Red Hat Integration 2021.Q2

Service Registry ユーザーガイド

Service Registry 2.0 - テクノロジープレビュー

Last Updated: 2023-02-22

Red Hat Integration 2021.Q2 Service Registry ユーザーガイド

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

このガイドでは、Service Registry を紹介し、Service Registry Web コンソール、REST API、Maven プラグイン、または Java クライアントを使用してイベントスキーマと API 設計を管理する方法について説明します。このガイドでは、Java コンシューマーおよびプロデューサーアプリケーションで Kafka クライアントシリアライザーとデシリアライザーを使用する方法についても説明します。また、サポートされる Service Registry コンテンツタイプおよび任意のルール設定についても説明します。

目次

前書き

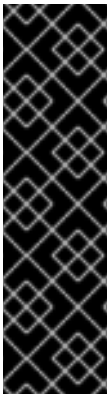
多様性を受け入れるオープンソースの強化

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 の概要

本章では、Service Registry の概念および機能を紹介し、レジストリーに保存されるサポート対象のアーティファクトタイプの詳細を提供します。

- 「Service Registry の概要」
- 「Service Registry のスキーマおよび API アーティファクトおよびグループ」
- 「Service Registry Web コンソールを使用したコンテンツの管理」
- 「Registry core REST API の概要」
- 「Service Registry ストレージのオプション」
- 「Kafka クライアントシリアライザー/デシリアライザーでのスキーマの検証」
- 「Kafka Connect コンバーターを使用した外部システムへのデータのストリーミング」
- 「Service Registry デモ例」
- 「Service Registry で利用可能なディストリビューション」



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。

テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、https://access.redhat.com/ja/support/offerings/techpreview を参照してください。

1.1. SERVICE REGISTRY の概要

Service Registry は、API およびイベント駆動型アーキテクチャー全体で標準的なイベントスキーマおよび API 設計を共有するためのデータストアです。Service Registry を使用して、クライアントアプリケーションからデータの構造を切り離し、REST インターフェイスを使用して実行時にデータ型と API の記述を共有および管理できます。

たとえば、クライアントアプリケーションは、再デプロイせずに最新のスキーマ更新を実行時に Service Registry との間で動的にプッシュまたはプルできます。開発者チームは、すでに実稼働でデプロイされているサービスに必要な既存のスキーマのレジストリーをクエリーでき、新規サービスを開発する際に新しいスキーマを登録できます。

クライアントアプリケーションが、クライアントアプリケーションでレジストリー URL を指定することで、Service Registry に保存されているスキーマおよび API 設計を使用できるようにすることができます。たとえば、レジストリーにはメッセージをシリアライズおよびデシリアライズするために使用されるスキーマを保存できます。その後、クライアントアプリケーションからスキーマを参照して、送受信されるメッセージとこれらのスキーマの互換性を維持することができます。

Service Registry を使用して、アプリケーションからデータ構造を切り離し、メッセージ全体のサイズを減らすことでコストを削減し、組織内のスキーマおよび API 設計の一貫性を高めて効率化します。Service Registry は、開発者および管理者がレジストリーコンテンツの管理を簡単に行えるように Web

コンソールを提供します。

さらに、オプションのルールを設定して、レジストリーコンテンツの展開を管理できます。たとえば、これらには、アップロードされたコンテンツが構文的および意味的に有効であること、または他のバージョンとの上位互換性と下位互換性があることを確認するためのルールが含まれます。設定済みのルールは新規バージョンをレジストリーにアップロードする前に渡す必要があります。これにより、無効または互換性のないスキーマや API 設計に無駄な時間を費やさないようにします。

Service Registry は、Apicurio Registry オープンソースコミュニティープロジェクトに基づいています。詳細は https://github.com/apicurio/apicurio-registry を参照してください。

Service Registry の機能

- 標準イベントスキーマと API 仕様の複数のペイロード形式
- AMQ Streams または PostgreSQL データベースのプラグ可能なレジストリーストレージオプション
- Web コンソール、REST API コマンド、Maven プラグイン、または Java クライアントを使用したレジストリーコンテンツ管理
- レジストリーコンテンツが時間とともにどのように進化するかを管理するためのコンテンツ検証とバージョン互換性のルール
- 外部システム用の Kafka Connect との統合を含む、Apache Kafka スキーマレジストリーの完全なサポート
- 実行時にメッセージタイプを検証する Kafka クライアントシリアライザー/デシリアライザー (Serdes)
- メモリーフットプリントが低く、デプロイメントの時間が高速化されるクラウドネイティブ Quarkus Java ランタイム
- 既存の Confluent または IBM スキーマレジストリークライアントアプリケーションとの互換性
- OpenShift での Service Registry の Operator ベースのインストール
- Red Hat Single Sign-On を使用した OpenID Connect (OIDC) 認証

1.2. SERVICE REGISTRY のスキーマおよび API アーティファクトおよびグループ

イベントスキーマや API 設計などの Service Registry に保存される項目は、レジストリー**アーティファクト**と呼ばれます。以下は、単純な株価アプリケーションの JSON 形式の Apache Avro スキーマアーティファクトの例を示しています。

```
{
  "type": "record",
  "name": "price",
  "namespace": "com.example",
  "fields": [
    {
      "name": "symbol",
      "type": "string"
    },
    {
```

```
        "name": "price",
        "type": "string"
      }
    ]
}
```

スキーマまたは API 設計がレジストリーのアーティファクトとして追加されると、クライアントアプリケーションはそのスキーマまたは API デザインを使用して、実行時にクライアントメッセージが正しいデータ構造に準拠することを確認できます。

Service Registry は、標準のイベントスキーマおよび API 仕様の幅広いメッセージペイロード形式をサポートしています。たとえば、サポートされている形式には、Apache Avro、Google Protobuf、GraphQL、AsyncAPI、OpenAPI などがあります。詳細は *8章Service Registry アーティファクトの参照* を参照してください。

スキーマおよび API グループ

アーティファクトグループ は、スキーマまたは API アーティファクトのオプションの名前付きコレクションです。各グループには、論理的に関連したスキーマまたは API 設計のセットが含まれており、通常、特定のアプリケーションまたは組織に属する単一のエンティティーにより管理されます。

スキーマと API 設計を追加するときに、オプションのアーティファクトグループを作成して、Service Registry でそれらを整理できます。たとえば、**development** および **production** アプリケーション環境、あるいは **sales** および **engineering** 組織に一致するグループを作成できます。

スキーマおよび API グループには複数のアーティファクトタイプを含めることができます。たとえば、同じグループ内に Protobuf、Avro、JSON Schema、OpenAPI、および AsyncAPI スキーマおよび API アーティファクトがあるとします。

Service Registry Web コンソール、コア REST API、Maven プラグイン、または Java クライアントアプリケーションを使用して、スキーマと API アーティファクトおよびオプションのグループを作成できます。以下の例は、REST API を使用する方法を示しています。

```
$ curl -X POST -H "Content-type: application/json; artifactType=AVRO" \
  -H "X-Registry-ArtifactId: share-price" \
  --data '{"type":"record","name":"price","namespace":"com.example", \
   "fields":[{"name":"symbol","type":"string"},{"name":"price","type":"string"}]}' \
  https://my-registry.example.com/apis/registry/v2/groups/my-group/artifacts
```

この例では、**my-group** という名前のアーティファクトグループに **share-price** のアーティファクト ID を持つ Avro スキーマを追加します。

注記

Service Registry Web コンソールを使用する場合にグループの指定は任意です。この場合、**default** グループが自動的に作成されます。v2 REST API または Maven プラグインを使用し、一意のグループを作成したくない場合は API パスで **default** グループを指定できます。

関連情報

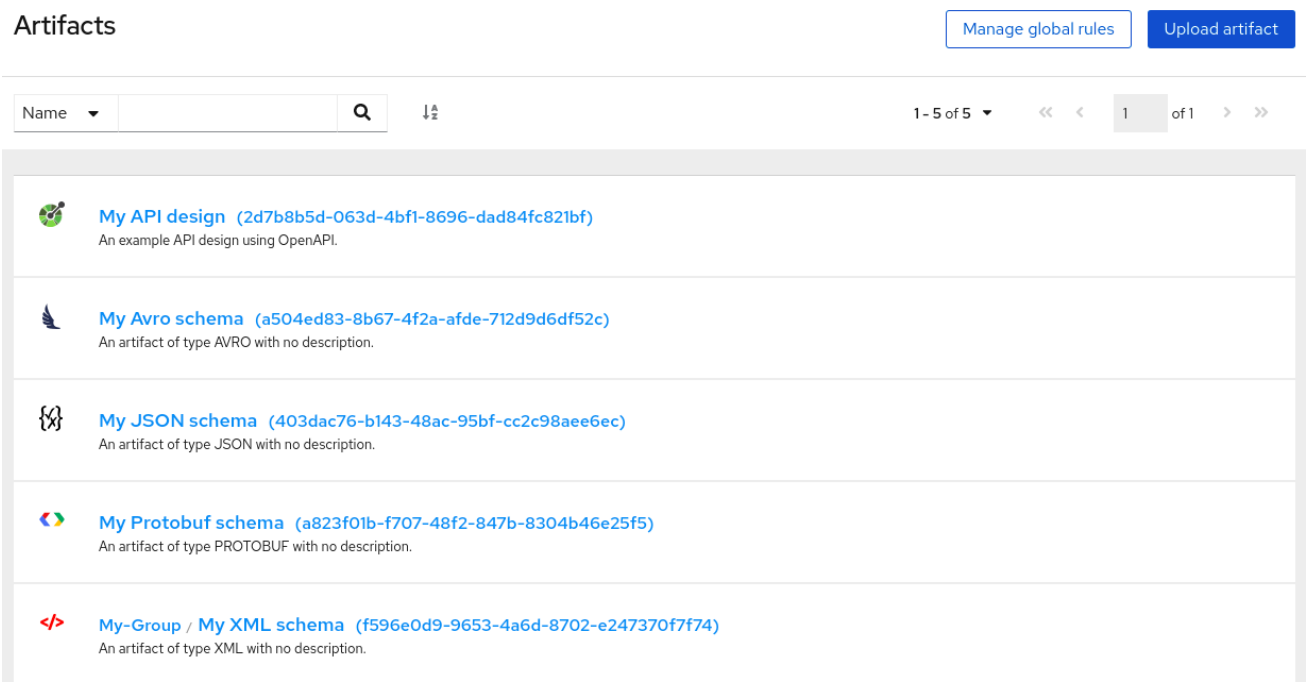
- スキーマおよびグループの詳細は、Cloud Native Computing Foundation (CNCF) Schema Registry API Version 0.1 を参照してください。

1.3. SERVICE REGISTRY WEB コンソールを使用したコンテンツの管理

Service Registry Web コンソールを使用して、レジストリーに保存されているスキーマと API アーティファクトおよびオプションのグループを閲覧および検索し、新しいスキーマと API アーティファクト、グループ、およびバージョンを追加できます。ラベル、名前、グループ、および説明でアーティファクトを検索できます。アーティファクトのコンテンツや利用可能なバージョンを表示するか、アーティファクトファイルをローカルでダウンロードできます。

Web コンソールを使用して、グローバルに、スキーマと API アーティファクトごとに、レジストリーコンテンツの任意のルールを設定することもできます。コンテンツの検証および互換性に関するこれらの任意のルールは、新しいスキーマと API アーティファクトまたはバージョンをレジストリーにアップロードする際に適用されます。詳細は *8章Service Registry アーティファクトの参照* を参照してください。

図1.1 Service Registry Web コンソール



Service Registry Web コンソールは、Service Registry デプロイメントのメインエンドポイント (たとえば、**http://MY-REGISTRY-URL/ui**) から利用できます。

関連情報

- *3章Web コンソールを使用した Service Registry コンテンツの管理*

1.4. REGISTRY CORE REST API の概要

Service Registry core REST API を使用すると、クライアントアプリケーションは Service Registry のスキーマおよび API アーティファクトを管理できます。この API では、以下を行うために作成、読み取り、更新、および削除の操作が提供されます。

アーティファクト

レジストリーに保存されたスキーマおよび API アーティファクトを管理します。アーティファクトのライフサイクル状態 (enabled、disabled、または deprecated) を管理することもできます。

アーティファクトのバージョン

スキーマまたは API アーティファクトの更新時に作成されるバージョンを管理します。アーティファクトバージョンのライフサイクル状態: enabled、disabled、または deprecated を管理することもできます。

アーティファクトのメタデータ

スキーマまたは API アーティファクトに関する詳細 (作成または変更された日時、現在の状態など) を管理します。アーティファクト名、説明、またはラベルを編集できます。アーティファクトグループと、アーティファクトが作成または変更された時期は読み取り専用です。

