Red Hat JBoss Data Grid 6.5

Data Grid for OpenShift

Using Data Grid for OpenShift
Using Data Grid for OpenShift
Abstract

Guide to using Data Grid for OpenShift image
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. FUNCTIONAL DIFFERENCES FOR JDG FOR OPENSIFT 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 instructions 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 OPENSIFT 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. 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.

2.5. 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. 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.

2.6.1. 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.

### 2.6.2. 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.

### 2.6.3. Examples

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

#### 2.6.3.1. 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`.

#### 2.6.3.2. 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`. 
### 2.6.4. Environment Variables

A full list of datasource environment variables may be found at [Datasource Environment Variables](#).

### 2.7. 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](#).

### 2.8. MANAGING JDG FOR OPENSLET 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/jboss-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/jboss-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.

---

[link:#Making-Configuration-Changes-Data-Grid] 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 OPENSIFT 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:

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

### 3.2. USING A MODIFIED JDG FOR OPENSIFT 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.

#### 3.3.1. Deploy 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.

**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.
Deploy JBoss Data Grid 6.5 server:

2. Identify the image stream for the JBoss Data Grid 6.5 image.

```bash
$ oc get is -n openshift | grep grid | cut -d ' ' -f 1
jboss-datagrid65-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.

```bash
$ oc new-app --name=carcache \
   --image-stream=jboss-datagrid65-openshift \
   -e INFINISPAN_CONNECTORS=hotrod \
   -e CACHE_NAMES=carcache

--> Found image d83b4b2 (3 months old) in image stream
"openshift/jboss-datagrid65-openshift" under tag "latest" for
"jboss-datagrid65-openshift"

JBoss Data Grid 6.5
-------------------
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.
```

Deploy binary build of EAP 6.4 / EAP 7.0 CarMart application:

4. Clone the source code.

```bash
$ 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.

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

```bash
$ 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 3.1. Standard Location of the Deployments Directory

<table>
<thead>
<tr>
<th>Name of the Underlying Base Image(s)</th>
<th>Standard Location of the Deployments Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAP for OpenShift 6.4 and 7.0</td>
<td>$JBOS_HOME/standalone/deployments</td>
</tr>
<tr>
<td>Java S2I for OpenShift</td>
<td>/deployments</td>
</tr>
<tr>
<td>JWS for OpenShift</td>
<td>$JWS_HOME/webapps</td>
</tr>
</tbody>
</table>

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
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.

   ```bash
   $ 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.

