Red Hat JBoss Data Grid 7.1

Data Grid for OpenShift

Using Data Grid for OpenShift

Last Updated: 2018-06-05

Red Hat JBoss Data Grid 7.1 Data Grid for OpenShift

Using Data Grid for OpenShift

Legal Notice

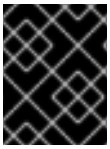
Abstract

Guide to using Data Grid for OpenShift image

Table of Contents

CHAPTER 1. INTRODUCTION

Red Hat JBoss Data Grid (JDG) is available as a containerized image that is designed for use with OpenShift. This image provides an in-memory distributed database so that developers can quickly access large amounts of data in a hybrid environment.



IMPORTANT

There are significant differences in supported configurations and functionality in the Data Grid for OpenShift image compared to the full release of JBoss Data Grid.

This topic details the differences between the JDG for OpenShift image and the full release of JBoss Data Grid, and provides instructions specific to running and configuring the JDG for OpenShift image. Documentation for other JBoss Data Grid functionality not specific to the JDG for OpenShift image can be found in the JBoss Data Grid documentation on the Red Hat Customer Portal .

CHAPTER 2. BEFORE YOU BEGIN

2.1. FUNCTIONALITY DIFFERENCES FOR JDG FOR OPENSHIFT IMAGES

There are several major functionality differences in the JDG for OpenShift image:

- The JBoss Data Grid Management Console is not available to manage JDG for OpenShift images.
- The JBoss Data Grid Management CLI is only bound locally. This means that you can only access the Management CLI of a container from within the pod.
- Library mode is not supported.
- Only JDBC is supported for a backing cache-store. Support for remote cache stores are present only for data migration purposes.

2.2. INITIAL SETUP

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

2.3. FORMING A CLUSTER USING THE JDG FOR OPENSHIFT IMAGES

Clustering is achieved through one of two discovery mechanisms: Kubernetes or DNS. This is accomplished by configuring the JGroups protocol stack in ***clustered-openshift.xml*** with either the ***<openshift.KUBE_PING/>*** or ***<openshift.DNS_PING/>*** elements. By default ***KUBE_PING*** is the pre-configured and supported protocol.

For ***KUBE_PING*** to work the following steps must be taken:

1. The ***OPENSHIFT_KUBE_PING_NAMESPACE*** environment variable must be set (as seen in the Configuration Environment Variables). If this variable is not set, then the server will act as if it is a single-node cluster, or a cluster that consists of only one node.
2. The ***OPENSHIFT_KUBE_PING_LABELS*** environment variable must be set (as seen in the Configuration Environment Variables). If this variable is not set, then pods outside the application (but in the same namespace) will attempt to join.
3. Authorization must be granted to the service account the pod is running under to be allowed to Kubernetes' REST api. This is done on the command line:

   Example 2.1. Policy commands

   Using the ***default*** service account in the ***myproject*** namespace:

   ```
   oc policy add-role-to-user view system:serviceaccount:$(oc project
   -q):default -n $(oc project -q)
   ```

   Using the ***eap-service-account*** in the ***myproject*** namespace:

   ```
   oc policy add-role-to-user view system:serviceaccount:$(oc project
   -q):eap-service-account -n $(oc project -q)
   ```

Once the above is configured images will automatically join the cluster as they are deployed; however, removing images from an active cluster, and therefore shrinking the cluster, is not supported.

2.4. ROLLING UPGRADES

In Red Hat JBoss Data Grid, rolling upgrades permit a cluster to be upgraded from one version to a new version without experiencing any downtime.

IMPORTANT

When performing a rolling upgrade it is recommended to not update any cache entries in the source cluster, as this may lead to data inconsistency.

2.4.1. Rolling Upgrades Using Hot Rod

IMPORTANT

Rolling upgrades on Red Hat JBoss Data Grid running in remote client-server mode, using the Hot Rod connector, **are working consistently** (allow seamless data migration with no downtime) **from version 6.6.2 through 7.1** . See:

- JDG-845 - Rolling Upgrade fixes
- JBoss Data Grid 7.0.1 Resolved Issues

for details.

Rolling upgrades on Red Hat JBoss Data Grid **from version 6.1 through 6.6.1, using the Hot Rod connector, are not working correctly yet**. See:

- JDG-831 - Rolling Upgrade from 6.1 to 7 not working
- JBoss Data Grid 7.1 Known Issues

for details.

This is a known issue in Red Hat JBoss Data Grid 7.1, and no workaround exists at this time.

Since JBoss Data Grid 6.5 for OpenShift image is based on version 6.5 of Red Hat JBoss Data Grid, **rolling upgrades from JBoss Data Grid 6.5 for OpenShift to JBoss Data Grid 7.1 for OpenShift, using the Hot Rod connector, are not possible without a data loss.**

2.4.2. Rolling Upgrades Using REST

See Example Workflow: Performing JDG rolling upgrade from JDG 6.5 for OpenShift image to JDG 7.1 for OpenShift image using the REST connector for an end-to-end example of performing JDG rolling upgrade using the REST connector.

2.5. ENDPOINTS

Clients can access JBoss Data Grid via REST, HotRod, and memcached endpoints defined as usual in the cache's configuration.

If a client attempts to access a cache via HotRod and is in the same project it will be able to receive the full cluster view and make use of consistent hashing; however, if it is in another project then the client will unable to receive the cluster view. Additionally, if the client is located outside of the project that contains the HotRod cache there will be additional latency due to extra network hops being required to access the cache.

IMPORTANT

Only caches with an exposed REST endpoint will be accessible outside of OpenShift.

2.6. CONFIGURING CACHES

A list of caches may be defined by the ***CACHE_NAMES*** environment variable. By default the following caches are created:

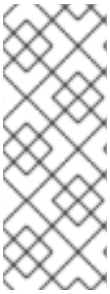
- ***default***
- ***memcached***

Each cache's behavior may be controlled through the use of cache-specific environment variables, with each environment variable expecting the cache's name as the prefix. For instance, consider the ***default*** cache, any configuration applied to this cache must begin with the ***DEFAULT_*** prefix. To define the number of cache entry owners for each entry in this cache the ***DEFAULT_CACHE_OWNERS*** environment variable would be used.

A full list of these is found at Cache Environment Variables.

2.6.1. Preserving Existing Content of the JBoss Data Grid Data Directory Across JDG for OpenShift Pod Restarts

The JBoss Data Grid server uses specified data directory for persistent data file storage (contains for example `___protobuf_metadata.dat` and `___script_cache.dat` files, or global state persistence configuration). When running on OpenShift, the data directory of the JBoss Data Grid server does not point to a persistent storage medium by default. This means the existing content of the data directory is deleted each time the JDG for OpenShift pod (the underlying JBoss Data Grid server) is restarted. To enable storing of data directory content to a persistent storage, deploy the JDG for OpenShift image using the ***datagrid71-partition*** application template with ***DATAGRID_SPLIT*** parameter set to ***true*** (default setting).



