Red Hat Integration 2020.Q1

Using Data Virtualization

TECHNOLOGY PREVIEW - User's guide to Data Virtualization

Last Updated: 2021-02-19

Red Hat Integration 2020.Q1 Using Data Virtualization

TECHNOLOGY PREVIEW - User's guide to Data Virtualization

Legal Notice

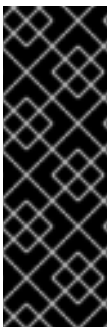
Abstract

Combine data from multiple sources so that applications can connect to a single, virtual data model

Table of Contents

CHAPTER 1. HIGH-LEVEL OVERVIEW OF DATA VIRTUALIZATION

Data virtualization is a container-native service that provides integrated access to multiple diverse data sources, including relational and noSQL databases, files, web services, and SaaS repositories through a single uniform API. Applications and users connect to a virtual database over standard interfaces (OData REST, or JDBC/ODBC) and can interact with data from all configured data sources as if the data were served from a single relational database.



IMPORTANT

Data virtualization is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview/.

The Red Hat data virtualization technology is based on Teiid, the open source data virtualization project. For more information about Teiid, see the Teiid community documentation.

CHAPTER 2. CREATING VIRTUAL DATABASES

To add a virtual database, you must complete the following tasks:

1. Install the Data Virtualization Operator.
2. Design and develop the database.
3. Create a custom resource (CR) file for deploying the database.
4. Deploy the virtual database to OpenShift by running the Data Virtualization Operator with the CR.

You can use any of the following methods to design a virtual database.

Create a virtual database from a DDL file

Define the entire contents of a virtual database, including the DDL, in a YAML file. For more information, see Chapter 3, *Creating a virtual database by embedding DDL statements in a custom resource (CR)* .

Create a virtual database as a Maven artifact

Create a virtual database from one or more DDL files and generate a Maven artifact for deployment For more information, see Chapter 4, *Creating a virtual database as a Maven artifact* .

Create a virtual Database as a fat JAR

Use the Teiid Spring Boot plug-in to create a Maven-based Java project with a DDL file, and then generate a fat JAR for deployment. For more information, see Chapter 5, *Creating a virtual database as a fat JAR*.

In each of the methods, you use SQL data definition language (DDL) to specify the structure of the virtual database, and you then configure the data sources that you want the virtual database to read from and write to.

There are advantages and disadvantages to using each method, the runtime virtualizations that any of the methods create have equivalent features. Choose a method based on the complexity of your project and on whether you want to be able to test the virtualization as a standalone component or on OpenShift only.

After you define the virtual database, you use the Data Virtualization Operator to deploy the virtualization from a custom resource (CR). The custom resource that you use to deploy a virtual database varies with the method that you used to design the virtual database. For more information, see Chapter 6, *Running the data virtualization operator to deploy a virtual database* .

After you set up connections to a data source, you can optionally configure authentication to Red Hat SSO to secure the connections, and enable single sign-on.

NOTE

You can also create virtual databases in Fuse Online (Technology Preview). Virtual databases that you create in Fuse Online provide a limited set of features.

Additional resources

- Section 6.1, “Installing the Data Virtualization Operator on OpenShift” .
- Chapter 8, *Creating and working with virtual databases in Fuse Online* .

- Chapter 7, *Securing data*.

2.1. COMPATIBLE DATA SOURCES

You can configure create virtual databases from a range of different data sources.

For each data source, you must provide the name of the *translator* that can interpret the commands and data that pass between the virtual database and the data source.

The following table lists the data source types from which you can create virtual databases, and the names of the translators for each data source:

Data source		Translator name
Amazon S3/ Ceph		amazon-s3
Google Sheets		google-spreadsheet
Data Grid (Infinispan)		infinispan-hotrod
MongoDB		mongodb
Relational databases		
	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

Data source	Translator name
Salesforce	salesforce
sFTP	file
SOAP	soap or ws

2.2. CREATING CUSTOM RESOURCES TO DEPLOY VIRTUALIZATIONS

Before you can use the Data Virtualization Operator to create a virtual database, you must specify properties for the data source in a custom resource (CR) file.

When you run the Data Virtualization Operator, it reads information from the CR that it needs to convert a data virtualization artifact into an image and deploy it to OpenShift.

Properties in the CR specify environment variables that store the credentials that the Operator requires to connect to a data source. You can specify the values directly in the CR, or provide references to an OpenShift *secret* that stores the values. For more information about creating secrets, see Section 7.4, "Using secrets to store data source credentials".

NOTE

Period characters (**.**) are not valid for use in environment variables. When you add variable names to the CR, use underscore characters (_) as separators.

The information that you add to the CR depends on the type of artifact that you created for the virtualization and the location of artifact. You can also supply configuration information in the CR.

NOTE

If you want OpenShift to create an HTTP endpoint for the deployed virtualization, add the property **spec/exposeVia3scale** to the CR, and set its value to **false**. If the value is set to **true** it is assumed that 3scale manages the endpoint, and no HTTP endpoint is created.

Additional resources

- Section 2.2.1, "Environment variables in custom resources"
- Section 3.1, "Creating a CR to deploy a DDL artifact"
- Section 4.1, "Creating a custom resource (CR) to deploy a Maven artifact"
- Section 5.3, "Creating a CR to deploy a fat JAR"

2.2.1. Environment variables in custom resources

You set environment variables in the custom resource file to enable your virtual database to connect to data sources.

Because you typically deploy virtual databases to multiple OpenShift environments, such as to a staging

and a production environment, you might want to define different data source properties for each environment. For example, the login credentials that you must provide to access a data source in the staging environment are probably different from the credentials that you use to access the data source in the production environment. To define unique values in each environment, you can use environment variables.

The environment variables that you define in a CR replace any static properties that you might set elswhere, for example, in the **application.properties** file for a fat JAR. If you define a property in the properties file and in the CR, the value in the CR file takes precedence.

You can combine the use of environment variables and secret objects to specify and protect the unique details for each environment. Instead of specifying static values for environment variables directly in the CR, you can store the values for each deployment environment in secret objects that are unique to each environment. The value of each environment variable in the CR contains only a key reference, which specifies the name of a secret object, and the name of a token in the secret. The token stores the actual value. At runtime, environment variables retrieve their values from the tokens.

By using secrets to store the values of your environment variables, you can use a single version of the CR across environments. The secret objects that you deploy in each environment must have the same name, but in each environment you assign token values that are specific to the environment.

Additional resources

- For more information about using secrets, see Section 7.4, “Using secrets to store data source credentials”.
- For information about adding a CR file, see Section 2.2, “Creating custom resources to deploy virtualizations”.

CHAPTER 3. CREATING A VIRTUAL DATABASE BY EMBEDDING DDL STATEMENTS IN A CUSTOM RESOURCE (CR)

You can define the structure of a virtual database by adding DDL statements directly within a custom resource file. During deployment, the Operator runs a source-to-image (S2I) build on OpenShift based on the dependencies that it detects in the virtual database artifact. To prevent build failures, ensure that any dependencies that your virtual database requires, such as JDBC driver dependencies, can be found at build time.

