Red Hat JBoss Data Grid 7.2

Data Grid for OpenShift

Developing and deploying Red Hat JBoss Data Grid for OpenShift

Last Updated: 2019-06-10

Red Hat JBoss Data Grid 7.2 Data Grid for OpenShift

Developing and deploying Red Hat JBoss Data Grid for OpenShift

Legal Notice

Abstract

Develop, test, and deploy JBoss Data Grid on Red Hat OpenShift.

Table of Contents

CHAPTER 1. INTRODUCTION

Red Hat JBoss Data Grid is available as a containerized image that you can deploy and use in OpenShift.

1.1. THE JBOSS DATA GRID FOR OPENSHIFT IMAGE

If you have deployed JBoss Data Grid on other platforms, as either a server or embedded library, you should note some differences with the Red Hat JBoss Data Grid for OpenShift image.

- The JBoss Data Grid Management Console is not available in OpenShift.
- The JBoss Data Grid Management CLI is accessible only in the pod where the application runs.
- Library mode is not supported.
- Only JDBC is supported as a cache store.

1.2. JBOSS DATA GRID DOCUMENTATION

Red Hat Data Grid documentation is available on the Red Hat Customer Portal.

1.3. VERSION INFORMATION

Find new features, enhancements, and bug fixes for JBoss Data Grid for OpenShift.

Red Hat Software Update	Description	Image Version
RHBA-2018:1831	Initial release of JBoss Data Grid for OpenShift 7.2.	1.0
RHBA-2018:2383	Errata fix for CVE-2018-10897.	1.0
RHBA-2018:2558	Cumulative patch for JBoss Data Grid 7.2.2.	1.1
RHEA-2018:2730	Cumulative patch that adds support for: - Conflict resolution parameters. - Red Hat terms-based registry (*registry.redhat.io*).	1.2
RHBA-2018:2736	Cumulative patch for JBoss Data Grid 7.2.3 that includes OpenJDK CVEs from RHSA-2018:2943.	1.2

Red Hat Software Update	Description	Image Version
RHEA-2018:3585	Cumulative patch that adds support for: - Custom configuration with JBoss Data Grid for OpenShift images. - JGroups **ASYM_ENCRYPT** protocol.	1.3
RHBA-2019:0293	Patch that fixes systemd CVEs from RHSA-2019:0049.	1.3
RHBA-2019:40142	Patch that fixes systemd CVEs from RHSA-2019:0368.	1.3

CHAPTER 2. AUTHENTICATING WITH THE RED HAT CONTAINER CATALOG

The Red Hat Container Catalog, *registry.redhat.io*, requires authentication to access JBoss Data Grid for OpenShift images and resources.

You can use the following authentication mechanisms:

Credentials

The username and password for your Red Hat customer account. These credentials let you pull resources from *registry.redhat.io* from a single host with the **docker login** command. You can also use these credentials to create service accounts and generate authentication tokens.

Registry Service Account Token

A randomly generated string that you use to authenticate multiple systems.
From a high level, do the following to get an authentication token:

1. Log in to *registry.redhat.io*.
2. Create a new **Registry Service Account** if necessary.
3. Generate tokens as required.

2.1. SETTING UP AUTHENTICATION WITH SERVICE ACCOUNT TOKENS

After you generate a service account token, do the following to set up authentication:

1. Navigate to your registry service account.
2. Select the **Docker Login** tab and copy the command.
3. Run the **docker login** command on each host system that pulls from *registry.redhat.io*.
4. Verify the token is added to the Docker configuration file.

```
$ cat ~/.docker/config.json
...
"registry.redhat.io": {
  "auth": "MTEwMDkx..."
 }
```

2.1.1. Adding Tokens to Pull Secrets

To pull secured container images that are not available on the internal registry for OpenShift Container Platform, create a pull secret from your Docker configuration file and add it to your service account as follows:

1. Log in to OpenShift.

```
$ oc login -u username -p password
```

2. Select your working project.

```
$ oc project myproject
```

3. Create the pull secret.

   ```
   $ oc create secret generic pull-secret-name \
     --from-file=.dockerconfigjson=path/to/.docker/config.json \
     --type=kubernetes.io/dockerconfigjson
   ```

4. Link the pull secret to your service account. This step lets you pull images from the secure registry to the pod.

   ```
   $ oc secrets link default pull-secret-name --for=pull
   ```

5. Mount the secret in the pod so that you can pull build images.

   ```
   $ oc secrets link builder pull-secret-name
   ```

For more information, including troubleshooting procedures, see Red Hat Container Registry Authentication.

CHAPTER 3. GETTING STARTED WITH RED HAT JBOSS DATA GRID FOR OPENSHIFT

JBoss Data Grid provides an JBoss Data Grid for OpenShift image stream and set of templates to help you quickly get up and running with JBoss Data Grid deployments on Red Hat OpenShift.

datagrid72-image-stream

Image stream for JBoss Data Grid.

datagrid72-basic

Run JBoss Data Grid for OpenShift without the need to create OpenShift Secrets.

datagrid72-https

Run JBoss Data Grid for OpenShift with an HTTPS route to securely access caches. Requires a **JKS** keystore in an OpenShift secret.

datagrid72-mysql

Run JBoss Data Grid for OpenShift with a MySQL database as an ephemeral cache store. Requires a **JKS** keystore in an OpenShift secret.

datagrid72-mysql-persistent

Run JBoss Data Grid for OpenShift with a MySQL database as a persistent cache store. Requires a **JKS** keystore in an OpenShift secret.

datagrid72-postgresql

Run JBoss Data Grid for OpenShift with a PostgreSQL database as an ephemeral cache store. Requires a **JKS** keystore in an OpenShift secret.

datagrid72-postgresql-persistent

Run JBoss Data Grid for OpenShift with a PostgreSQL database as a persistent cache store. Requires a **JKS** keystore in an OpenShift secret.

datagrid72-partition

Run JBoss Data Grid for OpenShift with a partitioned data directory that preserves metadata for cache entries when the pod restarts. Requires the **DATAGRID_SPLIT** environment variable. See Configuration Environment Variables.

3.1. IMPORTING JBOSS DATA GRID FOR OPENSHIFT IMAGE TEMPLATES

The first step to using the JBoss Data Grid for OpenShift image templates is to import them into OpenShift as follows:

1. On your master host(s), log in as a cluster administrator or a user with project administrator access to the **openshift** namespace.

   ```
   $ oc login -u system:admin
   ```

2. Import a specific template or all templates.

   - Import a specific template:

     ```
     $ oc create -n openshift -f \
     https://raw.githubusercontent.com/jboss-container-images/jboss-datagrid-7-openshift-image/1.3/templates/datagrid72-mysql.json
     ```

- Import all templates:

```
$ for resource in datagrid72-image-stream.json \
  datagrid72-basic.json \
  datagrid72-https.json \
  datagrid72-mysql-persistent.json \
  datagrid72-mysql.json \
  datagrid72-partition.json \
  datagrid72-postgresql.json \
  datagrid72-postgresql-persistent.json
do
  oc create -n openshift -f \
  https://raw.githubusercontent.com/jboss-container-images/jboss-datagrid-7-openshift-
image/1.3/templates/${resource}
done
```

TIP

Use the **oc create** command to import a new template. Use the **oc replace --force** command to overwrite an existing template.

3. Verify the templates are available on OpenShift.

```
$ oc get templates -n openshift | grep datagrid72
```

3.1.1. Working with the JBoss Data Grid for OpenShift Image

Importing the JBoss Data Grid for OpenShift image templates also imports the **jboss-datagrid72-openshift** image. When you create a new application from a template, or instantiate a template, you deploy the image in a pod that uses the configuration settings from the template.

In this way, the **jboss-datagrid72-openshift** image is a general purpose build of JBoss Data Grid. Each template configures the image for specific purposes.

3.1.1.1. Viewing Information about the JBoss Data Grid for OpenShift Image

Run the following command after you import the image templates to view the available image streams for JBoss Data Grid for OpenShift:

```
$ oc get is -n openshift | grep datagrid
```

The **oc get** command shows the **jboss-datagrid72-openshift** image stream is available in the **openshift** namespace. This image stream defines the JBoss Data Grid container image as an available resource for creating deployments.

Run the following command to view information about the **jboss-datagrid72-openshift** image stream:

```
$ oc describe is jboss-datagrid72-openshift -n openshift
```

The **oc describe** command shows the tags for the **jboss-datagrid72-openshift** image stream as well as the location for the container image in the registry.

3.1.1.2. Importing the JBoss Data Grid for OpenShift Image

You can optionally import the JBoss Data Grid for OpenShift image into the **openshift** namespace separately to the templates.

To import the JBoss Data Grid for OpenShift image, run the following command:

```
$ oc -n openshift import-image jboss-datagrid72-openshift:1.3
```

NOTE

JBoss Data Grid for OpenShift templates use the global **openshift** namespace as the default for the **jboss-datagrid72-openshift** image stream. You can set the **IMAGE_STREAM_NAMESPACE** environment variable to import templates in a different namespace or project. However you must also ensure that an image stream is available in that namespace.

