Red Hat Integration 2020.q1

データ仮想化の使用

テクノロジープレビュー - データ仮想化のユーザーガイド

Last Updated: 2023-02-28

Red Hat Integration 2020.q1 データ仮想化の使用

テクノロジープレビュー - データ仮想化のユーザーガイド

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

法律上の通知

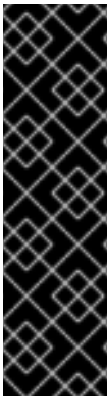
概要

アプリケーションが単一の仮想データモデルに接続できるように、複数のソースからデータを組み合わせる

目次

第1章 データ仮想化の概要

Data Virtualization はコンテナーネイティブサービスで、単一の統一された API 経由で、リレーショナルデータベースおよび noSQL データベース、ファイル、Web サービス、SaaS リポジトリーなど、複数の異なるデータソースとの統合を提供します。アプリケーションおよびユーザーは、標準のインターフェース（OData REST または JDBC/ODBC）で仮想データベースに接続し、単一のリレーショナルデータベースからデータが提供されるかのように設定済みのすべてのデータソースのデータと対話できます。



重要

Data Virtualization はテクノロジープレビューの機能です。テクノロジープレビュー機能は、Red Hat の本番環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあるため、Red Hat は本番環境での使用は推奨しません。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲についての詳細は、https://access.redhat.com/ja/support/offerings/techpreview/ を参照してください。

Red Hat データ仮想化技術は、オープンソースのデータ仮想化プロジェクトである Teiid をベースにしています。Teiid の詳細は、Teiid コミュニティーのドキュメント を参照してください。

第2章 仮想データベースの作成

仮想データベースを追加するには、以下のタスクを実行する必要があります。

1. Data Virtualization Operator をインストールします。
2. データベースを設計し、開発します。
3. データベースをデプロイするためのカスタムリソース(CR)ファイルを作成します。
4. CR で Data Virtualization Operator を実行して、仮想データベースを OpenShift にデプロイします。

以下の方法のいずれかを使用して仮想データベースを設計できます。

DDL ファイルからの仮想データベースの作成

YAML ファイルに DDL を含む仮想データベースの内容全体を定義します。詳細は、*3章カスタムリソース(CR)に DDL ステートメントを埋め込むことで仮想データベースの作成* を参照してください。

仮想データベースを Maven アーティファクトとして作成する

1つ以上の DDL ファイルから仮想データベースを作成し、デプロイメントの Maven アーティファクトを生成します。詳細は、*4章Maven アーティファクトとしての仮想データベースの作成* を参照してください。

仮想データベースをファット JAR として作成する

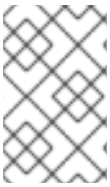
Teiid Spring Boot プラグインを使用して DDL ファイルで Maven ベースの Java プロジェクトを作成し、デプロイメント用の fat JAR を生成します。詳細は、*5章ファット JAR としての仮想データベースの作成* を参照してください。

各メソッドで、SQL データ定義言語(DDL)を使用して仮想データベースの構造を指定し、仮想データベースを読み書きするデータソースを設定します。

各メソッドを使用する利点と欠点があります。メソッドが作成するランタイム仮想化には同等の機能があります。プロジェクトの複雑性に基づいて方法を選択し、仮想化をスタンドアロンコンポーネントまたは OpenShift でのみテストできるようにするかどうかを選択します。

仮想データベースを定義した後に、Data Virtualization Operator を使用してカスタムリソース(CR)から仮想化をデプロイします。仮想データベースのデプロイに使用するカスタムリソースは、仮想データベースの設計に使用した方法によって異なります。詳細は、*6章Data Virtualization Operator の実行による仮想データベースのデプロイ* を参照してください。

データソースへの接続を設定したら、任意で Red Hat SSO への認証を設定して接続をセキュアにし、シングルサインオンを有効にできます。



注記

Fuse Online で仮想データベースを作成することもできます（テクノロジープレビュー）。Fuse Online で作成した仮想データベースは、限定された機能セットを提供します。

関連情報

- 「Data Virtualization Operator の OpenShift へのインストール」.
- *8章Fuse Online での仮想データベースの作成および操作*.

- *7章 データのセキュリティー保護.*

2.1. 互換性のあるデータソース

さまざまなデータソースから仮想データベースを作成できます。

データソースごとに、仮想データベースとデータソース間で渡すコマンドおよびデータを解釈できる **トランスレーター** の名前を指定する必要があります。

以下の表は、仮想データベースを作成するデータソースタイプと、各データソースの翻訳者名を示しています。

データソース		トランスレーターの名前
Amazon S3/ Ceph		amazon-s3
Google スプレッドシート		google-spreadsheet
Data Grid(Infinispan)		infinispan-hotrod
MongoDB		mongodb
リレーショナルデータベース		
	Amazon Athena	amazon-athena or jdbc-ansi
	Amazon Redshift	redshift
	Db2	db2
	Microsoft SQL Server (JDBC)	sqlserver
	MySQL	mysql
	Oracle	oracle
	PostgreSQL	postgresql
	SAP HANA(JDBC)	hana
OData		odata
OData4		odata4
OpenAPI		openapi
REST		ws

データソース	トランスレーターの名前
Salesforce	salesforce
sFTP	file
SOAP	SOAP または ws

2.2. 仮想化をデプロイするためのカスタムリソースの作成

Data Virtualization Operator を使用して仮想データベースを作成する前に、カスタムリソース(CR)ファイルにデータソースのプロパティーを指定する必要があります。

Data Virtualization Operator を実行すると、Data Virtualization アーティファクトをイメージに変換し、これを OpenShift にデプロイするために必要な CR から情報を読み取ります。

CR のプロパティーは、Operator がデータソースに接続するために必要な認証情報を保存する環境変数を指定します。CR に直接値を指定するか、または値を格納する OpenShift **シークレット** への参照を指定できます。シークレットの作成に関する詳細は、「シークレットの使用によるデータソースの認証情報の保存」 を参照してください。

注記

ピリオド文字(.)は、環境変数での使用は有効ではありません。変数名を CR に追加する場合は、アンダースコア(_)をセパレーターとして使用します。

CR に追加する情報は、仮想化用に作成したアーティファクトのタイプとアーティファクトの場所によって異なります。CR に設定情報を指定することもできます。

注記

OpenShift でデプロイされた仮想化の HTTP エンドポイントを作成する場合は、**spec/exposeVia3scale** プロパティーを CR に追加し、その値を **false** に設定します。値が **true** に設定されている場合、3scale はエンドポイントを管理し、HTTP エンドポイントは作成されません。

関連情報

- 「カスタムリソースの環境変数」
- 「DDL アーティファクトをデプロイするための CR の作成」
- 「Maven アーティファクトをデプロイするためのカスタムリソース(CR)の作成」
- 「ファット JAR をデプロイする CR の作成」

2.2.1. カスタムリソースの環境変数

カスタムリソースファイルに環境変数を設定して、仮想データベースをデータソースに接続できるようにします。

通常、仮想データベースを複数の OpenShift 環境（ステージングや実稼働環境など）にデプロイするた

め、環境ごとに異なるデータソースプロパティーを定義する必要がある場合があります。たとえば、ステージング環境でデータソースにアクセスするために指定する必要のあるログイン認証情報は、実稼働環境でデータソースにアクセスするために使用する認証情報とは異なる場合があります。各環境で一意の値を定義するには、環境変数を使用できます。

CR で定義する環境変数は、fat JAR の **application.properties** ファイルなど、els を設定する可能性のある静的プロパティーを置き換えます。プロパティーファイルおよび CR でプロパティーを定義する場合、CR ファイルの値が優先されます。

環境変数およびシークレットオブジェクトを組み合わせて、各環境の一意の情報を指定および保護することができます。環境変数の静的値を CR に直接指定する代わりに、各デプロイメント環境変数の値を各環境に固有のシークレットオブジェクトに保存できます。CR の各環境変数の値には、シークレットオブジェクトの名前とシークレットのトークン名を指定するキー参照のみが含まれます。トークンには実際の値が保存されます。ランタイム時に、環境変数はトークンから値を取得します。

シークレットを使用して環境変数の値を保存することにより、環境全体で単一バージョンの CR を使用できます。各環境にデプロイするシークレットオブジェクトは同じ名前を指定する必要がありますが、環境に固有のトークン値を割り当てるたびに、環境ごとに同じトークン値を割り当てます。

関連情報

- シークレットの使用についての詳細は、「シークレットの使用によるデータソースの認証情報の保存」 を参照してください。
- CR ファイルの追加に関する詳細は、「仮想化をデプロイするためのカスタムリソースの作成」を参照してください。

第3章 カスタムリソース(CR)に DDL ステートメントを埋め込むことで仮想データベースの作成

DDL ステートメントをカスタムリソースファイル内に直接追加することで、仮想データベースの構造を定義できます。デプロイメント時に、Operator は Source-to-Image(S2I)を仮想データベースアーティファクトで検出する依存関係に基づいて OpenShift 上でビルドします。ビルドの失敗を防ぐには、JDBC ドライバーの依存関係など、仮想データベースが必要とする依存関係がビルド時に見つけられるようにします。

CR で DDL を使用して仮想データベースを作成する利点

- シンプルかつ最小的。
- 仮想化のコードおよび設定には単一のファイルがあります。別個の CR ファイルを作成する必要はありません。
- 管理が容易になります。

CR で DDL を使用して仮想データベースを作成する欠点

- カスタムリソース(CR)ファイルに仮想データベースの DDL を埋め込むと、ファイルが大きくなります。
- DDL は CR YAML ファイルに組み込まれているため、DDL と設定の他の側面を個別にバージョンすることはできません。
- 複数の環境にデプロイする場合、プロパティーを設定マップまたはシークレットに保存し、カスタムリソースから独立させる必要があります。

前提条件

- Data Virtualization Operator がインストールされている OpenShift クラスターへの開発者または管理者アクセスがある。
- 互換性のあるデータソースがあり、OpenShift クラスターがアクセスできる。
- Data Virtualization Operator は、仮想データベースのビルド依存関係が含まれる Maven リポジトリーにアクセスできます。
- ログイン認証情報など、データソースの接続設定に関する情報が必要です。
- 作成する仮想データベースの DDL ファイルがあるか、またはデータベースを設計するために SQL コードを作成する方法を把握している。

