



Red Hat Integration 2022.Q3

Service Registry ユーザーガイド

Service Registry 2.3 の使用

Red Hat Integration 2022.Q3 Service Registry ユーザーガイド

Service Registry 2.3 の使用

Enter your first name here. Enter your surname here.

Enter your organisation's name here. Enter your organisational division here.

Enter your email address here.

法律上の通知

Copyright © 2022 | You need to change the HOLDER entity in the en-US/Service_Registry_User_Guide.ent file |.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

このガイドでは、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ルライザーとデシリアル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	5
多様性を受け入れるオープンソースの強化	5
第1章 SERVICE REGISTRY の概要	6
1.1. SERVICE REGISTRY とは	6
Service Registry の機能	7
1.2. SERVICE REGISTRY のスキーマと API アーティファクト	7
スキーマと API のグループ	8
他のスキーマと API への参照	8
サポートされているアーティファクトタイプ	9
1.3. SERVICE REGISTRY WEB コンソールを使用したコンテンツの管理	10
1.4. SERVICE REGISTRY コア REST API	11
他のスキーマレジストリー REST API との互換性	11
1.5. SERVICE REGISTRY ストレージのオプション	12
1.6. スキーマと JAVA クライアントシリアルライザー/デシリアルライザーを使用した KAFKA メッセージの検証	12
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	13
1.8. SERVICE REGISTRY デモ例	14
1.9. SERVICE REGISTRY で利用可能なディストリビューション	15
第2章 SERVICE REGISTRY のコンテンツルール	17
2.1. ルールを使用したレジストリーコンテンツの管理	17
2.1.1. ルールの適用時	17
2.1.2. ルールの優先順位	17
2.1.3. ルールの仕組み	18
2.1.4. コンテンツルールの設定	18
アーティファクトルールの設定	18
グローバルルールの設定	18
第3章 WEB コンソールを使用した SERVICE REGISTRY コンテンツの管理	20
3.1. SERVICE REGISTRY WEB コンソールを使用したアーティファクトの表示	20
3.2. SERVICE REGISTRY WEB コンソールを使用したアーティファクトの追加	22
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	24
3.4. WEB コンソールを使用した SERVICE REGISTRY インスタンス設定の設定	25
3.5. SERVICE REGISTRY WEB コンソールを使用したアーティファクト所有者の変更	27
3.6. SERVICE REGISTRY WEB コンソールを使用したレジストリーコンテンツのエクスポートとインポート	28
第4章 REST API を使用した SERVICE REGISTRY コンテンツの管理	29
4.1. SERVICE 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	29
4.2. SERVICE 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	30
4.3. SERVICE REGISTRY REST API コマンドを使用したアーティファクト参照の管理	31
4.4. SERVICE 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とインポート	34
第5章 MAVEN プラグインを使用した SERVICE REGISTRY コンテンツの管理	36
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	36
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	37
5.3. SERVICE REGISTRY MAVEN プラグインを使用したアーティファクト参照の追加	38
5.4. 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	40
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	43
6.1. SERVICE REGISTRY JAVA クライアント	43
6.2. SERVICE REGISTRY JAVA クライアントアプリケーションの作成	43

6.3. SERVICE REGISTRY JAVA クライアント設定	44
カスタムヘッダー設定	44
TLS 設定オプション	44
第7章 JAVA クライアントでシリアライザー/デシリアライザーを使用した KAFKA メッセージの検証	46
7.1. KAFKA クライアントアプリケーションおよび SERVICE REGISTRY	46
Service Registry スキーマテクノロジー	46
プロデューサースキーマの設定	47
コンシューマスキーマの設定	47
7.2. SERVICE REGISTRY でスキーマを検索するストラテジー	48
アーティファクトリーゾルバストラテジー	49
アーティファクトへの参照を返すストラテジー	49
DefaultSchemaResolver インターフェイス	49
レジストリー検索オプションの設定	49
7.3. スキーマの SERVICE REGISTRY への登録	50
Service Registry Web コンソール	50
curl コマンドの例	51
Maven プラグインの例	51
プロデューサークライアントを使用した設定例	52
7.4. KAFKA コンシューマクライアントからのスキーマの使用	52
7.5. KAFKA プロデューサークライアントからのスキーマの使用	53
7.6. KAFKA STREAMS アプリケーションからのスキーマの使用	53
第8章 JAVA クライアントでの KAFKA シリアライザー/デシリアライザーの設定	55
8.1. クライアントアプリケーションの SERVICE REGISTRY シリアライザー/デシリアライザーの設定	55
SerDe サービスの設定	55
SerDe 検索ストラテジーの設定	56
Kafka コンバーターの設定	56
さまざまなスキーマタイプの設定	56
8.2. SERVICE REGISTRY シリアライザー/デシリアライザー設定プロパティ	57
SchemaResolver インターフェイス	57
DefaultSchemaResolver クラス	57
レジストリー API アクセスオプションの設定	57
レジストリー検索オプションの設定	59
Kafka でレジストリーアーティファクトを読み書きするための設定	62
デシリアライザーのフォールバックオプションの設定	63
8.3. さまざまなクライアントシリアライザー/デシリアライザータイプの設定方法	65
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	65
8.3.1. Service Registry を使用した Avro SerDes の設定	66
8.3.2. Service Registry を使用した JSON スキーマ SerDes の設定	69
8.3.3. Service Registry を使用した Protobuf SerDes の設定	71
第9章 SERVICE REGISTRY アーティファクトの参照	74
9.1. SERVICE REGISTRY アーティファクトタイプ	74
9.2. SERVICE REGISTRY アーティファクトの状態	74
9.3. SERVICE REGISTRY アーティファクトのメタデータ	75
9.4. SERVICE REGISTRY コンテンツルールタイプ	76
9.5. SERVICE REGISTRY コンテンツルールの成熟度	77
9.6. SERVICE REGISTRY コンテンツルールの優先順位	78
付録A サブスクリプションの使用	79
アカウントへのアクセス	79
サブスクリプションのアクティベート	79
ZIP および TAR ファイルのダウンロード	79

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 とは」
- 「Service Registry のスキーマと API アーティファクト」
- 「Service Registry Web コンソールを使用したコンテンツの管理」
- 「Service Registry コア REST API」
- 「Service Registry ストレージのオプション」
- 「スキーマと Java 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 Kafka メッセージの検証」
- 「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」

1.1. SERVICE REGISTRY とは

Service Registry は、イベント駆動型および API の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します。

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

- Apache Avro、JSON スキーマ、Protobuf、AsyncAPI、OpenAPI などの標準イベントスキーマおよび 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
- 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
- Web コンソール、REST API、コマンドライン、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
- 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)
- 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
- 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
- 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 のスキーマと API アーティファクト

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。

Avro スキーマの例

```
{
  "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。

スキーマと API のグループ

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、Protobuf、Avro、JSON スキーマ、OpenAPI、または AsyncAPI アーティファクトをすべて同じグループに含めることができます。

Service Registry Web コンソール、コア REST API、コマンドライン、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グループを作成できます。次の簡単な例は、レジストリーコア REST API の使用を示しています。

```
$ 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を作成し、アーティファクト ID が **share-price** の 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。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** グループを指定します。

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](#) を参照してください。
- Service Registry コア REST API の詳細は、[Apicurio Registry REST API のドキュメント](#) を参照してください。

他のスキーマと API への参照

一部の Service Registry アーティファクトタイプには、別のアーティファクトファイルへの **アーティファクト参照** を含めることができます。再利用可能なスキーマまたは API コンポーネントを定義し、それらを複数の場所から参照して効率を高めることができます。たとえば、**\$ref** ステートメントを使用して JSON スキーマまたは OpenAPI で参照を指定したり、**import** ステートメントを使用して Google protobuf で参照を指定したり、ネストされた名前空間を使用して Apache Avro で参照を指定したりできます。

次の例は、ネストされた名前空間を使用して **Exchange** という名前の別のスキーマへの参照を含む、**TradeKey** という名前の単純な Avro スキーマを示しています。

ネストされた Exchange スキーマを持つ TradeKey スキーマ

```
{
  "namespace": "com.kubetrade.schema.trade",
  "type": "record",
  "name": "TradeKey",
  "fields": [
    {
      "name": "exchange",
      "type": "com.kubetrade.schema.common.Exchange"
    },
    {
      "name": "key",
      "type": "string"
    }
  ]
}
```

交換スキーマ

```
{
  "namespace": "com.kubetrade.schema.common",
  "type": "enum",
  "name": "Exchange",
  "symbols": ["GEMINI"]
}
```

アーティファクト参照は、アーティファクトタイプ固有の参照から内部 Service Registry 参照にマップされるアーティファクトメタデータのコレクションとして Service Registry に格納されます。Service Registry の各アーティファクト参照は、以下で構成されます。

- グループ ID
- アーティファクト ID
- アーティファクトのバージョン
- アーティファクト参照名

Service Registry コア REST API、Maven プラグイン、および Java シリアライザー/デシリアライザー (SerDes) を使用して、アーティファクト参照を管理できます。Service Registry は、アーティファクトコンテンツとともにアーティファクト参照を保存します。Service Registry は、すべてのアーティファクトリーファレンスのコレクションも保持しているため、コレクションを検索したり、特定のアーティファクトのすべてのリファレンスを一覧表示したりできます。

サポートされているアーティファクトタイプ

Service Registry は現在、次のアーティファクトタイプのアーティファクトリーファレンスのみをサポートしています。

- Avro
- Protobuf
- JSON スキーマ

関連情報

- アーティファクト参照の管理の詳細は、以下を参照してください。
 - [4章REST API を使用した Service Registry コンテンツの管理](#)
- Java コードの例については、[Apicurio Registry SerDes with references](#) を参照してください。

