Configure and run Data Grid services on OpenShift
Abstract

Data Grid Operator provides operational intelligence and reduces management complexity for deploying Data Grid on OpenShift.
# Table of Contents

**RED HAT DATA GRID** .................................................. 5

**DATA GRID DOCUMENTATION** ......................................... 6

**DATA GRID DOWNLOADS** ............................................... 7

**MAKING OPEN SOURCE MORE INCLUSIVE** ............................. 8

**CHAPTER 1. INSTALLING DATA GRID OPERATOR** .......................... 9

1.1. INSTALLING DATA GRID OPERATOR ON RED HAT OPENSHIFT ..... 9

1.2. INSTALLING DATA GRID OPERATOR FROM THE COMMAND LINE ... 11

1.3. DATA GRID CLUSTER UPGRADES ...................................... 11

Upgrade notifications

**CHAPTER 2. GETTING STARTED WITH INFINISPAN CR** .................. 12

2.1. INFINISPAN CUSTOM RESOURCE (CR) .................................. 12

2.2. CREATING DATA GRID CLUSTERS ..................................... 12

2.3. VERIFYING DATA GRID CLUSTERS ..................................... 13

2.4. STOPPING AND STARTING DATA GRID CLUSTERS .................... 14

**CHAPTER 3. SETTING UP DATA GRID SERVICES** ....................... 15

3.1. SERVICE TYPES ................................................... 15

3.1.1. Data Grid service ............................................. 15

3.1.2. Cache service ................................................ 15

3.2. CREATING DATA GRID SERVICE NODES ............................ 15

3.2.1. Data Grid service CR .......................................... 16

3.3. CREATING CACHE SERVICE NODES ................................ 18

3.3.1. Configuring automatic scaling .................................. 18

3.3.2. Configuring the number of owners ................................ 19

3.3.3. Cache service CR ............................................. 19

3.4. ALLOCATING STORAGE RESOURCES .................................. 21

3.4.1. Persistent volume claims ........................................ 22

3.5. JVM, CPU, AND MEMORY .......................................... 22

3.6. ADJUSTING LOG LEVELS ............................................. 23

3.6.1. Logging reference .............................................. 23

3.7. ADDING LABELS TO DATA GRID RESOURCES ....................... 24

**CHAPTER 4. CONFIGURING AUTHENTICATION** ......................... 26

4.1. DEFAULT CREDENTIALS ............................................. 26

4.2. RETRIEVING CREDENTIALS ......................................... 26

4.3. ADDING CUSTOM USER CREDENTIALS ................................ 26

4.4. CHANGING THE OPERATOR PASSWORD .............................. 27

4.5. DISABLING USER AUTHENTICATION ................................ 27

**CHAPTER 5. CONFIGURING CLIENT CERTIFICATE AUTHENTICATION** .... 29

5.1. CLIENT CERTIFICATE AUTHENTICATION ............................ 29

5.2. ENABLING CLIENT CERTIFICATE AUTHENTICATION ................ 29

5.3. PROVIDING CLIENT TRUSTSTORES .................................. 30

5.4. PROVIDING CLIENT CERTIFICATES .................................. 31

**CHAPTER 6. CONFIGURING ENCRYPTION** ............................... 32

6.1. ENCRYPTION WITH RED HAT OPENSHIFT SERVICE CERTIFICATES .... 32

6.2. RETRIEVING TLS CERTIFICATES .................................... 32

6.3. DISABLING ENCRYPTION ............................................ 33
12.7. ADDING PERSISTENT CACHE STORES

CHAPTER 13. RUNNING BATCH OPERATIONS .................................................. 64

CHAPTER 14. RUNNING INLINE BATCH OPERATIONS ....................................... 65
  14.1. CREATING CONFIGMAPS FOR BATCH OPERATIONS .......................... 65
  14.2. RUNNING BATCH OPERATIONS WITH CONFIGMAPS ....................... 66
  14.3. BATCH STATUS MESSAGES .............................................................. 67
  14.4. EXAMPLE BATCH OPERATIONS ...................................................... 67
        14.4.1. Caches .............................................................................. 67
        14.4.2. Counters .......................................................................... 68
        14.4.3. Protobuf schema ................................................................. 68
        14.4.4. Tasks ................................................................................ 68

CHAPTER 15. BACKING UP AND RESTORING DATA GRID CLUSTERS ................. 70
  15.1. RESTORING DATA GRID CLUSTERS .................................................. 71

CHAPTER 16. DEPLOYING CUSTOM CODE TO DATA GRID .............................. 73
  16.1. COPYING CODE ARTIFACTS ............................................................ 73
  16.2. DEPLOYING CUSTOM CODE ............................................................. 74

CHAPTER 17. SENDING CLOUD EVENTS FROM DATA GRID CLUSTERS .......... 76
  17.1. TECHNOLOGY PREVIEWS .................................................................. 76
  17.2. CLOUD EVENTS ............................................................................ 76
  17.3. ENABLING CLOUD EVENTS ............................................................ 76

CHAPTER 18. ESTABLISHING REMOTE CLIENT CONNECTIONS ..................... 78
  18.1. CLIENT CONNECTION DETAILS ....................................................... 78
  18.2. DATA GRID CACHES ...................................................................... 78
  18.3. CONNECTING THE DATA GRID CLI .................................................. 79
  18.4. ACCESSING DATA GRID CONSOLE .................................................. 80
  18.5. HOT ROD CLIENTS ........................................................................... 80
        18.5.1. Hot Rod client configuration API ............................................. 80
                On OpenShift ........................................................................ 81
                Outside OpenShift ................................................................ 81
        18.5.2. Hot Rod client properties ....................................................... 82
                On OpenShift ........................................................................ 82
                Outside OpenShift ................................................................ 82
        18.5.3. Configuring Hot Rod clients for certificate authentication ........ 83
        18.5.4. Creating caches from Hot Rod clients ...................................... 83
                Programmatically creating caches ............................................. 83
                Using Hot Rod client properties ............................................. 84
  18.6. ACCESSING THE REST API ............................................................... 84
  18.7. ADDING CACHES TO CACHE SERVICE NODES .................................... 85
        18.7.1. Default cache configuration .................................................... 85

Table of Contents
RED HAT DATA GRID

Data Grid is a high-performance, distributed in-memory data store.

**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.
DATA GRID DOCUMENTATION

Documentation for Data Grid is available on the Red Hat customer portal.

- Data Grid 8.2 Documentation
- Data Grid 8.2 Component Details
- Supported Configurations for Data Grid 8.2
- Data Grid 8 Feature Support
- Data Grid Deprecated Features and Functionality
DATA GRID DOWNLOADS

Access the Data Grid Software Downloads on the Red Hat customer portal.

NOTE

You must have a Red Hat account to access and download Data Grid software.
MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message.
CHAPTER 1. INSTALLING DATA GRID OPERATOR

Install Data Grid Operator into a OpenShift namespace to create and manage Data Grid clusters.

1.1. INSTALLING DATA GRID OPERATOR ON RED HAT OPENSIFT

Create subscriptions to Data Grid Operator on OpenShift so you can install different Data Grid versions and receive automatic updates.

Automatic updates apply to Data Grid Operator first and then for each Data Grid node. Data Grid Operator updates clusters one node at a time, gracefully shutting down each node and then bringing it back online with the updated version before going on to the next node.

Prerequisites

- Access to OperatorHub running on OpenShift. Some OpenShift environments, such as OpenShift Container Platform, can require administrator credentials.
- Have an OpenShift project for Data Grid Operator if you plan to install it into a specific namespace.

Procedure

1. Log in to the OpenShift Web Console.
2. Navigate to OperatorHub.
3. Find and select Data Grid Operator.
4. Select Install and continue to Create Operator Subscription.
5. Specify options for your subscription.
   - **Installation Mode**
     - You can install Data Grid Operator into a Specific namespace or All namespaces.
   - **Update Channel**
     - Get updates for Data Grid Operator 8.2.x.
   - **Approval Strategies**
     - Automatically install updates from the 8.2.x channel or require approval before installation.
6. Select Subscribe to install Data Grid Operator.
7. Navigate to Installed Operators to verify the Data Grid Operator installation.

1.2. INSTALLING DATA GRID OPERATOR FROM THE COMMAND LINE

As an alternative to installing Data Grid Operator through the OperatorHub on OpenShift, use the oc client to create subscriptions.

Prerequisites

- Have an oc client.
Procedure

1. Set up projects.
   a. Create a project for Data Grid Operator.
   b. If you want Data Grid Operator to control a specific Data Grid cluster only, create a project for that cluster.

```
$ oc new-project ${INSTALL_NAMESPACE}
$ oc new-project ${WATCH_NAMESPACE}
```

1. Creates a project into which you install Data Grid Operator.
2. Optionally creates a project for a specific Data Grid cluster if you do not want Data Grid Operator to watch all projects.

2. Create an **OperatorGroup** resource.

**Control all Data Grid clusters**

```
$ oc apply -f - << EOF
    apiVersion: operators.coreos.com/v1
    kind: OperatorGroup
    metadata:
        name: datagrid
        namespace: ${INSTALL_NAMESPACE}
EOF
```

**Control a specific Data Grid cluster**

```
$ oc apply -f - << EOF
    apiVersion: operators.coreos.com/v1
    kind: OperatorGroup
    metadata:
        name: datagrid
        namespace: ${INSTALL_NAMESPACE}
    spec:
        targetNamespaces:
        - ${WATCH_NAMESPACE}
EOF
```

3. Create a subscription for Data Grid Operator.

```
$ oc apply -f - << EOF
    apiVersion: operators.coreos.com/v1alpha1
    kind: Subscription
    metadata:
        name: datagrid-operator
        namespace: ${INSTALL_NAMESPACE}
    spec:
        channel: 8.2.x
        installPlanApproval: Automatic
        name: datagrid
EOF
```
NOTE

If you want to manually approve updates from the 8.2.x channel, change the value of the `spec.installPlanApproval` field to `Manual`.

4. Verify the installation.

```
$ oc get pods -n ${INSTALL_NAMESPACE}
NAME                                   READY   STATUS
infinispan-operator-<id>               1/1     Running
```

1.3. DATA GRID CLUSTER UPGRADES

Data Grid Operator can automatically upgrade Data Grid clusters when new versions become available. You can also perform upgrades manually if you prefer to control when they occur.

Upgrade notifications

If you upgrade Data Grid clusters manually and have upgraded the channel for your Data Grid Operator subscription from 8.1.x to 8.2.x you should apply the upgrade for the latest Data Grid 8.2.x version as soon as possible to avoid potential data loss that can result from an issue in 8.2.0. For more information, see Data Grid Operator 8.2 Release Notes.
CHAPTER 2. GETTING STARTED WITH INFINISPAN CR

After you install Data Grid Operator, learn how to create Data Grid clusters on OpenShift.

2.1. INFINISPAN CUSTOM RESOURCE (CR)

Data Grid Operator adds a new Custom Resource (CR) of type Infinispan that lets you handle Data Grid clusters as complex units on OpenShift.

Data Grid Operator watches for Infinispan Custom Resources (CR) that you use to instantiate and configure Data Grid clusters and manage OpenShift resources, such as StatefulSets and Services. In this way, the Infinispan CR is your primary interface to Data Grid on OpenShift.

The minimal Infinispan CR is as follows:

```yaml
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
spec:
  replicas: 2
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>apiVersion</td>
<td>Declares the version of the Infinispan API.</td>
</tr>
<tr>
<td>kind</td>
<td>Declares the Infinispan CR.</td>
</tr>
<tr>
<td>metadata.name</td>
<td>Specifies a name for your Data Grid cluster.</td>
</tr>
<tr>
<td>spec.replicas</td>
<td>Specifies the number of nodes in your Data Grid</td>
</tr>
<tr>
<td></td>
<td>cluster.</td>
</tr>
</tbody>
</table>

2.2. CREATING DATA GRID CLUSTERS

Use Data Grid Operator to create clusters of two or more Data Grid nodes.

Prerequisites

- Install Data Grid Operator.
- Have an oc client.

Procedure

1. Specify the number of Data Grid nodes in the cluster with spec.replicas in your Infinispan CR. For example, create a cr_minimal.yaml file as follows:

   ```bash
   $ cat > cr_minimal.yaml
   apiVersion: infinispan.org/v1
   kind: Infinispan
   ```
2. Apply your **Infinispan CR**.

```bash
$ oc apply -f cr_minimal.yaml
```

3. Watch Data Grid Operator create the Data Grid nodes.

```bash
$ oc get pods -w
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-infinispan-1</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>4s</td>
</tr>
<tr>
<td>example-infinispan-2</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>4s</td>
</tr>
<tr>
<td>example-infinispan-3</td>
<td>0/1</td>
<td>ContainerCreating</td>
<td>0</td>
<td>5s</td>
</tr>
<tr>
<td>infinispan-operator-0</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>3m</td>
</tr>
<tr>
<td>example-infinispan-3</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>8s</td>
</tr>
<tr>
<td>example-infinispan-2</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>8s</td>
</tr>
<tr>
<td>example-infinispan-1</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>8s</td>
</tr>
</tbody>
</table>

**Next Steps**

Try changing the value of `replicas:` and watching Data Grid Operator scale the cluster up or down.

### 2.3. VERIFYING DATA GRID CLUSTERS

Check that Data Grid nodes have successfully formed clusters.