手順

- YAML 形式で CR テキストファイルを作成し、これを .yaml または .yml 拡張子（**dv-customer.yaml**など）で保存します。
  以下の例は、postgreSQL データソースを使用する仮想データベースの CR に追加する要素を示しています。

  例： dv-customer.yaml

  ```
  apiVersion: teiid.io/v1alpha1
  ```

```
kind: VirtualDatabase
metadata:
  name: dv-customer
spec:
  replicas: 1 (1)
  env: (2)
  - name: SPRING_DATASOURCE_SAMPLEDB_USERNAME
    value: user
  - name: SPRING_DATASOURCE_SAMPLEDB_PASSWORD
    value: mypassword
  - name: SPRING_DATASOURCE_SAMPLEDB_DATABASENAME
    value: sampledb
  - name: SPRING_DATASOURCE_SAMPLEDB_JDBCURL
    value:
jdbc:postgresql://postgresql/$(SPRING_DATASOURCE_SAMPLEDB_DATABASENAME)
build:
    source:
      dependencies: (3)
      - org.postgresql:postgresql:42.1.4
      ddl: | (4)
        CREATE DATABASE customer OPTIONS (ANNOTATION 'Customer VDB');
        USE DATABASE customer;

        CREATE SERVER sampledb TYPE 'NONE' FOREIGN DATA WRAPPER postgresql;

        CREATE SCHEMA accounts SERVER sampledb;
        CREATE VIRTUAL SCHEMA portfolio;

        SET SCHEMA accounts;
        IMPORT FOREIGN SCHEMA public FROM SERVER sampledb INTO accounts
OPTIONS("importer.useFullSchemaName" 'false');

        SET SCHEMA portfolio;

        CREATE VIEW CustomerZip(id bigint PRIMARY KEY, name string, ssn string, zip
string) AS
            SELECT c.ID as id, c.NAME as name, c.SSN as ssn, a.ZIP as zip
            FROM accounts.CUSTOMER c LEFT OUTER JOIN accounts.ADDRESS a
            ON c.ID = a.CUSTOMER_ID;
      mavenRepositories: (5)
        central: https://repo.maven.apache.org/maven2
```

(1) デプロイするインスタンスの数を指定します。デフォルト設定は 1 です。

(2) この仮想化の設定プロパティを指定します。主にデータソースに接続する設定となります。この例のプロパティーは、PostgreSQL データベースへの接続に適用されます。サポートされるデータソースとそのプロパティーに関する情報は、「互換性のあるデータソース」 を参照してください。

(3) GAV 形式の Maven 依存関係 JAR ファイルの一覧(groupId:artifactid:version)を指定します。これらのファイルは、データソースの JDBC ドライバーファイルとカスタム依存関係を定義します。通常、Operator ビルドはパブリック Maven リポジトリーで利用可能なライブラリーを自動的に追加します。

(4)

DDL 形式で仮想データベースを定義します。DDL を使用して仮想データベースを定義する方法の詳細は、『Data Virtualization リファレンスガイド』の「 **DDL metadata for**

5 依存関係やその他の仮想データベースを含むプライベートまたはパブリック以外のリポジトリーの場所を指定します。複数のリポジトリーを指定できます。

YAML ファイルの作成後に、Data Virtualization Operator を実行して仮想データベースを OpenShift にデプロイできます。詳細は、*6章Data Virtualization Operator の実行による仮想データベースのデプロイ* を参照してください。

3.1. DDL アーティファクトをデプロイするための CR の作成

DDL を直接 CR に組み込み、仮想データベースを作成する場合、Data Virtualization Operator がデプロイメントに使用する CR がすでに存在します。DDL アーティファクトの CR の詳細は、*3章カスタムリソース(CR)に DDL ステートメントを埋め込むことで仮想データベースの作成* を参照してください。

CR で Data Virtualization Operator を実行し、仮想データベースを生成し、これを OpenShift にデプロイします。

関連情報

- *6章Data Virtualization Operator の実行による仮想データベースのデプロイ* .

第4章 MAVEN アーティファクトとしての仮想データベースの作成

Teiid Maven プラグインを使用して DDL ファイルを Maven アーティファクトに変換できます。DDL ファイルで仮想データベースの構造を定義し、ファイルを使用してアーティファクトを生成し、Maven リポジトリーにデプロイします。その後、Data Virtualization Operator はアーティファクトを Maven リポジトリーから OpenShift プロジェクトにデプロイできます。

これは、高レベルの柔軟性を提供する高度な手法で、複雑なプロジェクトに適しています。この方法では、1 つ以上の仮想データベースをインポートして、設計に組み込むマルチモジュールの Maven プロジェクトを作成できます。

pom.xml ファイルで Teiid プラグインの使用を指定します。**pom.xml** ファイルで、他の Maven 依存関係を定義することもできます。ビルドを実行すると、プラグインはファイルを読み取り、その内容を解決します。

Maven アーティファクトとして仮想データベースを作成する利点

- 仮想データベースや他の設定を表す DDL コード間の柔軟性の明確化。
- 複数の環境へのデプロイメントを容易にします。
- 仮想データベースレベルでのバージョン管理を提供します。
- 仮想データベースを、一貫した方法でプロジェクトおよびチーム間で共有できるようにします。
- 継続的インテグレーションおよび継続的デリバリー(CI/CD)のワークフローをサポートします。

仮想データベースを Maven アーティファクトとして作成するデメリット

- Maven の作業知識が必要です。

前提条件

- 互換性のあるデータソースがあり、OpenShift クラスターがアクセスできる。
- **pom.xml** ファイルを作成し、仮想データベースを構築するために必要な依存関係を指定する方法を把握している。
- ログイン認証情報など、データソースの接続設定に関する情報が必要です。
- Data Virtualization Operator は、仮想データベースのビルド依存関係が含まれる Maven リポジトリーにアクセスできます。
- Maven 3.2 以降がインストールされている。

手順

1. テキストエディターで POM ファイルを作成し、ビルドの依存関係を定義します。以下に例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.teiid</groupId>
  <artifactId>dv-customer</artifactId>
  <name>dv-customer</name>
  <description>Demo project to showcase maven based vdb</description>
  <packaging>vdb</packaging>
  <version>1.0</version>

  <build>
    <plugins>
      <plugin>
        <groupId>org.teiid</groupId>
        <artifactId>vdb-plugin</artifactId>
        <version>1.2.0</version>
        <extensions>true</extensions>
        <executions>
          <execution>
            <goals>
              <goal>vdb</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>
</project>
```

2. Maven プロジェクトを作成し、DDL ファイルから仮想データベース定義をインポートします。以下に例を示します。

```
vdb-project
├── pom.xml
└── src
    └── main
        └── vdb
            └── vdb.ddl
```

3. 仮想データベースの構造を指定する DDL ファイルを作成し、プロジェクトの **/src/main/vdb** ディレクトリーに **.ddl** エクステンションで保存します。たとえば、**vdb.ddl**です。
以下の例は、postgreSQL データソースを使用する仮想データベースの DDL ファイルのサンプルを示しています。

例： vdb.ddl

```
CREATE DATABASE customer OPTIONS (ANNOTATION 'Customer VDB');
USE DATABASE customer;

CREATE FOREIGN DATA WRAPPER postgresql;
CREATE SERVER sampledb TYPE 'NONE' FOREIGN DATA WRAPPER postgresql;

CREATE SCHEMA accounts SERVER sampledb;
CREATE VIRTUAL SCHEMA portfolio;
```

```
SET SCHEMA accounts;
IMPORT FOREIGN SCHEMA public FROM SERVER sampledb INTO accounts
OPTIONS("importer.useFullSchemaName" 'false');

SET SCHEMA portfolio;

CREATE VIEW CustomerZip(id bigint PRIMARY KEY, name string, ssn string, zip string) AS
  SELECT c.ID as id, c.NAME as name, c.SSN as ssn, a.ZIP as zip
  FROM accounts.CUSTOMER c LEFT OUTER JOIN accounts.ADDRESS a
  ON c.ID = a.CUSTOMER_ID;
```

DDL を使用して仮想データベースを定義する方法の詳細は、『Data Virtualization Reference』の「**DDL metadata for schema objects**」を参照してください。完全な DDL の定義については、本書では扱いません。

4. 仮想データベースアーティファクトをビルドします。Maven プロジェクトのルートフォルダーにターミナルウィンドウを開き、以下のコマンドを入力します。

```
mvn clean install
```

このコマンドは、ターゲットリポジトリーに **${project.name}-$2020.Q1.vdb** ファイルを生成します。

5. 以下のコマンドを入力して、アーティファクトをリモートリポジトリーにデプロイします。

```
mvn clean install deploy
```

Maven リポジトリーで仮想データベースアーティファクトが利用可能になった後に、YAML ベースのカスタムリソースを使用して仮想データベースを OpenShift にデプロイできます。YAML を使用して仮想データベース Maven アーティファクトをデプロイするためのカスタムリソースを作成する方法は、「Maven アーティファクトをデプロイするためのカスタムリソース(CR)の作成」 を参照してください。

Data Virtualization Operator を使用して仮想データベースをデプロイする方法は、*6章Data Virtualization Operator の実行による仮想データベースのデプロイ* を参照してください。

4.1. MAVEN アーティファクトをデプロイするためのカスタムリソース(CR)の作成

Maven アーティファクトとして作成する仮想化をデプロイする前に、Maven リポジトリーの場所を定義する CR を作成する必要があります。仮想化をデプロイする準備ができたら、この CR を Data Virtualization Operator に指定します。

前提条件

- *4章Maven アーティファクトとしての仮想データベースの作成* の説明に従って仮想化を作成している。
- Data Virtualization Operator がアクセスできる Maven リポジトリーに仮想化をデプロイしている。
- データソースにアクセスするためのログイン認証情報がある。
- YAML 形式のカスタムリソースファイルの作成を理解している。

手順

1. テキストエディターを開き、仮想化の名前を持つファイルを作成して、拡張子 **.yaml** で保存します（例： **dv-customer.yaml** ）。
2. 情報を追加して、カスタムリソースの種類、名前、およびソースを定義します。以下のアノテーションが付けられる例は、CR に含まれるコンテンツに関するガイダンスを示しています。

dv-customer.yaml

