Red Hat JBoss Data Virtualization 6.3

Red Hat JBoss Data Virtualization for OpenShift

Learn how to use Red Hat JBoss Data Virtualization with OpenShift.

Last Updated: 2018-04-25

Red Hat JBoss Data Virtualization 6.3 Red Hat JBoss Data Virtualization for OpenShift

Learn how to use Red Hat JBoss Data Virtualization with OpenShift.

Documentation Team

Legal Notice

Copyright © 2018 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

Table of Contents

CHAPTER 1. 6.3 IMAGE DEPRECATED 3

CHAPTER 2. RED HAT JBOSS DATA VIRTUALIZATION FOR OPENSHIFT 4

CHAPTER 3. BEFORE YOU BEGIN 5
3.1. COMPARISON: JDV FOR OPENSHIFT IMAGE AND RED HAT JDV 5
3.2. VERSION COMPATIBILITY AND SUPPORT 5
3.3. INITIAL SETUP 5

CHAPTER 4. GET STARTED 6
4.1. USING THE JDV FOR OPENSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES 6
4.2. PREPARING JDV PROJECT ARTIFACTS 6
4.2.1. S2I Artifacts 6
4.2.1.1. Virtual Databases (VDB) 6
4.2.1.2. Modules, Drivers, Translators, and Generic Deployments 7
4.2.2. Runtime Artifacts 9
4.2.2.1. Datasources 9
4.2.2.2. Resource Adapters 10
4.3. PREPARING AND DEPLOYING THE JDV FOR OPENSHIFT APPLICATION TEMPLATES 11
4.3.1. Configuring Keystores 11
4.3.2. Generating the Keystore Secret 11
4.3.3. Generating the Artifact Secrets 11
4.3.4. Creating the Service Account 12
4.3.5. Configuring Red Hat Single-Sign On Authentication 12
4.3.6. Using the OpenShift Web Console 13
4.4. USING JBOSS DATAGRID FOR OPENSHIFT WITH JDV FOR OPENSHIFT 13
4.4.1. Using the JDG for OpenShift Application Templates 14
4.4.2. JDG for OpenShift Authentication Environment Variables 14
4.4.3. JDG for OpenShift Resource Adapter Properties 15
4.4.4. Protocol Buffers 18

CHAPTER 5. TUTORIALS 20
5.1. EXAMPLE WORKFLOW: DEPLOYING A JDV PROJECT USING THE JDV FOR OPENSHIFT IMAGE 20
5.1.1. Preparing Red Hat JBoss Data Virtualization for OpenShift Deployment 20
5.1.2. Deployment 21
5.1.3. Defining Alternate Data Sources 21
5.2. EXAMPLE WORKFLOW: DEPLOYING THE JDG FOR OPENSHIFT AS A MATERIALIZATION TARGET QUICKSTART 22
5.2.1. Preparing the Project to Deploy JDG for OpenShift 22
5.2.2. Deploying JDG for OpenShift 22
5.2.3. Configure JDV for OpenShift to use JDG for OpenShift as a Materialization Target 23

CHAPTER 6. REFERENCE 24
6.1. GLOSSARY 24
6.2. ARTIFACT REPOSITORY MIRRORS 24

CHAPTER 1. 6.3 IMAGE DEPRECATED

WARNING

The **`jboss-datavirt-6/datavirt63-openshift`** image detailed in this guide is deprecated. See the Red Hat Container Catalog for the latest JDV for OpenShift images and accompanying documentation.

CHAPTER 2. RED HAT JBOSS DATA VIRTUALIZATION FOR OPENSHIFT

Red Hat JBoss Data Virtualization(JDV) is a lean, virtual data integration solution that unlocks trapped data and delivers it as easily consumable, unified, and actionable information. Red Hat JBoss Data Virtualization makes data spread across physically diverse systems — such as multiple databases, XML files, and Hadoop systems — appear as a set of tables in a local database.

Red Hat offers three JDV for OpenShift application templates:

Application Template	Description
datavirt63-basic	Application template for JBoss Data Virtualization 6.3 services built using S2I.
datavirt63-secure	Includes ability to configure certificates for serving secure content.
datavirt63-extensions-support	Includes support for installing extensions (e.g. third-party DB drivers) and the ability to configure certificates for serving secure content.

WARNING

These application templates are deprecated. See the Red Hat Container Catalog for the latest JDV for OpenShift images, templates and documentation.

CHAPTER 3. BEFORE YOU BEGIN

3.1. COMPARISON: JDV FOR OPENSHIFT IMAGE AND RED HAT JDV

JDV for OpenShift image is based on Red Hat JBoss Data Virtualization 6.3. There are some differences in functionality between the JDV for OpenShift image and Red Hat JBoss Data Virtualization:

- Cached results are automatically replicated to all members of the JDV for OpenShift cluster.

In addition, the JDV for OpenShift image is built on the Red Hat JBoss Enterprise Application Platform (EAP) for OpenShift image. As a result, the same differences exist for the JDV for OpenShift image. For more information on the EAP for OpenShift differences, see the Comparison: EAP and EAP for OpenShift Image section in the Red Hat JBoss Enterprise Application Platform for OpenShift Guide.

3.2. VERSION COMPATIBILITY AND SUPPORT

See the xPaaS part of the OpenShift and Atomic Platform Tested Integrations page for details about OpenShift image version compatibility.

3.3. INITIAL SETUP

The Tutorials in this guide follow on from and assume an OpenShift instance similar to that created in the OpenShift Primer.

