Red Hat Data Grid 7.3

Red Hat Data Grid for OpenShift

Run Data Grid on OpenShift

Last Updated: 2022-06-23

Red Hat Data Grid 7.3 Red Hat Data Grid for OpenShift

Run Data Grid on OpenShift

Abstract

Learn how to configure, customize, and run Red Hat Data Grid containers on Red Hat OpenShift.

Table of Contents

CHAPTER 1. RED HAT DATA GRID

Data Grid provides an elastically scalable in-memory data store for Red Hat OpenShift.

Schemaless data structure

Flexibility to store different objects as key-value pairs.

Grid-based data storage

Designed to distribute and replicate data across clusters.

Elastic scaling

Dynamically adjust the number of nodes to meet demand without service disruption.

Data interoperability

Store, retrieve, and query data in the grid from different endpoints.

1.1. DATA GRID DOCUMENTATION

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

1.2. DATA GRID REPOSITORIES

- Data Grid 7 OpenShift Image holds the container image and resources for Data Grid for OpenShift.
- OpenShift Quickstart Tutorials provide code examples with how-to and best practice demonstrations for running Data Grid on OpenShift.

1.3. DATA GRID IMAGE DETAILS

Red Hat Data Grid for OpenShift images are hosted on the Red Hat Container Registry, where you can find health indexes for the images along with information about each tagged version.

- Red Hat Data Grid for OpenShift Image Tags and Red Hat Advisories

CHAPTER 2. GETTING STARTED

2.1. SYSTEM REQUIREMENTS

To use Red Hat Data Grid for OpenShift, you need:

- A running Red Hat OpenShift cluster.
  For supported Red Hat OpenShift Container Platform versions, refer to Red Hat Data Grid Supported Configurations.

  TIP

  Use the Red Hat Container Development Kit to create a local OpenShift cluster with minishift.

- An **oc** client in your **$PATH**.

2.2. CREATING A DATA GRID FOR OPENSHIFT PROJECT

Set up an OpenShift project where you can run Data Grid for OpenShift pods.

1. Log in to your OpenShift cluster.
   If you are new to OpenShift, try the following tutorial: Logging in to an OpenShift Cluster .
2. Create an OpenShift project with the **oc new-project** command, for example:

```
$ oc new-project rhdg-helloworld --display-name="Red Hat Data Grid"
```

2.3. SETTING UP REGISTRY AUTHENTICATION

You must authenticate with the Red Hat Container Catalog, *registry.redhat.io*, to pull Data Grid images.

Use one of the following:

- Red Hat customer account username and password. Pull resources from *registry.redhat.io* with the **docker login** command.
- Registry service account tokens. Use authentication tokens to configure multiple hosts. Do the following:
  1. Log in to *registry.redhat.io*.
  2. Create or select a **Registry Service Account**
  3. Generate authentication tokens.

2.3.1. Configuring Hosts with Authentication Tokens

Add authentication tokens from your **Registry Service Account** to hosts as follows:

1. Select the **Docker Login** tab and copy the command.
2. Run the **docker login** command on each host that pulls from *registry.redhat.io*.

3. Verify your Docker configuration.

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

2.3.2. Creating Pull Secrets

To pull secured container images that are not available on the internal registry for OpenShift, do the following:

1. Log in to your OpenShift cluster.

2. Select your working project, for example:

```
$ oc project rhdg-helloworld
```

3. Create a generic pull secret with your Docker configuration.

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

4. Link the pull secret to your service account.

```
$ oc secrets link default ${SECRET_NAME} --for=pull
```

5. Mount the secret.

```
$ oc secrets link builder ${SECRET_NAME}
```

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

CHAPTER 3. CONFIGURING AUTHENTICATION AND ENCRYPTION

You need to configure authentication and encryption only if you are using a custom template or want to use your own keystores with the Data Grid deployment configuration templates.

3.1. ADDING KEYSTORES TO SECRETS

To configure authentication and encryption:

1. Create a keystore (**.jks**) with a trusted certificate.
   Both HTTPS and Hot Rod services can use the same keystore or you can create separate keystores.

2. Add the keystore as an OpenShift secret.

   a. Create a secret. For example, to create a secret named **rhdg-https-secret** from a keystore named **rhdg-https.jks**:

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

   b. Link the secret to the default service account.

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

3.2. CONFIGURING DEPLOYMENTS

Instantiate one of the secure templates with following parameters:

1. Set up HTTP and HTTPS hostnames:
   HOSTNAME_HTTP=my.example.hostname

   HOSTNAME_HTTPS=secure-my.example.hostname

2. Specify the name of the keystore: **HTTPS_KEYSTORE=keystore.jks**

3. Specify the path to the keystore: **HTTPS_KEYSTORE_DIR=/etc/datagrid-secret-volume**

4. Specify the name of the secret: **HTTPS_SECRET=rhdg-https-secret**

5. Specify credentials for the keystore:
   HTTPS_NAME=${USERNAME}

   HTTPS_PASSWORD=${PASSWORD}

6. Set the HTTP security domain for the user: **REST_SECURITY_DOMAIN=SecurityRealm**

7. Enforce client certificate authentication: **ENCRYPTION_REQUIRE_SSL_CLIENT_AUTH=true**

8. Enable authentication and encryption for the Hot Rod protocol:
   HOTROD_AUTHENTICATION=true

NOTE

The template automatically sets **HOTROD_ENCRYPTION=true** if you set a value for **HOSTNAME_HTTPS**.

3.3. SETTING UNIQUE KEYSTORES FOR THE HOT ROD PROTOCOL

To use a unique keystore for the Hot Rod protocol:

1. Specify the path to the keystore: **SSL_KEYSTORE_PATH=hr_keystore.jks**
2. Specify the keystore password: **SSL_KEYSTORE_PASSWORD=${PASSWORD}**
3. If necessary, do the following:
   a. Set a relative path to the keystore: **SSL_KEYSTORE_RELATIVE_TO=path/to/keystore/**
   b. Specify the private key password, if different to the keystore password: **SSL_KEY_PASSWORD=${PASSWORD}**
   c. Set the correct alias in the keystore if it contains multiple entries: **SSL_KEYSTORE_ALIAS=cert_alias**
4. Specify authorization credentials if you have not already: **USERNAME=${USERNAME}**

   PASSWORD=${PASSWORD}

NOTE

The Hot Rod endpoint always uses the **ApplicationRealm** to authorize users. If you want to use separate keystores for the Hot Rod and REST endpoints, you must set credentials with the **USERNAME** and **PASSWORD** parameters. Templates then configure the REST endpoint to use the **jdg-openshift** security realm. In this case the **REST_SECURITY_DOMAIN** environment variable does not take effect.