NOTE

Successful deployment of a JDG for OpenShift image using the ***datagrid71-partition*** template requires the ***${APPLICATION_NAME}-datagrid-claim*** persistent volume claim to be available, and the ***${APPLICATION_NAME}-datagrid-pvol*** persistent volume to be mounted at ***/opt/datagrid/standalone/partitioned_data*** path. See Persistent Storage Examples for guidance on how to deploy persistent volumes using different available plug-ins, and persistent volume claims.

2.7. DATASOURCES

Datasources are automatically created based on the value of some environment variables.

The most important variable is the ***DB_SERVICE_PREFIX_MAPPING*** which defines JNDI mappings for datasources. It must be set to a comma-separated list of **<*name*><database_type>=<PREFIX> triplet,**

where *_name_ is used as the pool-name in the datasource, ***database_type*** determines which database driver to use, and ***PREFIX*** is the prefix used in the names of environment variables, which are used to configure the datasource.

2.7.1. JNDI Mappings for Datasources

For each ***<name>-database_type>=PREFIX*** triplet in the ***DB_SERVICE_PREFIX_MAPPING*** environment variable, a separate datasource will be created by the launch script, which is executed when running the image.

The ***<database_type>*** will determine the driver for the datasource. Currently, only ***postgresql*** and ***mysql*** are supported.

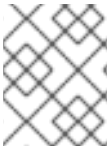
The ***<name>*** parameter can be chosen on your own. Do not use any special characters.

NOTE

The first part (before the equal sign) of the ***DB_SERVICE_PREFIX_MAPPING*** should be lowercase.

2.7.2. Database Drivers

The JDG for OpenShift image contains Java drivers for MySQL, PostgreSQL, and MongoDB databases deployed. Datasources are **generated only for MySQL and PostGreSQL databases .**



NOTE

For MongoDB databases there are no JNDI mappings created because this is not a SQL database.

2.7.3. Examples

The following examples demonstrate how datasources may be defined using the ***DB_SERVICE_PREFIX_MAPPING*** environment variable.

2.7.3.1. Single Mapping

Consider the value ***test-postgresql=TEST***.

This will create a datasource named ***java:jboss/datasources/test_postgresql***. Additionally, all of the required settings, such as username and password, will be expected to be provided as environment variables with the ***TEST_*** prefix, such as ***TEST_USERNAME*** and ***TEST_PASSWORD***.

2.7.3.2. Multiple Mappings

Multiple database mappings may also be specified; for instance, considering the following value for the ***DB_SERVICE_PREFIX_MAPPING*** environment variable: ***cloud-postgresql=CLOUD,test-mysql=TEST_MYSQL***.

NOTE

Multiple datasource mappings should be separated with commas, as seen in the above example.

This will create two datasources:

1. ***java:jboss/datasources/test_mysql***
2. ***java:jboss/datasources/cloud_postgresql***

MySQL datasource configuration, such as the username and password, will be expected with the ***TEST_MYSQL*** prefix, for example ***TEST_MYSQL_USERNAME***. Similarly the PostgreSQL datasource will expect to have environment variables defined with the ***CLOUD_*** prefix, such as ***CLOUD_USERNAME***.

2.7.4. Environment Variables

A full list of datasource environment variables may be found at Datasource Environment Variables.

2.8. SECURITY DOMAINS

To configure a new Security Domain the ***SECDOMAIN_NAME*** environment variable must be defined, which will result in the creation of a security domain named after the passed in value. This domain may be configured through the use of the Security Environment Variables.

2.9. MANAGING JDG FOR OPENSHIFT IMAGES

A major difference in managing an JDG for OpenShift image is that there is no Management Console exposed for the JBoss Data Grid installation inside the image. Because images are intended to be immutable, with modifications being written to a non-persistent file system, the Management Console is not exposed.

However, the JBoss Data Grid Management CLI (***JDG_HOME/bin/cli.sh***) is still accessible from within the container for troubleshooting purposes.

1. First open a remote shell session to the running pod:

```
$ oc rsh <pod_name>
```

2. Then run the following from the remote shell session to launch the JBoss Data Grid Management CLI:

```
$ /opt/datagrid/bin/cli.sh
```

WARNING

Any configuration changes made using the JBoss Data Grid Management CLI on a running container will be lost when the container restarts.

Making configuration changes to the JBoss Data Grid instance inside the JDG for OpenShift image is different from the process you may be used to for a regular release of JBoss Data Grid.

CHAPTER 3. GET STARTED

The Red Hat JBoss Data Grid images were automatically created during the installation of OpenShift along with the other default image streams and templates.

You can make changes to the JBoss Data Grid configuration in the image using either the S2I templates, or by using a modified JDG for OpenShift image.

3.1. USING THE JDG FOR OPENSHIFT IMAGE SOURCE-TO-IMAGE (S2I) PROCESS

The recommended method to run and configure the OpenShift JDG for OpenShift image is to use the OpenShift S2I process together with the application template parameters and environment variables.

The S2I process for the JDG for OpenShift image works as follows:

1. If there is a ***pom.xml*** file in the source repository, a Maven build is triggered with the contents of **`$MAVEN_ARGS`** environment variable.

2. By default the **`package`** goal is used with the **`openshift`** profile, including the system properties for skipping tests (**`-DskipTests`**) and enabling the Red Hat GA repository (**`-Dcom.redhat.xpaas.repo.redhatga`**).

3. The results of a successful Maven build are copied to ***JDG_HOME/standalone/deployments***. This includes all JAR, WAR, and EAR files from the directory within the source repository specified by **`$ARTIFACT_DIR`** environment variable. The default value of **`$ARTIFACT_DIR`** is the ***target*** directory.

    - Any JAR, WAR, and EAR in the ***deployments*** source repository directory are copied to the ***JDG_HOME/standalone/deployments*** directory.

    - All files in the ***configuration*** source repository directory are copied to ***JDG_HOME/standalone/configuration***.

    NOTE

    If you want to use a custom JBoss Data Grid configuration file, it should be named ***clustered-openshift.xml***.

4. All files in the ***modules*** source repository directory are copied to ***JDG_HOME/modules***.

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.

3.1.1. Using a Different JDK Version in the JDG for OpenShift image

The JDG for OpenShift image may come with multiple versions of OpenJDK installed, but only one is the default. For example, the JDG for OpenShift image comes with OpenJDK 1.7 and 1.8 installed, but OpenJDK 1.8 is the default.