3.1.2. Importing OpenShift Secrets

You must import or create OpenShift secrets that contain HTTPS and JGroups keystores before you can instantiate templates that require authentication.

JBoss Data Grid for OpenShift provides an example HTTPS and JGroups keystore that you can import as an OpenShift secret. However, this secret is intended for evaluation purposes only. You should not use it in production environments.

Do the following to import the example secret into your project namespace:

```
$ oc create \
  -f https://raw.githubusercontent.com/jboss-openshift/application-templates/master/secrets/datagrid-app-secret.json
```

For more information about creating secrets to secure network traffic, see Securing Network Traffic.

3.2. CONFIGURING JBOSS DATA GRID FOR OPENSHIFT DEPLOYMENTS

You configure JBoss Data Grid for OpenShift deployments with environment variables that you can set:

- on the command line when you create new applications from templates.
- in templates that you import into OpenShift projects. You can then create pre-configured deployments from those templates.

You can also set environment variables through the OpenShift Web Console. See the relevant OpenShift documentation.

3.2.1. Getting Started with Image Configuration

Run the following command to show the **datagrid72-basic** template:

```
$ oc describe template datagrid72-basic -n openshift
```

The output of the **oc describe** command shows information about the template as well as the parameters that are set in the template. When you instantiate the **datagrid72-basic** template, those parameters configure the following objects:

- **Service** defines a logical set of pods and access policies.
- **Route** exposes services externally to pods.
- **Deployment Configuration** configures triggers and replicas for the replication controller; also configures pod templates that contain exposed ports for services, environment variables for the image, and so on.

As an example, the output of the **oc describe** command shows the following template parameters that set credentials and name caches:

```
Parameters:

  Name:  USERNAME
  Display Name: Username
  Description: Data Grid username.
  Required: false
  Value: <none>

  Name:  PASSWORD
  Display Name: Password
  Description: Password for the Data Grid user.
  Required: false
  Value: <none>

  Name:  CACHE_NAMES
  Display Name: Cache Names
  Description: Comma-separated list of caches to create.
  Required: false
  Value: <none>
```

The output of the **oc describe** command shows the services, routes, and deployment configuration that the **datagrid72-basic** template configures:

```
Objects:
    Service  ${APPLICATION_NAME}
    Service  ${APPLICATION_NAME}-memcached
    Service  ${APPLICATION_NAME}-hotrod
    Service  ${APPLICATION_NAME}-ping
    Route  ${APPLICATION_NAME}
    DeploymentConfig ${APPLICATION_NAME}
```

When you instantiate the **datagrid72-basic** template, the launch script sets those parameters as environment variables for the image in the deployment configuration.

3.2.2. Setting Parameters on the Command Line

Learn how to set parameters for JBoss Data Grid deployments on the command line.

Complete the following steps to:

- Instantiate the **datagrid72-basic** template to create a new JBoss Data Grid for OpenShift deployment.
- Set parameters that:
  - Define credentials to access the cache over HTTPS and Hot Rod.
  - Create a cache named **mycache**.
  - Configure the cache to start eagerly.

3.2.2.1. Instantiating the Template

1. Create a new project.

```
$ oc new-project datagrid-env --display-name="Setting Environment Variables"
```

2. Deploy a new application with the **datagrid72-basic** template. Use the **-e** option to pass parameter and value pairs.
   a. Specify a username: **-e USERNAME=developer**
   b. Specify a password: **-e PASSWORD=<value>**
   The password cannot be the same as the username or **root**, **admin**, or, **administrator**. It must contain at least 8 characters, 1 alphabetic character(s), 1 digit(s), and 1 non-alphanumeric symbol(s).
   c. Create a cache named 'mycache': **-e CACHE_NAMES=mycache**
   d. Configure the cache to start eagerly: **-e MYCACHE_CACHE_START=EAGER**

```
$ oc new-app --template=datagrid72-basic --name=rhdg \
  -e USERNAME=developer -e PASSWORD=******** \
  -e CACHE_NAMES=mycache -e MYCACHE_CACHE_START=EAGER
```

3. Check the application status.

```
$ oc status
```

3.2.2.2. Listing Environment Variables

1. Retrieve the available pods in the project.

```
$ oc get pods

NAME                    READY     STATUS    RESTARTS   AGE
datagrid-app-1-<id>       0/1       Running   1          1m
datagrid-app-1-deploy   1/1       Running   0          1m
```

2. List environment variables for the pod named **datagrid-app-1-<id>**. Where **<id>** is a randomly generated string such as **67q5h**.

```
$ oc env pods/datagrid-app-1-<id> --list
```

```
# pods datagrid-app-1-<id>, container datagrid-app
CACHE_NAMES=mycache
MYCACHE_CACHE_START=EAGER
PASSWORD=********
USERNAME=developer
...
```

3.2.2.3. Changing Environment Variables

1. Change the deployment configuration so that the cache starts lazily.

```
$ oc env dc/datagrid-app -e MYCACHE_CACHE_START=LAZY
```

   This command triggers the replication controller to deploys a new version of the application.

2. Retrieve the updated list of pods.

```
$ oc get pods

NAME                  READY     STATUS    RESTARTS   AGE
datagrid-app-2-<id>     0/1       Running   0          58s
datagrid-app-2-deploy   1/1       Running   0          59s
```

3. List environment variables for the pod named **datagrid-app-2-<id>**.

```
$ oc env pods/datagrid-app-2-<id> --list

# pods datagrid-app-2-<id>, container datagrid-app
CACHE_NAMES=mycache
MYCACHE_CACHE_START=LAZY
PASSWORD=********
USERNAME=developer
...
```

3.2.3. Modifying JBoss Data Grid for OpenShift Image Templates

Learn how to set parameters for JBoss Data Grid deployments in reusable image templates.

Complete the following steps to:

- Export the **datagrid72-basic** template from Red Hat OpenShift.
- Modify the **datagrid72-basic** template to set parameters that:
    - Define credentials to access the cache over HTTPS and Hot Rod.
    - Create a cache named **mycache**.
    - Configure the cache to start eagerly.
- Import the modified template and instantiate it.

3.2.3.1. Exporting the Template

1. On your master host(s), log in as a cluster administrator or a user with project administrator access to the **openshift** namespace.

   ```
   $ oc login -u system:admin
   ```

2. Export the **datagrid72-basic** template to a file named **datagrid72-extended**.

   TIP

   You can export templates with any filename to your home (~/) directory.

   ```
   $ oc export template datagrid72-basic -n openshift > datagrid72-extended
   ```

3.2.3.2. Modifying the Template

1. Open the exported **datagrid72-extended** file with any text editor.

   TIP

   Templates define the deployment configuration in **yaml** or **json** format.

2. In the **labels** section, change the template label to **datagrid72-extended**.

   ```
   labels:
     template: datagrid72-extended
   ```

3. In the **metadata** section, change the template name to **datagrid72-extended**.

   ```
   metadata:
     name: datagrid72-extended
   ```

4. In the **parameters** section, add values for the **USERNAME**, **PASSWORD**, **CACHE_NAMES**, and **<CACHE_NAME>_CACHE_START** environment variables.

   ```
   parameters:
   - description: Data Grid username.
     displayName: Username
     name: USERNAME
     value: developer

   - description: Password for the Data Grid user.
     displayName: Password
     name: PASSWORD
     value: ********

   - description: Comma-separated list of caches to configure.
     displayName: Cache Names
     name: CACHE_NAMES
     value: mycache

   - description: Configures the cache to start eagerly or lazily.
     displayName: Cache Start
   ```

```
  name: MYCACHE_CACHE_START
  required: false
  value: EAGER
```

5. Add an 'env' definition for the **<CACHE_NAME>_CACHE_START** environment variable to the deployment configuration.

```
spec:
  containers:
    -env:
      -name: MYCACHE_CACHE_START
       value: ${MYCACHE_CACHE_START}
```

6. Save and close the **datagrid72-extended** file.

3.2.3.3. Importing and Instantiating the Modified Template

Import the modified template into the **openshift** namespace.

```
$ oc create -n openshift -f datagrid72-extended
```

After you import the modified template, instantiate it and then list environment variables for the deployed pod.

```
$ oc new-app --template=datagrid72-extended

$ oc status

$ oc get pods

$ oc env pods/datagrid-app-1-<id> --list

# pods datagrid-app-1-<id>, container datagrid-app
CACHE_NAMES=mycache
MYCACHE_CACHE_START=EAGER
PASSWORD=********
USERNAME=developer
...
```

3.3. INVOKING CACHE OPERATIONS THROUGH THE REST ENDPOINT

JBoss Data Grid provides a REST endpoint through which you can invoke cache operations using standard HTTP methods. The REST endpoint is available by default without the need for configuration.