**Procedure**

- Retrieve the **Infinispan CR** for Data Grid Operator.

```bash
$ oc get infinispan -o yaml
```

The response indicates that Data Grid nodes have received clustered views, as in the following example:

```json
conditions:
- message: 'View: [example-infinispan-0, example-infinispan-1]' status: "True" type: wellFormed
```

**TIP**

Do the following for automated scripts:

```bash
$ oc wait --for condition=wellFormed --timeout=240s infinispan/example-infinispan
```

Alternatively, you can retrieve cluster view from logs as follows:
2.4. STOPPING AND STARTING DATA GRID CLUSTERS

Stop and start Data Grid nodes in a graceful, ordered fashion to correctly preserve cluster state.

Clusters of Data Grid service nodes must restart with the same number of nodes that existed before shutdown. This allows Data Grid to restore the distribution of data across the cluster. After Data Grid Operator fully restarts the cluster you can safely add and remove nodes.

Procedure

1. Change the `spec.replicas` field to 0 to stop the Data Grid cluster.

   ```yaml
   spec:
     replicas: 0
   ```

2. Ensure you have the correct number of nodes before you restart the cluster.

   ```bash
   $ oc get infinispan example-infinispan -o=jsonpath='{.status.replicasWantedAtRestart}'
   ```

3. Change the `spec.replicas` field to the same number of nodes to restart the Data Grid cluster.

   ```yaml
   spec:
     replicas: 6
   ```
CHAPTER 3. SETTING UP DATA GRID SERVICES

Use Data Grid Operator to create clusters of either Cache service or Data Grid service nodes.

**IMPORTANT**

If you do not specify a value for the `spec.service.type` field, Data Grid Operator creates Cache service nodes by default.

You cannot change the `spec.service.type` field after you create nodes. To change the service type, you must delete the existing nodes and create new ones.

### 3.1. SERVICE TYPES

Services are stateful applications, based on the Data Grid Server image, that provide flexible and robust in-memory data storage.

#### 3.1.1. Data Grid service

Use Data Grid service if you want to:

- Back up data across global clusters with cross-site replication.
- Create caches with any valid configuration.
- Add file-based cache stores to save data in a persistent volume.
- Query values across caches using the Data Grid Query API.
- Use advanced Data Grid features and capabilities.

#### 3.1.2. Cache service

Use Cache service if you want a volatile, low-latency data store with minimal configuration. Cache service nodes:

- Automatically scale to meet capacity when data storage demands go up or down.
- Synchronously distribute data to ensure consistency.
- Replicates each entry in the cache across the cluster.
- Store cache entries off-heap and use eviction for JVM efficiency.
- Ensure data consistency with a default partition handling configuration.

**IMPORTANT**

Because Cache service nodes are volatile you lose all data when you apply changes to the cluster with the Infinispan CR or update the Data Grid version.

### 3.2. CREATING DATA GRID SERVICE NODES
To use custom cache definitions along with Data Grid capabilities such as cross-site replication, create clusters of Data Grid service nodes.

**Procedure**

1. Create an **Infinispan** CR that sets `spec.service.type: DataGrid` and configures any other Data Grid service resources.

   ```yaml
   apiVersion: infinispan.org/v1
   kind: Infinispan
   metadata:
     name: example-infinispan
   spec:
     replicas: 2
     service:
       type: DataGrid
   ```

2. Apply your **Infinispan** CR to create the cluster.

**3.2.1. Data Grid service CR**

This topic describes the **Infinispan** CR for Data Grid service nodes.

```yaml
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
  annotations:
    infinispan.org/monitoring: 'true'
spec:
  replicas: 6
  service:
    type: DataGrid
  container:
    storage: 2Gi
    ephemeralStorage: false
    storageClassName: my-storage-class
  sites:
    local:
      name: azure
      expose:
        type: LoadBalancer
        locations:
        - name: azure
          url: openshift://api.azure.host:6443
          secretName: azure-token
        - name: aws
          clusterName: example-infinispan
          namespace: rhdg-namespace
          url: openshift://api.aws.host:6443
          secretName: aws-token
  security:
    endpointSecretName: endpoint-identities
    endpointEncryption:
      type: Secret
```
Field | Description
---|---
metadata.name | Names your Data Grid cluster.
metadata.annotations.infinispan.org/monitoring | Automatically creates a ServiceMonitor for your cluster.
spec.replicas | Specifies the number of nodes in your cluster.
spec.service.type | Configures the type Data Grid service. A value of DataGrid creates a cluster with Data Grid service nodes.
spec.service.container | Configures the storage resources for Data Grid service nodes.
spec.service.sites | Configures cross-site replication.
spec.security.endpointSecretName | Specifies an authentication secret that contains Data Grid user credentials.
spec.security.endpointEncryption | Specifies TLS certificates and keystores to encrypt client connections.
spec.container | Specifies JVM, CPU, and memory resources for Data Grid nodes.
<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec.logging</td>
<td>Configures Data Grid logging categories.</td>
</tr>
<tr>
<td>spec.expose</td>
<td>Controls how Data Grid endpoints are exposed on the network.</td>
</tr>
<tr>
<td>spec.affinity</td>
<td>Configures anti-affinity strategies that guarantee Data Grid availability.</td>
</tr>
</tbody>
</table>

### 3.3. CREATING CACHE SERVICE NODES

Create Data Grid clusters with Cache service nodes for a volatile, low-latency data store with minimal configuration.

#### Procedure

1. Create an **Infinispan** CR that sets `spec.service.type: Cache` and configures any other Cache service resources.

   ```yaml
   apiVersion: infinispan.org/v1
   kind: Infinispan
   metadata: 
     name: example-infinispan
   spec: 
     replicas: 2
     service: 
       type: Cache
   ```

2. Apply your **Infinispan** CR to create the cluster.

#### 3.3.1. Configuring automatic scaling

If you create clusters with Cache service nodes, Data Grid Operator can automatically scale nodes up or down based on memory usage for the default cache.

**IMPORTANT**

Automatic scaling works with the default cache only. If you plan to add other caches to your cluster, you should not include the `autoscale` field in your **Infinispan** CR. In this case you should use eviction to control the size of the data container on each node.

To configure automatic scaling you set:

- A maximum threshold for memory usage on each Data Grid node in your cluster. When Data Grid Operator detects that any node in the cluster reaches the threshold, it creates a new node if possible. If Data Grid Operator cannot create a new node then it performs eviction when memory usage reaches 100 percent.

- A minimum threshold for memory usage across your Data Grid cluster.
When Data Grid Operator detects that memory usage falls below the minimum, it shuts down nodes.

Procedure

1. Add the `spec.autoscale` resource to your Infinispan CR to enable automatic scaling.

   **NOTE**
   Set a value of `true` for the `autoscale.disabled` field to disable automatic automatic scaling.

2. Specify a maximum threshold, as a percentage, for memory usage on each node with the `spec.autoscale.maxMemUsagePercent` field.

3. Set the maximum number of number of nodes for the cluster with the `spec.autoscale.maxReplicas` field.

4. Specify a minimum threshold, as a percentage, for cluster memory usage with the `spec.autoscale.minMemUsagePercent` field.

5. Set the minimum number of number of nodes for the cluster with the `spec.autoscale.minReplicas` field.

   ```yaml
   spec:
     service:
       type: Cache
       autoscale:
         disabled: false
         maxMemUsagePercent: 70
         maxReplicas: 5
         minMemUsagePercent: 30
         minReplicas: 2
   ```

6. Apply the changes.

### 3.3.2. Configuring the number of owners

The number of owners controls how many copies of each cache entry are replicated across your Data Grid cluster. The default for Cache service nodes is two, which duplicates each entry to prevent data loss.

Procedure

1. Specify the number of copies for each cache entry with the `spec.service.replicationFactor` field in your Infinispan CR.

   ```yaml
   spec:
     service:
       type: Cache
       replicationFactor: 3
   ```

2. Apply the changes.

### 3.3.3. Cache service CR
This topic describes the **Infinispan** CR for Cache service nodes.

```yaml
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
  annotations:
    infinispan.org/monitoring: 'true'
spec:
  replicas: 4
  service:
    type: Cache
    replicationFactor: 2
  autoscale:
    maxMemUsagePercent: 70
    maxReplicas: 5
    minMemUsagePercent: 30
    minReplicas: 2
  security:
    endpointSecretName: endpoint-identities
    endpointEncryption:
      type: Secret
      certSecretName: tls-secret
  container:
    extraJvmOpts: "-XX:NativeMemoryTracking=summary"
    cpu: "2000m"
    memory: 1Gi
  logging:
    categories:
      org.infinispan: trace
      org.jgroups: trace
  expose:
    type: LoadBalancer
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchLabels:
                app: infinispan-pod
                clusterName: example-infinispan
                infinispan_cr: example-infinispan
                topologyKey: "kubernetes.io/hostname"
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>metadata.name</code></td>
<td>Names your Data Grid cluster.</td>
</tr>
<tr>
<td><code>metadata.annotations.infinispan.org/monitoring</code></td>
<td>Automatically creates a <strong>ServiceMonitor</strong> for your cluster.</td>
</tr>
<tr>
<td><code>spec.replicas</code></td>
<td>Specifies the number of nodes in your cluster.</td>
</tr>
</tbody>
</table>
### 3.4. ALLOCATING STORAGE RESOURCES

You can allocate storage for Data Grid service nodes but not Cache service nodes.

By default, Data Grid Operator allocates $1Gi$ for the persistent volume claim. However you should adjust the amount of storage available to Data Grid service nodes so that Data Grid can preserve cluster state during shutdown.

**IMPORTANT**

If available container storage is less than the amount of available memory, data loss can occur.

**Procedure**

1. Allocate storage resources with the `spec.service.container.storage` field.
2. Optionally configure the `ephemeralStorage` and `storageClassName` fields as required.

```yaml
spec:
  service:
    type: DataGrid
  container:
```
3. Apply the changes.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec.service.container.storage</td>
<td>Specifies the amount of storage for Data Grid service nodes.</td>
</tr>
<tr>
<td>spec.service.container.ephemeralStorage</td>
<td>Defines whether storage is ephemeral or permanent. Set the value to <code>true</code> to use ephemeral storage, which means all data in storage is deleted when clusters shut down or restart. The default value is <code>false</code>, which means storage is permanent.</td>
</tr>
<tr>
<td>spec.service.container.storageClassName</td>
<td>Specifies the name of a StorageClass object to use for the persistent volume claim (PVC). If you include this field, you must specify an existing storage class as the value. If you do not include this field, the persistent volume claim uses the storage class that has the <code>storageclass.kubernetes.io/is-default-class</code> annotation set to <code>true</code>.</td>
</tr>
</tbody>
</table>

### 3.4.1. Persistent volume claims

Data Grid Operator creates a persistent volume claim (PVC) and mounts container storage at: `/opt/infinispan/server/data`

**Caches**

When you create caches, Data Grid permanently stores their configuration so your caches are available after cluster restarts. This applies to both Cache service and Data Grid service nodes.

**Data**

Data is always volatile in clusters of Cache service nodes. When you shutdown the cluster, you permanently lose the data.

Use a file-based cache store, by adding the `<file-store/>` element to your Data Grid cache configuration, if you want Data Grid service nodes to persist data during cluster shutdown.

### 3.5. JVM, CPU, AND MEMORY

You can set JVM options in `Infinispan` CR as well as CPU and memory allocation.

```yaml
spec:
  container:
    extraJvmOpts: "-XX:NativeMemoryTracking=summary"
    cpu: "1000m"
    memory: 1Gi
```
### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>spec.container.extraJvmOpts</td>
<td>Specifies JVM options.</td>
</tr>
<tr>
<td>spec.container.cpu</td>
<td>Allocates host CPU resources to Data Grid nodes, measured in CPU units.</td>
</tr>
<tr>
<td>spec.container.memory</td>
<td>Allocates host memory to Data Grid nodes, measured in bytes.</td>
</tr>
</tbody>
</table>

When Data Grid Operator creates Data Grid clusters, it uses `spec.container.cpu` and `spec.container.memory` to:

- Ensure that OpenShift has sufficient capacity to run the Data Grid node. By default Data Grid Operator requests 512Mi of `memory` and 0.5 `cpu` from the OpenShift scheduler.
- Constrain node resource usage. Data Grid Operator sets the values of `cpu` and `memory` as resource limits.

#### 3.6. Adjusting Log Levels

Change levels for different Data Grid logging categories when you need to debug issues. You can also adjust log levels to reduce the number of messages for certain categories to minimize the use of container resources.

**Procedure**

1. Configure Data Grid logging with the `spec.logging.categories` field in your Infinispan CR.

   ```yaml
   spec:
     logging:
       categories:
         org.infinispan: debug
         org.jgroups: debug
   ```

2. Apply the changes.

3. Retrieve logs from Data Grid nodes as required.

   ```bash
   $ oc logs -f $POD_NAME
   ```

#### 3.6.1. Logging Reference

Find information about log categories and levels.

**Table 3.1. Log Categories**

<table>
<thead>
<tr>
<th>Root category</th>
<th>Description</th>
<th>Default level</th>
</tr>
</thead>
<tbody>
<tr>
<td>org.infinispan</td>
<td>Data Grid messages</td>
<td>info</td>
</tr>
</tbody>
</table>
## Table 3.2. Log levels

<table>
<thead>
<tr>
<th>Log level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>trace</strong></td>
<td>Provides detailed information about running state of applications. This is the most verbose log level.</td>
</tr>
<tr>
<td><strong>debug</strong></td>
<td>Indicates the progress of individual requests or activities.</td>
</tr>
<tr>
<td><strong>info</strong></td>
<td>Indicates overall progress of applications, including lifecycle events.</td>
</tr>
<tr>
<td><strong>warn</strong></td>
<td>Indicates circumstances that can lead to error or degrade performance.</td>
</tr>
<tr>
<td><strong>error</strong></td>
<td>Indicates error conditions that might prevent operations or activities from being successful but do not prevent applications from running.</td>
</tr>
</tbody>
</table>

### Garbage collection (GC) messages

Data Grid Operator does not log GC messages by default. You can direct GC messages to **stdout** with the following JVM options:

```
extraJvmOpts: "-Xlog:gc*:stdout:time,level,tags"
```

## 3.7. ADDING LABELS TO DATA GRID RESOURCES

Attach key/value labels to pods and services that Data Grid Operator creates and manages. These labels help you identify relationships between objects to better organize and monitor Data Grid resources.

### NOTE

Red Hat subscription labels are automatically applied to Data Grid pods.

### Procedure

1. Open your **Infinispan** CR for editing.
2. Add any labels that you want Data Grid Operator to attach to resources with `metadata.annotations`.
3. Add values for your labels with `metadata.labels`.  

---

24
a. Specify labels that you want to attach to services with the
   `metadata.annotations.infinispan.org/targetLabels` field.

b. Specify labels that you want to attach to pods with the
   `metadata.annotations.infinispan.org/podTargetLabels` field.

c. Define values for your labels with the `metadata.labels` fields.