If you want the JDG for OpenShift image to use a different JDK version than the default, you must:

- Ensure that your ***pom.xml*** specifies to build your code using the intended JDK version.

- In the S2I application template, configure the image's **JAVA_HOME** environment variable to point to the intended JDK version. For example:

```
{
  "name": "JAVA_HOME",
  "value": "/usr/lib/jvm/java-1.7.0"
}
```

3.2. USING A MODIFIED JDG FOR OPENSHIFT IMAGE

An alternative method is to make changes to the image, and then use that modified image in OpenShift.

The JBoss Data Grid configuration file that OpenShift uses inside the JDG for OpenShift image is ***JDG_HOME/standalone/configuration/clustered-openshift.xml***, and the JBoss Data Grid startup script is ***JDG_HOME/bin/openshift-launch.sh***.

You can run the JDG for OpenShift image in Docker, make the required configuration changes using the JBoss Data Grid Management CLI (***JDG_HOME/bin/jboss-cli.sh***), and then commit the changed container as a new image. You can then use that modified image in OpenShift.

IMPORTANT

It is recommended that you do not replace the OpenShift placeholders in the JDG for OpenShift image configuration file, as they are used to automatically configure services (such as messaging, datastores, HTTPS) during a container's deployment. These configuration values are intended to be set using environment variables.

NOTE

Ensure that you follow the guidelines for creating images.

3.3. BINARY BUILDS

To deploy existing applications on OpenShift, you can use the binary source capability.

See Example Workflow: Deploying binary build of EAP 6.4 / EAP 7.0 Infinispan application together with JDG for OpenShift image for an end-to-end example of a binary build.

CHAPTER 4. TUTORIALS

4.1. EXAMPLE WORKFLOW: DEPLOYING BINARY BUILD OF EAP 6.4 / EAP 7.0 INFINISPAN APPLICATION TOGETHER WITH JDG FOR OPENSHIFT IMAGE

The following example uses CarMart quickstart to deploy EAP 6.4 / EAP 7.0 Infinispan application, accessing a remote JBoss Data Grid server running in the same OpenShift project.

4.1.1. Prerequisite

1. Create a new project.

```
$ oc new-project jdg-bin-demo
```

NOTE

For brevity this example will not configure clustering. See dedicated section if data replication across the cluster is desired.

4.1.2. Deploy JBoss Data Grid 7.1 server

2. Identify the image stream for the JBoss Data Grid 7.1 image.

```
$ oc get is -n openshift | grep grid | cut -d ' ' -f 1
jboss-datagrid71-openshift
```

3. Deploy the server. Also specify the following:

   a. **carcache** as the name of application,

   b. A Hot Rod based connector, and

   c. **carcache** as the name of the Infinispan cache to configure.

```
$ oc new-app --name=carcache \
--image-stream=jboss-datagrid71-openshift \
-e INFINISPAN_CONNECTORS=hotrod \
-e CACHE_NAMES=carcache
--> Found image d83b4b2 (3 months old) in image stream
"openshift/jboss-datagrid71-openshift" under tag "latest" for
"jboss-datagrid71-openshift"

    JBoss Data Grid 7.1
    -------------------
    Provides a scalable in-memory distributed database designed
for fast access to large volumes of data.

    Tags: datagrid, java, jboss, xpaas

    * This image will be deployed in deployment config "carcache"
    * Ports 11211/tcp, 11222/tcp, 8080/tcp, 8443/tcp, 8778/tcp
will be load balanced by service "carcache"
```

```
      * Other containers can access this service through the
hostname "carcache"

--> Creating resources ...
    deploymentconfig "carcache" created
    service "carcache" created
--> Success
    Run 'oc status' to view your app.
```

4.1.3. Deploy binary build of EAP 6.4 / EAP 7.0 CarMart application

4. Clone the source code.

```
$ git clone https://github.com/jboss-openshift/openshift-
quickstarts.git
```

5. Configure the Red Hat JBoss Middleware Maven repository .

6. Build the **`datagrid/carmart`** application.

```
$ cd openshift-quickstarts/datagrid/carmart/
```

```
$ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------
--------------
[INFO] Building JBoss JDG Quickstart: carmart 1.2.0.Final
[INFO] ------------------------------------------------------------
--------------
...
[INFO] Building war: /tmp/openshift-
quickstarts/datagrid/carmart/target/jboss-carmart.war
[INFO] ------------------------------------------------------------
--------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------
--------------
[INFO] Total time: 3.360 s
[INFO] Finished at: 2017-06-27T19:11:46+02:00
[INFO] Final Memory: 34M/310M
[INFO] ------------------------------------------------------------
--------------
```

7. Prepare the directory structure on the local file system.
Application archives in the **deployments/** subdirectory of the main binary build directory are copied directly to the standard deployments folder of the image being built on OpenShift. For the application to deploy, the directory hierarchy containing the web application data must be correctly structured.

   Create main directory for the binary build on the local file system and **deployments/** subdirectory within it. Copy the previously built WAR archive for the **carmart** quickstart to the **deployments/** subdirectory:

```
$ ls
pom.xml  README.md  README-openshift.md  README-tomcat.md  src
target
```

```
$ mkdir -p jdg-binary-demo/deployments
```

```
$ cp target/jboss-carmart.war jdg-binary-demo/deployments/
```

NOTE

Location of the standard deployments directory depends on the underlying base image, that was used to deploy the application. See the following table:

Table 4.1. Standard Location of the Deployments Directory

Name of the Underlying Base Image(s)	Standard Location of the Deployments Directory
EAP for OpenShift 6.4 and 7.0	***$JBOSS_HOME/standalone/deployments***
Java S2I for OpenShift	***/deployments***
JWS for OpenShift	***$JWS_HOME/webapps***

8. Identify the image stream for EAP 6.4 / EAP 7.0 image.

```
$ oc get is -n openshift | grep eap | cut -d ' ' -f 1
jboss-eap64-openshift
jboss-eap70-openshift
```

9. Create new binary build, specifying image stream and application name.

```
$ oc new-build --binary=true \
--image-stream=jboss-eap64-openshift \
--name=eap-app
--> Found image 8fbf0f7 (2 months old) in image stream
"openshift/jboss-eap64-openshift" under tag "latest" for "jboss-
eap64-openshift"

    JBoss EAP 6.4
    -------------
    Platform for building and running JavaEE applications on JBoss
EAP 6.4

    Tags: builder, javaee, eap, eap6

    * A source build using binary input will be created
      * The resulting image will be pushed to image stream "eap-
app:latest"
      * A binary build was created, use 'start-build --from-dir' to
```

```
trigger a new build

--> Creating resources with label build=eap-app ...
    imagestream "eap-app" created
    buildconfig "eap-app" created
--> Success
```

NOTE

Specify **`jboss-eap70-openshift`** as the image stream name in the aforementioned command to use EAP 7.0 image for the application.