Advantages of using DDL in the CR to create a virtual database

- Simple and minimalistic.
- Code and configuration for a virtualization are in a single file. No need to create a separate CR file.
- Easy to manage.

Disadvantages of using DDL in the CR to create a virtual database

- Embedding the DDL for the virtual database in the custom resource (CR) file results in a large file.
- Because the DDL is embedded in the CR YAML file, you cannot version the DDL and other aspects of the configuration independently.
- If you deploy to multiple environments, you must store properties in configuration maps or secrets to make them independent of the custom resource.

Prerequisites

- You have Developer or Administrator access to an OpenShift cluster in which the data virtualization operator is installed.
- You have a compatible data source and the OpenShift cluster can access it.
- The data virtualization operator has access to any Maven repositories that contain build dependencies for the virtual database.
- You have information about the connection settings for your data sources, including login credentials.
- You have a DDL file for the virtual database that you want to create, or you know how to write the SQL code to design the database.

Procedure

- Create a CR text file in YAML format and save it with a .yaml or .yml extension, for example **dv-customer.yaml**
  The following example shows the elements to include in a CR for a virtual database that uses a postgreSQL data source:

  Example: 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) Specifies the number of instances to deploy. The default setting is 1.

(2) Specifies the configuration properties for this virtualization, primarily the configuration for connecting to data sources. The properties in the example apply to a connection to a PostgreSQL database. For information about supported data sources and their properties, see Section 2.1, “Compatible data sources”.

(3) Specifies a list of Maven dependency JAR files in GAV format (groupId:artifactid:version). These files define the JDBC driver files and any custom dependencies for the data source. Typically, the Operator build automatically adds libraries that are available in public Maven repositories.

4 Defines the virtual database in DDL form. For information about how to use DDL to define a virtual database, see *DDL metadata for schema objects* in the Data virtualization

5 Specifies the location of any private or non-public repositories that contain dependencies or other virtual databases. You can specify multiple repositories.

After you create the YAML file, you can run the Data Virtualization Operator to deploy the virtual database to OpenShift. For more information, see Chapter 6, *Running the data virtualization operator to deploy a virtual database*.

3.1. CREATING A CR TO DEPLOY A DDL ARTIFACT

If you create a virtual databases by embedding DDL directly in a CR, you already have the CR that the Data Virtualization Operator uses for deployment. For information about the CR for a DDL artifact, see Chapter 3, *Creating a virtual database by embedding DDL statements in a custom resource (CR)* .

Run the Data Virtualization Operator with the CR to generate the virtual database and deploy it to OpenShift.

Additional resources

- Chapter 6, *Running the data virtualization operator to deploy a virtual database* .

CHAPTER 4. CREATING A VIRTUAL DATABASE AS A MAVEN ARTIFACT

You can use a Teiid Maven plugin to convert a DDL file into a Maven artifact. You define the structure of the virtual database in a DDL file and use the file to generate an artifact to deploy to a Maven repository. The Data Virtualization Operator can then deploy the artifact from the Maven repository to an OpenShift project.

This is an advanced method that provides a high level of flexibility and is suitable for complex projects. Using this method, you can create multi-module Maven projects in which you import one or more other virtual databases and incorporate them into your design.

You specify use of the Teiid plugin in your **pom.xml** file. You can also define other Maven dependencies in the **pom.xml** file. When you run the build, the plugin reads the file and resolves its contents.

Advantages of creating a virtual database as a Maven artifact

- Flexible, clean separation between the DDL code that represents the virtual database and other configuration settings.
- Enables easy deployment into multiple environments.
- Provides for versioning at the virtual database level.
- Enables a virtual database to be shared across projects and teams in a consistent way.
- Supports continuous integration and continuous delivery (CI/CD) workflows.

Disadvantages of creating a virtual database as a Maven artifact

- Requires a working knowledge of Maven.

Prerequisites

- You have a compatible data source and the OpenShift cluster can access it.
- You know how to create a **pom.xml** file to specify the dependencies that are required to build your virtual database.
- You have information about the connection settings for your data sources, including login credentials.
- The Data Virtualization Operator has access to the Maven repositories that contain build dependencies for the virtual database.
- You have Maven 3.2 or later installed.

Procedure

1. From a text editor, create a POM file to define build dependencies. For example,

```
<?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. Create a Maven project to import the virtual database definition from a DDL file. For example:

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

3. If you do not already have one, create a DDL file to specify the structure of the virtual database, and save it with a **.ddl** extension to the **/src/main/vdb** directory of your project. For example **vdb.ddl**
   The following example shows a sample DDL file for a virtual database that uses a postgreSQL data source:

   Example: 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;
```

For information about how to use DDL to define a virtual database, see *DDL metadata for schema objects* in the Data Virtualization Reference. Defining the complete DDL is beyond the scope of this document.

4. Build the virtual database artifact. Open a terminal window to the root folder of your Maven project, and type the following command:

```
mvn clean install
```

The command generates a **${project.name}-$2020.Q1.vdb** file in your target repository.

5. Deploy the artifact to a remote repository by typing the following command:

```
mvn clean install deploy
```

After the virtual database artifact is available in a Maven repository, you can use a YAML-based custom resource to deploy the virtual database to OpenShift. For information about using YAML to create a custom resource for deploying virtual database Maven artifacts, see Section 4.1, “Creating a custom resource (CR) to deploy a Maven artifact”.

For information about using the Data Virtualization Operator to deploy a virtual database, see Chapter 6, *Running the data virtualization operator to deploy a virtual database* .

4.1. CREATING A CUSTOM RESOURCE (CR) TO DEPLOY A MAVEN ARTIFACT

Before you can deploy a virtualization that you create as a Maven artifact, you must create a CR that defines the location of the Maven repository. When you are ready to deploy the virtualization, you provide this CR to the Data Virtualization Operator.

Prerequisites

- You created a virtualization according to the instructions in Chapter 4, *Creating a virtual database as a Maven artifact*.
- You deployed the virtualization to a Maven repository that the Data Virtualization Operator can access.
- You have the login credentials to access the data source.
- You are familiar with the creation of custom resource files in YAML format.

Procedure

1. Open a text editor, create a file with the name of the virtualization, and save it with the extension **.yaml**, for example, **dv-customer.yaml**.

2. Add information to define the custom resource kind, name, and source. The following annotated example provides guidance on the contents to include in the CR:

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) Specifies the credentials for signing in to the data source. Although this example shows credentials that are defined within the CR, in production use, use secrets to specify credentials, rather than exposing them in plain text. For information about adding credentials to secrets, see xref:

(2) Specifies the URL for connecting to the data source.

(3) Specifies the Maven location of the virtual database by providing the groupId, artifactId, and version (GAV) coordinates.

(4) If you are using a private Maven repository, specify its URL.

After you create the CR YAML file, you can run the Data Virtualization Operator to deploy the virtual database to OpenShift.

Run the Data Virtualization Operator with the CR to generate the virtual database and deploy it to OpenShift.

Additional resources

- Chapter 6, *Running the data virtualization operator to deploy a virtual database* .