Complete the following steps to:

- Create a new project and instantiate the **datagrid72-basic** template.
- Invoke cache operations with the HTTP **GET**, **POST**, and **DELETE** methods.

3.3.1. Creating a Project and Instantiate a Template

1. Log in to OpenShift.

```
$ oc login -u developer
```

2. Create a new project.

```
$ $ oc new-project datagrid --display-name="RHDG REST Example"
```

3. Instantiate the **datagrid72-basic** template.

```
$ oc new-app --template=datagrid72-basic --name=rhdg
```

3.3.2. Examining Deployed Services

1. View the deployment status.

```
$ oc status
```

The **oc status** command shows a **datagrid-app** HTTP service.

```
In project RHDG REST Example (datagrid) on server https://192.0.2.0:8443

http://datagrid-app-datagrid.192.0.2.0.nip.io (svc/datagrid-app)
  dc/datagrid-app deploys openshift/jboss-datagrid72-openshift:1.3
    deployment
```

2. Show details about the **datagrid-app** route.

```
$ oc describe route datagrid-app
```

The **oc describe route** command shows the route where the HTTP service is exposed.

```
Name:   datagrid-app
Namespace:  datagrid
Created:  4 minutes ago
Labels:   app=rhdg
   application=datagrid-app
   template=datagrid72-basic
   xpaas=<version>
Description:  Route for application's HTTP service.
Annotations:  openshift.io/generated-by=OpenShiftNewApp
   openshift.io/host.generated=true
Requested Host:  datagrid-app-datagrid.192.0.2.0.nip.io
     exposed on router router 4 minutes ago
```

3. Note the hostname and IP address for the route. In the following command examples, you must substitute **192.0.2.0** with the correct IP address for your route to the REST endpoint.

3.3.3. Invoking a Get Operation on the Cache

1. Attempt to get a value for a key named **a** from a cache named **default**.

```
$ curl -i -H "Accept:application/json" \
http://rhdgroute-datagrid.192.0.2.0.nip.io/rest/default/a
```

The key named **a** does not exist in the cache named **default**. As a result, you get an HTTP 404 error.

```
HTTP/1.1 404 Not Found
content-length: 0
Set-Cookie: 3abf86065a054efa9e7658b871f83223=b78127f864341eb60be6916d847b8b06;
path=/; HttpOnly
Cache-control: private
```

3.3.4. Inserting and Retrieving an Entry in the Cache

1. Insert a JSON formatted entry in a key named **a** into the cache named **default**.

```
$ curl -X POST -i -H "Content-type:application/json" \
-d "{\"name\":\"Red Hat Data Grid\"}" \
http://rhdgroute-datagrid.192.0.2.0.nip.io/rest/default/a
```

2. Get the value of the key that you inserted.

```
$ curl -i -H "Accept:application/json" \
http://rhdgroute-datagrid.192.0.2.0.nip.io/rest/default/a
```

You get an HTTP 200 response that contains the key value you set.

```
HTTP/1.1 200 OK
etag: 1187661430
last-modified: <time-stamp>
content-type: application/json
content-length: 34
Set-Cookie: 3abf86065a054efa9e7658b871f83223=b78127f864341eb60be6916d847b8b06;
path=/; HttpOnly
Cache-control: private

"{\"name\":\"Red Hat Data Grid\"}"
```

3.3.5. Deleting the Entry from the Cache

1. Delete the key named **a**.

```
$ curl -X DELETE -i \
http://rhdgroute-datagrid.192.0.2.0.nip.io/rest/default/a
```

2. Attempt to retrieve the key value again.

```
$ curl -i -H "Accept:application/json" \
http://rhdgroute-datagrid.192.0.2.0.nip.io/rest/default/a
```

You get an HTTP 404 error because you deleted the key.

CHAPTER 4. CONFIGURING CLUSTERING

The JBoss Data Grid for OpenShift images can use either the Kubernetes or DNS discovery mechanisms for clustering. These discovery mechanisms enable images to automatically join clusters.

By default, DNS is pre-configured in the JBoss Data Grid for OpenShift image templates. If you want to use Kubernetes as the discovery mechanism, or if you plan to build and deploy a custom image, you must configure cluster discovery.

NOTE

JBoss Data Grid does not support removing images from an active cluster.

4.1. CONFIGURING THE KUBERNETES DISCOVERY MECHANISM

To configure the Kubernetes discovery mechanism for clustering, do the following:

1. Set **openshift.KUBE_PING** as the value for the **JGROUPS_PING_PROTOCOL** environment variable.

   ```
   JGROUPS_PING_PROTOCOL=openshift.KUBE_PING
   ```

2. Specify the OpenShift project name as the value for the **OPENSHIFT_KUBE_PING_NAMESPACE** environment variable. If you do not set this variable, the server behaves like a single-node cluster.

   ```
   OPENSHIFT_KUBE_PING_NAMESPACE=PROJECT_NAME
   ```

3. Specify the label that is set at the service level as the value for the **OPENSHIFT_KUBE_PING_LABELS** environment variable. If you do not set this variable, pods outside the application but in the same namespace attempt to join.

   ```
   OPENSHIFT_KUBE_PING_LABELS=app=APP_NAME
   ```

4. Grant authorization to the service account the pod is running under so that it can access the Kubernetes REST API. You grant this authorization using the OpenShift CLI, as follows: Granting authorization for 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)
   ```

   Granting authorization for ***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)
   ```

5. Ensure port **8888** is defined as a ping port on the pod container, as follows:

   ```
   ports:
     - containerPort: 8888
       name: ping
       protocol: TCP
   ```

4.2. CONFIGURING THE DNS DISCOVERY MECHANISM

To configure the DNS discovery mechanism for clustering, do the following:

1. Set **openshift.DNS_PING** as the value for the **JGROUPS_PING_PROTOCOL** environment variable.

   ```
   JGROUPS_PING_PROTOCOL=openshift.DNS_PING
   ```

2. Specify the name of the ping service for the cluster as the value for the **OPENSHIFT_DNS_PING_SERVICE_NAME** environment variable.

   ```
   OPENSHIFT_DNS_PING_SERVICE_NAME=PING_SERVICE_NAME
   ```

3. Specify the port number where the ping service is exposed as the value for the **OPENSHIFT_DNS_PING_SERVICE_PORT** environment variable. The default value is **8888**.

   ```
   OPENSHIFT_DNS_PING_SERVICE_PORT=PING_PORT
   ```

4. Define a ping service that exposes the ping port, as in the following example:

   ```
   apiVersion: v1
   kind: Service
   spec:
     clusterIP: None
     ports:
       - name: ping
         port: 8888
         protocol: TCP
         targetPort: 8888
     selector: deploymentConfig=datagrid-app
   metadata:
     annotations:
       description: The JGroups ping port for clustering.
       service.alpha.kubernetes.io/tolerate-unready-endpoints: 'true'
   ```

IMPORTANT

You should configure **clusterIP: None** so that the service is headless. Likewise, the ping port must be named and include the **service.alpha.kubernetes.io/tolerate-unready-endpoints: 'true'** annotation.

CHAPTER 5. SECURING NETWORK TRAFFIC

Encrypt client to server and server to server traffic to secure network communication.

5.1. ENCRYPTING CLIENT TO SERVER COMMUNICATION

JBoss Data Grid for OpenShift uses **JKS** keystores that contain credentials and certificates to secure client-to-server traffic.

To encrypt client to server communication, do the following:

1. Create a **JKS** keystore (**.jks**) to encrypt traffic.
   You can use OpenSSL and the Java keytool to generate a **JKS** keystore. When you generate a TLS certificate for the keystore, specify the domain name for the deployment.

   > **IMPORTANT**
   >
   > Production environments should aways use TLS certificates signed by a verified certificate authority (CA).

2. Deploy the **JKS** keystore to OpenShift as a secret.

   a. Log in as the developer user.

   ```
   $ oc login -u developer
   ```

   b. Create a secret for the **JKS** keystore. For example, to create a secret named **jdg-https-secret** from a keystore named **jdg-https.jks**, do the following:

   ```
   $ oc create secret generic jdg-https-secret --from-file=jdg-https.jks
   ```

   c. Link the secret to the service account for your deployment. For example, to link a secret named **jdg-https-secret** to the default service account, do the following:

   ```
   $ oc secrets link default jdg-https-secret
   ```

3. Configure your deployment to use the **JKS** keystore with these environment variables:

   HOSTNAME_HTTP

   Specifies the HTTP service route for the deployment. Required only if you are using a JBoss Data Grid for OpenShift template.

   HOSTNAME_HTTPS

   Sets the HTTPS service route for the deployment. Required only if you are using a JBoss Data Grid for OpenShift template.

   HTTPS_SECRET

   Matches the OpenShift secret for the keystore. Required only if you are using a JBoss Data Grid for OpenShift template.

   HTTPS_KEYSTORE

   Specifies the **JKS** keystore for encrypting server to client traffic.

   HTTPS_NAME

Matches the username for the keystore.

HTTPS_PASSWORD

Matches the keystore password.