10. Start the binary build. Instruct **oc** executable to use main directory of the binary build we created in previous step as the directory containing binary input for the OpenShift build.

```
$ oc start-build eap-app --from-dir=jdg-binary-demo/ --follow
Uploading directory "jdg-binary-demo" as binary input for the build
...
build "eap-app-1" started
Receiving source from STDIN as archive ...
Copying all war artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all ear artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all rar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all jar artifacts from /home/jboss/source/. directory into
/opt/eap/standalone/deployments for later deployment...
Copying all war artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
'/home/jboss/source/deployments/jboss-carmart.war' ->
'/opt/eap/standalone/deployments/jboss-carmart.war'
Copying all ear artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
Copying all rar artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
Copying all jar artifacts from /home/jboss/source/deployments
directory into /opt/eap/standalone/deployments for later
deployment...
Pushing image 172.30.82.129:5000/jdg-bin-demo/eap-app:latest ...
Pushed 0/7 layers, 1% complete
Pushed 1/7 layers, 17% complete
Pushed 2/7 layers, 31% complete
Pushed 3/7 layers, 46% complete
Pushed 4/7 layers, 81% complete
Pushed 5/7 layers, 84% complete
Pushed 6/7 layers, 99% complete
Pushed 7/7 layers, 100% complete
Push successful
```

11. Create a new OpenShift application based on the build.

```
$ oc new-app eap-app
--> Found image ee25340 (3 minutes old) in image stream "jdg-bin-
demo/eap-app" under tag "latest" for "eap-app"

    jdg-bin-demo/eap-app-1:4bab3f63
    ------------------------------
    Platform for building and running JavaEE applications on JBoss
EAP 6.4

    Tags: builder, javaee, eap, eap6

    * This image will be deployed in deployment config "eap-app"
    * Ports 8080/tcp, 8443/tcp, 8778/tcp will be load balanced by
service "eap-app"
      * Other containers can access this service through the
hostname "eap-app"

--> Creating resources ...
    deploymentconfig "eap-app" created
    service "eap-app" created
--> Success
    Run 'oc status' to view your app.
```

12. Expose the service as route.

```
$ oc get svc -o name
service/carcache
service/eap-app
```

```
$ oc get route
No resources found.
```

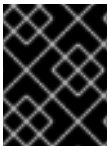
```
$ oc expose svc/eap-app
route "eap-app" exposed
```

```
$ oc get route
NAME      HOST/PORT                                        PATH
SERVICES   PORT       TERMINATION   WILDCARD
eap-app   eap-app-jdg-bin-demo.openshift.example.com
eap-app    8080-tcp                 None
```

13. Access the application.
    Access the CarMart application in your browser using the URL **http://eap-app-jdg-bin-demo.openshift.example.com/jboss-carmart**. You can view / remove existing cars (**Home** tab), or add a new car (**New car** tab).

4.2. EXAMPLE WORKFLOW: PERFORMING JDG ROLLING UPGRADE FROM JDG 6.5 FOR OPENSHIFT IMAGE TO JDG 7.1 FOR OPENSHIFT IMAGE USING THE REST CONNECTOR

The following example details the procedure to perform a rolling upgrade from JBoss Data Grid 6.5 for OpenShift image to JBoss Data Grid 7.1 for OpenShift image, using the REST connector.



IMPORTANT

When performing a rolling upgrade it is recommended to not update any cache entries in the source cluster, as this may lead to data inconsistency.

4.2.1. Start / Deploy the Source Cluster

A rolling upgrade to succeed, it assumes the Source Cluster with properties similar to the ones below:

- The name of the source JBoss Data Grid 6.5 cluster is **jdg65-cluster** and it has been deployed using the **datagrid65-basic** template or similar.
- The name of the replicated cache to synchronize its content during rolling upgrade is **clustercache**.
- The REST Infinispan connector has been configured for the application.
- The service name of the REST connector endpoint on JBoss Data Grid 6.5 cluster is **jdg65-cluster**.
- The **clustercache** replicated cache has been previously populated with some content to synchronize.

to be up and running.

NOTE

For demonstration purposes source JBoss Data Grid 6.5 for OpenShift cluster with aforementioned properties can be deployed running e.g. the following steps:

1. Create a dedicated OpenShift project.

```
$ oc new-project jdg-rest-rolling-upgrade-demo
```

2. Deploy source JBoss Data Grid 6.5 cluster with the REST connector enabled, utilizing replicated cache named **clustercache**.

```
$ oc new-app --template=datagrid65-basic \
-p APPLICATION_NAME=jdg65-cluster \
-p INFINISPAN_CONNECTORS=rest \
-p CACHE_NAMES=clustercache \
-e CLUSTERCACHE_CACHE_TYPE=replicated
--> Deploying template "openshift/datagrid65-basic" to
project jdg-rest-rolling-upgrade-demo

     datagrid65-basic
     ---------
     Application template for JDG 6.5 applications.


     * With parameters:
        * APPLICATION_NAME=jdg65-cluster
        * HOSTNAME_HTTP=
        * USERNAME=
        * PASSWORD=
        * IMAGE_STREAM_NAMESPACE=openshift
```

```
        * INFINISPAN_CONNECTORS=rest
        * CACHE_NAMES=clustercache
        * ENCRYPTION_REQUIRE_SSL_CLIENT_AUTH=
        * MEMCACHED_CACHE=default
        * REST_SECURITY_DOMAIN=
        * JGROUPS_CLUSTER_PASSWORD=kQiUcyhC # generated

--> Creating resources ...
    service "jdg65-cluster" created
    service "jdg65-cluster-memcached" created
    service "jdg65-cluster-hotrod" created
    route "jdg65-cluster" created
    deploymentconfig "jdg65-cluster" created
--> Success
    Run 'oc status' to view your app.
```

3. Populate **clustercache** with some content to synchronize later.

   a. Add some entries using JBoss Data Grid CLI.

      i. Given the following JBoss Data Grid CLI file to add cache entries non-interactively.

```
$ mkdir -p cache-entries
```

```
$ cat << EOD > cache-entries/cache-input.cli
cache clustercache
put key1 val1
put key2 val2
put key3 val3
put key4 val4
EOD
```

Please note a rolling upgrade will fail (BZ-1101512 - CLI UPGRADE command fails when testing data stored via CLI with REST encoding) **when storing cache data via CLI using the `--codec=rest` encoding parameter for the `put` commands in previous step.**