CHAPTER 5. CREATING A VIRTUAL DATABASE AS A FAT JAR

You can use the Teiid Springboot starter to create a virtualization file as a Fat JAR. You can then publish the JAR to a Maven repository and use a YAML-based custom resource to deploy the virtual database to OpenShift. For more information about the Teiid Spring Boot starter, see https://github.com/teiid/teiid-spring-boot.

The Spring Boot Maven plugin creates a self-contained Uber JAR or fat JAR that includes all of the application code and dependencies in a single JAR file.

You define the virtual database in the resource files for the project (for example, the DDL file and **application.properties**), and specify the dependencies that are required to build the virtual database as a Spring Boot Java executable in the **pom.xml** file. When you run the build, Maven reads the **pom.xml** file and resolves its contents to incorporate external resources into the build.

When you build the project, it creates a virtual database as a Spring Boot Java executable. You can then test the resulting executable locally.

After local testing is complete, you can deploy the JAR file to a Maven repository. Then after your FAT JAR is available in the Maven repository, you can use a YAML based custom resource similar to deploy the virtual database to OpenShift.

Advantages of creating a virtual database as a fat JAR

- Establishes a clean separation between the DDL code that represents the virtual database and the configuration.
- Provides for local testing of the virtualization outside of OpenShift. Of course, caching, authentication, and other capabilities that depend on the OpenShift environment do not work locally.
- Supports extensions such as user-defined functions (UDFs), custom translators, and so forth as part of the project and they will be incorporated into the runtime virtual database automatically.
- Suitable for deployment into multiple environments.
- Versioning is done at the level of the overall project.

Disadvantages of creating a virtual database as a fat JAR

- Requires a working knowledge of Java, Maven, Teiid Spring Boot starters, Spring, and Teiid.

Prerequisites

- You have a working knowledge of Java development, Maven, Teiid Spring Boot starters, Spring, and Teiid.
- You have Maven 3.2 or later installed.
- You have JDK 11 (Java Platform, Standard Edition 11 Development Kit) or later installed.
- You have a compatible data source and the OpenShift cluster can access it.
- You have a **pom.xml** file that specifies the dependencies that are required to build your virtual database.

- If the driver for your data source is not available from the public Maven repository, you have downloaded the driver and deployed it to your local Maven repository.
- The Data Virtualization operator has access to the Maven repositories that contain build dependencies for the virtual database.
- You have a DDL file for the virtual database that you want to create, or you know how to write SQL code and create DDL files.

Procedure

1. Create a Java Maven project with the following directory structure for your virtual database:

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

1. In the **pom.xml**, define the repository locations, drivers, and user credentials that are required to build your virtual database.
2. In the application library of the virtual database project, create a Java application file, **Application.java**
3. In the same directory, add a **Datasources.java** class file, and add a bean method for each data source that you want your virtual database to connect to. For an example of a **Datasources.java** file that is designed to work with a postgreSQL database, see Section 5.1, “Sample Datasources.java file”.
4. In **/src/main/resources**, add an **application.properties** file and connection properties for your data sources to it. For more information, see Section 5.2, “Specifying application properties”.
5. In **/resources/vdb.ddl**, add DDL statements to specify the structure of the virtual database, including any views. For example **vdb.ddl**
   The following example shows a sample DDL file for a virtual database that uses a postgreSQL data source:

   Example: 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;
```

For information about how to use DDL to define a virtual database, see *DDL metadata for schema objects* in the Data virtualization reference guide. Instructions for how to define the complete DDL for a virtual database is beyond the scope of this document.

6. Build the virtual database artifact. Open a terminal window and type the following command:

```
mvn clean install
```

The command generates a **${project.name}-$2020.Q1.vdb** file in your target repository.

7. Deploy the artifact to a remote repository by typing the following command:

```
mvn clean install deploy
```

After the virtual database artifact is available in a Maven repository, you can use a YAML-based custom resource to deploy the virtual database to OpenShift.

5.1. SAMPLE DATASOURCES.JAVA FILE

The **Datasources.java** file adds a class to represent a connection to a data source. The file also establishes a prefix in the **ConfigurationProperties** argument (**spring.datasource.sampledb**). This prefix must be used in the names of data source properties that you specify in the **application.properties** file.

You can define multiple data sources in **Datasources.java** by adding multiple classes, each with its own prefix designation. In each case you must add corresponding entries to the DDL file and to the **datasources** properties in the CR file.

To associate the Java bean with the data source that is defined in your DDL file, the bean name must be the same as the name of the **SERVER** and **resource-name** properties in the DDL file. For example, the following sample file establishes a connection to a PostgreSQL database called **sampledb**, which is the name that is assigned in the DDL file to the data source **SERVER** object and to its **resource-name** definition.

```
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. The prefix must match the prefix that you assign to properties that you define in the **application.properties** file.

2. The name **sampledb** in the prefix definition and in the method name must match the name in the **SERVER** and **resource-name** objects that are defined in the virtual database DDL file. The Spring Boot framework automatically associates the names of methods in the **Datasources.java** file with the names of data sources in the DDL file.

5.2. SPECIFYING APPLICATION PROPERTIES

When you create a virtualization as a fat JAR, you must supply an **application.properties** file. You can define some static properties for your virtual database application in an **application.properties** file in the **/src/main/resource** directory. Static properties are configuration settings that remain constant across different environments. After you deploy a virtual database on OpenShift, any modifications that you make to the **application.properties** file are not effective unless you rebuild and redeploy your virtual database.

You must prefix data source properties that you define in the **application.properties** file, with the configuration properties string that is specified in the **Datasources.java** file. The prefix establishes a connection between the properties and the Java class.

For example, if you establish the configuration properties prefix **spring.datasource.sampledb** in the **Datasources.java** file, then you must precede the names of the properties that you define in the **application.properties** file with that string, as in the following property definitions:

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

Prerequisites

- You have a **Datasources.java** file in your Java class folder that specifies an application prefix.

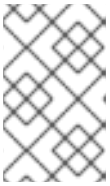
Procedure

1. Add an **application.properties** file to the **src/main/resources** folder of your Java project.

2. Within the file, define properties that are required to connect to your data source, such as authentication credentials.

NOTE

Properties that you do not define in the **application.properties** file, must be defined in the CR YAML file.



NOTE

If you define a property in application.properties and define a corresponding environment variables in the CR, the value in the CR takes precedence over the value that is set in the **application.properties** file.