```yaml
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  annotations:
    infinispan.org/targetLabels: svc-label1, svc-label2
    infinispan.org/podTargetLabels: pod-label1, pod-label2
  labels:
    svc-label1: svc-value1
    svc-label2: svc-value2
    pod-label1: pod-value1
    pod-label2: pod-value2
# The operator does not attach these labels to resources.
my-label: my-value
environment: development
```

4. Apply your Infinispan CR.

Additional resources

- [Labels and Selectors](#)

- [Labels: Kubernetes User Guide](#)
CHAPTER 4. CONFIGURING AUTHENTICATION

Application users need credentials to access Data Grid clusters. You can use default, generated credentials or add your own.

4.1. DEFAULT CREDENTIALS

Data Grid Operator generates base64-encoded default credentials stored in an authentication secrets named

<table>
<thead>
<tr>
<th>Username</th>
<th>Secret name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>developer</td>
<td>example-infinispan-generated-secret</td>
<td>Credentials for the default application user.</td>
</tr>
<tr>
<td>operator</td>
<td>example-infinispan-generated-operator-secret</td>
<td>Credentials that Data Grid Operator uses to interact with Data Grid resources.</td>
</tr>
</tbody>
</table>

4.2. RETRIEVING CREDENTIALS

Get credentials from authentication secrets to access Data Grid clusters.

Procedure

- Retrieve credentials from authentication secrets.
  
  ```
  $ oc get secret example-infinispan-generated-secret
  ```

  Base64-decode credentials.
  
  ```
  $ oc get secret example-infinispan-generated-secret \
  -o jsonpath="{.data.identities.yaml}" | base64 --decode
  ```

  credentials:
  - username: developer
    password: dIRs5cAAsHIeeRIL

4.3. ADDING CUSTOM USER CREDENTIALS

Configure access to Data Grid cluster endpoints with custom credentials.

NOTE

Modifying `spec.security.endpointSecretName` triggers a cluster restart.

Procedure

1. Create an `identities.yaml` file with the credentials that you want to add.
credentials:
- username: myfirstusername
  password: changeme-one
- username: mysecondusername
  password: changeme-two

2. Create an authentication secret from `identities.yaml`.

   $ oc create secret generic --from-file=identities.yaml connect-secret

3. Specify the authentication secret with `spec.security.endpointSecretName` in your Infinispan CR and then apply the changes.

   ```yaml
   spec:
     security:
       endpointSecretName: connect-secret
   ```

4.4. CHANGING THE OPERATOR PASSWORD

You can change the password for the operator user if you do not want to use the automatically generated password.

**Procedure**

- Update the `password` key in the `example-infinispan-generated-operator-secret` secret as follows:

   ```bash
   oc patch secret example-infinispan-generated-operator-secret -p=\"{"stringData":{"password":"supersecretoperatorpassword"}}\"
   ```

**NOTE**

You should update only the `password` key in the `generated-operator-secret` secret. When you update the password, Data Grid Operator automatically refreshes other keys in that secret.

4.5. DISABLING USER AUTHENTICATION

Allow users to access Data Grid clusters and manipulate data without providing credentials.

**IMPORTANT**

Do not disable authentication if endpoints are accessible from outside the OpenShift cluster via `spec.expose.type`. You should disable authentication for development environments only.

**Procedure**

1. Set `false` as the value for the `spec.security.endpointAuthentication` field in your Infinispan CR.
spec:
  security:
    endpointAuthentication: false

2. Apply the changes.
CHAPTER 5. CONFIGURING CLIENT CERTIFICATE AUTHENTICATION

Add client trust stores to your project and configure Data Grid to allow connections only from clients that present valid certificates. This increases security of your deployment by ensuring that clients are trusted by a public certificate authority (CA).

5.1. CLIENT CERTIFICATE AUTHENTICATION

Client certificate authentication restricts in-bound connections based on the certificates that clients present.

You can configure Data Grid to use trust stores with either of the following strategies:

Validate

To validate client certificates, Data Grid requires a trust store that contains any part of the certificate chain for the signing authority, typically the root CA certificate. Any client that presents a certificate signed by the CA can connect to Data Grid.

If you use the Validate strategy for verifying client certificates, you must also configure clients to provide valid Data Grid credentials if you enable authentication.

Authenticate

Requires a trust store that contains all public client certificates in addition to the root CA certificate. Only clients that present a signed certificate can connect to Data Grid.

If you use the Authenticate strategy for verifying client certificates, you must ensure that certificates contain valid Data Grid credentials as part of the distinguished name (DN).

5.2. ENABLING CLIENT CERTIFICATE AUTHENTICATION

To enable client certificate authentication, you configure Data Grid to use trust stores with either the Validate or Authenticate strategy.

Procedure

1. Set either Validate or Authenticate as the value for the spec.security.endpointEncryption.clientCert field in your Infinispan CR.

   NOTE
   
   The default value is None.

2. Specify the secret that contains the client trust store with the spec.security.endpointEncryption.clientCertSecretName field.
   By default Data Grid Operator expects a trust store secret named <cluster-name>-client-cert-secret.
The secret must be unique to each Infinispan CR instance in the OpenShift cluster. When you delete the Infinispan CR, OpenShift also automatically deletes the associated secret.

```
spec:
  security:
    endpointEncryption:
      type: Service
      certSecretName: tls-secret
      clientCert: Validate
      clientCertSecretName: example-infinispan-client-cert-secret
```

3. Apply the changes.

Next steps
Provide Data Grid Operator with a trust store that contains all client certificates. Alternatively you can provide certificates in PEM format and let Data Grid generate a client trust store.

5.3. PROVIDING CLIENT TRUSTSTORES
If you have a trust store that contains the required certificates you can make it available to Data Grid Operator.

Data Grid supports trust stores in PKCS12 format only.

Procedure

1. Specify the name of the secret that contains the client trust store as the value of the `metadata.name` field.

```
apiVersion: v1
kind: Secret
metadata:
  name: example-infinispan-client-cert-secret
  type: Opaque
stringData:
  truststore-password: changme
data:
  truststore.p12: "<base64_encoded_PKCS12_trust_store>"
```

2. Provide the password for the trust store with the `stringData.truststore-password` field.

3. Specify the trust store with the `data.truststore.p12` field.

4. Apply the changes.
5.4. PROVIDING CLIENT CERTIFICATES

Data Grid Operator can generate a trust store from certificates in PEM format.

Procedure

1. Specify the name of the secret that contains the client trust store as the value of the `metadata.name` field.

   **NOTE**
   
   The name must match the value of the `spec.security.endpointEncryption.clientCertSecretName` field.

2. Specify the signing certificate, or CA certificate bundle, as the value of the `data.trust.ca` field.

3. If you use the **Authenticate** strategy to verify client identities, add the certificate for each client that can connect to Data Grid endpoints with the `data.trust.cert.<name>` field.

   **NOTE**
   
   Data Grid Operator uses the `<name>` value as the alias for the certificate when it generates the trust store.

4. Optionally provide a password for the trust store with the `stringData.truststore-password` field.

   If you do not provide one, Data Grid Operator sets "password" as the trust store password.

   ```yaml
   apiVersion: v1
   kind: Secret
   metadata:
     name: example-infinispan-client-cert-secret
   type: Opaque
   stringData:
     truststore-password: changme
   data:
     trust.ca: "<base64_encoded_CA_certificate>"
     trust.cert.client1: "<base64_encoded_client_certificate>"
     trust.cert.client2: "<base64_encoded_client_certificate>"
   ``

5. Apply the changes.
CHAPTER 6. CONFIGURING ENCRYPTION

Encrypt connections between clients and Data Grid nodes with Red Hat OpenShift service certificates or custom TLS certificates.

6.1. ENCRYPTION WITH RED HAT OPENShift SERVICE CERTIFICATES

Data Grid Operator automatically generates TLS certificates that are signed by the Red Hat OpenShift service CA. Data Grid Operator then stores the certificates and keys in a secret so you can retrieve them and use with remote clients.

If the Red Hat OpenShift service CA is available, Data Grid Operator adds the following `spec.security.endpointEncryption` configuration to the Infinispan CR:

```yaml
spec:
  security:
    endpointEncryption:
      type: Service
      certServiceName: service.beta.openshift.io
      certSecretName: example-infinispan-cert-secret
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>spec.security.endpointEncryption.certServiceName</code></td>
<td>Specifies the service that provides TLS certificates.</td>
</tr>
<tr>
<td><code>spec.security.endpointEncryption.certSecretName</code></td>
<td>Specifies a secret with a service certificate and key in PEM format. Defaults to <code>&lt;cluster_name&gt;-cert-secret</code>.</td>
</tr>
</tbody>
</table>

**NOTE**

Service certificates use the internal DNS name of the Data Grid cluster as the common name (CN), for example:

**Subject**: CN = example-infinispan.mynamespace.svc

For this reason, service certificates can be fully trusted only inside OpenShift. If you want to encrypt connections with clients running outside OpenShift, you should use custom TLS certificates.

Service certificates are valid for one year and are automatically replaced before they expire.

6.2. RETRIEVING TLS CERTIFICATES

Get TLS certificates from encryption secrets to create client trust stores.

**Procedure**

- Retrieve `tls.crt` from encryption secrets as follows:
6.3. DISABLING ENCRYPTION

You can disable encryption so clients do not need TLS certificates to establish connections with Data Grid.

**IMPORTANT**

Do not disable encryption if endpoints are accessible from outside the OpenShift cluster via `spec.expose.type`. You should disable encryption for development environments only.

Procedure

1. Set `None` as the value for the `spec.security.endpointEncryption.type` field in your Infinispan CR.

   ```yaml
   spec:
     security:
       endpointEncryption:
         type: None
   ```

2. Apply the changes.

6.4. USING CUSTOM TLS CERTIFICATES

Use custom PKCS12 keystore or TLS certificate/key pairs to encrypt connections between clients and Data Grid clusters.

Prerequisites

- Create either a keystore or certificate secret.

**NOTE**

The secret must be unique to each Infinispan CR instance in the OpenShift cluster. When you delete the Infinispan CR, OpenShift also automatically deletes the associated secret.

Procedure

1. Add the encryption secret to your OpenShift namespace, for example:

   ```bash
   $ oc apply -f tls_secret.yaml
   ```

2. Specify the encryption secret with the `spec.security.endpointEncryption.certSecretName` field in your Infinispan CR.

   ```yaml
   spec:
     security:
   ```
endpointEncryption:
  type: Secret
certSecretName: tls-secret

3. Apply the changes.

### 6.4.1. Custom encryption secrets

This topic describes resources for custom encryption secrets.

#### Keystore secrets

```yaml
apiVersion: v1
class: Secret
metadata:
  name: tls-secret
type: Opaque
stringData:
  alias: server
  password: changeme
data:
  keystore.p12: "MIICgjIBAzCCCGCSqGCIb3DQEHA..."
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringData.alias</td>
<td>Specifies an alias for the keystore.</td>
</tr>
<tr>
<td>stringData.password</td>
<td>Specifies the keystore password.</td>
</tr>
<tr>
<td>data.keystore.p12</td>
<td>Adds a base64-encoded keystore.</td>
</tr>
</tbody>
</table>