```
apiVersion: teiid.io/v1alpha1
kind: VirtualDatabase
metadata:
  name: dv-customer
spec:
  replicas: 1
  env:
  - name: SPRING_DATASOURCE_SAMPLEDB_USERNAME (1)
  value: user
  - name: SPRING_DATASOURCE_SAMPLEDB_PASSWORD
  value: mypassword
  - name: SPRING_DATASOURCE_SAMPLEDB_DATABASENAME
  value: sampledb
  - name: SPRING_DATASOURCE_SAMPLEDB_JDBCURL (2)
  value:
jdbc:postgresql://postgresql/$(SPRING_DATASOURCE_SAMPLEDB_DATABASENAME)
  resources:
    memory: 1024Mi
    cpu: 2.0
  build:
    source:
      maven: com.example:customer-vdb:1.0.0:vdb (3)
    mavenRepositories: (4)
      central: https://repo.maven.apache.org/maven2
```

(1) データソースにサインインするための認証情報を指定します。この例では、CR 内で定義される認証情報を示していますが、実稼働環境ではシークレットを使用して認証情報をプレーンテキストで公開するのではなく認証情報を指定します。シークレットに認証情報を追加する方法は、以下を参照してください。

(2) データソースに接続するための URL を指定します。

(3) groupId、artifactId、および version(GAV)コーディネートを指定して、仮想データベースの Maven の場所を指定します。

(4) プライベート Maven リポジトリーを使用している場合は、その URL を指定します。

CR YAML ファイルの作成後に、Data Virtualization Operator を実行して仮想データベースを OpenShift にデプロイできます。

CR で Data Virtualization Operator を実行し、仮想データベースを生成し、これを OpenShift にデプロイします。

関連情報

- 6章*Data Virtualization Operator の実行による仮想データベースのデプロイ*.

第5章 ファット JAR としての仮想データベースの作成

Teiid Springboot スターターを使用して、仮想化ファイルを Fat JAR として作成できます。JAR を Maven リポジトリーに公開し、YAML ベースのカスタムリソースを使用して仮想データベースを OpenShift にデプロイできます。Teiid Spring Boot スターターに関する詳細は、https://github.com/teiid/teiid-spring-boot を参照してください。

Spring Boot Maven プラグインは、単一の JAR ファイルにすべてのアプリケーションコードおよび依存関係が含まれる自己完結型 Uber JAR または fat JAR を作成します。

プロジェクトのリソースファイル（DDL ファイルや **application.properties**など）で仮想データベースを定義し、**pom.xml** ファイルで仮想データベースを Spring Boot Java 実行可能ファイルとしてビルドするために必要な依存関係を指定します。ビルドを実行すると、Maven は **pom.xml** ファイルを読み込み、そのコンテンツを解決して、外部リソースをビルドに組み込みます。

プロジェクトをビルドすると、仮想データベースを Spring Boot Java 実行可能ファイルとして作成します。作成された実行ファイルをローカルでテストできます。

ローカルテストが完了したら、JAR ファイルを Maven リポジトリーにデプロイできます。FAT JAR を Maven リポジトリーで利用可能になると、仮想データベースを OpenShift にデプロイするのと同様に YAML ベースのカスタムリソースを使用できます。

ファット JAR として仮想データベースを作成する利点

- 仮想データベースと設定を表す DDL コード間のクリーン分離を確立します。
- OpenShift 外の仮想化のローカルテストを提供します。当然ながら、OpenShift 環境に依存する他の機能、キャッシュ、認証、およびその他の機能はローカルでは機能しません。
- プロジェクトの一部としてユーザー定義の関数(UDF)、カスタム翻訳者などの拡張機能をサポートするため、自動的にランタイム仮想データベースに組み込まれます。
- 複数の環境へのデプロイメントに適しています。
- バージョン管理は、プロジェクト全体のレベルで実行されます。

ファット JAR として仮想データベースを作成する際の欠点

- Java、Maven、Teiid Spring Boot スターター、Spring、および Teiid に関する作業知識が必要です。

前提条件

- Java 開発、Maven、Teiid Spring Boot スターター、Spring、および Teiid に関する作業知識がある。
- Maven 3.2 以降がインストールされている。
- JDK 11（Java Platform、Standard Edition 11 Development Kit）以降がインストールされている。
- 互換性のあるデータソースがあり、OpenShift クラスターがアクセスできる。
- 仮想データベースを構築するために必要な依存関係を指定する **pom.xml** ファイルがある。

- データソースのドライバーが公開 Maven リポジトリーから利用できない場合は、ドライバーをダウンロードし、ローカルの Maven リポジトリーにデプロイしています。
- Data Virtualization Operator は、仮想データベースのビルド依存関係が含まれる Maven リポジトリーにアクセスできます。
- 作成する仮想データベースの DDL ファイルがあるか、SQL コードを作成し、DDL ファイルを作成する方法を把握している。

手順

1. 仮想データベースの以下のディレクトリー構造で Java Maven プロジェクトを作成します。

```
dv-customer-fatjar/
├── pom.xml
└── src
    └── main
        ├── java
        │   └── io
        │       └── integration
        │           ├── Application.java
        │           └── DataSources.java
        └── resources
            ├── application.properties
            └── vdb.ddl
```

1. **pom.xml** で、仮想データベースの構築に必要なリポジトリーの場所、ドライバー、およびユーザーの認証情報を定義します。
2. 仮想データベースプロジェクトのアプリケーションライブラリーで、Java アプリケーションファイル **Application.java**を作成します。
3. 同じディレクトリーに、**Datasourcess.java** クラスファイルを追加し、仮想データベースが接続する各データソースに bean メソッドを追加します。postgreSQL データベースと連携するように設計された **Datasources.java** ファイルの例は、「Datasources.java ファイルのサンプル」を参照してください。
4. **/src/main/resources** に、**application.properties** ファイルとデータソースの接続プロパティーを追加します。詳細は、「アプリケーションプロパティーの指定」 を参照してください。
5. **/resources/vdb.ddl** で DDL ステートメントを追加して、ビューを含む仮想データベースの構造を指定します。たとえば、**vdb.ddl**です。
   以下の例は、postgreSQL データソースを使用する仮想データベースの DDL ファイルのサンプルを示しています。

   例： vdb.ddl

```
CREATE DATABASE customer OPTIONS (ANNOTATION 'Customer VDB');
USE DATABASE customer;

CREATE FOREIGN DATA WRAPPER postgresql;
CREATE SERVER sampledb TYPE 'NONE' FOREIGN DATA WRAPPER postgresql;

CREATE SCHEMA accounts SERVER sampledb;
CREATE VIRTUAL SCHEMA portfolio;
```

```
SET SCHEMA accounts;
IMPORT FOREIGN SCHEMA public FROM SERVER sampledb INTO accounts
OPTIONS("importer.useFullSchemaName" 'false');

SET SCHEMA portfolio;

CREATE VIEW CustomerZip(id bigint PRIMARY KEY, name string, ssn string, zip string) AS
  SELECT c.ID as id, c.NAME as name, c.SSN as ssn, a.ZIP as zip
  FROM accounts.CUSTOMER c LEFT OUTER JOIN accounts.ADDRESS a
  ON c.ID = a.CUSTOMER_ID;
```

DDL を使用して仮想データベースを定義する方法の詳細は、『Data Virtualization リファレンスガイド』の「**DDL metadata for schema objects**」を参照してください。仮想データベースの完全な DDL を定義する方法については、本書の対象外となります。

6. 仮想データベースアーティファクトをビルドします。端末を開き、以下のコマンドを入力します。

```
mvn clean install
```

このコマンドは、ターゲットリポジトリーに **${project.name}-$2020.Q1.vdb** ファイルを生成します。

7. 以下のコマンドを入力して、アーティファクトをリモートリポジトリーにデプロイします。

```
mvn clean install deploy
```

Maven リポジトリーで仮想データベースアーティファクトが利用可能になった後に、YAML ベースのカスタムリソースを使用して仮想データベースを OpenShift にデプロイできます。

5.1. DATASOURCES.JAVA ファイルのサンプル

Datasources.java ファイルは、データソースへの接続を表すクラスを追加します。このファイルは、**ConfigurationProperties** 引数(**spring.datasource.sampledb**)にプレフィックスも確立します。この接頭辞は、**application.properties** ファイルに指定するデータソースプロパティーの名前で使用する必要があります。

Datasources.java で複数のデータソースを定義するには、それぞれが独自のプレフィックス指定を持つ複数のクラスを追加します。それぞれの場合も、対応するエントリーを DDL ファイルおよび CR ファイルの **データソース** プロパティーに追加する必要があります。

Java Bean を DDL ファイルに定義されたデータソースに関連付けるには、Bean 名は DDL ファイルの **SERVER** および **resource-name** プロパティーの名前と同じである必要があります。たとえば、以下のサンプルファイルは **sampledb** という名前の PostgreSQL データベースへの接続を確立します。これは、DDL ファイルでデータソースの **SERVER** オブジェクトおよびその **resource-name** 定義に割り当てられる名前です。

```
package com.example;

import javax.sql.DataSource;

import org.springframework.boot.jdbc.DataSourceBuilder;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
public class DataSources {

    @ConfigurationProperties(prefix = "spring.datasource.sampledb") (1)
    @Bean
    public DataSource sampledb() {  (2)
        return DataSourceBuilder.create().build();
    }

}
```

(1) プレフィックスは、**application.properties** ファイルで定義するプロパティーに割り当てる接頭辞と一致する必要があります。

(2) プレフィックス定義の **sampledb** と メソッド name は、仮想データベースの DDL ファイルで定義される **SERVER** および **resource-name** オブジェクトの名前と一致する必要があります。Spring Boot フレームワークは、**Datasources.java** ファイルのメソッド名と DDL ファイルのデータソースの名前を自動的に関連付けます。

5.2. アプリケーションプロパティーの指定

ファット JAR として仮想化を作成する場合は、**application.properties** ファイルを指定する必要があります。仮想データベースアプリケーションの静的プロパティーは、**/src/main/resource** ディレクトリーの **application.properties** ファイルで定義できます。静的プロパティーは、異なる環境全体で定数を維持する構成設定です。OpenShift に仮想データベースをデプロイした後に、仮想データベースを再ビルドおよび再デプロイしない限り、**application.properties** ファイルに加えた変更は有効ではありません。

application.properties ファイルで定義するデータソースプロパティーの前に、**Datasources.java** ファイルに指定された設定プロパティー文字列を追加する必要があります。プレフィックスは、プロパティーと Java クラスとの間の接続を確立します。

たとえば、データソース **.java ファイルの設定プロパティー接頭辞 spring.datasource. sampledb** を確立する場合、以下のプロパティー定義のように、**application.properties** ファイルで定義するプロパティー名の前にその文字列を追加する必要があります。