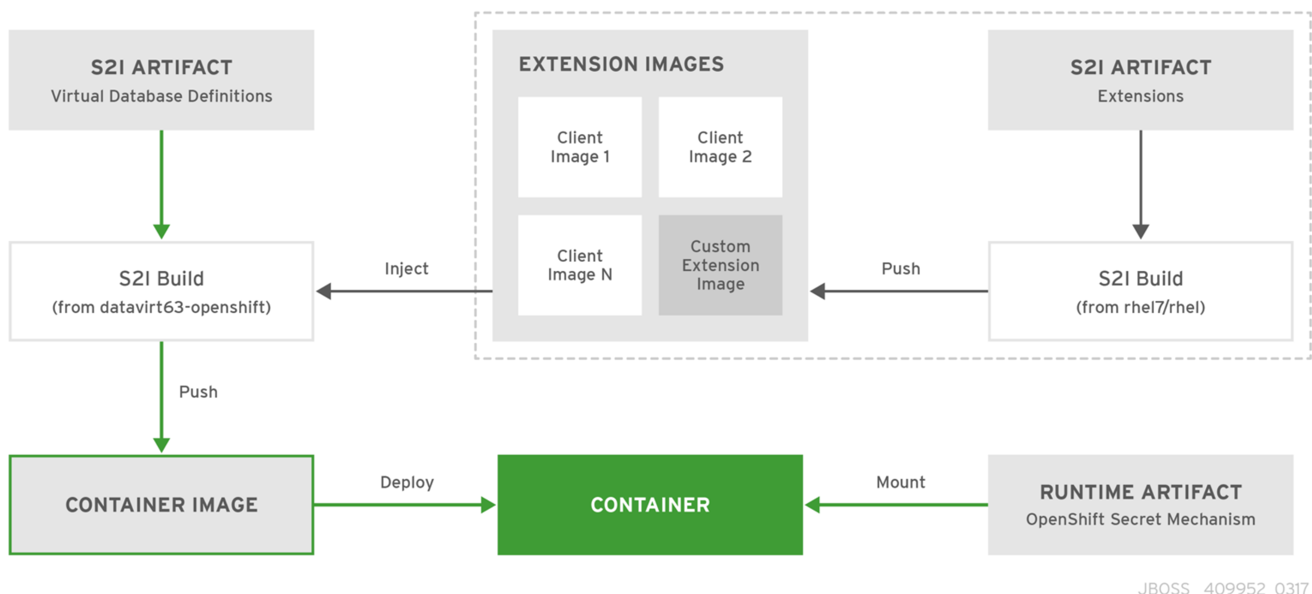
CHAPTER 4. GET STARTED

4.1. USING THE JDV FOR OPENSHIFT IMAGE STREAMS AND APPLICATION TEMPLATES

Red Hat JBoss Middleware for OpenShift images are pulled on demand from the Red Hat Registry: registry.access.redhat.com

4.2. PREPARING JDV PROJECT ARTIFACTS

The EAP for OpenShift image, on which the JDV for OpenShift image is built, extends database support in OpenShift using various artifacts. These artifacts are included in the built image through different mechanisms:
- S2I artifacts that are injected into the image during the S2I process, and
- Runtime artifacts from environment files provided through the OpenShift Secret mechanism.



4.2.1. S2I Artifacts

The S2I artifacts include the virtual databases files, modules, drivers, translators, and additional generic deployments that provide the necessary configuration infrastructure required for the JDV for OpenShift deployment. This configuration is built into the image during the S2I process so that only the datasources and associated resource adapters need to be added at runtime.

Refer to the Artifact Repository Mirrors section for additional guidance on how to instruct the S2I process to utilize the custom Maven artifacts repository mirror.

4.2.1.1. Virtual Databases (VDB)

VDBs contain aggregated data from multiple disparate datasources. This allows applications to access and query the data as though it is in a single database, using a single uniform API.

Deployment

***.vdb** and ***-vdb.xml** files can be included in a Git repository, or in the source for a Binary-type build.

The JDV for OpenShift image uses the file deployment method for deploying virtual databases. Create

an empty marker file in the same directory and with the same name as the VDB with the additional extension **.dodeploy**. For example, if the VDB file is *database.vdb*, the marker file must be called *database.vdb.dodeploy*.

For more information on the file deployment method, see the Deploy a VDB via File Deployment section in the Red Hat JBoss Data Virtualization Administration and Configuration Guide.

4.2.1.2. Modules, Drivers, Translators, and Generic Deployments

There are three options for including these S2I artifacts in the JDV for OpenShift image:

1. Include the artifact in the application source deployment directory. The artifact is downloaded during the build and injected into the image. This is similar to deploying an application on the EAP for OpenShift image.

2. Include the **CUSTOM_INSTALL_DIRECTORIES** environment variable, a list of comma-separated list of directories used for installation and configuration of artifacts for the image during the S2I process. There are two methods for including this information in the S2I:
   2a) An **install.sh** script in the nominated installation directory. The install script will be executed during the S2I process and will operate with impunity.
   Install.sh script example:

   ```
   #!/bin/bash

   injected_dir=$1
   source /usr/local/s2i/install-common.sh
   install_deployments ${injected_dir}/injected-deployments.war
   install_modules ${injected_dir}/modules
   configure_drivers ${injected_dir}/drivers.env
   source /usr/local/s2i/install-teiid-common.sh
   configure_translators ${injected_dir}/translators.env
   ```

   The **install.sh** script is responsible for customizing the base JDV for OpenShift image using APIs provided by the **install-common.sh**, which is contained in the underlying base EAP for OpenShift image, and **install-teiid-common.sh**, which is contained in the base JDV for OpenShift image. It allows for the greatest amount of flexibility for configuring the JDV for OpenShift image. These shell scripts contain functions that are used by the **install.sh** script to install and configure the modules, drivers, translators, and generic deployments.