#### Certificate secrets

```yaml
apiVersion: v1
class: Secret
metadata:
  name: tls-secret
type: Opaque
data:
  tls.key: "LS0tLS1CRUdJTiBQQUk ...
  tls.crt: "LS0tLS1CRUdJTiBDRV...
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data.tls.key</td>
<td>Adds a base64-encoded TLS key.</td>
</tr>
<tr>
<td>data.tls.crt</td>
<td>Adds a base64-encoded TLS certificate.</td>
</tr>
</tbody>
</table>
CHAPTER 7. CONFIGURING USER ROLES AND PERMISSIONS

Secure access to Data Grid services by configuring role-based access control (RBAC) for users. This requires you to assign roles to users so that they have permission to access caches and Data Grid resources.

7.1. ENABLING SECURITY AUTHORIZATION

By default authorization is disabled to ensure backwards compatibility with Infinispan CR instances. Complete the following procedure to enable authorization and use role-based access control (RBAC) for Data Grid users.

Procedure

1. Set `true` as the value for the `spec.security.authorization.enabled` field in your Infinispan CR.

   ```yaml
   spec:
     security:
       authorization:
         enabled: true
   ```

2. Apply the changes.

7.2. USER ROLES AND PERMISSIONS

Data Grid Operator provides a set of default roles that are associated with different permissions.

Table 7.1. Default roles and permissions

<table>
<thead>
<tr>
<th>Role</th>
<th>Permissions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>ALL</td>
<td>Superuser with all permissions including control of the Cache Manager lifecycle.</td>
</tr>
<tr>
<td>deployer</td>
<td>ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR, CREATE</td>
<td>Can create and delete Data Grid resources in addition to application permissions.</td>
</tr>
<tr>
<td>application</td>
<td>ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR</td>
<td>Has read and write access to Data Grid resources in addition to observer permissions. Can also listen to events and execute server tasks and scripts.</td>
</tr>
<tr>
<td>observer</td>
<td>ALL_READ, MONITOR</td>
<td>Has read access to Data Grid resources in addition to monitor permissions.</td>
</tr>
<tr>
<td>monitor</td>
<td>MONITOR</td>
<td>Can view statistics for Data Grid clusters.</td>
</tr>
</tbody>
</table>
Data Grid Operator credentials
Data Grid Operator generates credentials that it uses to authenticate with Data Grid clusters to perform internal operations. By default Data Grid Operator credentials are automatically assigned the **admin** role when you enable security authorization.

Additional resources

- How security authorization works ([Data Grid Security Guide](#)).

**7.3. ASSIGNING ROLES AND PERMISSIONS TO USERS**

Assign users with roles that control whether users are authorized to access Data Grid cluster resources. Roles can have different permission levels, from read-only to unrestricted access.

**NOTE**

Users gain authorization implicitly. For example, "admin" gets **admin** permissions automatically. A user named "deployer" has the **deployer** role automatically, and so on.

**Procedure**

1. Create an `identities.yaml` file that assigns roles to users.

   ```yaml
   credentials:
   - username: admin
     password: changeme
   - username: my-user-1
     password: changeme
   roles:
     - admin
   - username: my-user-2
     password: changeme
   roles:
     - monitor
   
   ```

2. Create an authentication secret from `identities.yaml`. If necessary, delete the existing secret first.

   ```bash
   $ oc delete secret connect-secret --ignore-not-found
   $ oc create secret generic --from-file=identities.yaml connect-secret
   
   ```

3. Specify the authentication secret with `spec.security.endpointSecretName` in your **Infinispan** CR and then apply the changes.

   ```yaml
   spec:
     security:
       endpointSecretName: connect-secret
   
   ```

**7.4. ADDING CUSTOM ROLES AND PERMISSIONS**

You can define custom roles with different combinations of permissions.
Procedure

1. Open your **Infinispan** CR for editing.

2. Specify custom roles and their associated permissions with the `spec.security.authorization.roles` field.

```yaml
spec:
  security:
    authorization:
      enabled: true
    roles:
      - name: my-role-1
        permissions:
        - ALL
      - name: my-role-2
        permissions:
        - READ
        - WRITE
```

3. Apply the changes.
CHAPTER 8. CONFIGURING NETWORK ACCESS TO DATA GRID

Exposé Data Grid clusters so you can access Data Grid Console, the Data Grid command line interface (CLI), REST API, and Hot Rod endpoint.

8.1. GETTING THE SERVICE FOR INTERNAL CONNECTIONS

By default, Data Grid Operator creates a service that provides access to Data Grid clusters from clients running on OpenShift.

This internal service has the same name as your Data Grid cluster, for example:

```
metadata:
  name: example-infinispan
```

Procedure

- Check that the internal service is available as follows:

  ```
  $ oc get services
  NAME               TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
  example-infinispan ClusterIP   192.0.2.0        <none>        11222/TCP
  ```

8.2. EXPOSING DATA GRID THROUGH LOAD BALANCERS

Use a load balancer service to make Data Grid clusters available to clients running outside OpenShift.

**NOTE**

To access Data Grid with unencrypted Hot Rod client connections you must use a load balancer service.

Procedure

1. Include `spec.expose` in your `Infinispan` CR.
2. Specify `LoadBalancer` as the service type with the `spec.expose.type` field.
3. Optionally specify a node port to which the load balancer forwards traffic with the `spec.expose.nodePort` field.

```
spec:
  expose:
    type: LoadBalancer
    nodePort: 30000
```
4. Apply the changes.
5. Verify that the `-external` service is available.
$ oc get services | grep external

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-infinispan-external</td>
<td>LoadBalancer</td>
<td>192.0.2.24</td>
<td>hostname.com</td>
<td>11222/TCP</td>
</tr>
</tbody>
</table>

8.3. EXPOSING DATA GRID THROUGH NODE PORTS

Use a node port service to expose Data Grid clusters on the network.

**Procedure**

1. Include `spec.expose` in your Infinispan CR.

2. Specify `NodePort` as the service type with the `spec.expose.type` field.

3. Define the port where Data Grid is exposed with the `spec.expose.nodePort` field. If you do not set a value for the `spec.expose.nodePort` field, the platform selects an available port.

   ```yaml
   spec:
     expose:
       type: NodePort
       nodePort: 30000
   ```

4. Apply the changes.

5. Verify that the `-external` service is available.

$ oc get services | grep external

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-infinispan-external</td>
<td>NodePort</td>
<td>192.0.2.24</td>
<td>&lt;none&gt;</td>
<td>11222:30000/TCP</td>
</tr>
</tbody>
</table>

8.4. EXPOSING DATA GRID THROUGH ROUTES

Use an OpenShift Route with passthrough encryption to make Data Grid clusters available on the network.

**Procedure**

1. Include `spec.expose` in your Infinispan CR.

2. Specify `Route` as the service type with the `spec.expose.type` field.

3. Optionally add a hostname with the `spec.expose.host` field.

   ```yaml
   spec:
     expose:
       type: Route
       host: www.example.org
   ```

4. Apply the changes.
5. Verify that the route is available.

```
$ oc get routes
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>CLASS</th>
<th>HOSTS</th>
<th>ADDRESS</th>
<th>PORTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>example-infinispan</td>
<td>&lt;none&gt;</td>
<td>*</td>
<td>443</td>
<td>73s</td>
<td></td>
</tr>
</tbody>
</table>

**Route ports**

When you create a route, it exposes a port on the network that accepts client connections and redirects traffic to Data Grid services that listen on port **11222**.

The port where the route is available depends on whether you use encryption or not.

<table>
<thead>
<tr>
<th>Port</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>Encryption is disabled.</td>
</tr>
<tr>
<td>443</td>
<td>Encryption is enabled.</td>
</tr>
</tbody>
</table>

### 8.5. NETWORK SERVICES

Reference information for network services that Data Grid Operator creates and manages.

#### 8.5.1. Internal service

- Allow Data Grid nodes to discover each other and form clusters.
- Provide access to Data Grid endpoints from clients in the same OpenShift namespace.

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cluster_name&gt;</code></td>
<td>11222</td>
<td>TCP</td>
<td>Internal access to Data Grid endpoints</td>
</tr>
<tr>
<td><code>&lt;cluster_name&gt;-ping</code></td>
<td>8888</td>
<td>TCP</td>
<td>Cluster discovery</td>
</tr>
</tbody>
</table>

#### 8.5.2. External service

Provides access to Data Grid endpoints from clients outside OpenShift or in different namespaces.

**NOTE**

You must create the external service with Data Grid Operator. It is not available by default.
### 8.5.3. Cross-site service

Allows Data Grid to back up data between clusters in different locations.

<table>
<thead>
<tr>
<th>Service</th>
<th>Port</th>
<th>Protocol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;cluster_name&gt;-external</code></td>
<td>11222</td>
<td>TCP</td>
<td>External access to Data Grid endpoints.</td>
</tr>
<tr>
<td><code>&lt;cluster_name&gt;-site</code></td>
<td>7900</td>
<td>TCP</td>
<td>JGroups RELAY2 channel for cross-site communication.</td>
</tr>
</tbody>
</table>
CHAPTER 9. MONITORING DATA GRID SERVICES

Data Grid exposes metrics that can be used by Prometheus and Grafana for monitoring and visualizing the cluster state.

**NOTE**

This documentation explains how to set up monitoring on OpenShift Container Platform. If you’re working with community Prometheus deployments, you might find these instructions useful as a general guide. However you should refer to the Prometheus documentation for installation and usage instructions.

See the Prometheus Operator documentation.

9.1. CREATING A PROMETHEUS SERVICE MONITOR

Data Grid Operator automatically creates a Prometheus **ServiceMonitor** that scrapes metrics from your Data Grid cluster.

**Procedure**

Enable monitoring for user-defined projects on OpenShift Container Platform.

When the **ServiceMonitor** resource is detected, Data Grid Operator does the following:

- Creates a **ServiceMonitor** named `<cluster_name>-monitor`.
- Adds the `infinispan.org/monitoring: 'true'` annotation to your **Infinispan** CR:

```yaml
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
annotations:
  infinispan.org/monitoring: 'true'
```

**NOTE**

To authenticate with Data Grid, Prometheus uses the **operator** credentials.

**Verification**

You can check that Prometheus is scraping Data Grid metrics as follows:

1. In the OpenShift Web Console, select the `</> Developer` perspective and then select **Monitoring**.
2. Open the **Dashboard** tab for the namespace where your Data Grid cluster runs.
3. Open the **Metrics** tab and confirm that you can query Data Grid metrics such as:

```
vendor_cache_manager_default_cluster_size
```

**Additional resources**
Enabling monitoring for user-defined projects

9.1.1. Disabling the Prometheus service monitor

You can disable the ServiceMonitor if you do not want Prometheus to scrape metrics for your Data Grid cluster.

Procedure

1. Set 'false' as the value for the `infinispan.org/monitoring` annotation in your Infinispan CR.

   ```yaml
   apiVersion: infinispan.org/v1
   kind: Infinispan
   metadata:
     name: example-infinispan
   annotations:
     infinispan.org/monitoring: 'false'
   ```

2. Apply the changes.

9.2. INSTALLING THE GRAFANA OPERATOR

To support various needs, Data Grid Operator integrates with the community version of the Grafana Operator to create dashboards for Data Grid services.

Until Grafana is integrated with OpenShift user workload monitoring, the only option is to rely on the community version. You can install the Grafana Operator on OpenShift from the OperatorHub and should create a subscription for the alpha channel.

However, as is the policy for all Community Operators, Red Hat does not certify the Grafana Operator and does not provide support for it in combination with Data Grid. When you install the Grafana Operator you are prompted to acknowledge a warning about the community version before you can continue.

9.3. CREATING GRAFANA DATA SOURCES

Create a GrafanaDatasource CR so you can visualize Data Grid metrics in Grafana dashboards.

Prerequisites

- Have an oc client.

- Have `cluster-admin` access to OpenShift Container Platform.

- Enable monitoring for user-defined projects on OpenShift Container Platform.

- Install the Grafana Operator from the alpha channel and create a Grafana CR.

Procedure

1. Create a ServiceAccount that lets Grafana read Data Grid metrics from Prometheus.

   ```yaml
   apiVersion: v1
   kind: ServiceAccount
   ```
metadata:
  name: infinispan-monitoring

a. Apply the **ServiceAccount**.

$ oc apply -f service-account.yaml

b. Grant **cluster-monitoring-view** permissions to the **ServiceAccount**.

$ oc adm policy add-cluster-role-to-user cluster-monitoring-view -z infinispan-monitoring

2. Create a Grafana data source.

a. Retrieve the token for the **ServiceAccount**.