CHAPTER 4. SETTING UP DATA GRID FOR OPENSHIFT SERVICES

4.1. DATA GRID FOR OPENSHIFT SERVICES

Data Grid services are stateful applications that you can easily scale up or down without losing data.

cache-service

An easy-to-use Data Grid for OpenShift cluster designed to accelerate application response time with high-performance caching.

- Data in memory is distributed evenly across nodes. You define the initial size of the Data Grid cluster when you create the service. Distribution is also synchronous. When propagating data to another node, the sending node waits for the operation to complete before the thread continues.
- Single copies of cache entries by default. If a pod restarts, data in that pod is lost. For more resiliency with your data, you can easily enable replication when you create the service.
- Cache entries are stored off-heap for JVM efficiency. When the cache size reaches the amount of memory available to the pod, entries are evicted. You can optionally change the eviction policy to throw a **ContainerFullException**.

datagrid-service

A full distribution of Data Grid for OpenShift that lets you create multiple different cache configurations. Gives you advanced capabilities such as indexing and querying as well as Prometheus monitoring.

4.1.1. Container Storage

cache-service and **datagrid-service** containers have storage volumes mounted at **/opt/datagrid/standalone/data**.

The volume size is 1GB by default. You can adjust the size with **datagrid-service** but not **cache-service**.

Ephemeral or Permanent

When you remotely create caches, you control whether they are ephemeral or permanent. Permanent caches survive container restarts because the cache definitions are saved in the storage volume. Default caches are always permanent.

Persistent

datagrid-service lets you persist cache entries and indexes to the storage volume. If you require more guarantees for your data, you can optionally persist to external file-based storage or a database via a cache store.

4.1.2. Partition Handling

By default, Data Grid for OpenShift services use a partition handling configuration to ensure data consistency.

- **DENY_READ_WRITES** conflict resolution strategy that denies read and write operations for cache entries unless all the owners of a segment are in the same partition.

- **REMOVE_ALL** merge policy that removes entries from the cache when conflicts are detected.

NOTE

Network partitioning applies only when data is replicated across a cluster.

4.1.3. Confirming Service Availability

The templates for **cache-service** and **datagrid-service** are available on Red Hat OpenShift Online and Red Hat OpenShift Container Platform in the **openshift** namespace.

Run the following command to verify that the service templates are available:

```
$ oc get templates -n openshift | grep 'cache-service\|datagrid-service'
```

4.1.3.1. Importing Templates

If necessary, import **cache-service** and **datagrid-service** as follows:

1. Log in to your OpenShift cluster.
2. Import the service templates:

```
$ for resource in cache-service-template.yaml \
  datagrid-service-template.yaml
do
  oc create -n openshift -f \
  https://raw.githubusercontent.com/jboss-container-images/jboss-datagrid-7-openshift-
image/7.3-v1.9/services/${resource}
done
```

TIP

Overwrite existing templates with **oc replace --force**.

4.2. CREATING CACHE SERVICES

Use **cache-service** to quickly set up clusters that give you optimal performance and ease of use with minimal configuration.

1. Create the service with the **new-app** command.
2. Set template parameters and environment variables as appropriate.

For Example:

- Create **cache-service** with minimal configuration:

```
$ oc new-app cache-service \
  -p APPLICATION_USER=${USERNAME} \
  -p APPLICATION_PASSWORD=${PASSWORD}
```

- Create a **cache-service** cluster with three nodes and data replication:

```
$ oc new-app cache-service \
  -p APPLICATION_USER=${USERNAME} \
  -p APPLICATION_PASSWORD=${PASSWORD} \
  -p NUMBER_OF_INSTANCES=3 \
  -p REPLICATION_FACTOR=2
```

Template Parameters

- **APPLICATION_NAME** specifies a name for the application. The default is **cache-service**.
- **NUMBER_OF_INSTANCES** sets the number of nodes in the Data Grid for OpenShift cluster. The default is **1**.
- **TOTAL_CONTAINER_MEM** configures the total amount of memory, in MiB, available to the container. The default is **512**.
- **APPLICATION_USER** creates a user to securely access the cache. There is no default value. You must always create a user.
- **APPLICATION_PASSWORD** specifies a password for the user. If you do not set a password, the service template randomly generates one and stores it as a secret.
- **REPLICATION_FACTOR** specifies the number of copies for each cache entry. The default is **1**.
- **EVICTION_POLICY** defines how **cache-service** behaves when the size of the cache reaches the amount of memory available to the pod. There are two values:
  - **evict** removes entries from the cache. This is the default.
  - **reject** throws **ContainerFullException** instead of adding new entries.

Environment Variables

- **AB_PROMETHEUS_ENABLE** allows you to collect JMX metrics to monitor and analyze Data Grid and has the following values:

  false

  Disables monitoring with the default Prometheus agent.

  true

  Enables monitoring with the default Prometheus agent. The Prometheus Operator must be installed and running. You must also Set Up Monitoring after you create the service.
- **AB_PROMETHEUS_JMX_EXPORTER_PORT** defines the port on which Data Grid publishes JMX metrics. The default is **9779**.

Verifying the Application

Command output displays parameter values and resources when you create **cache-service**, as in the following example:

```
--> Deploying template "rhdg-helloworld/cache-service" to project rhdg-helloworld

     Red Hat Cache Service
     ---------
     Red Hat Data Grid is an in-memory, distributed key/value store.
```

```
      * With parameters:
         ...

--> Creating resources ...
    secret "cache-service" created
    service "cache-service-ping" created
    service "cache-service" created
    statefulset "cache-service" created
--> Success
    ...
```

TIP

Try the Hello World Quickstart Tutorial.

4.3. CREATING DATA GRID SERVICES

Use **datagrid-service** to set up a cluster that you can use with different cache configurations and more complex Data Grid capabilities.

1. Create the service with the **new-app** command.
2. Set template parameters and environment variables as appropriate.

For Example:

- Create **datagrid-service** with minimal configuration:

```
$ oc new-app datagrid-service \
  -p APPLICATION_USER=${USERNAME} \
  -p APPLICATION_PASSWORD=${PASSWORD}
```

- Create a **datagrid-service** cluster with three nodes and monitoring enabled:

```
$ oc new-app datagrid-service \
  -p APPLICATION_USER=${USERNAME} \
  -p APPLICATION_PASSWORD=${PASSWORD} \
  -p NUMBER_OF_INSTANCES=3
  -e AB_PROMETHEUS_ENABLE=true
```

Template Parameters

- **APPLICATION_NAME** specifies a name for the application. The default is **datagrid-service**.
- **NUMBER_OF_INSTANCES** sets the number of nodes in the Data Grid for OpenShift cluster. The default is 1.
- **TOTAL_CONTAINER_MEM** configures the total amount of memory, in MiB, available to the container. The default is 512.
- **APPLICATION_USER** creates a user to securely access the cache. There is no default value. You must always create a user.

