Red Hat Integration 2021.Q2

OpenShift での Service Registry のインストールおよびデプロイ

Service Registry 2.0 - テクノロジープレビュー

Last Updated: 2022-12-04

Red Hat Integration 2021.Q2 OpenShift での Service Registry のインストールおよびデプロイ

Service Registry 2.0 - テクノロジープレビュー

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

法律上の通知

概要

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 Service 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、Service Registry のデプロイメントおよびセキュリティーを設定および管理する方法について説明します。また、Service Registry Operator についての参照情報を提供します。

目次

前書き

多様性を受け入れるオープンソースの強化

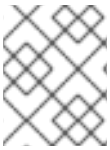
Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 SERVICE REGISTRY OPERATOR クイックスタート

本章では、コマンドラインで Service Registry Operator をすばやくインストールする方法を説明します。

このクイックスタート例では、SQL ストレージオプションを使用して Service Registry をデプロイします。

- 「Quickstart Service Registry Operator のインストール」
- 「Quickstart Service Registry 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 を使用することです。推奨されるストレージオプションは SQL または Kafka 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導入することができます。

前提条件

- Red Hat Integration Downloads に移動し、製品バージョンを選択し、Service Registry CRD **.zip** ファイルの例をダウンロードする必要があります。

手順

1. インストール用のプロジェクトを作成します。たとえば、**service-registry**:

```
NAMESPACE="service-registry"
oc new-project "$NAMESPACE"
```

2. **install/** フォルダーにあるファイルを適用します。

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

1.2. QUICKSTART SERVICE REGISTRY デプロイメント

新しい Service Registry デプロイメントを作成するには、SQL ストレージオプションを使用します。そのためには、外部 PostgreSQL ストレージを前提条件として設定する必要があります。

前提条件

- Service Registry Operator がすでにインストールされていることを確認する。
- OpenShift クラスターから PostgreSQL データベースにアクセスできる。

手順

1. データベース接続を設定して、**ApicurioRegistry** カスタムリソース (CR) を作成します。次に例を示します。

   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. Operator がデプロイされているのと同じ namespace に **ApicurioRegistry** CR を作成します。

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

第2章 OPENSHIFT での SERVICE REGISTRY のインストール

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

- 「OpenShift OperatorHub からの Service Registry のインストール」

前提条件

- Service Registry User Guide の概要を読む。

重要

Service Registry はテクノロジープレビュー機能としてのみ提供されます。テクノロジープレビュー機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。Red Hat は実稼働環境でこれらを使用することを推奨していません。

これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、https://access.redhat.com/ja/support/offerings/techpreview を参照してください。

2.1. OPENSHIFT OPERATORHUB からの SERVICE REGISTRY のインストール

OperatorHub から OpenShift クラスターに Service 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。詳細は、OpenShift のドキュメント を参照してください。

注記

環境に応じて、複数の Service 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、Service 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。
     - **2.0.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自動的に 2.1.x を無視します。
     - **2.x**: 2.1.0 や 2.0.1 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自動的に 2.1.x にアップグレードされます。
   - **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。

その他のリソース

- 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 での SERVICE REGISTRY ストレージのデプロイ

本章では、AMQ Streams で Service Registry データストレージをインストールおよび設定する方法を説明します。

- 「OpenShift OperatorHub からの AMQ Streams のインストール:」
- 「OpenShift での Kafka ストレージを使用した Service Registry の設定」
- 「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」

前提条件

- *2章OpenShift での Service Registry のインストール*

3.1. OPENSHIFT 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。詳細は、OpenShift のドキュメント 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 のインストールに関する詳細は、AMQ Streams on OpenShift の使用 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** および **amq-streams-1.7.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。