$ oc serviceaccounts get-token infinispan-monitoring

eyJhbGciOiJSUzI1NiIsImtpZCI6Imc4O...

b. Define a **GrafanaDataSource** that includes the token in the **spec.datasources.secureJsonData.httpHeaderValue1** field, as in the following example:

```yaml
apiVersion: integreatly.org/v1alpha1
kind: GrafanaDataSource
metadata:
  name: grafanadatasource
spec:
  name: datasource.yaml
datasources:
  - access: proxy
    editable: true
    isDefault: true
    jsonData:
      httpHeaderName1: Authorization
      timeInterval: 5s
      tlsSkipVerify: true
    name: Prometheus
    secureJsonData:
      httpHeaderValue1: |
        Bearer
        eyJhbGciOiJSUzI1NiIsImtpZCI6Imc4O...
      type: prometheus

3. Apply the **GrafanaDataSource**.

$ oc apply -f grafana-datasource.yaml

Next steps

Enable Grafana dashboards with the Data Grid Operator configuration properties.

### 9.4. CONFIGURING DATA GRID DASHBOARDS
Data Grid Operator provides global configuration properties that let you configure Grafana dashboards for Data Grid clusters.

**NOTE**

You can modify global configuration properties while Data Grid Operator is running.

**Prerequisites**

- Data Grid Operator must watch the namespace where the Grafana Operator is running.

**Procedure**

1. Create a **ConfigMap** named **infinispan-operator-config** in the Data Grid Operator namespace.

   ```yaml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: infinispan-operator-config
   data:
     grafana.dashboard.namespace: example-infinispan
     grafana.dashboard.name: infinispan
     grafana.dashboard.monitoring.key: middleware
   ```

2. Specify the namespace of your Data Grid cluster with the **data.grafana.dashboard.namespace** property.

   **NOTE**
   Deleting the value for this property removes the dashboard. Changing the value moves the dashboard to that namespace.

3. Specify a name for the dashboard with the **data.grafana.dashboard.name** property.

4. If necessary, specify a monitoring key with the **data.grafana.dashboard.monitoring.key** property.

5. Create **infinispan-operator-config** or update the configuration.

   ```bash
   $ oc apply -f infinispan-operator-config.yaml
   ```

6. Open the Grafana UI, which is available at:

   ```bash
   $ oc get routes grafana-route -o jsonpath=\"{.spec.host}\"
   ```
CHAPTER 10. SETTING UP CROSS-SITE REPLICATION

Ensure service availability with Data Grid Operator by configuring cross-site replication to back up data between Data Grid clusters.

10.1. USING DATA GRID OPERATOR TO MANAGE CROSS-SITE CONNECTIONS

Data Grid Operator in one data center can discover a Data Grid cluster that Data Grid Operator manages in another data center. This discovery allows Data Grid to automatically form cross-site views and create global clusters.

The following illustration provides an example in which Data Grid Operator manages a Data Grid cluster at a data center in New York City, NYC. At another data center in London, LON, Data Grid Operator also manages a Data Grid cluster.

Data Grid Operator uses the Kubernetes API to establish a secure connection between the OpenShift Container Platform clusters in NYC and LON. Data Grid Operator then creates a cross-site replication service so Data Grid clusters can back up data across locations.

IMPORTANT

Data Grid Operator in each OpenShift cluster must have network access to the remote Kubernetes API.

NOTE

When you configure automatic connections, Data Grid clusters do not start running until Data Grid Operator discovers all backup locations in the configuration.

Each Data Grid cluster has one site master node that coordinates all backup requests. Data Grid Operator identifies the site master node so that all traffic through the cross-site replication service goes to the site master.

If the current site master node goes offline then a new node becomes site master. Data Grid Operator automatically finds the new site master node and updates the cross-site replication service to forward backup requests to it.

10.1.1. Creating service account tokens
Generate service account tokens on each OpenShift cluster that acts as a backup location. Clusters use these tokens to authenticate with each other so Data Grid Operator can create a cross-site replication service.

**Procedure**

1. Log in to an OpenShift cluster.

2. Create a service account.
   For example, create a service account at **LON**:
   ```bash
   $ oc create sa lon
   serviceaccount/lon created
   ```

3. Add the view role to the service account with the following command:
   ```bash
   $ oc policy add-role-to-user view system:serviceaccount:<namespace>:lon
   ```

4. If you use a node port service to expose Data Grid clusters on the network, you must also add the **cluster-reader** role to the service account:
   ```bash
   $ oc adm policy add-cluster-role-to-user cluster-reader -z <service-account-name> -n <namespace>
   ```

5. Repeat the preceding steps on your other OpenShift clusters.

**Additional resources**

- Using service accounts in applications

### 10.1.2. Exchanging service account tokens

After you create service account tokens on your OpenShift clusters, you add them to secrets on each backup location. For example, at **LON** you add the service account token for **NYC**. At **NYC** you add the service account token for **LON**.

**Prerequisites**

- Get tokens from each service account.
  Use the following command or get the token from the OpenShift Web Console:
  ```bash
  $ oc sa get-token lon
  eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9...
  ```

**Procedure**

1. Log in to an OpenShift cluster.

2. Add the service account token for a backup location with the following command:
   ```bash
   $ oc create secret generic <token-name> --from-literal=token=<token>
   ```
For example, log in to the OpenShift cluster at NYC and create a lon-token secret as follows:

$ oc create secret generic lon-token --from-literal=token=eyJhbGciOiJSUzI1NiIsImtpZCI6IiJ9...

3. Repeat the preceding steps on your other OpenShift clusters.

10.1.3. Configuring Data Grid Operator to handle cross-site connections

Configure Data Grid Operator to establish cross-site views with Data Grid clusters.

Prerequisites

- Create secrets that contain service account tokens for each backup location.

Procedure

1. Create an Infinispan CR for each Data Grid cluster.

2. Specify the name of the local site with spec.service.sites.local.name.

3. Set the value of the service.sites.local.expose.type field to either NodePort or LoadBalancer. If you use a NodePort service for cross-site replication, you can configure a static port within the default range of 30000 to 32767 as follows:

   ```yaml
   spec:
     service:
       type: DataGrid
       sites:
         local:
           name: LON
           expose:
             type: NodePort
             nodePort: 30000
   ```

4. Provide the name, URL, and secret for each Data Grid cluster that acts as a backup location with spec.service.sites.locations.

5. If Data Grid cluster names or namespaces at the remote site do not match the local site, specify those values with the clusterName and namespace fields. The following are example Infinispan CR definitions for LON and NYC:

   - LON
     ```yaml
     apiVersion: infinispan.org/v1
     kind: Infinispan
     metadata:
       name: example-infinispan
     spec:
       replicas: 3
       service:
         type: DataGrid
     ```
Be sure to adjust logging categories in your Infinispan CR to decrease log levels for JGroups TCP and RELAY2 protocols. This prevents a large number of log files from using container storage.
6. Configure your **Infinispan** CRs with any other Data Grid service resources and then apply the changes.

7. Verify that Data Grid clusters form a cross-site view.
   a. Retrieve the **Infinispan** CR.
      
      ```
      $ oc get infinispan -o yaml
      ```
   b. Check for the **type: CrossSiteViewFormed** condition.

**Next steps**

If your clusters have formed a cross-site view, you can start adding backup locations to caches.

**10.1.4. Resources for managed cross-site connections**

This topic describes resources for cross-site connections that Data Grid Operator manages.

```yaml
spec:
  service:
    type: DataGrid
  sites:
    local:
      name: LON
      expose:
        type: LoadBalancer
    locations:
      - name: NYC
        clusterName: <nyc_cluster_name>
        namespace: <nyc_cluster_namespace>
        url: openshift://api.site-b.devcluster.openshift.com:6443
        secretName: nyc-token
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>service.type</strong>: DataGrid</td>
<td>Data Grid supports cross-site replication with Data Grid service clusters only.</td>
</tr>
<tr>
<td><strong>service.sites.local.name</strong></td>
<td>Names the local site where a Data Grid cluster runs.</td>
</tr>
<tr>
<td><strong>service.sites.local.expose.type</strong></td>
<td>Specifies the network service for cross-site replication. Data Grid clusters use this service to communicate and perform backup operations. You can set the value to <strong>NodePort</strong> or <strong>LoadBalancer</strong>.</td>
</tr>
<tr>
<td></td>
<td>If you use <strong>NodePort</strong> you can also configure the port with the <strong>service.site.local.expose.nodePort</strong> field.</td>
</tr>
<tr>
<td><strong>service.sites.locations</strong></td>
<td>Provides connection information for all backup locations.</td>
</tr>
</tbody>
</table>
### 10.2. MANUALLY CONNECTING DATA GRID CLUSTERS

You can specify static network connection details to perform cross-site replication with Data Grid clusters running outside OpenShift. Manual cross-site connections are necessary in any scenario where access to the Kubernetes API is not available outside the OpenShift cluster where Data Grid runs.

You can use both automatic and manual connections for Data Grid clusters in the same Infinispan CR. However, you must ensure that Data Grid clusters establish connections in the same way at each site.

#### Prerequisites

Manually connecting Data Grid clusters to form cross-site views requires predictable network locations for Data Grid services.

You need to know the network locations before they are created, which requires you to:

- Have the host names and ports for each Data Grid cluster that you plan to configure as a backup location.
- Have the host name of the `<cluster-name>-site` service for any remote Data Grid cluster that is running on OpenShift. You must use the `<cluster-name>-site` service to form a cross-site view between a cluster that Data Grid Operator manages and any other cluster.

#### Procedure

1. Create an Infinispan CR for each Data Grid cluster.

2. Specify the name of the local site with `spec.service.sites.local.name`.

3. Set the value of the `service.sites.local.expose.type` field to either NodePort or LoadBalancer.
   - If you use a NodePort service for cross-site replication, you can configure a static port within the default range of 30000 to 32767 as follows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>service.sites.locations.name</code></td>
<td>Specifies a backup location that matches <code>.spec.service.sites.local.name</code>.</td>
</tr>
<tr>
<td><code>service.sites.locations.url</code></td>
<td>Specifies the URL of the Kubernetes API for the backup location.</td>
</tr>
<tr>
<td><code>service.sites.locations.secretName</code></td>
<td>Specifies the secret that contains the service account token for the backup site.</td>
</tr>
<tr>
<td><code>service.sites.locations.clusterName</code></td>
<td>Specifies the cluster name at the backup location if it is different to the cluster name at the local site.</td>
</tr>
<tr>
<td><code>service.sites.locations.namespace</code></td>
<td>Specifies the namespace of the Data Grid cluster at the backup location if it does not match the namespace at the local site.</td>
</tr>
</tbody>
</table>
spec:
service:
  type: DataGrid
sites:
  local:
    name: LON
    expose:
      type: NodePort
      nodePort: 30000

4. Provide the name and static URL for each Data Grid cluster that acts as a backup location with `spec.service.sites.locations`, for example:

- **LON**

  ```yaml
  apiVersion: infinispan.org/v1
  kind: Infinispan
  metadata:
    name: example-infinispan
  spec:
    replicas: 3
    service:
      type: DataGrid
      sites:
        local:
          name: LON
          expose:
            type: LoadBalancer
          locations:
            - name: NYC
              url: infinispan+xsite://infinispan-nyc.myhost.com:7900
  logging:
    categories:
      org.jgroups.protocols.TCP: error
      org.jgroups.protocols.relay.RELAY2: error
  ```

- **NYC**

  ```yaml
  apiVersion: infinispan.org/v1
  kind: Infinispan
  metadata:
    name: example-infinispan
  spec:
    replicas: 2
    service:
      type: DataGrid
      sites:
        local:
          name: NYC
          expose:
            type: LoadBalancer
          locations:
            - name: LON
              url: infinispan+xsite://infinispan-lon.myhost.com
  logging:
  ```
5. Configure your Infinispan CRs with any other Data Grid service resources and then apply the changes.

6. Verify that Data Grid clusters form a cross-site view.
   a. Retrieve the Infinispan CR.
      ```
      $ oc get infinispan -o yaml
      ```
   b. Check for the type: CrossSiteViewFormed condition.

**Next steps**

If your clusters have formed a cross-site view, you can start adding backup locations to caches.

**10.2.1. Resources for manual cross-site connections**

This topic describes resources for cross-site connections that you maintain manually.

```
spec:
  service:
    type: DataGrid
  sites:
    local:
      name: LON
      expose:
        type: LoadBalancer
        locations:
        - name: NYC
          url: infinispan+xsite://infinispan-nyc.myhost.com:7900
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>service.type: DataGrid</td>
<td>Data Grid supports cross-site replication with Data Grid service clusters only.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>service.sites.local.name</code></td>
<td>Names the local site where a Data Grid cluster runs.</td>
</tr>
<tr>
<td><code>service.sites.local.expose.type</code></td>
<td>Specifies the network service for cross-site replication. Data Grid clusters use this service to communicate and perform backup operations. You can set the value to NodePort or LoadBalancer. If you use NodePort you can also configure the port with the <code>service.site.local.expose.nodePort</code> field.</td>
</tr>
<tr>
<td><code>service.sites.locations</code></td>
<td>Provides connection information for all backup locations.</td>
</tr>
<tr>
<td><code>service.sites.locations.name</code></td>
<td>Specifies a backup location that matches <code>spec.service.sites.local.name</code>.</td>
</tr>
<tr>
<td><code>service.sites.locations.url</code></td>
<td>Specifies the static URL for the backup location in the format of <code>infinispan+xsite://&lt;hostname&gt;:&lt;port&gt;</code>. The default port is 7900.</td>
</tr>
</tbody>
</table>

### 10.3. CONFIGURING SITES IN THE SAME OPENSHIFT CLUSTER

For evaluation and demonstration purposes, you can configure Data Grid to back up between nodes in the same OpenShift cluster.

**Procedure**

1. Create an Infinispan CR for each Data Grid cluster.
2. Specify the name of the local site with `spec.service.sites.local.name`.
3. Set ClusterIP as the value of the `spec.service.sites.local.expose.type` field.
4. Provide the name of the Data Grid cluster that acts as a backup location with `spec.service.sites.locations.clusterName`.
5. If both Data Grid clusters have the same name, specify the namespace of the backup location with `spec.service.sites.locations.namespace`.