   ```bash
   $ 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).
CHAPTER 4. REFERENCE

4.1. 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
   buildconfig "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.

4.2. 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 4.1. Information Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>JBOSS_DATAGRID_VERSION</td>
<td>The full release that the containerized image is based from.</td>
<td>6.5.1.GA</td>
</tr>
<tr>
<td>JBOSS_HOME</td>
<td>The directory where the JBoss distribution is located.</td>
<td>/opt/datagrid</td>
</tr>
<tr>
<td>JBOSS_IMAGE_NAME</td>
<td>Image name, same as Name label</td>
<td>jboss-datagrid-6/datagrid65-openshift</td>
</tr>
<tr>
<td>JBOSS_IMAGE_RELEASE</td>
<td>Image release, same as Release label</td>
<td>Example: dev</td>
</tr>
<tr>
<td>JBOSS_IMAGE_VERSION</td>
<td>Image version, same as Version label</td>
<td>Example: 1.2</td>
</tr>
<tr>
<td>JBOSS_MODULES_SYSTEM_PKGS</td>
<td></td>
<td>org.jboss.logmanager</td>
</tr>
<tr>
<td>JBOSS_PRODUCT</td>
<td></td>
<td>datagrid</td>
</tr>
<tr>
<td>LAUNCH_JBOSS_IN_BACKGROUND</td>
<td>Allows the data grid server to be gracefully shutdown even when there is no terminal attached.</td>
<td>true</td>
</tr>
</tbody>
</table>

### 4.3. 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 4.2. Configuration Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CACHE_CONTAINER_START</td>
<td>Should this cache container be started on server startup, or lazily when requested by a service or deployment. Defaults to LAZY</td>
<td>Example: EAGER</td>
</tr>
<tr>
<td>CACHE_CONTAINER_STATISTICS</td>
<td>Determines if the cache container collects statistics. Disable for optimal performance. Defaults to true.</td>
<td>Example: false</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CACHE_NAMES</td>
<td>List of caches to configure. Defaults to <code>default, memcached</code>, and each defined cache will be configured as a distributed-cache with a mode of <code>SYNC</code>.</td>
<td>Example: <code>addressbook, addressbook_indexed</code></td>
</tr>
<tr>
<td>CONTAINER_SECURITY_CUSTOM_ROLE_MAPPER_CLASS</td>
<td>Class of the custom principal to role mapper.</td>
<td>Example: <code>com.acme.CustomRoleMapper</code></td>
</tr>
<tr>
<td>CONTAINER_SECURITY_IDENTITY_ROLE_MAPPER</td>
<td>Set a role mapper for this cache container. Valid values are: <code>identity-role-mapper, common-name-role-mapper, cluster-role-mapper, custom-role-mapper</code>.</td>
<td>Example: <code>identity-role-mapper</code></td>
</tr>
<tr>
<td>CONTAINER_SECURITY_ROLES</td>
<td>Define role names and assign permissions to them.</td>
<td>Example: <code>admin=ALL, reader=READ, writer=WRITE</code></td>
</tr>
<tr>
<td>DB_SERVICE_PREFIX_MAPPING</td>
<td>Define a comma-separated list of datasources to configure.</td>
<td>Example: <code>test-mysql=TEST_MYSQL</code></td>
</tr>
<tr>
<td>DEFAULT_CACHE</td>
<td>Indicates the default cache for this cache container.</td>
<td>Example: <code>addressbook</code></td>
</tr>
<tr>
<td>ENCRYPTION_REQUIRE_SSL_CLIENT_AUTH</td>
<td>Whether to require client certificate authentication. Defaults to <code>false</code>.</td>
<td>Example: <code>true</code></td>
</tr>
<tr>
<td>HOTROD_AUTHENTICATION</td>
<td>If defined the hotrod-connectors will be configured with authentication in the <code>ApplicationRealm</code>.</td>
<td>Example: <code>true</code></td>
</tr>
<tr>
<td>HOTROD_ENCRYPTION</td>
<td>If defined the hotrod-connectors will be configured with encryption in the <code>ApplicationRealm</code>.</td>
<td>Example: <code>true</code></td>
</tr>
<tr>
<td>HOTROD_SERVICE_NAME</td>
<td>Name of the OpenShift service used to expose HotRod externally.</td>
<td>Example: <code>DATAGRID_APP_HOTROD</code></td>
</tr>
<tr>
<td>INFINISPAN_CONNECTORS</td>
<td>Comma separated list of connectors to configure. Defaults to <code>hotrod, memcached, rest</code>. Note that if authorization or authentication is enabled on the cache then memcached should be removed as this protocol is inherently insecure.</td>
<td>Example: <code>hotrod</code></td>
</tr>
</tbody>
</table>
### Variable Name | Description | Example Value
--- | --- | ---
