetPort: 8443
```

5. 接続が機能していることを確認します。

- a. SSH を使用してクラスターの Pod に接続します (Apicurio Registry Pod を使用できます)。

```
oc rsh -n default example-apicurioregistry-deployment-vx28s-4-lmtqb
```

- b. **Service**リソースから Apicurio Registry Pod のクラスター IP を見つけます (Web コンソールの **Location** 列を参照)。その後、テスト要求を実行します (自己署名証明書を使用するので、セキュアでないフラグが必要になります)。

```
curl -k https://172.30.209.198:8443/health
[...]
```

5.5. OPENSIFT クラスター外から APICURIO REGISTRY への HTTPS 接続の設定

以下の手順では、OpenShift クラスター外からの接続に対して HTTPS エッジターミネーションを使用したルートを開示するために Apicurio Registry デプロイメントを設定する方法を説明します。

前提条件

- Service Registry Operator がインストールされている。
- [セキュアなルートを作成するための OpenShift ドキュメント](#) を読む。

手順

1. Service Registry Operator によって作成される HTTP ルートの他に、2 つ目の **Route** を追加します。以下の例を参照してください。

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  [...]
labels:
  app: example-apicurioregistry
  [...]
spec:
  host: example-apicurioregistry-default.apps.example.com
  to:
    kind: Service
    name: example-apicurioregistry-service-9whd7
    weight: 100
  port:
    targetPort: 8080
  tls:
    termination: edge
    insecureEdgeTerminationPolicy: Redirect
    wildcardPolicy: None
```



注記

`insecureEdgeTerminationPolicy: Redirect` 設定プロパティーが設定されていることを確認してください。

証明書を指定しない場合、OpenShift はデフォルトを使用します。または、以下のコマンドを使用してカスタムの自己署名証明書を生成することもできます。

```
openssl genrsa 2048 > host.key &&  
openssl req -new -x509 -nodes -sha256 -days 365 -key host.key -out host.cert
```

次に、OpenShift CLI を使用してルートを作成します。

```
oc create route edge \  
  --service=example-apicurioregistry-service-9whd7 \  
  --cert=host.cert --key=host.key \  
  --hostname=example-apicurioregistry-default.apps.example.com \  
  --insecure-policy=Redirect \  
  -n default
```

第6章 APICURIO REGISTRY デプロイメントの設定と管理

本章では、OpenShift での Apicurio Registry デプロイメントのオプションの設定および管理方法について説明します。

- [「OpenShift での Apicurio Registry ヘルスチェックの設定」](#)
- [「Apicurio Registry ヘルスチェックの環境変数」](#)
- [「Apicurio Registry 環境変数の管理」](#)
- [「Apicurio Registry Web コンソールを設定する」](#)
- [「Apicurio Registry ログの設定」](#)
- [「Apicurio Registry イベントソーシングの設定」](#)

6.1. OPENSHIFT での APICURIO REGISTRY ヘルスチェックの設定

liveness および readiness プローブのオプションの環境変数を設定して、OpenShift の Apicurio Registry サーバーの健全性を監視できます。

- アプリケーションが進行可能な場合は **liveness プローブ** のテスト。アプリケーションが進行不可能な場合、OpenShift は障害のある Pod を自動的に再起動します。
- アプリケーションが要求を処理する準備ができている場合は **readiness プローブ** のテスト。アプリケーションが準備できていない場合、リクエストに圧倒されてしまい、プローブが失敗した期間は OpenShift がリクエストの送信を停止します。他の Pod が OK の場合は、引き続き要求を受け取ります。



重要

liveness および readiness 環境変数のデフォルト値はほとんどの場合を想定して設計されており、環境で必要とされる場合にのみ変更する必要があります。デフォルトへの変更は、ハードウェア、ネットワーク、および保存されたデータ量によって異なります。これらの値は、不要なオーバーヘッドを回避するために、できるだけ低く抑える必要があります。