HTTPS_KEYSTORE_DIR

Specifies the directory that contains the **JKS** keystore. You do not need to set this environment variable if you are using a JBoss Data Grid for OpenShift template. The templates set this environment variable by default.

TIP

Use the **HOTROD_ENCRYPTION** environment variable to configure the Hot Rod connector to use encryption. See Endpoint Configuration.

5.2. ENCRYPTING TRAFFIC BETWEEN CLUSTERED SERVERS

JBoss Data Grid for OpenShift uses JGroups technology to secure traffic between clustered servers with the following options:

Authentication

Uses the JGroups **AUTH** protocol that requires nodes to authenticate with a password when joining the cluster.
You configure authentication with the **JGROUPS_CLUSTER_PASSWORD** environment variable. This environment variable sets a password for nodes to use when joining the cluster. The password must be the same across the cluster.

Symmetric encryption

Uses the JGroups **SYM_ENCRYPT** protocol to secure traffic with a JGroups keystore (**.jceks**). This is the default encryption protocol.
The JGroups **AUTH** protocol is optional with symmetric encryption.

The JGroups keystore contains credentials that each node in the cluster uses to secure communication.

Asymmetric encryption

Uses the JGroups **ASYM_ENCRYPT** protocol to secure traffic with public/private key encryption.
The JGroups **AUTH** protocol is required with asymmetric encryption.

The coordinator node generates a secret key. When a node joins the cluster, it requests the secret key from the coordinator and provides its public key. The coordinator encrypts the secret key with the public key and returns it to the node. The node then decrypts and installs the secret so that it can securely communicate with other nodes in the cluster.

5.2.1. Setting Up Symmetric Encryption

To use symmetric encryption, do the following:

1. Create a JGroups keystore (**.jceks**) that contains credentials to encrypt traffic.
   You can use the Java keytool to generate a JGroups keystore.

2. Deploy the JGroups keystore to OpenShift as a secret.
   a. Log in as the developer user.

      ```
      $ oc login -u developer
      ```

   b. Create a secret for the JGroups keystore. For example, to create a secret named **jgroups-secret** from a keystore named **jgroups.jceks**, do the following:

      ```
      $ oc create secret generic jgroups-secret --from-file=jgroups.jceks
      ```

   c. Link the secret to the default service account.

      ```
      $ oc secrets link default jgroups-secret
      ```

3. Configure your deployment to use the JGroups keystore with these environment variables:

   JGROUPS_ENCRYPT_KEYSTORE

   Specifes the JGroups keystore for encrypting cluster traffic.

   JGROUPS_ENCRYPT_SECRET

   Matches the OpenShift secret for the keystore.

   JGROUPS_ENCRYPT_NAME

   Matches the username for the keystore.

   JGROUPS_ENCRYPT_PASSWORD

   Matches the keystore password.

   JGROUPS_ENCRYPT_KEYSTORE_DIR

   Specifies the directory where the JGroups keystore resides. You do not need to set this environment variable if you are using a JBoss Data Grid for OpenShift template. The templates set this environment variable by default.

4. If required, set a password for nodes to use when joining the cluster. with the **JGROUPS_CLUSTER_PASSWORD** environment variable.

5.2.2. Setting Up Asymmetric Encryption

To use asymmetric encryption, do the following:

1. Configure authentication with the **JGROUPS_CLUSTER_PASSWORD** environment variable.

2. Set the value of the **JGROUPS_ENCRYPT_PROTOCOL** environment variable to **ASYM_ENCRYPT**.

CHAPTER 6. CONFIGURING PERSISTENT DATASOURCES

JBoss Data Grid lets you persist data stored in the cache to a datasource. There are two types of datasources for Red Hat JBoss Data Grid for OpenShift:

- Internal datasources that run on OpenShift. These datasources are available through the Red Hat Container Registry and do not require you to configure additional environment files.

  NOTE

  Internal datasources include PostgreSQL, MySQL, and MongoDB. However, Red Hat JBoss Data Grid for OpenShift currently supports PostgreSQL and MySQL only.

- External datasources that do not run on OpenShift. You must configure these external datasources with environment files that you add to OpenShift Secrets.

6.1. CONFIGURING INTERNAL DATASOURCES

The ***DB_SERVICE_PREFIX_MAPPING*** environment variable defines JNDI mappings for internal datasources.

You can define multiple JNDI mappings as comma-separated values for the ***DB_SERVICE_PREFIX_MAPPING*** environment variable. When you run the JBoss Data Grid for OpenShift image, the launch script creates a separate datasource for each JNDI mapping. The JBoss Data Grid for OpenShift then automatically discovers each datasource.

To define a JNDI mapping, specify a value for the environment variable in the following format:

<poolname>-<database_type>=<PREFIX>

- **<poolname>** is the **pool-name** attribute for the datasource. Use any alphanumeric value that is meaningful and easy to identify. The value cannot contain special characters. Likewise, the value must contain lowercase characters only.

- **<database_type>** specifies the database driver to use. The value must contain lowercase characters only.

  NOTE

  Only **mysql** and **postgresql** are supported values for **<database_type>**.

- **<PREFIX>** is used for the names of environment variables that configure the datasource.

6.1.1. Single Datasource Example

If you specify ***test-postgresql=TEST*** as the value for the ***DB_SERVICE_PREFIX_MAPPING*** environment variable, it creates a datasource with the following name:

java:jboss/datasources/test_postgresql

You must use the ***TEST_*** prefix when specifying other environment variables for the datasource. For example, to set the username and password, use ***TEST_USERNAME*** and ***TEST_PASSWORD*** as the environment variables.

6.1.2. Multiple Datasource Example

If you specify ***cloud-postgresql=CLOUD,test-mysql=TEST_MYSQL*** as the value for the ***DB_SERVICE_PREFIX_MAPPING*** environment variable, it creates two datasources with the following names:

- *java:jboss/datasources/test_mysql*
- *java:jboss/datasources/cloud_postgresql*

When specifying other environment variables for the datasources, you must use the ***TEST_MYSQL*** prefix to configure the MySQL datasource. For example, use ***TEST_MYSQL_USERNAME*** as the environment variable to specify the username.

Similarly, you must use the ***CLOUD_*** prefix to configure the PostgreSQL datasource. For example, use ***CLOUD_USERNAME*** as the environment variable to specify the username.

6.2. CONFIGURING EXTERNAL DATASOURCES

To use an external datasource, you define a custom image template and then use the Source-to-Image (S2I) build tool to create an image. S2I is a framework that takes application source code as an input and produces a new image that runs the assembled application as output.

The following high-level steps provide an overview of the process:

1. Specify the **CUSTOM_INSTALL_DIRECTORIES** environment variable in the image template JSON. This variable defines the location where S2I artifacts reside, as in the following example:

```
{
   "name": "CUSTOM_INSTALL_DIRECTORIES",
   "value": "extensions/*"
}
```

2. Create an **install.sh** script in that directory. This script installs the modules and drivers for the external datasource in the image.
   The following is an example **install.sh** script:

```
#!/bin/bash

# Import the common functions for installing modules and configuring drivers
source /usr/local/s2i/install-common.sh

# Directory where this script is located
injected_dir=$1

# Install the modules for the datasource
install_modules ${injected_dir}/modules

# Configure the drivers for the datasource
configure_drivers ${injected_dir}/drivers.properties
```

3. Include a **modules** subdirectory that contains a **module.xml** file and the driver for the datasource. The resulting image uses the module to load classes and define dependencies.

As an example, you plan to use Derby as an external datasource. You need to obtain a driver such as **derby-10.12.1.1.jar** and place it in the following directory: **modules/org/apache/derby/main/**

In the same directory, you also need to create a **module.xml** file that defines the driver as a resource and declares dependencies.