Shell script	Functions contained within script
install-common.sh	configure_translators
install-teiid-common.sh	install_modules configure_drivers install_deployments

   Modules

   A module is a logical grouping of classes used for class loading and dependency management. Modules are defined in the **EAP_HOME**/**modules**/ directory of the application server. Each module exists as a subdirectory, for example **EAP_HOME**/**modules**/**org**/**apache**/. Each module

directory then contains a slot subdirectory, which defaults to main and contains the **module.xml** configuration file and any required JAR files.

module.xml example:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.0" name="org.apache.derby">
    <resources>
        <resource-root path="derby-10.12.1.1.jar"/>
        <resource-root path="derbyclient-10.12.1.1.jar"/>
    </resources>
    <dependencies>
        <module name="javax.api"/>
        <module name="javax.transaction.api"/>
    </dependencies>
</module>
```

Drivers and translators are installed as modules. The **install_modules** function in the **install.sh** copies the respective JAR files to the modules directory in EAP, along with the **module.xml**. The **configure_drivers** and **configure_translators** functions of the **install.sh** add additional information to the **$JBOSS_HOME/standalone/configuration/standalone-openshift.xml** configuration file.

Drivers

Drivers are installed as modules. The **install_modules** function in the **install.sh** copies the respective JAR files to the modules directory in EAP. The driver is then configured in the **install.sh** by the **configure_drivers function**, the configuration properties for which are defined in a runtime artifact environment file.

drivers.env example:

```
#DRIVER
DRIVERS=DERBY
DERBY_DRIVER_NAME=derby
DERBY_DRIVER_MODULE=org.apache.derby
DERBY_DRIVER_CLASS=org.apache.derby.jdbc.EmbeddedDriver
DERBY_XA_DATASOURCE_CLASS=org.apache.derby.jdbc.EmbeddedXADataSource
```

Translators

Translators are installed as modules. The **install_modules** function in the **install.sh** copies the JAR files for the translator. The translator is then configured in the **install.sh** by the **configure_translators** function, the configuration properties for which are in a runtime artifact environment file.

translators.env example:

```
#TRANSLATOR
TRANSLATORS=TESTTRANSLATOR

TESTTRANSLATOR_NAME=test
TESTTRANSLATOR_MODULE=org.jboss.teiid.translator.test
```

Generic Deployments

Deployable archive files, such as JARs, WARs, RARs, or EARs, can be deployed from an injected image using the **install_deployments** function supplied by the API in the **install-common.sh** contained in the EAP for OpenShift image.

2b) If the **CUSTOM_INSTALL_DIRECTORIES** environment variable has been declared but no **install.sh** scripts are found in the custom installation directories, the following artifact directories will be copied to their respective destinations in the built image:
- **modules**/* copied to **$JBOSS_HOME**/**modules**/**system**/**layers**/**openshift**
- **configuration**/* copied to **$JBOSS_HOME**/**standalone**/**configuration**
- **deployments**/* copied to **$JBOSS_HOME**/**standalone**/**deployments**
This is a basic configuration approach compared to the **install.sh** alternative, and requires the artifacts to be structured appropriately.

4.2.2. Runtime Artifacts

4.2.2.1. Datasources

There are three types of datasources:
1. Default internal datasources. These are PostgreSQL, MySQL, and MongoDB. These datasources are available on OpenShift by default through the Red Hat Registry and do not require additional environment files to be configured. Set the DB_SERVICE_PREFIX_MAPPING to the name of the OpenShift service for the database to be discovered and used as a datasource. 2. Other internal datasources. These are datasources not available by default through the Red Hat Registry but run on OpenShift. Configuration of these datasources is provided by environment files added to OpenShift Secrets. 3. External datasources that are not run on OpenShift. Configuration of external datasources is provided by environment files added to OpenShift Secrets.

Datasource environment file example:

```
# derby datasource
ACCOUNTS_DERBY_DATABASE=accounts
ACCOUNTS_DERBY_JNDI=java:/accounts-ds
ACCOUNTS_DERBY_DRIVER=derby
ACCOUNTS_DERBY_USERNAME=derby
ACCOUNTS_DERBY_PASSWORD=derby
ACCOUNTS_DERBY_TX_ISOLATION=TRANSACTION_READ_UNCOMMITTED
ACCOUNTS_DERBY_JTA=true

# Connection info for xa datasource
ACCOUNTS_DERBY_XA_CONNECTION_PROPERTY_DatabaseName=/home/jboss/source/data
/databases/derby/accounts

# _HOST and _PORT are required, but not used
ACCOUNTS_DERBY_SERVICE_HOST=dummy
ACCOUNTS_DERBY_SERVICE_PORT=1527
```

The **DATASOURCES** property is a comma-separated list of datasource property prefixes. These prefixes are then appended to all properties for that datasource. Multiple datasources can then be included in a single environment file. Alternatively, each datasource can be provided in separate environment files.

Datasources contain two types of properties: connection pool-specific properties and data driver-specific properties. Data-driver specific properties use the generic **XA_CONNECTION_PROPERTY**, as the driver itself is configured as a driver S2I artifact. The suffix of the driver property is specific to the

particular driver for the datasource.
In the above example, ***ACCOUNTS*** is the datasource prefix, **XA_CONNECTION_PROPERTY** is the generic driver property, and ***DatabaseName*** is the property specific to the driver.

The datasources environment files are added to the OpenShift Secret for the project. These environment files are then called within the JDV template using the **ENV_FILES** environment property, the value of which is a comma-separated list of fully qualified environment files.

For example:

```
{
    “Name”: “ENV_FILES”,
    “Value”: “/etc/jdv-extensions/datasources1.env,/etc/jdv-
extensions/datasources2.env”
}
```

4.2.2.2. Resource Adapters

Configuration of resource adapters is provided by environment files added to OpenShift Secrets.

Resource adapter environment file example:

```
#RESOURCE_ADAPTER
RESOURCE_ADAPTERS=QSFILE

QSFILE_ID=fileQS
QSFILE_MODULE_SLOT=main
QSFILE_MODULE_ID=org.jboss.teiid.resource-adapter.file
QSFILE_CONNECTION_CLASS=org.teiid.resource.adapter.file.FileManagedConnect
ionFactory
QSFILE_CONNECTION_JNDI=java:/marketdata-file
QSFILE_PROPERTY_ParentDirectory=/home/jboss/source/injected/injected-
files/data
QSFILE_PROPERTY_AllowParentPaths=true
```

The **RESOURCE_ADAPTERS** property is a comma-separated list of resource adapter property prefixes. These prefixes are then appended to all properties for that resource adapter. Multiple resource adapter can then be included in a single environment file. Alternatively, each resource adapter can be provided in separate environment files.

The resource adapter environment files are added to the OpenShift Secret for the project namespace. These environment files are then called within the JDV template using the **ENV_FILES** environment property, the value of which is a comma-separated list of fully qualified environment files.

For example:

```
{
    “Name”: “ENV_FILES”,
    “Value”: “/etc/jdv-extensions/resourceadapter1.env,/etc/jdv-
extensions/resourceadapter2.env”
}
```

4.3. PREPARING AND DEPLOYING THE JDV FOR OPENSHIFT APPLICATION TEMPLATES

4.3.1. Configuring Keystores

The JDV for OpenShift image requires two keystores:
- An SSL keystore to provide private and public keys for https traffic encryption.
- A JGroups keystore to provide private and public keys for network traffic encryption between nodes in the cluster.

These keystores are expected by the JDV for OpenShift image, even if the application uses only http on a single-node OpenShift instance. Self-signed certificates do not provide secure communication and are intended for internal testing purposes.

WARNING

For production environments Red Hat recommends that you use your own SSL certificate purchased from a verified Certificate Authority (CA) for SSL-encrypted connections (HTTPS).

See the JBoss Enterprise Application Platform Security Guide for more information on how to create a keystore with self-signed or purchased SSL certificates.

4.3.2. Generating the Keystore Secret

OpenShift uses objects called **Secrets** to hold sensitive information, such as passwords or keystores. See the Secrets chapter in the OpenShift documentation for more information.

The JDV for OpenShift image requires a secret that holds the two keystores described earlier. This provides the necessary authorization to applications in the project.

Use the SSL and JGroups keystore files to create a secret for the project:

```
$ oc secret new <jdv-secret-name> <ssl.jks> <jgroups.jceks>
```

After the secret has been generated, it can be associated with a service account.

4.3.3. Generating the Artifact Secrets

Files for runtime artifacts are passed to the JDV for OpenShift image using the OpenShift secret mechanism. This includes the environment files for the data sources and resource adapters, as well as any additional data files. These files need to be present locally so as to create secrets for them.

```
$ oc secrets new <datavirt-app-config> <datasource.env>
<resourceadapter.env> <additional/data/files/>
```

The Red Hat JBoss Data Virtualization for OpenShift application template uses *datavirt-app-config* as a default value for the *CONFIGURATION_NAME* environment variable. This value is also used to

configure a storage volume for the project. If a different secret name is used, the value of the *CONFIGURATION_NAME* environment variable must be updated.

If the project does not require any runtime artifacts, the secret must still be present in the OpenShift project or the deployment will fail. An empty secret can be generated and supplied:

```
$ touch <empty.env>
$ oc secrets new <datavirt-app-config> <empty.env>
```

4.3.4. Creating the Service Account

Service accounts are API objects that exist within each project and allow users to associate certain secrets and roles with applications in a project namespace. This provides the application with the necessary authorization to run with all required privileges.

The service account that you create must be configured with the correct permissions to view pods in Kubernetes. This is required in order for clustering with the JDV for OpenShift image to work. You can view the top of the log files to see whether the correct service account permissions have been configured.

The JDV for OpenShift templates have a default **SERVICE_ACCOUNT_NAME** variable of *datavirt-service-account*. If a different service account name is used, this variable must be configured with the appropriate service account name.

1. Create a service account to be used for the JDV deployment:

   ```
   $ oc create serviceaccount <service-account-name>
   ```

2. Add the **view** role to the service account. This enables the service account to view all the resources in the application namespace in OpenShift, which is necessary for managing the cluster.

   ```
   $ oc policy add-role-to-user view system:serviceaccount:<project-
   name>:<service-account-name> -n <project-name>
   ```

3. Add the secrets created for the project to the service account:

   ```
   $ oc secret link <service-account-name> <jdv-secret-name> <jdv-
   datasource-secret> <jdv-resourceadapter-secret> <jdv-datafiles-
   secret>
   ```

4.3.5. Configuring Red Hat Single-Sign On Authentication

JDV for OpenShift can be configured to use Red Hat Single-Sign On (SSO) for authentication. The **datavirt63-secure-s2i** application template includes SSO environment variables for automatic SSO client registration.

See the Automatic and Manual SSO Client Registration Methods for more information on automatic SSO client registration.

Once configured, the SSO user and the SSO realm token can be used to authenticate the specified endpoints in the JDV for OpenShift project.

4.3.6. Using the OpenShift Web Console

Log in to the OpenShift web console:

1. Click **Add to project** to list all of the default image streams and templates.

2. Use the **Filter by keyword** search bar to limit the list to those that match **datavirt**. You may need to click **See all** to show the desired application template.

3. Select an application template and configure the deployment parameters as required.

4. Click **Create** to deploy the application template.

4.4. USING JBOSS DATAGRID FOR OPENSHIFT WITH JDV FOR OPENSHIFT

The JBoss Datagrid for OpenShift image can be configured so that it integrates with a JDV for OpenShift deployment. There are two use cases for this integration:

- Using JDG as a datasource for JDV.

- Using JDG as an external materialization target for JDV and one or more databases. When deployed as a materialization target, JDG for OpenShift uses in-memory caching to store common queries to other remote databases, increasing performance. For more information on external materialization, see the External Materialization and Red Hat JBoss Data Grid chapter in the Red Hat JBoss Data Virtualization Caching Guide.

In both of these use cases, both the JDG for OpenShift and JDV for OpenShift deployments need to be configured. This includes:

- Specifying the cache names in the JDG for OpenShift application template.

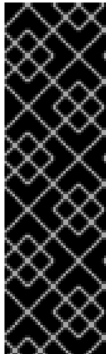
    - Specifying the cache names to be used for the JDG for OpenShift deployment with the appropriate environment variable.

- Including JDG-specific properties in a resource adapter environment file.

- If protocol buffers are specified in a resource adapter, the relevant ***.proto*** files need to be included in the project. If protocol buffers are not specified, JDV for OpenShift will use standard Java serialization mechanism to interact with JDG for OpenShift.

The JDV for OpenShift templates automatically include the client dependencies necessary for communicating with the JDG for Openshift image during the s2i build process. These modules are mounted on the JDV for OpenShift at:

```
${CONTEXT_DIR}/extensions/datagrid65
```



IMPORTANT

Red Hat JBoss Data Grid, and the JDG for OpenShift image, support multiple protocols; however, when deployed with JDV for OpenShift, only the Hot Rod protocol is supported for JDG for OpenShift.

The Hot Rod protocol is not encrypted.

For more information on the Hot Rod protocol, see the Remote Querying chapter of the Red Hat JBoss DataGrid Infinispan Query Guide.

4.4.1. Using the JDG for OpenShift Application Templates

Whether using JDG for OpenShift as a datasource or materialization target, the cache names need to be specified in the application template so that they can be used with JDV for OpenShift. The environment variable to specify these cache names is different depending on whether JDG for OpenShift is to be used as a datasource or a materialization:

Using JDG for OpenShift as a datasource:

CACHE_NAMES

Comma-separated list of the cache names to be used for the JDG for OpenShift datasource.

CACHE_TYPE_DEFAULT

Must be set with value `replicated`. Alternate cache methods are not supported.

Using JDG for OpenShift as a materialization target:

DATAVIRT_CACHE_NAMES

Comma-separated list of the cache names to be used for the JDG for OpenShift materialization target. When the image is built, three caches will be created per cache name provided: {cachename}, {cachename}_staging, and {cachename}_alias. These three caches enable JBoss Data Grid to simultaneously maintain and refresh materialization caches.

4.4.2. JDG for OpenShift Authentication Environment Variables

To use JDG for OpenShift as an authenticated datasource, additional environment variables must be provided in the JDG for OpenShift application template. These environment variables provide authentication details for the Hot Rod protocol, which will be provided to JDV for OpenShift in the resource adapter, and authorization details for the caches in the JDG for OpenShift deployment.

For more information on these environment variables, see the Red Hat JBoss Data Grid for OpenShift Guide.

JDG for OpenShift Authentication Environment Variables

Environment Variable	Description	Example value
USERNAME	Username for the JDG user	jdg-user
PASSWORD	Password for the JDG user	JBoss.123
HOTROD_AUTHENTICATION	Enable Hot Rod authentication	true

Environment Variable	Description	Example value
CONTAINER_SECURITY_ROLE_MAPPER	Role mapper for the Hot Rod protocol	identity-role-mapper
CONTAINER_SECURITY_ROLES	Security roles and permissions attributed to the role mapper	admin=ALL

JDG for OpenShift Cache Authorization Environment Variables

Resource Adapter Property	Description	Example value
<*cache-name*>_CACHE_SECURITY_AUTHORIZATION_ENABLED	Enables authorization checks for this cache	true
<*cache-name*>_CACHE_SECURITY_AUTHORIZATION_ROLES	Sets the valid roles required to access this cache	admin

4.4.3. JDG for OpenShift Resource Adapter Properties

To use JDG for OpenShift with JDV for OpenShift, properties specific to JDG are required within a resource adapter. As with all resource adapters, these can be included as a separate resource adapter environment file or along with other resource adapters in a larger environment file and supplied to the build as an OpenShift secret.

The properties in the table below are in addition to the standard properties required by JDV for OpenShift to configure a resource adapter:
JDG_ID
JDG_MODULE_SLOT
JDG_MODULE_ID
JDG_CONNECTION_CLASS
JDG_CONNECTION_JNDI

Additional Properties for the JDG Resource Adapter

Resource Adapter Property	Description	Required
CacheTypeMap	Cache Type Map(cacheName:className[;pkFieldName[:cacheKeyJavaType]])	True
ProtobufDefinitionFile	Class Path to the Google Protobuf Definition file that's packaged in a jar	True if using protocol buffers

Resource Adapter Property	Description	Required
MessageMarshallers	Class names for the message marshallers, (marshaller, [marshaller,..]), that are to be registered for serialization	True if using protocol buffers
MessageDescriptor	Protobuf message descriptor class name for the root object in cache	
Module	Module defining the protobuf classes (module_slot, [module_id,..])	
RemoteServerList	Server List (host:port[;host:port…]) to connect to	
HotRodClientPropertiesFile	HotRod Client properties file for configuring connection to remote cache	
CacheJndiName	JNDI Name to find CacheContainer	
ChildClasses	Child pojo classes, with annotations, that are accessible from the root class [className[;className];])	

Additional Resource Adapter for using JDG for OpenShift as a Datasource

Resource Adapter Property	Description	Required
AuthUserName	Authentication username for the JDG user	True if using JDG as an authenticated datasource
AuthPassword	Authentication password for the JDG user	True if using JDG as an authenticated datasource
AuthApplicationRealm	Authorized application realm for the JDG user	True if using JDG as an authenticated datasource
AuthServerName	Name of the JDG server	True if using JDG as an authenticated datasource
AuthSASLMechanism	Encryption type used for authentication	True if using JDG as an authenticated datasource

Resource Adapter Property	Description	Required
AdminUserName	Username for the JDG admin user	True if using JDG as an authenticated datasource
AdminPassword	Password for the JDG user	True if using JDG as an authenticated datasource

The following is an example of a JDV for OpenShift resource adapter for integrating with JDG for OpenShift. This example is taken from the quickstart used in Example Workflow: Deploying the JDG for OpenShift as a Materialization Target Quickstart Tutorials:

Resource Adapter for Using JDG for OpenShift as Materialization Cache Example

