Red Hat JBoss Data Virtualization 6.4

Development Guidelines and Recommended Practices Guide

Last Updated: 2018-04-09

Red Hat JBoss Data Virtualization 6.4 Development Guidelines and Recommended Practices Guide

David Sage
dlesage@redhat.com

Legal Notice

Abstract

This guide is intended for both system architects and developers planning to use Red Hat JBoss Data Virtualization. It provides suggestions and optimizations that aim to make using the product smoother. For instance, it gives an overview of the virtual database life-cycle and also provides suggestions for developing in collaborative environments. It allows you to put sound policies in place before you begin to use the product and to avoid common pitfalls once you are running it. The chapters are structured so that suggestions are provided for the planning, deployment and development phases of working with Red Hat JBoss Data Virtualization. This Guide should be read in conjunction with other books in the Red Hat JBoss Data Virtualization documentation suite as this book is focused on providing suggestions, not going into full details of how to execute procedures found in other guides.

Table of Contents

1. DESIGNING YOUR SYSTEM ARCHITECTURE

When you are planning to integrate **Red Hat JBoss Data Virtualization** into your business environment, there are a number of factors to consider. During the design phase, you need to understand the development life-cycle for a virtual database, think about which deployment strategies you wish to use and how you will deploy the virtual database and what impact the product will have on the performance of your systems. This chapter discusses some advantages and disadvantages of different design and deployment approaches and presents some other suggestions for your consideration as you plan your deployment.

1.1. Development Life-cycle

In order to tune the performance of **Red Hat JBoss Data Virtualization**, you need to understand the development life-cycle. You can then optimize each stage.

The development life-cycle consists of three stages:

1. Deploying a virtual database.
2. Updating the deployed database.
3. Rolling back the deployed database.

To improve the performance of the virtual database, change the connection type at each stage of its life-cycle:

Life-cycle Stage	Connection Type to Use	Reason
Deployment	BY_VERSION	This allows the connection to find the new version.
Updating	Change the connection type of the old version from ANY to NONE, then change the connection type of the new version from BY_VERSION to ANY.	This stops the old version from receiving new connections. The new connections will go to the updated version instead.
Rollback	Change the connection type of the new version from ANY to NONE. Change the old connection type from NONE back to ANY.	This prevents any further connections being made to the new version of the database. It activates the old version, allowing it to receive connections once more.

1.2. Version Control

If you are deploying an updated virtual database, you will not want to disrupt your current processing by accidentally confusing which version you are replacing. In order to prevent yourself from making a mistake, you can use a naming convention like this to visually identify the version of the virtual database you are using, without needing to open it:

```
{vdbname}.{version}.vdb
```

1.3. Virtual Database Deployment Options

There are different options you can use to deploy your virtual database. Each has different advantages making them best suited to different situations.

Option	When to Use
Teiid Designer Plug-In	This method is only suitable for situations in which you want to undertake system testing. Do not use it beyond the development phase.
Admin Console	This provides a simple option for production deployment to a single **Red Hat JBoss Data Virtualization** instance.
Manually move the virtual database	Helpful for monitoring the entire life-cycle and retaining manual control.
Red Hat JBoss Operations Network (JON)	This method is best suited for deploying a virtual database to a whole cluster of **Red Hat JBoss Data Virtualization** nodes at once.
Admin Shell	This command line tool is ideal for when you are scripting and automating tasks, and incorporating commands using other tools.

1.4. Creating and Managing a Virtual Database

Here is how you can create and manage a virtual database, using **Red Hat JBoss Developer Studio**'s Teiid Designer plug-in. It shows you how to perform a test deployment and how to reuse a virtual database model in another database for increased efficiency.

1. To create an empty virtual database, launch **Red Hat JBoss Developer Studio**'s New wizard, open the Teiid Designer category folder and select Teiid VDB.

   TIP

   You can also select one or more models in a model project. To do so, right-click and select New → Teiid VDB action.