The following is an example **module.xml** file:

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.3" name="org.apache.derby">
 <resources>
   <resource-root path="derby-10.12.1.1.jar"/>
   <resource-root path="derbyclient-10.12.1.1.jar"/>
 </resources>

 <dependencies>
   <module name="javax.api"/>
   <module name="javax.transaction.api"/>
 </dependencies>
</module>
```

4. Define the driver configuration properties in a **drivers.property** environment variable file. The following is an example **drivers.property** file:

```
#DRIVERS
DRIVERS=DERBY

DERBY_DRIVER_NAME=derby
DERBY_DRIVER_MODULE=org.apache.derby
DERBY_DRIVER_CLASS=org.apache.derby.jdbc.EmbeddedDriver
DERBY_XA_DATASOURCE_CLASS=org.apache.derby.jdbc.EmbeddedXADataSource
```

5. After you build and deploy the image, specify environment variables for the datasource. The following example shows a datasource definition with the ***DATASOURCES*** environment variable:

```
# Set a unique prefix for the datasource
DATASOURCES=ACCOUNTS_DERBY
# Specify other environment variables using the prefix
ACCOUNTS_DERBY_DATABASE=accounts
ACCOUNTS_DERBY_JNDI=java:/accounts-ds
ACCOUNTS_DERBY_DRIVER=derby
ACCOUNTS_DERBY_JTA=true
ACCOUNTS_DERBY_NONXA=false
ACCOUNTS_DERBY_USERNAME=username
ACCOUNTS_DERBY_PASSWORD=password
ACCOUNTS_DERBY_XA_CONNECTION_PROPERTY_DatabaseName=/opt/eap/standalone
/data/databases/derby/accounts
# _HOST and _PORT are required but not used
ACCOUNTS_ORACLE_HOST=dummy
ACCOUNTS_ORACLE_PORT=1527
```

CHAPTER 7. MANAGING RED HAT JBOSS DATA GRID FOR OPENSHIFT

A major difference in managing an JBoss Data Grid 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 JBoss Data Grid for OpenShift image is different from the process you may be used to for a regular release of JBoss Data Grid.

CHAPTER 8. BUILDING RED HAT JBOSS DATA GRID FOR OPENSHIFT IMAGES

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

You can change the JBoss Data Grid configuration in the image using the S2I process or by using a modified JBoss Data Grid for OpenShift image.

8.1. USING THE JBOSS DATA GRID FOR OPENSHIFT IMAGE SOURCE-TO-IMAGE (S2I) PROCESS

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

The S2I process for the JBoss Data Grid 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 **$MAVEN_ARGS** environment variable.
2. By default the **package** goal is used with the **openshift** profile, including the system properties for skipping tests (**-DskipTests**) and enabling the Red Hat GA repository (**-Dcom.redhat.xpaas.repo.redhatga**).
3. The results of a successful Maven build are copied to ***JDG_HOME/standalone/deployments***. This includes all JAR files from the directory within the source repository specified by **$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.

The JBoss Data Grid server supports only JAR deployments, which can include custom filters and converters. The JBoss Data Grid server does not support WAR and EAR deployments.

- 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***.

1. 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.

8.2. USING A MODIFIED JBOSS DATA GRID 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 JBoss Data Grid 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 JBoss Data Grid 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 JBoss Data Grid 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.

8.3. 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 7.1 Infinispan application together with JBoss Data Grid for OpenShift image for an end-to-end example of a binary build.

CHAPTER 9. DEPLOYING JBOSS DATA GRID FOR OPENSHIFT WITH CUSTOM CONFIGURATION FILES

You can use the OpenShift **ConfigMap** API to create a deployment that uses custom configuration instead of using the source-to-image (S2I) build process.

NOTE

- Changes to the configuration via **ConfigMap** do not cause pods to redeploy automatically. You must manually redeploy pods if you update **standalone.xml**.
- JBoss Data Grid for OpenShift deployments that you create with custom configuration files do not support shared persistent volumes that you configure with the **DATAGRID_SPLIT** environment variable.

9.1. SETTING UP THE CONFIGURATION FILES AND CUSTOM TEMPLATE

Create a ConfigMap that contains your configuration files and mount it to a specific directory as follows:

1. Mount your configuration files, the ConfigMap content, in the following directory: **/opt/datagrid/standalone/configuration/user**

   At a minimum, this directory must contain **standalone.xml** to configure JBoss Data Grid. This directory can also contain **logging.properties**, **application-role.properties**, and other properties files that are available with the JBoss Data Grid distribution.

   Note the following requirements for your custom configuration:

   - You must explicitly define all cache and endpoint configuration in **standalone.xml**. You cannot use environment variables to configure caches or endpoints after you create a deployment.
   - Your cache container must be named **clustered** so that the default ReadinessProbe works.

     ```
     <cache-container name="clustered">
       ...
     </cache-container>
     ```

   - To encrypt client to server traffic, you must configure the server identity in **standalone.xml**. You cannot use environment variables to configure HTTPS after you create a deployment.

2. Create a custom template for your JBoss Data Grid for OpenShift deployment.

   a. Ensure that the template exposes the required ports and services.

   b. Set the **USER_CONFIG_MAP** environment variable to a value of **true**.

TIP

Add placeholders to your custom **standalone.xml** if you want to make environment variables available in your deployment.

For example, the following is a placeholder for the **JGROUPS_PING_PROTOCOL**:

```
<!-- ##JGROUPS_PING_PROTOCOL## -->
```

Refer to clustered-openshift.xml to review the default XML file for JBoss Data Grid for OpenShift. This file contains all the available placeholders.

You can find examples for deployments with custom configuration in the following files:

- Example standalone.xml
- Example Configuration Template

9.2. CREATING DEPLOYMENTS WITH CUSTOM CONFIGURATION

To deploy JBoss Data Grid for OpenShift with a custom configuration, do the following:

1. On your master host(s), log in as a cluster administrator or a user with project administrator access to the **openshift** namespace.

   ```
   $ oc login -u system:admin
   ```

2. Import your custom template into the **openshift** namespace.

   ```
   $ oc create -n openshift -f path/to/template.yaml
   ```

3. Create a ConfigMap from the directory where your custom configuration resides.

   - To create a ConfigMap with **standalone.xml** only, do the following:

     ```
     $ oc create configmap datagrid-config --from-file=./standalone.xml
     ```

   - To create a ConfigMap with **standalone.xml** and other configuration files, do the following:

     ```
     $ oc create configmap datagrid-config \
       --from-file=path/to/configuration
     ```

     Where **path/to/configuration** is the local directory that contains the configuration files.

     The ConfigMap name should match the name that you specify in your custom template. The example template uses the name **datagrid-config**.

4. Deploy JBoss Data Grid for OpenShift with your custom configuration.

   ```
   $ oc new-app user-config
   ```

   The application name should match the name that you specify in your custom template. The example template uses the name **user-config**.

When you deploy JBoss Data Grid for OpenShift, the configuration files are copied to the **/opt/datagrid/standalone/configuration** directory for the application.

CHAPTER 10. UPGRADING RED HAT JBOSS DATA GRID FOR OPENSHIFT BETWEEN RELEASES

Rolling upgrades of JBoss Data Grid allow you to upgrade a cluster from one version to a new version without experiencing any downtime.

For complete details on rolling upgrades with JBoss Data Grid, see Rolling Upgrades in the JBoss Data Grid documentation.

IMPORTANT

As of 7.2, JBoss Data Grid supports rolling upgrades using Hot Rod only. In earlier releases, JBoss Data Grid allowed you to perform rolling upgrades using the REST interface.

Additionally, JBoss Data Grid supports rolling upgrades using Hot Rod from version 6.6.2 and later. If you plan to perform a rolling upgrade from a version earlier than 6.6.2, you must first upgrade to JBoss Data Grid 6.6.2.

CHAPTER 11. DEPLOYING AN EAP INFINISPAN APPLICATION WITH THE JBOSS DATA GRID FOR OPENSHIFT IMAGE

Complete the steps in this tutorial to see how you can deploy an EAP Infinispan application with the JBoss Data Grid for OpenShift image.

This tutorial uses CarMart quickstart to deploy EAP 6.4 / EAP 7.1 Infinispan application that accesses a remote JBoss Data Grid server running in the same OpenShift project.

11.1. IMPORTING THE LATEST EAP AND JBOSS DATA GRID FOR OPENSHIFT IMAGE STREAMS AND TEMPLATES

EAP and JBoss Data Grid for OpenShift images are pulled on demand from the Red Hat Registry. As a first step, import the EAP and JBoss Data Grid for OpenShift image streams and templates into the namespace of your OpenShift project.

11.1.1. Log In with Administrator Access

Importing EAP image streams and templates requires administration privileges in the **openshift** namespace (global project). On your master host(s), you must log in as a cluster administrator or a user with project administrator access to the **openshift** namespace.

For example, log in with the default **system:admin** user on the master as follows:

```
$ oc login -u system:admin
```

11.1.2. Importing the EAP Images

To import EAP 6.4, run the following command:

```
$ oc -n openshift import-image jboss-eap64-openshift:1.8
```

To import EAP 7.1, run the following command:

```
$ oc -n openshift import-image jboss-eap71-openshift:1.2
```

11.1.3. Creating the JBoss Data Grid for OpenShift Image Resources

Import the image and templates into Red Hat OpenShift. See Importing Image Templates.

11.2. CREATING A PROJECT

Create a new project as follows:

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

11.3. DEPLOYING THE JBOSS DATA GRID 7.2 SERVER

Deploy the server and specify the following:

- **carcache-hotrod** as the name of application,
- A Hot Rod based connector, and
- **carcache** as the name of the Infinispan cache to configure.

```
$ oc new-app --name=carcache-hotrod \
--image-stream=jboss-datagrid72-openshift:1.3 \
-e INFINISPAN_CONNECTORS=hotrod \
-e CACHE_NAMES=carcache \
-e HOTROD_SERVICE_NAME=carcache-hotrod \
-e HOTROD_AUTHENTICATION=true \
-e USERNAME=jdguser \
-e PASSWORD=P@ssword1
--> Found image d83b4b2 (3 months old) in image stream "openshift/jboss-datagrid72-openshift" under tag "latest" for "jboss-datagrid72-openshift"

    JBoss Data Grid 7.2
    -------------------
    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.
```

11.4. DEPLOYING A BINARY BUILD OF EAP 6.4 / EAP 7.1 CARMART APPLICATION

1. Clone the source code.

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

2. Configure the Red Hat JBoss Middleware Maven repository .

3. Build the **datagrid/carmart** application.