```
spring.datasource.sampledb.username=<username>
spring.datasource.sampledb.password=<password>
```

前提条件

- アプリケーション接頭辞を指定する Java クラスフォルダーに **Datasources.java** ファイルがある。

手順

1. **application.properties** ファイルを Java プロジェクトの **src/main/resources** フォルダーに追加します。
2. ファイルで、認証情報など、データソースへの接続に必要なプロパティーを定義します。

注記

application.properties ファイルに定義しないプロパティーは CR YAML ファイルに定義する必要があります。

注記

application.properties でプロパティーを定義し、CR で対応する環境変数を定義する場合、CR の値は **application.properties** ファイルに設定される値よりも優先されます。

以下に例を示します。

```
spring.datasource.sampledb.jdbc-url=jdbc:postgresql://localhost/sampledb (1) (2)
spring.datasource.sampledb.username=user (3)
spring.datasource.sampledb.password=user
spring.datasource.sampledb.driver-class-name=org.postgresql.Driver (4)
spring.datasource.sampledb.platform=sampledb (5)

# spring overrides
spring.teiid.model.package=io.integration
spring.main.allow-bean-definition-overriding=true

# open jdbc/odbc ports
teiid.jdbc-secure-enable=true
teiid.pg-secure-enable=true
teiid.jdbc-enable=true
teiid.pg-enable=true

# How to debug?
#logging.level.org.teiid=DEBUG (6)
```

(1) 仮想データベースがローカルの postgreSQL データベースにデータソースとして接続するために使用する JDBC URL。

(2) これらの各プロパティーで使用される接頭辞は、**Datasources.java** ファイルで定義される接頭辞と一致します。

(3) ここに記載のユーザー名およびパスワードの値は、プレーンテキストで表示されます。認証認証情報のセキュアなストレージを有効にするには、CR ファイルで環境変数を使用し、これらの値を定義するシークレットオブジェクトを参照します。

(4) データソースへの接続に必要なドライバー。ここで参照するドライバーは、**pom.xml** ファイルで依存関係として定義する必要があります。仮想データベースを fat JAR として作成するための **pom.xml** ファイルの例は、**teiid/dv-customer-fatjar** リポジトリーを参照してください。

(5) データソースの名前

(6) このステートメントのコメントを解除して、デバッグロギングを有効にします。

関連情報

- 「ファット JAR をデプロイする CR の作成」

5.3. ファット JAR をデプロイする CR の作成

teiid-springboot スターターから仮想データベースを開発したら、作成された JAR を Maven リポジトリーにデプロイします。次に、仮想データベースを OpenShift にデプロイする YAML カスタムリソースファイルを作成します。

fat JAR として作成された仮想データベースをデプロイするための CR ファイルは、「Maven アーティファクトをデプロイするためのカスタムリソース(CR)の作成」 で説明されているように、Maven アーティファクトとして作成される仮想データベースをデプロイするために使用する CR のようになります。Maven GAV コーディネートのみが異なります。この場合、CR は JAR ファイルの Maven コーディネートを提供します。

前提条件

- *5章 ファット JAR としての仮想データベースの作成* の説明に従って、仮想化を致命的 JAR として作成している。
- Data Virtualization Operator がアクセスできる Maven リポジトリーに仮想化をデプロイしている。
- データソースにアクセスするためのログイン認証情報がある。
- YAML 形式のカスタムリソースファイルの作成を理解している。

手順

1. .テキストエディターを開き、仮想化の名前を持つファイルを作成して、拡張子 **.yaml** で保存します（例： **dv-customer.yaml** ）。
2. 情報を追加して、カスタムリソースの種類、名前、およびソースを定義します。
以下の例は、fat JAR として作成される仮想データベースをデプロイするように設計された CR を示しています。

```
apiVersion: teiid.io/v1alpha1
kind: VirtualDatabase
metadata:
  name: dv-customer
spec:
  replicas: 1
  env:
  - name: SPRING_DATASOURCE_SAMPLEDB_USERNAME (1)
    value: user
  - name: SPRING_DATASOURCE_SAMPLEDB_PASSWORD
    value: mypassword
  - name: SPRING_DATASOURCE_SAMPLEDB_DATABASENAME
    value: sampledb
  - name: SPRING_DATASOURCE_SAMPLEDB_JDBCURL
    value:
jdbc:postgresql://postgresql/$(SPRING_DATASOURCE_SAMPLEDB_DATABASENAME)
  resources:
    memory: 1024Mi
    cpu: 2.0
  build:
    source:
      maven: org.teiid:dv-customer-fatjar:1.1 (2)
```

1 postgreSQL データソースの環境変数のサンプル。

- この例では、データソース認証情報の値はクリアテキストで定義されます。ただし、「仮想化をデプロイするためのカスタムリソースの作成」 で説明したように、CR に直接認証情報を指定することは安全ではありません。認証情報を保護するには、OpenShift シークレットから認証情報を参照します。詳細は、「シークレットの使用によるデータソースの認証情報の保存」 を参照してください。

注記

CR で定義する環境変数も **application.properties** ファイルのプロパティーとして定義される場合、CR の値は **application.properties** ファイルに設定される値よりも優先されます。

2 *5章 ファット JAR としての仮想データベースの作成* の Maven リポジトリーにデプロイしたファット JAR アーティファクトの Maven コーディネート。

CR YAML ファイルの作成後に、Data Virtualization Operator を実行して仮想データベースを OpenShift にデプロイできます。

関連情報

- *6章Data Virtualization Operator の実行による仮想データベースのデプロイ*.

第6章 DATA VIRTUALIZATION OPERATOR の実行による仮想データベースのデプロイ

data-virtualization Operator は、仮想データベースの設定やデプロイメントを自動化するのに役立ちます。

Data Virtualization Operator は仮想データベースカスタムリソース(CR)を処理し、仮想データベースオブジェクトを OpenShift にデプロイします。異なる CR で Operator を実行すると、範囲のデータソースから仮想データベースを作成できます。

注記

このテクノロジープレビューで OpenShift にデプロイする仮想データベースは、Fuse Online では利用できません。

6.1. DATA VIRTUALIZATION OPERATOR の OPENSHIFT へのインストール

Data Virtualization Operator をインストールして、仮想データベースイメージを YAML ベースのカスタムリソース(CR)から OpenShift にデプロイできます。

Data Virtualization Operator を OpenShift 3.11 以降にインストールできます。OpenShift 4.2 以降では、Operator は OperatorHub で利用できます。

Operator を OpenShift クラスターに追加したら、さまざまなデータソースから仮想データベースイメージをビルドし、デプロイできます。

前提条件

- OpenShift 3.11 または 4.2 以降のクラスターへの cluster-admin アクセスがある。
- **oc** コマンドラインツールを使用して OpenShift 3.11 クラスターと対話したり、OpenShift 4.2 以上の Web コンソールにアクセスしたりできます。
- 2020.Q1 リリースが Red Hat Integration である。
- OpenShift サーバーへの開発者アクセスがあり、OpenShift コンソールおよび CLI を使用できること。

手順

- 実行中の OpenShift のバージョンに応じて、以下のいずれかの方法を使用して Operator をインストールします。

OpenShift 3.11 へのインストール

1. ターミナルウインドウから、クラスター管理者として OpenShift クラスターにログインします。

```
oc login
```

2. 仮想データベースをデプロイするプロジェクトを作成または開きます。

3. 以下のコマンドを入力します。

```
export OP_ROOT=https://raw.githubusercontent.com/teiid/teiid-operator/7.6-0.0.x/deploy
oc create -f $OP_ROOT/crds/virtualdatabase.crd.yaml (1)
oc create -f $OP_ROOT/service_account.yaml
oc create -f $OP_ROOT/role.yaml
oc create -f $OP_ROOT/role_binding.yaml
oc create -f $OP_ROOT/operator.yaml
```

(1) クラスターに CRD を事前に作成した場合、コマンドはエラーを返し、CRD がすでに存在していることを報告します。メッセージは無視できます。

4. 以下のコマンドを入力して、Red Hat イメージレジストリーへのアクセスに使用できるプルシークレットを作成します。

```
oc create secret docker-registry dv-pull-secret /
--docker-server=registry.redhat.io /
--docker-username=$username / (1)
--docker-password=$password /
--docker-email=$email_address
oc secrets link builder dv-pull-secret
oc secrets link builder dv-pull-secret --for=pull
```

(1) Red Hat カスタマーポータルへのログインに使用するユーザー名とパスワードを置き換えます。

コマンドがエラーなしで完了する場合、Operator は現在の OpenShift プロジェクト内の OpenShift インスタンスにデプロイされます。

5. Data Virtualization Operator が Red Hat レジストリーからイメージを取得できるようにするには、ステップ 4 で作成したシークレットを Operator のサービスアカウントにリンクします。

```
oc secrets link dv-operator dv-pull-secret --for=pull
```

OpenShift 4.2 以降へのインストール

1. ターミナルウィンドウで以下のコマンドを入力し、OpenShift クラスターにログインし、Red Hat イメージレジストリーへのアクセスに使用できるプルシークレットを作成します。