```
RESOURCE_ADAPTERS=MAT_CACHE
MAT_CACHE_ID=infinispanRemQSDSL
MAT_CACHE_MODULE_SLOT=main
MAT_CACHE_MODULE_ID=org.jboss.teiid.resource-adapter.infinispan.dsl
MAT_CACHE_CONNECTION_CLASS=org.teiid.resource.adapter.infinispan.dsl.Infin
ispanManagedConnectionFactory
MAT_CACHE_CONNECTION_JNDI=java:/infinispanRemoteDSL

MAT_CACHE_PROPERTY_CacheTypeMap="addressbook:org.jboss.as.quickstarts.data
grid.hotrod.query.domain.Person;id"
MAT_CACHE_PROPERTY_ProtobufDefinitionFile=/quickstart/addressbook.proto
MAT_CACHE_PROPERTY_MessageDescriptor=quickstart.Person
MAT_CACHE_PROPERTY_Module=com.client.quickstart.addressbook.pojos
MAT_CACHE_PROPERTY_MessageMarshallers=org.jboss.as.quickstarts.datagrid.ho
trod.query.domain.Person:org.jboss.as.quickstarts.datagrid.hotrod.query.ma
rshallers.PersonMarshaller,org.jboss.as.quickstarts.datagrid.hotrod.query.
domain.PhoneNumber:org.jboss.as.quickstarts.datagrid.hotrod.query.marshall
ers.PhoneNumberMarshaller,org.jboss.as.quickstarts.datagrid.hotrod.query.d
omain.PhoneType:org.jboss.as.quickstarts.datagrid.hotrod.query.marshallers
.PhoneTypeMarshaller
MAT_CACHE_PROPERTY_RemoteServerList=${DATAGRID_APP_HOTROD_SERVICE_HOST}:${
DATAGRID_APP_HOTROD_SERVICE_PORT}
# Additionally, the following are required when using JDG to materialize
views
MAT_CACHE_PROPERTY_StagingCacheName=addressbook_staging
MAT_CACHE_PROPERTY_AliasCacheName=addressbook_alias
```

In the above example, a line break separates the standard JDV for OpenShift resource adapter configuration from the additional properties required for JDG for OpenShift, which carry a **PROPERTY** suffix appended to the **MAT_CACHE** resource adapter prefix.

In the above example:

- **PROPERTY_CacheTypeMap** carries the cache name (**addressbook**), the class name (**Person**), and the field name (**id**) relative to the protocol buffer.
- **PROPERTY_ProtobufDefinitionFile** provides the relative filepath to the **addressbook.proto** protocol buffer file.

- **PROPERTY_MessageMarshallers** provides two marshallers: **Person** and **PhoneType**.
- **PROPERTY_StagingCacheName** and **PROPERTY_AliasCacheName** provide the staging and alias cache names respectively, which are necessary to maintain and refresh the **addressbook** materialization cache.

The following is an example of a JDV for OpenShift resource adapter for using JDG for OpenShift as a datasource.

Resource Adapter for Using JDG for OpenShift as an Authenticated Datasource Example

```
RESOURCE_ADAPTERS=JDG

JDG_ID=infinispanRemQSDSL
JDG_MODULE_SLOT=main
JDG_MODULE_ID=org.jboss.teiid.resource-adapter.infinispan.dsl
JDG_CONNECTION_CLASS=org.teiid.resource.adapter.infinispan.dsl.InfinispanM
anagedConnectionFactory
JDG_CONNECTION_JNDI=java:/infinispanRemoteDSL

JDG_PROPERTY_CacheTypeMap="addressbook:com.redhat.xpaas.datagrid.hotrod.qu
ery.domain.Person;id"
JDG_PROPERTY_ProtobufDefinitionFile=/xpaas/addressbook.proto
JDG_PROPERTY_MessageDescriptor=xpaas.Person
JDG_PROPERTY_Module=com.client.xpaas.person.pojos
JDG_PROPERTY_MessageMarshallers=com.redhat.xpaas.datagrid.hotrod.query.dom
ain.Person:com.redhat.xpaas.datagrid.hotrod.query.marshallers.PersonMarsha
ller
JDG_PROPERTY_RemoteServerList=${DATAGRID_APP_HOTROD_SERVICE_HOST}:${DATAGR
ID_APP_HOTROD_SERVICE_PORT}

JDG_PROPERTY_AuthUserName=jdg-user
JDG_PROPERTY_AuthPassword=JBoss.123
JDG_PROPERTY_AuthApplicationRealm=ApplicationRealm
JDG_PROPERTY_AuthServerName=jdg-server
JDG_PROPERTY_AuthSASLMechanism=DIGEST-MD5
JDG_PROPERTY_AdminUserName=jdg-user
JDG_PROPERTY_AdminPassword=JBoss.123
```

In the above example, line breaks separate the standard JDV for OpenShift resource adapter configuration, the additional properties required for JDG for Openshift, and the authentication properties for the JDG datasource. The **PROPERTY_AuthUserName** and **PROPERTY_AdminUserName** and associated passwords correspond to the values provided to the JDG for OpenShift application template in JDG for OpenShift Authentication Environment Variables section.

4.4.4. Protocol Buffers

Protocol buffers provide a mechanism for serializing structured data. Protocol buffers are supported in Hot Rod as an alternative to standard Java serialization mechanism.
For more information on Protocol Buffers, see link:https://developers.google.com/protocol-buffers/docs/overview

The following is an example of a .proto file taken from the **dynamicvdb-datafederation** quickstart used in Example Workflow: Deploying the JDG for OpenShift as a Materialization Target Quickstart.

Example addressbook.proto file:

```
package quickstart;