```
$ cd openshift-quickstarts/datagrid71/carmart/
```

```
$ mvn clean package -Premote-jbossas,openshift
[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/ROOT.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] ------------------------------------------------------------------------
```

4. Verify 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 **deployments** folder of the image being built on OpenShift. For the application to deploy, the directory hierarchy that contains the web application data must be correctly structured.

   However, the carmart application already includes the correct directory structure after building:

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

```
$ ls deployments
ROOT.war
```

NOTE

The location of the standard deployments directory depends on the underlying base image that was used to deploy the application.

Table 11.1. 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 7.1	***$JBOSS_HOME/standalone/deployments***
Java S2I for OpenShift	***/deployments***
JWS for OpenShift	***$JWS_HOME/webapps***

5. Identify the image stream for the EAP 6.4 / EAP 7.1 image.

```
$ oc get is -n openshift | grep eap | cut -d ' ' -f 1
jboss-eap64-openshift
jboss-eap71-openshift
```

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

```
$ oc new-build --binary=true \
--image-stream=jboss-eap64-openshift:1.8 \
--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-eap71-openshift** as the image stream name in the preceding command to use EAP 7.1 image for the application.

7. Start the binary build. Instruct the **oc** executable to use the main directory of the binary build from the previous step as the directory that contains binary input for the OpenShift build.

```
$ oc start-build eap-app --from-dir=deployments/ --follow
Uploading directory "deployments" 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
```

8. 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.
```

9. 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
```

10. Access the application.
Access the CarMart application in your browser using the URL **http://eap-app-jdg-bin-demo.openshift.example.com/**. You can view and remove existing cars from the **Home** tab or add new cars from the **New car** tab.

CHAPTER 12. ENVIRONMENT VARIABLES

You configure Red Hat JBoss Data Grid for OpenShift deployments with environment variables.

12.1. IMAGE INFORMATION

The following environment variables provide information about the image. You should not modify these environment variables.

JBOSS_DATAGRID_VERSION

Displays the version of Red Hat JBoss Data Grid on which the container is based.

JBOSS_HOME

Displays the directory that contains the distribution: **/opt/datagrid**.

JBOSS_IMAGE_NAME

Displays the name of the image.

JBOSS_IMAGE_RELEASE

Displays the image release label.

JBOSS_IMAGE_VERSION

Displays the image version.

JBOSS_MODULES_SYSTEM_PKGS

Lists JBoss system modules.

JBOSS_PRODUCT

Displays the product label: **datagrid**.

LAUNCH_JBOSS_IN_BACKGROUND

Allows graceful shutdowns.

12.2. CONTAINER CONFIGURATION

Configure containers with the following environment variables:

USERNAME

Sets the name for the JBoss Data Grid user.

PASSWORD

Sets the password for the JBoss Data Grid user.

DATAGRID_SPLIT

Determines if the data directory for each node should be split in a mesh. The value is **true** or **false** (default).
If you set the value to **true**, you must also configure a persistent volume mounted on **/opt/datagrid/standalone/partitioned_data**.

NOTE

Use the **datagrid72-partition** template to deploy an example application that preserves cache metadata between restarts. Ensure that the ***${APPLICATION_NAME}-datagrid-claim*** persistent volume claim is available and that the **${APPLICATION_NAME}-datagrid-pvol** persistent volume is mounted on **/opt/datagrid/standalone/partitioned_data**.

JAVA_OPTS_APPEND

Appends options to the **JAVA_OPTS** environment variable on startup.
For example, **JAVA_OPTS_APPEND**=**-Dfoo=bar**

JGROUPS_CLUSTER_PASSWORD

Matches the password for accessing JGroups configuration. It must be the same across the cluster. By default, the image uses the value for the **OPENSHIFT_KUBE_PING_LABELS** variable; however, JBoss application templates generate random values.

See Securing Network Traffic for information about using JGroups keystores to encrypt cluster communication.

OPENSHIFT_KUBE_PING_LABELS

Specifies the clustering labels selector.
For example, **OPENSHIFT_KUBE_PING_LABELS**=**application=eap-app**

OPENSHIFT_KUBE_PING_NAMESPACE

Specifies the clustering project namespace.

TRANSPORT_LOCK_TIMEOUT

Sets the time to wait to acquire a distributed lock. The default value is **240000**.
JBoss Data Grid uses a distributed lock to maintain a coherent transaction log during state transfer or rehashing, which means that only one cache can perform state transfer or rehashing at a time. This constraint is in place because more than one cache could be involved in a transaction.

12.3. CACHE CONFIGURATION

Configure caches with the following environemnt variables:

CACHE_NAMES

Defines cache instances in your configuration.
If you do not specify cache names, the launch script adds configuration for caches named **default** and **memcached**. The default cache configuration is a distributed-cache in **SYNC** mode.

TIP

Give each cache instance in your configuration a unique name. Use underscore characters (_) and descriptive labels to help you distinguish between cache instances. This ensures that you do not have conflicts when applying cache-specific configuration.

For example, **CACHE_NAMES**=**addressbook, addressbook_indexed**

CACHE_CONTAINER_START

Configures how the cache container starts. Specify one of the following:

- **LAZY** Starts the cache container when requested by a service or deployment. This is the default.
- **EAGER** Starts the cache container when the server starts.

CACHE_CONTAINER_STATISTICS

Configures the cache container to collect statistics. The value is **true** (default) or **false**. You can set the value to **false** to improve performance.

DEFAULT_CACHE

Sets the default cache for the cache container.

12.3.1. Cache Container Security Configuration

Configure security for the cache container with the following environment variables:

CONTAINER_SECURITY_CUSTOM_ROLE_MAPPER_CLASS

Specifies the class of the custom principal to role mapper.
For example, **CONTAINER_SECURITY_CUSTOM_ROLE_MAPPER_CLASS=com.acme.CustomRoleMapper**

CONTAINER_SECURITY_ROLE_MAPPER

Sets a role mapper for this cache container with the following values:

- **identity-role-mapper** Uses the Principal name as the role name. This is the default role mapper if you do not specify one and use the **CONTAINER_SECURITY_ROLES** environment variable to define role names.
- **common-name-role-mapper** Uses the Common Name (CN) as the role name if the Principal name is a Distinguished Name (DN). For example, the DN **cn=managers,ou=people,dc=example,dc=com** is mapped to the **manager** role name.
- **cluster-role-mapper** Uses the **ClusterRegistry** to store Principal name to role mappings.
- **custom-role-mapper** Takes the fully-qualified class name of an implementation of the **org.infinispan.security.impl.PrincipalRoleMapper** interface.
  For more information see Role Mapping in the Developer Guide.

CONTAINER_SECURITY_ROLES

Defines role names and assigns permissions to them.
For example, **CONTAINER_SECURITY_ROLES=admin=ALL, reader=READ, writer=WRITE**

12.3.2. Cache Specific Configuration

You can control behavior for each cache in your configuration with these environment variables.

To set an environment variable, you specify the cache name as a prefix for the variable.

IMPORTANT

You must specify the cache name as a prefix in capital letters (all caps) otherwise the configuration does not take effect.

For example, you create two separate cache instances: **MyCache** and **MYCACHE**. You then set **MyCache_CACHE_TYPE=replicated** to configure the **MyCache** instance. This configuration does not take effect. However, if you set **MYCACHE_CACHE_TYPE=replicated** the configuration takes effect for both the **MyCache** and **MYCACHE** instances.

<CACHE_NAME>_CACHE_TYPE

Determines whether this cache should be distributed or replicated. You can specify either **distributed** (default) or **replicated**.

<CACHE_NAME>_CACHE_START

Configures how the cache starts. Specify one of the following:

- **LAZY** Starts the cache when requested by a service or deployment. This is the default.
- **EAGER** Starts the cache when the server starts.

<CACHE_NAME>_CACHE_BATCHING

Enables invocation batching for this cache. The value is **true** or **false** (default).

<CACHE_NAME>_CACHE_STATISTICS

Configures the cache to collect statistics. The value is **true** (default) or **false**. You can set the value to **false** to improve performance.

<CACHE_NAME>_CACHE_MODE

Sets the clustered cache mode. Specify one of the following:

- **ASYNC** for asynchronous operations.
- **SYNC** for synchronous operations.

<CACHE_NAME>_CACHE_QUEUE_SIZE

Sets the threshold at which the replication queue is flushed when the cache is in **ASYNC** mode. The default value is **0** (flushing is disabled).