```
oc login
oc create secret docker-registry dv-pull-secret /
--docker-server=registry.redhat.io /
--docker-username=$username / (1)
--docker-password=$password /
--docker-email=$email_address
oc secrets link builder dv-pull-secret
oc secrets link builder dv-pull-secret --for=pull
```

(1) Red Hat カスタマーポータルのログインクレデンシャルを使用します。

2. クラスター管理者として OpenShift Web コンソールにログインします。

3. OpenShift メニューから **Operators** を展開し、**OperatorHub** をクリックします。
4. **Red Hat, Inc. が提供する Data Virtualization Operator 7.6.0**をクリックしてから、**Install** をクリックします。
5. **Create Operator Subscription** ページから、選択した namespace が Operator をインストールするプロジェクトの名前と一致することを確認し、**Subscribe** をクリックします。
Installed Operators ページには、**Data Virtualization Operator** を一覧表示し、インストールのステータスを報告します。
6. OpenShift メニューから **Workloads** を展開し、Pods をクリックして Operator **Pod** のステータスを確認します。数分後に、Operator サービスの Pod の実行が開始されます。
7. Data Virtualization Operator が Red Hat レジストリーからイメージを取得できるようにするには、ステップ 1 で作成したシークレットを Operator のサービスアカウントにリンクします。

```
oc secrets link dv-operator dv-pull-secret --for=pull
```

関連情報

- 「仮想データベースのデプロイ」.

6.2. 仮想データベースのデプロイ

仮想データベースおよびその対応する CR ファイルを作成した後に、Data Virtualization Operator を実行してデータベースを Openshift にデプロイします。

前提条件

- クラスター管理者は、仮想データベースをデプロイする OpenShift クラスターに Data Virtualization Operator を追加しました。
- Data Virtualization Operator がインストールされている OpenShift クラスターにアクセスできる。
- 仮想データベースの設定およびデプロイ方法についての情報を提供する YAML 形式の CR がある。
- Operator はビルドに必要な依存関係が含まれる Maven リポジトリーにアクセスできます。
- OpenShift は CR で参照されるデータソースにアクセスできます。

手順

1. ターミナルウインドウから OpenShift にログインし、仮想データベースを作成するプロジェクトを開きます。
2. コンピューターで、CR を含む **.yaml** ファイルが含まれるディレクトリーに切り替えます。
3. 以下のコマンドを入力して Operator を実行し、仮想データベースを作成します。

```
oc create -f <cr_filename.yaml>
```

& **lt;cr_filename.yaml&** gt; をデータソースの CR ファイルの名前に置き換えます。以下に例を示します。

```
oc create -f dv-customer.yaml
```

デプロイメントが完了すると、仮想データベースサービスが OpenShift クラスターに追加されます。サービスの名前は、カスタムリソースで指定された名前と一致します。

4. 以下のコマンドを入力して、仮想データベースが作成されていることを確認します。

```
oc get vdbs
```

OpenShift はプロジェクト内の仮想データベースの一覧を返します。特定の仮想化が利用可能かどうかを確認するには、以下のコマンドを入力します。

```
oc get vdb <dv-name>
```

デプロイされたサービスは、以下のクライアントからの接続をサポートします。

- ポート 31000 経由の JDBC クライアント。
- ポート 5432 を介した ODBC クライアントを含む PostgreSQL クライアント。
- OData クライアント（HTTP エンドポイントおよびルート経由）。

第7章 データのセキュリティー保護

データへの不正アクセスを防ぐには、以下の手段を実装できます。

- OpenID-Connect 認証および OAuth2 承認を有効にするように、OpenShift で Red Hat Single Sign-On との統合を設定します。
- ロールベースのアクセス制御を仮想データベースに適用します。
- 3Scale を設定して OData API エンドポイントをセキュアにします。
- データベースクライアント（ODBC と JDBC）と仮想データベース間の通信を暗号化します。

7.1. 仮想データベースの ODATA API のセキュリティー保護

データ仮想化を Red Hat Single Sign-On および Red Hat 3scale API Management と統合すると、高度な承認および認証制御を仮想データベースサービスの OData エンドポイントに適用できます。

Red Hat Single Sign-On テクノロジーは OpenID-Connect を認証メカニズムとして使用して API をセキュアにし、OAuth2 を承認メカニズムとして使用します。データ仮想化は、Red Hat Single Sign-On のみまたは 3scale と統合することができます。

デフォルトでは、3scale システムが同じクラスターおよび namespace に定義されている限り、仮想データベースの作成後にそのデータベースへの OData インターフェースは 3scale によって検出可能です。Red Hat Single Sign-On で OData API へのアクセスのセキュリティーを保護することにより、ユーザーロールを定義し、API エンドポイントへのロールベースのアクセスを実装できます。設定が完了したら、ビュー、列、またはデータソースレベルで仮想データベースへのアクセスを制御できます。許可されたユーザーのみが API エンドポイントにアクセスでき、各ユーザーにはそのロール（ロールベースのアクセス）に適したレベルのアクセスが許可されます。3scale を API へのゲートウェイとして使用することで、3scale の API 管理機能を活用し、API の使用状況を承認アカウントに対応して追跡および請求を行うことができます。

ユーザーがログインすると、3scale は Red Hat Single Sign-On パッケージを使用した認証をネゴシエートします。認証に成功すると、3scale は検証のためにセキュリティートークンを OData API に渡します。その後、OData API はトークンからパーミッションを読み取り、仮想データベースに定義されたデータロールに適用します。

前提条件

- Red Hat Single Sign-On が OpenShift クラスターで実行されている。Red Hat Single Sign-On のデプロイに関する詳細は、Red Hat Single Sign-On for OpenShift のドキュメントを参照してください。
- 仮想データベースをホストする OpenShift クラスターに Red Hat 3scale API Management がインストールされている。
- 3scale と Red Hat Single Sign-On の間でインテグレーションを設定している。詳細は、『Using the Developer Portal』の「Configuring Red Hat Single Sign-On integration」を参照してください。
  - **realm-management** ロールおよび **manage-clients** ロールが割り当てられている。
  - API ユーザーと指定したクレデンシャルを作成している。
  - OpenID-Connect を認証メカニズムとして使用し、OAuth2 を承認メカニズムとして使用するように 3scale を設定している。

7.1.1. OData をセキュア化するための Red Hat Single Sign-On の設定

Red Hat Single Sign-On に構成設定を追加して、データ仮想化との統合を有効にする必要があります。

前提条件

- Red Hat Single Sign-On が OpenShift クラスターで実行されている。Red Hat Single Sign-On のデプロイに関する詳細は、Red Hat Single Sign-On for OpenShift[Red Hat Single Sign-On] のドキュメントを参照してください。
- Data Virtualization Operator を実行して、Red Hat Single Sign-On が稼働しているクラスターに仮想データベースを作成します。

手順

1. ブラウザーから Red Hat Single Sign-On 管理コンソールにログインします。
2. データ仮想化サービスのレルムを作成します。
   a. master レルムのメニューから、**Master** にマウスをかざし、**Add realm** をクリックします。
   b. **datavirt** などのレルムの名前を入力してから **Create** をクリックします。
3. ロールを追加します。
   a. メニューから **Roles** をクリックします。
   b. **Add Role** をクリックします。
   c. ロールの名前（ **ReadRole** など）を入力し、**Save** をクリックします。
   d. 必要に応じて他のロールを作成して、組織の LDAP または Active Directory のロールにマッピングします。外部アイデンティティープロバイダーからのユーザーデータの反復に関する詳細は、『サーバー 管理ガイド』 を参照してください。
4. ユーザーを追加します。
   a. メニューから **Users** をクリックし、**Add user** をクリックします。
   b. **Add user** フォームで、ユーザー名（user など）を入力し、割り当てる他のユーザープロパティーを指定してから **Save** をクリックします。
   user フィールドのみは必須です。
   c. ユーザーの詳細ページから、**Credentials** タブをクリックします。
   d. ユーザーのパスワードを入力して確認し、**Reset Password** をクリックして、プロンプトが表示されたら **Change password** をクリックします。
5. ユーザーにロールを割り当てます。
   a. **Role Mappings** タブをクリックします。
   b. **Available Roles** フィールドで **ReadRole** をクリックしてから **Add selected** をクリックします。
6. **developer** という名前の 2 番目のユーザーを作成し、パスワードとロールをユーザーに割り当てます。

7. データ仮想化クライアントエントリーを作成します。
   クライアントエントリーは、データ仮想化サービスを SSO クライアントアプリケーションとして表します。メニューから **Clients** . をクリックします。**Create** をクリックして **Add Client** ページを開きます。**Client ID** フィールドにクライアントの名前を入力します（例： **dv-client** ）。**Client Protocol** フィールドで、**openid-connect** を選択します。**Root URL** フィールドを空白のままにして、**Save** をクリックします。

これで、Data Virtualization サービスの CR に SSO プロパティーを追加する準備が整いました。

7.1.2. カスタムリソースファイルへの SSO プロパティーの追加

Red Hat Single Sign-On が仮想データベースの OData エンドポイントをセキュアにするように設定した後、Red Hat Single Sign-On と統合するように仮想データベースを設定する必要があります。SSO を使用するように仮想データベースを設定するには、サービスの初回デプロイ時に使用した CR に SSO プロパティーを追加します（例： **dv-customer.yaml**）。プロパティーを環境変数として追加します。SSO 設定は、仮想データベースを再デプロイした後に有効になります。

この手順では、以下の Red Hat Single Sign-On プロパティーを CR に追加します。

Realm (KEYCLOAK_REALM)

仮想データベースの Red Hat Single Sign-On で作成したレルムの名前。

認証サーバーの URL(KEYCLOAK_AUTH_SERVER_URL)

Red Hat Single Sign-On サーバーのベース URL。通常、形式は https://host:port/auth です。

Resource name(KEYCLOAK_RESOURCE)

データ仮想化サービス用に Red Hat Single Sign-On で作成したクライアントの名前。

SSL 要件(KEYCLOAK_SSL_REQUIRED)

レルムへの要求に SSL/TLS が必要であるかどうかを指定します。すべての要求、外部要求のみ、または none に SSL/TLS を要求できます。

アクセスタイプ(KEYCLOAK_PUBLIC_CLIENT)

クライアントの OAuth アプリケーションタイプ。パブリックアクセスタイプは、ブラウザーからサインインするクライアント側のクライアント用です。

前提条件

- Data Virtualization Operator を実行して仮想データベースを作成している。
- 仮想データベースがデプロイされているクラスターで、Red Hat Single Sign-On が稼働している。
- 仮想データベースをデプロイするために使用した **dv-customer.yaml** などの CR YAML ファイルがある。
- Red Hat Single Sign-On 管理コンソールへの管理者アクセスがある。

手順

1. Red Hat Single Sign-On 管理コンソールにログインして、必要な認証プロパティーの値を見つけます。
2. テキストエディターで、仮想データベースのデプロイに使用した CR YAML ファイルを開き、Red Hat Single Sign-On プロパティーの値に基づく認証環境変数を定義します。
   以下に例を示します。