```yaml
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-cluster
spec:
  replicas: 1
  expose:
    type: LoadBalancer
  service:
    type: DataGrid
```
6. Configure your **Infinispan** CRs with any other Data Grid service resources and then apply the changes.

7. Verify that Data Grid clusters form a cross-site view.
   
a. Retrieve the **Infinispan** CR.

   ```bash
   $ oc get infinispan -o yaml
   ```

b. Check for the **type: CrossSiteViewFormed** condition.
CHAPTER 11. GUARANTEEING AVAILABILITY WITH ANTI-AFFINITY

Kubernetes includes anti-affinity capabilities that protect workloads from single points of failure.

11.1. ANTI-AFFINITY STRATEGIES

Each Data Grid node in a cluster runs in a pod that runs on an OpenShift node in a cluster. Each Red Hat OpenShift node runs on a physical host system. Anti-affinity works by distributing Data Grid nodes across OpenShift nodes, ensuring that your Data Grid clusters remain available even if hardware failures occur.

Data Grid Operator offers two anti-affinity strategies:

- `kubernetes.io/hostname`
  Data Grid replica pods are scheduled on different OpenShift nodes.

- `topology.kubernetes.io/zone`
  Data Grid replica pods are scheduled across multiple zones.

Fault tolerance
Anti-affinity strategies guarantee cluster availability in different ways.

NOTE
The equations in the following section apply only if the number of OpenShift nodes or zones is greater than the number of Data Grid nodes.

Scheduling pods on different OpenShift nodes
Provides tolerance of $x$ node failures for the following types of cache:

- Replicated: $x = \text{spec.replicas} - 1$
- Distributed: $x = \text{num_owners} - 1$

Scheduling pods across multiple zones
Provides tolerance of $x$ zone failures when $x$ zones exist for the following types of cache:

- Replicated: $x = \text{spec.replicas} - 1$
- Distributed: $x = \text{num_owners} - 1$

NOTE

- `spec.replicas`
  Defines the number of pods in each Data Grid cluster.

- `num_owners`
  Is the cache configuration attribute that defines the number of replicas for each entry in the cache.

11.2. CONFIGURING ANTI-AFFINITY
Specify where OpenShift schedules pods for your Data Grid clusters to ensure availability.

**Procedure**

1. Add the `spec.affinity` block to your *Infinispan* CR.
2. Configure anti-affinity strategies as necessary.
3. Apply your *Infinispan* CR.

### 11.2.1. Anti-affinity strategy configurations

Configure anti-affinity strategies in your *Infinispan* CR to control where OpenShift schedules Data Grid replica pods.

<table>
<thead>
<tr>
<th>Topology keys</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>topologyKey: &quot;topology.kubernetes.io/zone&quot;</code></td>
<td>Schedules Data Grid replica pods across multiple zones.</td>
</tr>
<tr>
<td><code>topologyKey: &quot;kubernetes.io/hostname&quot;</code></td>
<td>Schedules Data Grid replica pods on different OpenShift nodes.</td>
</tr>
</tbody>
</table>

**Schedule pods on different OpenShift nodes**

The following is the anti-affinity strategy that Data Grid Operator uses if you do not configure the `spec.affinity` field in your *Infinispan* CR:

```yaml
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchLabels:
                app: infinispan-pod
                clusterName: <cluster_name>
                infinispan_cr: <cluster_name>
            topologyKey: "kubernetes.io/hostname"
```

**Requiring different nodes**

In the following example, OpenShift does not schedule Data Grid pods if different nodes are not available:

```yaml
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: infinispan-pod
```
NOTE

To ensure that you can schedule Data Grid replica pods on different OpenShift nodes, the number of OpenShift nodes available must be greater than the value of `spec.replicas`.

Schedule pods across multiple OpenShift zones
The following example prefers multiple zones when scheduling pods but schedules Data Grid replica pods on different OpenShift nodes if it is not possible to schedule across zones:

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
        - weight: 100
          podAffinityTerm:
            labelSelector:
              matchLabels:
                app: infinispan-pod
                clusterName: <cluster_name>
                infinispan_cr: <cluster_name>
            topologyKey: "topology.kubernetes.io/zone"
        - weight: 90
          podAffinityTerm:
            labelSelector:
              matchLabels:
                app: infinispan-pod
                clusterName: <cluster_name>
                infinispan_cr: <cluster_name>
            topologyKey: "kubernetes.io/hostname"
```

Requiring multiple zones
The following example uses the zone strategy only when scheduling Data Grid replica pods:

```
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution:
        - labelSelector:
            matchLabels:
              app: infinispan-pod
              clusterName: <cluster_name>
              infinispan_cr: <cluster_name>
          topologyKey: "topology.kubernetes.io/zone"
```
CHAPTER 12. CREATING CACHES WITH DATA GRID OPERATOR

Use Cache CRs to add cache configuration with Data Grid Operator and control how Data Grid stores your data.

IMPORTANT
Creating caches with Data Grid Operator is available as a technology preview.

12.1. TECHNOLOGY PREVIEWS

Technology Preview features or capabilities are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete.

Red Hat does not recommend using Technology Preview features or capabilities for production. These features provide early access to upcoming product features, which enables you to test functionality and provide feedback during the development process.

For more information, see Red Hat Technology Preview Features Support Scope.

12.2. DATA GRID CACHES

Cache configuration defines the characteristics and features of the data store and must be valid with the Data Grid schema. Data Grid recommends creating standalone files in XML or JSON format that define your cache configuration. You should separate Data Grid configuration from application code for easier validation and to avoid the situation where you need to maintain XML snippets in Java or some other client language.

To create caches with Data Grid clusters running on OpenShift, you should:

- Use Cache CR as the mechanism for creating caches through the OpenShift front end.
- Use Batch CR to create multiple caches at a time from standalone configuration files.
- Access Data Grid Console and create caches in XML or JSON format.

You can use Hot Rod or HTTP clients but Data Grid recommends Cache CR or Batch CR unless your specific use case requires programmatic remote cache creation.

12.3. CACHE CRS

Find out details for configuring Data Grid caches with Cache CR.

When using Cache CRs, the following rules apply:

- Cache CRs apply to Data Grid service nodes only.
- You can create a single cache for each Cache CR.
- If your Cache CR contains both a template and an XML configuration, Data Grid Operator uses the template.
- If you edit caches in the OpenShift Web Console, the changes are reflected through the user
interface but do not take effect on the Data Grid cluster. You cannot edit caches. To change cache configuration, you must first delete the cache through the console or CLI and then re-create the cache.

- Deleting Cache CRs in the OpenShift Web Console does not remove caches from Data Grid clusters. You must delete caches through the console or CLI.

**NOTE**

In previous versions, you need to add credentials to a secret so that Data Grid Operator can access your cluster when creating caches.

That is no longer necessary. Data Grid Operator uses the **operator** user and corresponding password to perform cache operations.

### 12.4. CREATING CACHES FROM XML

Complete the following steps to create caches on Data Grid service clusters using valid `infinispan.xml` configuration.

**Procedure**

1. Create a Cache CR that contains an XML cache configuration.
   a. Specify a name for the Cache CR with the **metadata.name** field.
   b. Specify the target Data Grid cluster with the **spec.clusterName** field.
   c. Name your cache with the **spec.name** field.

   **NOTE**

   The `name` attribute in the XML configuration is ignored. Only the **spec.name** field applies to the resulting cache.

   d. Add an XML cache configuration with the **spec.template** field.

   ```yaml
   apiVersion: infinispan.org/v2alpha1
   kind: Cache
   metadata:
     name: mycachedefinition
   spec:
     clusterName: example-infinispan
     name: mycache
     template: <distributed-cache name="mycache" mode="SYNC"><persistence><file-store/></persistence></distributed-cache>
   ```

2. Apply the Cache CR, for example:

   ```bash
   $ oc apply -f mycache.yaml
   cache.infinispan.org/mycachedefinition created
   ```

### 12.5. CREATING CACHES FROM TEMPLATES
Complete the following steps to create caches on Data Grid service clusters using cache templates.

**Prerequisites**

- Identify the cache template you want to use for your cache.
  You can find a list of available templates in Data Grid Console.

**Procedure**

1. Create a **Cache** CR that specifies the name of a template to use.
   a. Specify a name for the **Cache** CR with the `metadata.name` field.
   b. Specify the target Data Grid cluster with the `spec.clusterName` field.
   c. Name your cache with the `spec.name` field.
   d. Specify a cache template with the `spec.template` field.
      The following example creates a cache named "mycache" from the `org.infinispan.DIST_SYNC` cache template:

      ```yaml
      apiVersion: infinispan.org/v2alpha1
      kind: Cache
      metadata:
        name: mycachedefinition
      spec:
        clusterName: example-infinispan
        name: mycache
        templateName: org.infinispan.DIST_SYNC
      ```

2. Apply the **Cache** CR, for example:

   ```bash
   $ oc apply -f mycache.yaml
   cache.infinispan.org/mycachedefinition created
   ```

**12.6. ADDING Backup LOCATIONS TO CACHES**

When you configure Data Grid clusters to perform cross-site replication, you can add backup locations to your cache configurations.

**Procedure**

1. Create cache configurations that name remote sites as backup locations.
   Data Grid replicates data based on cache names. For this reason, site names in your cache configurations must match site names, `spec.service.sites.local.name`, in your Infinispan CRs.

2. Configure backup locations to go offline automatically with the **take-offline** element.
   a. Set the amount of time, in milliseconds, before backup locations go offline with the `min-wait` attribute.

3. Define any other valid cache configuration.

4. Add backup locations to the named cache on all sites in the global cluster.
For example, if you add LON as a backup for NYC you should add NYC as a backup for LON.

The following configuration examples show backup locations for caches:

- **NYC**

```xml
<distributed-cache name="customers">
  <encoding media-type="application/x-protostream"/>
  <backups>
    <backup site="LON" strategy="SYNC">
      <take-offline min-wait="120000"/>
    </backup>
  </backups>
</distributed-cache>
```

- **LON**

```xml
<replicated-cache name="customers">
  <encoding media-type="application/x-protostream"/>
  <backups>
    <backup site="NYC" strategy="ASYNC">
      <take-offline min-wait="120000"/>
    </backup>
  </backups>
</replicated-cache>
```

Additional resources

- Data Grid Guide to Cross-Site Replication

### 12.6.1. Performance considerations with taking backup locations offline

Backup locations can automatically go offline when remote sites become unavailable. This prevents nodes from attempting to replicate data to offline backup locations, which can have a performance impact on your cluster because it results in error.

You can configure how long to wait before backup locations go offline. A good rule of thumb is one or two minutes. However, you should test different wait periods and evaluate their performance impacts to determine the correct value for your deployment.

For instance when OpenShift terminates the site master pod, that backup location becomes unavailable for a short period of time until Data Grid Operator elects a new site master. In this case, if the minimum wait time is not long enough then the backup locations go offline. You then need to bring those backup locations online and perform state transfer operations to ensure the data is in sync.

Likewise, if the minimum wait time is too long, node CPU usage increases from failed backup attempts which can lead to performance degradation.

### 12.7. ADDING PERSISTENT CACHE STORES

You can add persistent cache stores to Data Grid service nodes to save data to the persistent volume. Data Grid creates a Single File cache store, `.dat` file, in the `/opt/infinispan/server/data` directory.
**Procedure**

- Add the `<file-store/>` element to the `persistence` configuration in your Data Grid cache, as in the following example:

```
<distributed-cache name="persistent-cache" mode="SYNC">
  <encoding media-type="application/x-protostream"/>
  <persistence>
    <file-store/>
  </persistence>
</distributed-cache>
```
CHAPTER 13. RUNNING BATCH OPERATIONS

Data Grid Operator provides a **Batch** CR that lets you create Data Grid resources in bulk. **Batch** CR uses the Data Grid command line interface (CLI) in batch mode to carry out sequences of operations.

**NOTE**

Modifying a **Batch** CR instance has no effect. Batch operations are "one-time" events that modify Data Grid resources. To update .spec fields for the CR, or when a batch operation fails, you must create a new instance of the **Batch** CR.
CHAPTER 14. RUNNING INLINE BATCH OPERATIONS

Include your batch operations directly in a Batch CR if they do not require separate configuration artifacts.

Procedure

1. Create a Batch CR.
   
   a. Specify the name of the Data Grid cluster where you want the batch operations to run as the value of the spec.cluster field.
   
   b. Add each CLI command to run on a line in the spec.config field.

```
apiVersion: infinispan.org/v2alpha1
kind: Batch
metadata:
  name: mybatch
spec:
  cluster: example-infinispan
  config:
    create cache --template=org.infinispan.DIST_SYNC mycache
    put --cache=mycache hello world
    put --cache=mycache hola mundo
```

2. Apply your Batch CR.

   
   $ oc apply -f mybatch.yaml

3. Check the status.Phase field in the Batch CR to verify the operations completed successfully.

14.1. CREATING CONFIGMAPS FOR BATCH OPERATIONS

Create a ConfigMap so that additional files, such as Data Grid cache configuration, are available for batch operations.

Prerequisites

For demonstration purposes, you should add some configuration artifacts to your host filesystem before you start the procedure:

- Create a /tmp/mybatch directory where you can add some files.

  
  $ mkdir -p /tmp/mybatch

- Create a Data Grid cache configuration.

  
  $ cat > /tmp/mybatch/mycache.xml<<EOF
  <distributed-cache name="mycache" mode="SYNC">
    <encoding media-type="application/x-protostream"/>
    <memory max-count="1000000" when-full="REMOVE"/>
  </distributed-cache>
  EOF
1. Create a **batch** file that contains all commands you want to run. For example, the following **batch** file creates a cache named "mycache" and adds two entries to it:

```
create cache mycache --file=/etc/batch/mymcache.xml
put --cache=mymcache hello world
put --cache=mymcache hola mundo
```

**IMPORTANT**

The **ConfigMap** is mounted in Data Grid pods at `/etc/batch`. You must prepend all `--file=` directives in your batch operations with that path.

2. Ensure all configuration artifacts that your batch operations require are in the same directory as the **batch** file.

```
$ ls /tmp/mybatch
batch
mycache.xml
```

3. Create a **ConfigMap** from the directory.

```
$ oc create configmap mybatch-config-map --from-file=/tmp/mybatch
```

### 14.2. RUNNING BATCH OPERATIONS WITH CONFGMAPS

Run batch operations that include configuration artifacts.