<CACHE_NAME>_CACHE_QUEUE_FLUSH_INTERVAL

Specifies the wakeup time, in milliseconds, for the thread that flushes the replication queue in **ASYNC** mode. The default value is **10**.

<CACHE_NAME>_CACHE_REMOTE_TIMEOUT

Specifies the timeout, in milliseconds, to wait for acknowledgement when making remote calls in **SYNC** mode. If the timeout is reached, the remote call is aborted and an exception is thrown. The default value is **17500**.

<CACHE_NAME>_CACHE_OWNERS

Specifies the number of cluster-wide replicas for each cache entry. The default value is **2**.

<CACHE_NAME>_CACHE_SEGMENTS

Specifies the number of hash space segments per cluster. The recommended value is **10 * cluster size**. The default value is **80**.

<CACHE_NAME>_CACHE_L1_LIFESPAN

Specifies the maximum lifespan, in milliseconds, of an entry placed in the L1 cache. The default value is **0** (L1 is disabled).

<CACHE_NAME>_CACHE_MEMORY_EVICTION_TYPE

Defines the maximum limit for entries in the cache. You can set the following values:

- **COUNT** Measures the number of entries in the cache. When the count exceeds the maximum, JBoss Data Grid evicts unused entries.
- **MEMORY** Measures the amount of memory that all entries in the cache take up. When the total amount of memory exceeds the maximum, JBoss Data Grid evicts unused entries.

<CACHE_NAME>_CACHE_MEMORY_STORAGE_TYPE

Defines how JBoss Data Grid stores entries in the cache. You can set the following values:

Storage Type	Description	Eviction Type	Policy
object	Stores entries as objects in the Java heap. This is the default storage type.	COUNT	TinyLFU
binary	Stores entries as **bytes[]** in the Java heap.	COUNT or MEMORY	TinyLFU
off-heap	Stores entries as **bytes[]** in native memory outside the Java.	COUNT or MEMORY	LRU

<CACHE_NAME>_CACHE_MEMORY_EVICTION_SIZE

Configures the size of the cache before eviction starts. Set the value to a number greater than zero.

- For **COUNT**, the size is the maximum number of entries the cache can hold before eviction starts.
- For **MEMORY**, the size is the maximum number of bytes the cache can take from memory before eviction starts. For example, a value of **10000000000** is 10 GB.
  Try different cache sizes to determine the optimal setting. A cache size that is too large can cause JBoss Data Grid to run out of memory. At the same time, a cache size that is too small wastes available memory.

NOTE

If you configure a JDBC store, passivation is automatically enabled when you set the eviction size to a value that is greater than zero.

<CACHE_NAME>_CACHE_MEMORY_EVICTION_STRATEGY

Controls how JBoss Data Grid performs eviction. You can set the following values:

Strategy	Description
NONE	JBoss Data Grid does not evict entries. This is the default setting unless you configure eviction.
REMOVE	JBoss Data Grid removes entries from memory so that the cache does not exceed the configured size. This is the default setting when you configure eviction.

Strategy	Description
MANUAL	JBoss Data Grid does not perform eviction. Eviction takes place manually by invoking the **evict()** method from the **Cache** API.
EXCEPTION	JBoss Data Grid does not write new entries to the cache if doing so would exceed the configured size. Instead of writing new entries to the cache, JBoss Data Grid throws a **ContainerFullException**.

<CACHE_NAME>_CACHE_MEMORY_OFF_HEAP_ADDRESS_COUNT

Specifies the number of pointers that are available in the hash map to prevent collisions when using **OFFHEAP** storage. Preventing collisions in the hash map improves performance.
Set the value to a number that is greater than the number of cache entries. By default **address-count** is 2^20, or 1048576. The parameter is always rounded up to a power of 2.

<CACHE_NAME>_CACHE_EXPIRATION_LIFESPAN

Specifies the maximum lifespan, in milliseconds, of a cache entry, after which the entry is expired cluster-wide. The default value is **-1** (entries never expire).

<CACHE_NAME>_CACHE_EXPIRATION_MAX_IDLE

Specifies the maximum idle time, in milliseconds, that cache entries are maintained in the cache. If the idle time is exceeded, then the entry is expired cluster-wide. The default value is **-1** (expiration is disabled).

<CACHE_NAME>_CACHE_EXPIRATION_INTERVAL

Specifies the interval, in milliseconds, between runs to purge expired entries from memory and any cache stores. The default value is **5000**. Set **-1** to disable expiration.

<CACHE_NAME>_JDBC_STORE_TYPE

Sets the type of JDBC store to configure. You can set the following values:

- **string**
- **binary**

<CACHE_NAME>_JDBC_STORE_DATASOURCE

Defines the jndiname of the datasource.
For example,
<CACHE_NAME>_JDBC_STORE_DATASOURCE=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. The defaule value is **ispn_entry**.

<CACHE_NAME>_CACHE_INDEX

Sets the indexing mode of the cache. You can set the following values:

- **NONE** This is the default.

- LOCAL
- ALL

<CACHE_NAME>_INDEXING_PROPERTIES

Specifies a comma-separated list of properties to pass to the indexing system.
For example, **<CACHE_NAME>_INDEXING_PROPERTIES**=**default.directory_provider=ram**

<CACHE_NAME>_CACHE_SECURITY_AUTHORIZATION_ENABLED

Enables authorization checks for this cache. The value is **true** or **false** (default).

<CACHE_NAME>_CACHE_SECURITY_AUTHORIZATION_ROLES

Sets the roles required to access this cache.
For example, **<CACHE_NAME>_CACHE_SECURITY_AUTHORIZATION_ROLES**=**admin, reader, writer**

<CACHE_NAME>_CACHE_PARTITION_HANDLING_ENABLED

Configures the cache to enter degraded mode if it loses too many nodes. The value is **true** (default) or **false**.
Deprecated: The **CACHE_PARTITION_HANDLING_ENABLED** environment variable is deprecated. Use **CACHE_PARTITION_HANDLING_WHEN_SPLIT** and **CACHE_PARTITION_MERGE_POLICY** instead.

To achieve the same configuration as

- **CACHE_PARTITION_HANDLING_ENABLED=false**, do not set environment variables so that default values take effect as follows:

```
<CACHE_NAME>_CACHE_PARTITION_HANDLING_WHEN_SPLIT=ALLOW_READ_W
RITES
<CACHE_NAME>_CACHE_PARTITION_MERGE_POLICY=NONE
```

- **CACHE_PARTITION_HANDLING_ENABLED=true**, set environment variables as follows:

```
<CACHE_NAME>_CACHE_PARTITION_HANDLING_WHEN_SPLIT=DENY_READ_WRI
TES
<CACHE_NAME>_CACHE_PARTITION_MERGE_POLICY=NONE
```

<CACHE_NAME>_CACHE_PARTITION_HANDLING_WHEN_SPLIT

Configures the strategy for handling partitions between nodes in a cluster when network events isolate nodes from each other. Partitions function as independent clusters until JBoss Data Grid merges cache entries to re-form a single cluster. You can set the following values:

Partition Handling Strategy	Description
ALLOW_READ_WRITES	Nodes from any partition can read or write cache entries. This is the default value.

Partition Handling Strategy	Description
DENY_READ_WRITES	Nodes enter degraded mode if: * One or more hash space segments in the partition have no owners. The **owners** are the number of cluster-wide replicas for cache entries. * The partition has less than half the nodes from the most recent stable cluster topology. In degraded mode, only nodes in the same partition can read or write cache entries. All owners, or copies, for a cache entry must exist on the same partition, otherwise the read or write operation fails with an **AvailabilityException**.
ALLOW_READS	Nodes enter degraded mode similarly to the **DENY_READ_WRITES** strategy. Nodes from any partition can read cache entries. In degraded mode, only nodes in the same partition can write cache entries. All owners, or copies, for a cache entry must exist on the same partition, otherwise the write operation fails with an **AvailabilityException**.

For more information, see Handling Network Partitions in the Administration and Configuration Guide.

<CACHE_NAME>_CACHE_PARTITION_MERGE_POLICY

Configures how JBoss Data Grid resolves conflicts between cache entries when merging partitions. You can set the following values:

Merge Policy	Description
NONE	Do not resolve conflicts when merging partitions. This is the default value.
PREFERRED_ALWAYS	Always use the **preferredEntry**. The **preferredEntry** is the primary replica of a cache entry that resides in the partition that contains the most nodes. If the number of nodes is equal between partitions, the **preferredEntry** is the cache entry that resides in the partition with the highest topology ID, which means that topology is more recent.

Merge Policy	Description
PREFERRED_NON_NULL	Use the **preferredEntry** if it has a value (non-null). If the **preferredEntry** does not have a value, use the first entry defined in **otherEntries**.
REMOVE_ALL	Remove entries (key and value) from the cache if conflicts exist.

<CACHE_NAME>_STATE_TRANSFER_TIMEOUT

Sets the amount of time, in milliseconds, to wait for other cache instances in the cluster to transfer state to the cache. If other cache instances do not transfer state before the timeout occurs, the application throws an exception and aborts startup. The default value is **240000** (4 minutes).
You must use a custom template to set this environment variable. It does not take effect if you set the state transfer timeout in the default JBoss Data Grid for OpenShift templates.