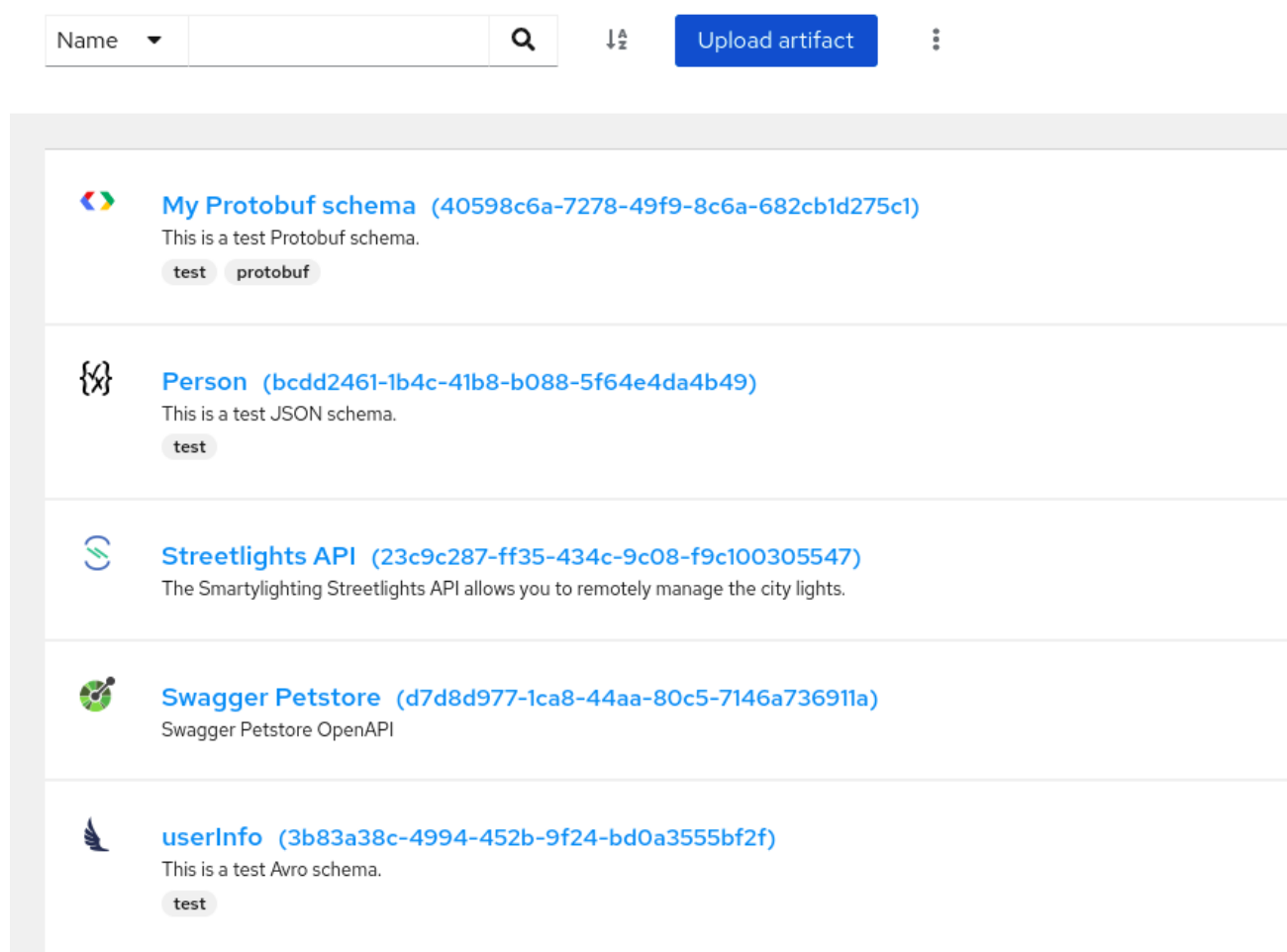
1.3. SERVICE REGISTRY WEB コンソールを使用したコンテンツの管理

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。

また、レジストリーコンテンツのオプションのルールを、グローバルに、およびスキーマと API アーティファクトごとに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、[9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 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. SERVICE REGISTRY コア REST API

クライアントアプリケーションは、Service Registry コア v2 REST API の使用を参照して、Service Registry のスキーマと 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 は、それぞれの REST API の実装を含めることで、次のスキーマレジストリーとの互換性を提供します。

- Service Registry コア v1
- Confluent スキーマレジストリー v6
- IBM Event Streams スキーマレジストリー v1
- CNCF CloudEvents スキーマレジストリー v0

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。詳細については、[Confluent Schema Registry の置換](#) を参照してください。

関連情報

- 詳細な参照情報は、[Apicurio Registry REST API 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 インスタンス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。

表1.1 Service Registry データストレージオプション

ストレージオプション	説明
PostgreSQL 12 database	PostgreSQL は、実稼働環境でのパフォーマンス、安定性、およびデータ管理 (バックアップ/復元など) に推奨されるデータストレージオプションです。
AMQ Streams 2.1	Kafka ストレージは、データベース管理の専門知識が利用できない実稼働環境、または Kafka のストレージが特定の要件である実稼働環境向けに提供されています。

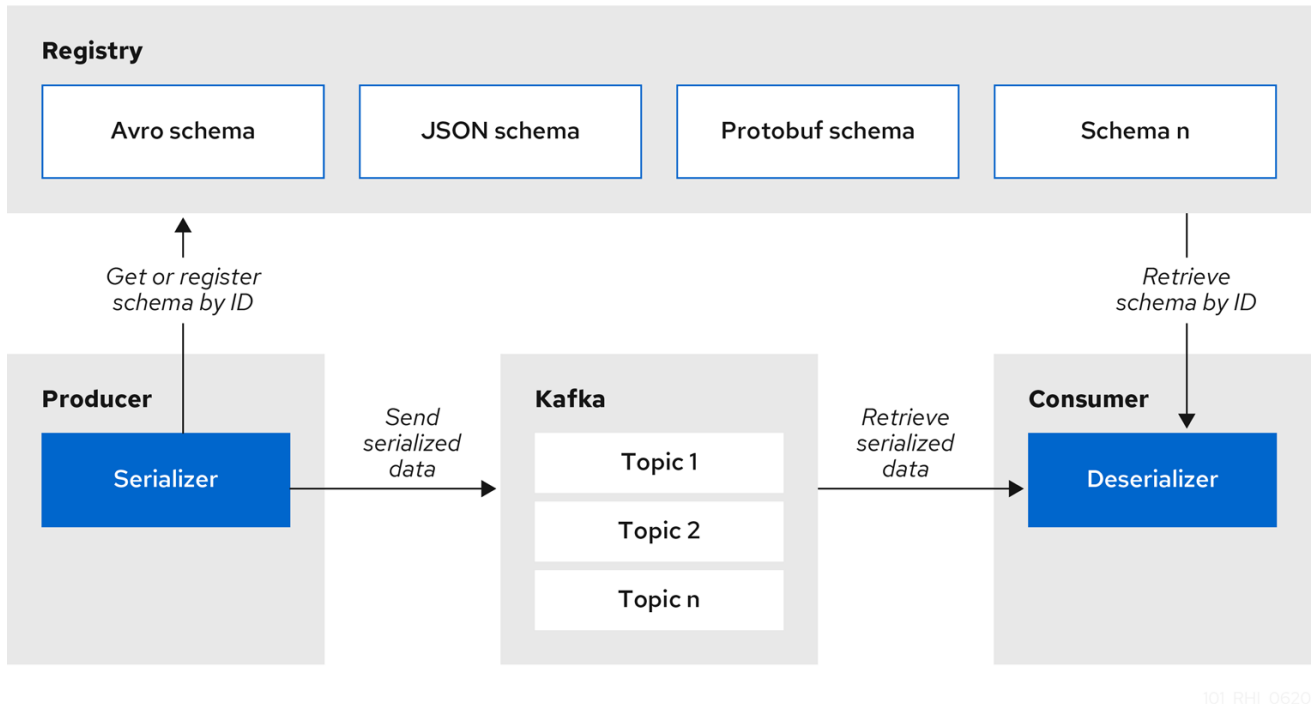
関連情報

- ストレージオプションの詳細は、[OpenShift での Service Registry のインストールおよびデプロイ](#) を参照してください。

1.6. スキーマと JAVA 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 KAFKA メッセージの検証

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 SerDes 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
- JSON スキーマ

Service Registry Maven リポジトリ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 クライアントアプリケーション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。

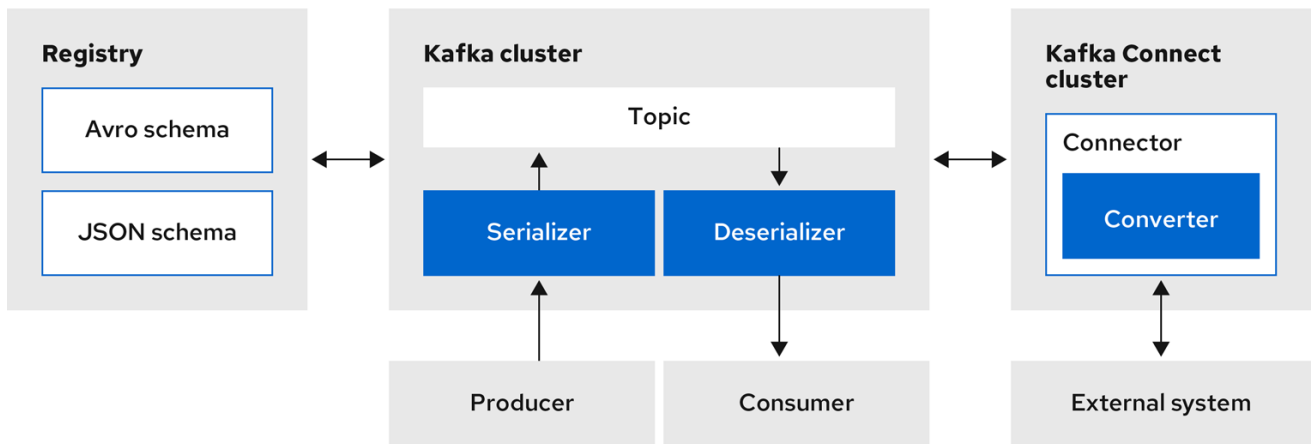
関連情報

- [7章 Java クライアントでシリアライザー/デシリアライザーを使用した Kafka メッセージの検証](#)