For example:

```
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. The JDBC URL that the virtual database uses to connect to a local postgreSQL database as its data source.
2. The prefix that is used in each of these properties matches the prefix that is defined in the **Datasources.java** file.
3. The user name and password values listed here are displayed in plain text. To enable secure storage of authentication credentials, use environment variables in a CR file to reference the secret object that defines these values.
4. The driver that is required to connect to the data source. The driver that you reference here must be defined as a dependency in the **pom.xml** file. For an example of a **pom.xml** file for creating a virtual database as a fat JAR, see the **teiid/dv-customer-fatjar** repository.
5. The name of the data source.
6. Uncomment this statement to enable debug logging.

Additional resources

- Section 5.3, “Creating a CR to deploy a fat JAR”

5.3. CREATING A CR TO DEPLOY A FAT JAR

After you develop a virtual database from the teiid-springboot starter, you deploy the resulting JAR to Maven repository. You then create a YAML custom resource file for deploying the virtual database to OpenShift.

The CR file for deploying a virtual database created as a fat JAR resembles the CR that you use to deploy a virtual database that is created as a Maven artifact, as described in Section 4.1, "Creating a custom resource (CR) to deploy a Maven artifact". Only the Maven GAV coordinates differ. In this case, the CR provides the Maven coordinates of the JAR file.

Prerequisites

- You created a virtualization as a fat JAR, according to the instructions in Chapter 5, *Creating a virtual database as a fat JAR*.
- You deployed the virtualization to a Maven repository that the Data Virtualization Operator can access.
- You have the login credentials to access the data source.
- You are familiar with the creation of custom resource files in YAML format.

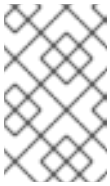
Procedure

1. . Open a text editor, create a file with the name of the virtualization, and save it with the extension **.yaml**, for example, **dv-customer.yaml**.
2. Add information to define the custom resource kind, name, and source.
   The following example shows a CR that is designed to deploy a virtual database that is created as a fat JAR:

```
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) Sample environment variables for a postgreSQL data source.

- In the example, values for data source credentials are defined in clear text. However, as mentioned in], specifying credentials directly in the CR is not secure. To protect credentials, reference them from an OpenShift secret. For more information, see xref:deploying-secrets[.



NOTE

If an environment variable that you define in the CR is also defined as a property in the **application.properties** file, the value in the CR takes precedence over the value that is set in the **application.properties** file.

2 The Maven coordinates of a fat JAR artifact that you deployed to a Maven repository in Chapter 5, *Creating a virtual database as a fat JAR* .

After you create the CR YAML file, you can run the Data Virtualization Operator to deploy the virtual database to OpenShift.

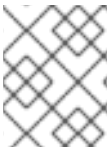
Additional resources

- Chapter 6, *Running the data virtualization operator to deploy a virtual database* .

CHAPTER 6. RUNNING THE DATA VIRTUALIZATION OPERATOR TO DEPLOY A VIRTUAL DATABASE

The data-virtualization operator helps to automate the configuration and deployment of virtual databases.

The Data Virtualization Operator processes a virtual database custom resource (CR) to deploy a virtual database object on OpenShift. By running the operator with different CRs, you can create virtual databases from a range of data sources.



NOTE

Virtual databases that you deploy to OpenShift in this Technology Preview are not available in Fuse Online.

6.1. INSTALLING THE DATA VIRTUALIZATION OPERATOR ON OPENSHIFT

Install the Data Virtualization Operator so that you can use it to deploy virtual database images to OpenShift from YAML-based custom resources (CRs).

You can install the data virtualization operator on OpenShift 3.11 or greater. On OpenShift 4.2 and later the Operator is available in the OperatorHub.

After you add the operator to your OpenShift cluster, you can use it to build and deploy virtual database images from a range of data sources.

Prerequisites

- You have cluster-admin access to an OpenShift 3.11 or 4.2 or greater cluster.
- You can use the **oc** command-line tool to connect to interact with your OpenShift 3.11 cluster, or you have access to the OpenShift 4.2 or greater web console.
- You have the 2020.Q1 release of Red Hat Integration.
- You have Developer access to an OpenShift server and you are familiar with using the OpenShift console and CLI.

Procedure

- Install the operator using one of the following methods, depending on the version of OpenShift that you are running.

Installing on OpenShift 3.11

1. From a terminal window, log in to the OpenShift cluster as a cluster administrator.

```
oc login
```

2. Create or open a project where you want to deploy a virtual database.
3. Type the following commands:

```
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) If you previously created a CRD in the cluster, the command returns an error, reporting that the CRD already exists. You can ignore the message.

4. Type the following commands to create a pull secret that you can use to access the Red Hat image registry:

```
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) Substitute the user name and password that you use to log in to the Red Hat Customer Portal.

If the command completes with no errors, the operator is deployed to your OpenShift instance within the current OpenShift project.

5. To enable the data virtualization operator to retrieve images from the Red Hat registry so that you can create virtual databases, link the secret that you created in Step 4 to the service account for the operator.

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

Installing on OpenShift 4.2 or greater

1. From a terminal window, type the following commands to log in to the OpenShift cluster and create a pull secret that you can use to access the Red Hat image registry:

```
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) Use your Red Hat Customer Portal login credentials.

2. Log in to the OpenShift web console as a cluster administrator.

3. From the OpenShift menu, expand **Operators** and click **OperatorHub**.

4. Click **Data Virtualization Operator 7.6.0 provided by Red Hat, Inc.**, and then click **Install**.

5. From the **Create Operator Subscription** page, verify that the selected namespace matches the name of the project where you want to install the operator, and then click **Subscribe**. The **Installed Operators** page lists the **Data Virtualization Operator** and reports the status of the installation.

6. From the OpenShift menu, expand **Workloads** and click **Pods** to check the status of the operator pod. After a few minutes, the pod for the operator service begins to run.

7. To enable the data virtualization operator to retrieve images from the Red Hat registry so that you can create virtual databases, link the secret that you created in Step 1 to the service account for the operator.

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

Additional resources

- Section 6.2, “Deploying virtual databases” .

6.2. DEPLOYING VIRTUAL DATABASES

After you create a virtual database and its corresponding CR file, run the Data Virtualization Operator to deploy the database to Openshift.

Prerequisites

- A cluster administrator added the Data Virtualization Operator to the OpenShift cluster where you want to deploy the virtual database.
- You have access to an OpenShift cluster in which the Data Virtualization Operator is installed.
- You have a CR in YAML format that provides information about how to configure and deploy the virtual database.
- The Operator has access to the Maven repositories that contain the dependencies that the build requires.
- OpenShift can access the data source that is referenced in the CR.

Procedure

1. From a terminal window, log in to OpenShift and open the project where you want to create the virtual database.
2. On your computer, change to the directory that contains the **.yaml** file that contains the CR.
3. Type the following command to run the operator to create the virtual database:

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

Replace **<cr_filename.yaml>** with the name of the CR file for your data source. For example,

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

After the deployment completes, a virtual database service is added to the OpenShift cluster. The name of the service matches the name that is specified in the custom resource.

4. Type the following command to verify that the virtual database is created:

```
oc get vdbs
```

OpenShift returns the list of virtual databases in the project. To see whether a particular virtualization is available, type the following command:

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

The deployed service supports connections from the following clients:

- JDBC clients through port 31000.
- postgreSQL clients, including ODBC clients, through port 5432.
- OData clients, through an HTTP endpoint and route.

CHAPTER 7. SECURING DATA

To prevent unauthorized access to data, you can implement the following measures:

- Configure integration with Red Hat Single Sign-On in OpenShift to enable OpenID-Connect authentication and OAuth2 authorization.
- Apply role-based access controls to your virtual database.
- Configure 3Scale to secure OData API endpoints.
- Encrypt communications between database clients (ODBC and JDBC) and the virtual database.

7.1. SECURING ODATA APIS FOR A VIRTUAL DATABASE

You can integrate data virtualization with Red Hat Single Sign-On and Red Hat 3scale API Management to apply advanced authorization and authentication controls to the OData endpoints for your virtual database services.

The Red Hat Single Sign-On technology uses OpenID-Connect as the authentication mechanism to secure the API, and uses OAuth2 as the authorization mechanism. You can integrate data virtualization with Red Hat Single Sign-On alone, or along with 3scale.

By default, after you create a virtual database, the OData interface to it is discoverable by 3scale, as long as the 3scale system is defined to same cluster and namespace. By securing access to OData APIs through Red Hat Single Sign-On, you can define user roles and implement role-based access to the API endpoints. After you complete the configuration, you can control access in the virtual database at the level of the view, column, or data source. Only authorized users can access the API endpoint, and each user is permitted a level of access that is appropriate to their role (role-based access). By using 3scale as a gateway to your API, you can take advantage of 3scale's API management features, allowing you to tie API usage to authorized accounts for tracking and billing.

When a user logs in, 3scale negotiates authentication with the Red Hat Single Sign-On package. If the authentication succeeds, 3scale passes a security token to the OData API for verification. The OData API then reads permissions from the token and applies them to the data roles that are defined for the virtual database.

Prerequisites

- Red Hat Single Sign-On is running in the OpenShift cluster. For more information about deploying Red Hat Single Sign-On, see the Red Hat Single Sign-On for OpenShift documentation.
- You have Red Hat 3scale API Management installed in the OpenShift cluster that hosts your virtual database.
- You have configured integration between 3scale and Red Hat Single Sign-On. For more information, see Configuring Red Hat Single Sign-On integration in Using the Developer Portal.
  - You have assigned the *realm-management* and *manage-clients* roles.
  - You created API users and specified credentials.
  - You configured 3scale to use OpenID-Connect as the authentication mechanism and OAuth2 as the authorization mechanism.

7.1.1. Configuring Red Hat Single Sign-On to secure OData

You must add configuration settings in Red Hat Single Sign-On to enable integration with data virtualization.

Prerequisites

- Red Hat Single Sign-On is running in the OpenShift cluster. For information about deploying Red Hat Single Sign-On, see the link:Red Hat Single Sign-On for OpenShift[Red Hat Single Sign-On] documentation.
- You run the Data Virtualization Operator to create a virtual database in the cluster where Red Hat Single Sign-On is running.

Procedure

1. From a browser, log in to the Red Hat Single Sign-On Admin Console.
2. Create a realm for your data virtualization service.
   a. From the menu for the master realm, hover over **Master** and then click **Add realm**.
   b. Type a name for the realm, such as **datavirt**, and then click **Create**.
3. Add roles.
   a. From the menu, click **Roles**.
   b. Click **Add Role**.
   c. Type a name for the role, for example **ReadRole**, and then click **Save**.
   d. Create other roles as needed to map to the roles in your organization's LDAP or Active Directory. For information about federating user data from external identity providers, see the Server Administration Guide.
4. Add users.
   a. From the menu, click **Users**, and then click **Add user**.
   b. On the **Add user** form, type a user name, for example, **user**, specify other user properties that you want to assign, and then click **Save**.
   Only the user field is mandatory.
   c. From the details page for the user, click the **Credentials** tab.
   d. Type and confirm a password for the user, click **Reset Password**, and then click **Change password** when prompted.
5. Assign roles to the user.
   a. Click the **Role Mappings** tab.
   b. In the **Available Roles** field, click **ReadRole** and then click **Add selected**.
6. Create a second user called **developer**, and assign a password and roles to the user.
7. Create a data virtualization client entry.

The client entry represents the data virtualization service as an SSO client application. .. From the menu, click **Clients**. .. Click **Create** to open the **Add Client** page. .. In the **Client ID** field, type a name for the client, for example, **dv-client**. .. In the **Client Protocol** field, choose **openid-connect**. .. Leave the **Root URL** field blank, and click **Save**.

You are now ready to add SSO properties to the CR for the data virtualization service.

7.1.2. Adding SSO properties to the custom resource file

After you configure Red Hat Single Sign-On to secure the OData endpoints for a virtual database, you must configure the virtual database to integrate with Red Hat Single Sign-On. To configure the virtual database to use SSO, you add SSO properties to the CR that you used when you first deployed the service (for example, **dv-customer.yaml**). You add the properties as environment variables. The SSO configuration takes effect after you redeploy the virtual database.

In this procedure you add the following Red Hat Single Sign-On properties to the CR:

Realm (KEYCLOAK_REALM)

The name of the realm that you created in Red Hat Single Sign-On for your virtual database.

Authentication server URL (KEYCLOAK_AUTH_SERVER_URL)

The base URL of the Red Hat Single Sign-On server. It is usually of the form https://host:port/auth.

Resource name(KEYCLOAK_RESOURCE)

The name of the client that you create in Red Hat Single Sign-On for the data virtualization service.

SSL requirement (KEYCLOAK_SSL_REQUIRED)

Specifies whether requests to the realm require SSL/TLS. You can require SSL/TLS for all requests, external requests only, or none.

Access type (KEYCLOAK_PUBLIC_CLIENT)

The OAuth application type for the client. Public access type is for client-side clients that sign in from a browser.

Prerequisites

- You ran the Data Virtualization Operator to create a virtual database.
- Red Hat Single Sign-On is running in the cluster where the virtual database is deployed.
- You have the CR YAML file, for example, **dv-customer.yaml** that you used to deploy the virtual database.
- You have have administrator access to the Red Hat Single Sign-On Admin Console.

Procedure

1. Log in to the Red Hat Single Sign-On Admin Console to find the values for the required authentication properties.
2. In a text editor, open the CR YAML file that you used to deploy your virtual database, and define authentication environment variables that are based on the values of your Red Hat Single Sign-On properties.
   For example:

```
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. Declare a build source dependency for the following Maven artifact for securing data virtualizations: **org.teiid:spring-keycloak**
   For example:

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

4. Save the CR.

You are now ready to define data roles in the DDL for the virtual database.

7.1.3. Defining data roles in the virtual database DDL

After you configure Red Hat Single Sign-On to integrate with data virtualization, to complete the required configuration changes, define role-based access policies in the DDL for the virtual database. Depending on how you deployed the virtual database, the DDL might be embedded in the CR file, or exist as a separate file.

You add the following information to the DDL file:

- The name of the role. Roles that you define in the DDL must map to roles that you created earlier in Red Hat Single Sign-On.

  TIP

  For the sake of clarity, match the role names in the DDL file to the role names that you specified in Red Hat Single Sign-On. Consistent naming makes it easier to correlate how the roles that you define in each location relate to each other.

- The database access to allow to users who are granted the specified role. For example, SELECT permissions on a particular table view.

Prerequisites

- You configured Red Hat Single Sign-On to work with data virtualization as described in Section 7.1.1, “Configuring Red Hat Single Sign-On to secure OData” .
- You added SSO properties to the CR file for the virtual database, as described in .

Procedure

1. In a text editor, open the file that contains the DDL description that you used to deploy the virtual database.

2. Insert statements to add any roles that you defined for virtual database users in Red Hat Single Sign-On. For example, to add a role with the name **ReadRole** add the following statement to the DDL:

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

   Add separate **CREATE ROLE** statements for each role that you want to implement for the virtual database.