アーティファクトルール

特定のスキーマまたは API アーティファクトのコンテンツ展開を管理するルールを設定して、無効または互換性のないコンテンツがレジストリーに追加されないようにします。アーティファクトルールは、設定されたグローバルルールを上書きします。

グローバルルール

すべてのスキーマおよび API アーティファクトのコンテンツ展開を管理するルールを設定して、無効または互換性のないコンテンツがレジストリーに追加されないようにします。グローバルルールは、アーティファクトに独自の特定のアーティファクトルールが設定されていない場合にのみ適用されます。

検索

スキーマと API アーティファクトおよびバージョンを、名前、グループ、説明、ラベルなどで参照または検索します。

Admin

.zip ファイルでレジストリーコンテンツをエクスポートまたはインポートし、実行時にレジストリーサーバーインスタンスのログレベルを管理します。

他のスキーマレジストリー REST API との互換性

Service Registry バージョン 2 は、対応する REST API の実装を含めることで、以下のスキーマレジストリーとの API 互換性を提供します。

- Service Registry バージョン 1
- Confluent スキーマレジストリーバージョン 6
- IBM スキーマレジストリーバージョン 1
- Cloud Native Computing Foundation スキーマレジストリーバージョン 0

Confluent クライアントライブラリーを使用するアプリケーションは Service Registry をドロップイン置換として使用できます。詳細は、Replacing Confluent Schema Registry with Red Hat Integration Service Registry を参照してください。

その他のリソース

- 詳細は、Apicurio Registry REST API のドキュメント を参照してください。
- Service Registry REST API のコア API と、互換性のあるすべての API の API ドキュメントは、Service Registry のデプロイのメインエンドポイント (**http://MY-REGISTRY-URL/apis** など) から利用できます。

1.5. SERVICE REGISTRY ストレージのオプション

Service Registry は、レジストリーデータの基礎となるストレージに対して以下のオプションを提供します。

- PostgreSQL 12 database
- AMQ Streams 1.7

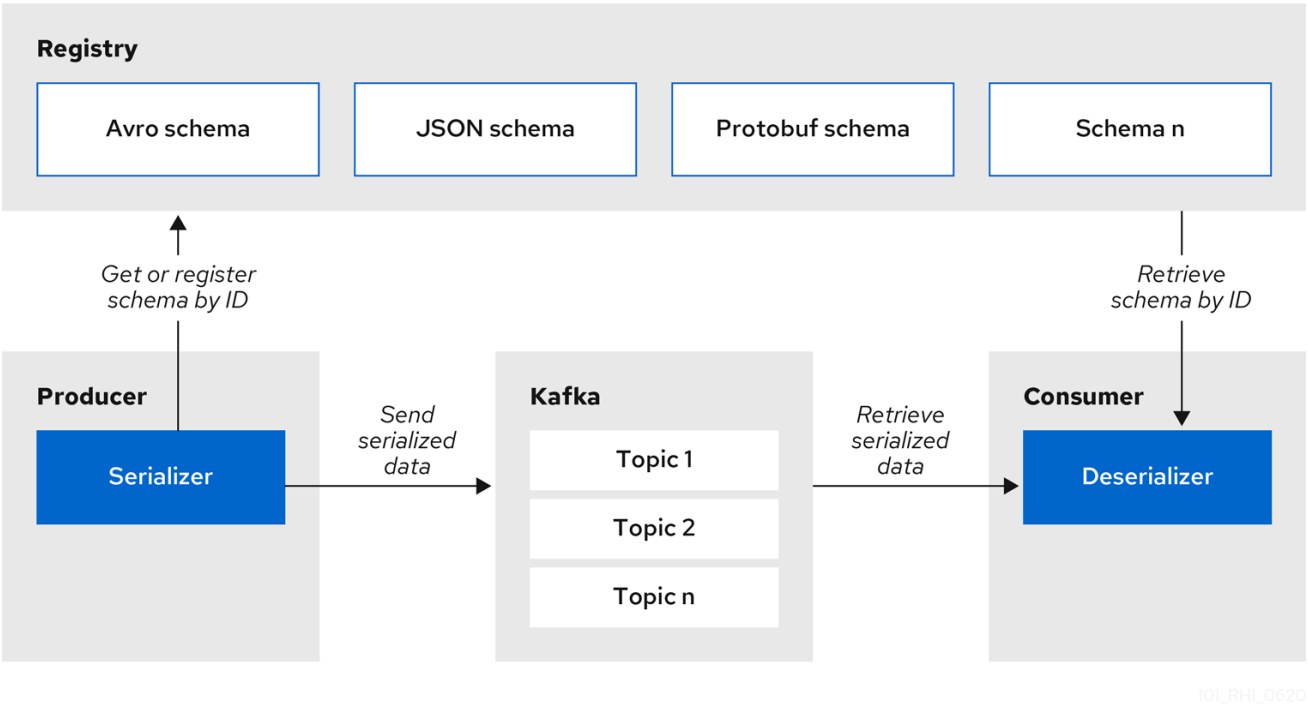
その他のリソース

- ストレージオプションの詳細は、Installing and deploying Service Registry on OpenShift を参照してください。

1.6. KAFKA クライアントシリアライザー/デシリアライザーでのスキーマの検証

Kafka プロデューサーアプリケーションは、シリアライザーを使用して、特定のイベントスキーマに準拠するメッセージをエンコードできます。Kafka コンシューマーアプリケーションはデシリアライザーを使用して、特定のスキーマ ID に基づいてメッセージが適切なスキーマを使用してシリアライズされたことを検証できます。

図1.2 Service Registry および Kafka クライアント SerDe アーキテクチャー



Service Registry は、実行時に以下のメッセージタイプを検証するために Kafka クライアントシリアライザー/デシリアライザー (SerDes) を提供します。

- Apache Avro
- Google プロトコルバッファー
- JSON スキーマ

Service Registry Maven リポジトリーおよびソースコードディストリビューションには、これらのメッセージタイプの Kafka SerDe 実装が含まれ、Kafka クライアント開発者がレジストリーと統合するために使用できます。これらの実装には、サポートされているメッセージタイプごとにカスタム Java クラスが含まれます (例: **io.apicurio.registry.serde.avro**)。これを使用して、クライアントアプリケーションは、検証のために実行時にレジストリーからスキーマをプルできます。

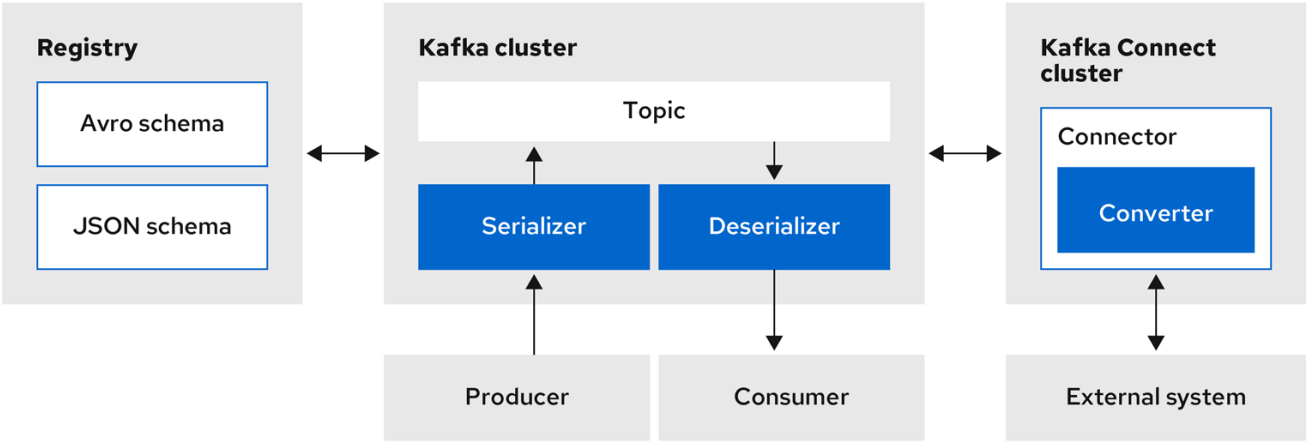
関連情報

- *7章Java で Kafka クライアントシリアライザー/デシリアライザーを使用したスキーマの検証*

1.7. KAFKA CONNECT コンバーターを使用した外部システムへのデータのストリーミング

Apache Kafka Connect と Service Registry を使用して、Kafka と外部システム間でデータをストリーミングできます。Kafka Connect を使用すると、異なるシステムのコネクターを定義して、大量のデータを Kafka ベースのシステムに出し入れできます。

図1.3 Service Registry および Kafka Connect アーキテクチャー



Service Registry は、Kafka Connect に次の機能を提供します。

- Kafka Connect スキーマのストレージ
- Apache Avro および JSON スキーマの Kafka Connect コンバーター
- スキーマを管理するレジストリー REST API

Avro および JSON スキーマコンバーターを使用して、Kafka Connect スキーマを Avro または JSON スキーマにマッピングすることができます。これらのスキーマは、メッセージのキーと値をコンパクトな Avro バイナリー形式または人間が判読できる JSON 形式にシリアライズすることができます。メッセージにはスキーマ情報が含まれず、スキーマ ID のみが含まれるため、変換された JSON も冗長性が低くなります。

Service Registry は、Kafka トピックで使用される Avro および JSON スキーマを管理および追跡できます。スキーマは Service Registry に保存され、メッセージコンテンツから切り離されるため、各メッセージには小さなスキーマ識別子だけを含める必要があります。Kafka など I/O 律速のシステムの場合、これはプロデューサーおよびコンシューマーのトータルスループットが向上することを意味します。

Service Registry が提供する Avro および JSON スキーマのシリアライザーとデシリアライザー (SerDes) は、このユース ケースの Kafka プロデューサーとコンシューマーによっても使用されます。変更イベントを使用するために作成する Kafka コンシューマーアプリケーションは、Avro または JSON Serdes を使用して変更イベントをデシリアライズすることができます。これらの Serde は Kafka ベースのシステムにインストールし、Kafka Connect や Debezium および Camel Kafka Connector などの Kafka Connect ベースのシステムと共に使用できます。

その他のリソース

- Apache Kafka Connect のドキュメント

- Avro serialization in Debezium User Guide
- Getting Started with Camel Kafka Connector
- Demonstration of using Kafka Connect with Debezium and Apicurio Registry

1.8. SERVICE REGISTRY デモ例

Service Registry は、異なるユースケースでレジストリーを使用する方法を示すオープンソースサンプルアプリケーションを提供します。たとえば、これには、Kafka シリアライザーおよびデシリアライザー (SerDe) クラスによって使用されるスキーマを保存することが含まれます。これらの Java クラスは、Kafka メッセージペイロードをシリアライズ、デシリアライズ、または検証する操作を生成または消費するときに使用するレジストリーからスキーマをフェッチします。

これらのサンプルアプリケーションには、以下が含まれます。

- Simple Avro
- Simple JSON Schema
- Confluent SerDes integration
- Avro Bean
- Custom ID strategy
- Simple Avro Maven
- REST client
- Mix Avro schemas
- Cloud Events

詳細は、 https://github.com/Apicurio/apicurio-registry-examples を参照してください

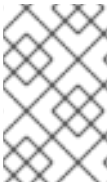
1.9. SERVICE REGISTRY で利用可能なディストリビューション

表1.1 Service Registry Operator およびイメージ

ディストリビューション	場所	リリースカテゴリー
Service Registry Operator	**Operators → OperatorHub** の OpenShift Web コンソール	テクノロジープレビュー
Service Registry Operator のコンテナーイメージ	Red Hat Ecosystem Catalog	テクノロジープレビュー
AMQ Streams での Kafka ストレージのコンテナーイメージ	Red Hat Ecosystem Catalog	テクノロジープレビュー
PostgreSQL でのデータベーストレージのコンテナーイメージ	Red Hat Ecosystem Catalog	テクノロジープレビュー

表1.2 Service Registry zip ダウンロード

ディストリビューション	場所	リリースカテゴリー
Example custom resource definitions for installation	Red Hat Integration のソフトウェアダウンロード	テクノロジープレビュー
Kafka Connect コンバーター	Red Hat Integration のソフトウェアダウンロード	テクノロジープレビュー
Maven リポジトリー	Red Hat Integration のソフトウェアダウンロード	テクノロジープレビュー
ソースコード	Red Hat Integration のソフトウェアダウンロード	テクノロジープレビュー



注記

利用可能な Service Registry ディストリビューションにアクセスするには、Red Hat Integration のサブスクリプションが必要で、Red Hat カスタマーポータルにログインする必要があります。