その他のリソース

- Operator の OpenShift クラスターへの追加
- AMQ Streams on OpenShift の使用

3.2. OPENSHIFT での KAFKA ストレージを使用した SERVICE REGISTRY の設定

ここでは、AMQ Streams on OpenShift を使用して Service Registry に Kafka ベースのストレージを設定する方法を説明します。**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 Service 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。
- Service Registry がすでにインストールされている。*2章OpenShift での Service Registry のインストール* 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。詳細は、Using AMQ Streams on OpenShift を参照してください。

3. クラスターの準備ができたら、**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 ServiceRegistry 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** とクリックします。

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、OpenShift で Service Registry ルートが作成されるまで待機します。

8. **Networking** > **Route** をクリックして、Service Registry Web コンソールの新規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/
```

関連情報

- AMQ Streams を使用した Kafka クラスターおよびトピックの作成に関する詳細は、AMQ Streams on OpenShift の使用 を参照してください。

3.3. TLS セキュリティーでの 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 をインストールする。
- AMQ Streams Operator をインストールする、または 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、Service 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
```

ServiceRegistry がデータの保存に使用するデフォルトの Kafka トピック名は **kafkasql-journal** です。このトピックは Service Registry によって自動的に作成されます。適切な環境変数 (デフォルト値) を設定して、この動作またはデフォルトのトピック名を上書きできます。

- **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:
    retention.ms: 604800000
    segment.bytes: 1073741824
```

5. **Kafka User** リソースを作成し、Service 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** を使用できます。

```
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

Service Registry が必要とするトピックおよびリソースに合わせて承認を設定する必要があります。これは、単純な許容例です。

6. **Workloads** をクリックしてから **Secrets** をクリックし、Service Registry が Kafka クラスターに接続するために AMQ Stremas によって作成される 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. Service 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
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

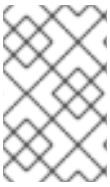
プレーンでセキュアでないユースケースとは別の **bootstrapServers** アドレスを使用する必要があります。アドレスは TLS 接続をサポートする必要があり、指定された **Kafka** リソースの **type:tls** フィールドにあります。

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 をインストールする。
- AMQ Streams Operator をインストールする、または 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、Service 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-scram
  # 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
```

ServiceRegistry がデータの保存に使用するデフォルトの Kafka トピック名は **kafkasql-journal** です。このトピックは Service Registry によって自動的に作成されます。適切な環境変数 (デフォルト値) を設定して、この動作またはデフォルトのトピック名を上書きできます。

- **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。

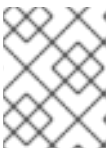
4. **Kafka Topic** タブをクリックし、**Create Kafka Topic** をクリックして、**kafkasql-journal** トピックを作成します。