前提条件

- クラスター管理者として OpenShift クラスターにアクセスできる。
- OpenShift に Apicurio Registry がすでにインストールされている必要があります。
- AMQ Streams または PostgreSQL で選択した Apicurio Registry ストレージがインストールされ、設定されている。

手順

1. OpenShift Container Platform Web コンソールで、クラスター管理者権限を持つアカウントを使用してログインします。
2. **Installed Operators > Red Hat Integration - Service Registry Operator** をクリックします。

3. **ApicurioRegistry** タブで、**example-apicurioregistry** などのデプロイメントの Operator カスタムリソースをクリックします。
4. メインの概要ページで、**Deployment Name** セクションと Apicurio Registry デプロイメントの対応する **DeploymentConfig** 名を見つけます (例: **example-apicurioregistry**)。
5. 左側のナビゲーションメニューで **Workloads > Deployment Configs** をクリックし、**DeploymentConfig** 名を選択します。
6. **Environment** タブをクリックして、**Single values env** セクションに環境変数を入力します。以下に例を示します。
 - **NAME: LIVENESS_STATUS_RESET**
 - **VALUE: 350**
7. 下部にある **Save** をクリックします。
代わりに、OpenShift **oc** コマンドを使用して、これらの手順を実行することもできます。詳細は、[OpenShift CLI のドキュメント](#) を参照してください。

関連情報

- [「Apicurio Registry ヘルスチェックの環境変数」](#)
- [Monitoring application health](#)

6.2. APICURIO REGISTRY ヘルスチェックの環境変数

このセクションでは、OpenShift の Apicurio Registry ヘルスチェックに使用できる環境変数について説明します。これには、OpenShift 上の Apicurio Registry サーバーの健全性を監視する liveness および readiness プローブが含まれます。手順の例は、[「OpenShift での Apicurio Registry ヘルスチェックの設定」](#) を参照してください。



重要

以下の環境変数は参考としてのみ提供されます。デフォルト値はほとんどの場合を想定して設計されており、環境に必要な場合のみ変更する必要があります。デフォルトへの変更は、ハードウェア、ネットワーク、および保存されたデータ量によって異なります。これらの値は、不要なオーバーヘッドを回避するために、できるだけ低く抑える必要があります。

liveness 環境変数

表6.1 Apicurio Registry liveness プローブの環境変数

名前	説明	型	デフォルト
LIVENESS_ERROR_THRESHOLD	liveness プロブが失敗するまでに発生する可能性のある liveness の問題またはエラーの数。	Integer	1

名前	説明	型	デフォルト
LIVENESS_COUNTER_RESET	しきい値となる数のエラーが発生する期間。たとえば、この値が 60 でしきい値が 1 の場合、1 分間に 2 件のエラーが発生するとチェックが失敗します。	秒	60
LIVENESS_STATUS_RESET	liveness プローブが OK ステータスにリセットされるために、エラーなしで経過する必要のある秒数。	秒	300
LIVENESS_ERRORS_IGNORED	無視された liveness 例外のコンマ区切りリスト。	文字列	io.grpc.StatusRuntimeException,org.apache.kafka.streams.errors.InvalidStateStoreException



注記

OpenShift は liveness チェックに失敗した Pod を自動的に再起動するため、liveness 設定は readiness 設定とは異なり、OpenShift 上の Apicurio Registry の動作に直接影響を与えません。

readiness 環境変数

表6.2 Apicurio Registry readiness プローブの環境変数

名前	説明	型	デフォルト
READINESS_ERROR_THRESHOLD	readiness プローブが失敗するまでに発生する可能性のある readiness の問題またはエラーの数。	Integer	1
READINESS_COUNTER_RESET	しきい値となる数のエラーが発生する期間。たとえば、この値が 60 でしきい値が 1 の場合、1 分間に 2 件のエラーが発生するとチェックが失敗します。	秒	60
READINESS_STATUS_RESET	liveness プローブが OK ステータスにリセットされるために、エラーなしで経過する必要のある秒数。ここでは、Pod が通常の動作に戻るまでの準備ができていない状態の期間を意味します。	秒	300