   TIP

   Virtual Databases are aware of models and files found within their model project. You cannot add models to a virtual database from a different project.

2. To open the new virtual database for editing in Teiid Designer's VDB Editor, in the Explorer view right-click the virtual database and click Open.

3. Test the data source select the source model in your workspace and then click the Modelling → Create Data Source action.
   This extracts the connection profile data from your source model and creates a corresponding data source on your default **Red Hat JBoss Data Virtualization** server.

4. Go to the Model Explorer and right-click on the virtual database.

5. Select the Modelling → Execute VDB option.
   Your virtual database is deployed to the server and a connection profile specific to your virtual database is created.

6. Enter some custom SQL code (for example a SELECT query such as SELECT * FROM TableXXXX) into your scrapbook.

7. Select the entire SQL statement you have just input and then right-click and select Execute Selected Text.

8. To reuse a virtual database, ensure your virtual database is deployed and then launch the Teiid Designer plug-in's JDBC Import Wizard by clicking Import → Teiid Designer → JDBC Database >> Source Model.

9. Select a connection profile from the Connection Profile drop down box that appears.

10. Click Next twice.

11. Under Select Database Objects, select a single schema to use in the creation of the virtual database source model. (The schema names are the names of the visible models in the deployed virtual database.)

12. Click Next.

13. Under Into Folder, choose the name of the folder in which to save the model.

    TIP

    The folder you use must not already contain a model of the same name.

14. Click Finish.

2. DESIGNING YOUR VIRTUAL DATABASE DEPLOYMENT

In this section, you will learn some advice that will help you assess your hardware requirements and design your system architecture in order to run **Red Hat JBoss Data Virtualization** effectively. You will be provided with use cases for using multiple (versus single) virtual databases and the advantages of using caching, high availability clustering, and load balancing.

2.1. Assessing Your Hardware Requirements

2.1.1. Use the JBoss Data Virtualization Sizing Tool

You need to determine your hardware needs before installing **Red Hat JBoss Data Virtualizaton**, in order to ensure you have the resources to run the product effectively.

Red Hat has developed a web application called the **JBoss Data Virtualization Sizing Architecture Tool** to help you do so. This application asks a series of questions about your needs and then presents

recommendations, including how many servers you need, how much memory is required, and what size Java Virtual Machine is needed for each node.

1. Go to https://access.redhat.com/labs/jbossdvsat
2. Answer each question and then click Next to step through each page of the questionnaire.

2.1.2. Scaling and Load Balancing

Sometimes you will experience heavy demand on your server clusters which can lead to performance problems, for example when a large number of queries need to be run at peak periods to consolidate data for major reports. To counteract performance issues caused by increased load demands, you can increase the scale of your cluster by adding more nodes.

Basic load balancing capabilities are built into the **Red Hat JBoss Data Virtualization** JDBC driver. These capabilities ensure that after a connection is established, client nodes balance query requests between the available servers.

An option for TCP load balancing is to use the HA (high availability) Proxy. Using this proxy provides these advantages:

- It allows you to scale the system up without having to change the cluster address.
- It hides the **Red Hat JBoss Data Virtualization** servers from direct user access. When things change, clients do not have to be updated.

2.1.3. Supported Hardware Configurations

You should ensure your environment is fully supported before installing **Red Hat JBoss Data Virtualization**. See https://access.redhat.com/articles/703663

2.1.4. Local Environment Installation

You can run your **Red Hat JBoss Data Virtualization** server inside **Red Hat JBoss Developer Studio** but it is best to install it outside instead for development environments.

2.2. Planning Your System Configuration

2.2.1. Model Layering

With hardware needs met, you need to design the data abstraction layer. You need this level of abstraction in order to separate the physical data sources from the data modelling. By doing so, you can later change the underlying physical hardware and data sources without impacting on the business process layers.