```
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:
    retention.ms: 604800000
    segment.bytes: 1073741824
```

5. **Kafka User** リソースを作成し、Service 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-scram
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

Service Registry が必要とするトピックおよびリソースに合わせて承認を設定する必要があります。これは、単純な許容例です。

6. **Workloads** をクリックしてから **Secrets** をクリックし、Service Registry が Kafka クラスターに接続するために AMQ Stremas によって作成される 2 つのシークレットを見つけます。

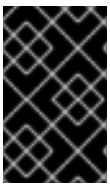
- **my-cluster-cluster-ca-cert** - Kafka クラスターの PKCS12 トラストストアが含まれています
- **my-user** - ユーザーのキーストアが含まれます

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. Service 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
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。

第4章 POSTGRESQL データベースでの SERVICE REGISTRY ストレージのデプロイ

本章では、PostgreSQL データベースで Service 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 Service Registry の設定」
- 「Service Registry PostgreSQL ストレージのバックアップ」
- 「Service Registry PostgreSQL ストレージの復元」

前提条件

- *2章OpenShift での Service Registry のインストール*

4.1. OPENSHIFT 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。詳細は、OpenShift のドキュメント 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、ご使用の環境に適した Operator を検索します。たとえば、**Crunchy PostgreSQL for OpenShift** や **PostgreSQL Operator by Dev4Ddevs.com** です。
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**: **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。

その他のリソース

- Operator の OpenShift クラスターへの追加
- Crunchy PostgreSQL Operator QuickStart

4.2. OPENSHIFT での POSTGRESQL データベースストレージを使用した SERVICE 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 Service 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 Service 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。
- Service Registry がすでにインストールされている。*2章OpenShift での Service Registry のインストール* 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。(例: 「OpenShift OperatorHub からの PostgreSQL データベースのインストール」)。

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. Service 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. Service 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** とクリックします。
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、OpenShift で Service Registry ルートが作成されるまで待機します。
10. **Networking** > **Route** をクリックして、Service Registry Web コンソールの新規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. SERVICE 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、Service 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 ユーティリティーを使用して、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. SERVICE REGISTRY POSTGRESQL ストレージの復元

psql ユーティリティーを使用して、 **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。「Service 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 SERVICE REGISTRY デプロイメントのセキュリティー保護

本章では、OpenShift での Service 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 を使用した Service Registry API および Web コンソールのセキュア化」
- 「Service Registry の認証および承認設定オプション」
- 「OpenShift クラスター内から Service Registry への HTTPS 接続の設定」
- 「OpenShift クラスター外から Service Registry への HTTPS 接続の設定」

Service Registry は、Open ID Connect(OIDC)をベースとした Red Hat Single Sign-On を使用した Service Registry Web コンソールおよびコア REST API の認証をサポートします。これは、Red Hat Single Sign-On Operator をインストールするか、Service Registry 設定オプションを使用して有効にできます。

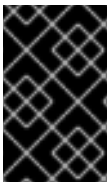
Service Registry は、Red Hat Single Sign-On でロールベースの認証および承認を提供します。Service Registry は、アーティファクト作成者のみが書き込みアクセスを持つスキーマまたは API レベルでコンテンツベースの承認も提供します。OpenShift クラスターの内部または外部から Service Registry への HTTPS 接続を設定することもできます。

その他のリソース

- 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。
  - Service Registry Java client configuration
  - Service Registry serializer/deserializer configuration

5.1. RED HAT SINGLE SIGN-ON を使用した SERVICE REGISTRY API および WEB コンソールのセキュア化

以下の手順では、Red Hat Single Sign-On により保護されるように Service Registry デプロイメントを設定する方法を説明します。



重要

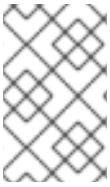
この手順の設定例は、開発およびテストのみを目的としています。手順を単純にするために、実稼働環境で推奨される HTTPS やその他のセキュリティーは使用しません。詳細は、Red Hat Single Sign-On のドキュメントを参照してください。

Service Registry は以下のユーザーロールをサポートします。

表5.1 Service Registry ユーザーロール

名前	機能
sr-admin	完全なアクセス。制限はありません。

名前	機能
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。
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。

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、Service Registry デプロイメントのセキュリティーを保護するために、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。

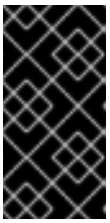
```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: example-keycloak
  labels:
    app: sso
spec:
  instances: 1
  externalAccess:
    enabled: True
  podDisruptionBudget:
    enabled: True
```

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 Service 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** のサンプルレルムを作成します。

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  name: registry-keycloakrealm
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/v1beta1
kind: Ingress
metadata:
  name: keycloak-http
  labels:
    app: keycloak
spec:
  rules:
    - host: keycloak-http.local
```

```
    http:
      paths:
        - path: /
          pathType: ImplementationSpecific
          backend:
            serviceName: keycloak-http
            servicePort: 8080
```