第2章 SERVICE REGISTRY のコンテンツルール

本章では、レジストリーコンテンツを管理するために使用されるオプションのルールを紹介し、利用可能なルール設定の詳細を説明します。

- 「ルールを使用したレジストリーコンテンツの管理」
- 「ルールの適用時」
- 「ルールの仕組み」
- 「コンテンツルールの設定」

2.1. ルールを使用したレジストリーコンテンツの管理

レジストリーコンテンツの展開を管理するために、レジストリーに追加されるアーティファクトコンテンツの任意のルールを設定できます。設定されたグローバルルールまたはアーティファクトルールはすべて、新しいアーティファクトバージョンをレジストリーにアップロードする前に渡す必要があります。設定されたアーティファクトルールは、設定されたグローバルルールを上書きします。

これらのルールの目的は、無効なコンテンツがレジストリーに追加されないようにすることです。たとえば、次の理由でコンテンツが無効になる可能性があります。

- 特定のアーティファクトタイプ (**AVRO** や **PROTOBUF など**) の構文が無効です
- 有効な構文で、セマンティクスが仕様に違反している
- 新しいコンテンツに現在のアーティファクトバージョンに関連する変更の違反が含まれる場合の非互換性

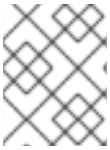
Service Registry Web コンソール、REST API コマンド、または Java クライアントアプリケーションを使用して、これらのコンテンツルールを追加できます。

2.2. ルールの適用時

ルールは、コンテンツがレジストリーに追加される場合にのみ適用されます。これには、以下の REST 操作が含まれます。

- アーティファクトの追加
- アーティファクトの更新
- アーティファクトバージョンの追加

ルールに違反した場合、Service Registry は HTTP エラーを返します。応答本文には、違反したルールと、何が問題だったのかを示すメッセージが含まれます。



注記

アーティファクトにルールが設定されていない場合、現在設定されているグローバルルールのセットがあればそれが適用されます。

2.3. ルールの仕組み

各ルールには、名前と任意の設定情報があります。レジストリーストレージは、各アーティファクトのルール一覧とグローバルルールの一覧を維持します。リスト内の各ルールは、ルール実装に固有の名前と設定プロパティーのセットで設定されます。

アーティファクトの現在のバージョン (存在する場合) および追加されるアーティファクトの新しいバージョンのコンテンツを含むルールが提供されます。ルール実装は、アーティファクトがルールを渡すかどうかに応じて true または false を返します。そうでない場合、レジストリーはその理由を HTTP エラー応答で報告します。一部のルールは、コンテンツの以前のバージョンを使用しない場合があります。たとえば、互換性ルールは以前のバージョンを使用しますが、構文またはセマンティック妥当性ルールは使用しません。

関連情報

詳細は *8章Service Registry アーティファクトの参照* を参照してください。

2.4. コンテンツルールの設定

アーティファクトごとに個別にルールを設定することも、グローバルに設定することもできます。Service Registry は、特定のアーティファクトに設定したルールを適用します。そのレベルでルールが設定されていない場合、Service Registry はグローバルに設定されたルールを適用します。グローバルルールが設定されていない場合は、ルールが適用されません。

アーティファクトルールの設定
Service Registry Web コンソールまたは REST API を使用してアーティファクトルールを設定できます。詳細は以下を参照してください。

- *3章Web コンソールを使用した Service Registry コンテンツの管理*
- Apicurio Registry REST API ドキュメント

グローバルルールの設定
グローバルルールは、複数の方法で設定できます。

- REST API で **/rules** 操作を使用します
- Service Registry Web コンソールの使用
- Service Registry アプリケーションプロパティーを使用したデフォルトのグローバルルールの設定

デフォルトのグローバルルールの設定

アプリケーションレベルで Service Registry を設定して、グローバルなルールを有効または無効にすることができます。以下のアプリケーションプロパティー形式を使用して、インストール後の設定を行わずに、インストール時にデフォルトのグローバルルールを設定できます。

```
registry.rules.global.<ruleName>
```

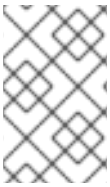
現在、以下のルール名がサポートされています。

- **compatibility**
- **validity**

application プロパティーの値は、設定されたルールに固有の有効な設定オプションである必要があります。以下の表は、各ルールの有効な値を示しています。

表2.1 Service Registry のコンテンツルール

ルール	値
Validity	**FULL**
	SYNTAX_ONLY
	NONE
互換性	**BACKWARD**
	BACKWARD_TRANSITIVE
	FORWARD
	FORWARD_TRANSITIVE
	FULL
	FULL_TRANSITIVE
	NONE



注記

これらのアプリケーションプロパティーは、Java システムプロパティーとして設定することも、Quarkus **application.properties** ファイルに含めることもできます。詳細は、Quarkus のドキュメント を参照してください。

第3章 WEB コンソールを使用した SERVICE REGISTRY コンテンツの管理

本章では、Service Registry Web コンソールを使用して、レジストリーに保存されているスキーマおよび API アーティファクトを管理する方法を説明します。これには、レジストリーコンテンツのアップロードと参照、およびオプションのルールの設定が含まれます。

- 「Service Registry Web コンソールを使用したアーティファクトの追加」
- 「Service Registry Web コンソールを使用したアーティファクトの表示」
- 「Service Registry Web コンソールを使用したコンテンツルールの設定」

3.1. SERVICE REGISTRY WEB コンソールを使用したアーティファクトの追加

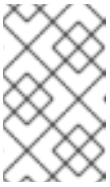
Service Registry Web コンソールを使用して、イベントスキーマと API デザインアーティファクトをレジストリーにアップロードできます。アップロード可能なアーティファクトタイプに関する詳細は、8章*Service Registry アーティファクトの参照* を参照してください。本セクションでは、Service Registry アーティファクトのアップロード、アーティファクトルールの適用、および新しいアーティファクトバージョンの追加の簡単な例を紹介します。

前提条件

- Service Registry が環境にインストールされ、実行されている。

手順

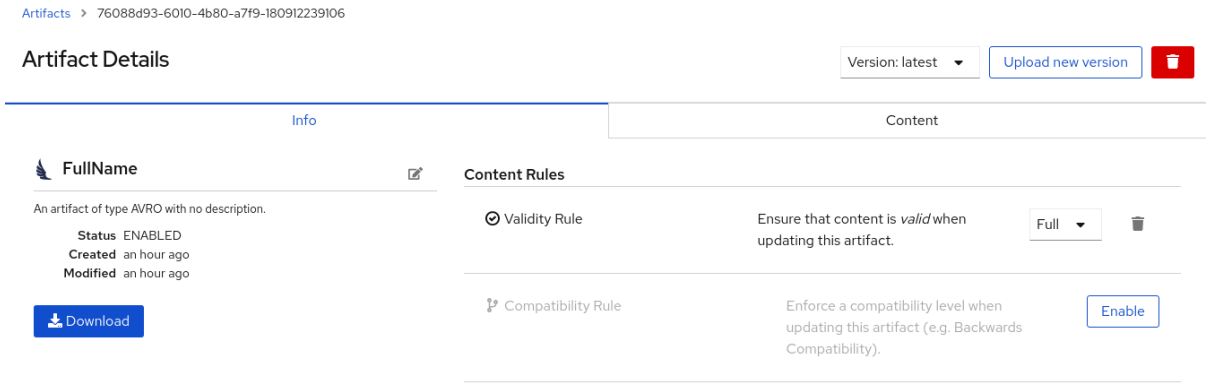
1. Service Registry Web コンソールに接続します。
   http://MY_REGISTRY_URL/ui
2. **Upload artifact** をクリックし、以下の項目を指定します。
   - **Group & ID**: デフォルトの空の設定を使用して ID および **default** グループを自動的に生成するか、またはオプションのアーティファクトグループまたは ID を入力します。
   - **Type**: デフォルトの **Auto-Detect** 設定を使用してアーティファクトタイプを自動的に検出し、ドロップダウンからアーティファクトタイプを選択します (例: **Avro Schema** または **OpenAPI**)。

     注記

     Service Registry サーバーは、**Kafka Connect スキーマ**アーティファクトタイプを自動的に検出できません。このアーティファクトタイプを手動で選択する必要があります。
   - **Artifact**: ドラッグアンドドロップまたは **Browse** をクリックして、 **my-schema.json** や **my-openapi.json** などのファイルをアップロードします。
3. **Upload** をクリックし、**Artifact Details** を表示します。

図3.1 Service Registry Web コンソールのアーティファクトの詳細



- **info**: アーティファクト名とオプションのグループ、説明、ライフサイクルのステータス、作成時、および最終更新日を表示します。**Edit Artifact Metadata** 鉛筆アイコンをクリックしてアーティファクト名と説明を編集するか、またはラベルを追加し、**Download** をクリックしてアーティファクトファイルをローカルにダウンロードします。また、有効化および設定できるアーティファクト**コンテンツルール**も表示します。
- **ドキュメント** (OpenAPI のみ): 自動生成される REST API ドキュメントを表示します。
- **Content**: 全アーティファクトコンテンツの読み取り専用ビューを表示します。

4. **Content Rules** で **Enable** をクリックして **Validity Rule** または **Compatibility Rule** を設定し、ドロップダウンから適切なルール設定を選択します。詳細は *8章Service Registry アーティファクトの参照* を参照してください。
5. **Upload new version** をクリックして新しいアーティファクトバージョンを追加し、ドラッグアンドドロップまたは **Browse** をクリックしてファイル (**my-schema.json** や **my-openapi.json** など) をアップロードします。
6. アーティファクトを削除するには、**Upload new versio** の横にあるゴミ箱アイコンをクリックします。

警告

アーティファクトを削除すると、アーティファクトとそのバージョンがすべて削除され、元に戻すことはできません。アーティファクトバージョンはイミュータブルで、個別に削除できません。

関連情報

- 「Service Registry Web コンソールを使用したアーティファクトの表示」
- 「Service Registry Web コンソールを使用したコンテンツルールの設定」

3.2. SERVICE REGISTRY WEB コンソールを使用したアーティファクトの表示

Service Registry Web コンソールを使用して、レジストリーに保存されているイベントスキーマおよび

API デザインアーティファクトを参照できます。本セクションでは、Service Registry アーティファクト、グループ、バージョン、およびアーティファクトルールを表示する簡単な例を紹介します。レジストリーに保存されているアーティファクトタイプの詳細は、*8章Service Registry アーティファクトの参照* を参照してください。

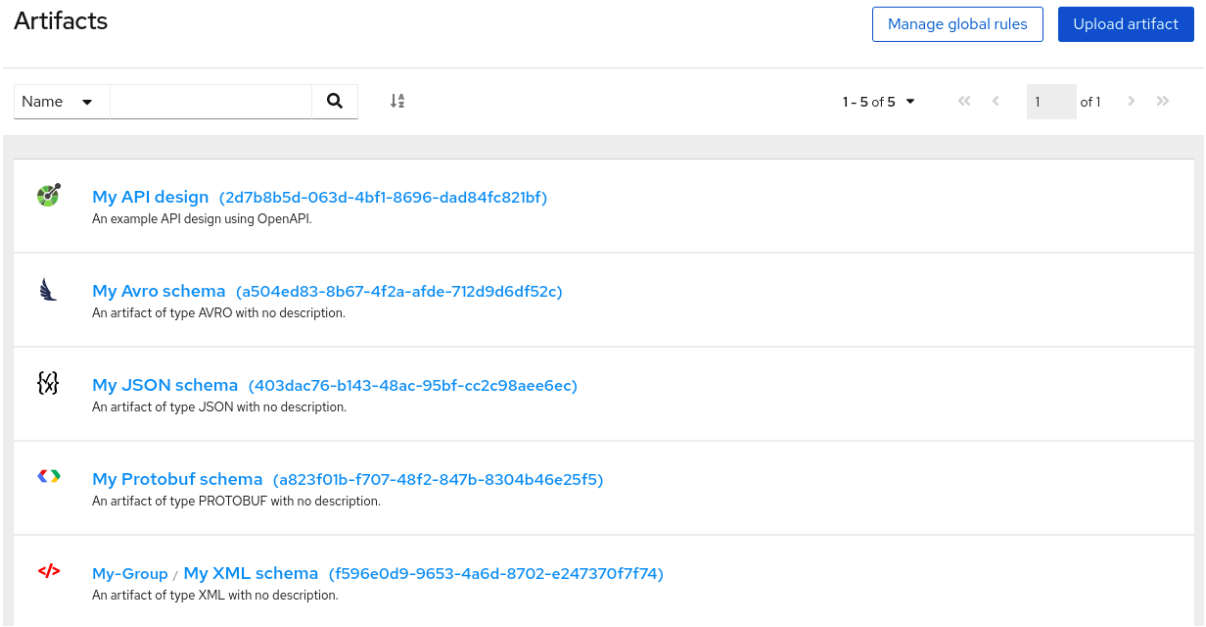
前提条件

- Service Registry が環境にインストールされ、実行されている。
- Service Registry Web コンソール、REST API コマンド、Maven プラグイン、または Java クライアントアプリケーションを使用して、アーティファクトがレジストリーに追加されている必要があります。