```
env:
  - name: KEYCLOAK_REALM
    value: master
  - name: KEYCLOAK_AUTH_SERVER_URL
    value: http://rh-sso-datavirt.openshift.example.com/auth
    - name: KEYCLOAK_RESOURCE
      value: datavirt
  - name: KEYCLOAK_SSL_REQUIRED
    value: external
  - name: KEYCLOAK_PUBLIC_CLIENT
    value: true
```

3. データ仮想化のセキュリティーを保護するために、以下の Maven アーティファクトのビルドソース依存関係を宣言します(**org.teiid:spring-keycloak**)。
以下に例を示します。

```
env:
  ....
  build:
    source:
      dependencies:
        - org.teiid:spring-keycloak
```

4. CR を保存します。

これで、仮想データベースの DDL にデータロールを定義する準備が整いました。

7.1.3. 仮想データベース DDL でのデータロールの定義

Red Hat Single Sign-On がデータ仮想化と統合するように設定した後、必要な設定変更を完了するには、仮想データベースの DDL でロールベースのアクセスポリシーを定義します。仮想データベースのデプロイ方法によっては、DDL を CR ファイルに埋め込むか、または別のファイルとして存在する可能性があります。

DDL ファイルに以下の情報を追加します。

- ロール名DDL で定義するロールは、Red Hat Single Sign-On で先に作成したロールにマップする必要があります。

  ヒント

  明確にするために、DDL ファイルのロール名を Red Hat Single Sign-On で指定したロール名に一致させます。一貫性を保つことで、各場所で定義するロールがどのように相互に関連するかを相互に関連付けるのが容易になります。

- 指定されたロールが付与されているユーザーにデータベースアクセスを許可します。たとえば、特定のテーブルビューでの SELECT パーミッションなど。

前提条件

- 「OData をセキュア化するための Red Hat Single Sign-On の設定」 の説明に従って、データ仮想化と連携するように Red Hat Single Sign-On を設定している。
- の説明に従って、仮想データベースの CR ファイルに SSO プロパティーを追加している 。

手順

1. テキストエディターで、仮想データベースをデプロイするために使用した DDL の記述が含まれるファイルを開きます。

2. ステートメントを挿入して、Red Hat Single Sign-On の仮想データベースユーザーに定義したロールを追加します。たとえば、**ReadRole** という名前のロールを追加するには、以下のステートメントを DDL に追加します。

   ```
   CREATE ROLE ReadRole WITH FOREIGN ROLE ReadRole;
   ```

   仮想データベースに実装するロールごとに、個別の **CREATE ROLE** ステートメントを追加します。

3. ロールが割り当てられたユーザーのデータベースオブジェクトに対するアクセスレベルを指定する文を挿入します。以下に例を示します。

   ```
   GRANT SELECT ON TABLE "portfolio.CustomerZip" TO ReadRole
   ```

   仮想データベースに実装する各ロールに個別の **GRANT** ステートメントを追加します。

4. CR または DDL ファイルを保存して閉じます。
   これで、仮想データベースを再デプロイできるようになりました。Data Virtualization Operator を実行して仮想データベースをデプロイする方法は、*6章Data Virtualization Operator の実行による仮想データベースのデプロイ* を参照してください。

   仮想データベースを再デプロイしたら、Red Hat Single Sign-On 管理コンソールにリダイレクト URL を追加します。詳細は、「Red Hat Single Sign-On 管理コンソールでのデータ仮想化クライアントのリダイレクト URI の追加」 を参照してください。

7.1.4. Red Hat Single Sign-On 管理コンソールでのデータ仮想化クライアントのリダイレクト URI の追加

仮想データベースの SSO を有効にして再デプロイしたら、「OData をセキュア化するための Red Hat Single Sign-On の設定」 で作成したデータ仮想化クライアントのリダイレクト URI を指定します。

リダイレクト URI、またはコールバック URL は、OpenID Connect を使用して認証し、リダイレクトメカニズムを使用してアイデンティティープロバイダーと通信する OData クライアントなどの **パブリック** クライアントに必要です。

OIDC クライアントのリダイレクト URI を追加する方法は、「 NameOfRHSSOServerAdmin 」を参照してください。

前提条件

- 仮想データベースの SSO を有効にし、Data Virtualization Operator を使用してこれを再デプロイしている。
- Red Hat Single Sign-On 管理コンソールへの管理者アクセスがある。

手順

1. ブラウザーで Red Hat Single Sign-On 管理コンソールにログインします。

2. データ仮想化サービスのクライアントを作成したセキュリティーレルムから、メニューで **Clients** をクリックし、以前に作成したデータ仮想化クライアントの ID（ **dv-client**など）をクリックします。

3. **Valid Redirect URIs** フィールドに OData サービスのルート URL を入力し、アスタリスクを追加します。例： **http://datavirt.odata.example.com/***

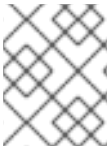
4. Red Hat Single Sign-On が OData API への呼び出しをインターセプトするかどうかをテストします。

   a. ブラウザーから、OData エンドポイントのアドレスを入力します。以下に例を示します。

   ```
   http://datavirt.odata.example.com/odata/CustomerZip
   ```

   ログインページにより、認証情報の入力が求められます。

5. 許可されたユーザーの認証情報でサインインします。
   データの表示は、サインインに使用するアカウントのロールによって異なります。



注記

odata/$metadata などの一部のエンドポイントは、他のサービスで検出できるようにセキュリティーフィルタリングから除外されます。

7.2. エンドポイントトラフィック暗号化のカスタム証明書

Data Virtualization は TLS 証明書を使用して、JDBC と ODBC データベースクライアントと仮想データベースサービス間のネットワークトラフィックを暗号化します。独自のカスタム証明書を指定するか、OpenShift 認証局が生成するサービス証明書を使用できます。カスタム TLS 証明書を指定しない場合、Data Virtualization Operator はサービス証明書を自動的に生成します。

サービス証明書は、内部および外部クライアントの暗号化された通信によく似ています。ただし、内部クライアント（つまり、同じ OpenShift クラスターにデプロイされるクライアント）のみがサービス証明書の信頼性を検証できます。

OpenShift サービス証明書には以下の特徴があります。

- PEM base-64 でエンコードされた形式のパブリックキー証明書(**tls.crt**)およびプライベートキー(**tls.key**)で構成されます。
- OpenShift Pod の暗号化シークレットに保存されます。
- OpenShift CA によって署名されます。
- 1 年間有効です。
- 有効期限前に自動的に置換されます。
- 内部クライアントのみが検証できます。

外部クライアントは、OpenShift 認証局が生成する証明書の有効性を認識しません。外部クライアントが証明書を検証できるようにするには、信頼できるサードパーティーの認証局(CA)からのカスタム証明書を指定する必要があります。このような証明書は汎用的に認識され、どのクライアントでも検証できます。カスタム証明書を仮想データベースに追加するには、Data Virtualization Operator を実行してサービスを作成する前に、OpenShift にデプロイする暗号化シークレットに証明書に関する情報を指定します。

暗号化シークレットを OpenShift にデプロイすると、仮想データベースの作成時に Data Virtualization Operator で利用可能になります。Operator は CR の仮想データベースの名前に一致する名前でシークレットを検出し、指定された証明書を使用してデータベースクライアントとの接続を暗号化するようにサービスを自動的に設定します。

7.3. カスタム TLS 証明書を使用したデータベースクライアントとエンドポイント間の通信の暗号化

カスタム TLS 証明書を OpenShift に追加して、JDBC または ODBC クライアントと仮想データベースサービス間の通信を暗号化できます。カスタム証明書は信頼できるサードパーティーの認証局(CA)によって発行されるため、クライアントは証明書で CA 署名を認証できます。

OpenShift Pod がカスタム証明書を使用してトラフィックを暗号化するように設定するには、証明書の詳細を OpenShift シークレットに追加し、仮想データベースを作成する namespace にシークレットをデプロイします。サービスを作成する前に、シークレットを作成する必要があります。

前提条件

- 信頼されたサードパーティーの CA からの TLS 証明書がある。
- シークレットおよび仮想データベースを作成する OpenShift プロジェクトへの開発者または管理者アクセスがある。

手順

1. YAML ファイルを作成し、**kubernetes.io/tls** タイプのシークレットを定義し、以下の情報を追加します。
   - TLS キーペアの公開鍵と秘密鍵。
   - 作成する仮想データベースの名前。
   - 仮想データベースを作成する OpenShift namespace。
     以下に例を示します。

```
apiVersion: v1
kind: Secret
type: kubernetes.io/tls
metadata:
  name: dv-customer (1)
  namespace: myproject (2)

data: (3)
  tls.crt: >-
     -----BEGIN CERTIFICATE-----
     [...]
     -----END CERTIFICATE-----
  tls.key: >-
     -----BEGIN PRIVATE KEY-----
     [...]
     -----END PRIVATE KEY-----
```

1. シークレットの名前。シークレット名は、Data Virtualization Operator が仮想データベースを作成するために使用する CR YAML ファイルの仮想データベースオブジェクトの名前と一致する必要があります（例： **dv-customer** ）。
2. 仮想データベースサービスがデプロイされる OpenShift namespace（例： **myproject** ）。
3. **data** 値は、base64 でエンコードされた PEM 形式の TLS 公開鍵証明書(**tls.crt**)の内容と、プライベート暗号化キー(**tls.key**)で構成されています。

2. ファイルを **tls_secret.yaml** として保存します。
3. 端末ウィンドウを開き、シークレットを追加する OpenShift プロジェクトにサインインしてから、以下のコマンドを入力します。