host の値を変更して、Service 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.<namespace>.svc:9092'
```

5.2. SERVICE REGISTRY の認証および承認設定オプション

本セクションでは、Red Hat Single Sign-On を使用した Service Registry の認証および承認オプションについて説明します。

Red Hat Single Sign-On を使用して、Service 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 レルムとユーザーは Open ID Connect(OIDC)を使用して Service 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。

Service 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。Service 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。Service 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。

前提条件

- Red Hat Single Sign-On がインストールされ、稼働中で、Red Hat Single Sign-On レルムおよびユーザーが設定されている。詳細は、『Getting Started with Red Hat Single Sign-On 』を参照してください。
- Service Registry がインストールされ、稼働中である。

Red Hat Single Sign-On を使用した Service Registry 認証

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

表5.2 Service 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	**true** に設定すると、従う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 レルム。	String	-
KEYCLOAK_API_CLIENT_ID	Service Registry REST API のクライアント ID。	文字列	**registry-api**
KEYCLOAK_UI_CLIENT_ID	Service Registry Web コンソールのクライアント ID。	文字列	**apicurio-registry**

Red Hat Single Sign-On の Service Registry ユーザーロール

Service Registry の認証が有効になっている場合は、Red Hat Single Sign-On で Service Registry ユーザーを以下のユーザーロールの 1 つ以上に割り当てる必要があります。

表5.3 認証および承認のための Service Registry 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	説明
sr-admin	あり	はい	可能	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	あり	はい	×	グローバルルールの設定やインポート/エクスポートを除く、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のみを設定できます。

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	説明
sr-readonly	あり	いいえ	不可能	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。

Service Registry アーティファクトの所有者のみの承認オプション

以下のオプションを **true** に設定して、Service Registry のスキーマおよび API アーティファクトへの更新に対する所有者のみの承認を有効にします。

表5.4 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ー	型	デフォルト値
REGISTRY_AUTH_OWNER_ONLY_AUTHORIZATION	**registry.auth.owner-only-authorization**	Boolean	**false**

関連情報

- Red Hat Single Sign-On を使用したオープンソースの Docker ベースの認証例は、https://github.com/Apicurio/apicurio-registry/tree/master/distro/docker-compose 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。
- Service 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。

5.3. OPENSHIFT クラスター内から SERVICE 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 Service 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 Service 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. Service 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. Service 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 (Service 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 Service 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
[...]
```

5.4. OPENSHIFT クラスター外から SERVICE 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公開するために Service 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 SERVICE REGISTRY デプロイメントの管理

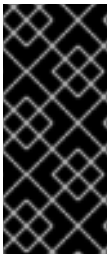
本章では、OpenShift での Service Registry デプロイメントのオプションの設定および管理方法について説明します。

- 「OpenShift での Service Registry ヘルスチェックの設定」
- 「Service Registry ヘルスチェックの環境変数」
- 「Service Registry Web コンソールの設定」
- 「Service Registry ロギングの設定」

6.1. OPENSHIFT での SERVICE 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 Service 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 Service Registry がインストールされている。
- AMQ Streams または PostgreSQL で選択した Service 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**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 Service 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**

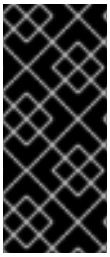
7. 下部にある **Save** をクリックします。
   代わりに、OpenShift **oc** コマンドを使用して、これらの手順を実行することもできます。詳細は、OpenShift CLI のドキュメント を参照してください。

その他のリソース

- 「Service Registry ヘルスチェックの環境変数」
- Monitoring application health

6.2. SERVICE REGISTRY ヘルスチェックの環境変数

このセクションでは、OpenShift の Service 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 Service 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 Service 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 Service Registry liveness プローブの環境変数

名前	詳細	型	デフォルト
LIVENESS_ERROR_THRESHOLD	liveness プロー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**
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**

名前	詳細	型	デフォルト
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 Service Registry の動作に直接影響を与えません。

readiness 環境変数

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

名前	詳細	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	詳細	型	デフォルト
READINESS_TIMEOUT	readiness は 2 つの操作のタイムアウトを追跡します。 ● 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 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。	秒	**5**

関連情報

- 「OpenShift での Service Registry ヘルスチェックの設定」
- Monitoring application health

6.3. SERVICE REGISTRY WEB コンソールの設定

デプロイメント環境専用の Service 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することができます。本セクションでは、Service Registry Web コンソールにオプションの環境変数を設定する方法を説明します。