名前	説明	型	デフォルト
READINESS_TIMEOUT	<p>readiness は 2 つの操作のタイムアウトを追跡します。</p> <ul style="list-style-type: none"> ストレージリクエストが完了するまでの時間 HTTP REST API リクエストが応答を返すまでの時間 <p>これらの操作に設定されたタイムアウトよりも時間がかかった場合、これは readiness 問題またはエラーとしてカウントされます。この値は、両方の操作のタイムアウトを制御します。</p>	秒	5

関連情報

- 「[OpenShift での Apicurio Registry ヘルスチェックの設定](#)」
- [Monitoring application health](#)

6.3. APICURIO REGISTRY 環境変数の管理

Service Registry Operator は最も一般的な Apicurio Registry 設定を管理しますが、手動で調整できるオプションがいくつかあります。Apicurio Registry **Deployment** リソースに環境変数を設定することで、これらを更新することができます。**ApicurioRegistry** CR で特定の設定オプションが使用できない場合は、環境変数を使用して調整できます。

手順

OpenShift Web コンソール

1. **Installed Operators** タブを選択してから、**Red Hat Integration - Service Registry Operator** を選択します。
2. **ApicurioRegistry** タブで、Apicurio Registry デプロイメントの **ApicurioRegistry** CR をクリックします。
3. メインの概要ページで、Apicurio Registry インスタンスをデプロイするために Operator によって管理される **Deployment** の名前が含まれる **managedResources** セクションを表示します。
4. 左側のメニューの **Workloads > Deployments** でその **Deployment** を見つけます。
5. 正しい名前の **Deployment** を選択し、**Environment** タブを選択します。
6. 環境変数を **Single values (env)** セクションに追加または変更できます。
7. 下部にある **Save** をクリックします。

OpenShift CLI

1. Apicurio Registry がインストールされているプロジェクトを選択します。

2. **oc get apicurioregistry** を実行して、**ApicurioRegistry** CR のリストを取得します
3. 設定する Apicurio Registry インスタンスを表す CR で **oc describe** を実行します。
4. **status** セクションで **managedResource** を表示します。
5. その **Deployment** を見つけて、**oc edit** と入力します。
6. **spec.template.spec.containers[0].env** セクションで環境変数を追加または変更します。

6.4. APICURIO REGISTRY WEB コンソールを設定する

オプションの環境変数を設定して、デプロイメント環境専用に Apicurio Registry Web コンソールを設定したり、その動作をカスタマイズしたりできます。

前提条件

- Apicurio Registry はすでにインストールされています。

Web コンソールのデプロイメント環境の設定

ブラウザで Apicurio Registry Web コンソールにアクセスすると、いくつかの初期設定が読み込まれます。次の設定が重要です。

- コア Apicurio Registry サーバー REST API の URL
- Apicurio Registry Web コンソールクライアントの URL

通常、Apicurio Registry はこれらの設定を自動的に検出して生成しますが、一部のデプロイメント環境ではこの自動検出が失敗する場合があります。その場合には、環境のこれらの URL を明示的に設定するように環境変数を設定できます。

手順

以下の環境変数を設定し、デフォルトの URL を上書きします。

- **REGISTRY_UI_CONFIG_APIURL**: コア Apicurio Registry サーバー REST API の URL を指定します。例: **https://registry.my-domain.com/apis/registry**
- **REGISTRY_UI_CONFIG_UIURL**: Apicurio Registry Web コンソールクライアントの URL を指定します。たとえば、**https://registry.my-domain.com/ui**