- **APPLICATION_PASSWORD** specifies a password for the user. If you do not set a password, the service template randomly generates one and stores it as a secret.
- **REPLICATION_FACTOR** specifies the number of copies for each cache entry. The default is 2.
- **TOTAL_CONTAINER_STORAGE** configures the size, in GiB, of the file-based storage volume. The default is 1.

Environment Variables

- **AB_PROMETHEUS_ENABLE** allows you to collect JMX metrics to monitor and analyze Data Grid and has the following values:

  false

  Disables monitoring with the default Prometheus agent.

  true

  Enables monitoring with the default Prometheus agent. The Prometheus Operator must be installed and running. You must also Set Up Monitoring after you create the service.

- **AB_PROMETHEUS_JMX_EXPORTER_PORT** defines the port on which Data Grid publishes JMX metrics. The default is **9779**.

Verifying the Application

Command output displays parameter values and resources when you create **datagrid-service**, as in the following example:

```
--> Deploying template "rhdg-helloworld/datagrid-service" to project rhdg-helloworld

     Red Hat Data Grid Service
     ---------
     Red Hat Data Grid is an in-memory, distributed key/value store.

     * With parameters:
        ...

--> Creating resources ...
    secret "datagrid-service" created
    service "datagrid-service-ping" created
    service "datagrid-service" created
    statefulset "datagrid-service" created
--> Success
    ...
```

TIP

Try the Hello World Quickstart Tutorial.

CHAPTER 5. INVOKING THE DATA GRID REST API

Data Grid services expose a REST endpoint at port **8443**.

By default, Data Grid requires user authentication for data access and encryption for client connections.

Authentication

Data Grid authorizes data access requests with credentials that you specify with the **APPLICATION_USER** and **APPLICATION_PASSWORD** parameters.

Encryption

When Data Grid pods start they generate TLS certificate/key pairs and save them in the **service-certs** secret. The TLS certificates are signed by the OpenShift certificate authority (CA).

5.1. CREATING EXTERNAL ROUTES TO THE REST API

REST clients running outside OpenShift access Data Grid pods through routes with **reencrypt** termination.

Procedure

1. Create a route with **reencrypt** termination.

```
$ oc create route reencrypt ${ROUTE_NAME} \
  --port=https \
  --service ${APPLICATION_NAME}
```

For example:

```
$ oc create route reencrypt cache-service-https-route \
  --port=https \
  --service cache-service
```

2. Run **oc get routes** to find the HTTPS route hostname, for example:

```
$ oc get routes

NAME                        HOST/PORT
cache-service-https-route   cache-service-https-route-rhdg-helloworld.192.0.2.0.nip.io
```

5.2. MAKING REST CALLS

Prerequisite

- Configure REST clients for authentication and encryption.

  On OpenShift

  Create truststores with the CA bundle mounted in the pod at: **/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt**

  Outside OpenShift

  Create truststores with the CA for your OpenShift environment.

Procedure

- Invoke the Data Grid REST API as appropriate.
  For example, invoke a **PUT** call to add a key:value pair:

  ```
  curl -X PUT \
    -u ${USERNAME}:${PASSWORD} \
    -H 'Content-type: text/plain' \
    -d 'world' \
    https://${HOSTNAME_FOR_HTTPS_ROUTE}/rest/default/hello
  ```

5.2.1. Using the OpenShift CA to Make REST Calls

In cases where the CA certificate is not valid, such as local OpenShift clusters or Red Hat OpenShift Container Platform development installations, you can use **service-ca.crt** to make REST calls.

Procedure

1. Get **service-ca.crt** from Data Grid pods.

   ```
   $ oc rsync ${pod_name}:/var/run/secrets/kubernetes.io/serviceaccount/..data/service-ca.crt .
   ```

2. Pass **service-ca.crt** when you invoke REST calls.

   ```
   curl -X PUT \
     -u ${USERNAME}:${PASSWORD} \
     --cacert service-ca.crt \
     -H 'Content-type: text/plain' \
     -d 'world' \
     https://${HOSTNAME_FOR_HTTPS_ROUTE}/rest/default/hello
   ```

CHAPTER 6. CONFIGURING HOT ROD CLIENTS

Data Grid services expose a Hot Rod endpoint at port **11222**.

By default, Data Grid requires user authentication for data access and encryption for client connections.

Authentication

Data Grid authorizes data access requests with credentials that you specify with the **APPLICATION_USER** and **APPLICATION_PASSWORD** parameters.

Encryption

When Data Grid pods start they generate TLS certificate/key pairs and save them in the **service-certs** secret. The TLS certificates are signed by the OpenShift certificate authority (CA).

6.1. CONFIGURING TRUSTSTORES WITH HOT ROD

Set **trustStorePath** to the location of a valid certificate in PEM format in your Hot Rod client configuration. The Hot Rod Java client builds an in-memory Java keystore with all certificates found in the path.

On OpenShift

- Specify the OpenShift certificate authority (CA) bundle.
  /var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt

Outside OpenShift

1. Get **tls.crt** from the **service-certs** secret.

```
$ oc get secret service-certs \
  -o jsonpath='{.data.tls\.crt}' \
  | base64 -d > tls.crt
```

2. Specify the path to **tls.crt** in your client configuration.

6.2. CLIENT INTELLIGENCE

Client intelligence refers to mechanisms the Hot Rod protocol provides so that clients can locate and send requests to Data Grid pods.

On OpenShift

Clients can access the internal IP addresses for pods so you can use any client intelligence. The default intelligence, **HASH_DISTRIBUTION_AWARE**, is recommended because it allows clients to route requests to primary owners, which improves performance.

Outside OpenShift

Use **BASIC** intelligence only.

Reference

- org.infinispan.client.hotrod.configuration.ClientIntelligence

6.3. CREATING EXTERNAL ROUTES FOR HOT ROD

Hot Rod clients running outside OpenShift access Data Grid pods through routes with **passthrough** termination.

Prerequisites

- Configure Data Grid Server to encrypt client connections.

Procedure

1. Create a route with **passthrough** termination.

   ```
   $ oc create route passthrough ${ROUTE_NAME} \
     --port=hotrod \
     --service ${APPLICATION_NAME}
   ```

   For example:

   ```
   $ oc create route passthrough cache-service-hotrod-route \
     --port=hotrod \
     --service cache-service
   ```

2. Get the Hot Rod route hostname from **.spec.host**.

   ```
   $ oc get route cache-service-hotrod-route -o jsonpath="{.spec.host}"

   cache-service-hotrod-route-rhdg-helloworld.192.0.2.0.nip.io
   ```

6.4. HOSTNAMES FOR DATA GRID SERVICES

Use the hostname for Data Grid that corresponds to the location of your Hot Rod client.