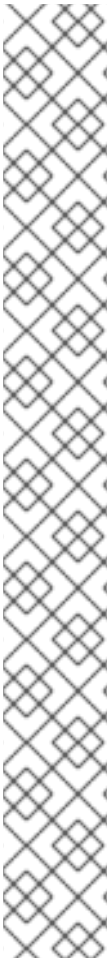
To overcome this issue, we do not specify codec to be used for encoding of cache entries (cache entries will be stored using the default **none** encoding).

      ii. Get the name of the JBoss Data Grid 6.5 pod.

```
$ export JDG65_POD=$(oc get pods -o name \
| grep -Po "[^/]+$" | grep "jdg65" \
| grep -v "deploy")
```

      iii. Copy the **`cache-input.cli`** file to JBoss Data Grid 6.5 pod.

```
$ oc rsync --no-perms=true ./cache-entries/
$JDG65_POD:/tmp
sending incremental file list
cache-input.cli
```



```
sent 182 bytes  received 40 bytes  444.00 bytes/sec
total size is 75  speedup is 0.34
```

iv. Add entries to **clustercache** by executing commands from **`cli-input.cli`** file.

```
$ oc rsh $JDG65_POD /opt/datagrid/bin/cli.sh \
--connect=localhost --file=/tmp/cache-input.cli
Picked up JAVA_TOOL_OPTIONS: -
Duser.home=/home/jboss -Duser.name=jboss
```

b. Add some entries directly via remote REST client.

i. Get the host value of the route for **jdg65-cluster**.

```
$ export JDG65_ROUTE=$(oc get routes \
--no-headers | grep "jdg65" | tr -s ' ' \
| cut -d ' ' -f2)
```

ii. Add example **`ExampleKey`** entry to **clustercache** using the REST endpoint remotely.

```
$ curl -X PUT -d "ExampleValue" \
$JDG65_ROUTE/rest/clustercache/ExampleKey
```

4.2.2. Deploy the Target Cluster

Perform the following to deploy JBoss Data Grid 7.1 cluster with the name of **jdg71-cluster**, using **datagrid71-basic** template, and **clustercache** as the name of the replicated cache to synchronize during the rolling upgrade:

IMPORTANT

The **datagrid71-basic** template uses ephemeral, in-memory datastore for the most frequently accessed data. Therefore any cache data, synchronized during a rolling upgrade will be available only during lifecycle of a JDG 7.1 pod (from rolling upgrade to pod restart). Use persistent templates (**datagrid71-mysql-persistent** or **datagrid71-postgresql-persistent**) to preserve the previously synchronized cache data across pod restarts.

```
$ oc new-app --template=datagrid71-basic \
-p APPLICATION_NAME=jdg71-cluster \
-p INFINISPAN_CONNECTORS=rest \
-p CACHE_NAMES=clustercache \
-p CACHE_TYPE_DEFAULT=replicated \
-p MEMCACHED_CACHE=""
--> Deploying template "openshift/datagrid71-basic" to project jdg-rest-
rolling-upgrade-demo

     Red Hat JBoss Data Grid 7.1 (Ephemeral, no https)
     ---------
     Application template for JDG 7.1 applications.
```

```
     A new data grid service has been created in your project. It supports
connector type(s) "rest".


     * With parameters:
        * Application Name=jdg71-cluster
        * Custom http Route Hostname=
        * Username=
        * Password=
        * ImageStream Namespace=openshift
        * Infinispan Connectors=rest
        * Cache Names=clustercache
        * Datavirt Cache Names=
        * Default Cache Type=replicated
        * Encryption Requires SSL Client Authentication?=
        * Memcached Cache Name=
        * REST Security Domain=
        * JGroups Cluster Password=3Aux1ORc # generated

--> Creating resources ...
    service "jdg71-cluster" created
    service "jdg71-cluster-memcached" created
    service "jdg71-cluster-hotrod" created
    route "jdg71-cluster" created
    deploymentconfig "jdg71-cluster" created
--> Success
    Run 'oc status' to view your app.
```

4.2.3. Configure REST Store for Caches on the Target Cluster

For each cache in the Target Cluster, intended to be synchronized during a rolling upgrade, configure a **RestCacheStore** with the following settings:

1. Ensure that the host and port values point to the Source Cluster.
2. Ensure that the path value points to the REST endpoint of the Source Cluster.

Given the following helper script to add a REST store to all replicated caches defined in **CACHE_NAMES** array:

```
$ mkdir -p update-cache
```

NOTE

Edit the definition of the **REST_SERVICE** variable below to match the name of the REST service endpoint for your environment. Also, edit the definition of **CACHE_NAMES** variable in the following helper script to contain names of all caches, that should be equipped with the definition of a REST store.

```
$ cat << \EOD > ./update-cache/add-rest-store-to-cache.sh
#!/bin/bash

export JDG_CONF=/opt/datagrid/standalone/configuration/clustered-
```

```
openshift.xml
export REST_SERVICE="jdg65-cluster"
read -r -d '' REST_STORE_ELEM << EOV
<rest-store path="/rest/cachename" shared="true" purge="false"
passivation="false">
<connection-pool connection-timeout="60000" socket-timeout="60000" tcp-no-
delay="true"/>
<remote-server outbound-socket-binding="remote-store-rest-server"/>
</rest-store>
EOV

declare -a CACHE_NAMES=("clustercache")

for CACHE in "${CACHE_NAMES[@]}"
do
  # Replace 'cachename' with actual cachename
  REST_STORE_ELEM=${REST_STORE_ELEM//cachename/${CACHE}}
  # Replace newline character with newline and two tabs (in escaped form)
  export REST_STORE_ELEM=${REST_STORE_ELEM//$'\n'/\\n\\t\\t}
  # sed pattern to locate cache definition
  CACHE_PATTERN="\(<replicated-cache[[:space:]]name=\"${CACHE}\"[^<]\+\)\
(</replicated-cache>\)"
  # Add REST store definition to cache entry
  sed -i "s#${CACHE_PATTERN}#\1\n\t\t${REST_STORE_ELEM}\n\2#g" $JDG_CONF
done

# sed pattern to locate host / port settings for REST connector
REST_HOST_PATTERN="\(<remote-destination host=\"\)remote-host\(\"
port=\"8080\"/>\)"
# Set host value to point to the Source Cluster
sed -i "s#${REST_HOST_PATTERN}#\1${REST_SERVICE}\2#g" $JDG_CONF
EOD
```

, perform the following:

1. Get the name of the JBoss Data Grid 7.1 pod.

```
$ export JDG71_POD=$(oc get pods -o name \
| grep -Po "[^/]+$" | grep "jdg71" | grep -v "deploy")
```

2. Copy the **add-rest-store-to-cache.sh** script to JBoss Data Grid 7.1 pod.

```
$ oc rsync --no-perms=true update-cache/ $JDG71_POD:/tmp
sending incremental file list

sent 71 bytes  received 11 bytes  54.67 bytes/sec
total size is 892  speedup is 10.88
```

3. Run the script to:

   a. Add REST store definition to each replicated cache from **CACHE_NAMES** array.

   b. Set host and port to point to the Source Cluster.

```
$ oc rsh $JDG71_POD /bin/bash /tmp/add-rest-store-to-cache.sh
```

4. Restart the JBoss Data Grid 7.1 server in order the corresponding caches to recognize the REST store configuration.

```
$ oc rsh $JDG71_POD /opt/datagrid/bin/cli.sh \
--connect ':reload'
Picked up JAVA_TOOL_OPTIONS: -Duser.home=/home/jboss -
Duser.name=jboss
{
    "outcome" => "success",
    "result" => undefined
}
```

WARNING

When restarting the server it is important to restart just the JBoss Data Grid process within the running container, not the whole container, since in the latter case the JBoss Data Grid container would be recreated with the default configuration from scratch, without the REST store(s) to be defined for specific cache(s).