**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`

**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.

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 (datagrid65-https, datagrid65-mysql, datagrid65-mysql-persistent, datagrid65-postgresql, or datagrid65-postgresql-persistent), and at the same time the HTTPS_NAME parameter is set when deploying that template.

4.4. 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 4.3. Cache Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;CACHE_NAME&gt;_CACHE_TYPE</td>
<td>Determines whether this cache should be distributed or replicated. Defaults to distributed.</td>
<td>Example: replicated</td>
</tr>
<tr>
<td>&lt;CACHE_NAME&gt;_CACHE_START</td>
<td>Determines if this cache should be started on server startup, or lazily when requested by a service or deployment. Defaults to LAZY.</td>
<td>Example: EAGER</td>
</tr>
<tr>
<td>&lt;CACHE_NAME&gt;_CACHE_BATCHING</td>
<td>Enables invocation batching for this cache. Defaults to false.</td>
<td>Example: true</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>---------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_STATISTICS</code></td>
<td>Determines whether or not the cache collects statistics. Disable for optimal performance. Defaults to <code>true</code>.</td>
<td>Example: <code>false</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_MODE</code></td>
<td>Sets the clustered cache mode, <code>ASYNC</code> for asynchronous operations, or <code>SYNC</code> for synchronous operations.</td>
<td>Example: <code>ASYNC</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_QUEUE_SIZE</code></td>
<td>In <code>ASYNC</code> mode this attribute can be used to trigger flushing of the queue when it reaches a specific threshold. Defaults to <code>0</code>, which disables flushing.</td>
<td>Example: <code>100</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_QUEUE_FLUSH_INTERVAL</code></td>
<td>In <code>ASYNC</code> 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 <code>10</code>.</td>
<td>Example: <code>20</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_REMOTE_TIMEOUT</code></td>
<td>In <code>SYNC</code> 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 <code>17500</code>.</td>
<td>Example: <code>25000</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_OWNERS</code></td>
<td>Number of cluster-wide replicas for each cache entry. Defaults to <code>2</code>.</td>
<td>Example: <code>5</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_SEGMENTS</code></td>
<td>Number of hash space segments per cluster. The recommended value is <code>10 * cluster size</code>. Defaults to <code>80</code>.</td>
<td>Example: <code>30</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_L1_LIFESPAN</code></td>
<td>Maximum lifespan, in milliseconds, of an entry placed in the L1 cache. Defaults to <code>0</code>, indicating that L1 is disabled.</td>
<td>Example: <code>100</code>.</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_EVICTION_STRATEGY</code></td>
<td>Sets the cache eviction strategy. Available options are <code>UNORDERED</code>, <code>FIFO</code>, <code>LRU</code>, <code>LIRS</code>, and <code>NONE</code> (to disable eviction). Defaults to <code>NONE</code>.</td>
<td>Example: <code>FIFO</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_EVICTION_MAX_ENTRIES</code></td>
<td>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 <code>-1</code> indicates no limit. Defaults to <code>10000</code>.</td>
<td>Example: <code>-1</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_EXPIRATION_LIFESPAN</code></td>
<td>Maximum lifespan, in milliseconds, of a cache entry, after which the entry is expired cluster-wide. Defaults to <code>-1</code>, indicating that the entries never expire.</td>
<td>Example: <code>10000</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_EXPIRATION_MAX_IDLE</code></td>
<td>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 <code>-1</code>, indicating that the entries never expire.</td>
<td>Example: <code>10000</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_CACHE_EXPIRATION_INTERVAL</code></td>
<td>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 <code>-1</code>. Defaults to <code>5000</code>.</td>
<td>Example: <code>-1</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_JDBC_STORE_TYPE</code></td>
<td>Type of JDBC store to configure. This value may either be <code>string</code> or <code>binary</code>.</td>
<td>Example: <code>string</code></td>
</tr>
<tr>