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



101_RHL_0620

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 形式にシリアル化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 SerDes 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。

関連情報

- [Apache Kafka Connect のドキュメント](#)
- [Apache Avro シリアライゼーションを使用するように Debezium を設定する](#)
- [Camel Kafka Connector のスタートガイド](#)
- [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 Service Registry を使用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 (SerDes) 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。

- Apache Avro SerDes
- Google Protobuf SerDes
- JSON Schema SerDes
- Confluent SerDes
- Custom ID strategy
- REST client
- Cloud Events
- Quarkus と Kafka
- Quarkus と Kafka を使用した Apache Camel

関連情報

- 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

Service Registry には、次のディストリビューションオプションがあります。

表1.2 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
Service Registry Operator	Operators → OperatorHub の OpenShift Web コンソール	一般公開 (GA)
Service Registry Operator のコンテナイメージ	Red Hat Ecosystem Catalog	一般公開 (GA)
AMQ Streams での Kafka ストレージのコンテナイメージ	Red Hat Ecosystem Catalog	一般公開 (GA)
PostgreSQL でのデータベースストレージのコンテナイメージ	Red Hat Ecosystem Catalog	一般公開 (GA)

表1.3 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 ソフトウェアのダウンロード	一般公開 (GA)
Service Registry v1 to v2 migration tool	Red Hat ソフトウェアのダウンロード	一般公開 (GA)
Maven repository	Red Hat ソフトウェアのダウンロード	一般公開 (GA)
Source code	Red Hat ソフトウェアのダウンロード	一般公開 (GA)
Kafka Connect converters	Red Hat ソフトウェアのダウンロード	一般公開 (GA)



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」](#)
- [「ルールの優先順位」](#)
- [「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、任意のコンテンツルールを有効にできます。

2.1.1. 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。

2.1.2. ルールの優先順位

Service Registry のコンテンツルールは、グローバルレベルおよびアーティファクトレベルで設定できます。優先順位は次のとおりです。

- Artifact ルールと同等のグローバルルールを有効にすると、Artifact ルールがグローバルルールをオーバーライドします。

- Artifact ルールを無効にして、同等のグローバルルールを有効にすると、グローバルルールが適用されます。
- アーティファクトレベルおよびグローバルレベルでルールを無効にすると、すべてのアーティファクトのルールが無効になります。
- アーティファクトレベルでルール値を **NONE** に設定すると、有効なグローバルルールがオーバーライドされます。この場合、アーティファクトルール値 **NONE** がこのアーティファクトでは優先されますが、有効なグローバルルールは、ルールがアーティファクトレベルで無効になっている他のすべてのアーティファクトに引き続き適用されます。

2.1.3. ルールの仕組み

各ルールには、名前と設定情報があります。レジストリーは、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、ルール実装の名前と設定で構成されます。

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。

その他のリソース

詳細は、[9章Service Registry アーティファクトの参照](#) を参照してください。

2.1.4. 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。



注記

これらのアプリケーションプロパティは、Java システムプロパティ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表示」
- 「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」
- 「Web コンソールを使用した Service Registry インスタンス設定の設定」
- 「Service Registry Web コンソールを使用したアーティファクト所有者の変更」
- 「Service Registry Web コンソールを使用したレジストリーコンテンツのエクスポートとインポート」

3.1. 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 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。

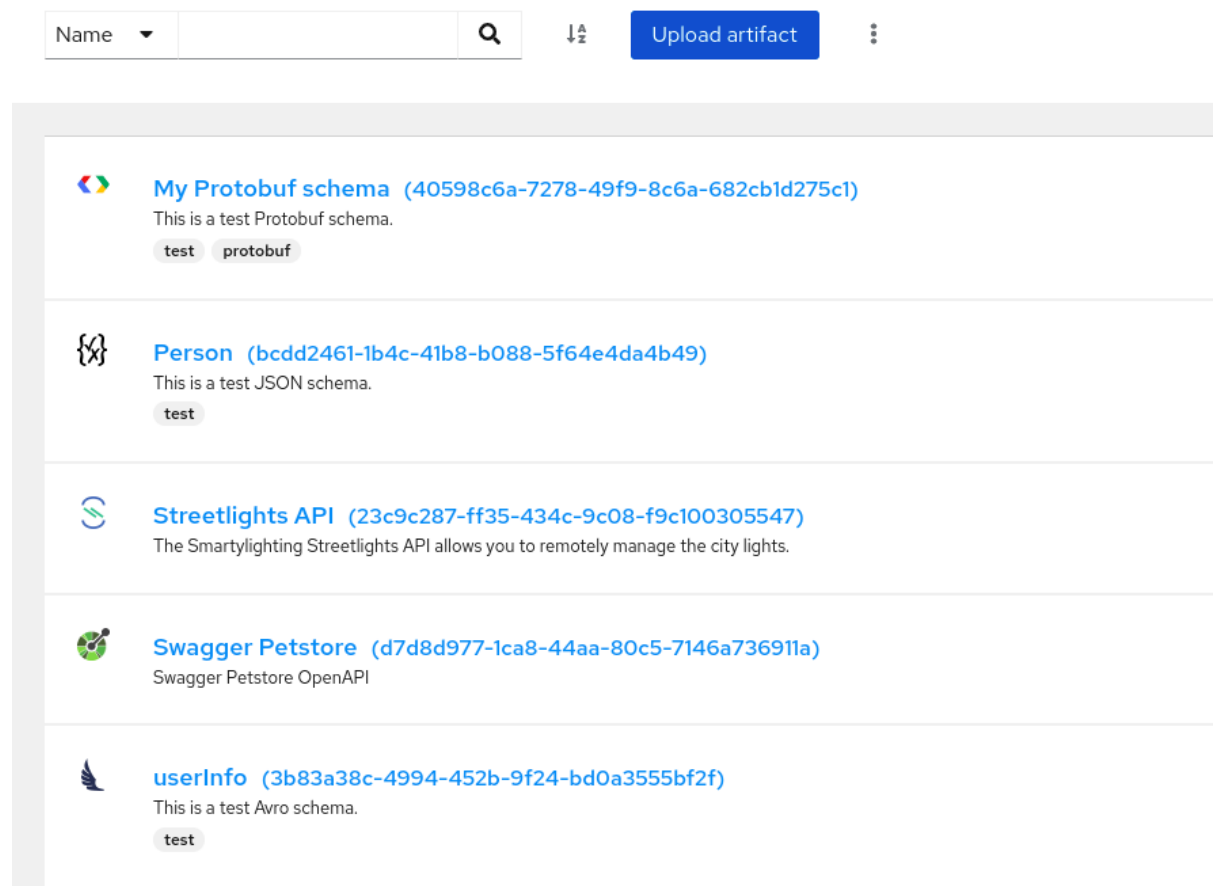
前提条件

- Service Registry が環境にインストールされ、実行している。
- Service Registry Web コンソールにログインしている。
`http://MY_REGISTRY_URL/ui`
- Web コンソール、コマンドライン、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 Service Registry に追加されました。

手順

1. **アーティファクト** タブで、Service Registry に保存されているアーティファクトのリストを参照するか、検索文字列を入力してアーティファクトを見つけます。リストから選択して、名前、グループ、ラベル、グローバル ID などの特定の条件で検索できます。

図3.1 Service Registry Web コンソールでのアーティファクト



2. アーティファクトをクリックすると、次の詳細が表示されます。
 - **概要:** 名前、オプションのグループと ID、グローバル ID、コンテンツ ID、ラベル、プロパティなどのアーティファクトバージョンのメタデータを表示します。また、アーティファクトコンテンツに対して設定できる有効性および互換性のルールも表示されます。
 - **ドキュメント** (OpenAPI および AsyncAPI のみ): 自動生成された 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。JSON コンテンツの場合、JSON または YAML をクリックして、好みの形式を表示できます。
3. このアーティファクトの追加バージョンが追加されている場合は、ページヘッダーのバージョンリストから選択できます。
4. アーティファクトの内容を **my-protobuf-schema.proto** などのローカルファイルに保存するには、ページの最後にある **Download** 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」](#)
- [「Service Registry Web コンソールを使用したコンテンツルールの設定」](#)
- [9章Service Registry アーティファクトの参照](#)

3.2. 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 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。このセクションでは、Service Registry アーティファクトをアップロードし、新しいアーティファクトバージョンを追加する簡単な例を示します。

前提条件

- Service Registry が環境にインストールされ、実行している。
- Service Registry Web コンソールにログインしている。
`http://MY_REGISTRY_URL/ui`

手順

1. **アーティファクト** タブで **アーティファクトのアップロード** をクリックし、次の詳細を指定します。
 - **Group & ID:** デフォルトの空の設定を使用して、アーティファクト ID を自動的に生成し、アーティファクトを **デフォルト** のアーティファクトグループに追加します。または、オプションのアーティファクトグループ名または ID を入力することもできます。
 - **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、リスト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 は、**Kafka Connect Schema** 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:** 次のいずれかのオプションを使用して、アーティファクトの場所を指定します。
 - **ファイルから:** **Browse** をクリックしてファイルを選択するか、ファイルをドラッグアンドドロップします。たとえば、**my-openapi.json** または **my-schema.proto** です。
 - **URL から:** 有効かつアクセス可能な URL を入力し、**Fetch** をクリックします。例: **https://petstore3.swagger.io/api/v3/openapi.json**。
2. **Upload** をクリックし、アーティファクトの詳細を表示します。
 - **概要:** 名前、アーティファクト ID、グローバル ID、コンテンツ ID、ラベル、プロパティなどのアーティファクトバージョンのメタデータを表示します。また、アーティファクトコンテンツに対して設定できる有効性および互換性のルールも表示されます。
 - **ドキュメント** (OpenAPI および AsyncAPI のみ): 自動生成された 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。JSON コンテンツの場合、**JSON** または **YAML** をクリックして、好みの形式を表示できます。次の例は、Protobuf スキーマアーティファクトの例を示しています。



図3.2 Service Registry Web コンソールのアーティファクトの詳細

Artifacts > 40598c6a-7278-49f9-8c6a-682cb1d275c1

My Protobuf schema

Version: latest ▼ Delete Upload new version

Info Content

 **Version metadata**  [Edit](#)

Name
My Protobuf schema

ID
40598c6a-7278-49f9-8c6a-682cb1d275c1

Description
This is a test Protobuf schema.