前提条件

- Service Registry がすでにインストールされている。

Web コンソールのデプロイメント環境の設定

ユーザーがブラウザーで Service Registry Web コンソールに移動すると、一部の初期設定が読み込まれます。以下の 2 つの重要な設定プロパティーがあります。

- バックエンド Service Registry REST API の URL
- フロントエンド Service Registry Web コンソールの URL

通常、Service 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**: バックエンド Service Registry REST API の URL を設定します。たとえば、**https://registry.my-domain.com/apis/registry**
- **REGISTRY_UI_CONFIG_UIURL**: フロントエンド Service Registry Web コンソールの URL を設定します。たとえば、**https://registry.my-domain.com/ui**

読み取り専用モードでのコンソールの設定
オプション機能として、Service 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、Service 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
- アーティファクトの新規バージョンのアップロード
- アーティファクトのメタデータの更新
- アーティファクトの削除

手順

以下の環境変数を設定して、Service Registry の Web コンソールを読み取り専用モードにします。

- **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。

6.4. SERVICE REGISTRY ロギングの設定

実行時に Service Registry のログを設定できます。Service 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、Service Registry /**admin** REST API を使用して、実行時に Service 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

- Service 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 Service 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。

特定ロガーの現在のログレベルの確認

- 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"}
```

特定ロガーのログレベルの変更

- 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"}
```

デフォルトのログレベルに戻す

特定のロガーの設定をデフォルトのログレベルに戻すことができます。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変更します。

```
$ 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"}
```

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 Service 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。

- 「Service Registry カスタムリソース」
- 「Service Registry CR 仕様」
- 「Service Registry CR のステータス」
- 「Service Registry Operator ラベル」
- 「Service Registry 管理リソース」

7.1. SERVICE REGISTRY カスタムリソース

Service Registry Operator は、OpenShift 上の Service 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、Service 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. SERVICE REGISTRY CR 仕様

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	-	-	Service 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**	データベースコネクションユーザー
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/kafkasql/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/kafkasql/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/kafkasql/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/kafkasql/security/scram/user	string	**必須**	SCRAM ユーザー名
configuration/kafkasql/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/kafkasql/security/scram/mechanism	string	**SCRAM-SHA-512**	SASL メカニズム
configuration/ui	-	-	Service Registry Web コンソール設定
configuration/ui/readOnly	string	**false**	Service Registry Web コンソールを読み取り専用モードに設定します。
configuration/logLevel	string	**INFO**	Service Registry のログレベル。**INFO** の 1 つ、**DEBUG**
configuration/security	-	-	Service Registry Web コンソールおよび REST API セキュリティー設定
configuration/security/keycloak	-	-	Keycloak を使用した Web コンソールおよび REST API セキュリティー設定
configuration/security/keycloak/url	string	**required**	Keycloak URL、 **/auth** で終わる必要があります
configuration/security/keycloak/realm	string	**必須**	Keycloak レルム

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	-	-	Service 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 Service Registry Pod 数
deployment/host	string	**自動生成**	Service Registry コンソールおよび API が利用できるホスト/URL。可能な場合は、Service Registry Operator はクラスターの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**	Service Registry デプロイメントのアフィニティー設定
deployment/tolerations	k8s.io/api/core/v1 []Toleration slice	**空**	Service Registry のデプロイメント許容の設定

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. SERVICE REGISTRY CR のステータス

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 Service Registry に関する情報が含まれるセクション。
info/host	string	Service Registry UI および REST API にアクセスできる URL。
conditions	-	Service 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. SERVICE REGISTRY 管理リソース

Service 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以下のとおりです。

- **Deployment**
- **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 Service 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	Service 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 **rht.*`**	Red Hat 製品のメータリングラベル。

その他のリソース

- 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

Service 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. **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、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** カテゴリーで **Red Hat Integration** エントリーを見つけます。
3. 必要な Service 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。