<td><code>&lt;CACHE_NAME&gt;_JDBC_STORE_DATASOURCE</code></td>
<td>Defines the jndiname of the datasource.</td>
<td>Example: <code>java:jdbc/ExampleDS</code></td>
</tr>
</tbody>
</table>
### 4.5. DATASOURCE ENVIRONMENT VARIABLES

Datasource properties may be configured with the following environment variables:

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;NAME&gt;_&lt;DATABASE_TYPE&gt;_SERVICE_HOST</code></td>
<td>Defines the database server’s hostname or IP to be used in the datasource’s <code>connection_url</code> property.</td>
<td><code>192.168.1.3</code></td>
</tr>
<tr>
<td><code>&lt;NAME&gt;_&lt;DATABASE_TYPE&gt;_SERVICE_PORT</code></td>
<td>Defines the database server’s port for the datasource.</td>
<td><code>5432</code></td>
</tr>
<tr>
<td>Variable Name</td>
<td>Description</td>
<td>Example Value</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td><code>&lt;PREFIX&gt;_BACKGROUND_VALIDATION</code></td>
<td>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).</td>
<td>Example: true</td>
</tr>
<tr>
<td><code>&lt;PREFIX&gt;_BACKGROUND_VALIDATION_MILLIS</code></td>
<td>Specifies frequency of the validation (in milliseconds), when the <code>&lt;background-validation&gt;</code> database connection validation mechanism is enabled ( <code>&lt;PREFIX&gt;_BACKGROUND_VALIDATION</code> variable is set to true). Defaults to 10000.</td>
<td>Example: 20000</td>
</tr>
<tr>
<td><code>&lt;PREFIX&gt;_CONNECTION_CHECKER</code></td>
<td>Specifies a connection checker class that is used to validate connections for the particular database in use.</td>
<td>Example: org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker</td>
</tr>
<tr>
<td><code>&lt;PREFIX&gt;_DATABASE</code></td>
<td>Defines the database name for the datasource.</td>
<td>Example: myDatabase</td>
</tr>
<tr>
<td><code>&lt;PREFIX&gt;_DRIVER</code></td>
<td>Defines Java database driver for the datasource.</td>
<td>Example: postgresql</td>
</tr>
<tr>
<td><code>&lt;PREFIX&gt;_EXCEPTION_SORTER</code></td>
<td>Specifies the exception sorter class that is used to properly detect and clean up after fatal database connection exceptions.</td>
<td>Example: org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter</td>
</tr>
<tr>
<td><code>&lt;PREFIX&gt;_JNDI</code></td>
<td>Defines the JNDI name for the datasource. Defaults to java:jboss/datasources/&lt;name&gt;_&lt;database_type&gt;, 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.</td>
<td>Example: java:jboss/datasources/test-postgresql</td>
</tr>
</tbody>
</table>
### 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*

### 4.6. SECURITY ENVIRONMENT VARIABLES

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

#### Table 4.5. Security Environment Variables

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SECDOMAIN_NAME</strong></td>
<td>Define in order to enable the definition of an additional security domain.</td>
<td>Example: <em>myDomain</em></td>
</tr>
<tr>
<td><strong>SECDOMAIN_PASSWORD_STACKING</strong></td>
<td>If defined, the password-stacking module option is enabled and set to the value <strong>useFirstPass</strong>.</td>
<td>Example: <em>true</em></td>
</tr>
</tbody>
</table>
### Variable Name

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Description</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SECDOMAIN_LOGIN_MODULE</td>
<td>The login module to be used. Defaults to <em>UsersRoles</em>.</td>
<td>Example: <em>UsersRoles</em></td>
</tr>
<tr>
<td>SECDOMAIN_USERS_PROPERTIES</td>
<td>The name of the properties file containing user definitions. Defaults to <em>users.properties</em>.</td>
<td>Example: <em>users.properties</em></td>
</tr>
<tr>
<td>SECDOMAIN_ROLES_PROPERTIES</td>
<td>The name of the properties file containing role definitions. Defaults to <em>roles.properties</em>.</td>
<td>Example: <em>roles.properties</em></td>
</tr>
</tbody>
</table>

### 4.7. EXPOSED PORTS

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

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>8443</td>
<td>Secure Web</td>
</tr>
<tr>
<td>8778</td>
<td>-</td>
</tr>
<tr>
<td>11211</td>
<td>memcached</td>
</tr>
<tr>
<td>11222</td>
<td>internal hotrod</td>
</tr>
<tr>
<td>11333</td>
<td>external hotrod</td>
</tr>
</tbody>
</table>

**IMPORTANT**

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

### 4.8. 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.