4.2.4. Do Not Dump the Key Set During REST Rolling Upgrades

The REST rolling upgrades use case is designed to fetch all the data from the Source Cluster without using the **`recordKnownGlobalKeyset`** operation.

WARNING

Do not invoke the **`recordKnownGlobalKeyset`** operation for REST rolling upgrades. If you invoke this operation, it will cause data corruption and REST rolling upgrades will not complete successfully.

4.2.5. Synchronize Cache Data Using the REST Connector

Run the **`upgrade --synchronize=rest`** on the Target Cluster for all caches to be migrated. Optionally, use the **`--all`** switch to synchronize all caches in the cluster.

```
$ oc rsh $JDG71_POD /opt/datagrid/bin/cli.sh -c \
--commands='cd /subsystem=datagrid-infinispan/cache-container=clustered, \
cache clustercache,upgrade --synchronize=rest'
Picked up JAVA_TOOL_OPTIONS: -Duser.home=/home/jboss -Duser.name=jboss
ISPN019500: Synchronized 5 entries using migrator 'rest' on cache
'clustercache'
```

4.2.6. Use the Synchronized Data from the JBoss Data Grid 7.1 (Target) cluster

All the requested data have been just synchronized. You can now point the client application(s) to the Target Cluster.

1. Get the value of **key1** from the JBoss Data Grid 7.1 cache via CLI.

```
$ oc rsh $JDG71_POD /opt/datagrid/bin/cli.sh -c \
--commands='cd /subsystem=datagrid-infinispan/cache-
container=clustered, \
cache clustercache,get key1' \
| grep '"' | base64 -di; echo
val1
```

2. Get the value of **ExampleKey** from the JBoss Data Grid 7.1 cache via remote REST call.

   a. Get the value of JBoss Data Grid 7.1 route.

```
$ JDG71_ROUTE=$(oc get routes | grep jdg71 \
| tr -s ' ' | cut -d ' ' -f2)
```

   b. Get the value of **ExampleKey** via remote REST client.

```
$ curl -X GET \
$JDG71_ROUTE/rest/clustercache/ExampleKey; echo
ExampleValue
```

4.2.7. Disable the RestCacheStore on the Target Cluster

Once the Target Cluster has obtained all data from the Source Cluster, disable the **RestCacheStore** (for each cache it has been previously configured) on the Target Cluster using the following command:

```
$ oc rsh $JDG71_POD /opt/datagrid/bin/cli.sh -c \
--commands='cd /subsystem=datagrid-infinispan/cache-container=clustered, \
cache clustercache,upgrade --disconnectsource=rest'
Picked up JAVA_TOOL_OPTIONS: -Duser.home=/home/jboss -Duser.name=jboss
ISPN019501: Disconnected 'rest' migrator source on cache 'clustercache'
```

The Source Cluster can now be decommissioned.

CHAPTER 5. REFERENCE

5.1. 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/jdg
```

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

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

3. Verify the setting

```
oc env bc/jdg --list
# buildconfigs jdg
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.

5.2. INFORMATION ENVIRONMENT VARIABLES

The following information environment variables are designed to convey information about the image and should not be modified by the user:

Table 5.1. Information Environment Variables

Variable Name	Description	Value
JBOSS_DATAGRID_VERSION	The full release that the containerized image is based from.	***7.1.0.GA***
JBOSS_HOME	The directory where the JBoss distribution is located.	***/opt/datagrid***
JBOSS_IMAGE_NAME	Image name, same as ***Name*** label	***jboss-datagrid-7/datagrid71-openshift***
JBOSS_IMAGE_RELEASE	Image release, same as ***Release*** label	Example: ***dev***
JBOSS_IMAGE_VERSION	Image version, same as ***Version*** label	Example: ***1.2***
JBOSS_MODULES_SYSTEM_PKGS		***org.jboss.logmanager***
JBOSS_PRODUCT		***datagrid***
LAUNCH_JBOSS_IN_BACKGROUND	Allows the data grid server to be gracefully shutdown even when there is no terminal attached.	***true***

5.3. CONFIGURATION ENVIRONMENT VARIABLES

Configuration environment variables are designed to conveniently adjust the image without requiring a rebuild, and should be set by the user as desired.

Table 5.2. Configuration Environment Variables

Variable Name	Description	Example Value
ADMIN_GROUP	Comma-separated list of groups / roles to configure for the JDG user specified via the ***USERNAME*** variable.	Example: ***admin,___schema_manager,___script_manager***

Variable Name	Description	Example Value
CACHE_CONTAINER_START	Should this cache container be started on server startup, or lazily when requested by a service or deployment. Defaults to ***LAZY***	Example: ***EAGER***
CACHE_CONTAINER_STATISTICS	Determines if the cache container collects statistics. Disable for optimal performance. Defaults to ***true***.	Example: ***false***
CACHE_NAMES	List of caches to configure. Defaults to ***default, memcached***, and each defined cache will be configured as a distributed-cache with a mode of ***SYNC***.	Example: ***addressbook, addressbook_indexed***
CONTAINER_SECURITY_CUSTOM_ROLE_MAPPER_CLASS	Class of the custom principal to role mapper.	Example: ***com.acme.CustomRoleMapper***
CONTAINER_SECURITY_ROLE_MAPPER	Set a role mapper for this cache container. Valid values are: ***identity-role-mapper***, ***common-name-role-mapper***, ***cluster-role-mapper***, ***custom-role-mapper***.	Example: ***identity-role-mapper***
CONTAINER_SECURITY_ROLES	Define role names and assign permissions to them.	Example: ***admin=ALL***, ***reader=READ***, ***writer=WRITE***
DATAGRID_SPLIT	Allow multiple instances of JBoss Data Grid server to share the same persistent volume. If enabled (set to ***true***) each instance will use a separate area within the persistent volume as its data directory. Such persistent volume is required to be mounted at ***/opt/datagrid/standalone/partitioned_data*** path. Not set by default.	Example: ***true***
DB_SERVICE_PREFIX_MAPPING	Define a comma-separated list of datasources to configure.	Example: ***test-mysql=TEST_MYSQL***
DEFAULT_CACHE	Indicates the default cache for this cache container.	Example: ***addressbook***