手順

1. Service Registry Web コンソールに接続します。
   http://MY_REGISTRY_URL/ui
2. レジストリーに保存されているアーティファクト一覧を参照するか、検索文字列を入力してアーティファクトを検索します。特定の **Name**、**Group**、**Description**、または **Labels** で検索できます。

   図3.2 Service Registry Web コンソールでのアーティファクトの閲覧



3. **View artifact** をクリックして **Artifact Details** を表示します。
   - **info**: アーティファクト名とオプションのグループ、説明、ライフサイクルのステータス、作成時、および最終更新日を表示します。**Edit Artifact Metadata** 鉛筆アイコンをクリックしてアーティファクト名と説明を編集するか、またはラベルを追加し、**Download** をクリックしてアーティファクトファイルをローカルにダウンロードします。また、有効化および設定できるアーティファクト**コンテンツルール**も表示します。
   - **ドキュメント** (OpenAPI のみ): 自動生成される REST API ドキュメントを表示します。
   - **Content**: 全アーティファクトコンテンツの読み取り専用ビューを表示します。
4. 追加バージョンが追加されている場合は、ドロップダウンから異なるアーティファクト**Version** を表示します。

その他のリソース

- 「Service Registry Web コンソールを使用したアーティファクトの追加」
- 「Service Registry Web コンソールを使用したコンテンツルールの設定」

3.3. SERVICE REGISTRY WEB コンソールを使用したコンテンツルールの設定

Service Registry Web コンソールを使用して、無効なコンテンツがレジストリーに追加されないようにオプションのルールを設定できます。設定されたアーティファクトルールまたはグローバルルールはすべて、新しいアーティファクトバージョンをレジストリーにアップロードする前に渡す必要があります。設定されたアーティファクトルールは、設定されたグローバルルールを上書きします。詳細は 2章*Service Registry のコンテンツルール* を参照してください。

本セクションでは、グローバルルールとアーティファクトルールを設定する簡単な例を紹介します。選択可能なさまざまなルールタイプおよび関連する設定の詳細は、8章*Service Registry アーティファクトの参照* を参照してください。

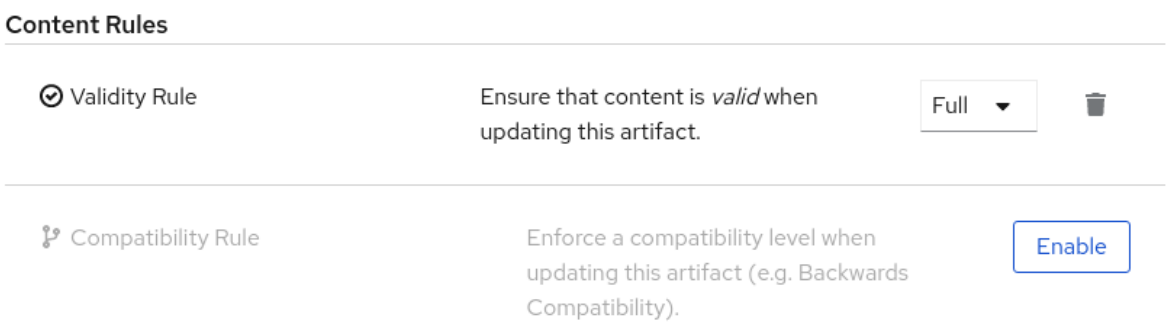
前提条件

- Service Registry が環境にインストールされ、実行されている。
- アーティファクトルールの場合、Service Registry Web コンソール、REST API コマンド、Maven プラグイン、または Java クライアントアプリケーションを使用して、アーティファクトがレジストリーに追加されている必要があります。

手順

1. Service Registry Web コンソールに接続します。
http://MY_REGISTRY_URL/ui
2. アーティファクトルールの場合、レジストリーに保存されているアーティファクト一覧を参照するか、検索文字列を入力してアーティファクトを検索します。特定のアーティファクト **Name**、**Group**、**Description**、または **Labels** で検索できます。
3. **View artifact** をクリックして **Artifact Details** を表示します。
4. **Content Rules** で **Enable** をクリックしてアーティファクトの **有効性ルール**または **互換性ルール**を設定し、ドロップダウンから適切なルール設定を選択します。詳細は 8章*Service Registry アーティファクトの参照* を参照してください。

図3.3 Service Registry Web コンソールでのコンテンツルールの設定



5. グローバルルールの場合は、ツールバーの右上の **Manage global rules** をクリックし、**Enable** をクリックして **Validity Rule** または **Compatibility Rule** を設定し、ドロップダウンから適切なルール設定を選択します。詳細は 8章*Service Registry アーティファクトの参照* を参照してください。

6. アーティファクトルールまたはグローバルルールを無効にするには、ルールの横にあるゴミ箱アイコンをクリックします。

関連情報

- 「Service Registry Web コンソールを使用したアーティファクトの追加」

第4章 REST API を使用した SERVICE REGISTRY コンテンツの管理

クライアントアプリケーションは、Registry REST API 操作を使用して、Service Registry のスキーマおよび API アーティファクトを管理できます (例: 実稼働環境用にデプロイされる CI/CD パイプラインなど)。Registry REST API は、レジストリーに保存されるアーティファクト、バージョン、メタデータ、およびルールの作成、読み取り、更新、および削除操作を提供します。詳細は、Apicurio Registry REST API のドキュメント を参照してください。

本章では、Service Registry core REST API について説明し、これを使用してレジストリーに保存されているスキーマおよび API アーティファクトを管理する方法を説明します。

- 「Registry REST API コマンドを使用したスキーマおよび API アーティファクトの管理」
- 「Registry REST API コマンドを使用したスキーマおよび API アーティファクトのバージョンの管理」
- 「Registry REST API コマンドを使用したレジストリーコンテンツのエクスポートおよびインポート」

前提条件

- 「Registry core REST API の概要」

関連情報

- Apicurio Registry REST API ドキュメント

4.1. REGISTRY REST API コマンドを使用したスキーマおよび API アーティファクトの管理

本セクションでは、レジストリー v2 コア REST API を使用して、レジストリーに Apache Avro スキーマアーティファクトを追加および取得するための単純な curl ベースの例を紹介します。

前提条件

- Service Registry が環境にインストールされ、実行されている。

手順

1. **/groups/{group}/artifacts** 操作を使用してアーティファクトをレジストリーに追加します。以下の **curl** コマンドの例は、株価アプリケーションの単純なアーティファクトを追加します。

```
$ curl -X POST -H "Content-type: application/json; artifactType=AVRO" \
  -H "X-Registry-ArtifactId: share-price" \ 1
  --data '{"type":"record","name":"price","namespace":"com.example", \
   "fields":[{"name":"symbol","type":"string"},{"name":"price","type":"string"}]}' \ 2
  http://MY-REGISTRY-HOST/apis/registry/v2/groups/my-group/artifacts 3
```

1 この例では、 **share-price** のアーティファクト ID を持つ Avro スキーマアーティファクトを追加します。一意のアーティファクト ID を指定しない場合、Service Registry は UUID として自動的に生成します。

2 **MY-REGISTRY-HOST** は、Service Registry がデプロイされているホスト名です。例: **my-cluster-service-registry-myproject.example.com**。

3 この例では、API パスで **my-group** のグループ ID を指定します。一意のグループ ID を指定しない場合は、API パスで **../groups/default** を指定する必要があります。

2. 応答に、アーティファクトが追加されたことを確認するために、想定される JSON ボディーが含まれていることを確認します。以下に例を示します。

```
{"createdBy":"","createdOn":"2021-04-16T09:07:51+0000","modifiedBy":"",
"modifiedOn":"2021-04-16T09:07:51+0000","id":"share-price","version":"1", 1
"type":"AVRO","globalId":2,"state":"ENABLED","groupId":"my-group","contentId":2} 2
```

1 アーティファクトの追加時にバージョンが指定されなかったため、デフォルトのバージョン **1** が自動的に作成されます。

2 これはレジストリーに追加された 2 つ目のアーティファクトであるため、グローバル ID とコンテンツ ID の値は **2** になります。

3. API パスでアーティファクト ID を使用して、レジストリーからアーティファクトコンテンツを取得します。この例では、指定された ID は **share-price** です。

```
$ curl http://MY-REGISTRY-URL/apis/registry/v2/groups/my-group/artifacts/share-price \
{"type":"record","name":"price","namespace":"com.example", \
  "fields":[{"name":"symbol","type":"string"},{"name":"price","type":"string"}]}
```

関連情報

- REST API のサンプル要求の詳細は、Apicurio Registry REST API のドキュメント を参照してください。

4.2. REGISTRY REST API コマンドを使用したスキーマおよび API アーティファクトのバージョンの管理

v2 REST API を使用してスキーマおよび API アーティファクトを Service Registry に追加する際にアーティファクトバージョンを指定しない場合、Service Registry は自動的にアーティファクトバージョンを生成します。新規アーティファクト作成時のデフォルトのバージョンは **1** です。

Service Registry は、**X-Registry-Version** HTTP リクエストヘッダーを文字列として使用してバージョンを指定できるカスタムバージョン管理もサポートしています。カスタムバージョン値を指定すると、アーティファクトの作成または更新時に通常割り当てられるデフォルトのバージョンが上書きされます。バージョンを必要とする REST API 操作を実行する場合は、このバージョン値を使用できます。

本セクションでは、レジストリー v2 コア REST API を使用して、レジストリーに カスタム Apache Avro スキーマアバージョンを追加および取得するための単純な curl ベースの例を紹介します。REST API を使用してアーティファクトを追加または更新したり、アーティファクトバージョンを追加したりするときにカスタムバージョンを指定できます。

前提条件

- Service Registry が環境にインストールされ、実行されている。

手順

手順

1. **/groups/{group}/artifacts** 操作を使用して、レジストリーにアーティファクトバージョンを追加します。以下の **curl** コマンドの例は、株価アプリケーションの単純なアーティファクトを追加します。

```
$ curl -X POST -H "Content-type: application/json; artifactType=AVRO" \
  -H "X-Registry-ArtifactId: my-share-price" -H "X-Registry-Version: 1.1.1" \ 1
  --data '{"type":"record","name":" p","namespace":"com.example", \
  "fields":[{"name":"symbol","type":"string"},{"name":"price","type":"string"}]}' \ 2
  http://MY-REGISTRY-HOST/apis/registry/v2/groups/my-group/artifacts 3
```

1. この例では、アーティファクト ID が **my-share-price** でバージョンが **1.1.1** の Avro スキーマアーティファクトを追加します。バージョンを指定しないと、Service Registry によってデフォルトのバージョン **1** が自動的に生成されます。

2. **MY-REGISTRY-HOST** は、Service Registry がデプロイされているホスト名です。例: **my-cluster-service-registry-myproject.example.com**。

3. この例では、API パスで **my-group** のグループ ID を指定します。一意のグループ ID を指定しない場合は、API パスで **../groups/default** を指定する必要があります。

2. 応答に、カスタムアーティファクトバージョンが追加されたことを確認するために、想定される JSON ボディーが含まれていることを確認します。以下に例を示します。

```
{"createdBy":"","createdOn":"2021-04-16T10:51:43+0000","modifiedBy":"",
"modifiedOn":"2021-04-16T10:51:43+0000","id":"my-share-price","version":"1.1.1", 1
"type":"AVRO","globalId":3,"state":"ENABLED","groupId":"my-group","contentId":3} 2
```

1. アーティファクトの追加時に、**1.1.1** のカスタムバージョンが指定されました。

2. これはレジストリーに追加された 3 つ目のアーティファクトであるため、グローバル ID とコンテンツ ID の値は **3** になります。

3. API パスでアーティファクト ID とバージョンを使用して、レジストリーからアーティファクトコンテンツを取得します。この例では、指定された ID は **my-share-price** であり、バージョンは **1.1.1** です。

```
$ curl http://MY-REGISTRY-URL/apis/registry/v2/groups/my-group/artifacts/my-share-price/versions/1.1.1 \
{"type":"record","name":"price","namespace":"com.example", \
  "fields":[{"name":"symbol","type":"string"},{"name":"price","type":"string"}]}
```

関連情報

- REST API のサンプル要求の詳細は、Apicurio Registry REST API のドキュメント を参照してください。

4.3. REGISTRY REST API コマンドを使用したレジストリーコンテンツのエクスポートおよびインポート

本セクションでは、レジストリー v2 コア REST API を使用して、既存のレジストリーデータをある

Service Registry インスタンスから別の Service Registry インスタンスに **.zip** 形式でエクスポートおよびインポートするシンプルな curl ベースの例を示します。たとえば、これは Service Registry v2.x インスタンスから別のインスタンスに移行またはアップグレードする場合に役立ちます。

前提条件

- Service Registry が環境にインストールされ、実行されている。

手順

1. 既存のソースの Service Registry インスタンスからレジストリーデータをエクスポートします。

```
$ curl http://MY-REGISTRY-HOST/apis/registry/v2/admin/export \
  --output my-registry-data.zip
```