/* @Indexed */
message Person {

   /* @IndexedField */
   required string name = 1;
   /* @IndexedField(index=true, store=false) */
   required int32 id = 2;
   optional string email = 3;
   enum PhoneType {
      MOBILE = 0;
      HOME = 1;
      WORK = 2;
   }

   /* @Indexed */
   message PhoneNumber {
      /* @IndexedField */
      required string number = 1;
      /* @IndexedField(index=false, store=false) */
      optional PhoneType type = 2 [default = HOME];
   }
   /* @IndexedField(index=true, store=false) */
   repeated PhoneNumber phone = 4;
}
```

The above **addressbook.proto** example contains the object **Person**, which is made up of three primitive fields - **name**, **id**, and **email** - and an embedded object **PhoneNUmber**, which itself contains two primitive fields - **number** and **PhoneType** - and a repeatable field named **phone**.

CHAPTER 5. TUTORIALS

These tutorials build on from the environment created in the OpenShift Primer and from the information in the previous chapter.

For a different end-to-end demonstration for deploying JDV for OpenShift, see the four-part Red Hat JBoss Data Virtualization on OpenShift article series on the Red Hat Developer Program Blog:

- Part 1: Getting started
- Part2: Service enable your data
- Part 3: Data federation
- Part4: Bringing data from outside to inside the PaaS

5.1. EXAMPLE WORKFLOW: DEPLOYING A JDV PROJECT USING THE JDV FOR OPENSHIFT IMAGE

This Example Worfklow downloads and deploys the **dynamicvdb-datafederation** quickstart available in the **jboss-openshift** GitHub. This quickstart federates data from a relational data source (H2, Derby, MySQL, or PostgreSQL) and an EXCEL file. H2 and Derby data sources are embedded in the JDV for OpenShift pod; MySQL and PostgreSQL data sources deploy in separate pods. The default quickstart deploys a H2 data source.

After deployment, this example will demonstrate how to use the **QS-DB-TYPE** environment variable to define an alternate data source. This variable is only applicable to the quickstart and is not valid for other deployments.

5.1.1. Preparing Red Hat JBoss Data Virtualization for OpenShift Deployment

1. Create a new project:

   ```
   $ oc new-project jdv-app-demo
   ```

2. Create a service account to be used for the Red Hat JBoss Data Virtualization for OpenShift deployment:

   ```
   $ oc create serviceaccount datavirt-service-account
   ```

3. Add the view role to the service account. This enables the service account to view all the resources in the jdv-app-demo namespace, which is necessary for managing the cluster.

   ```
   $ oc policy add-role-to-user view system:serviceaccount:jdv-app-
   demo:datavirt-service-account
   ```

4. The Red Hat JBoss Data Virtualization for OpenShift template requires an SSL keystore and a JGroups keystore.
   These keystores are expected even if the application will not use https.
   This example uses ‘keytool’, a package included with the Java Development Kit, to generate self-signed certificates for these keystores. The following commands will prompt for passwords.

   a. Generate a secure key for the SSL keystore:

```
$ keytool -genkeypair -alias https -storetype JKS -keystore
keystore.jks
```

b. Generate a secure key for the JGroups keystore:

```
$ keytool -genseckey -alias jgroups -storetype JCEKS -keystore
jgroups.jceks
```

5. Use the SSL and JGroup keystore files to create the keystore secret for the project:

```
$ oc secret new datavirt-app-secret keystore.jks jgroups.jceks
```

6. Create a secret with the **datasources.env** file, an environment file containing the data sources and necessary resource adapters. This file needs to be stored locally. Either clone the **jboss-openshift/openshift-quickstarts** repository from GitHub, or download the environment file from:
https://github.com/jboss-openshift/openshift-quickstarts/blob/master/datavirt/dynamicvdb-datafederation/datasources.env

```
$ oc secrets new datavirt-app-config datasources.env
```

7. Link the keystore and environment secrets to the service account created earlier:

```
$ oc secrets link datavirt-service-account datavirt-app-secret
datavirt-app-config
```

5.1.2. Deployment

Use the OpenShift web console to configure a JDV for OpenShift application template with the quickstart details.

1. Log in to the OpenShift web console and select the *jdv-app-demo* project space.

2. Click **Add to Project** to list all of the default image streams and templates.

3. Use the **Filter by keyword** search bar to limit the list to those that match **`datavirt`** and select **datavirt63-secure-s2i**.

4. Specify the following:
SOURCE_REPOSITORY_URL: https://github.com/jboss-openshift/openshift-quickstarts
SOURCE_REPOSITORY_REF: master
CONTEXT_DIR: datavirt/dynamicvdb-datafederation/app

5. Click **Deploy**.

5.1.3. Defining Alternate Data Sources

The **dynamicvdb-datafederation** quickstart uses the **QS_DB_TYPE** environment variable to determine the data source to use. This value is set to **h2** by default. After the quickstart has been deployed, this variable can be added to the deployment configuration and modified so that an alternate data source is used.
For example, to modify the deployment configuration so that the deployment uses a MySQL data source:

```
$ oc env dc/datavirt-app QS_DB_TYPE=mysql5
```

The **dynamicvdb-datafederation** quickstart has been prepared with H2, Derby, MySQL, and PostgreSQL data sources.

5.2. EXAMPLE WORKFLOW: DEPLOYING THE JDG FOR OPENSHIFT AS A MATERIALIZATION TARGET QUICKSTART

This Example Worfklow continues from Example Workflow: Deploying a JDV Project Using the JDV for OpenShift Image, and presumes that the **dynamicvdb-datafederation** quickstart has been successfully deployed. In this example, a JDG for OpenShift instance will be prepared and deployed so that JDV for OpenShift can use it as a materialization target.

5.2.1. Preparing the Project to Deploy JDG for OpenShift

1. Ensure you are in the **jdv-app-demo** project:

   ```
   $ oc project jdv-app-demo
   ```

2. Create a service account for JDG for OpenShift:

   ```
   $ oc create serviceaccount datagrid-service-account
   ```

3. Add the view role to the service account. This enables the service account to view all the resources in the jdv-app-demo namespace, which is necessary for managing the cluster.

   ```
   $ oc policy add-role-to-user view system:serviceaccount:jdv-app-
   demo:datagrid-service-account
   ```

4. Use the JGroup keystore file (created in Example Workflow: Deploying a JDV Project Using the JDV for OpenShift Image to create the keystore secret for the project:

   ```
   $ oc secret new datagrid-app-secret jgroups.jceks
   ```

5. Link the keystore secret to the service account created earlier:

   ```
   $ oc secrets link datagrid-service-account datagrid-app-secret
   ```

5.2.2. Deploying JDG for OpenShift

A JDG for OpenShift instance needs to be deployed, with necessary caches specified, so that JDV for OpenShift can use it as a materialization target for the databases. Any of the JDG for OpenShift templates can be used, however non-persistent templates are recommended to gain clustering and high availability benefits. This example uses the **datagrid65-https** template to take advantage of SSL.

1. Log in to the OpenShift web console and select the *jdv-app-demo* project space.

2. Click **Add to Project** to list all of the default image streams and templates.

3. Use the **Filter by keyword** search bar to limit the list to those that match `datagrid`. You may need to click **See all** to show the desired application template.

4. Select the **datagrid65-https** template.

    a. In the **DATAVIRT_CACHE_NAMES** environment variable field, enter **`addressbook`**. This configures the three caches required (**addressbook**, **addressbook_staging**, and **addressbook_alias**) for **addressbook** to be used as a materialization cache for the database.

    b. Ensure the **CACHE_TYPE_DEFAULT** environment variable is set to **`replicated`**.

5. Click **Deploy**

5.2.3. Configure JDV for OpenShift to use JDG for OpenShift as a Materialization Target

After the JDG for OpenShift instance is running, modify the JDV for OpenShift deployment so that it uses JDG as a materialization target. Add the **DATAGRID_MATERIALIZATION** environment variable to the build configuration:

```
$ oc env bc/datavirt-app DATAGRID_MATERIALIZATION=true
```

CHAPTER 6. REFERENCE

6.1. GLOSSARY

DATA SOURCE

Repository for data. Query languages enable users to retrieve and manipulate data stored in these repositories.

MODULE

A module is a logical grouping of classes used for class loading and dependency management. Modules are defined in the **EAP_HOME**/**modules**/ directory of the application server. Each module exists as a subdirectory, for example **EAP_HOME**/**modules**/**org**/**apache**/. Each module directory then contains a slot subdirectory, which defaults to main and contains the module.xml configuration file and any required JAR files.
See the Red Hat JBoss Enterprise Application Platform Configuration Guide for more information on modules and module structure.

RESOURCE ADAPTERS

Deployable Java EE component that provides communication between a Java EE application and an Enterprise Information System (EIS) using the Java Connector Architecture (JCA) specification. A resource adapter is often provided by EIS vendors to allow easy integration of their products with Java EE applications.
An EIS can be any other software system within an organization. Examples include: Enterprise Resource Planning (ERP) systems, db systems, email servers, and proprietary messaging systems.

A resource adapter is packaged in a RAR (Resource Adapter Archive) file which can be deployed to JBoss EAP. A RAR file may also be included in an EAR (Enterprise Archive) deployment.

TRANSLATOR

Provides an abstraction layer between the query engine and the physical data source. This layer converts query commands into source specific commands and executes them using a resource adapter. The translator also converts the result data that comes from the physical source into the form that the query engine requires

VIRTUAL DATABASE (VDB)

A VDB is a container for components that integrate data from multiple disparate data sources, allowing applications to access and query the data as though it is a single database, using a single uniform API.
A VDB is composed of various data models and configuration information that describes which, and how, data sources are to be integrated. In particular, 'source models' are used to represent the structure and characteristics of the incorporated physical data sources, and 'view models' represent the structure and characteristics of the integrated data that is exposed to applications.

6.2. ARTIFACT REPOSITORY MIRRORS

A repository in Maven holds build artifacts and dependencies of various types (all the project jars, library jar, plugins or any other project specific artifacts). It also specifies locations from where to download artifacts from, while performing the S2I build. Besides using central repositories, it is a common practice for organizations to deploy a local custom repository (mirror).

Benefits of using a mirror are:

- Availability of a synchronized mirror, which is geographically closer and faster.
- Ability to have greater control over the repository content.
- Possibility to share artifacts across different teams (developers, CI), without the need to rely on public servers and repositories.
- Improved build times.

Often, a repository manager can serve as local cache to a mirror. Assuming that the repository manager is already deployed and reachable externally at ***http://10.0.0.1:8080/repository/internal/***, the S2I build can then use this manager by supplying the **`MAVEN_MIRROR_URL`** environment variable to the build configuration of the application as follows:

1. Identify the name of the build configuration to apply **`MAVEN_MIRROR_URL`** variable against:

```
oc get bc -o name
buildconfig/jdv
```

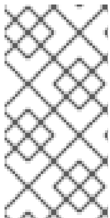
2. Update build configuration of **`jdv`** with a **`MAVEN_MIRROR_URL`** environment variable

```
oc env bc/jdv
MAVEN_MIRROR_URL="http://10.0.0.1:8080/repository/internal/"
buildconfig "jdv" updated
```

3. Verify the setting

```
oc env bc/jdv --list
# buildconfigs jdv
MAVEN_MIRROR_URL=http://10.0.0.1:8080/repository/internal/
```

4. Schedule new build of the application



NOTE

During application build, you will notice that Maven dependencies are pulled from the repository manager, instead of the default public repositories. Also, after the build is finished, you will see that the mirror is filled with all the dependencies that were retrieved and used during the build.