読み取り専用モードでの Web コンソールの設定

オプション機能として、Apicurio Registry の Web コンソールを読み取り専用モードに設定することができます。このモードでは、Apicurio Registry Web コンソールでユーザーが登録されたアーティファクトを変更できる機能がすべて無効になります。たとえば、これには以下が含まれます。

- アーティファクトの作成
- 新しいアーティファクトバージョンのアップロード
- アーティファクトメタデータの更新
- アーティファクトの削除

手順

次の環境変数を設定します。

- **REGISTRY_UI_FEATURES_READONLY: true** に設定すると、読み取り専用モードが有効になります。デフォルトは **false** です。

6.5. APICURIO REGISTRY ログの設定

実行時に Apicurio Registry のログ設定を設定できます。Apicurio Registry は、詳細なロギングのために特定のロガーのログレベルを設定する REST エンドポイントを提供します。本セクションでは、Apicurio Registry /**admin** REST API を使用して、実行時に Apicurio Registry ログレベルを表示および設定する方法を説明します。

前提条件

- Apicurio Registry インスタンスにアクセスするための URL を取得するか、OpenShift にデプロイした場合に Apicurio Registry ルートを取得します。この簡単な例では、**localhost:8080** の URL を使用しています。

手順

1. この **curl** コマンドを使用して、ロガー **io.apicurio.registry.storage** の現在のログレベルを取得します。

```
$ curl -i localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"INFO"}
```

2. この **curl** コマンドを使用して、ロガー **io.apicurio.registry.storage** のログレベルを **DEBUG** に変更します。

```
$ curl -X PUT -i -H "Content-Type: application/json" --data '{"level":"DEBUG"}'
localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"DEBUG"}
```

3. この **curl** コマンドを使用して、ロガー **io.apicurio.registry.storage** のログレベルをデフォルト値に戻します。

```
$ curl -X DELETE -i localhost:8080/apis/registry/v2/admin/loggers/io.apicurio.registry.storage
HTTP/1.1 200 OK
[...]
Content-Type: application/json
{"name":"io.apicurio.registry.storage","level":"INFO"}
```

6.6. APICURIO REGISTRY イベントソーシングの設定

Apicurio Registry は、変更がレジストリーに加えられたときにイベントを送信するように設定できます。たとえば、Apicurio Registry は、スキーマおよび API アーティファクトが作成、更新、削除された場合などにイベントをトリガーできます。このように、イベントをアプリケーションおよびサードパー

ティーのインテグレーションに送信するように Apicurio Registry を設定できます。

イベントの転送に使用できるさまざまなプロトコルがあります。現在実装されているプロトコルは HTTP および Apache Kafka です。ただし、プロトコルに関係なく、イベントは CNCF CloudEvents 仕様を使用して送信されます。

すべてのイベントタイプは、**io.apicurio.registry.events.dto.RegistryEventType** で定義されています。たとえば、イベントタイプには次のものがあります。

- **io.apicurio.registry.artifact-created**
- **io.apicurio.registry.artifact-updated**
- **io.apicurio.registry.artifact-rule-created**
- **io.apicurio.registry.global-rule-created**

Java システムプロパティーまたは同等の環境変数を使用して、Apicurio Registry でクラウドイベントを設定できます。

前提条件

- Apicurio Registry クラウドイベントの送信先となるアプリケーションが必要です。たとえば、カスタムアプリケーションやサードパーティーアプリケーションなどです。

HTTP を使用した Apicurio Registry イベントソーシングの設定

このセクションの例は、**http://my-app-host:8888/events** で実行されているカスタムアプリケーションを示しています。

手順

1. HTTP プロトコルを使用する場合は、以下のようにイベントをアプリケーションに送信するように Apicurio Registry を設定します。
 - **registry.events.sink.my-custom-consumer=http://my-app-host:8888/events**
2. 必要に応じて、複数のイベントコンシューマーを以下のように設定できます。
 - **registry.events.sink.my-custom-consumer=http://my-app-host:8888/events**
 - **registry.events.sink.other-consumer=http://my-consumer.com/events**