MY-REGISTRY-HOST は、ソース ServiceRegistry がデプロイされているホスト名です。例: **my-cluster-source-registry-myproject.example.com**。

2. レジストリーデータをターゲット Service Registry インスタンスにインポートします。

```
$ curl -X POST "http://MY-REGISTRY-HOST/apis/registry/v2/admin/import" \
  -H "Content-Type: application/zip" --data-binary @my-registry-data.zip
```

MY-REGISTRY-HOST は、ターゲットの Service Registry がデプロイされているホスト名です。例: **my-cluster-target-registry-myproject.example.com**。

関連情報

- 詳細は、Apicurio Registry REST API documentation の **admin** エンドポイントを参照してください。
- Service Registry バージョン 1.x から 2.x に移行するためのエクスポートツールの詳細については、Apicurio Registry export utility for 1.x versions を参照してください。

第5章 MAVEN プラグインを使用した SERVICE REGISTRY コンテンツの管理

本章では、Service Registry Maven プラグインを使用して、レジストリーに保存されているスキーマおよび API アーティファクトを管理する方法を説明します。

- 「Maven プラグインを使用したスキーマおよび API アーティファクトの追加」
- 「Maven プラグインを使用したスキーマおよび API アーティファクトのダウンロード」
- 「Maven プラグインを使用したスキーマおよび API アーティファクトのテスト」

前提条件

- 1章*Service Registry の概要* を参照してください。
- Service Registry が環境にインストールされ、実行されている。
- Maven が使用している環境にインストールおよび設定されている。

5.1. MAVEN プラグインを使用したスキーマおよび API アーティファクトの追加

Maven プラグインの最も一般的なユースケースは、ビルド中にアーティファクトを追加することです。これは、**register** 実行目標を使用して実現できます。

手順

- Maven **pom.xml** ファイルを更新して、 **apicurio-registry-maven-plugin** を使用してアーティファクトを登録します。以下の例は、Apache Avro および GraphQL スキーマの登録を示しています。

```
<plugin>
  <groupId>io.apicurio</groupId>
  <artifactId>apicurio-registry-maven-plugin</artifactId>
  <version>${apicurio.version}</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>register</goal> (1)
      </goals>
      <configuration>
        <registryUrl>http://REGISTRY-URL/apis/registry/v2</registryUrl> (2)
        <artifacts>
          <artifact>
            <groupId>TestGroup</groupId> (3)
            <artifactId>FullNameRecord</artifactId>
            <file>${project.basedir}/src/main/resources/schemas/record.avsc</file>
            <ifExists>FAIL</ifExists>
          </artifact>
          <artifact>
            <groupId>TestGroup</groupId>
            <artifactId>ExampleAPI</artifactId> (4)
```

```
                <type>GRAPHQL</type>
                <file>${project.basedir}/src/main/resources/apis/example.graphql</file>
                <ifExists>RETURN_OR_UPDATE</ifExists>
                <canonicalize>true</canonicalize>
              </artifact>
            </artifacts>
          </configuration>
        </execution>
      </executions>
    </plugin>
```

1. スキーマアーティファクトをレジストリーにアップロードするための実行目標として **register** を指定します。
2. **../apis/registry/v2** エンドポイントでService Registry URL を指定します。
3. Service Registry アーティファクトグループ ID を指定します。一意のグループを使用しない場合は、**default** のグループを指定できます。
4. 指定したグループ ID、アーティファクト ID、および場所を使用して複数のアーティファクトをアップロードできます。

その他のリソース

- Service Registry Maven プラグインの詳細は、Registry demonstration example を参照してください。

5.2. MAVEN プラグインを使用したスキーマおよび API アーティファクトのダウンロード

Maven プラグインを使用して Service Registry からアーティファクトをダウンロードできます。これは、たとえば、登録されたスキーマからコードを生成する場合などに便利です。

手順

- Maven **pom.xml** ファイルを更新して、**apicurio-registry-maven-plugin** を使用してアーティファクトをダウンロードします。以下の例は、Apache Avro および GraphQL スキーマのダウンロードを示しています。

```
<plugin>
  <groupId>io.apicurio</groupId>
  <artifactId>apicurio-registry-maven-plugin</artifactId>
  <version>${apicurio.version}</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>download</goal> (1)
      </goals>
      <configuration>
        <registryUrl>http://REGISTRY-URL/apis/registry/v2</registryUrl> (2)
        <artifacts>
          <artifact>
```

```
            <groupId>TestGroup</groupId> 3
            <artifactId>FullNameRecord</artifactId> 4
            <file>${project.build.directory}/classes/record.avsc</file>
            <overwrite>true</overwrite>
          </artifact>
          <artifact>
            <groupId>TestGroup</groupId>
            <artifactId>ExampleAPI</artifactId>
            <version>1</version>
            <file>${project.build.directory}/classes/example.graphql</file>
            <overwrite>true</overwrite>
          </artifact>
        </artifacts>
      </configuration>
    </execution>
  </executions>
</plugin>
```

1. 実行目標として **download** を指定します。
2. **../apis/registry/v2** エンドポイントでサービスレジストリー URL を指定します。
3. Service Registry アーティファクトグループ ID を指定します。一意のグループを使用しない場合は、**default** のグループを指定できます。
4. アーティファクト ID を使用すると、複数のアーティファクトを指定したディレクトリーにダウンロードできます。

その他のリソース

- Service Registry Maven プラグインの詳細は、Registry demonstration example を参照してください。

5.3. MAVEN プラグインを使用したスキーマおよび API アーティファクトのテスト

実際にアーティファクトを変更せずに、アーティファクトをが登録できることを確認する場合があります。これは、ルールが Service Registry に設定されている場合に役に立ちます。アーティファクトのコンテンツが設定済みのルールのいずれかに違反する場合、アーティファクトのテストに失敗します。

注記

アーティファクトがテストに合格した場合でも、Service Registry にはコンテンツが追加されません。

手順

- Maven **pom.xml** ファイルを更新して、 **apicurio-registry-maven-plugin** を使用してアーティファクトをテストします。Apache Avro スキーマのテストの例を以下に示します。

```
<plugin>
  <groupId>io.apicurio</groupId>
  <artifactId>apicurio-registry-maven-plugin</artifactId>
```

```
<version>${apicurio.version}</version>
<executions>
  <execution>
    <phase>generate-sources</phase>
    <goals>
      <goal>test-update</goal> (1)
    </goals>
    <configuration>
      <registryUrl>http://REGISTRY-URL/apis/registry/v2</registryUrl> (2)
      <artifacts>
        <artifact>
          <groupId>TestGroup</groupId> (3)
          <artifactId>FullNameRecord</artifactId>
          <file>${project.basedir}/src/main/resources/schemas/record.avsc</file> (4)
        </artifact>
        <artifact>
          <groupId>TestGroup</groupId>
          <artifactId>ExampleAPI</artifactId>
          <type>GRAPHQL</type>
          <file>${project.basedir}/src/main/resources/apis/example.graphql</file>
        </artifact>
      </artifacts>
    </configuration>
  </execution>
</executions>
</plugin>
```

1. スキーマアーティファクトをテストするための実行目標として **test-update** を指定します。
2. **../apis/registry/v2** エンドポイントでサービスレジストリー URL を指定します。
3. Service Registry アーティファクトグループ ID を指定します。一意のグループを使用しない場合は、**default** のグループを指定できます。
4. アーティファクト ID を使用すると、指定のディレクトリーから複数のアーティファクトをテストできます。

その他のリソース

- Service Registry Maven プラグインの詳細は、Registry demonstration example を参照してください。

第6章 JAVA クライアントを使用した SERVICE REGISTRY コンテンツの管理

本章では、Service Registry Java クライアントを使用する方法を説明します。

- 「Service Registry Java クライアント」
- 「Service Registry クライアントアプリケーションの作成」
- 「Service Registry Java クライアント設定」

6.1. SERVICE REGISTRY JAVA クライアント

Java クライアントアプリケーションを使用して、Service Registry に保存されているアーティファクトを管理できます。Service Registry Java クライアントクラスを使用して、レジストリーに保存されているアーティファクトを作成、読み取り、更新、または削除できます。グローバルなルールの管理やレジストリーデータのインポートおよびエクスポートなど、クライアントを使用して管理機能を実行することもできます。

正しい依存関係をプロジェクトに追加すると、Service Registry Java クライアントにアクセスできます。詳細は 「Service Registry クライアントアプリケーションの作成」 を参照してください。

Service Registry クライアントは、JDK によって提供される HTTP クライアントを使用して実装されます。これにより、カスタムヘッダーの追加や TLS (Transport Layer Security) 認証のオプションの有効化など、使用をカスタマイズすることができます。詳細は 「Service Registry Java クライアント設定」 を参照してください。

6.2. SERVICE REGISTRY クライアントアプリケーションの作成

本セクションでは、Java クライアントアプリケーションを使用して Service Registry に保存されているアーティファクトを管理する方法を説明します。

前提条件

- 1章*Service Registry の概要* を参照してください。
- Service Registry が環境にインストールされ、実行されている。

手順

1. 以下の依存関係を Maven プロジェクトに追加します。

```
<dependency>
    <groupId>io.apicurio</groupId>
    <artifactId>apicurio-registry-client</artifactId>
    <version>${apicurio-registry.version}</version>
</dependency>
```

2. 以下のようにレジストリークライアントを作成します。

```
public class ClientExample {

    private static final RegistryRestClient client;
```

```
    public static void main(String[] args) throws Exception {
        // Create a registry client
        String registryUrl = "https://my-registry.my-domain.com/apis/registry/v2"; (1)
        RegistryClient client = RegistryClientFactory.create(registryUrl); (2)
    }
}
```

1. **https://my-registry.my-domain.com** のサンプルレジストリー URL を指定すると、クライアントは自動的に **/apis/registerry/v2** を追加します。
2. Service Registry クライアント作成時の他のオプションは、次のセクションの Java クライアント設定を参照してください。

3. クライアントが作成されると、クライアントを介して Service Registry REST API からのすべての操作を使用することができます。詳細は、Apicurio Registry REST API のドキュメント を参照してください。

その他のリソース

- Service Registry クライアントを使用およびカスタマイズする方法のオープンソースの例については、Registry REST クライアントのデモの例 を参照してください。
- 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で Service Registry Kafka クライアントシリアライザー/デシリアライザー (SerDes) を使用する方法は、*7章Java で Kafka クライアントシリアライザー/デシリアライザーを使用したスキーマの検証* を参照してください。

6.3. SERVICE REGISTRY JAVA クライアント設定

Service Registry Java クライアントには、クライアントファクトリーを基にした以下の設定オプションが含まれます。

表6.1 Service Registry Java クライアント設定オプション

オプション	説明	引数
プレーンクライアント	実行中のレジストリーと対話するために使用される基本的な REST クライアント。	**baseUrl**
カスタム設定のあるクライアント	ユーザーによって提供される設定を使用するレジストリークライアント。	**baseUrl、Map<String Object> configs**
カスタム設定および認証のあるクライアント	カスタム設定を含むマップを受け入れるレジストリークライアント。これは、呼び出しにカスタムヘッダーを追加する場合に便利です。また、リクエストの認証に使用する **auth** インスタンスも必要です。	**baseUrl, Map<String Object> configs, Auth auth**

カスタムヘッダー設定
カスタムヘッダーを設定するには、**configs** マップキーに **apicurio.registry.request.headers** 接頭辞を追加する必要があります。たとえば、値が **Basic:xxxxx** の **apicurio.registry.request.headers.Authorization** のキーは、値が **Basic: xxxxx** の **Authorization** の

ヘッダーになります。

TLS 設定オプション
以下のプロパティーを使用して、Service Registry Java クライアントの Transport Layer Security (TLS) 認証を設定できます。

- **apicurio.registry.request.ssl.truststore.location**
- **apicurio.registry.request.ssl.truststore.password**
- **apicurio.registry.request.ssl.truststore.type**
- **apicurio.registry.request.ssl.keystore.location**
- **apicurio.registry.request.ssl.keystore.password**
- **apicurio.registry.request.ssl.keystore.type**
- **apicurio.registry.request.ssl.key.password**

第7章 JAVA で KAFKA クライアントシリアライザー/デシリアライザーを使用したスキーマの検証

Service Registry によって、Java で書かれた 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のクライアントシリアライザー/デシリアライザー (SerDes) が提供されます。Kafka プロデューサーアプリケーションは、シリアライザーを使用して、特定のイベントスキーマに準拠するメッセージをエンコードします。Kafka コンシューマーアプリケーションはデシリアライザーを使用して、特定のスキーマ ID に基づいてメッセージが適切なスキーマを使用してシリアライズされたことを検証します。これにより、スキーマが一貫して使用されるようにし、実行時にデータエラーが発生しないようにします。