In the Same OpenShift Namespace

Use **APPLICATION_NAME**.

For example:

```
.host("cache-service")
```

In Different OpenShift Namespaces

Use the internal service DNS name in this form:
APPLICATION_NAME.SERVICE_NAMESPACE.svc

For example:

```
.host("cache-service.rhdg-helloworld.svc")
```

Outside OpenShift

Use the Hot Rod route hostname.

For example:

```
.host("cache-service-hotrod-route-rhdg-helloworld.192.0.2.0.nip.io")
```

6.5. CONFIGURING HOT ROD CLIENTS PROGRAMMATICALLY

Use the **ConfigurationBuilder** class to programmatically configure Hot Rod clients to access Data Grid clusters.

1. Call the **create()** method to create a configuration bean that you can pass to the **RemoteCacheManager**.
2. Use the **authentication()** and **ssl()** methods to configure authentication and encryption.

Reference

- org.infinispan.client.hotrod.configuration.ConfigurationBuilder

6.5.1. Hot Rod Configuration Builder On OpenShift

Configuration bean for Hot Rod clients running on OpenShift:

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.addServer()
 // Connection
 .host("${APPLICATION_NAME}.${SERVICE_NAMESPACE}.svc").port(11222)
 .security()
        // Authentication
        .authentication().enable()
        .username("${USERNAME}")
        .password("${PASSWORD}")
        .serverName("${APPLICATION_NAME}")
        .saslMechanism("DIGEST-MD5")
        .saslQop(SaslQop.AUTH)
        // Encryption
        .ssl()
        .trustStorePath(/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt);
```

6.5.2. Hot Rod Configuration Builder Outside OpenShift

Configuration bean for Hot Rod clients running outside OpenShift:

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.addServer()
 // Connection
 .host("${HOTROD_ROUTE_HOSTNAME}").port(443)
 // Use BASIC client intelligence.
 .clientIntelligence(ClientIntelligence.BASIC)
 .security()
        // Authentication
        .authentication().enable()
        .username("${USERNAME}")
        .password("${PASSWORD}")
```

```
            .serverName("${APPLICATION_NAME}")
            .saslMechanism("DIGEST-MD5")
            .saslQop(SaslQop.AUTH)
            // Encryption
            .ssl()
            .sniHostName("${HOTROD_ROUTE_HOSTNAME}")
            .trustStorePath(path/to/tls.crt);
```

6.6. SETTING HOT ROD CLIENT PROPERTIES

Use Hot Rod client configuration properties to specify Data Grid hostnames and ports, authentication details, and TLS certificates.

Procedure

1. Create a **hotrod-client.properties** file that contains your Hot Rod client configuration.
2. Add **hotrod-client.properties** to the classpath.

Reference

- org.infinispan.client.hotrod.configuration

6.6.1. Hot Rod Configuration Properties On OpenShift

Configuration properties for Hot Rod clients running on OpenShift:

```
# Connection
infinispan.client.hotrod.server_list=${APPLICATION_NAME}.${SERVICE_NAMESPACE}.svc:11222

# Authentication
infinispan.client.hotrod.use_auth=true
infinispan.client.hotrod.auth_username=${USERNAME}
infinispan.client.hotrod.auth_password=${PASSWORD}
infinispan.client.hotrod.auth_server_name=${APPLICATION_NAME}
infinispan.client.hotrod.sasl_mechanism=DIGEST-MD5
infinispan.client.hotrod.sasl_properties.javax.security.sasl.qop=auth

# Encryption
infinispan.client.hotrod.trust_store_path=/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
```

6.6.2. Hot Rod Configuration Properties Outside OpenShift

Configuration properties for Hot Rod clients running outside OpenShift:

```
# Connection
infinispan.client.hotrod.server_list=${HOTROD_ROUTE_HOSTNAME}:443

# Use BASIC client intelligence.
infinispan.client.hotrod.client_intelligence=BASIC

# Authentication
infinispan.client.hotrod.use_auth=true
infinispan.client.hotrod.auth_username=${USERNAME}
infinispan.client.hotrod.auth_password=${PASSWORD}
infinispan.client.hotrod.auth_server_name=${APPLICATION_NAME}
infinispan.client.hotrod.sasl_mechanism=DIGEST-MD5
infinispan.client.hotrod.sasl_properties.javax.security.sasl.qop=auth

# Encryption
infinispan.client.hotrod.sni_host_name=${HOTROD_ROUTE_HOSTNAME}
infinispan.client.hotrod.trust_store_path=path/to/tls.crt
```

CHAPTER 7. REMOTELY CREATING CACHES

When you remotely create caches with the **cache-service**, you can configure caches to be ephemeral or permanent and if data is replicated across the cluster.

You can define any custom configuration when you remotely create caches with the **datagrid-service**.

Remotely create cache definitions with the **cache-service** and **datagrid-service** through the Hot Rod protocol as follows:

1. Instantiate the **RemoteCacheManager** class to connect to the service.

2. Call the **createCache()** method to create a cache, as in the following example:

```
private static void createCache(String appName) {
    //Connect to the Hot Rod service.
    final String host = appName;
    //Use the configuration bean.
    ConfigurationBuilder cfg = ...

    System.out.printf("--- Connecting to %s ---%n", appName);

    //Create a new RemoteCacheManager and start it.
    final RemoteCacheManager remote = new RemoteCacheManager(cfg.build());

    //Set a name for the cache.
    final String cacheName = "custom";

    System.out.printf("--- Creating cache in %s ---%n", appName);

    //Perform remote administration operations.
    remote.administration()
      //Include a flag to make the cache permanent.
      .withFlags(CacheContainerAdmin.AdminFlag.PERMANENT)
      //Create a cache on the remote server.
      //Pass null instead of XML to use the default cache configuration.
      .createCache(cacheName, null);

    System.out.printf("--- Cache '%s' created in '%s' ---%n", cacheName, appName);
}
```

NOTE

If the named cache already exists, an exception is thrown. Alternatives are to:

- Call the **getOrCreateCache** method in **RemoteCacheManagerAdmin** to return the cache name instead of throwing an exception.

- Call the **removeCache** method in **RemoteCacheManagerAdmin** to destroy the cache and then call **createCache** again.

TIP

Try one of the Quickstart Tutorials:

- Creating Caches with Cache Service
- Creating Caches with Data Grid Service

CHAPTER 8. DEFINING FILE-BASED CACHE STORES

Define a file-based cache store with **datagrid-service** to persist data to external storage.