Status
ENABLED

Created
2 hours ago

Modified
2 hours ago

Global ID
1

Content ID
1

Labels
test protobuf

Properties
my-key=my-value

3. **Overview** タブで、**Edit** 鉛筆アイコンをクリックして、名前や説明などのアーティファクトメタデータを編集します。
オプションで、検索用のラベルのコンマ区切りリストを入力したり、アーティファクトに関連付けられた任意のプロパティのキーと値のペアを追加したりすることもできます。プロパティを追加するには、次の手順を実行します。
 - a. **Add property** をクリックします。
 - b. キー名と値を入力します。
 - c. 複数のプロパティを追加するには、最初の 2 つの手順を繰り返します。
 - d. **Save** をクリックします。
4. アーティファクトの内容を **my-protobuf-schema.proto** などのローカルファイルに保存するには、ページの最後にある **Download** をクリックします。

5. 新しいアーティファクトバージョンを追加するには、ページヘッダーで **Upload new version** をクリックし、ドラッグアンドドロップするか **Browse** をクリックして、**my-avro-schema.json** や **my-openapi.json** などのファイル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、ページヘッダーの **Delete** 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。

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」](#)
- [9章Service Registry アーティファクトの参照](#)

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 Service Registry 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。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。

前提条件




- Service Registry が環境にインストールされ、実行している。
- Service Registry Web コンソールにログインしている。
http://MY_REGISTRY_URL/ui
- Web コンソール、コマンドライン、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 Service Registry に追加されました。

手順

1. **Artifacts** タブで、Service Registry のアーティファクトのリストを参照するか、検索文字列を入力してアーティファクトを見つけます。リストから選択して、アーティファクト名、グループ、ラベル、またはグローバル ID などの特定の基準で検索できます。
2. アーティファクトをクリックして、そのバージョンの詳細とコンテンツルールを表示します。
3. **Content rules** で **Enable** をクリックして、アーティファクトコンテンツの有効性ルールまたは互換性ルールを設定し、一覧から適切なルール設定を選択します (有効性ルールの場合は **Full** など)。

図3.3 Service Registry Web コンソールのアーティファクトコンテンツルール

Content rules

 Validity rule	Ensure that content is <i>valid</i> when updating this artifact.	Full ▼ 
 Compatibility rule	Enforce a compatibility level when updating this artifact (for example, Backwards Compatibility).	Enable

4. グローバルルールにアクセスするには、Service Registry インスタンスをクリックし、**Global rules** タブをクリックします。**Enable** をクリックして、すべてのアーティファクトコンテンツに対してグローバルな妥当性ルールまたは互換性ルールを設定し、リストから適切なルール設定を選択します。
5. 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。

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」](#)
- [9章Service Registry アーティファクトの参照](#)

3.4. WEB コンソールを使用した SERVICE REGISTRY インスタンス設定の設定

管理者は、Service Registry Web コンソールを使用して、実行時に Service Registry インスタンスの動的設定を指定できます。認証、承認、API 互換性などの機能の設定オプションを管理できます。



注記

Service Registry インスタンスのデプロイ時に認証がすでに有効になっている場合、認証と承認の設定は Web コンソールにのみ表示されます。詳細は、[OpenShift での Service Registry のインストールおよびデプロイ](#) を参照してください。

前提条件

- Service Registry インスタンスがすでにデプロイされている。
- 管理者アクセスで Service Registry Web コンソールにログインしている。
`http://MY_REGISTRY_URL/ui`

手順

1. Service Registry Web コンソールで、**Settings** タブをクリックします。
2. この Service Registry インスタンスに対して設定するオプションを選択します。

表3.1 認証設定

設定	説明
HTTP Basic 認証	認証が有効な場合のみ表示されます。選択すると、Service Registry ユーザーは、OAuth に加えて HTTP 基本認証を使用して認証できます。デフォルトでは選択されていません。

表3.2 認可設定

設定	説明
匿名の読み取りアクセス	すでに認証が選択されている場合のみ表示されます。選択すると、Service Registry は、認証情報がない匿名ユーザーからの要求に対して、読み取り専用アクセスを許可します。この設定は、このインスタンスを使用してスキーマまたは API を外部に公開する場合に役立ちます。デフォルトでは選択されていません。
アーティファクトの所有者のみの承認	認証が有効な場合のみ表示されます。選択すると、アーティファクトを作成したユーザーのみがそのアーティファクトを変更できます。デフォルトでは選択されていません。
アーティファクトグループの所有者のみの承認	認証がすでに有効で、 Artifact 所有者のみの許可 が選択されている場合にのみ表示されます。選択すると、アーティファクトグループを作成したユーザーのみが、そのアーティファクトグループへの書き込みアクセス権 (そのグループのアーティファクトの追加や削除など) を持ちます。デフォルトでは選択されていません。
認証された読み取りアクセス	認証が有効な場合のみ表示されます。選択すると、Service Registry は、ユーザーロールに関係なく、認証されたユーザーからのリクエストに対して、少なくとも読み取り専用アクセスを許可します。デフォルトでは選択されていません。

表3.3 互換性設定

設定	説明
レガシー ID モード (互換 API)	選択すると、Confluent Schema Registry 互換性 API は contentId の代わりに globalId をアーティファクト識別子として使用します。この設定は、v1 Core Registry API をベースにする、従来の Service Registry インスタンスから移行する場合に役立ちます。デフォルトでは選択されていません。

表3.4 Web コンソールの設定

設定	説明
ダウンロードリンクの有効期限	インスタンスからアーティファクトデータをエクスポートする場合など、セキュリティ上の理由で有効期限が切れる前に、生成された .zip ダウンロードファイルへのリンクがアクティブである秒数。デフォルトは 30 秒です。

設定	説明
UI 読み取り専用モード	選択すると、Service Registry Web コンソールが読み取り専用を設定され、作成、読み取り、更新、または削除操作が禁止されます。Core Registry API を使用して行われた変更は、この設定の影響を受けません。デフォルトでは選択されていません。

関連情報

- [OpenShift での Service Registry のインストールおよびデプロイ](#)

3.5. SERVICE REGISTRY WEB コンソールを使用したアーティファクト所有者の変更

管理者として、またはスキーマまたは API アーティファクトの所有者として、Service Registry Web コンソールを使用して、アーティファクトの所有者を別のユーザーアカウントに変更できます。

たとえば、この機能は、所有者または管理者のみがアーティファクトを変更できるように、**Settings** タブで Service Registry インスタンスに対して **アーティファクトの所有者のみの許可** オプションが設定されている場合に役立ちます。所有者ユーザーが組織を離れた場合、または所有者アカウントが削除された場合は、所有者の変更が必要になる場合があります。



注記

アーティファクトの所有者のみの承認 設定とアーティファクトの **所有者** フィールドは、Service Registry インスタンスのデプロイ時に認証が有効になっている場合に **のみ** 表示されます。詳細は、[Installing and deploying Service Registry on OpenShift](#) を参照してください。

前提条件

- Service Registry インスタンスがデプロイされ、アーティファクトが作成されている。
- アーティファクトの現在の所有者または管理者として Service Registry Web コンソールにログインしている。

`http://MY_REGISTRY_URL/ui`

手順

1. **アーティファクト** タブで、Service Registry に保存されているアーティファクトのリストを参照するか、検索文字列を入力してアーティファクトを見つけます。リストから選択して、名前、グループ、ラベル、グローバル ID などの条件で検索できます。
2. 再割り当てするアーティファクトをクリックします。
3. **Version metadata** セクションで、**Owner** フィールドの横にある鉛筆アイコンをクリックします。
4. **New owner** フィールドで、アカウント名を選択または入力します。
5. **Change owner** をクリックします。

関連情報

- [OpenShift での Service Registry のインストールおよびデプロイ](#)

3.6. SERVICE REGISTRY WEB コンソールを使用したレジストリーコンテンツのエクスポートとインポート

管理者は、Service Registry Web コンソールを使用して、ある Service Registry インスタンスからデータをエクスポートし、それを別の Service Registry インスタンスにインポートできます。この機能を使用して、異なるインスタンス間でデータを簡単に移行できます。

次の例は、Service Registry インスタンス間で **.zip** ファイル内の既存のデータをエクスポートおよびインポートする方法を示しています。Service Registry インスタンスに含まれるすべてのアーティファクトデータは、**.zip** ファイルにエクスポートされます。



注記

別の Service Registry インスタンスからエクスポートされた Service Registry データのみをインポートできます。

前提条件

- Service Registry インスタンスは次のように作成されている。
 - エクスポート元のソースインスタンスには、少なくとも1つのスキーマまたは API アーティファクトが含まれている
 - インポート先のターゲットインスタンスは、一意の ID を保持するために空である
- 管理者アクセスで Service Registry Web コンソールにログインしている。
http://MY_REGISTRY_URL/ui

手順

1. ソース Service Registry インスタンスの Web コンソールで、**Artifacts** タブを表示します。
2. **Upload artifact** の横にあるオプションアイコン (3 つの縦リーダー) をクリックし、**Download all artifacts (.zip file)** を選択して、このインスタンスのレジストリーデータを **.zip** ダウンロードファイルにエクスポートします。
3. ターゲット Service Registry インスタンスの Web コンソールで、**Artifacts** タブを表示します。
4. **Upload artifact** の横にあるオプションアイコンをクリックし、**Upload multiple artifacts** を選択します。
5. 以前にエクスポートした **.zip** ダウンロードファイルをドラッグアンドドロップするか、参照します。
6. **Upload** をクリックして、データがインポートされるまで待ちます。

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。

- [「Service 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」](#)
- [「Service 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」](#)
- [「Service Registry REST API コマンドを使用したアーティファクト参照の管理」](#)
- [「Service 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とインポート」](#)