3. Insert statements that specify the level of access that users with the role have to database objects. For example,

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

   Add separate **GRANT** statements for each role that you want to implement for the virtual database.

4. Save and close the CR or DDL file.
   You are now ready to redeploy the virtual database. For information about how to run the Data Virtualization Operator to deploy the virtual database, see Chapter 6, *Running the data virtualization operator to deploy a virtual database*.

   After you redeploy the virtual database, add a redirect URL in the Red Hat Single Sign-On Admin Console. For more information, see Section 7.1.4, "Adding a redirect URI for the data virtualization client in the Red Hat Single Sign-On Admin Console".

7.1.4. Adding a redirect URI for the data virtualization client in the Red Hat Single Sign-On Admin Console

After you enable SSO for your virtual database and redeploy it, specify a redirect URI for the data virtualization client that you created in Section 7.1.1, "Configuring Red Hat Single Sign-On to secure OData".

Redirect URIs, or callback URLs are required for *public* clients, such as OData clients that use OpenID Connect to authenticate, and communicate with an identity provider through the redirect mechanism.

For more information about adding redirect URIs for OIDC clients, see the NameOfRHSSOServerAdmin.

Prerequisites

- You enabled SSO for a virtual database and used the Data Virtualization Operator to redeploy it.
- You have administrator access to the Red Hat Single Sign-On Admin Console.

Procedure

1. From a browser, sign in to the Red Hat Single Sign-On Admin Console.

2. From the security realm where you created the client for the data virtualization service, click **Clients** in the menu, and then click the ID of the data virtualization client that you created previously (for example, **dv-client**).

3. In the **Valid Redirect URIs** field, type the root URL for the OData service and append an asterisk to it. For example, **http://datavirt.odata.example.com/***

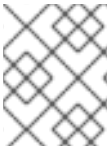
4. Test whether Red Hat Single Sign-On intercepts calls to the OData API.

   a. From a browser, type the address of an OData endpoint, for example:

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

   A login page prompts you to provide credentials.

5. Sign in with the credentials of an authorized user.
   Your view of the data depends on the role of the account that you use to sign in.



NOTE

Some endpoints, such as **odata/$metadata** are excluded from security filtering so that they can be discovered by other services.

7.2. CUSTOM CERTIFICATES FOR ENDPOINT TRAFFIC ENCRYPTION

Data Virtualization uses TLS certificates to encrypt network traffic between JDBC and ODBC database clients and a virtual database service. You can supply your own custom certificate, or use a service certificate that is generated by the OpenShift certificate authority. If you do not supply a custom TLS certificate, the Data Virtualization Operator generates a service certificate automatically.

Service certificates provide for encrypted communications for internal and external clients alike. However, only internal clients, that is, clients that are deployed in the same OpenShift cluster, can validate the authenticity of a service certificate.

OpenShift service certificates have the following characteristics:

- Consist of a public key certificate (**tls.crt**) and a private key (**tls.key**) in PEM base-64-encoded format.
- Stored in an encryption secret in the OpenShift pod.
- Signed by the OpenShift CA.
- Valid for one year.
- Replaced automatically before expiration.
- Can be validated by internal clients only.

External clients do not recognize validity of certificates generated by the OpenShift certificate authority. To enable external clients to validate certificates, you must provide custom certificates from trusted, third-party certificate authorities (CAs). Such certificates are universally recognized, and can be verified by any client. To add a custom certificate to a virtual database, you supply information about the certificate in an encryption secret that you deploy to OpenShift before you run the Data Virtualization Operator to create the service.

When you deploy the encryption secret to OpenShift, it becomes available to the Data Virtualization Operator when it creates a virtual database. The Operator detects the secret with the name that matches the name of the virtual database in the CR, and it automatically configures the service to use the specified certificate to encrypt connections with database clients.

7.3. USING CUSTOM TLS CERTIFICATES TO ENCRYPT COMMUNICATIONS BETWEEN DATABASE CLIENTS AND ENDPOINTS

You can add a custom TLS certificate to OpenShift to encrypt communications between JDBC or ODBC clients and a virtual database service. Because custom certificate are issued by trusted third-party certificate authorities (CA), clients can authenticate the CA signature on the certificate.

To configure an OpenShift pod to use a custom certificate to encrypt traffic, you add the certificate details to an OpenShift secret and deploy the secret to the namespace where you want to create the virtual database. You must create the secret before you create the service.

Prerequisites

- You have a TLS certificate from a trusted, third-party CA.
- You have Developer or Administrator access to the OpenShift project where you want to create the secret and virtual database.