Use the **XMLStringConfiguration** class to provide XML configuration as a string through the Hot Rod interface.

- XML must be valid with the Data Grid configuration schema.
- Location of your file store should be under the storage volume mounted at **/opt/datagrid/standalone/data** because the **data** folder is a **PersistentVolume** that allows data to survive when the container restarts.

As an example, the following **main** method creates a cache with a distributed configuration that includes a file store:

```
public static void main(String[] args) {

  ConfigurationBuilder cfg = ...

  RemoteCacheManager rcm = new RemoteCacheManager(build);

  String xml = String.format(
    "<infinispan>" +
      "<cache-container>" +
        "<distributed-cache name=\"%1$s\">" +
          "<persistence passivation=\"false\">" +
            "<file-store " +
              "shared=\"false\" " +
              "fetch-state=\"true\" " +
              "path=\"${jboss.server.data.dir}/datagrid-infinispan/%1$s\"" +
            "/>" +
          "</persistence>" +
        "</distributed-cache>" +
      "</cache-container>" +
    "</infinispan>",
    "cacheName"
  );

  RemoteCache<Object, Object> index = rcm.administration()
  //Include a flag to make the cache permanent.
  .withFlags(CacheContainerAdmin.AdminFlag.PERMANENT)
  //Create a cache with the XML configuration
  .createCache("cacheName", new XMLStringConfiguration(xml));

  System.out.println(index.size());

}
```

For information about valid **file-store** configuration options, see the Data Grid configuration schema .

See the Javadocs for more information:

- org.infinispan.commons.configuration.XMLStringConfiguration
- org.infinispan.client.hotrod.RemoteCacheManager

- org.infinispan.client.hotrod.configuration.ConfigurationBuilder

CHAPTER 9. USING DATA GRID DEPLOYMENT CONFIGURATION TEMPLATES

9.1. DATA GRID DEPLOYMENT CONFIGURATION TEMPLATES

Data Grid provides a set of templates that can help you deploy Data Grid for OpenShift with different configurations.

IMPORTANT

As of Data Grid 7.3, these deployment configuration templates are deprecated. You should use **cache-service** or **datagrid-service** service templates instead. For more information see Red Hat Data Grid Supported Configurations .

Template	Description
datagrid73-basic	Run Data Grid for OpenShift without authentication or encryption.
datagrid73-https	Run Data Grid for OpenShift with an HTTPS route to securely access caches. Requires an OpenShift secret for encrypting network traffic.
datagrid73-mysql	Run Data Grid for OpenShift with a MySQL database as an ephemeral cache store. Requires an OpenShift secret for encrypting network traffic.
datagrid73-mysql-persistent	Run Data Grid for OpenShift with a MySQL database as a persistent cache store. Requires an OpenShift secret for encrypting network traffic.
datagrid73-postgresql	Run Data Grid for OpenShift with a PostgreSQL database as an ephemeral cache store. Requires an OpenShift secret for encrypting network traffic.
datagrid73-postgresql-persistent	Run Data Grid for OpenShift with a PostgreSQL database as a persistent cache store. Requires an OpenShift secret for encrypting network traffic.
datagrid73-partition	Run Data Grid for OpenShift with a partitioned data directory that preserves metadata for cache entries when the pod restarts.

9.2. IMPORTING DEPLOYMENT CONFIGURATION TEMPLATES

Import the Data Grid for OpenShift deployment configuration templates into OpenShift as follows:

1. Log in to your OpenShift cluster.

2. Import a specific template or all templates.

   - Import a specific template:

     ```
     $ oc create -f \
     https://raw.githubusercontent.com/jboss-container-images/jboss-datagrid-7-openshift-
     image/7.3-v1.8/templates/datagrid73-mysql.json
     ```

   - Import all templates:

     ```
     $ for resource in datagrid73-image-stream.json \
       datagrid73-basic.json \
       datagrid73-https.json \
       datagrid73-mysql-persistent.json \
       datagrid73-mysql.json \
       datagrid73-partition.json \
       datagrid73-postgresql.json \
       datagrid73-postgresql-persistent.json
     do
       oc create -f \
       https://raw.githubusercontent.com/jboss-container-images/jboss-datagrid-7-openshift-
     image/7.3-v1.8/templates/${resource}
     done
     ```

   TIP

   Use **oc create** to import new templates. Use **oc replace --force** to overwrite existing templates.

   Specify the namespace into which to import the templates with the **-n** option. For example, **-n openshift** imports resources into the global **openshift** namespace and requires administrative permissions.

3. Import the Data Grid image.

   ```
   $ oc -n openshift import-image jboss-datagrid73-openshift:1.9
   ```

4. Verify the templates are available on OpenShift.

   ```
   $ oc get templates -n openshift | grep datagrid73
   ```

9.3. IMPORTING OPENSHIFT SECRETS

Some Data Grid for OpenShift deployment configuration templates require HTTPS and JGroups keystores.

Data Grid for OpenShift provides HTTPS and JGroups keystores that you can import for evaluation purposes. However, you should not use this secret in production environments.

Import the secret with the keystores into your project namespace as follows:

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

For more information, see:

- Configuring Authentication and Encryption
- Configuring JGroups Encryption

9.4. DEPLOYING DATA GRID FOR OPENSHIFT

1. Create a new deployment with the **new-app** command.
2. Specify a template with the **--template** option.
3. Set environment variables to configure the deployment with the **-e** option.
   For example, to create a deployment with the **datagrid73-basic** template that includes a cache named **mycache** that starts eagerly, run the following command:

   ```
   $ oc new-app --template=datagrid73-basic \
     -p USERNAME=${USERNAME} \
     -p PASSWORD=${PASSWORD} \
     -p CACHE_NAMES=mycache \
     -e MYCACHE_CACHE_START=EAGER
   ```

   See Environment Variables for information about supported environment variables.

9.5. CONFIGURING DATA GRID FOR OPENSHIFT

After you create a Data Grid for OpenShift deployment, you can configure it with environment variables.

For example, you have a deployment configuration (**dc**) named **datagrid-app** with a cache named **mycache**. Configure **mycache** to start lazily as follows:

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

When you modify a deployment configuration, the replication controller deploys a new version. Get the updated deployment configuration as follows:

```
$ 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
```

Verify configuration changes as follows:

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

# pods datagrid-app-2-<id>, container datagrid-app
CACHE_NAMES=mycache
MYCACHE_CACHE_START=LAZY
```

```
PASSWORD=${PASSWORD}
USERNAME=${USERNAME}
...
```

CHAPTER 10. SETTING UP MONITORING

Collect JMX metrics with the Prometheus Operator to monitor events and get statistics for Data Grid clusters.

From a high-level, you set up monitoring capabilities as follows:

1. Configure Data Grid with the **AB_PROMETHEUS_ENABLE** environment variable set to **true**.
2. Install the Prometheus Operator and expose the Prometheus web UI.
3. Export Data Grid metrics to Prometheus.
4. Enable Prometheus to monitor Data Grid for metrics.

10.1. DEPLOYING THE PROMETHEUS OPERATOR

To install the Prometheus Operator, you should refer to the following documentation:

- Prometheus Operator
- Getting Started
- Prometheus RBAC

TIP

Try the Monitoring Data Grid Quickstart Tutorial.

10.2. EXPOSING DATA GRID METRICS TO PROMETHEUS

Add a service that exposes JMX metrics from Data Grid to Prometheus.

1. Create a **service-metrics.yaml** file that defines a metrics service.