本章では、プロデューサーおよびコンシューマークライアントアプリケーションで Kafka クライアント SerDe を使用する方法について説明します。

- 「Kafka クライアントアプリケーションおよび Service Registry」
- 「Service Registry でスキーマを検索するストラテジー」
- 「Service Registry シリアライザー/デシリアライザーの設定」
- 「異なるクライアントのシリアライザー/デシリアライザータイプの使用」
- 「Service Registry を使用した Avro SerDe の設定」
- 「Service Registry を使用した JSON スキーマ SerDe の設定」
- 「Service Registry を使用した Protobuf SerDe の設定」
- 「スキーマの Service Registry への登録」
- 「Kafka コンシューマークライアントからのスキーマの使用」
- 「Kafka プロデューサークライアントからのスキーマの使用」
- 「Kafka Streams アプリケーションからのスキーマの使用」

前提条件

- 1章*Service Registry の概要* を読む。
- Service Registry がインストールされている。
- Kafka プロデューサーおよびコンシューマークライアントアプリケーションが作成済みである。
  Kafka クライアントアプリケーションの詳細は、Using AMQ Streams on Openshift を参照してください。

7.1. KAFKA クライアントアプリケーションおよび SERVICE REGISTRY

Service Registry は、クライアントアプリケーション設定からスキーマ管理を分離します。クライアントコードに URL を指定すると、Java クライアントアプリケーションが Service Registry からスキーマを使用できます。

Service Registry にスキーマを保存して、メッセージをシリアライズおよびデシリアライズできます。クライアントアプリケーションからスキーマが参照され、送受信されるメッセージとこれらのスキーマ

の互換性を維持するようにします。Kafka クライアントアプリケーションは実行時にスキーマを Service Registry からプッシュまたはプルできます。

スキーマは進化するので、Service Registry でルールを定義できます。たとえば、スキーマの変更が有効で、アプリケーションによって使用される以前のバージョンとの互換性を維持するようにします。Service Registry は、変更済みのスキーマと以前のスキーマバージョンを比較することで、互換性をチェックします。

Service Registry スキーマテクノロジー

Service Registry は、以下のようなスキーマテクノロジーのスキーマレジストリーサポートを提供します。

- Avro
- Protobuf
- JSON スキーマ

これらのスキーマテクノロジーは、Service Registry によって提供される Kafka クライアントのシリアライザー/デシリアライザー (SerDe) サービスを介してクライアントアプリケーションで使用できます。Service Registry によって提供される SerDe クラスの成熟度および使用法は異なる場合があります。以降のセクションでは、各スキーマタイプの詳細を提供します。

プロデューサースキーマの設定

プロデューサークライアントアプリケーションは、シリアライザーを使用して、特定のブローカートピックに送信するメッセージを正しいデータ形式にします。

プロデューサーが Service Registry を使用してシリアライズできるようにするには、以下を行います。

- スキーマを Service Registry に定義、登録します (存在しない場合)。
- 以下を使用して プロデューサークライアントコードの設定を行います。
  - Service Registry の URL
  - メッセージで使用する Service Registry シリアライザー
  - Kafka メッセージを Service Registry のスキーマアーティファクトにマップするストラテジー
  - Service Registry でのシリアライズに使用するスキーマを検索または登録するストラテジー

スキーマを登録したら、Kafka および Service Registry を開始するときに、スキーマにアクセスして、プロデューサーにより Kafka ブローカートピックに送信されるメッセージをフォーマットできます。または、設定に応じて、プロデューサーは最初の使用時にスキーマを自動的に登録できます。

スキーマがすでに存在する場合、Service Registry に定義される互換性ルールに基づいて、レジストリー REST API を使用して新バージョンのスキーマを作成できます。バージョンは、スキーマの進化にともなう互換性チェックに使用します。グループ ID、アーティファクト ID、およびバージョンは、スキーマを識別する一意のタプルを表します。

コンシューマースキーマの設定

コンシューマークライアントアプリケーションは、デシリアライザーを使用することで、そのアプリケーションが消費するメッセージを特定のブローカートピックから正しいデータ形式にします。

コンシューマーがデシリアライズに Service Registry を使用できるようにするには、以下を実行します。

- スキーマを Service Registry に定義、登録します (存在しない場合)。
- 以下を使用して コンシューマークライアントコードを設定します。
  - Service Registry の URL
  - メッセージで使用する Service Registry デシリアライザー
  - デシリアライズの入力データストリーム

グローバル ID を使用したスキーマの取得

デフォルトでは、スキーマは、消費されるメッセージに指定されるグローバル ID を使用してデシリアライザーによって Service Registry から取得されます。スキーマグローバル ID は、プロデューサーアプリケーションの設定に応じて、メッセージヘッダーまたはメッセージペイロードに配置できます。

メッセージペイロードでグローバル ID を見つけるとき、データの形式は、コンシューマーへの信号として使用されるマジックバイトで始まり、通常通りグローバル ID とメッセージデータが続きます。以下に例を示します。

```
# ...
[MAGIC_BYTE]
[GLOBAL_ID]
[MESSAGE DATA]
```

これで、Kafka および Service Registry を開始するとき、スキーマにアクセスして、Kafka ブローカートピックから受信するメッセージをフォーマットできます。

コンテンツ ID を使用したスキーマの取得

または、アーティファクトコンテンツの一意の ID であるコンテンツ ID に基づいて、Service Registry からスキーマを取得するように設定できます。グローバル ID はアーティファクトバージョンの一意の ID です。

コンテンツ ID はバージョンを一意に識別しませんが、バージョンコンテンツのみを一意に識別します。複数のバージョンがまったく同じコンテンツを共有している場合、グローバル ID は異なりますが、コンテンツ ID は同じです。Confluent スキーマレジストリーは、デフォルトでコンテンツ ID を使用します。

7.2. SERVICE REGISTRY でスキーマを検索するストラテジー

Kafka クライアントのシリアライザーは **検索ストラテジー** を使用して、メッセージスキーマが Service Registry に登録されるアーティファクト ID およびグローバル ID を決定します。特定のトピックとメッセージについて、 **ArtifactResolverStrategy** Java インターフェイスのさまざまな実装を使用して、レジストリー内のアーティファクトへの参照を返すことができます。

各ストラテジーのクラスは、**io.apicurio.registry.serde.strategy** パッケージに含まれています。Avro SerDe の特定のストラテジークラスは、**io.apicurio.registry.serde.avro.strategy package** に含まれています。デフォルトのストラテジー、**TopicIdStrategy** は、メッセージを受信する Kafka トピックと同じ名前の Service Registry アーティファクトを検索します。

例

```
public ArtifactReference artifactReference(String topic, boolean isKey, T schema) {
    return ArtifactReference.builder()
        .groupId(null)
```

```
            .artifactId(String.format("%s-%s", topic, isKey ? "key" : "value"))
            .build();
```

- **topic** パラメーターは、メッセージを受信する Kafka トピックの名前です。
- **isKey** パラメーターは、メッセージキーがシリアライズされている場合は **true** であり、メッセージ値がシリアライズされている場合は **false** です。
- **schema** パラメーターは、シリアライズ/デシリアライズされたメッセージのスキーマです。
- 返される **ArtifactReference** には、スキーマが登録されているアーティファクト ID が含まれています。

使用する検索アップストラテジーは、スキーマを保存する方法と場所によって異なります。たとえば、同じ Avro メッセージタイプを持つ Kafka トピックが複数ある場合、**レコード ID** を使用するストラテジーを使用することがあります。

KeystoneOpenIdcaceResolverStrategy インターフェイス

アーティファクトリーゾルバーストラテジーは、Kafka トピックおよびメッセージ情報を Service Registry のアーティファクトにマップする方法を提供します。マッピングの共通規則は、Kafka メッセージのキーと値のどちらにシリアライザーを使用するかによって、Kafka トピック名と **key** または **value** を結合することです。

ただし、Service Registry によって提供されるストラテジーを使用するか、**io.apicurio.registry.serde.strategy.ArtifactResolverStrategy** を実装するカスタム Java クラスを作成することにより、マッピングに代替規則を使用できます。

アーティファクト参照を返すストラテジー

Service Registry は、 **ArtifaceResolverStrategy** の実装に基づいてアーティファクト参照を返すために次のストラテジーを提供します。

RecordIdStrategy

スキーマのフルネームを使用する Avro 固有のストラテジー。

TopicRecordIdStrategy

トピック名およびスキーマのフルネームを使用する Avro 固有のストラテジー。

TopicIdStrategy

トピック名と、**key** または **value** 接尾辞を使用するデフォルトストラテジー。

SimpleTopicIdStrategy

トピック名のみを使用する単純なストラテジー。

DefaultSchemaResolver インターフェイス

デフォルトのスキーマリゾルバーは、アーティファクトリーゾルバーストラテジーによって提供されるアーティファクト参照の下に登録されたスキーマの特定バージョンを見つけて識別します。すべてのアーティファクトのすべてのバージョンには、グローバルで一意の識別子が 1 つだけあり、それを使用してそのアーティファクトの内容を取得できます。このグローバル ID はすべての Kafka メッセージに含まれていいるため、デシリアライザーは Apicurio レジストリーからスキーマを適切に取得できます。

デフォルトのスキーマリゾルバーは、既存のアーティファクトバージョンを検索できます。見つからない場合は、使用するストラテジーに応じて登録できます。**io.apicurio.registry.serde.SchemaResolver** を実装するカスタム Java クラスを作成することにより、独自のストラテジーを提供することもできます。ただし、**DefaultSchemaResolver** を使用して、代わりに設定プロパティーを指定することをお勧めします。

グローバル ID を返すストラテジー

DefaultSchemaResolver を使用する場合、アプリケーションのプロパティーを使用してその動作を設定できます。次の表は、一般的に使用される例を示しています。

表7.1 Service Registry のグローバル ID 設定オプション

プロパティー	タイプ	説明	デフォルト
apicurio.registry.find-latest	**boolean**	シリアライザーが対応するグループ ID およびアーティファクト ID のレジストリー内で最新のアーティファクトの検索を試行するかどうかを指定します。	**false**
apicurio.registry.use-id	**String**	指定された ID を Kafka に書き込むようシリアライザーに指示し、この ID を使用してスキーマを見つけるようにデシリアライザーに指示します。	**-**
apicurio.registry.auto-register	**boolean**	シリアライザーがレジストリーでアーティファクトを作成しようとするかどうかを指定します。JSON スキーマシリアライザーはこれをサポートしません。	**false**
apicurio.registry.check-period-ms	**String**	グローバル ID をキャッシュする期間をミリ秒単位で指定します。設定されていない場合は、グローバル ID は毎回フェッチされます。	**-**

ヒント

アプリケーションプロパティーを Java システムプロパティーとして設定することも、Quarkus **application.properties** ファイルに含めることもできます。詳細は、Quarkus のドキュメント を参照してください。

その他のリソース

- 詳細は、SerdeConfig Java class を参照しください。

7.3. SERVICE REGISTRY シリアライザー/デシリアライザーの設定

本セクションの定数例を使用して、特定のクライアントシリアライザー/デシリアライザー (SerDe) サービスおよびスキーマ検索ストラテジーを直接クライアントアプリケーションに設定できます。または、ファイルまたはインスタンスで対応する Service Registry アプリケーションプロパティーを設定できます。

以下のセクションでは、SerDe 定数および設定オプションの例を紹介します。

SerDe サービスの設定

```
public class SerdeConfig {

  public static final String REGISTRY_URL = "apicurio.registry.url"; (1)
```

```
  public static final String ID_HANDLER = "apicurio.registry.id-handler"; (2)
  public static final String ENABLE_CONFLUENT_ID_HANDLER = "apicurio.registry.as-confluent";
(3)
```

(1) Service Registry の必須の URL。

(2) ID 処理を拡張することで、他の ID 形式をサポートし、その形式に Service Registry SerDe サービスとの互換性を持たせます。たとえば、ID 形式を **Long** から **Integer** に変更すると Confluent ID 形式がサポートされます。

(3) Confluent ID の処理を簡素化します。**true** に設定すると、**Integer** がグローバル ID の検索に使用されます。この設定は、**ID_HANDLER** オプションと一緒に使用しないでください。

関連情報

- 詳細は、SerdeConfig Java class を参照してください。

SerDe 検索ストラテジーの設定

```
public class SerdeConfig {

  public static final String ARTIFACT_RESOLVER_STRATEGY = "apicurio.registry.artifact-resolver-strategy";
...
```

関連情報

- 詳細は 「Service Registry でスキーマを検索するストラテジー」 を参照してください。

Kafka コンバーターの設定

```
public class SerdeBasedConverter<S, T> extends SchemaResolverConfigurer<S, T> implements Converter, Closeable {

  public static final String REGISTRY_CONVERTER_SERIALIZER_PARAM = "apicurio.registry.converter.serializer"; (1)
  public static final String REGISTRY_CONVERTER_DESERIALIZER_PARAM = "apicurio.registry.converter.deserializer"; (2)
```