前提条件

- [1章 Service Registry の概要](#)

その他のリソース

- [Apicurio Registry REST API ドキュメント](#)

4.1. SERVICE 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

このセクションでは、Service Registry コア REST API を使用して、Service Registry で Apache Avro スキーマアーティファクトを追加および取得する簡単な curl ベースの例を示します。

前提条件

- Service Registry が環境にインストールされ、実行している

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" \
-H "Authorization: Bearer $ACCESS_TOKEN" \
--data '{"type": "record", "name": "price", "namespace": "com.example", \
  "fields": [{"name": "symbol", "type": "string"}, {"name": "price", "type": "string"}]}' \
MY-REGISTRY-URL/apis/registry/v2/groups/my-group/artifacts
```

- 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。
 - **MY-REGISTRY-URL** 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**。
 - 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",
  "type": "AVRO",
  "globalId": 2,
  "state": "ENABLED",
  "groupId": "my-group",
  "contentId": 2
}
```

- 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。
 - 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 -H "Authorization: Bearer $ACCESS_TOKEN" \
  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. SERVICE 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 コア 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、実行している

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" \
-H "Authorization: Bearer $ACCESS_TOKEN" \
--data '{"type": "record", "name": "p", "namespace": "com.example", \
"fields": [{"name": "symbol", "type": "string"}, {"name": "price", "type": "string"}]}' \
MY-REGISTRY-URL/apis/registry/v2/groups/my-group/artifacts
```

- 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。
 - **MY-REGISTRY-URL** 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**。
 - 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",
"type": "AVRO", "globalId": 3, "state": "ENABLED", "groupId": "my-group", "contentId": 3}
```

- 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。
 - 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 -H "Authorization: Bearer $ACCESS_TOKEN" \
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. SERVICE REGISTRY REST API コマンドを使用したアーティファクト参照の管理

Apache Avro、Protobuf、JSON スキーマなどの Service Registry アーティファクトタイプには、あるアーティファクトファイルから別のアーティファクトファイルへの **アーティファクト参照** を含めることができます。再利用可能なスキーマと API アーティファクトを定義し、それらを複数の場所から参照することで、効率を高めることができます。

このセクションでは、Service Registry コア REST API を使用して、Service Registry の単純な Avro スキーマアーティファクトへのアーティファクト参照を追加および取得する簡単な curl ベースの例を示します。

この例では、最初に **ItemId** という名前のスキーマアーティファクトを作成します。

ItemId スキーマ

```
{
  "namespace": "com.example.common",
  "name": "ItemId",
  "type": "record",
  "fields": [
    {
      "name": "id",
      "type": "int"
    }
  ]
}
```

次に、この例では、ネストされた **ItemId** アーティファクトへの参照を含む、**Item** という名前のスキーマアーティファクトを作成します。

ネストされた ItemId スキーマを持つアイテムスキーマ

```
{
  "namespace": "com.example.common",
  "name": "Item",
  "type": "record",
  "fields": [
    {
      "name": "itemId",
      "type": "com.example.common.ItemId"
    }
  ]
}
```

前提条件

- Service Registry が環境にインストールされ、実行している

手順

1. **/groups/{group}/artifacts** オペレーションを使用して、ネストされたアーティファクト参照を作成する **ItemId** スキーマアーティファクトを追加します。

```
$ curl -X POST MY-REGISTRY-URL/apis/registry/v2/groups/my-group/artifacts \
-H "Content-Type: application/json; artifactType=AVRO" \
-H "X-Registry-ArtifactId: ItemId" \
```

```
-H "Authorization: Bearer $ACCESS_TOKEN" \
--data '{"namespace": "com.example.common", "type": "record", "name": "ItemId", "fields": [{"name": "id", "type": "int"}]}'
```

- この例では、**ItemId** 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。
 - **MY-REGISTRY-URL** 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**。
 - 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 ボディが含まれていることを確認します。以下に例を示します。

```
{ "name": "ItemId", "createdBy": "", "createdOn": "2022-04-14T10:50:09+0000", "modifiedBy": "", "modifiedOn": "2022-04-14T10:50:09+0000", "id": "ItemId", "version": "1", "type": "AVRO", "globalId": 1, "state": "ENABLED", "groupId": "my-group", "contentId": 1, "references": [] }
```

3. **/groups/{group}/artifacts** 操作を使用して、**ItemId** スキーマへのアーティファクト参照を含む **アイテム** スキーマアーティファクトを追加します。

```
$ curl -X POST MY-REGISTRY-URL/apis/registry/v2/groups/my-group/artifacts \
-H 'Content-Type: application/create.extended+json' \
-H "X-Registry-ArtifactId: Item" \
-H 'X-Registry-ArtifactType: AVRO' \
-H "Authorization: Bearer $ACCESS_TOKEN" \
--data-raw '{
  "content": "{\r\n  \"namespace\": \"com.example.common\", \r\n  \"name\": \"Item\", \r\n  \"type\": \"record\", \r\n  \"fields\": [\r\n    {\r\n      \"name\": \"itemId\", \r\n      \"type\": \"com.example.common.ItemId\" \r\n    } \r\n  ] \r\n}",
  "references": [
    {
      "groupId": "my-group",
      "artifactId": "ItemId",
      "name": "com.example.common.ItemId",
      "version": "1"
    }
  ]
}'
```

- アーティファクト参照の場合には、**application/json** コンテンツタイプを拡張する **application/create.extended+json** のカスタムコンテンツタイプを指定する必要があります。
4. アーティファクトが参照を使用して作成されたことを確認するために、予期される JSON 本文が応答に含まれていることを確認します。以下に例を示します。

```
{ "name": "Item", "createdBy": "", "createdOn": "2022-04-14T11:52:15+0000", "modifiedBy": "", "modifiedOn": "2022-04-14T11:52:15+0000", "id": "Item", "version": "1", "type": "AVRO", "globalId": 2, "state": "ENABLED", "groupId": "my-group", "contentId": 2, "references": [1] }
```

5. 参照を含むアーティファクトのグローバル ID を指定して、Service Registry からアーティファクト参照を取得します。この例では、指定されたグローバル ID は **2** です。

```
$ curl MY-REGISTRY-URL/apis/registry/v2/ids/globalIds/2/references
```

6. 応答に、このアーティファクト参照に必要とされる JSON 本文が含まれていることを確認してください。以下に例を示します。

```
[{"groupId":"my-group","artifactId":"ItemId","version":"1","name":"com.example.common.ItemId"}]
```

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。

4.4. SERVICE 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とインポート

管理者は、Service Registry REST API を使用して、ある Service Registry インスタンスからデータをエクスポートし、それを別の Service Registry インスタンスにインポートできるため、異なるインスタンス間でデータを移行できます。

本セクションでは、コアレジストリー v2 REST API を使用して、既存のレジストリーデータのある 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。Service Registry インスタンスに含まれるすべてのアーティファクトデータは、**.zip** ファイルにエクスポートされます。



注記

別の Service Registry インスタンスからエクスポートされた Service Registry データのみをインポートできます。

前提条件

- Service Registry が環境にインストールされ、実行している
- Service Registry インスタンスが作成されている。
 - データのエクスポート元のソースインスタンスには、少なくとも1つのスキーマまたは API アーティファクトが含まれている
 - データをインポートするターゲットインスタンスは、一意の ID を保持するために空である

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 MY-REGISTRY-URL/apis/registry/v2/admin/export \
-H "Authorization: Bearer $ACCESS_TOKEN" \
--output my-registry-data.zip
```

MY-REGISTRY-URL 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 "MY-REGISTRY-URL/apis/registry/v2/admin/import" \  
-H "Content-Type: application/zip" -H "Authorization: Bearer $ACCESS_TOKEN" \  
--data-binary @my-registry-data.zip
```

MY-REGISTRY-URL 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」
- 「Service Registry Maven プラグインを使用したアーティファクト参照の追加」
- 「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テスト」

前提条件

- [1章Service Registry の概要](#) を参照
- 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。

前提条件

- Service Registry が環境にインストールされ、実行している

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> ❶
      </goals>
      <configuration>
        <registryUrl>MY-REGISTRY-URL/apis/registry/v2</registryUrl> ❷
        <authServerUrl>MY-AUTH-SERVER</authServerUrl>
        <clientId>MY-CLIENT-ID</clientId>
        <clientSecret>MY-CLIENT-SECRET</clientSecret> ❸
        <artifacts>
          <artifact>
```



```

    <groupId>TestGroup</groupId> 4
    <artifactId>FullNameRecord</artifactId>
    <file>${project.basedir}/src/main/resources/schemas/record.avsc</file>
    <ifExists>FAIL</ifExists>
  </artifact>
  <artifact>
    <groupId>TestGroup</groupId>
    <artifactId>ExampleAPI</artifactId> 5
    <type>GRAPHQL</type>
    <file>${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. 認証が必要な場合は、認証サーバーおよびクライアントの認証情報を指定できます。
4. 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 ID を使用しない場合は、**default** のグループを指定できます。
5. 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登録できます。

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。

前提条件

- Service Registry が環境にインストールされ、実行している

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

```

```

<goals>
  <goal>download</goal> ❶
</goals>
<configuration>
  <registryUrl>MY-REGISTRY-URL/apis/registry/v2</registryUrl> ❷
  <authServerUrl>MY-AUTH-SERVER</authServerUrl>
  <clientId>MY-CLIENT-ID</clientId>
  <clientSecret>MY-CLIENT-SECRET</clientSecret> ❸
  <artifacts>
    <artifact>
      <groupId>TestGroup</groupId> ❹
      <artifactId>FullNameRecord</artifactId> ❺
      <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Service Registry URL を指定します。
3. 認証が必要な場合は、認証サーバーおよびクライアントの認証情報を指定できます。
4. 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。
5. 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。

5.3. SERVICE REGISTRY MAVEN プラグインを使用したアーティファクト参照の追加

Apache Avro、Protobuf、JSON スキーマなどの Service Registry アーティファクトタイプには、あるアーティファクトファイルから別のアーティファクトファイルへの **アーティファクト参照** を含めることができます。再利用可能なスキーマと API アーティファクトを定義し、アーティファクト参照で複数の場所から参照して、効率を高めることができます。