Variable Name	Description	Example Value
ENCRYPTION_REQUIRE_SSL_CLIENT_AUTH	Whether to require client certificate authentication. Defaults to ***false***.	Example: ***true***
HOTROD_AUTHENTICATION	If defined the hotrod-connectors will be configured with authentication in the ***ApplicationRealm***.	Example: ***true***
HOTROD_ENCRYPTION	If defined the hotrod-connectors will be configured with encryption in the ***ApplicationRealm***.	Example: ***true***
HOTROD_SERVICE_NAME	Name of the OpenShift service used to expose HotRod externally.	Example: ***DATAGRID_APP_HOTROD***
INFINISPAN_CONNECTORS	Comma separated list of connectors to configure. Defaults to ***hotrod***, ***memcached***, ***rest***. Note that if authorization or authentication is enabled on the cache then memcached should be removed as this protocol is inherently insecure.	Example: ***hotrod***
JAVA_OPTS_APPEND	The contents of ***JAVA_OPTS_APPEND*** is appended to ***JAVA_OPTS*** on startup.	Example: ***-Dfoo=bar***
JGROUPS_CLUSTER_PASSWORD	A password to control access to JGroups. Needs to be set consistently cluster-wide. The image default is to use the ***OPENSHIFT_KUBE_PING_LABELS*** variable value; however, the JBoss application templates generate and supply a random value.	Example: ***miR0JaDR***
MEMCACHED_CACHE	The name of the cache to use for the Memcached connector.	Example: ***memcached***
OPENSHIFT_KUBE_PING_LABELS	Clustering labels selector.	Example: ***application=eap-app***

Variable Name	Description	Example Value
OPENSHIFT_KUBE_PING_NAMESPACE	Clustering project namespace.	Example: ***myproject***
PASSWORD	Password for the JDG user.	Example: ***p@ssw0rd***
REST_SECURITY_DOMAIN	The security domain to use for authentication and authorization purposes. Defaults to ***none*** (no authentication).	Example: ***other***
TRANSPORT_LOCK_TIMEOUT	Infinispan uses a distributed lock to maintain a coherent transaction log during state transfer or rehashing, which means that only one cache can be doing state transfer or rehashing at the same time. This constraint is in place because more than one cache could be involved in a transaction. This timeout controls the time to wait to acquire a distributed lock. Defaults to ***240000***.	Example: ***120000***
USERNAME	Username for the JDG user.	Example: ***openshift***

NOTE

HOTROD_ENCRYPTION is defined:

- If set to a non-empty string (e.g. ***true***), or
- If JDG for OpenShift image was deployed using some of the application templates allowing configuration of HTTPS (***datagrid71-https***, ***datagrid71-mysql***, ***datagrid71-mysql-persistent***, ***datagrid71-postgresql***, or ***datagrid71-postgresql-persistent***), and at the same time the ***HTTPS_NAME*** parameter is set when deploying that template.

5.4. CACHE ENVIRONMENT VARIABLES

The following environment variables all control behavior of individual caches; when defining these values for a particular cache substitute the cache's name for ***CACHE_NAME***.

Table 5.3. Cache Environment Variables

Variable Name	Description	Example Value

Variable Name	Description	Example Value
<CACHE_NAME>_CACHE_TYPE	Determines whether this cache should be distributed or replicated. Defaults to ***distributed***.	Example: ***replicated***
<CACHE_NAME>_CACHE_START	Determines if this cache should be started on server startup, or lazily when requested by a service or deployment. Defaults to ***LAZY***.	Example: ***EAGER***
<CACHE_NAME>_CACHE_BATCHING	Enables invocation batching for this cache. Defaults to ***false***.	Example: ***true***
<CACHE_NAME>_CACHE_STATISTICS	Determines whether or not the cache collects statistics. Disable for optimal performance. Defaults to ***true***.	Example: ***false***
<CACHE_NAME>_CACHE_MODE	Sets the clustered cache mode, ***ASYNC*** for asynchronous operations, or ***SYNC*** for synchronous operations.	Example: ***ASYNC***
<CACHE_NAME>_CACHE_QUEUE_SIZE	In ***ASYNC*** mode this attribute can be used to trigger flushing of the queue when it reaches a specific threshold. Defaults to ***0***, which disables flushing.	Example: ***100***
<CACHE_NAME>_CACHE_QUEUE_FLUSH_INTERVAL	In ***ASYNC*** mode this attribute controls how often the asynchronous thread runs to flush the replication queue. This should be a positive integer that represents thread wakeup time in milliseconds. Defaults to ***10***.	Example: ***20***
<CACHE_NAME>_CACHE_REMOTE_TIMEOUT	In ***SYNC*** mode the timeout, in milliseconds, used to wait for an acknowledgement when making a remote call, after which the call is aborted and an exception is thrown. Defaults to ***17500***.	Example: ***25000***

Variable Name	Description	Example Value
<CACHE_NAME>_CACHE_OWNERS	Number of cluster-wide replicas for each cache entry. Defaults to ***2***.	Example: ***5***
<CACHE_NAME>_CACHE_SEGMENTS	Number of hash space segments per cluster. The recommended value is ***10 * cluster size***. Defaults to ***80***.	Example: ***30***
<CACHE_NAME>_CACHE_L1_LIFESPAN	Maximum lifespan, in milliseconds, of an entry placed in the L1 cache. Defaults to ***0***, indicating that L1 is disabled.	Example: ***100***.
<CACHE_NAME>_CACHE_EVICTION_STRATEGY	Sets the cache eviction strategy. Available options are ***UNORDERED***, ***FIFO***, ***LRU***, ***LIRS***, and ***NONE*** (to disable eviction). Defaults to ***NONE***.	Example: ***FIFO***
<CACHE_NAME>_CACHE_EVICTION_MAX_ENTRIES	Maximum number of entries in a cache instance. If selected value is not a power of two the actual value will default to the least power of two larger than the selected value. A value of ***-1*** indicates no limit. Defaults to ***10000***.	Example: ***-1***
<CACHE_NAME>_CACHE_EXPIRATION_LIFESPAN	Maximum lifespan, in milliseconds, of a cache entry, after which the entry is expired cluster-wide. Defaults to ***-1***, indicating that the entries never expire.	Example: ***10000***
<CACHE_NAME>_CACHE_EXPIRATION_MAX_IDLE	Maximum idle time, in milliseconds, a cache entry will be maintained in the cache. If the idle time is exceeded, then the entry will be expired cluster-wide. Defaults to ***-1***, indicating that the entries never expire.	Example: ***10000***