(1) Service Registry Kafka コンバーターと使用するために必要なシリアライザー。

(2) Service Registry Kafka コンバーターと使用するために必要なデシリアライザー。

その他のリソース

- 詳細は、SerdeBasedConverter Java class を参照してください。

さまざまなスキーマタイプの設定

さまざまなスキーマ技術に SerDe を設定する方法は、以下を参照してください。

- 「Service Registry を使用した Avro SerDe の設定」
- 「Service Registry を使用した JSON スキーマ SerDe の設定」

- 「Service Registry を使用した Protobuf SerDe の設定」

7.4. 異なるクライアントのシリアライザー/デシリアライザータイプの使用

Kafka クライアントアプリケーションでスキーマを使用する場合は、ユースケースに応じて、使用する特定のスキーマタイプを選択する必要があります。Service Registry は、Apache Avro、JSON スキーマ、および Google Protobuf 用の SerDe Java クラスを提供します。以下のセクションでは、各タイプを使用するように Kafka アプリケーションを設定する方法を説明します。

また、Kafka を使用してカスタムシリアライザーおよびデシリアライザークラスを実装し、Service Registry REST Java クライアントを使用して Service Registry 機能を活用することもできます。

シリアライザー/デシリアライザーの Kafka アプリケーション設定
Kafka アプリケーションで Service Registry によって提供される SerDe クラスを使用するには、正しい設定プロパティーを設定する必要があります。以下の簡単な Avro の例は、Kafka プロデューサーアプリケーションでシリアライザーを設定する方法と、Kafka コンシューマーアプリケーションでデシリアライザーを設定する方法を示しています。

Kafka プロデューサーのシリアライザー設定の例

```
// Create the Kafka producer
private static Producer<Object, Object> createKafkaProducer() {
    Properties props = new Properties();

    // Configure standard Kafka settings
    props.putIfAbsent(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, SERVERS);
    props.putIfAbsent(ProducerConfig.CLIENT_ID_CONFIG, "Producer-" + TOPIC_NAME);
    props.putIfAbsent(ProducerConfig.ACKS_CONFIG, "all");

    // Use Service Registry-provided Kafka serializer for Avro
    props.putIfAbsent(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class.getName());
    props.putIfAbsent(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
AvroKafkaSerializer.class.getName());

    // Configure the Service Registry location
    props.putIfAbsent(SerdeConfig.REGISTRY_URL, REGISTRY_URL);

    // Register the schema artifact if not found in the registry.
    props.putIfAbsent(SerdeConfig.AUTO_REGISTER_ARTIFACT, Boolean.TRUE);

    // Create the Kafka producer
    Producer<Object, Object> producer = new KafkaProducer<>(props);
    return producer;
}
```

Kafka コンシューマーのデシリアライザー設定の例

```
// Create the Kafka consumer
private static KafkaConsumer<Long, GenericRecord> createKafkaConsumer() {
    Properties props = new Properties();

    // Configure standard Kafka settings
    props.putIfAbsent(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, SERVERS);
    props.putIfAbsent(ConsumerConfig.GROUP_ID_CONFIG, "Consumer-" + TOPIC_NAME);
```

```
    props.putIfAbsent(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
    props.putIfAbsent(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
    props.putIfAbsent(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");

    // Use Service Registry-provided Kafka deserializer for Avro
    props.putIfAbsent(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class.getName());
    props.putIfAbsent(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
AvroKafkaDeserializer.class.getName());

    // Configure the Service Registry location
    props.putIfAbsent(SerdeConfig.REGISTRY_URL, REGISTRY_URL);

    // No other configuration needed because the schema globalId the deserializer uses is sent
    // in the payload. The deserializer extracts the globalId and uses it to look up the schema
    // from the registry.

    // Create the Kafka consumer
    KafkaConsumer<Long, GenericRecord> consumer = new KafkaConsumer<>(props);
    return consumer;
}
```

関連情報

- アプリケーションの例については Simple Avro example を参照してください。

7.4.1. Service Registry を使用した Avro SerDe の設定

Service Registry は、以下の Kafka クライアントシリアライザーおよび Apache Avro のデシリアライザークラスを提供します。

- **io.apicurio.registry.serde.avro.AvroKafkaSerializer**
- **io.apicurio.registry.serde.avro.AvroKafkaDeserializer**

Avro シリアライザーの設定

以下のように Avro シリアライザークラスを設定することができます。

- Service Registry の URL
- アーティファクトリーゾルバーストラテジー
- ID の場所
- ID エンコーディング
- Avro datum プロバイダー
- Avro エンコーディング

ID の場所

シリアライザーは、スキーマの一意の ID を Kafka メッセージの一部として渡し、コンシューマーがデシリアライズに正しいスキーマを使用できるようにします。ID は、メッセージのペイロードまたはメッセージヘッダーに存在できます。デフォルトの場所はメッセージペイロードです。メッセージヘッダーで ID を送信するには、以下の設定プロパティーを設定します。

```
props.putIfAbsent(SerdeConfig.ENABLE_HEADERS, "true")
```

プロパティー名は **apicurio.registry.headers.enabled** です。

ID エンコーディング

Kafka メッセージボディーに渡すときにスキーマ ID をエンコードする方法をカスタマイズできます。**apicurio.registry.id-handler** 設定プロパティーを、**io.apicurio.registry.serde.IdHandler** インターフェイスを実装するクラスに設定します。Service Registry は以下の実装を提供します。

- **io.apicurio.registry.serde.DefaultIdHandler**:ID を 8 バイト長として格納します
- **io.apicurio.registry.serde.Legacy4ByteIdHandler**:ID を 4 バイト整数として格納します

Service Registry はスキーマ ID を long として表しますが、従来の理由、または他のレジストリーまたは SerDe クラスとの互換性のため、ID の送信時に 4 バイトの使用が推奨される場合があります。

Avro datum プロバイダー

Avro は、データを読み書きするためのさまざまなデータライターとリーダーを提供します。Service Registry は、3 つの異なるタイプをサポートします。

- Generic
- Specific
- Reflect

Service Registry **AvroDatumProvider** は、使用されるタイプの抽象概念であり、**DefaultAvroDatumProvider** がデフォルトで使用されます。

以下の設定オプションを設定できます。

- **apicurio.registry.avro-datum-provider**: **AvroDatumProvider** 実装の完全修飾 Java クラス名を指定します (例えば、**io.apicurio.registry.serde.avro.ReflectAvroDatumProvider**)。
- **apicurio.registry.use-specific-avro-reader**: **DefaultAvroDatumProvider** を使用するときに特定のタイプを使用するには、**true** に設定します

Avro エンコーディング

Avro を使用してデータをシリアライズする場合は、Avro バイナリーエンコーディング形式を使用して、データを可能な限り効率的な形式でエンコードすることができます。Avro は JSON としてデータのエンコードもサポートします。これにより、ロギングやデバッグなどの各メッセージのペイロードの検証が容易になります。

apicurio.registry.avro.encoding プロパティーを **JSON** または **BINARY** の値で設定することにより、Avro エンコーディングを設定できます。デフォルトは **BINARY** です。

Avro デシリアライザーの設定

Avro デシリアライザーの設定設定を、シリアライザーの設定と一致するように、Avro デシリアライザークラスを設定する必要があります。

- Service Registry の URL
- ID エンコーディング

- Avro datum プロバイダー
- Avro エンコーディング

これらの設定オプションは、シリアライザーセクションを参照してください。プロパティー名と値は同じです。

注記

デシリアライザーの設定時には、以下のオプションは必要ありません。

- アーティファクトリーゾルバーストラテジー
- ID の場所

デシリアライザークラスは、メッセージからこれらのオプションの値を判断できます。シリアライザーはメッセージの一部として ID を送信するため、ストラテジーは必要ありません。

ID の位置は、メッセージペイロードの先頭にあるマジックバイトを確認することで決定されます。そのバイトが見つかると、設定されたハンドラーを使用してメッセージペイロードから ID が読み取られます。マジックバイトが見つからない場合、ID はメッセージヘッダーから読み込まれます。

その他のリソース

- Avro 設定の詳細は、AvroKafkaSerdeConfig Java class を参照してください。
- アプリケーションの例については Simple Avro example を参照してください。

7.4.2. Service Registry を使用した JSON スキーマ SerDe の設定

Service Registry は、以下の Kafka クライアントシリアライザーおよび Json スキーマのデシリアライザークラスを提供します。

- **io.apicurio.registry.serde.jsonschema.JsonSchemaKafkaSerializer**
- **io.apicurio.registry.serde.jsonschema.JsonSchemaKafkaDeserializer**

Apache Avro とは異なり、JSON スキーマはシリアライズテクノロジーではなく、検証テクノロジーです。その結果、JSON スキーマの設定オプションは大きく異なります。たとえば、データは常に JSON としてエンコードされるため、エンコーディングオプションはありません。

JSON スキーマシリアライザーの設定

JSON スキーマシリアライザークラスを以下のように設定できます。

- Service Registry の URL
- アーティファクトリーゾルバーストラテジー
- スキーマ検証

唯一の標準以外の設定プロパティーは、デフォルトで有効になっている JSON スキーマバリデーションです。これを無効にするには、**apicurio.registry.serde.validation-enabled** を **false** に設定します。以下に例を示します。

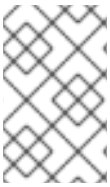
```
props.putIfAbsent(SerdeConfig.VALIDATION_ENABLED, Boolean.FALSE)
```

JSON スキーマデシリアライザーの設定

JSON スキーマデシリアライザークラスを以下のように設定できます。

- Service Registry の URL
- スキーマ検証
- データをデシリアライズするためのクラス

スキーマをロードできるように、Service Registry の場所を指定する必要があります。他の設定はオプションです。



注記

デシリアライザーの検証は、シリアライザーが Kafka メッセージでグローバル ID を渡す場合にのみ機能します。これは、シリアライザーで検証が有効になっている場合にのみ発生します。

その他のリソース

- 詳細は、JsonSchemaKafkaDeserializerConfig Java class を参照してください。
- アプリケーションの例については Simple JSON Schema example を参照してください。

7.4.3. Service Registry を使用した Protobuf SerDe の設定

Service Registry は、以下の Kafka クライアントシリアライザーおよび Google Protobuf のデシリアライザークラスを提供します。

- **io.apicurio.registry.serde.protobuf.ProtobufKafkaSerializer**
- **io.apicurio.registry.serde.protobuf.ProtobufKafkaDeserializer**

Protobuf シリアライザーの設定

Protobuf シリアライザークラスを以下のように設定できます。

- Service Registry の URL
- アーティファクトリーゾルバーストラテジー
- ID の場所
- ID エンコーディング
- スキーマ検証

これらの設定オプションの詳細は、以下のセクションを参照してください。

- 「Service Registry シリアライザー/デシリアライザーの設定」
- 「Service Registry を使用した Avro SerDe の設定」

Protobuf デシリアライザーの設定

シリアライザーの以下の設定と一致するように、Protobuf デシリアライザークラスを設定する必要があります。

- Service Registry の URL
- ID エンコーディング

設定プロパティー名と値はシリアライザーの場合と同じです。

注記

デシリアライザーの設定時には、以下のオプションは必要ありません。

- アーティファクトリーゾルバーストラテジー
- ID の場所

デシリアライザークラスは、メッセージからこれらのオプションの値を判断できます。シリアライザーはメッセージの一部として ID を送信するため、ストラテジーは必要ありません。

ID の位置は、メッセージペイロードの先頭にあるマジックバイトを確認することで決定されます。そのバイトが見つかると、設定されたハンドラーを使用してメッセージペイロードから ID が読み取られます。マジックバイトが見つからない場合、ID はメッセージヘッダーから読み込まれます。

注記

Protobuf デシリアライザーは、正確な Protobuf Message 実装へデシリアライズしません。**DynamicMessage** インスタンスへデシリアライズします。設定しない場合は、適切な API はありません。

その他のリソース

- アプリケーションの例は Protobuf Bean and Protobuf Find Latest examples を参照してください。

7.5. スキーマの SERVICE REGISTRY への登録

スキーマを Apache Avro などの適切な形式で定義したら、スキーマを Service Registry に追加できます。

以下の方法を使用してスキーマを追加できます。

- Service Registry Web コンソール
- Service Registry REST API を使用する curl コマンド
- Service Registry に付属する Maven プラグイン
- クライアントコードに加えられたスキーマ設定