**Prerequisites**

- Create a **ConfigMap** that contains any files your batch operations require.

**Procedure**

1. Create a **Batch** CR that specifies the name of a Data Grid cluster as the value of the **spec.cluster** field.

```
$ cat > mybatch.yaml<<EOF
apiVersion: infinispan.org/v2alpha1
kind: Batch
metadata:
  name: mybatch
spec:
  cluster: example-infinispan
configMap: mybatch-config-map
EOF
```

2. Set the name of the **ConfigMap** that contains your **batch** file and configuration artifacts with the **spec.configMap** field.
3. Apply your **Batch** CR.

   $ oc apply -f mybatch.yaml

4. Check the `status.Phase` field in the **Batch** CR to verify the operations completed successfully.

### 14.3. BATCH STATUS MESSAGES

Verify and troubleshoot batch operations with the `status.Phase` field in the **Batch** CR.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Succeeded</td>
<td>All batch operations have completed successfully.</td>
</tr>
<tr>
<td>Initializing</td>
<td>Batch operations are queued and resources are initializing.</td>
</tr>
<tr>
<td>Initialized</td>
<td>Batch operations are ready to start.</td>
</tr>
<tr>
<td>Running</td>
<td>Batch operations are in progress.</td>
</tr>
<tr>
<td>Failed</td>
<td>One or more batch operations were not successful.</td>
</tr>
</tbody>
</table>

**Failed operations**

Batch operations are not atomic. If a command in a batch script fails, it does not affect the other operations or cause them to rollback.

**NOTE**

If your batch operations have any server or syntax errors, you can view log messages in the **Batch** CR in the `status.Reason` field.

### 14.4. EXAMPLE BATCH OPERATIONS

Use these example batch operations as starting points for creating and modifying Data Grid resources with the **Batch** CR.

**NOTE**

You can pass configuration files to Data Grid Operator only via a **ConfigMap**.

The **ConfigMap** is mounted in Data Grid pods at `/etc/batch` so you must prepend all `--file=` directives with that path.

**14.4.1. Caches**

- Create multiple caches from configuration files.

  ```
  echo "creating caches..."
  ```
Create a template from a file and then create caches from the template.

```
create cache mytemplate --file=/etc/batch/mycache.xml
create cache sessions --template=mytemplate
create cache tokens --template=mytemplate
```

14.4.2. Counters

Use the Batch CR to create multiple counters that can increment and decrement to record the count of objects.

You can use counters to generate identifiers, act as rate limiters, or track the number of times a resource is accessed.

```
echo "creating counters..."
create counter --concurrency-level=1 --initial-value=5 --storage=PERSISTENT --type=weak mycounter1
create counter --initial-value=3 --storage=PERSISTENT --type=strong mycounter2
create counter --initial-value=13 --storage=PERSISTENT --type=strong --upper-bound=10 mycounter3
```

14.4.3. Protobuf schema

Register Protobuf schema to query values in caches. Protobuf schema (.proto files) provide metadata about custom entities and controls field indexing.

```
echo "creating schema..."
schema --upload=person.proto person.proto
schema --upload=book.proto book.proto
schema --upload=author.proto book.proto
```

14.4.4. Tasks

Upload tasks that implement org.infinispan.tasks.ServerTask or scripts that are compatible with the javax.script scripting API.

```
echo "creating tasks..."
task upload --file=/etc/batch/myfirstscript.js myfirstscript
```
task upload --file=/etc/batch/mysecondscript.js mysecondscript
task upload --file=/etc/batch/mythirdscript.js mythirdscript
echo "list tasks"
lst tasks

Additional resources

- Data Grid CLI Guide
CHAPTER 15. BACKING UP AND RESTORING DATA GRID CLUSTERS

Data Grid Operator watches for custom resources (CR) that let you back up and restore Data Grid cluster state for disaster recovery or when migrating between Data Grid versions.

Backup CR
Archives Data Grid cluster content to a persistent volume.

Restore CR
Restores archived content to a Data Grid cluster.

NOTE
Modifying existing Backup or Restore CR instances has no effect. Backup and restore operations are "one-time" events that modify Data Grid resources. To update .spec fields for the CR, or when a backup or restore operation fails, you must create a new instance of the Backup or Restore CR.

= Backing up Data Grid clusters

Create a backup file that stores Data Grid cluster state to a persistent volume.

Prerequisites

- Create an Infinispan CR of spec.service.type: DataGrid.
- Have some resources on your Data Grid cluster to back up. Backups archive all resources that the Cache Manager controls, including caches, cache entries, cache templates, Protobuf schema, counters, scripts, and so on.

IMPORTANT
Data Grid backups do not provide snapshot isolation. If a write operation occurs on a cache entry that the backup operation has already archived, that write might not be backed up. To ensure that you archive the exact state of the cluster, make sure there are no active client connections to the cluster before you back it up.

Procedure

1. Create a Backup CR.
   a. Name the Backup CR with the metadata.name field.
   b. Specify the Data Grid cluster to backup with the spec.cluster field.

```yaml
apiVersion: infinispan.org/v2alpha1
kind: Backup
metadata:
  name: my-backup
spec:
  cluster: source-cluster
```
2. Add the `spec.resources` field if you only want to back up certain resources.

```
spec:
  resources:
    templates:
      - distributed-sync-prod
      - distributed-sync-dev
    caches:
      - cache-one
      - cache-two
    counters:
      - counter-name
    protoSchemas:
      - authors.proto
      - books.proto
    tasks:
      - wordStream.js
```

You can also use the * wildcard character to back up all resources.

```
spec:
  resources:
    caches:
      - "***"
    protoSchemas:
      - "***"
```

3. Apply your Backup CR.

```
$ oc apply -f my-backup.yaml
```

A new pod joins the Data Grid cluster and creates the backup file. When the operation is complete, the pod leaves the cluster and logs the following message:

```
ISPN005044: Backup file created 'my-backup.zip'
```

The resulting backup file is stored in the `/opt/infinispan/backups` directory.

4. Run the following command to verify that the backup is successful:

```
$ oc describe Backup my-backup
```

## 15.1. RESTORING DATA GRID CLUSTERS

Restore Data Grid cluster state from a backup archive.

**Prerequisites**

- Create a Backup CR on a source cluster.
- Create a Data Grid cluster of Data Grid service nodes where you want to restore state.
IMPORTANT

Make sure there are no active client connections to the cluster before you restore the backup. Cache entries that you restore from a backup could overwrite more recent cache entries. For example, a client does \texttt{cache.put(k=2)} before you restore a backup that contains \texttt{k=1}.

Procedure

1. Create a \texttt{Restore} CR.
   a. Name the \texttt{Restore} CR with the \texttt{metadata.name} field.
   b. Specify a \texttt{Backup} CR to use with the \texttt{spec.backup} field.
   c. Specify the Data Grid cluster to restore with the \texttt{spec.cluster} field.

   ```yaml
   apiVersion: infinispan.org/v2alpha1
   kind: Restore
   metadata:
     name: my-restore
   spec:
     backup: my-backup
     cluster: target-cluster
   ```

2. Add the \texttt{spec.resources} field to restore specific resources only.

   ```yaml
   spec:
     resources:
       templates:
         - distributed-sync-prod
         - distributed-sync-dev
       caches:
         - cache-one
         - cache-two
       counters:
         - counter-name
       protoSchemas:
         - authors.proto
         - books.proto
       tasks:
         - wordStream.js
   ```

3. Apply your \texttt{Restore} CR.

   ```bash
   $ oc apply -f my-restore.yaml
   ```

   A new pod joins the Data Grid cluster and restores state from the backup file. When the operation is complete, the pod leaves the cluster and logs the following message:

   ```
   ISPN005045: Restore 'my-backup' complete
   ```

4. Open the Data Grid Console or establish a CLI connection to verify the caches and data are restored to the cluster.
CHAPTER 16. DEPLOYING CUSTOM CODE TO DATA GRID

Add custom code, such as scripts and event listeners, to your Data Grid clusters.

16.1. COPYING CODE ARTIFACTS

Before you can deploy custom code to Data Grid clusters, you need to add it to a persistent volume (PV).

This procedure explains how to use a temporary pod that mounts a persistent volume claim (PVC) that:

- Lets you add code artifacts to the PV (perform a write operation).
- Allows Data Grid pods to load code artifacts from the PV (perform a read operation).

To perform these read and write operations, you need certain PV access modes. However, support for different PVC access modes is platform dependent.

It is beyond the scope of this document to provide instructions for creating PVCs with different platforms. For simplicity, the following procedure shows a PVC with the `ReadWriteMany` access mode.

In some cases only the `ReadOnlyMany` or `ReadWriteOnce` access modes are available. You can use a combination of those access modes by reclaiming and reusing PVCs with the same `spec.volumeName`.

NOTE

Using `ReadWriteOnce` access mode results in all Data Grid nodes in a cluster being scheduled on the same OpenShift node.

Procedure

1. Change to the namespace for your Data Grid cluster.

   ```bash
   $ oc project rhdg-namespace
   ```

2. Create a PVC for your custom code artifacts, for example:

   ```yaml
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: datagrid-libs
   spec:
     accessModes:
     - ReadWriteMany
     requests:
       storage: 100Mi
   ```

3. Apply your PVC.

   ```bash
   $ oc apply -f datagrid-libs.yaml
   ```

4. Create a pod that mounts the PVC, for example:
5. Add the pod to the Data Grid namespace and wait for it to be ready.

```
$ oc apply -f datagrid-libs-pod.yaml
$ oc wait --for=condition=ready --timeout=2m pod/datagrid-libs-pod
```

6. Copy your code artifacts to the pod so that they are loaded into the PVC. For example to copy code artifacts from a local `libs` directory, do the following:

```
$ oc cp --no-preserve=true libs datagrid-libs-pod:/tmp/
```

7. Delete the pod.

```
$ oc delete pod datagrid-libs-pod
```

Additional resources

- Configuring persistent storage
- Persistent Volumes
- Access Modes
- How to manually reclaim and reuse OpenShift Persistent volumes that are "Released" (Red Hat Knowledgebase)

### 16.2. DEPLOYING CUSTOM CODE

Make your custom code available to Data Grid clusters.

**Prerequisites**

- Create a persistent volume claim (PVC) and copy your code artifacts to it.

**Procedure**
• Specify the persistent volume with `spec.dependencies.VolumeClaimName` in your Infinispan CR and then apply the changes.

```yaml
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
spec:
  replicas: 2
  dependencies:
    volumeClaimName:datagrid-libs
  service:
    type: DataGrid
```

**NOTE**

If you update your custom code on the persistent volume, you must restart the Data Grid cluster so it can load the changes.
CHAPTER 17. SENDING CLOUD EVENTS FROM DATA GRID CLUSTERS

Configure Data Grid as a Knative source by sending **CloudEvents** to Apache Kafka topics.

Sending cloud events with Red Hat OpenShift Serverless is available as a technology preview.

17.1. TECHNOLOGY PREVIEWS

Technology Preview features or capabilities are not supported with Red Hat production service-level agreements (SLAs) and might not be functionally complete.

Red Hat does not recommend using Technology Preview features or capabilities for production. These features provide early access to upcoming product features, which enables you to test functionality and provide feedback during the development process.

For more information, see Red Hat Technology Preview Features Support Scope.

17.2. CLOUD EVENTS

You can send **CloudEvents** from Data Grid clusters when entries in caches are created, updated, removed, or expired.

Data Grid sends structured events to Kafka in JSON format, as in the following example:

```json
{
   "specversion": "1.0",
   "source": "/infinispan/<cluster_name>/<cache_name>",
   "type": "org.infinispan.entry.created",
   "time": "<timestamp>",
   "subject": "<key-name>",
   "id": "key-name:CommandInvocation:node-name:0",
   "data": {
      "property": "value"
   }
}
```

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Prefixes events for Data Grid cache entries with <strong>org.infinispan.entry</strong>.</td>
</tr>
<tr>
<td>data</td>
<td>Entry value.</td>
</tr>
<tr>
<td>subject</td>
<td>Entry key, converted to string.</td>
</tr>
<tr>
<td>id</td>
<td>Generated identifier for the event.</td>
</tr>
</tbody>
</table>

17.3. ENABLING CLOUD EVENTS
Configure Data Grid to send **CloudEvents**.

**Prerequisites**

- Set up an Kafka cluster that listens for Data Grid topics.

**Procedure**

1. Add `spec.cloudEvents` to your **Infinispan** CR.
   
   a. Configure the number of acknowledgements with the `spec.cloudEvents.acks` field. Values are "0", "1", or "all".
   
   b. List Kafka servers to which Data Grid sends events with the `spec.cloudEvents.bootstrapServers` field.
   
   c. Specify the Kafka topic for Data Grid events with the `spec.cloudEvents.cacheEntriesTopic` field.

   ```yaml
   spec:
     cloudEvents:
       acks: "1"
       bootstrapServers: my-cluster-kafka-bootstrap_1.<namespace_1>.svc:9092,my-cluster-kafka-bootstrap_2.<namespace_2>.svc:9092
       cacheEntriesTopic: target-topic
   ```

2. Apply your changes.
CHAPTER 18. ESTABLISHING REMOTE CLIENT CONNECTIONS

Connect to Data Grid clusters from the Data Grid Console, Command Line Interface (CLI), and remote clients.

