Red Hat Data Grid 8.3

Data Grid Operator Guide

Create Data Grid clusters on OpenShift

Last Updated: 2022-08-10

Red Hat Data Grid 8.3 Data Grid Operator Guide

Create Data Grid clusters on OpenShift

Legal Notice

Abstract

Data Grid Operator provides operational intelligence and reduces management complexity for deploying Data Grid on OpenShift.

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 8.3 Documentation
- Data Grid 8.3 Component Details
- Supported Configurations for Data Grid 8.3
- 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

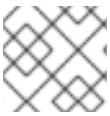
You must have a Red Hat account to access and download Data Grid software.

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.

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your feedback on our technical content and encourage you to tell us what you think. If you'd like to add comments, provide insights, correct a typo, or even ask a question, you can do so directly in the documentation.



NOTE

You must have a Red Hat account and be logged in to the customer portal.

To submit documentation feedback from the customer portal, do the following:

1. Select the **Multi-page HTML** format.
2. Click the **Feedback** button at the top-right of the document.
3. Highlight the section of text where you want to provide feedback.
4. Click the **Add Feedback** dialog next to your highlighted text.
5. Enter your feedback in the text box on the right of the page and then click **Submit**.

We automatically create a tracking issue each time you submit feedback. Open the link that is displayed after you click **Submit** and start watching the issue or add more comments.

Thank you for the valuable feedback.

CHAPTER 1. DATA GRID OPERATOR

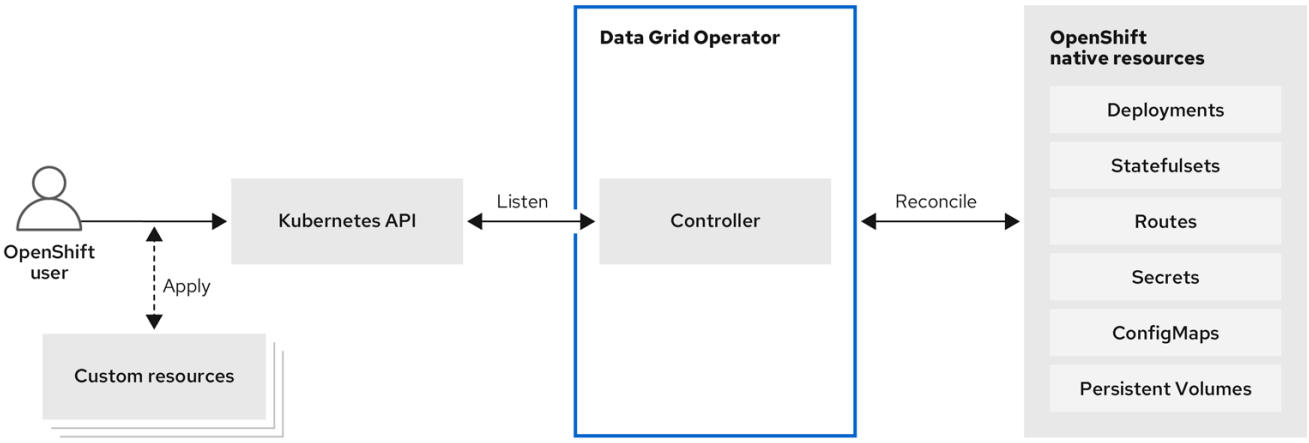
Data Grid Operator provides operational intelligence and reduces management complexity for deploying Data Grid on OpenShift and Red Hat OpenShift.

1.1. DATA GRID OPERATOR DEPLOYMENTS

When you install Data Grid Operator, it extends the Kubernetes API with Custom Resource Definitions (CRDs) for deploying and managing Data Grid clusters on Red Hat OpenShift.

To interact with Data Grid Operator, OpenShift users apply Custom Resources (CRs) through the OpenShift Web Console or **oc** client. Data Grid Operator listens for **Infinispan** CRs and automatically provisions native resources, such as StatefulSets and Secrets, that your Data Grid deployment requires. Data Grid Operator also configures Data Grid services according to the specifications in **Infinispan** CRs, including the number of pods for the cluster and backup locations for cross-site replication.