12.4. ENDPOINT CONFIGURATION

Clients can access JBoss Data Grid via REST, Hot Rod, and Memcached endpoints that you define in the cache configuration.

Clients that run in the same project as JBoss Data Grid for OpenShift can access the cache via Hot Rod and receive a full cluster view. These clients can also use consistent hashing capabilities.

However, when clients run in a different project to JBoss Data Grid for OpenShift, they need to access the JBoss Data Grid cluster using an OpenShift service that exposes the HotRod endpoint externally. Depending on your network configuration, clients might not have access to some pods and must use **BASIC** client intelligence. In these cases, clients might require extra network hops to access data, which can increase network latency.

External access to clients running in OpenShift requires routes with passthrough encryption termination. Clients must also use **BASIC** client intelligence and the fully qualified domain name as a TLS/SNI host name. Alternatively, you can expose the JBoss Data Grid cluster behind a Load Balancer service that is externally available.

Configure endpoints with the following environment variables:

INFINISPAN_CONNECTORS

Defines a comma-separated list of connectors to configure. Defaults to **hotrod**, **memcached**, **rest**. If authorization or authentication is enabled on the cache then you should remove **memcached** because this protocol is inherently insecure.

MEMCACHED_CACHE

Sets the cache name for the Memcached connector. Defaults to **memcached** if you do not specify a cache name with the **CACHE_NAMES** environment variable.

HOTROD_SERVICE_NAME

Defines the name of the {openshiftshort} service for the external Hot Rod connector.
The external hotrod connector is available only if you define this environment variable.

For example, if you set **HOTROD_SERVICE_NAME=DATAGRID_APP_HOTROD** the Hot Rod external connector returns **DATAGRID_APP_HOTROD:11333**.

HOTROD_AUTHENTICATION

Configures the **hotrod-connectors** with authentication in the **ApplicationRealm**. The value is **true** or **false** (default).

HOTROD_ENCRYPTION

Configures the **hotrod-connectors** with encryption in the **ApplicationRealm**. The value is **true** or **false** (default).
If you enable this environment variable, you must also set environment variables to encrypt client to server communication. See Securing Network Traffic.

ENCRYPTION_REQUIRE_SSL_CLIENT_AUTH

Specifies if client certificate authentication is required. The value is **true** or **false** (default).

REST_SECURITY_DOMAIN

Specifies the security domain to use for authentication and authorization purposes. The default value is **none** (no authentication).

REST_STORE_AS_STRING

Specifies if JBoss Data Grid saves entries as Java strings when written to the cache via the REST API. The value is **true** or **false** (default).
Set the value to **true** if you are upgrading the image from a previous version and plan to read persisted cache entries.

NOTE

JBoss Data Grid version 7.1 and earlier: When you write entries to the cache through the REST endpoint, JBoss Data Grid stores them as Java strings.

JBoss Data Grid version 7.2 and later: JBoss Data Grid stores cache entries as **bytes[]** to enable data interoperability between clients and protocols.

If you upgrade JBoss Data Grid for OpenShift images from an previous version to version 7.2, JBoss Data Grid returns null values when you attempt to read cache entries that are persisted to a data store. To resolve the null values, set **REST_STORE_AS_STRING=true**.

12.4.1. Exposed Ports

JBoss Data Grid for OpenShift exposes endpoints on the following ports by default:

Port Number	Protocol	Use
8080	TCP	HTTP Access
8443	TCP	HTTPS Access
8778	TCP	Remote JMX Access

Port Number	Protocol	Use
11211	TCP	Memcached Access
11222	TCP	Internal Hotrod Access
11333	TCP	External Hotrod Access

NOTE

From the same OpenShift namespace, the Hot Rod endpoint is accessible at **${pod_IP_address}:11222**.

If you set the **HOTROD_SERVICE_NAME** environment variable, the Hot Rod external connector returns **${service_name}:11333** for the endpoint.

12.5. DATASOURCE CONFIGURATION

You can configure datasources with the following environment variables:

DB_SERVICE_PREFIX_MAPPING

Defines a comma-separated list of datasources to configure.
For example, **DB_SERVICE_PREFIX_MAPPING=test-mysql=TEST_MYSQL**. See Configuring Persistent Datasources for more information.

<NAME>_<DATABASE_TYPE>_SERVICE_HOST

Defines the database server hostname or IP for the datasource **connection_url** property.
For example, **<NAME>_<DATABASE_TYPE>_SERVICE_HOST=192.0.2.0**

<NAME>_<DATABASE_TYPE>_SERVICE_PORT

Defines the database server port.

<PREFIX>_USERNAME

Defines the user for the datasource.

<PREFIX>_PASSWORD

Defines the password for the datasource.

<PREFIX>_DATABASE

Defines the database name for the datasource.
For example, **<PREFIX>_DATABASE=myDatabase**.

<PREFIX>_DRIVER

Defines Java database driver for the datasource.
For example, **<PREFIX>_DRIVER=postgresql**

<PREFIX>_BACKGROUND_VALIDATION

Specifies if a background thread validates database connections before they are used. The value is **true** or **false** (default). By default, the **<validate-on-match>** method is enabled.

<PREFIX>_BACKGROUND_VALIDATION_MILLIS

Specifies how often validation occurs, in milliseconds, if you set the **<PREFIX>_BACKGROUND_VALIDATION** environment variable to **true**. The default value is **10000**.

<PREFIX>_CONNECTION_CHECKER

Specifies a connection checker class that validates connections to the database.
For example,
<PREFIX>_CONNECTION_CHECKER=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQL

<PREFIX>_EXCEPTION_SORTER

Specifies the exception sorter class that detects and cleans up after fatal database connection exceptions.
For example,
<PREFIX>_EXCEPTION_SORTER=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionS

<PREFIX>_JNDI

Defines the JNDI name for the datasource.
Defaults to **java:jboss/datasources/<name>_<database_type>**. The launch script automatically generates the value from the **DB_SERVICE_PREFIX_MAPPING** environment variable.

For example, **<PREFIX>_JNDI=java:jboss/datasources/test-postgresql**

<PREFIX>_JTA

Defines the Java Transaction API (JTA) option for non-XA datasources. The value is **true** (default) or **false**.

<PREFIX>_MAX_POOL_SIZE

Defines the maximum pool size for the datasource.

<PREFIX>_MIN_POOL_SIZE

Defines the minimum pool size for the datasource.

<PREFIX>_NONXA

Defines the datasource as a non-XA datasource. The value is **true** or **false** (default).

<PREFIX>_TX_ISOLATION

Defines the java.sql.Connection transaction isolation level for the database.
For example, **<PREFIX>_TX_ISOLATION=TRANSACTION_READ_UNCOMMITTED**

<PREFIX>_URL

Defines the connection URL for a non-XA datasource.
If you do not specify a connection URL, the launch script automatically generates it from other environment variables as follows: **url="jdbc:${DRIVER}://${HOST}:${PORT}/${DATABASE}"**.

However, the launch script constructs the correct connection URLs only for internal datasources such as PostgreSQL and MySQL. If you use any other non-XA datasource you must specify the connection URL.

For example, **<PREFIX>_URL=jdbc:postgresql://localhost:5432/postgresdb**

<PREFIX>_XA_CONNECTION_PROPERTY_<PROPERTY_NAME>

Defines connection properties for an XA datasource.
Consult the appropriate driver documentation for your datasource to find which XA properties you can set on the connection.

For example,
<PREFIX>_XA_CONNECTION_PROPERTY_DatabaseName=/opt/eap/standalone/data/databases

This example adds the following to the configuration:

```
<xa-datasource-property
name="DatabaseName">/opt/eap/standalone/data/databases/db/accounts</xa-datasource-
property>
```

12.6. SECURITY DOMAIN CONFIGURATION

Use the following environment variables to customize the security domain for the container:

SECDOMAIN_NAME

Defines additional security domains.
For example: **SECDOMAIN_NAME=myDomain**

SECDOMAIN_PASSWORD_STACKING

Enables the password staking module and sets the **useFirstPass** option. The value is **true** or **false** (default).

SECDOMAIN_LOGIN_MODULE

Specifies a login module to use. The default value is **UsersRoles**

SECDOMAIN_USERS_PROPERTIES

Specifies the properties file that contains user definitions. The default value is **users.properties**.

SECDOMAIN_ROLES_PROPERTIES

Specifies the properties file that contains role definitions. The default value is **roles.properties**.

CHAPTER 13. REFERENCE

13.1. 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.

13.2. JBOSS DATA GRID FOR OPENSHIFT LOGS

In addition to viewing the OpenShift logs, you can troubleshoot a running JBoss Data Grid 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 JBoss Data Grid for OpenShift Image does not have a file log handler configured. Logs are only sent to the container's standard out.