Apache Kafka を使用した Apicurio Registry イベントソーシングの設定

このセクションの例では、**my-registry-events** という名前の Kafka トピックが **my-kafka-host:9092** で動作していることを示します。

手順

1. Kafka プロトコルを使用する場合は、以下のように Kafka トピックを設定します。
 - **registry.events.kafka.topic=my-registry-events**
2. **KAFKA_BOOTSTRAP_SERVERS** 環境変数を使用して、Kafka プロデューサーを設定できます。
 - **KAFKA_BOOTSTRAP_SERVERS=my-kafka-host:9092**

または、**registry.events.kafka.config** 接頭辞を使用して kafka プロデューサーのプロパティを設定できます。例: **registry.events.kafka.config.bootstrap.servers=my-kafka-host:9092**

3. 必要に応じて、イベントの生成に使用する Kafka トピックパーティションを設定することもできます。

- **registry.events.kafka.topic-partition=1**

関連情報

- 詳細は、[CNCF CloudEvents 仕様](#) を参照してください。

第7章 SERVICE REGISTRY OPERATOR の設定リファレンス

本章では、Service Registry Operator をデプロイするように Apicurio Registry を設定するために使用されるカスタムリソースの詳細情報を提供します。

- [「Apicurio Registry カスタムリソース」](#)
- [「Apicurio Registry CR スペック」](#)
- [「Apicurio レジストリーの CR ステータス」](#)
- [「Apicurio Registry が管理するリソース」](#)
- [「Service Registry Operator ラベル」](#)

7.1. APICURIO REGISTRY カスタムリソース

Service Registry Operator は、OpenShift 上の Apicurio Registry の単一デプロイメントを表す **ApicurioRegistry** custom resource (CR) を定義します。

これらのリソースオブジェクトはユーザーによって作成および維持され、Apicurio Registry のデプロイおよび設定方法を Service Registry Operator に指示します。

ApicurioRegistry CR の例

次のコマンドは、**ApicurioRegistry** リソースを表示します。

```
oc get apicurioregistry
oc edit apicurioregistry example-apicurioregistry
```

```
apiVersion: registry.apicur.io/v1
kind: ApicurioRegistry
metadata:
  name: example-apicurioregistry
  namespace: demo-kafka
  # ...
spec:
  configuration:
    persistence: kafkasql
    kafkasql:
      bootstrapServers: 'my-cluster-kafka-bootstrap.demo-kafka.svc:9092'
  deployment:
    host: >-
      example-apicurioregistry.demo-kafka.example.com
status:
  conditions:
  - lastTransitionTime: "2021-05-03T10:47:11Z"
    message: ""
    reason: Reconciled
    status: "True"
    type: Ready
  info:
    host: example-apicurioregistry.demo-kafka.example.com
  managedResources:
  - kind: Deployment
```

```

name: example-apicurioregistry-deployment
namespace: demo-kafka
- kind: Service
  name: example-apicurioregistry-service
  namespace: demo-kafka
- kind: Ingress
  name: example-apicurioregistry-ingress
  namespace: demo-kafka

```



重要

デフォルトで、Service Registry Operator は独自のプロジェクト namespace のみを監視します。したがって、operator を手動でデプロイする場合は、同じ namespace に **ApicurioRegistry** CR を作成する必要があります。Operator **Deployment** リソースの **WATCH_NAMESPACE** 環境変数を更新することで、この動作を修正することができます。

関連情報

- [Extending the Kubernetes API with Custom Resource Definitions](#)