Procedure

1. Create a YAML file to define a secret of type **kubernetes.io/tls**, and include the following information:
   - The public and private keys of the TLS key pair.
   - The name of the virtual database that you want to create.
   - The OpenShift namespace in which you want to create the virtual database.
     For example:

```
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. The name of the secret. The secret name must match the name of the virtual database object in the CR YAML file that the Data Virtualization Operator uses to create a virtual database, for example, **dv-customer**.
2. The OpenShift namespace in which the virtual database service is deployed, for example, **myproject**.
3. The **data** value is made up of the contents of the TLS public key certificate (**tls.crt**), and the private encryption key (**tls.key**) in base64-encoded PEM format.

2. Save the file as **tls_secret.yaml**.
3. Open a terminal window, sign in to the OpenShift project where you want to add the secret, and then type the following command:

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

4. After you deploy the TLS secret to OpenShift, run the Data Virtualization Operator to create a virtual database with the name that is specified in the secret.
When the Operator creates the virtual database, it matches the name in the secret to the name specified for the service in the CR. The Operator then configures the service to use the secret to encrypt client communications with the service.

7.4. USING SECRETS TO STORE DATA SOURCE CREDENTIALS

Create and deploy secret objects to store values for your environment variables.

Although secrets exist primarily to protect sensitive data by obscuring the value of a property, you can use them to store the value of any property.

Prerequisites

- You have the login credentials and other information that are required to access the data source.

Procedure

1. Create a secrets file to contain the credentials for your data source, and save it locally as a YAML file. For example,

   Sample secrets.yml file

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

2. Deploy the secret object on OpenShift.

a. Log in to OpenShift, and open the project that you want to use for your virtual database. For example,
oc login --token=<token> --server=https://<server>oc project <projectName>

b. Run the following command to deploy the secret file:
oc create -f ./secret.yaml

3. Set an environment variable to retrieve its value from the secret.

a. In the environment variable, use the format **valueFrom:/secretKeyRef** to specify that the variable retrieves it value from a key in the secret that you created in Step 1.
For example, in the following excerpt, the **SPRING_DATASOURCE_SAMPLEDB_PASSWORD** retrieves its value from a reference to the **database-password** key of the **postgresql** secret:

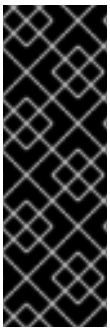
```
- name: SPRING_DATASOURCE_SAMPLEDB_PASSWORD
 valueFrom:
   secretKeyRef:
     name: postgresql
     key: database-password
```

Additional resources

- For more information about how to use secrets on OpenShift, see Providing sensitive data to pods in the OpenShift documentation.

CHAPTER 8. CREATING AND WORKING WITH VIRTUAL DATABASES IN FUSE ONLINE

In Fuse Online, you can create a virtual database that integrates data from multiple data sources that you choose. After you deploy the resulting virtual database service, application developers and other database users can connect to as if it were a single physical database.



IMPORTANT

Data virtualization in Fuse Online is a Technology Preview feature only. Technology Preview features are not supported with Red Hat production service level agreements (SLAs) and might not be functionally complete. Red Hat does not recommend using them in production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process. For more information about the support scope of Red Hat Technology Preview features, see https://access.redhat.com/support/offerings/techpreview/.

After you create a virtual database in Fuse Online, you can use Fuse Online tools to:

- Add or remove data sources.
- Add or edit views of data from different tables or sources.
- Submit SQL queries to test that views return the expected results.
- Modify the schema that defines the virtual database.
- Publish the virtual database to make it available on OpenShift.
- Delete the virtual database.

Prerequisites

- You have the 2020.Q1 release of Red Hat Integration, and you are running 7.6.
- The data virtualization UI for 7.6 was enabled during installation. For more information, see Enabling data virtualization in Fuse Online on OCP in the Installing and Operating Fuse Online on OpenShift Container Platform.

8.1. CREATING VIRTUAL DATABASES IN FUSE ONLINE

In Fuse Online, you can create virtual databases that import views from applications or services that are available from the **Connections** page.

For each virtual database that you create, you must import data sources, and select the tables from each data source that you want to include. The views in the resulting virtual database map directly to the database tables that you import. After the initial creation, you can add views to a virtual database that join data from more than one table.

NOTE

In this Technology Preview, you can create virtual databases in Fuse Online only from relational databases, MongoDB, and Salesforce.

Prerequisites

- Your Fuse Online environment has a connection to one or more of the following data sources:
  - Relational database, such as postgreSQL or MySQL.
  - MongoDB database
  - Salesforce database

Procedure

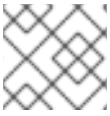
1. From the navigation sidebar in Fuse Online, click **Data**.
2. Click **Create Data Virtualization**.
3. On the **Create New Data Virtualization** page, type a name for the virtual database and click **Create**.
   - Provide a name that informs people about the database contents or purpose, and that is short enough for application developers and SQL users to easily insert in their code.
   - Names can include only alphanumeric ([a-z]|[A-Z], [0-9]), and hyphen (-) characters.
4. On the page for your virtualization, click **Import Data Source**.
5. On the **Import Data Source** page, click the tile for an active data source, and then click **Next**.
6. From the list of tables that are available, select one or more tables to include in your virtual database and then click **Done**.
   A confirmation message reports when the import completes. The **Views** tab for the draft virtualization lists a view for each table that you imported.

   You can now edit the existing views, create another view, or publish the virtual database to make it available for use.

8.2. ADDING A VIEW TO A VIRTUAL DATABASE IN FUSE ONLINE

Add a view to a virtual database to provide a view of the data in a new table.

After you first create a virtual database, it contains only the views that you imported from the initial data source. Add views to the virtual database if you want to incorporate data from other tables. You can add views based on tables in the original data source, or from other data sources.



NOTE

In this Technology Preview release, you can only add one table to each view.

Prerequisites

- The virtual database that you want to add a view to is available in Fuse Online in a *Draft* or *Published* state. You cannot use Fuse Online to add views to virtual databases that were created outside of Fuse Online.
- A Fuse Online connection exists to the data source that contains the table that you want integrate.

- You know the name of the table that you want to use in the view.

Procedure

1. From the navigation sidebar in Fuse Online, click **Data**.
2. From the list on the **Data Virtualizations** page, find the virtual database that you want to modify and click **Edit**.
3. Click **Create a View**.
4. Expand a data source to view the tables that it contains.
5. Select the table that you want to add to the virtual database, and then click **Next**.
6. On the **Create a View** page, type a name in the **View Name** field, and then click **Done**.
   The **View Editor** displays the SQL for the view that you created. The **Preview** panel displays the data in the view.
7. If no data displays, click **Refresh**.
8. Click **Done** to close the view.
   If the virtual database was previously published, you must republish it to make the new view available.

Additional resources

- Experienced SQL programmers can also add views by directly editing the default SQL statements for the virtual database. For more information, see Section 8.3, "Using the View Editor in Fuse Online to modify the DDL that defines a virtual database".
- Section 8.5, "Publishing virtual databases in Fuse Online to make them available for access"
- Section 8.4, "Previewing a virtual database in Fuse Online by submitting SQL test queries"
- Section 8.3, "Using the View Editor in Fuse Online to modify the DDL that defines a virtual database"

8.3. USING THE VIEW EDITOR IN FUSE ONLINE TO MODIFY THE DDL THAT DEFINES A VIRTUAL DATABASE

The process of creating a virtual database in Fuse Online is designed to automate many tasks and hide the complexities of the underlying SQL code.

When you create a view for a virtual database, Fuse Online automatically generates the data definition language (DDL) that defines the view. The DDL is a set of SQL statements that describe the view's schema, tables, columns, and other fields.

Fuse Online provides tools to add basic views for a virtual database, but if you know SQL and you want greater control in designing a view, you can directly edit the DDL for the view. In Fuse Online, developers can use the embedded View Editor to modify these SQL statements. To assist you, this SQL editor includes a code-completion feature that provides a list of SQL keywords.

After you save your changes, a built-in validation tool runs to ensure that the SQL code does not contain syntax errors.

Prerequisites

- You have experience using a data definition language (DDL) that is based on the SQL-MED specification to define database structures and to integrate externally stored data.

Procedure

1. From the navigation sidebar in Fuse Online, click **Data**.
2. On the **Data Virtualizations** page, find the virtual database that you want to modify and click **Edit**.
3. In the **Views** tab, find the view that you want to edit, and then click **Edit**.
4. Update the SQL as needed. As you edit, press Ctrl+Space to open the code completion tool.