```
$ oc apply -f tls_secret.yaml
```

4. TLS シークレットを OpenShift にデプロイした後に、Data Virtualization Operator を実行して、シークレットに指定された名前で仮想データベースを作成します。
Operator が仮想データベースを作成する場合、これはシークレットの名前を CR のサービスに指定した名前に一致します。次に、Operator はシークレットを使用してサービスとのクライアント通信を暗号化するようにサービスを設定します。

7.4. シークレットの使用によるデータソースの認証情報の保存

シークレットオブジェクトを作成してデプロイし、環境変数の値を保存します。

シークレットは主に、プロパティーの値を隠すことで機密データを保護するため存在しますが、それらを使用して任意のプロパティーの値を保存できます。

前提条件

- データソースにアクセスするために必要なログイン認証情報およびその他の情報がある。

手順

1. データソースの認証情報を含めるためにシークレットファイルを作成し、これを YAML ファイルとしてローカルに保存します。以下に例を示します。

secrets.yml ファイルのサンプル

```
apiVersion: v1
kind: Secret
metadata:
  name: postgresql
type: Opaque
stringData:
  database-user: bob
  database-name: sampledb
  database-password: bob_password
```

2. OpenShift にシークレットオブジェクトをデプロイします。

a. OpenShift にログインし、仮想データベースに使用するプロジェクトを開きます。以下に例を示します。
oc login --token=<token> --server=https://<server> oc project <projectName>

b. 以下のコマンドを実行してシークレットファイルをデプロイします。
oc create -f ./secret.yaml

3. シークレットから値を取得するには、環境変数を設定します。

a. 環境変数で **valueFrom:/secretKeyRef** 形式を使用して、手順 1 で作成したシークレットのキーから変数がこの値を取得するように指定します。
たとえば、以下の抜粋では **SPRING_DATASOURCE_SAMPLEDB_PASSWORD** は **postgresql** シークレットの **database-password** キーへの参照から値を取得します。