7.2. APICURIO REGISTRY CR スペック

spec は、オペレーターがアーカイブするための望ましい状態または設定を提供するために使用される **ApicurioRegistry** CR の一部です。

ApicurioRegistry CR 仕様コンテンツ

以下のブロック例には、可能な **spec** 設定オプションの完全なツリーが含まれます。フィールドによっては、必須ではないものや、同時に定義してはいけないものもあります。

```

spec:
  configuration:
    persistence: <string>
    sql:
      dataSource:
        url: <string>
        userName: <string>
        password: <string>
      kafkasql:
        bootstrapServers: <string>
    security:
      tls:
        truststoreSecretName: <string>
        keystoreSecretName: <string>
      scram:
        mechanism: <string>
        truststoreSecretName: <string>
        user: <string>
        passwordSecretName: <string>
    ui:
      readOnly: <string>
    logLevel: <string>
    security:
      keycloak:

```



```

url: <string>
realm: <string>
apiClientId: <string>
uiClientId: <string>
deployment:
  replicas: <int32>
  host: <string>
  affinity: <k8s.io/api/core/v1 Affinity struct>
  tolerations: <k8s.io/api/core/v1 []Toleration slice>

```

以下の表は、各設定オプションについて説明しています。

表7.1 ApicurioRegistry CR 仕様設定オプション

設定オプション	型	デフォルト値	説明
configuration	-	-	Apicurio Registry アプリケーションの設定セクション
configuration/persistence	string	必須	ストレージバックエンド。 sql の1つ、 kafkasql
configuration/sql	-	-	SQL ストレージバックエンドの設定
configuration/sql/dataSource	-	-	SQL ストレージバックエンドのデータベース接続設定
configuration/sql/dataSource/url	string	必須	データベース接続 URL 文字列
configuration/sql/dataSource/username	string	必須	データベース接続ユーザー
configuration/sql/dataSource/password	string	空	データベース接続パスワード
configuration/kafkasql	-	-	Kafka ストレージバックエンドの設定
configuration/kafkasql/bootstrapServers	string	必須	Streams ストレージバックエンドの Kafka ブートストラップサーバー URL。
configuration/kafkasql/security/tls	-	-	Kafka ストレージバックエンドの TLS 認証を設定するセクション。

設定オプション	型	デフォルト値	説明
configuration/kafkaql/security/tls/truststoreSecretName	string	必須	Kafka の TLS トラストストアが含まれるシークレットの名前
configuration/kafkaql/security/tls/keystoreSecretName	string	必須	ユーザー TLS キーストアを含むシークレットの名前
configuration/kafkaql/security/scram/truststoreSecretName	string	必須	Kafka の TLS トラストストアが含まれるシークレットの名前
configuration/kafkaql/security/scram/user	string	必須	SCRAM ユーザー名
configuration/kafkaql/security/scram/passwordSecretName	string	必須	SCRAM ユーザーパスワードが含まれるシークレットの名前
configuration/kafkaql/security/scram/mechanism	string	SCRAM-SHA-512	SASL メカニズム
configuration/ui	-	-	Apicurio Registry Web コンソールの設定
configuration/ui/readOnly	string	false	Apicurio Registry Web コンソールを読み取り専用モードに設定する
configuration/logLevel	string	INFO	Apicurio レジストリーのログレベル。 INFO の 1 つ、 DEBUG
configuration/security	-	-	Apicurio Registry Web コンソールと REST API のセキュリティ設定
configuration/security/keycloak	-	-	Keycloak を使用した Web コンソールおよび REST API セキュリティ設定
configuration/security/keycloak/url	string	必須	Keycloak URL、 /auth で終わる必要があります
configuration/security/keycloak/realm	string	必須	Keycloak レalm