5. After you complete your changes, click **Save**.
   Fuse Online validates the SQL and returns an error if the view contains invalid code.

   After the SQL validates, the preview panel shows the result of the updates that you made to the view. The preview displays the first fifteen rows of the results set.
6. Click **Done** to close the **View Editor** and return to the list of views.
   If the virtual database was previously published, you must republish it to put your changes into effect.

Additional resources

- Section 8.5, "Publishing virtual databases in Fuse Online to make them available for access"
- For more information about using SQL in data virtualization DDL files, see the Teiid Reference Guide.
- You can modify the results set by altering the default query to specify different row limits or row offsets. For more information, see Section 8.4, "Previewing a virtual database in Fuse Online by submitting SQL test queries"

8.4. PREVIEWING A VIRTUAL DATABASE IN FUSE ONLINE BY SUBMITTING SQL TEST QUERIES

Before you publish a virtual database and make it available to applications, you can run test queries against its views to verify that it returns the information that you expect.

Although the default preview shows you the first 15 results returned when a SQL **SELECT * FROM** statement is submitted to a virtual database view, you can use the embedded SQL client in Fuse Online to send modified test queries to your views. You can adjust the default results set by specifying the row limits and row offsets.

If the view that you query originates from a non-SQL data source, the data virtualization engine converts the SQL query into a format that the data source can interpret.

Prerequisites

- You have a valid virtual database that was created in Fuse Online.

Procedure

Procedure

1. From the navigation sidebar in Fuse Online, click **Data**.
2. On the **Data Virtualizations** page, click **Edit** in the entry for the virtual database that contains the view that you want to test.
3. Click the **SQL Client** tab.
4. From the **View** field, select the view that you want to test.
5. In the **Row Limit** field, specify the number of rows to display.
6. In the **Row Offset** field, specify the number of rows to skip.
7. Click **Submit**. The **Query Results** table displays the result set.

8.5. PUBLISHING VIRTUAL DATABASES IN FUSE ONLINE TO MAKE THEM AVAILABLE FOR ACCESS

After you define a virtual database in Fuse Online, you must publish it to make it available for users and applications to access.

Publishing a virtual database builds the schema definition that you implemented by importing data sources and views into a runtime image. Fuse Online deploys the runtime image to OpenShift as a virtual database container image that you can scale independently.

After you publish the virtual database, it becomes available as a service and is represented on the Fuse Online **Connections** page. The service behaves like any relational database, and clients can connect to it over standard interfaces. It can be incorporated into Fuse Online integration workflows, and it is available to JDBC and OData clients.

Prerequisites

- You created a virtual database in Fuse Online.
- You added any views that you want to the virtual database.

Procedure

1. From the navigation sidebar in Fuse Online, click **Data**.
2. On the **Data Virtualizations** page, find a virtual database that you want to publish, and from the overflow menu, click **Publish**.
   A confirmation message notifies you that the virtual database was submitted for publishing, and a progress bar reports the status of the process.
   - If the publishing process succeeds, Fuse Online makes the following updates:
     - The status label of the virtual database entry on the **Data virtualizations** page changes from **Draft** to **Published**.
     - The virtual database entry displays a URL link to the OData endpoint for the virtual database.
     - The virtual database service is added to the **Connections** page, and a JDBC connection to it is created.

You can verify the JDBC URL by opening the entry for the virtual database service from the **Connections** page.

- If the publishing process fails, the entry is flagged with the label **Error**.

8.6. DELETING A VIRTUAL DATABASE IN FUSE ONLINE

You can permanently delete virtual databases that you create in Fuse Online. You can delete virtual databases whether they are published or in draft.

The data sources that a virtual database consumes are not affected by the deletion. Connections between Fuse Online and the data sources remain in place.

Prerequisites

- You have a virtual database that was created in Fuse Online and you want to remove it.

Procedure

1. From the navigation sidebar in Fuse Online, click **Data**.
2. On the **Data Virtualizations** page, click the overflow menu for the virtual database that you want to delete, and then click **Delete**.
3. When prompted, click **Delete** to confirm that you want to delete the virtual database.
   A confirmation message reports when the virtualization is deleted.

CHAPTER 9. VIRTUAL DATABASE MONITORING

Prometheus is an open-source systems and service monitoring and alerting toolkit that you can use to monitor services deployed in your Red Hat OpenShift environment. Prometheus collects and stores metrics from configured services at given intervals, evaluates rule expressions, displays the results, and can trigger alerts if a specified condition becomes true.

IMPORTANT

Red Hat support for Prometheus is limited to the setup and configuration recommendations provided in Red Hat product documentation.

Prometheus communicates with the data virtualization service to retrieve metrics and data. Users can query the data, or use a visualization took such as Grafana to view trends in a dashboard.

To enable monitoring of an OpenShift service, the service must expose an endpoint to Prometheus. This endpoint is an HTTP interface that provides a list of metrics and the current values of the metrics. When you use the Data Virtualization Operator to create a virtual database, the data virtualization service automatically exposes an HTTP endpoint to Prometheus. Prometheus periodically scrapes each target-defined endpoint and writes the collected data to its database.

After Prometheus is deployed to an OpenShift cluster, the metrics for any virtualization that you deploy to the same cluster are exposed to the Prometheus service automatically.

For more information about using Prometheus to monitor services in Red Hat Integration, see Monitoring Red Hat Integration .

CHAPTER 10. MIGRATING LEGACY VIRTUAL DATABASE FILES TO DDL FORMAT

The data virtualization Technology Preview requires that you define the structure of virtual databases in SQL-MED DDL (data definition language) format. By contrast, the structure of legacy Teiid virtual databases, such as those that run on Wildfly, or on the Red Hat JBoss Data Virtualization offering, are defined by using files that are in **.xml** or **.vdb** format.

You can reuse the virtual database designs that you developed for a legacy deployment, but you must first update the format of the files. A migration tool is available to convert your files. After your convert the files you can rebuild the virtual databases as container images and deploy them to OpenShift.

Migration considerations

The following features that were supported in virtual databases in JBoss Data Virtualization and Teiid might be limited or unavailable in this Technology Preview release of data virtualization:

Data source compatibility

You cannot use all data sources with this release. For a list of compatible data sources, see Section 2.1, “Compatible data sources”.

Internal distributed materialization

Not available.

Resultset caching

Not available.

Use of runtime metadata to import other virtual databases

DDL must be used to specify metadata for virtual databases.

Runtime manipulation of multisource vdb sources

Not available.

You can use the migration utility in the following two ways:

To validate a VDB file only

Use this method to check whether the utility can a successfully convert a VDB file. The utility converts the VDB file and reports validation errors to the terminal. If there are no validation errors, the utility displays the resulting DDL, but it does not save the converted DDL to a file.

To validate and a VDB file and save it to a DDL file

The file is saved only if there are no validation errors.

The migration tool works on XML files only. Files with a **.vdb** file extension are file archives that contain multiple folders. If you have legacy files in **.vdb** format, use Teiid Designer to export the files to XML format, and then run the migration tool to convert the resulting XML files.

Prerequisites

- You have a legacy virtual database file in **.xml** format.
- You download the **settings.xml** file from the Teiid OpenShift repository. Maven uses the information in the file to run the migration tool.

10.1. VALIDATING A LEGACY VIRTUAL DATABASE XML FILE AND VIEWING IT IN DDL FORMAT

You can run a test conversion on a legacy virtual database to check for validation errors and view the resulting DDL file. When you run the migration tool in this way, the converted DDL file is not saved.

Procedure

1. Open the directory that contains the **settings.xml** file that you downloaded from the Teiid OpenShift repository, and type the following command:

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

For example:

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

The migration tool checks the specified **.xml** file, and reports any validation errors. If there are no validation errors, the migration tool displays a **.ddl** version of the virtual database on the screen.

10.2. CONVERTING A LEGACY VIRTUAL DATABASE XML FILE AND SAVING IT AS A DDL FILE

You can run the migration tool to convert a legacy virtual database file to **.ddl** format, and then save the **.ddl** file to a specified directory. The migration tool checks the **.xml** file that you provide for validation errors. If there are no validation errors, the migration tool converts the file to **.ddl** format and saves it to the file name and directory that you specify.

Procedure

- Open the directory that contains the **settings.xml** file that you downloaded from the Teiid OpenShift repository, and type the following command:

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

For example:

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