```
- name: SPRING_DATASOURCE_SAMPLEDB_PASSWORD
 valueFrom:
   secretKeyRef:
     name: postgresql
     key: database-password
```

関連情報

- OpenShift でシークレットを使用する方法の詳細は、OpenShift ドキュメント の「機密データの Pod の提供」を 参照してください。

第8章 FUSE ONLINE での仮想データベースの作成および操作

Fuse Online では、選択した複数のデータソースからデータを統合する仮想データベースを作成できます。作成される仮想データベースサービスのデプロイ後、アプリケーション開発者およびその他のデータベースユーザーは単一の物理データベースであるかのように接続できます。

重要

Fuse Online の Data Virtualization はテクノロジープレビューの機能です。テクノロジープレビューの機能は、Red Hat の実稼働環境のサービスレベルアグリーメント (SLA) ではサポートされず、機能的に完全ではないことがあります。Red Hat は実稼働環境でこれらを使用することを推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、https://access.redhat.com/ja/support/offerings/techpreview を参照してください。

Fuse Online で仮想データベースを作成した後、Fuse Online ツールを使用して以下を行うことができます。

- データソースの追加または削除。
- 異なるテーブルまたはソースからのデータの追加または編集。
- SQL クエリーを送信し、想定された結果がビューによって返されるかどうかをテスト。
- 仮想データベースを定義するスキーマの変更。
- 仮想データベースを公開して、OpenShift で利用できるようにする。
- 仮想データベースの削除。

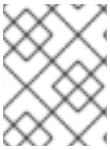
前提条件

- Red Hat Integration の 2020.Q1 リリースがあり、7.6 を実行している。
- 7.6 の Data Virtualization UI がインストール時に有効化されている。詳細は、『Installing and Operating Fuse Online on OpenShift Container Platform』の「Enabling data virtualization in Fuse Online on OCP」を参照してください。

8.1. FUSE ONLINE での仮想データベースの作成

Fuse Online では、**Connections** ページで使用できるアプリケーションまたはサービスからビューをインポートする仮想データベースを作成できます。

作成する仮想データベースごとに、データソースをインポートし、追加する各データソースからテーブルを選択する必要があります。作成される仮想データベースのビューは、インポートするデータベーステーブルに直接マップされます。最初の作成を行った後に、複数のテーブルからデータを結合する仮想データベースにビューを追加できます。



注記

このテクノロジープレビューでは、リレーショナルデータベース、MongoDB、および Salesforce からのみ、Fuse Online で仮想データベースを作成できます。

前提条件

- 以下の 1 つまたは複数のデータソースへのコネクションが Fuse Online 環境に存在する必要があります。
  - PostgreSQL や MySQL などのリレーショナルデータベース。
  - MongoDB データベース
  - Salesforce データベース

手順

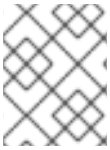
1. Fuse Online のナビゲーションサイドバーで **Data** をクリックします。
2. **Create Data Virtualization** をクリックします。
3. **Create New Data Virtualization** ページで、仮想データベースの名前を入力して **Create** をクリックします。
   - 指定する名前は、データベースのコンテンツや目的が分かるようにし、アプリケーションの開発者や SQL ユーザーが簡単にコードに挿入できる長さにします。
   - 名前には、英数字 ([a-z] [A-Z]、[0-9])、およびハイフン (-) のみを使用できます。
4. 仮想化のページで、**Import Data Source** をクリックします。
5. **Import Data Source** ページで、アクティブなデータソースのタイルをクリックし、**Next** をクリックします。
6. 利用可能なテーブルの一覧から、仮想データベースに追加するテーブルを 1 つ以上選択し、**Done** をクリックします。
   インポートが完了すると確認メッセージが表示されます。ドラフト仮想化の **Views** タブには、インポートした各テーブルのビューが一覧表示されます。

   これで、既存のビューの編集や他のビューの作成を行えるようになり、仮想データベースをパブリッシュして使用できるようにすることが可能になりました。

8.2. FUSE ONLINE での仮想データベースへのビューの追加

ビューを仮想データベースに追加し、新しいテーブルのデータのビューを提供します。

最初に仮想データベースを作成すると、最初のデータソースからインポートしたビューのみが含まれます。他のテーブルからデータを組み込む場合は、仮想データベースにビューを追加します。元のデータソースのテーブルまたは他のデータソースからテーブルに基づいてビューを追加できます。



注記

このテクノロジープレビューリリースでは、各ビューにテーブルを 1 つだけ追加できます。

前提条件

- ビューを追加する仮想データベースは、**Draft** または **Published** 状態の Fuse Online で利用できます。Fuse Online を使用して、Fuse Online の外部で作成された仮想データベースにビューを追加することはできません。

- 統合するテーブルが含まれるデータソースに Fuse Online コネクションが存在する必要があります。
- ビューで使用するテーブルの名前を知っている必要があります。

手順

1. Fuse Online のナビゲーションサイドバーで **Data** をクリックします。
2. **Data Virtualizations** ページの一覧から、変更する仮想データベースを見つけ、**Edit** をクリックします。
3. **Create a View** をクリックします。
4. データソースを展開して、含まれるテーブルを表示します。
5. 仮想データベースに追加するテーブルを選択し、**Next** をクリックします。
6. **Create a View** ページで **View Name** フィールドに名前を入力し、**Done** をクリックします。
View Editor は作成したビューの SQL を表示します。**Preview** パネルにビューのデータが表示されます。
7. データが表示されない場合は、**Refresh** をクリックします。
8. **Done** をクリックして、ビューを閉じます。
仮想データベースがこれまでにパブリッシュされていた場合は、仮想データベースを再パブリッシュし、新しいビューを使用できるようにする必要があります。

その他のリソース

- 経験のある SQL プログラマーは、仮想データベースのデフォルトの SQL ステートメントを直接編集して、ビューを追加することもできます。詳細は、「Fuse Online で View Editor を使用して仮想データベースを定義する DDL を変更」 を参照してください。
- 「Fuse Online で仮想データベースをパブリッシュしてアクセスできるようにする」
- 「SQL テストクエリーの送信による Fuse Online での仮想データベースのプレビュー」
- 「Fuse Online で View Editor を使用して仮想データベースを定義する DDL を変更」

8.3. FUSE ONLINE で VIEW EDITOR を使用して仮想データベースを定義する DDL を変更

Fuse Online で仮想データベースを作成するプロセスは、多くのタスクを自動化し、基礎となる SQL コードの複雑さを表さないように設計されています。

仮想データベースのビューを作成すると、Fuse Online でビューを定義するデータ定義言語 (DDL) が自動的に生成されます。DDL は、ビューのスキーマ、テーブル、列、およびその他のフィールドを記述する SQL ステートメントのセットです。

Fuse Online は、仮想データベースの基本的なビューを追加するツールを提供しますが、SQL を理解していて、ビューをより細かく設計したい場合は、ビューの DDL を直接編集できます。Fuse Online では、開発者は組み込みのビューエディターを使用してこれらの SQL ステートメントを変更できます。さらに、この SQL エディターには、SQL キーワードの一覧を提供するコード補完機能が含まれています。

変更を保存した後、SQL コードに構文エラーが含まれないようにするため、組み込みの検証ツールが実行されます。

前提条件

- SQL-MED 仕様に基づいたデータ定義言語 (DDL) を使用して、データベースの構造を定義し、外部に保存されたデータを統合した経験。

手順

1. Fuse Online のナビゲーションサイドバーで **Data** をクリックします。
2. **Data Virtualizations** ページで、変更する仮想データベースを見つけ、**Edit** をクリックします。
3. **Views** タブで、編集するビューを見つけ、**Edit** をクリックします。
4. 必要に応じて SQL を更新します。編集時に Ctrl+Space を押して、コード補完ツールを開きます。
5. 変更が完了したら、**Save** をクリックします。
   Fuse Online で SQL が検証され、ビューに無効なコードが含まれる場合はエラーが返されます。

   SQL の検証後、ビューの更新の結果がプレビューパネルに表示されます。プレビューには、結果セットの最初の 15 行が表示されます。
6. **Done** をクリックして **View Editor** を閉じ、ビューの一覧に戻ります。
   仮想データベースがこれまでにパブリッシュされていた場合は、仮想データベースを再パブリッシュし、変更を反映する必要があります。

関連情報

- 「Fuse Online で仮想データベースをパブリッシュしてアクセスできるようにする」
- データ仮想化 DDL ファイルで SQL を使用する方法については、『Teiid Reference Guide』を参照してください。
- 結果セットを変更するには、デフォルトのクエリーを変更し、別の行制限や行オフセットを指定します。詳細はを参照してください。 「SQL テストクエリーの送信による Fuse Online での仮想データベースのプレビュー」

8.4. SQL テストクエリーの送信による FUSE ONLINE での仮想データベースのプレビュー

仮想データベースをパブリッシュしてアプリケーションで使用できるようにする前に、ビューに対してテストクエリーを実行し、想定する情報が返されるかどうかを検証することができます。

SQL の **SELECT * FROM** ステートメントが仮想データベースビューに送信されると、デフォルトのプレビューには最初の 15 件の結果が表示されますが、Fuse Online の組み込み SQL クライアントを使用すると、変更したテストクエリーをビューに送信できます。行の制限と行オフセットを指定すると、デフォルトの結果セットを調整できます。

クエリー対象のビューが SQL でないデータソースからである場合、データ仮想化エンジンは SQL クエリーをそのデータソースが解釈できる形式に変換します。

前提条件

- Fuse Online で作成した有効な仮想データベースが必要です。

手順

1. Fuse Online のナビゲーションサイドバーで **Data** をクリックします。
2. **Data Virtualizations** ページで、テストするビューが含まれる仮想データベースのエントリーにある **Edit** をクリックします。
3. **SQL Client** タブをクリックします。
4. **View** フィールドで、テストするビューを選択します。
5. **Row Limit** フィールドに、表示する行数を指定します。
6. **Row Offset** フィールドに、スキップする行数を指定します。
7. **Submit** をクリックします。**Query Results** テーブルに結果セットが表示されます。

8.5. FUSE ONLINE で仮想データベースをパブリッシュしてアクセスできるようにする

Fuse Online で仮想データベースを定義したら、ユーザーおよびアプリケーションがアクセスできるようにパブリッシュする必要があります。

仮想データベースをパブリッシュすると、データソースとビューをランタイムイメージにインポートして実装した、スキーマ定義がビルドされます。Fuse Online は、ランタイムイメージを個別にスケーリングできる仮想データベースコンテナーイメージとして、OpenShift にデプロイします。

仮想データベースをパブリッシュした後、サービスとして利用できるようになり、Fuse Online の **Connections** ページに表示されます。サービスはリレーショナルデータベースのように動作し、クライアントは標準のインターフェース上で接続できます。Fuse Online インテグレーションワークフローに組み込むことができ、JDBC および OData クライアントで利用できます。

前提条件

- Fuse Online で仮想データベースが作成済みである必要があります。
- 必要なビューを仮想データベースに追加済みである必要があります。

手順

1. Fuse Online のナビゲーションサイドバーで **Data** をクリックします。
2. **Data Virtualizations** ページで、公開する仮想データベースを見つけ、オーバーフローメニューから **Publish** をクリックします。
   パブリッシュするために仮想データベースが送信されたことを伝える確認メッセージが表示され、その処理の状況が進捗バーに表示されます。
   - パブリッシュプロセスに成功すると、Fuse Online は以下の更新を行います。
     - **Data Virtualizations** ページの仮想データベースエントリーのステータスラベルが **Draft** から **Published** に変わります。

- 仮想データベースの OData エンドポイントへの URL が仮想データベースエントリーに表示されます。
- 仮想データベースサービスが **Connections** ページに追加され、そのサービスへの JDBC コネクションが作成されます。
  Connections ページから仮想データベースサービスのエントリーを開くと、JDBC の URL を確認できます。

- パブリッシュの処理に失敗した場合、エントリーに **Error** というラベルが付けられます。

8.6. FUSE ONLINE での仮想データベースの削除

Fuse Online で作成した仮想データベースを永久に削除することができます。仮想データベースは、パブリッシュ済みまたは下書きであるかに関わらず削除することができます。

仮想データベースによって消費されるデータソースは削除の影響を受けません。Fuse Online とデータソース間のコネクションはそのまま維持されます。

前提条件

- Fuse Online で作成した仮想データベースがあり、これを削除する必要があります。

手順

1. Fuse Online のナビゲーションサイドバーで **Data** をクリックします。
2. **Data Virtualizations** ページで、削除する仮想データベースのオーバーフローメニューをクリックし、**Delete** をクリックします。
3. プロンプトが表示されたら、**Delete** をクリックし、仮想データベースを削除することを確認します。
   仮想化が削除されると確認メッセージが表示されます。

第9章 仮想データベースの監視

Prometheus は、Red Hat OpenShift 環境にデプロイされたサービスの監視に使用できる、システムおよびサービスのオープンソースの監視およびアラートツールキットです。Prometheus は、指定の間隔で設定されたサービスからメトリクスを収集および保存します。さらに、ルール式の評価や結果の表示を行い、指定の条件が true になるとアラートをトリガーできます。

重要

Prometheus に対する Red Hat の サポートは、Red Hat 製品ドキュメントに記載の設定および構成の推奨事項に限定されます。

Prometheus は Data Virtualization サービスと通信して、メトリクスおよびデータを取得します。ユーザーはデータをクエリーしたり、Grafana などのビジュアライゼーションを使用してダッシュボードで傾向を表示できます。

OpenShift サービスのモニタリングを有効にするには、サービスはエンドポイントを Prometheus に公開する必要があります。このエンドポイントは、メトリクスのリストとメトリクスの現在の値を提供する HTTP インターフェースです。Data Virtualization Operator を使用して仮想データベースを作成する場合、Data Virtualization サービスは HTTP エンドポイントを Prometheus に自動的に公開します。Prometheus は定期的にターゲット定義の各エンドポイントをスクレイピングし、収集したデータをそのデータベースに書き込みます。

Prometheus を OpenShift クラスターにデプロイすると、同じクラスターにデプロイする仮想化のメトリクスが自動的に Prometheus サービスに公開されます。

Prometheus を使用した Red Hat Integration のサービスの監視に関する詳細は、「Monitoring Red Hat Integration」を参照してください。

第10章 従来の仮想データベースファイルの DDL 形式への移行

データ仮想化テクノロジープレビューでは、SQL-MED DDL（データ定義言語）形式で仮想データベースの構造を定義する必要があります。一方、Wildfly や Red Hat JBoss Data Virtualization のオファリングなど、レガシーの Teiid 仮想データベースの構造は、**.xml** または **.vdb** 形式のファイルを使用して定義します。

レガシーデプロイメント用に開発した仮想データベースの設計を再利用できますが、まずファイルの形式を更新する必要があります。ファイルの変換には、移行ツールを使用できます。ファイルを変換した後、仮想データベースをコンテナーイメージとして再ビルドし、OpenShift にデプロイできます。

移行に関する考慮事項

以下の機能は、JBoss Data Virtualization および Teiid の仮想データベースでサポートされている機能は、このテクノロジープレビューのデータ仮想化リリースで制限または利用できない可能性があります。

データソースの互換性

本リリースでは、すべてのデータソースを使用することはできません。互換性のあるデータソースの一覧は、「互換性のあるデータソース」 を参照してください。

内部分散マテリアル

利用できません。

ResultSet キャッシュ

利用できません。

ランタイムメタデータを使用した他の仮想データベースのインポート

DDL を使用して仮想データベースのメタデータを指定する必要があります。

マルチソース vdb ソースのランタイム操作

利用できません。

移行ユーティリティーは、次の 2 つの方法で使用できます。

VDB ファイルのみを検証する手順

この方法を使用して、ユーティリティーが VDB ファイルを正常に変換できるかどうかを確認します。このユーティリティーは VDB ファイルを変換し、検証エラーをターミナルに報告します。検証エラーがない場合は、ユーティリティーは結果の DDL を表示しますが、変換された DDL はファイルに保存されません。

VDB ファイルおよび VDB ファイルを検証し、DDL ファイルに保存するには、以下を行います。

このファイルは、検証エラーがない場合にのみ保存されます。

移行ツールは XML ファイルでのみ機能します。**.vdb** ファイル拡張子を持つファイルは、複数のフォルダーが含まれるファイルアーカイブです。**.vdb** 形式のレガシーファイルがある場合は、Teiid Designer を使用してファイルを XML 形式にエクスポートしてから、移行ツールを実行して、作成された XML ファイルを変換します。

前提条件

- **.xml** 形式のレガシーの仮想データベースファイルがある。
- Teiid OpenShift リポジトリーから **settings.xml** ファイル をダウンロードします。Maven はファイルの情報を使用して移行ツールを実行します。

10.1. レガシー仮想データベースの XML ファイルを検証し、DDL 形式で表示

レガシー仮想データベースでテスト変換を実行して検証エラーを確認し、生成される DDL ファイルを表示できます。このように移行ツールを実行すると、変換された DDL ファイルは保存されません。

手順

1. Teiid OpenShift リポジトリーからダウンロードした **settings.xml** ファイルが含まれるディレクトリーを開き、以下のコマンドを入力します。

```
$ mvn -s settings.xml exec:java -Dvdb=<path_to_vdb_xml_file>
```

以下に例を示します。

```
$ mvn -s settings.xml exec:java -Dvdb=rdbms-example/src/main/resources/vdb.xml
```

移行ツールは指定の **.xml** ファイルを確認し、検証エラーを報告します。検証エラーがない場合、移行ツールには画面に仮想データベースの **.ddl** バージョンが表示されます。

10.2. レガシー仮想データベースの XML ファイルの変換および DDL ファイルとしての保存

移行ツールを実行して、レガシーの仮想データベースファイルを **.ddl** 形式に変換してから、**.ddl** ファイルを指定されたディレクトリーに保存できます。移行ツールは、指定する **.xml** ファイルで検証エラーを確認します。検証エラーがない場合、移行ツールはファイルを **.ddl** 形式に変換し、これを指定したファイル名とディレクトリーに保存します。

手順

- Teiid OpenShift リポジトリーからダウンロードした **settings.xml** ファイルが含まれるディレクトリーを開き、以下のコマンドを入力します。

```
$ mvn -s settings.xml exec:java -Dvdb=<path_to_vdb_xml_file> -Doutput=
<path_to_save_ddl_file>
```

以下に例を示します。

```
$ mvn -s settings.xml exec:java -Dvdb=rdbms-example/src/main/resources/vdb.xml -
Doutput=rdbms-example/src/main/resources/vdb.ddl
```