Here is an example model that fulfils these requirements:

Layer Number	Layer Name	Role
Layer 1 (“bottom-most” layer)	Physical Source Layer	This layer consists of the models that map directly to the physical data source.

Layer 2	Virtual Base Layer	This layer consists of a one-to-one mapping with the physical source model. It decouples the physical data from the business layer. It provides a level of isolation from the physical data sources so that they can be changed or swapped. This layer can then be adapted so that the layers above it are not impaired.
Layer 3	Common Object Layer	Use this layer to build the business objects that APIs can access. For example an 'Order' could be referenced by the API used by an OLTP application for entering orders, while another API is exposed solely for reporting on order totals.
Layer 4 (“top-most” layer)	Data Abstraction Layer (API)	This is the layer that is exposed to external applications and business tools. Choose an interface (such as JDBC or a web services interface) to expose to provide these tools with access.

2.3. Choose Between a Single or Multiple Virtual Databases

A single virtual database provides the user with the simplest option in terms of deployment and maintenance. However, if you plan to have multiple applications, using multiple virtual databases provides these advantages:

- They provide you with greater flexibility to perform parallel development and testing.
- Applications can move one virtual database through its life-cycle without impacting other virtual databases.
- You can model data entitlements for one virtual database and, if there is a problem, the exposure is limited. This therefore presents less of a security risk.

2.4. Virtual Database Reuse Feature

If you have a number of developers working on your virtual database simultaneously, there is a risk that changes in one part of the virtual database can have unintended consequences in others. To limit this risk, you can use the Virtual Database Reuse Feature. This allows a virtual database to import other virtual databases upon deployment. You can use it to break a single large virtual database into a number of smaller, separate virtual database that all work together.

By doing so, you can let your developers work on different virtual databases that all reference the same underlying models while, at the same time, segregating the deployment changes to limit the impact of any problems should they arise.

2.5. Cache Data

Caching allows you to improve query performance and ease network loads. It provides data replication so that information consolidated from multiple disparate data sources can be provided in a timely manner.

It also allows your system to work when the data sources are disconnected. By replicating data in the cache, you can offload processing to your local server.

There are various scenarios in which you should use caching. Here are some of the most common:

- You are using **Red Hat JBoss Data Virtualization** to provide static data to an application to populate pick lists. In this case, using an internal materialized (cached) table populated from a data source allows you to perform queries on static data.
- You want to improve the performance of complex queries. If a query that is frequently used runs slowly (such as a query needed to produce a weekly report), it is a good candidate for materialization. By caching the query, the results will be returned more quickly, as you are consolidating data for future use. This will allow you to provide business data in a timely manner.
- You want to create a snapshot of a subset of data. You can use the temporary caching table to store a subset of data in memory in order to access it rapidly.

3. CONNECTING DATA SOURCES

3.1. Tune the Connection Pool

The default data source connection pool value is not recommended for production servers, because it cannot cope with large numbers of concurrent queries. See the **Red Hat JBoss EAP** Tuning Guide's JCA Container section for suggestions on how to tune the connection pool.

To test the setting, be sure to choose one of your complex queries that contains a join across multiple data sources. This is because, in order to obtain the through-put, the connection pools have to be large enough to service all parallel queries issued to **Red Hat JBoss Data Virtualization**.

3.2. Standardize Drivers

You may encounter performance issues if you have different versions of the JBDC driver on different machines and different servers.

Ensure you standardize on a common JDBC driver version across all server instances. For example, if your data source is MySQL, ensure every server uses the same version of the MySQL JDBC driver JAR file. Ensure you use Type 4 drivers where possible as well because they are easier to deploy and allow you to perform previews.

3.3. Standardize Connection Profiles

You may want to import data models to multiple **Red Hat JBoss Developer Studio** environments if you are working with multiple developers.

If you are doing so, establish a standard naming convention and configuration for each **Red Hat JBoss Developer Studio** session's connection profiles. Otherwise, when a model is imported and the connection profile does not match what is expected, you will have to reset the profile before you can preview any data or deploy your virtual database.