```
apiVersion: v1
kind: Service
metadata:
  annotations:
    description: Expose Data Grid metrics to Prometheus.
  labels:
    app: datagrid-service
    application: datagrid-service
    template: datagrid-service
    metrics: datagrid
  name: datagrid-app-metrics
spec:
  ClusterIP: None
  ports:
      # Set the port name where Data Grid publishes metrics.
      # You add the port name to service-monitor.yaml.
    - name: web
      port: 8080
      protocol: TCP
      targetPort: 9779
  selector:
    deploymentConfig: datagrid-service
  sessionAffinity: None
```

2. Apply **service-metrics.yaml**.

```
$ oc apply -f service-metrics.yaml
```

10.3. ENABLING PROMETHEUS TO MONITOR DATA GRID

A service monitor lets Prometheus connect to the Data Grid metrics service.

1. Create a **service-monitor.yaml** file that holds the definition for a **ServiceMonitor** object.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  name: datagrid-service-monitor
  labels:
    team: frontend
spec:
  selector:
    matchLabels:
      metrics: datagrid
  endpoints:
      # Set the name of the port where Data Grid publishes metrics.
      # You create this port in service-metrics.yaml.
    - port: web
```

2. Apply **service-monitor.yaml**.

```
$ oc apply -f service-monitor.yaml
```

CHAPTER 11. CONFIGURING DATA GRID FOR OPENSHIFT CLUSTERS

11.1. CONFIGURING CLUSTER DISCOVERY

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

Data Grid for OpenShift templates and services use DNS by default. If you deploy Data Grid for OpenShift directly from the image or custom template, you must configure the appropriate discovery mechanism.

11.1.1. Configuring DNS_PING

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_SERVICE_NAME}
```

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-service
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.

11.1.2. Configuring KUBE_PING

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=${PING_NAMESPACE}
```

3. Specify a cluster label with 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=labelKey=labelValue
```

4. Grant authorization to the service account the pod is running under so that it can access the Kubernetes REST API. For example, grant authorization to ***datagrid-service-account*** as follows:

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

11.2. CONFIGURING JGROUPS ENCRYPTION

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.

11.2.1. 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 to your OpenShift cluster.

   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
   ```

   d. Mount the secret to the container.

   ```
   $ oc set volumes dc/datagrid \
     --add -t secret \
     --secret-name='jgroups-secret' \
     --mount-path='/keystores/jgroups'
   ```

3. Set the value of the **JGROUPS_ENCRYPT_PROTOCOL** environment variable to **SYM_ENCRYPT** for each node in the cluster.

4. Configure each node in the cluster to use the JGroups keystore with the following environment variables:

   JGROUPS_ENCRYPT_KEYSTORE

   Specifes the JGroups keystore for encrypting cluster traffic.

JGROUPS_ENCRYPT_KEYSTORE_DIR
: Specifies the directory where the JGroups keystore resides.

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.

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

11.2.2. 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** for each node in the cluster.

CHAPTER 12. CUSTOMIZING DATA GRID FOR OPENSHIFT

Use the Data Grid image with custom configuration either with the Source-to-Image (S2I) process or via the **ConfigMap** API.

NOTE

Red Hat encourages you to use the Data Grid for OpenShift image and **datagrid-service** and **cache-service** templates to create Data Grid clusters. While it is possible to create Data Grid pods with custom configuration, **datagrid-service** and **cache-service** are designed for high performance and are suitable for a wide range of use cases with little to no configuration required.

Go to Setting Up Data Grid for OpenShift Services and learn how to quickly create Data Grid clusters.

Source-to-Image (S2I)

Use the S2I process and binary builds to create custom Data Grid images.
Add cache definitions and endpoint configurations to **openshift-clustered.xml** then use S2I capabilities to build a custom Data Grid image that uses that configuration file. You can then create Data Grid pods with the image as required.

To modify the configuration you must build a new image. However, when you rebuild your custom image, the Data Grid pods automatically redeploy with the new configuration changes.

ConfigMap API

Use the Red Hat OpenShift **ConfigMap** API to dynamically configure Data Grid pods.
Define custom configuration in **standalone.xml** that maps to a **ConfigMap** object in the namespace as the Data Grid pod. You can modify the Data Grid configuration and then redeploy pods to load configuration changes. However, Data Grid pods do not automatically redeploy when you modify **standalone.xml**. You must manually redeploy pods to load the configuration changes.

- You should explicitly define all cache and endpoint configuration in **standalone.xml** before you create Data Grid pods.
  Environment variables for cache and endpoint configuration do not take effect unless the placeholder exists before deployment. For example, the following is a placeholder for the **JGROUPS_PING_PROTOCOL**:

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

  See **clustered-openshift.xml** to review available placeholders.

- To encrypt client to server traffic, you must configure the server identity in **standalone.xml**.

- The **DATAGRID_SPLIT** environment variable does not take effect with Data Grid for OpenShift pods that you customize via the **ConfigMap** API. These pods cannot use shared persistent volumes with **DATAGRID_SPLIT**.

12.1. CLONING THE DATA GRID EXAMPLES

1. Clone the Data Grid for OpenShift image source repository.

```
$ git clone git@github.com:jboss-container-images/jboss-datagrid-7-openshift-image.git .
```

2. View the contents of the **docs/examples** directory, for example:

```
$ cd jboss-datagrid-7-openshift-image/docs/examples/s2i/configuration

$ tree
.
├── s2i
│   └── configuration
│       └── clustered-openshift.xml
└── user-configuration
    ├── standalone.xml
    └── user-config-template.yaml
```

S2I

Injects **clustered-openshift.xml** to **${JBOSS_HOME}/standalone/configuration/** inside the Data Grid for OpenShift image.

TIP

Add custom **logging.properties** and **application-role.properties** to the **configuration** directory to include them when you build a custom image.

ConfigMap

Maps **standalone.xml** to the **/opt/datagrid/standalone/configuration/user** directory inside the Data Grid for OpenShift image.