18.1. CLIENT CONNECTION DETAILS

Before you can connect to Data Grid, you need to retrieve the following pieces of information:

- Service hostname
- Port
- Authentication credentials, if required
- TLS certificate, if you use encryption

Service hostnames

The service hostname depends on how you expose Data Grid on the network or if your clients are running on OpenShift.

For clients running on OpenShift, you can use the name of the internal service that Data Grid Operator creates.

For clients running outside OpenShift, the service hostname is the location URL if you use a load balancer. For a node port service, the service hostname is the node host name. For a route, the service hostname is either a custom hostname or a system-defined hostname.

Ports

Client connections on OpenShift and through load balancers use port 11222.

Node port services use a port in the range of 30000 to 60000. Routes use either port 80 (unencrypted) or 443 (encrypted).

Additional resources

- Configuring Network Access to Data Grid
- Retrieving Credentials
- Retrieving TLS Certificates

18.2. DATA GRID CACHES

Cache configuration defines the characteristics and features of the data store and must be valid with the Data Grid schema. Data Grid recommends creating standalone files in XML or JSON format that define your cache configuration. You should separate Data Grid configuration from application code for easier validation and to avoid the situation where you need to maintain XML snippets in Java or some other client language.

To create caches with Data Grid clusters running on OpenShift, you should:
• Use **Cache CR** as the mechanism for creating caches through the OpenShift front end.

• Use **Batch CR** to create multiple caches at a time from standalone configuration files.

• Access Data Grid Console and create caches in XML or JSON format.

You can use Hot Rod or HTTP clients but Data Grid recommends **Cache CR** or **Batch CR** unless your specific use case requires programmatic remote cache creation.

### 18.3. CONNECTING THE DATA GRID CLI

Use the command line interface (CLI) to connect to your Data Grid cluster and perform administrative operations.

**Prerequisites**

• Download a CLI distribution so you can connect to Data Grid clusters on OpenShift.

The Data Grid CLI is available with the server distribution or as a native executable.

Follow the instructions in *Getting Started with Data Grid Server* for information on downloading and installing the CLI as part of the server distribution. For the native CLI, you should follow the installation instructions in the *README* file that is included in the ZIP download.

**NOTE**

It is possible to open a remote shell to a Data Grid node and access the CLI.

```
$ oc rsh example-infinispan-0
```

However using the CLI in this way consumes memory allocated to the container, which can lead to out of memory exceptions.

**Procedure**

1. Create a CLI connection to your Data Grid cluster.

   **Using the server distribution**

   ```
   $ bin/cli.sh -c https://$SERVICE_HOSTNAME:$PORT --trustall
   ```

   **Using the native CLI**

   ```
   $ ./redhat-datagrid-cli -c https://$SERVICE_HOSTNAME:$PORT --trustall
   ```

   Replace `$SERVICE_HOSTNAME:$PORT` with the hostname and port where Data Grid is available on the network.

2. Enter your Data Grid credentials when prompted.

3. Perform CLI operations as required, for example:

   a. List caches configured on the cluster with the `ls` command.
Additional resources

- Getting Started with Data Grid Server
- Data Grid Software Downloads
- Using the Data Grid Command Line Interface

18.4. ACCESSING DATA GRID CONSOLE

Access the console to create caches, perform administrative operations, and monitor your Data Grid clusters.

Prerequisites

- Expose Data Grid on the network so you can access the console through a browser. For example, configure a load balancer service or create a route.

Procedure

- Access the console from any browser at $SERVICE_HOSTNAME:$PORT. Replace $SERVICE_HOSTNAME:$PORT with the hostname and port where Data Grid is available on the network.

18.5. HOT ROD CLIENTS

Hot Rod is a binary TCP protocol that Data Grid provides for high-performance data transfer capabilities with remote clients.

Client intelligence

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

Hot Rod clients running on OpenShift can access internal IP addresses for Data Grid nodes 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.

Hot Rod clients running outside OpenShift must use BASIC intelligence.

18.5.1. Hot Rod client configuration API

You can programatically configure Hot Rod client connections with the ConfigurationBuilder interface.
NOTE

$\text{SERVICE\_HOSTNAME:PORT}$ denotes the hostname and port that allows access to your Data Grid cluster. You should replace these variables with the actual hostname and port for your environment.

On OpenShift

Hot Rod clients running on OpenShift can use the following configuration:

```java
import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
import org.infinispan.client.hotrod.configuration.SaslQop;
import org.infinispan.client.hotrod.impl.ConfigurationProperties;
...

ConfigurationBuilder builder = new ConfigurationBuilder();
    builder.addServer()
        .host("$\text{SERVICE\_HOSTNAME}$")
        .port(ConfigurationProperties.DEFAULT_HOTROD_PORT)
        .security().authentication()
        .username("username")
        .password("changeme")
        .realm("default")
        .saslQop(SaslQop.AUTH)
        .saslMechanism("SCRAM-SHA-512")
        .ssl()
        .sniHostName("$\text{SERVICE\_HOSTNAME}$")
        .trustStoreFileName("/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt")
        .trustStoreType("pem");
```

Outside OpenShift

Hot Rod clients running outside OpenShift can use the following configuration:

```java
import org.infinispan.client.hotrod.configuration.ClientIntelligence;
import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
import org.infinispan.client.hotrod.configuration.SaslQop;
...

ConfigurationBuilder builder = new ConfigurationBuilder();
    builder.addServer()
        .host("$\text{SERVICE\_HOSTNAME}$")
        .port("$\text{PORT}$")
        .security().authentication()
        .username("username")
        .password("changeme")
        .realm("default")
        .saslQop(SaslQop.AUTH)
        .saslMechanism("SCRAM-SHA-512")
        .ssl()
        .sniHostName("$\text{SERVICE\_HOSTNAME}$")
        .trustStoreFileName("/path/to/truststore.pkcs12")
        .trustStorePassword("trust\_store\_password")
        .trustStoreType("PKCS12");
    builder.clientIntelligence(ClientIntelligence.BASIC);
```
18.5.2. Hot Rod client properties

You can configure Hot Rod client connections with the `hotrod-client.properties` file on the application classpath.

**NOTE**

`$SERVICE_HOSTNAME:$PORT` denotes the hostname and port that allows access to your Data Grid cluster. You should replace these variables with the actual hostname and port for your environment.

**On OpenShift**

Hot Rod clients running on OpenShift can use the following properties:

```properties
# Connection
infinispan.client.hotrod.server_list=$SERVICE_HOSTNAME:$PORT

# Authentication
infinispan.client.hotrod.use_auth=true
infinispan.client.hotrod.auth_username=developer
infinispan.client.hotrod.auth_password=$PASSWORD
infinispan.client.hotrod.auth_server_name=$CLUSTER_NAME
infinispan.client.hotrod.sasl_properties.javax.security.sasl.qop=auth
infinispan.client.hotrod.sasl_mechanism=SCRAM-SHA-512

# Encryption
infinispan.client.hotrod.sni_host_name=$SERVICE_HOSTNAME
infinispan.client.hotrod.trust_store_file_name=/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
infinispan.client.hotrod.trust_store_type=pem
```

**Outside OpenShift**

Hot Rod clients running outside OpenShift can use the following properties:

```properties
# Connection
infinispan.client.hotrod.server_list=$SERVICE_HOSTNAME:$PORT

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

# Authentication
infinispan.client.hotrod.use_auth=true
infinispan.client.hotrod.auth_username=developer
infinispan.client.hotrod.auth_password=$PASSWORD
infinispan.client.hotrod.auth_server_name=$CLUSTER_NAME
infinispan.client.hotrod.sasl_properties.javax.security.sasl.qop=auth
infinispan.client.hotrod.sasl_mechanism=SCRAM-SHA-512

# Encryption
infinispan.client.hotrod.sni_host_name=$SERVICE_HOSTNAME
# Create a client trust store with tls.crt from your project.
infinispan.client.hotrod.trust_store_file_name=/path/to/truststore.pkcs12
infinispan.client.hotrod.trust_store_password=trust_store_password
infinispan.client.hotrod.trust_store_type=PCKS12
```
18.5.3. Configuring Hot Rod clients for certificate authentication

If you enable client certificate authentication, clients must present valid certificates when negotiating connections with Data Grid.

**Validate strategy**

If you use the **Validate** strategy, you must configure clients with a keystore so they can present signed certificates. You must also configure clients with Data Grid credentials and any suitable authentication mechanism.

**Authenticate strategy**

If you use the **Authenticate** strategy, you must configure clients with a keystore that contains signed certificates and valid Data Grid credentials as part of the distinguished name (DN). Hot Rod clients must also use the **EXTERNAL** authentication mechanism.

**NOTE**

If you enable security authorization, you should assign the Common Name (CN) from the client certificate a role with the appropriate permissions.

The following example shows a Hot Rod client configuration for client certificate authentication with the **Authenticate** strategy:

```java
import org.infinispan.client.hotrod.configuration.ConfigurationBuilder;
...
ConfigurationBuilder builder = new ConfigurationBuilder();
    builder.security()
        .authentication()
        .saslMechanism("EXTERNAL")
        .ssl()
        .keyStoreFileName("/path/to/keystore")
        .keyStorePassword("keystorepassword".toCharArray())
        .keyStoreType("PCKS12");
```

18.5.4. Creating caches from Hot Rod clients

You can remotely create caches on Data Grid clusters running on OpenShift with Hot Rod clients. However, Data Grid recommends that you create caches using Data Grid Console, the CLI, or with Cache CRs instead of with Hot Rod clients.

**Programmatically creating caches**

The following example shows how to add cache configurations to the **ConfigurationBuilder** and then create them with the **RemoteCacheManager**:

```java
import org.infinispan.client.hotrod.DefaultTemplate;
import org.infinispan.client.hotrod.RemoteCache;
import org.infinispan.client.hotrod.RemoteCacheManager;
...

builder.remoteCache("my-cache")
    .templateName(DefaultTemplate.DIST_SYNC);
builder.remoteCache("another-cache")
```
This example shows how to create a cache named CacheWithXMLConfiguration using the XMLStringConfiguration() method to pass the cache configuration as XML:

```java
import org.infinispan.client.hotrod.RemoteCacheManager;
import org.infinispan.commons.configuration.XMLStringConfiguration;
...

private void createCacheWithXMLConfiguration() {
    String cacheName = "CacheWithXMLConfiguration";
    String xml = String.format("<distributed-cache name="%s">" +
        "<encoding media-type="application/x-protostream"/>" +
        "<locking isolation="READ_COMMITTED"/>" +
        "<transaction mode="NON_XA"/>" +
        "<expiration lifespan="60000" interval="20000"/>" +
        "</distributed-cache>", cacheName);
    manager.administration().getOrCreateCache(cacheName, new XMLStringConfiguration(xml));
    System.out.println("Cache with configuration exists or is created.");
}
```

Using Hot Rod client properties
When you invoke cacheManager.getCache() calls for named caches that do not exist, Data Grid creates them from the Hot Rod client properties instead of returning null.

Add cache configuration to hotrod-client.properties as in the following example:

```
# Add cache configuration
infinispan.client.hotrod.cache.my-cache.template_name=org.infinispan.DIST_SYNC
infinispan.client.hotrod.cache.another-cache.configuration=<infinispan><cache-container>
    <distributed-cache name="another-cache"/></cache-container></infinispan>
infinispan.client.hotrod.cache.my-other-cache.configuration_url=file:/path/to/configuration.xml
```

18.6. ACCESSING THE REST API

Data Grid provides a RESTful interface that you can interact with using HTTP clients.

Prerequisites

- Expose Data Grid on the network so you can access the REST API.
  For example, configure a load balancer service or create a route.

Procedure
- Access the REST API with any HTTP client at `${SERVICE_HOSTNAME}:${PORT}/rest/v2`.
  Replace `${SERVICE_HOSTNAME}:${PORT}` with the hostname and port where Data Grid is available on the network.

**Additional resources**
- Data Grid REST API

**18.7. ADDING CACHES TO CACHE SERVICE NODES**

Cache service nodes include a default cache configuration with recommended settings. This default cache lets you start using Data Grid without the need to create caches.

**NOTE**

Because the default cache provides recommended settings, you should create caches only as copies of the default. If you want multiple custom caches you should create Data Grid service nodes instead of Cache service nodes.

**Procedure**

- Access the Data Grid Console and provide a copy of the default configuration in XML or JSON format.

- Use the Data Grid CLI to create a copy from the default cache as follows:

  ```bash
  [//containers/default]> create cache --template=default mycache
  ```

**18.7.1. Default cache configuration**

This topic describes default cache configuration for Cache service nodes.

```xml
<distributed-cache name="default"
    mode="SYNC"
    owners="2">
    <memory storage="OFF_HEAP"
        max-size="<maximum_size_in_bytes>"
        when-full="REMOVE"/>
    <partition-handling when-split="ALLOW_READ_WRITES"
        merge-policy="REMOVE_ALL"/> 
</distributed-cache>
```

**Default caches:**

- Use synchronous distribution to store data across the cluster.

- Create two replicas of each entry on the cluster.

- Store cache entries as bytes in native memory (off-heap).

- Define the maximum size for the data container in bytes. Data Grid Operator calculates the maximum size when it creates nodes.
- Evict cache entries to control the size of the data container. You can enable automatic scaling so that Data Grid Operator adds nodes when memory usage increases instead of removing entries.

- Use a conflict resolution strategy that allows read and write operations for cache entries, even if segment owners are in different partitions.

- Specify a merge policy that removes entries from the cache when Data Grid detects conflicts.