You can import or export connection profiles in XML format or in an encrypted binary format if your security needs require it.

If you are collaborating with multiple developers, they can all access identically named profiles, which simplifies the process of sharing source model information.

3.4. Other Optimizations

There are a number of other optimizations you can perform to boost the performance of your connected data sources:

- Undeploy old, unused virtual databases to reduce memory usage.
- Push queries down to the data source, having optimized your driver for this so that they are not being performed at the server level.
- Set the maximum number of threads to use to a reasonable number for your situation.
- Set cardinality.
- Use temporary tables to cache data.
- Make use of SQL hints to optimize queries.

3.5. Develop Your Own Connectors and Translators

You can develop your own custom connectors and translators to work with specific data sources. To speed up development, here are a number of suggestions you can employ:

- Use the JDV translator and/or connector archetypes. These generate the project template, with resources, automatically. Therefore, you can then immediately begin adding your own custom code and compiling to for deployment.
- Create your Maven projects from archetypes using **Red Hat JBoss Developer Studio**. This will help you save time as you just have to customize them.
- To become familiar with the Teiid Designer plug-in and how to model connectors and translators, work through our documentation and the cheat sheets found within Teiid Designer.

3.5.1. View Your Custom Connector in the List of Connection Profiles

After creating your custom connector, you need to add it to the list of connection profiles.

1. In Teiid Designer, click on your custom connector.
2. Click the connection profile drop-down.
3. Click Generic JDBC.
4. Change the driver and all properties to match your custom data source.
5. Right-click on the Source Model
6. Click Modelling → Set Translator Name
7. Enter the custom translator name.

8. Save and exit.

4. DEVELOPING WITH RED HAT JBOSS DATA VIRTUALIZATION

In this chapter, you will find hints for developers. Primarily for those working in collaborative environments, you will be shown good naming convention practices, how to share virtual databases easily, how to structure your folders and how to deal with some performance issues.

4.1. Use a Source Control System

Use a source control system such as GitHub, SVN or CVS to manage your dynamic VDBs. However source control systems should not be used for JAR files like Teiid Designer VDBs and source models. (A source control system allows you to work collaboratively and provides version control and history logs. It also allows you to roll back your dynamic VDB to an earlier version if something goes wrong.)

IMPORTANT

Ensure that the system you choose integrates with **Red Hat JBoss Developer Studio**. The aforementioned ones all do.

4.1.1. Set Up Your Project

If you are targeting a single virtual database, you should set up a project in Teiid Designer for modelling your sources and views. This is because model and virtual database validation are performed within the scope of a project.

4.1.1.1. Share a Virtual Database With Your Team

1. Right-click Export → Teiid Designer → Model Project set. Click **Next**. This makes it shareable.
2. Choose the actual project and the name and location for the ZIP archive. This exports the whole project as a ZIP file on disk.
3. Your team can then right-click Import → General → Existing Projects → Archive and select the ZIP archive containing the project.

4.2. Structure Your Folders

Utilize folders to organize your sources and views. This will allow you to locate files easily.

- Teiid Designer's New Model Project wizard can auto-create folders for such things as sources, views and web_services.
- Store UDF JAR files in a folder called "lib".
- Store your other files for inclusion in your virtual database in a folder named "otherFiles".

4.3. Naming Conventions

Establish consistent naming conventions to avoid confusion in collaborative environments. You should decide whether to use upper case, lower case, or a mixture.

These are the Items you will need to name consistently:

Teiid Designer projects

Multiple developers must use the same project names. There can be multiple projects (for instance, by application) and there can be sharing of models across projects when creating a virtual database. However, in Teiid Designer, the references from a virtual database to a model includes the project name, so using a different name would result in the virtual database's models being left unresolved.