12.2. CREATING S2I BUILDS WITH CUSTOM CONFIGURATION

12.2.1. Setting Up the Data Grid Image

You need the Data Grid for OpenShift image as source for your customization.

1. Confirm if the Data Grid for OpenShift image is available.

```
$ oc get images | grep jboss-datagrid73-openshift
```

2. If the image is not available, create an image stream and import it.

```
$ oc create -f https://raw.githubusercontent.com/jboss-container-images/jboss-datagrid-7-openshift-image/7.3-v1.9/templates/datagrid73-image-stream.json

$ oc import-image jboss-datagrid73-openshift --from='registry.redhat.io/jboss-datagrid-7/datagrid73-openshift:1.9'
```

12.2.2. Creating Custom Data Grid Images

1. Create a new binary build using the Data Grid for OpenShift image.

```
$ oc new-build jboss-datagrid73-openshift:1.9 --binary=true --name=custom-datagrid
```

2. Navigate to the **s2i** directory so you can pass **--from-dir="."**. You must include the **configuration** directory so that the S2I process can detect your custom configuration. To use the example configuration, do the following:

```
$ cd jboss-datagrid-7-openshift-image/docs/examples/s2i/
```

3. Build the Data Grid for OpenShift image with your custom configuration.

```
$ oc start-build custom-datagrid --from-dir="." --follow

Uploading directory "." as binary input for the build ...

...
Push successful
```

4. Check that your image is available.

```
$ oc get images | grep custom-datagrid
```

12.2.3. Verifying Custom Data Grid Images

1. Confirm that the build pod is running.

```
$ oc get pods

NAME                    READY     STATUS      RESTARTS   AGE
custom-datagrid-1-build   0/1       Completed   0          38s
```

2. Create a Data Grid application with your custom configuration.

```
$ oc new-app custom-datagrid \
  -p APPLICATION_USER=${USERNAME} \
  -p APPLICATION_PASSWORD=${PASSWORD}
```

3. Wait for your custom Data Grid application to start running.

```
$ oc get pods -w

NAME                    READY     STATUS        RESTARTS   AGE
custom-datagrid-1-build   0/1       Completed     0          8m
custom-datagrid-1-<id>    1/1       Running       0          11s
```

4. Remotely access the bash shell for your custom Data Grid pod.

```
$ oc exec -it ${pod-name} -- /bin/bash
```

5. View **clustered-openshift.xml** to verify the configuration, for example:

```
$ cat /opt/datagrid/configuration/clustered-openshift.xml
```

If **clustered-openshift.xml** contains your custom configuration then the Data Grid pod is using it. You can optionally use the Data Grid command line interface to verify your configuration, for example:

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

6. End the remote session after you verify your configuration.

```
$ exit
```

12.3. CREATING CUSTOM DATA GRID FOR OPENSHIFT PODS WITH THE CONFIGMAP API

1. Create a custom template for your Data Grid for OpenShift pod.
   a. Expose the required ports and services in your template.
   b. Add a **configMap** object to the custom template.
   c. Add a config volume for the container at **/opt/datagrid/standalone/configuration/user**.
   d. Import your custom template into OpenShift.
      To use the example template, do the following:

      ```
      $ cd jboss-datagrid-7-openshift-image/docs/examples/user-configuration/
      ```

      ```
      $ oc create -f user-config-template.yaml
      ```

2. Create a ConfigMap in your OpenShift project, for example:

   ```
   $ oc create configmap user-config --from-file="."
   ```

3. Create Data Grid pods with your custom configuration.

   ```
   $ oc new-app user-config \
     -p APPLICATION_NAME=${USERNAME} \
     -e USER_CONFIG_MAP=true
   ```

   Where:

   - **APPLICATION_NAME** is a required parameter in the example template and defaults to **custom-datagrid**.
   - **USER_CONFIG_MAP=true** applies the ConfigMap to the Data Grid pod. This is set in the example template as follows:

     ```
     - env:
       - name: USER_CONFIG_MAP
         value: "true"
     ```

12.3.1. Verifying Custom Data Grid for OpenShift Pods with the ConfigMap API

1. Wait for your custom Data Grid application to start running.

```
$ oc get pods -w

NAME            READY     STATUS    RESTARTS   AGE
user-config-0   0/1       Running   7          17m
```

2. Check the container logs.

```
$ oc logs ${pod-name} | grep standalone.xml

INFO Running jboss-datagrid-7/datagrid73-openshift image, version 1.9 with user
standalone.xml
```

TIP

Try the Customizing Data Grid Service Deployments Quickstart Tutorial.

12.4. CONFIGURING PERSISTENT DATASOURCES

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

12.4.1. 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 Data Grid for OpenShift image, the launch script creates a separate datasource for each JNDI mapping. The 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:

${POOL_NAME}-${DATABASE_TYPE}=${PREFIX}

- **${POOL_NAME}** 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.

12.4.1.1. 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.

12.4.1.2. 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.

12.4.2. 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
```

12.4.3. Datasource 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, **EXAMPLE_MYSQL_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, **CLOUD_DATABASE=myDatabase**.

${PREFIX}_DRIVER

Defines Java database driver for the datasource.
For example, **CLOUD_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, **CLOUD_CONNECTION_CHECKER=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker**

${PREFIX}_EXCEPTION_SORTER

Specifies the exception sorter class that detects and cleans up after fatal database connection exceptions.
For example, **CLOUD_EXCEPTION_SORTER=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter**

${PREFIX}_JNDI

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

For example, **CLOUD_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, **CLOUD_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, **CLOUD_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, **CLOUD_XA_CONNECTION_PROPERTY_DatabaseName=/opt/eap/standalone/data/databases/db/accounts**

This example adds the following to the configuration:

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

CHAPTER 13. EMBEDDED DATA GRID (LIBRARY MODE)

Embedding Data Grid in a custom OpenShift application, or running in Library Mode, is intended for specific uses only:

- Using local or distributed caching in custom Java applications to retain full control of the cache lifecycle. Additionally, when using features that are available only with embedded Data Grid such as distributed streams.
- Reducing network latency to improve the speed of cache operations. Although it is worth noting that the Hot Rod protocol provides near-cache capabilities that achieve equivalent performance of a standard client-server architecture.

Embedding Data Grid also has some limitations:

- Persistence to a cache store is not currently supported. You can embed Data Grid for in-memory caching only.
- Only **DNS_PING** is supported for clustering.
- **TCP** is supported for the ping port.
- **UDP** is not supported with embedded Data Grid.

IMPORTANT

Red Hat highly discourages embedding Data Grid to build custom caching servers to handle remote client requests. Use the **datagrid-service** server implementation. It is fully supported and benefits from regular updates with performance improvements and security fixes.

TIP

Try the Embedded Data Grid Quickstart.

CHAPTER 14. TROUBLESHOOTING DATA GRID FOR OPENSHIFT

14.1. LAUNCHING THE COMMAND LINE INTERFACE (CLI)

Access the Data Grid Management CLI to troubleshoot Data Grid for OpenShift as follows:

1. Open a remote shell session to the running pod.

    ```
    $ oc rsh ${POD_NAME}
    ```

2. Launch the Data Grid CLI from the remote shell.

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

NOTE

The Data Grid Management CLI is bound to the pod in which it runs. Changes that you make with the CLI are not saved if the container restarts.

14.2. VIEWING POD LOGS

Run the following command to view log messages for a running pod:

```
$ oc logs -f ${POD_NAME}
```

CHAPTER 15. REFERENCE

15.1. PROBES

Data Grid for OpenShift provides a liveness probe and a readiness probe to perform container health checks.

Liveness probe

The liveness probe is located in the container at **/opt/datagrid/bin/livenessProbe.sh**. The liveness probe tests server status and restarts pods if the following events occur:

- Data Grid for OpenShift boots with errors.
- Custom deployment configurations do not successfully deploy.
- One or more caches fail to instantiate, which usually occurs if the cache configuration is not valid.

Readiness probe

The readiness probe is located in the container at **/opt/datagrid/bin/readinessProbe.sh**. The readiness probe determines if the pod is ready to receive requests and checks Data Grid cache level **MBeans** to ensure the following:

- All cache instances are initialized.
- All cache instances have joined the cluster, if using distributed cache mode.
- Initial state transfer is complete. If state transfer is in progress, the pod is not marked as ready.
- All cache instances in the Cache manager are running.

To configure custom deployments to use the liveness probe and readiness probe, run the following commands:

```
$ oc set probe dc/datagrid \
  --readiness \
  -- /bin/bash \
  -c /opt/datagrid/bin/readinessProbe.sh

$ oc set probe dc/datagrid \
  --liveness \
  -- /bin/bash \
  -c /opt/datagrid/bin/livenessProbe.sh
```

15.2. PORTS

Data Grid for OpenShift uses the following ports:

Port Number	Protocol	Use
8080	TCP	HTTP Access
8443	TCP	HTTPS Access
8888	TCP	JGroups Ping
11222	TCP	Hot Rod Access

Data Grid deployment configuration templates also use the following ports:

Port Number	Protocol	Use
11211	TCP	Memcached Access
11333	TCP	External Hot Rod Access
8778	TCP	Remote JMX Access

NOTE

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

15.3. ADMINISTRATION CONSOLE

The Data Grid Administration Console is not supported on Red Hat OpenShift.

To monitor events and get statistics for Data Grid for OpenShift clusters, you should use Prometheus. See Setting Up Monitoring.

You can also use the Management CLI to troubleshoot Data Grid for OpenShift pods. See Launching the Command Line Interface.

15.4. ENVIRONMENT VARIABLES

15.4.1. Monitoring and Logging

AB_PROMETHEUS_ENABLE

Allows you to collect JMX metrics to monitor and analyze Data Grid. The default value is **false**. Set the value to **true** to enable monitoring with the default Prometheus agent.
The Prometheus Operator must be installed and running. You must also Set Up Monitoring.

AB_PROMETHEUS_JMX_EXPORTER_PORT

Defines the port on which Data Grid publishes JMX metrics. The default is **9779**.

LOGGING_CATEGORIES

Adjusts the categories and levels for which Data Grid captures log messages, for example:

```
$ LOGGING_CATEGORIES=org.infinipan.core=WARN,org.infinispan.config=DEBUG
```

Log categories correspond to Java package names and use standard log levels: **TRACE**, **DEBUG**, **INFO**, **WARN**, **ERROR**, and **FATAL**.

IMPORTANT

If you specify **LOGGING_CATEGORIES**, Data Grid does not set the following default loggers. Instead, Data Grid uses the default level of **INFO** for all packages that you do not explicitly specify with **LOGGING_CATEGORIES**.

Table 15.1. Default Loggers

Category	Level
com.arjuna	**WARN**
sun.rmi	**WARN**
org.jboss.as.config	**DEBUG**

15.4.2. Containers

USERNAME

Creates a user in the security realm who is authorized to access data.

NOTE

By default, the Hot Rod endpoint uses the **ApplicationRealm** security realm and the REST endpoint uses the **jdg-openshift** security realm.

PASSWORD

Specifies the password for the 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**.

JAVA_OPTS_APPEND

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

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**.
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.

15.4.3. Caches

Creating and configuring caches with **cache-service** and **datagrid-service**

Do not use environment variables to create and configure caches with **cache-service** or **datagrid-service**.

These environment variables are intended for use with the deployment configuration templates only and are deprecated.

You should remote create caches with **cache-service** and **datagrid-service** dynamically through the Hot Rod endpoint. For more information, see Remotely Creating Caches.

CACHE_NAMES

Defines cache instances in your configuration.
If you are using the Data Grid deployment configuration templates and you do not define any cache instances, the launch script adds a default 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.

15.4.3.1. Cache Container Security Configuration

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.

CONTAINER_SECURITY_ROLES

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

15.4.3.2. Named Cache Configuration

You specify the cache name as a prefix for the environment variable 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, 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, Data Grid evicts unused entries.

${CACHE_NAME}_CACHE_MEMORY_STORAGE_TYPE

Defines how 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

Storage Type	Description	Eviction Type	Policy
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 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 Data Grid performs eviction. You can set the following values:

Strategy	Description
NONE	Data Grid does not evict entries. This is the default setting unless you configure eviction.
REMOVE	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.
MANUAL	Data Grid does not perform eviction. Eviction takes place manually by invoking the **evict()** method from the **Cache** API.
EXCEPTION	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, 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, **MYCACHE_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, **MYCACHE_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, **MYCACHE_CACHE_SECURITY_AUTHORIZATION_ROLES=admin, reader, writer**

${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 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.
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**.

${CACHE_NAME}_CACHE_PARTITION_MERGE_POLICY

Configures how 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 Data Grid for OpenShift templates.

15.4.4. Security Domain

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**.

15.4.5. Endpoints

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

Clients that run in the same project as 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 Data Grid for OpenShift, they need to access the Data Grid cluster using an OpenShift service that exposes the Hot Rod 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 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 OpenShift service for the external Hot Rod connector.
The external Hot Rod connector is available with deployment configuration templates only if you define this environment variable. **cache-service** and **datagrid-service** do not use the external Hot Rod connector.

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

REST_STORE_AS_STRING

Specifies if 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

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

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

If you upgrade Data Grid for OpenShift images to version 7.2 or later, 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**.