このセクションでは、Service Registry Maven プラグインを使用して、Service Registry に格納されている単純な Avro スキーマアーティファクトへのアーティファクト参照を登録する簡単な例を示します。この例では、次の **Exchange** スキーマアーティファクトが Service Registry にすでに作成されていることを前提としています。

交換スキーマ

■

```
{
  "namespace": "com.kubetrade.schema.common",
  "type": "enum",
  "name": "Exchange",
  "symbols": ["GEMINI"]
}
```

次に、この例では、ネストされた **Exchange** スキーマアーティファクトへの参照を含む **TradeKey** スキーマアーティファクトを作成します。

ネストされた Exchange スキーマを持つ TradeKey スキーマ

```
{
  "namespace": "com.kubetrade.schema.trade",
  "type": "record",
  "name": "TradeKey",
  "fields": [
    {
      "name": "exchange",
      "type": "com.kubetrade.schema.common.Exchange"
    },
    {
      "name": "key",
      "type": "string"
    }
  ]
}
```

前提条件

- Service Registry が環境にインストールされ、実行している
- **Exchange** スキーマアーティファクトは Service Registry にすでに作成されている

手順

apicurio-registry-maven-plugin を使用して **TradeKey** スキーマを登録するように Maven **pom.xml** ファイルを更新します。これには、次のように **Exchange** スキーマへのネストされた参照が含まれます。

```
<plugin>
  <groupId>io.apicurio</groupId>
  <artifactId>apicurio-registry-maven-plugin</artifactId>
  <version>${apicurio-registry.version}</version>
  <executions>
    <execution>
      <id>register-artifact</id>
      <goals>
        <goal>register</goal> ❶
      </goals>
      <configuration>
        <registryUrl>MY-REGISTRY-URL/apis/registry/v2</registryUrl> ❷
        <authServerUrl>MY-AUTH-SERVER</authServerUrl>
        <clientId>MY-CLIENT-ID</clientId>
        <clientSecret>MY-CLIENT-SECRET</clientSecret> ❸
      </configuration>
    </execution>
  </executions>
</plugin>
```

```

<artifacts>
  <artifact>
    <groupId>test-group</groupId> 4
    <artifactId>TradeKey</artifactId>
    <version>2.0</version>
    <type>AVRO</type>
    <file>
      ${project.basedir}/src/main/resources/schemas/TradeKey.avsc
    </file>
    <ifExists>RETURN_OR_UPDATE</ifExists>
    <canonicalize>true</canonicalize>
    <references>
      <reference> 5
        <name>com.kubetrade.schema.common.Exchange</name>
        <groupId>test-group</groupId>
        <artifactId>Exchange</artifactId>
        <version>2.0</version>
        <type>AVRO</type>
        <file>
          ${project.basedir}/src/main/resources/schemas/Exchange.avsc
        </file>
        <ifExists>RETURN_OR_UPDATE</ifExists>
        <canonicalize>true</canonicalize>
      </reference>
    </references>
  </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. 認証が必要な場合は、認証サーバーおよびクライアントの認証情報を指定できます。
4. 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 ID を使用しない場合は、**default** のグループを指定できます。
5. グループ ID、アーティファクト ID、バージョン、タイプ、および場所を使用して、Service Registry アーティファクト参照を指定します。この方法で、複数のアーティファクト参照を登録できます。

5.4. 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

Maven プラグインを使用してアーティファクトをテストする場合には、アーティファクトがテストに合格しても、Service Registry にコンテンツが追加されません。

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

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>${apicurio.version}</version>
  <executions>
    <execution>
      <phase>generate-sources</phase>
      <goals>
        <goal>test-update</goal> ❶
      </goals>
      <configuration>
        <registryUrl>MY-REGISTRY-URL/apis/registry/v2</registryUrl> ❷
        <authServerUrl>MY-AUTH-SERVER</authServerUrl>
        <clientId>MY-CLIENT-ID</clientId>
        <clientSecret>MY-CLIENT-SECRET</clientSecret> ❸
        <artifacts>
          <artifact>
            <groupId>TestGroup</groupId> ❹
            <artifactId>FullNameRecord</artifactId>
            <file>${project.basedir}/src/main/resources/schemas/record.avsc</file> ❺
          </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Service Registry URL を指定します。
3. 認証が必要な場合は、認証サーバーおよびクライアントの認証情報を指定できます。
4. 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。

5. 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。

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 Java 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 Java 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 JAVA 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

- Service Registry が環境にインストールされ、実行している

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";
RegistryClient client = RegistryClientFactory.create(registryUrl);
}
}
```

- <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/registry/v2` を追加します。
- 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。

クライアントが作成されると、Service Registry REST API で使用可能なすべての操作をクライアントで使用できます。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ルライザー/デシリアルライザー (SerDes) を使用する方法は、[7章Java クライアントでシリアルライザー/デシリアルライザーを使用した Kafka メッセージの検証](#) 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。たとえば、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。リクエストを認証するための認証サーバーも提供する必要があります。	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**

関連情報

- 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) の認証を設定する方法の詳細は、[7章 Java クライアントでシリアライザー/デシリアライザーを使用した Kafka メッセージの検証](#) を参照してください。

第7章 JAVA クライアントでシリアライザー/デシリアライザーを使用した KAFKA メッセージの検証

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 SerDes を使用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」](#)
- [「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 [OpenShift での AMQ Streams のデプロイとアップグレード](#) を参照してください。

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 (SerDes) 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 SerDes 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的 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的 ID です。

コンテンツ ID はバージョンを一意的に識別しませんが、バージョンコンテンツのみを一意的に識別します。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 Schema Registry 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、**ArtifactReferenceResolverStrategy** 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 SerDes 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。

アーティファクトリーゾルバーストラテジー

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.ArtifactReferenceResolverStrategy** 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。

アーティファクトへの参照を返すストラテジー

Service Registry は、**ArtifactReferenceResolverStrategy** の実装に基づいてアーティファクトへの参照を返すために、次の戦略を提供します。

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、グローバルで一意的識別子が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.resolver.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を指定することをお勧めします。

レジストリー検索オプションの設定

DefaultSchemaResolver を使用する場合、アプリケーションのプロパティ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 検索設定オプション

プロパティ	タイプ	説明	デフォルト
-------	-----	----	-------

プロパティ	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。	None
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。	なし

7.3. 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" \ ❶
--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 ❷
```

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>${apicurio.version}</version>
<executions>
  <execution>
    <phase>generate-sources</phase>
    <goals>
      <goal>register</goal> ❶
    </goals>
    <configuration>
      <registryUrl>http://REGISTRY-URL/apis/registry/v2</registryUrl> ❷
      <artifacts>
        <artifact>
          <groupId>TestGroup</groupId> ❸
          <artifactId>FullNameRecord</artifactId>
          <file>${project.basedir}/src/main/resources/schemas/record.avsc</file>
          <ifExists>FAIL</ifExists>
        </artifact>
        <artifact>
          <groupId>TestGroup</groupId>
          <artifactId>ExampleAPI</artifactId> ❹
          <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"); ❶
try (RegistryService service = RegistryClient.create(registryUrl_node1)) {
    String artifactId = ApplicationImpl.INPUT_TOPIC + "-value";
    try {
        service.getArtifactMetaData(artifactId); ❷
    } catch (WebApplicationException e) {
        CompletionStage <ArtifactMetaData> csa = service.createArtifact(
            "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。

7.4. 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()); ❶
props.putIfAbsent(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
    AvroKafkaDeserializer.class.getName()); ❷

```

- 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。

7.5. 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.BootstrapServersConfig, "my-cluster-kafka-
bootstrap:9092");
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    AvroKafkaSerializer.class.getName()); ❶
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    AvroKafkaSerializer.class.getName()); ❷
props.put(SerdeConfig.FIND_LATEST_ARTIFACT, Boolean.TRUE); ❸

```

- 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。

7.6. 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>(); ❶

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

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); ❸
```

- ❶. Service Registry によって提供される Avro シリアライザー。
 - ❷. Service Registry によって提供される Avro デシリアライザー。
 - ❸. 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 JAVA クライアントでの KAFKA シリアライザー/デシリアライザーの設定

本章では、プロデューサーおよびコンシューマー Java クライアントアプリケーションで Kafka SerDes を設定する方法について詳しく説明します。

- 「クライアントアプリケーションの Service Registry シリアライザー/デシリアライザーの設定」
- 「Service Registry シリアライザー/デシリアライザー設定プロパティ」
- 「さまざまなクライアントシリアライザー/デシリアライザータイプの設定方法」
- 「Service Registry を使用した Avro SerDes の設定」
- 「Service Registry を使用した JSON スキーマ SerDes の設定」
- 「Service Registry を使用した Protobuf SerDes の設定」

前提条件

- 7章 [Java クライアントでシリアライザー/デシリアライザーを使用した Kafka メッセージの検証](#) を読む。

8.1. クライアントアプリケーションの 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を設定できます。

以下のセクションでは、一般的に使用される 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"; ❶
    public static final String ID_HANDLER = "apicurio.registry.id-handler"; ❷
    public static final String ENABLE_CONFLUENT_ID_HANDLER = "apicurio.registry.as-confluent";
    ❸
```

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

- 設定オプションの詳細は、[「Service Registry シリアライザー/デシリアライザー設定プロパティ」](#) 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"; ❶
    public static final String SCHEMA_RESOLVER = "apicurio.registry.schema-resolver"; ❷
    ...
}
```

1. アーティファクトリーゾルバーストラテジーを実装し、Kafka SerDe とアーティファクト ID の間をマッピングする Java クラス。デフォルトはトピック ID ストラテジーです。これはシリアライザークラスによってのみ使用されます。
2. スキーマリゾルバーを実装する Java クラス。デフォルトは **DefaultSchemaResolver** です。これは、シリアライザーおよびデシリアライザークラスによって使用されます。