スキーマを登録するまでは、クライアントアプリケーションは Service Registry を使用できません。

Service Registry Web コンソール
Service Registry のインストール時に、**ui** エンドポイントから Web コンソールに接続できます。

http://MY-REGISTRY-URL/ui

コンソールから、スキーマを追加、表示、および設定できます。また、レジストリーに無効なコンテンツが追加されないようにするルールを作成することもできます。

curl コマンドの例

```
curl -X POST -H "Content-type: application/json; artifactType=AVRO" \
  -H "X-Registry-ArtifactId: share-price" \ 1
  --data '{
    "type":"record",
    "name":"price",
    "namespace":"com.example",
    "fields":[{"name":"symbol","type":"string"},
    {"name":"price","type":"string"}]}'
  https://my-cluster-my-registry-my-project.example.com/apis/registry/v2/groups/my-group/artifacts -s
2
```

1 シンプルな Avro スキーマアーティファクト。

2 Service Registry を公開する OpenShift ルート名。

Maven プラグインの例

```
<plugin>
  <groupId>io.apicurio</groupId>
  <artifactId>apicurio-registry-maven-plugin</artifactId>
  <version>${apicurio.version}</version>
  <executions>
     <execution>
       <phase>generate-sources</phase>
       <goals>
          <goal>register</goal> 1
       </goals>
       <configuration>
          <registryUrl>http://REGISTRY-URL/apis/registry/v2</registryUrl> 2
          <artifacts>
             <artifact>
                <groupId>TestGroup</groupId> 3
                <artifactId>FullNameRecord</artifactId>
                <file>${project.basedir}/src/main/resources/schemas/record.avsc</file>
                <ifExists>FAIL</ifExists>
             </artifact>
             <artifact>
                <groupId>TestGroup</groupId>
                <artifactId>ExampleAPI</artifactId> 4
                <type>GRAPHQL</type>
                <file>${project.basedir}/src/main/resources/apis/example.graphql</file>
                <ifExists>RETURN_OR_UPDATE</ifExists>
                <canonicalize>true</canonicalize>
             </artifact>
          </artifacts>
       </configuration>
```

```
    </execution>
  </executions>
</plugin>
```

1. スキーマアーティファクトをレジストリーにアップロードするための実行目標として **register** を指定します。
2. **../apis/registry/v2** エンドポイントでService Registry URL を指定します。
3. Service Registry アーティファクトグループ ID を指定します。
4. 指定したグループ ID、アーティファクト ID、および場所を使用して複数のアーティファクトをアップロードできます。

プロデューサークライアントを使用した設定例

```
String registryUrl_node1 = PropertiesUtil.property(clientProperties, "registry.url.node1",
    "https://my-cluster-service-registry-myproject.example.com/apis/registry/v2"); (1)
try (RegistryService service = RegistryClient.create(registryUrl_node1)) {
    String artifactId = ApplicationImpl.INPUT_TOPIC + "-value";
    try {
        service.getArtifactMetaData(artifactId); (2)
    } catch (WebApplicationException e) {
        CompletionStage <ArtifactMetaData> csa = service.createArtifact(
            ArtifactType.AVRO,
            artifactId,
            new ByteArrayInputStream(LogInput.SCHEMA$.toString().getBytes())
        );
        csa.toCompletableFuture().get();
    }
}
```

1. 複数の URL ノードに対してプロパティーを登録できます。
2. アーティファクト ID に基づいてスキーマがすでに存在しているかを確認します。

7.6. KAFKA コンシューマークライアントからのスキーマの使用

この手順では、Service Registry からのスキーマを使用するように Java で書かれた Kafka コンシューマークライアントを設定する方法について説明します。

前提条件

- Service Registry がインストールされている必要があります。
- スキーマが Service Registry に登録されている必要があります。

手順

1. Service Registry の URL でクライアントを設定します。以下に例を示します。

```
String registryUrl = "https://registry.example.com/apis/registry/v2";
Properties props = new Properties();
props.putIfAbsent(SerdeConfig.REGISTRY_URL, registryUrl);
```

2. Service Registry デシリアライザーでクライアントを設定します。以下に例を示します。

```
// Configure Kafka settings
props.putIfAbsent(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, SERVERS);
props.putIfAbsent(ConsumerConfig.GROUP_ID_CONFIG, "Consumer-" + TOPIC_NAME);
props.putIfAbsent(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, "true");
props.putIfAbsent(ConsumerConfig.AUTO_COMMIT_INTERVAL_MS_CONFIG, "1000");
props.putIfAbsent(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
// Configure deserializer settings
props.putIfAbsent(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
    AvroKafkaDeserializer.class.getName()); (1)
props.putIfAbsent(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
    AvroKafkaDeserializer.class.getName()); (2)
```

(1) Service Registry によって提供されるデシリアライザー。

(2) デシリアライズは Apache Avro JSON 形式です。

7.7. KAFKA プロデューサークライアントからのスキーマの使用

この手順では、Service Registry からのスキーマを使用するように Java で書かれた Kafka プロデューサークライアントを設定する方法について説明します。

前提条件

- Service Registry がインストールされている必要があります。
- スキーマが Service Registry に登録されている必要があります。

手順

1. Service Registry の URL でクライアントを設定します。以下に例を示します。

```
String registryUrl = "https://registry.example.com/apis/registry/v2";
Properties props = new Properties();
props.putIfAbsent(SerdeConfig.REGISTRY_URL, registryUrl);
```

2. クライアントをシリアライザーで設定し、Service Registry でスキーマを検索するようにストラテジーを設定します。以下に例を示します。

```
props.put(CommonClientConfigs.BOOTSTRAP_SERVERS_CONFIG, "my-cluster-kafka-
bootstrap:9092");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
AvroKafkaSerializer.class.getName()); (1)
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
AvroKafkaSerializer.class.getName()); (2)
props.put(SerdeConfig.FIND_LATEST_ARTIFACT, FindLatestIdStrategy.class.getName());
(3)
```

1. Service Registry により提供されるメッセージキーのシリアライザー。
2. Service Registry により提供されるメッセージ値のシリアライザー。
3. スキーマのグローバル ID を検索する検索ストラテジー。

7.8. KAFKA STREAMS アプリケーションからのスキーマの使用

この手順では、Service Registry からの Apache Avro スキーマを使用するように Java で書かれた Kafka Streams クライアントを設定する方法について説明します。

前提条件

- Service Registry がインストールされている必要があります。
- スキーマが Service Registry に登録されている必要があります。

手順

1. Service Registry URL を使用して Java クライアントを作成して設定します。

```
String registryUrl = "https://registry.example.com/apis/registry/v2";

RegistryService client = RegistryClient.cached(registryUrl);
```

2. シリアライザーおよびデシリアライザーを設定します。

```
Serializer<LogInput> serializer = new AvroKafkaSerializer<LogInput>(); 1

Deserializer<LogInput> deserializer = new AvroKafkaDeserializer <LogInput>(); 2

Serde<LogInput> logSerde = Serdes.serdeFrom(
    serializer,
    deserializer
);

Map<String, Object> config = new HashMap<>();
config.put(SerdeConfig.REGISTRY_URL, registryUrl);
config.put(AvroKafkaSerdeConfig.USE_SPECIFIC_AVRO_READER, true);
logSerde.configure(config, false); 3
```

1. Service Registry によって提供される Avro シリアライザー。
2. Service Registry によって提供される Avro デシリアライザー。
3. Avro 形式でデシリアライズ用の Service Registry URL および Avro リーダーを設定します。

3. Kafka Streams クライアントを作成します。

```
KStream<String, LogInput> input = builder.stream(
    INPUT_TOPIC,
```

```
    Consumed.with(Serdes.String(), logSerde)
);
```

第8章 SERVICE REGISTRY アーティファクトの参照

本章では、Service Registry に保存されるサポート対象のアーティファクトタイプ、状態、メタデータ、およびコンテンツルールの詳細を説明します。

- 「Service Registry アーティファクトタイプ」
- 「Service Registry アーティファクトの状態」
- 「Service Registry アーティファクトのメタデータ」
- 「Service Registry コンテンツルールタイプ」
- 「Service Registry コンテンツルールの成熟度」

関連情報

- 詳細は、Apicurio Registry REST API のドキュメント を参照してください。

8.1. SERVICE REGISTRY アーティファクトタイプ

以下のアーティファクトタイプは Service Registry に保存および管理できます。

表8.1 Service Registry アーティファクトタイプ

型	説明
ASYNCAPI	AsyncAPI 仕様
AVRO	Apache Avro スキーマ
GRAPHQL	GraphQL スキーマ
JSON	JSON スキーマ
KCONNECT	Apache Kafka Connect スキーマ
OPENAPI	OpenAPI 仕様
PROTOBUF	Google プロトコルバッファースキーマ
WSDL	Web Services Definition Language
XSD	XML Schema Definition

8.2. SERVICE REGISTRY アーティファクトの状態

以下は、Service Registry の有効なアーティファクト状態です。

表8.2 Service Registry アーティファクトの状態

状態	説明
ENABLED	基本状態、全ての操作が可能です。
DISABLED	アーティファクトとそのメタデータは、Service Registry Web コンソールを使用して表示および検索できますが、そのコンテンツはどのクライアントでもフェッチできません。
非推奨	アーティファクトは完全に使用可能ですが、アーティファクトのコンテンツがフェッチされるたびに、ヘッダーが REST API 応答に追加されます。Service Registry Rest Client は、非推奨となったコンテンツが見つかったときにも警告をログに記録します。

8.3. SERVICE REGISTRY アーティファクトのメタデータ

アーティファクトが Service Registry に追加されると、メタデータプロパティーのセットがアーティファクトの内容と共に保存されます。このメタデータは、設定可能な一部のプロパティーと、生成された読み取り専用プロパティーのセットで設定されます。

表8.3 Service Registry メタデータプロパティー

プロパティー	型	編集可能
id	string	false
type	ArtifactType	false
state	ArtifactState	true
version	integer	false
createdBy	string	false
createdOn	date	false
modifiedBy	string	false
modifiedOn	date	false
name	string	true
description	string	true
labels	文字列の配列	true

プロパティー	型	編集可能
properties	map	true

アーティファクトメタデータの更新

- Service Registry REST API を使用して、メタデータエンドポイントを使用して編集可能なプロパティーのセットを更新できます。
- **state** プロパティーは、状態遷移 API を使用することでのみ編集できます。たとえば、アーティファクトを **deprecated** または **disabled** としてマークできます。

関連情報

詳細は、Apicurio Registry REST API documentation の **/artifacts/{artifactId}/meta** セクションを参照してください。

8.4. SERVICE REGISTRY コンテンツルールタイプ

Service Registry のコンテンツ展開を管理するには、以下のルールタイプを指定できます。

表8.4 Service Registry コンテンツルールタイプ

型	説明
VALIDITY	レジストリーに追加する前にデータを検証します。このルールに使用できる設定値は以下のとおりです。 - **FULL**: 検証はシンタックスとセマンティックの両方で行われます。 - **SYNTAX_ONLY**: 検証は構文のみです。

型	説明
COMPATIBILITY	新たに追加されたアーティファクトが以前に追加したバージョンと互換性があることを確認します。このルールに使用できる設定値は以下のとおりです。 ● **FULL**: 新しいアーティファクトは、最後に追加されたアーティファクトと上位および下位互換性があります。 ● **FULL_TRANSITIVE**: 新しいアーティファクトは、以前に追加されたすべてのアーティファクトと上位互換性および下位互換性があります。 ● **BACKWARD**: 新しいアーティファクトを使用するクライアントは、最後に追加されたアーティファクトを使用して書き込まれたデータを読み取りできます。 ● **BACKWARD_TRANSITIVE**: 新しいアーティファクトを使用しているクライアントは、以前に追加されたすべてのアーティファクトを使用して書き込まれたデータを読み取ることができます。 ● **FORWARD**: 最後に追加されたアーティファクトを使用するクライアントは、新しいアーティファクトを使用して書き込まれたデータを読み取りできます。 ● **FORWARD_TRANSITIVE**: 以前に追加されたすべてのアーティファクトを使用しているクライアントは、新しいアーティファクトを使用して書き込まれたデータを読み取ることができます。 ● **NONE**: 下位互換性および上位互換性チェックはすべて無効になります。

8.5. SERVICE REGISTRY コンテンツルールの成熟度

すべてのコンテンツルールは、Service Registry でサポートされるすべてのアーティファクトタイプに対して完全に実装されるわけではありません。以下の表は、各ルールおよびアーティファクトタイプの現在の成熟度レベルを示しています。

表8.5 Service Registry コンテンツルールの成熟度マトリックス

アーティファクトタイプ	検証ルール	互換性ルール
Avro	Full	Full
Protobuf	Full	なし

アーティファクトタイプ	検証ルール	互換性ルール
JSON スキーマ	Full	Full
OpenAPI	Full	なし
AsyncAPI	Syntax Only	なし
GraphQL	Syntax Only	なし
Kafka Connect	Syntax Only	なし
WSDL	Syntax Only	なし
XSD	Syntax Only	なし

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。