設定オプション	型	デフォルト値	説明
configuration/security/keycloak/apiClientId	string	registry-client-api	REST API の Keycloak クライアント
configuration/security/keycloak/uiClientId	string	registry-client-ui	Web コンソールの Keycloak クライアント
deployment	-	-	Apicurio Registry デプロイメント設定のセクション
deployment/replicas	正の整数	1	デプロイする Apicurio Registry Pod の数
deployment/host	string	自動生成	Apicurio Registry コンソールと API が利用可能なホスト/URL。可能な場合は、Service Registry Operator はクラスタールーターの設定に基づいて正しい値を判別しようとします。値は一度だけ自動生成されるため、ユーザーは後で上書きすることができます。
deployment/affinity	k8s.io/api/core/v1 Affinity struct	空	Apicurio Registry デプロイメントアフィニティー設定
deployment/tolerations	k8s.io/api/core/v1 []Toleration slice	空	Apicurio レジストリーのデプロイ容認の設定



注記

オプションが **必須** とされている場合は、有効になっている他の設定オプションの条件である可能性があります。空の値は受け入れられる可能性がありますが、Operator は指定されたアクションを実行しません。

7.3. APICURIO レジストリーの CR ステータス

status は、Service Registry Operator によって管理される CR のセクションであり、現在のデプロイメントとアプリケーションの状態の説明が含まれています。

ApicurioRegistry CR ステータスのコンテンツ

status セクションには、次のフィールドが含まれています。

```
status:
  info:
    host: <string>
```

```

conditions: <list of:>
- type: <string>
  status: <string, one of: True, False, Unknown>
  reason: <string>
  message: <string>
  lastTransitionTime: <string, RFC-3339 timestamp>
managedResources: <list of:>
- kind: <string>
  namespace: <string>
  name: <string>

```

表7.2 ApicurioRegistry CR status フィールド

status フィールド	型	説明
info	-	デプロイされた Apicurio Registry に関する情報が含まれるセクション。
info/host	string	Apicurio Registry UI および REST API にアクセスできる URL。
conditions	-	Apicurio Registry のステータス、またはそのデプロイメントに関連した operator のステータスを報告する条件の一覧。
conditions/type	string	条件の型。
conditions/status	string	状態のステータス、 True 、 False 、 Unknown のいずれか。
conditions/reason	string	条件の最後の遷移の理由を示すプログラムによる識別子。
conditions/message	string	遷移の詳細を示す人が判読できるメッセージ。
conditions/lastTransitionTime	string	最後にある状態から別の状態に遷移した時間。
managedResources	-	Service Registry Operator によって管理される OpenShift リソースの一覧
managedResources/kind	string	リソースの種類。
managedResources/namespace	string	リソースの namespace。
managedResources/name	string	リソース名。

7.4. APICURIO REGISTRY が管理するリソース

Apicurio Registry のデプロイ時に Service Registry Operator によって管理されるリソースは次のとおりです。

- デプロイメント
- サービス
- Ingress (および Route)
- PodDisruptionBudget

7.5. SERVICE REGISTRY OPERATOR ラベル

通常 Service Registry Operator によって管理されるリソースには、以下のようにラベルが付けられます。

表7.3 管理されたリソースの Service Registry Operator ラベル

ラベル	説明
app	指定された ApicurioRegistry CR の名前に基づく、リソースが属する Apicurio Registry デプロイメントの名前。
apicur.io/type	デプロイメントのタイプ: apicurio-registry または operator
apicur.io/name	デプロイの名前: app または apicurio-registry-operator と同じ値
apicur.io/version	Apicurio Registry または Service Registry Operator のバージョン
app.kubernetes.io/*	アプリケーションのデプロイメントに推奨される Kubernetes ラベルのセット。
com.company および rh.*	Red Hat 製品のメタリングラベル。

関連情報

- [アプリケーションデプロイメントに推奨される Kubernetes ラベル](#)

付録A サブスクリプションの使用

Apicurio Registry は、ソフトウェアサブスクリプションから提供されます。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. **My Subscriptions** 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、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. **Integration and Automation** カテゴリで **Red Hat Integration** エントリーを見つけます。
3. 目的の Apicurio Registry 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 **Download** 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

Red Hat Enterprise Linux に RPM パッケージをインストールするには、システムを登録する必要があります。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。