JNDI data sources

Use a consistent naming convention for JNDI Data Source names across all the servers used in the software development life-cycle. The JNDI names used on the server directly impact your Teiid Designer development practices and the ability to move a virtual database throughout its development life-cycle without having to make changes to the artefact. To ensure that virtual databases are portable use the same JNDI names through your development life-cycle and package the virtual database use the standard, agreed upon, JNDI names. (You can reset the JNDI name using the AdminConsole, without having to modify the virtual database.)

source models

Use these to reconcile types with data sources, so do not change the data types at the source. Instead create a view model to make that adjustment.

view models (think about names based on layering)

Model your user-defined functions in view models. Java class path, method name, and jar file reference are required properties. By adding a view model to a virtual database you are also automatically adding the referenced jar file which will deploy to the server when the virtual database is deployed.

functions and procedures

Ensure these are named consistently.

data source drivers

Standardize on a common JDBC driver version across server instances. For instance, if the source is MySQL, every server should use the same version of the MySQL JDBC driver JAR. It is best to use JDBC type 4 drivers, if you can, for ease of deployment. Type 4 drivers also allow you preview data in Teiid Designer. Otherwise, you can manually change the driver to Type 4 or configure the module on the server for the driver prior to use.

connection profiles

Establish a standard naming convention and configuration for connection profiles across **Red Hat JBoss Developer Studio** environments. Otherwise, when a model is imported and the connection profile does not match, you will have to reset the connection profile before you can preview data or deploy your virtual database.

If you wish to define a set of shareable connection profiles, the **Red Hat JBoss Developer Studio** Data Tools project allows you to import or export them in XML format. There is an option for a binary encrypted format, too. If your developers can access identically-named profiles, it will simplify the tasks of sharing and updating source model information.

VDBs

As discussed elsewhere, use the naming conventions for version control.

web services

Ensure these are named consistently also.

4.4. Perform Updates on Data Sources

Use XA data sources if you are going to be doing updates, as they can span multiple resources.

IMPORTANT

Do not use Teiid Designer to create the data sources. This is due to a limitation in the tool, whereby it creates a 'local' data source, not an XA data source. To create an XA data source, use an example template or generate one with the **Admin Console**.

4.5. Use Translator or Connector Archetypes

Use the **Red Hat JBoss Data Virtualization** translator and/or connector archetypes. When these are used, they automatically create the project template, with resources, so you can begin adding your custom code and compiling your package for deployment immediately.

1. Right-click the archetype and click **Create a New Maven Project**.
2. In the wizard, select the “JBoss Nexus Archetypes” catalogue.
3. Filter for “teiid”.
4. Select the one you wish to use.
5. Click **Done**.

4.6. Improve ModeShape Performance

If you find the memory usage is increasing as you use ModeShape, even after you close JCR sessions properly, you will need to define a cache container for workspaces.

1. Open your EAP_HOME/standalone/configuration/standalone.xml file text editor and add this code to define a cache-container (assuming your repository is named “X”):

```
<cache-container name="ws-cache-container" default-cache="X/system"
module="org.modeshape">
       <local-cache name="X/system">
          <eviction strategy="LRU" max-entries="100"/>
          <expiration lifespan="10000" interval="1000" max-
idle="5000"/>
       </local-cache>
       <local-cache name="X/default">
          <eviction strategy="LRU" max-entries="100"/>
          <expiration lifespan="10000" interval="1000" max-
idle="5000"/>
       </local-cache>
</cache-container>
```

2. Find the “repository” section of the file and add this code to force the system to use the cache configuration you just created:

```
 <repository name="X"...>
    <workspaces cache-container="ws-cache-container">
 ...
```

3. Save the file and exit.

4.7. Improve Dashboard Builder Performance

To avoid performance issues with the Dashboard Builder, do not load a data set with too many rows (dependant on your system's capacity) into the tool's memory because the SQL load and execution times lengthen as the amount of data grows and operations to measure performance take longer.