Variable Name	Description	Example Value
<CACHE_NAME>_CACHE_EXPIRATION_INTERVAL	Interval, in milliseconds, between subsequent runs to purge expired entries from memory and any cache stores. If you wish to disable the periodic eviction process altogether, then set the interval to ***-1***. Defaults to ***5000***.	Example: ***-1***
<CACHE_NAME>_JDBC_STORE_TYPE	Type of JDBC store to configure. This value may either be ***string*** or ***binary***.	Example: ***string***
<CACHE_NAME>_JDBC_STORE_DATASOURCE	Defines the jndiname of the datasource.	Example: ***java:jboss/datasources/ExampleDS***
<CACHE_NAME>_KEYED_TABLE_PREFIX	Defines the prefix prepended to the cache name used when composing the name of the cache entry table. Defaults to ***ispn_entry***.	Example: ***JDG***
<CACHE_NAME>_CACHE_INDEX	The indexing mode of the cache. Valid values are ***NONE***, ***LOCAL***, and ***ALL***. Defaults to ***NONE***.	Example: ***ALL***
<CACHE_NAME>_INDEXING_PROPERTIES	Comma separated list of properties to pass on to the indexing system.	Example: ***default.directory_provider=ram***
<CACHE_NAME>_CACHE_SECURITY_AUTHORIZATION_ENABLED	Enables authorization checks for this cache. Defaults to ***false***.	Example: ***true***
<CACHE_NAME>_CACHE_SECURITY_AUTHORIZATION_ROLES	Sets the valid roles required to access this cache.	Example: ***admin, reader, writer***
<CACHE_NAME>_CACHE_PARTITION_HANDLING_ENABLED	If enabled, then the cache will enter degraded mode when it loses too many nodes. Defaults to ***true***.	Example: ***false***

5.5. DATASOURCE ENVIRONMENT VARIABLES

Datasource properties may be configured with the following environment variables:

Table 5.4. Datasource Environment Variables

Variable Name	Description	Example Value
<NAME>_<DATABASE_TYPE>_SERVICE_HOST	Defines the database server's hostname or IP to be used in the datasource's ***connection_url*** property.	Example: ***192.168.1.3***
<NAME>_<DATABASE_TYPE>_SERVICE_PORT	Defines the database server's port for the datasource.	Example: ***5432***
<PREFIX>_BACKGROUND_VALIDATION	When set to ***true*** database connections are validated periodically in a background thread prior to use. Defaults to ***false*** (***<validate-on-match>*** method is enabled by default instead).	Example: ***true***
<PREFIX>_BACKGROUND_VALIDATION_MILLIS	Specifies frequency of the validation (in miliseconds), when the ***<background-validation>*** database connection validation mechanism is enabled (***<PREFIX>_BACKGROUND_VALIDATION*** variable is set to***true***). Defaults to ***10000***.	Example: ***20000***
<PREFIX>_CONNECTION_CHECKER	Specifies a connection checker class that is used to validate connections for the particular database in use.	Example: ***org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker***
<PREFIX>_DATABASE	Defines the database name for the datasource.	Example: ***myDatabase***
<PREFIX>_DRIVER	Defines Java database driver for the datasource.	Example: ***postgresql***
<PREFIX>_EXCEPTION_SORTER	Specifies the exception sorter class that is used to properly detect and clean up after fatal database connection exceptions.	Example: ***org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter***
<PREFIX>_JNDI	Defines the JNDI name for the datasource. Defaults to ***java:jboss/datasources/<name>_<database_type>***, where ***name*** and ***database_type*** are taken from the triplet definition. This setting is useful if you want to override the default generated JNDI name.	Example: ***java:jboss/datasources/test-postgresql***

Variable Name	Description	Example Value
<PREFIX>_JTA	Defines Java Transaction API (JTA) option for the non-XA datasource (XA datasource are already JTA capable by default). Defaults to ***true***.	Example: ***false***
<PREFIX>_MAX_POOL_SIZE	Defines the maximum pool size option for the datasource.	Example: ***20***
<PREFIX>_MIN_POOL_SIZE	Defines the minimum pool size option for the datasource.	Example: ***1***
<PREFIX>_NONXA	Defines the datasource as a non-XA datasource. Defaults to ***false***.	Example: ***true***
<PREFIX>_PASSWORD	Defines the password for the datasource.	Example: ***password***
<PREFIX>_TX_ISOLATION	Defines the java.sql.Connection transaction isolation level for the database.	Example: ***TRANSACTION_READ_UNCOMMITTED***
<PREFIX>_URL	Defines connection URL for the datasource.	Example: ***jdbc:postgresql://localhost:5432/postgresdb***
<PREFIX>_USERNAME	Defines the username for the datasource.	Example: ***admin***

5.6. SECURITY ENVIRONMENT VARIABLES

The following environment variables may be defined to customize the environment's security domain:

Table 5.5. Security Environment Variables

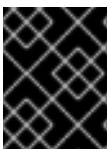
Variable Name	Description	Example Value
SECDOMAIN_NAME	Define in order to enable the definition of an additional security domain.	Example: ***myDomain***
SECDOMAIN_PASSWORD_STACKING	If defined, the password-stacking module option is enabled and set to the value ***useFirstPass***.	Example: ***true***

Variable Name	Description	Example Value
SECDOMAIN_LOGIN_MODULE	The login module to be used. Defaults to ***UsersRoles***.	Example: ***UsersRoles***
SECDOMAIN_USERS_PROPERTIES	The name of the properties file containing user definitions. Defaults to ***users.properties***.	Example: ***users.properties***
SECDOMAIN_ROLES_PROPERTIES	The name of the properties file containing role definitions. Defaults to ***roles.properties***.	Example: ***roles.properties***

5.7. EXPOSED PORTS

The following ports are exposed by default in the JDG for OpenShift Image:

Value	Description
8443	Secure Web
8778	-
11211	memcached
11222	internal hotrod
11333	external hotrod



IMPORTANT

The external hotrod connector is only available if the ***HOTROD_SERVICE_NAME*** environment variables has been defined.

5.8. TROUBLESHOOTING

In addition to viewing the OpenShift logs, you can troubleshoot a running JDG for OpenShift Image container by viewing its logs. These are outputted to the container's standard out, and are accessible with the following command:

```
$ oc logs -f <pod_name> <container_name>
```

NOTE

By default, the OpenShift JDG for OpenShift Image does not have a file log handler configured. Logs are only sent to the container's standard out.