その他のリソース

- 検索ストラテジーの詳細は、[7章 Java クライアントでシリアライザー/デシリアライザーを使用した Kafka メッセージの検証](#) を参照してください。
- 設定オプションの詳細は、[「Service Registry シリアライザー/デシリアライザー設定プロパティ」](#) 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"; ❶
    public static final String REGISTRY_CONVERTER_DESERIALIZER_PARAM =
"apicurio.registry.converter.deserializer"; ❷
}
```

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 SerDes の設定」](#)
- [「Service Registry を使用した JSON スキーマ SerDes の設定」](#)
- [「Service Registry を使用した Protobuf SerDes の設定」](#)

8.2. SERVICE REGISTRY シリアライザー/デシリアライザー設定プロパティ

本セクションでは、Service Registry Kafka シリアライザー/デシリアライザー (SerDes) の Java 設定プロパティに関する参照情報を提供します。

SchemaResolver インターフェイス

Service Registry SerDes は、レジストリーへのアクセスを抽象化し、サポートされるすべての形式の SerDes クラスに同じ検索ロジックを適用する **SchemaResolver** インターフェイスをベースとしています。

表8.1 SchemaResolver インターフェイスの設定プロパティ

Constant	プロパティ	説明	型	デフォルト
SCHEMA_RESOLVER	apicurio.registry.schema-resolver	シリアライザーおよびデシリアライザーによって使用されます。 SchemaResolver を実装する完全修飾 Java クラス名。	String	io.apicurio.registry.resolver.DefaultSchemaResolver



注記

DefaultSchemaResolver が推奨され、ほとんどのユースケースに役立つ機能を提供します。一部の高度なユースケースでは、**SchemaResolver** のカスタム実装を使用する場合があります。

DefaultSchemaResolver クラス

DefaultSchemaResolver を使用して、次のような機能を設定できます。

- レジストリー API へのアクセス
- レジストリーでアーティファクトを検索する方法
- Kafka との間でアーティファクト情報を読み書きする方法
- デシリアライザーのフォールバックオプション

レジストリー API アクセスオプションの設定

DefaultSchemaResolver は、コアレジストリー API へのアクセスを設定するために次のプロパティを提供します。

表8.2 レジストリー API にアクセスするための設定プロパティ

Constant	プロパティ	説明	型	デフォルト
----------	-------	----	---	-------

Constant	プロパティ	説明	型	デフォルト
REGISTRY_URL	apicurio.registry.url	シリアルライザーおよびデシリアルライザーによって使用されます。レジストリー API にアクセスするための URL。	String	None
AUTH_SERVICE_URL	apicurio.auth.service.url	シリアルライザーおよびデシリアルライザーによって使用されます。認証サービスの URL。OAuth クライアントクレデンシャルフローを使用してセキュアなレジストリーにアクセスする際に必須です。	String	None
AUTH_REALM	apicurio.auth.realm	シリアルライザーおよびデシリアルライザーによって使用されます。認証サービスにアクセスするためのレルム。OAuth クライアントクレデンシャルフローを使用してセキュアなレジストリーにアクセスする際に必須です。	String	None
AUTH_CLIENT_ID	apicurio.auth.client.id	シリアルライザーおよびデシリアルライザーによって使用されます。認証サービスにアクセスするためのクライアント ID。OAuth クライアントクレデンシャルフローを使用してセキュアなレジストリーにアクセスする際に必須です。	String	None
AUTH_CLIENT_SECRET	apicurio.auth.client.secret	シリアルライザーおよびデシリアルライザーによって使用されます。認証サービスにアクセスするためのクライアントシークレット。OAuth クライアントクレデンシャルフローを使用してセキュアなレジストリーにアクセスする際に必須です。	String	None

Constant	プロパティ	説明	型	デフォルト
AUTH_USERNAME	apicurio.auth.username	シリアライザーおよびデシリアライザーによって使用されます。レジストリーにアクセスするためのユーザー名HTTP Basic 認証を使用してセキュアなレジストリーにアクセスする際に必須です。	String	None
AUTH_PASSWORD	apicurio.auth.password	シリアライザーおよびデシリアライザーによって使用されます。レジストリーにアクセスするためのパスワードHTTP Basic 認証を使用してセキュアなレジストリーにアクセスする際に必須です。	String	None

レジストリー検索オプションの設定

DefaultSchemaResolver は、次のプロパティを使用して、ServiceRegistry でアーティファクトを検索する方法を設定します。

表8.3 レジストリーアーティファクト検索の設定プロパティ

Constant	プロパティ	説明	型	デフォルト
ARTIFACT_RESOLVER_STRATEGY	apicurio.registry.artifact-resolver-strategy	シリアライザーによってのみ使用されます。 ArtifactReferenceResolverStrategy を実装し、各 Kafka メッセージを ArtifactReference (groupid, artifactId、および version) にマップする完全修飾 Java クラス名。たとえば、デフォルトのストラテジーはトピック名をスキーマ artifactId として使用します。	String	io.apicurio.registry.serdes.strategy.TopicIdStrategy

Constant	プロパティ	説明	型	デフォルト
EXPLICIT_ARTIFACT_GROUP_ID	apicurio.registry.artifact.group-id	シリアライザーによってのみ使用されます。アーティファクトのクエリーまたは作成に使用される groupId を設定します。 ArtifactResolverStrategy によって返される groupId をオーバーライドします。	String	None
EXPLICIT_ARTIFACT_ID	apicurio.registry.artifact.artifact-id	シリアライザーによってのみ使用されます。アーティファクトのクエリーまたは作成に使用される artifactId を設定します。 ArtifactResolverStrategy によって返される artifactId をオーバーライドします。	String	None
EXPLICIT_ARTIFACT_VERSION	apicurio.registry.artifact.version	シリアライザーによってのみ使用されます。アーティファクトの問い合わせまたは作成に使用されるアーティファクトバージョンを設定します。 ArtifactResolverStrategy によって返されるバージョンをオーバーライドします。	String	None
FIND_LATEST_ARTIFACT	apicurio.registry.find-latest	シリアライザーによってのみ使用されます。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。	boolean	false

Constant	プロパティ	説明	型	デフォルト
AUTO_REGISTER_ARTIFACT	apicurio.registry.automato-register	シリアライザーによってのみ使用されます。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の機能をサポートしません。	boolean	false
AUTO_REGISTER_ARTIFACT_IF_EXISTS	apicurio.registry.automato-register.if-exists	シリアライザーによってのみ使用されます。アーティファクトがすでに存在するためにアーティファクトの作成で競合が発生した場合のクライアントの動作を設定します。使用可能な値は、 FAIL 、 UPDATE 、 RETURN 、または RETURN_OR_UPDATE です。	String	RETURN_OR_UPDATE
CHECK_PERIOD_MS	apicurio.registry.check-period-ms	シリアライザーおよびデシリアライザーによって使用されます。自動エビクションの前にアーティファクトをキャッシュする期間を指定します。設定されていない場合、アーティファクトは毎回フェッチされます。	String	None

Constant	プロパティ	説明	型	デフォルト
USE_ID	apicurio.registry.use-id	シリアライザーおよびデシリアライザーによって使用されます。指定された IdOption をアーティファクトの識別子として使用するよう設定します。オプションは globalId と contentId です。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。	String	globalId

Kafka でレジストリーアーティファクトを読み書きするための設定

DefaultSchemaResolver は、次のプロパティを使用して、アーティファクト情報の Kafka への書き込みおよび Kafka からの読み取り方法を設定します。

表8.4 Kafka でアーティファクト情報を読み書きするための設定プロパティ

Constant	プロパティ	説明	型	デフォルト
ENABLE_HEADERS	apicurio.registry.headers.enabled	シリアライザーおよびデシリアライザーによって使用されます。メッセージペイロードではなく、Kafka メッセージヘッダーに対してアーティファクト識別子を読み書きするよう設定します。	boolean	true
HEADERS_HANDLER	apicurio.registry.headers.handler	シリアライザーおよびデシリアライザーによって使用されます。 HeadersHandler を実装し、Kafka メッセージヘッダーに対してアーティファクト識別子を読み書きする完全修飾 Java クラス名。	String	io.apicurio.registry.serde.headers.DefaultHeadersHandler

Constant	プロパティ	説明	型	デフォルト
ID_HANDLER	apicurio.registry.id-handler	シリアライザーおよびデシリアライザーによって使用されます。 IdHandler を実装し、メッセージペイロードに対してアーティファクト識別子を読み書きするクラスの完全修飾 Java クラス名。 apicurio.registry.headers.enabled が false に設定されている場合にのみ使用されます。	String	io.apicurio.registry.serde.DefaultIdHandler
ENABLE_CONFLUENT_ID_HANDLER	apicurio.registry.as-confluent	シリアライザーおよびデシリアライザーによって使用されます。 IdHandler のレガシー Confluent-compatible 実装を有効にするためのショートカット。 apicurio.registry.headers.enabled が false に設定されている場合にのみ使用されます。	boolean	true

デシリアライザーのフォールバックオプションの設定

DefaultSchemaResolver は、次のプロパティを使用して、すべてのデシリアライザーのフォールバックプロバイダーを設定します。

表8.5 デシリアライザーのフォールバックプロバイダーの設定プロパティ

Constant	プロパティ	説明	型	デフォルト
----------	-------	----	---	-------

Constant	プロパティ	説明	型	デフォルト
FALLBACK_ARTIFACT_PROVIDER	apicurio.registry.fallback.provider	デシリアライザーによってのみ使用されます。デシリアライズに使用されるアーティファクトを解決するための FallbackArtifactProvider のカスタム実装を設定します。 FallbackArtifactProvider は、検索に失敗した場合にレジストリーから取得するフォールバックアーティファクトを設定します。	String	io.apicurio.registry.serde.fallback.DefaultFallbackArtifactProvider

DefaultFallbackArtifactProvider は、次のプロパティを使用して、デシリアライザーのフォールバックオプションを設定します。

表8.6 デシリアライザーのフォールバックオプションの設定プロパティ

Constant	プロパティ	説明	型	デフォルト
FALLBACK_ARTIFACT_ID	apicurio.registry.fallback.artifact-id	デシリアライザーによってのみ使用されます。デシリアライズに使用されるアーティファクトを解決するためのフォールバックとして使用される artifactId を設定します。	String	None
FALLBACK_ARTIFACT_GROUP_ID	apicurio.registry.fallback.group-id	デシリアライザーによってのみ使用されます。デシリアライズに使用されるグループのフォールバックとして使用される groupId を設定します。	String	None
FALLBACK_ARTIFACT_VERSION	apicurio.registry.fallback.version	デシリアライザーによってのみ使用されます。デシリアライズに使用されるアーティファクトを解決するために使用されるバージョンを設定します。	String	None

その他のリソース

- 詳細は、[SerdeConfig Java class](#) を参照してください。
- アプリケーションプロパティを Java システムプロパティ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。

8.3. さまざまなクライアントシリアライザー/デシリアライザータイプの設定方法

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。

8.3.1. Service Registry を使用した Avro SerDes の設定

このトピックでは、Apache Avro の Kafka クライアントシリアライザーおよびデシリアライザー (SerDes) クラスを使用する方法について説明します。

Service Registry は、Avro 用に次の Kafka クライアント SerDes 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ゾルバストラテジー
- 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。

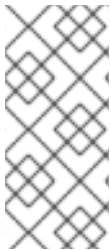
apicurio.registry.avro.encoding プロパティを **JSON** または **BINARY** の値で設定することにより、Avro エンコーディングを設定できます。デフォルトは **BINARY** です。

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ったら、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。

Avro SerDes とアーティファクトの参照

レコードがネスト化された Avro メッセージとスキーマを使用する場合に、ネストされたレコードごとに新しいアーティファクトが登録されます。たとえば、次の **TradeKey** スキーマには、ネストされた **Exchange** スキーマが含まれています。

ネストされた Exchange スキーマを持つ TradeKey スキーマ

```
{
  "namespace": "com.kubetrade.schema.trade",
  "type": "record",
  "name": "TradeKey",
  "fields": [
    {
      "name": "exchange",
      "type": "com.kubetrade.schema.common.Exchange"
    },
    {
      "name": "key",
      "type": "string"
    }
  ]
}
```


交換スキーマ

```
{
  "namespace": "com.kubetrade.schema.common",
  "type": "enum",
  "name": "Exchange",
  "symbols": ["GEMINI"]
}
```

これらのスキーマを Avro SerDes で使用すると、Service Registry に、**TradeKey** スキーマ用と、**Exchange** スキーマ用の 2 つのアーティファクトが作成されます。**TradeKey** スキーマを使用するメッセージがシリアライズまたはデシリアライズされるたびに、両方のスキーマが取得されるため、定義を異なるファイルに分割できます。

関連情報

- Avro 設定の詳細は、[AvroKafkaSerdeConfig Java クラス](#) を参照してください。
- Java のサンプルアプリケーションについては、以下を参照してください。
 - [簡単な Avro の例](#)
 - [参照付き SerDes の例](#)

8.3.2. Service Registry を使用した JSON スキーマ SerDes の設定

このトピックでは、JSON スキーマの Kafka クライアントシリアライザーおよびデシリアライザー (SerDes) クラスを使用する方法について説明します。

Service Registry は、JSON スキーマ用に次の Kafka クライアント SerDes 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。

JSON スキーマの SerDes とアーティファクトの参照

JSON スキーマ SerDes はメッセージペイロードからスキーマを検出できないため、事前にスキーマアーティファクトを登録する必要があります、これはアーティファクト参照にも適用されます。

スキーマの内容に応じて、**\$ref** の値は URL に置き換えます。SerDes はこの URL を使用して参照のスキーマを解決しようとし、主要なスキーマに対するデータの検証、ネスト化されたスキーマに対するネスト化された値の検証など、検証は通常どおりに行われます。Service Registry でアーティファクトを参照するためのサポートも実装されています。

たとえば、次の **city.json** スキーマは **city.json** スキーマを参照しています。

city.json スキーマを参照する citizen.json スキーマ

```
{
  "$id": "https://example.com/citizen.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "Citizen",
  "type": "object",
  "properties": {
    "firstName": {
      "type": "string",
      "description": "The citizen's first name."
    },
    "lastName": {
      "type": "string",
      "description": "The citizen's last name."
    },
    "age": {
      "description": "Age in years which must be equal to or greater than zero.",
      "type": "integer",
      "minimum": 0
    },
    "city": {
      "$ref": "city.json"
    }
  }
}
```

```

    }
  }
}

```

city.json スキーマ

```

{
  "$id": "https://example.com/city.schema.json",
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "City",
  "type": "object",
  "properties": {
    "name": {
      "type": "string",
      "description": "The city's name."
    },
    "zipCode": {
      "type": "integer",
      "description": "The zip code.",
      "minimum": 0
    }
  }
}

```

この例では、citizen に city が含まれます。Service Registry では、city アーティファクトへの参照を持つ citizen アーティファクトが、**city.json** という名前を使用して作成されます。SerDes では、citizen スキーマをフェッチすると、citizen スキーマから参照されるため、city スキーマもフェッチされます。データをシリアライズ/デシリアライズする場合、ネストされたスキーマを解決するために参照名が使用され、citizen スキーマとネストされた city スキーマに対する検証が可能になります。

関連情報

- 詳細は、[JsonSchemaKafkaDeserializerConfig Java クラス](#) を参照してください。
- Java のサンプルアプリケーションについては、以下を参照してください。
 - [簡単な JSON スキーマの例](#)
 - [参照付き SerDes の例](#)

8.3.3. Service Registry を使用した Protobuf SerDes の設定

このトピックでは、Google Protobuf の Kafka クライアントシリアライザーおよびデシリアライザー (SerDes) クラスを使用する方法について説明します。

Service Registry は、Protobuf 用に次の Kafka クライアント SerDes 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。

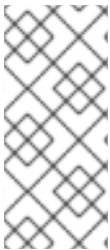
- [「クライアントアプリケーションの Service Registry シリアライザー/デシリアライザーの設定」](#)
- [「Service Registry を使用した Avro SerDes の設定」](#)

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ったら、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。

Protobuf SerDes とアーティファクト参照

import ステートメントを含む複雑な Protobuf メッセージが使用される場合に、インポートされた Protobuf メッセージは個別のアーティファクトとして Service Registry に保存されます。次に、Service Registry がメインスキーマを取得して Protobuf メッセージをチェックすると、完全なメッセージスキーマをチェックしてシリアル化できるように参照されているスキーマも取得されます。

たとえば、次の **table_info.proto** スキーマファイルには、インポートされた **mode.proto** スキーマファイルが含まれています。

インポートされた .mode.proto ファイルを含む table_info.proto ファイル

```
syntax = "proto3";
package sample;
option java_package = "io.api.sample";
option java_multiple_files = true;

import "sample/mode.proto";

message TableInfo {

  int32 winIndex = 1;
  Mode mode = 2;
  int32 min = 3;
  int32 max = 4;
  string id = 5;
  string dataAdapter = 6;
  string schema = 7;
  string selector = 8;
  string subscription_id = 9;
}
```

mode.proto ファイル

```
syntax = "proto3";
package sample;
option java_package = "io.api.sample";
option java_multiple_files = true;

enum Mode {

  MODE_UNKNOWN = 0;
  RAW = 1;
  MERGE = 2;
  DISTINCT = 3;
  COMMAND = 4;
}
```

この例では、**TableInfo** 用と **Mode** 用の 2 つの Protobuf アーティファクトが Service Registry に格納されます。ただし、**Mode** は **TableInfo** の一部であるため、SerDes 内のメッセージをチェックするために **TableInfo** がフェッチされるたびに、**Mode** も **TableInfo** で参照されるアーティファクトとして返されます。

関連情報

- Java のサンプルアプリケーションについては、以下を参照してください。
 - [Protobuf Bean と Protobuf Find Latest の例](#)
 - [参照付き SerDes の例](#)

第9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」](#)
- [「Service Registry コンテンツルールの優先順位」](#)

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。

9.1. SERVICE REGISTRY アーティファクトタイプ

Service Registry では、さまざまなスキーマおよび API アーティファクトタイプを保存して管理できます。

表9.1 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

9.2. SERVICE REGISTRY アーティファクトの状態

Service Registry の有効なアーティファクトの状態は、**ENABLED**、**DISABLED**、および **DEPRECATED** です。

表9.2 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。

9.3. 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と、生成された読み取り専用プロパティのセットで設定されます。

表9.3 Service Registry メタデータプロパティ

プロパティ	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

プロパティ	型	編集可能
labels	文字列の配列	true
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のセットを更新できます。
- **state** プロパティは、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。

9.4. SERVICE REGISTRY コンテンツルールタイプ

VALIDITY および **COMPATIBILITY** ルールタイプを指定して、Service Registry でのコンテンツの変化を制御できます。

表9.4 Service Registry コンテンツルールタイプ

型	説明
VALIDITY	<p>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。</p> <ul style="list-style-type: none"> • 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。 • NONE: すべての検証チェックが無効になります。

型	説明
COMPATIBILITY	<p>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。</p> <ul style="list-style-type: none"> ● 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。

9.5. 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9.5 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	完全	完全
JSON スキーマ	Full	Full

アーティファクトタイプ	検証ルール	互換性ルール
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	None

9.6. SERVICE REGISTRY コンテンツルールの優先順位

アーティファクトを追加して更新すると、Service Registry はルールを適用して、アーティファクトコンテンツの有効性と互換性をチェックします。次の表に示すように、設定されたアーティファクトルールは、同等の設定済みグローバルルールをオーバーライドします。

表9.6 Service Registry コンテンツルールの優先順位

アーティファクトルール	グローバルルール	このアーティファクトに適用されるルール	他のアーティファクトで利用できるグローバルルール
有効	有効	アーティファクト	はい
Disabled	有効	グローバル	はい
Disabled	Disabled	None	いいえ
有効、None に設定	有効	None	はい
Disabled	有効、None に設定	None	いいえ

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。