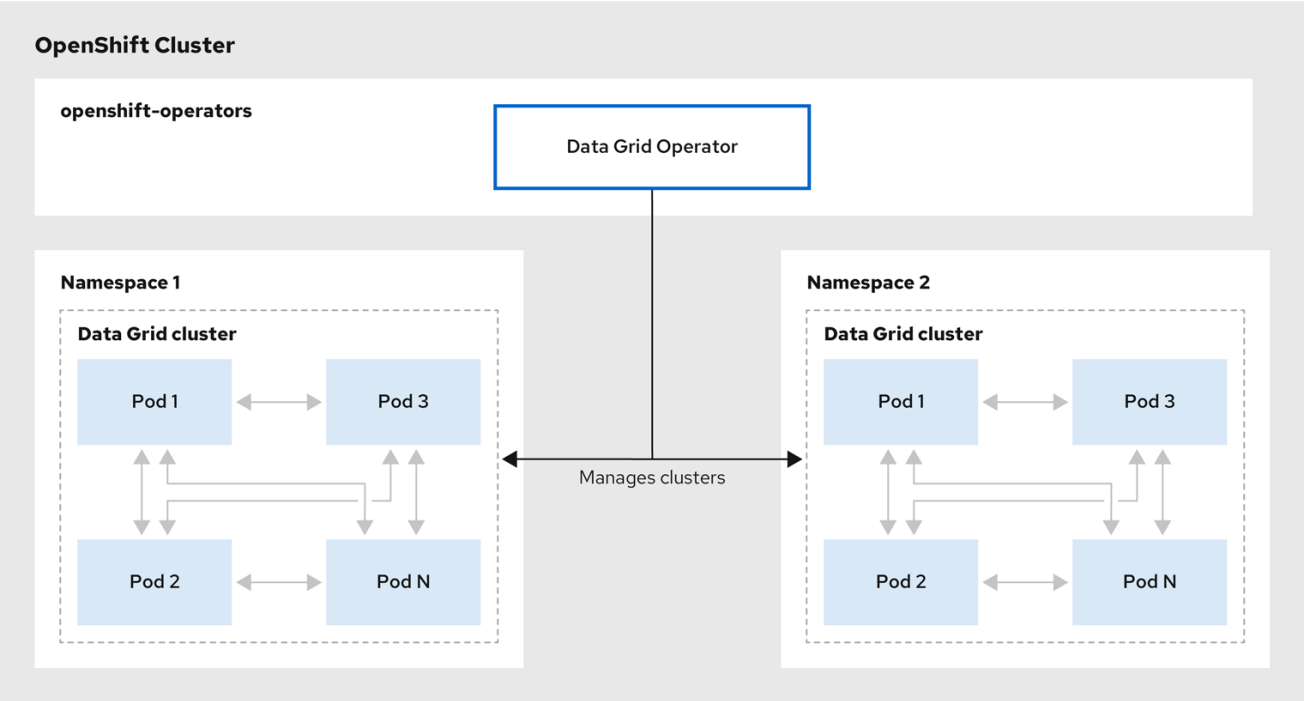
Figure 1.1. Custom resources



1.2. CLUSTER MANAGEMENT

A single Data Grid Operator installation can manage multiple Data Grid clusters in separate namespaces. Each time a user applies CRs to modify the deployment, Data Grid Operator applies the changes globally to all Data Grid clusters.

Figure 1.2. Operator-managed clusters



1.3. RESOURCE RECONCILIATION

Data Grid Operator reconciles custom resources such as the **Cache** CR with resources on your Data Grid cluster.

Bidirectional reconciliation synchronizes your CRs with changes that you make to Data Grid resources through the Data Grid Console, command line interface (CLI), or other client application and vice versa. For example if you create a cache through the Data Grid Console then Data Grid Operator adds a declarative Kubernetes representation.

To perform reconciliation Data Grid Operator creates a **listener** pod for each Data Grid cluster that detects modifications for **Infinispan** resources.

Notes about reconciliation

- When you create a cache through the Data Grid Console, CLI, or other client application, Data Grid Operator creates a corresponding **Cache** CR with a unique name that conforms to the Kubernetes naming policy.
- Declarative Kubernetes representations of Data Grid resources that Data Grid Operator creates with the **listener** pod are linked to **Infinispan** CRs.
  Deleting **Infinispan** CRs removes any associated resource declarations.

CHAPTER 2. INSTALLING THE NATIVE DATA GRID CLI AS A CLIENT PLUGIN

Data Grid provides a command line interface (CLI) compiled to a native executable that you can install as a plugin for **oc** clients. You can then use your **oc** client to:

- Create Data Grid Operator subscriptions and remove Data Grid Operator installations.
- Set up Data Grid clusters and configure services.
- Work with Data Grid resources via remote shells.

2.1. INSTALLING THE NATIVE DATA GRID CLI PLUGIN

Install the native Data Grid Command Line Interface (CLI) as a plugin for **oc** clients.

Prerequisites

- Have an **oc** client.
- Download the native Data Grid CLI distribution from the Data Grid software downloads.

Procedure

1. Extract the **.zip** archive for the native Data Grid CLI distribution.
2. Copy the native executable, or create a hard link, to a file named "kubectl-infinispan", for example:

```
cp redhat-datagrid-cli kubectl-infinispan
```

3. Add **kubectl-infinispan** to your **PATH**.
4. Verify that the CLI is installed.

```
oc plugin list

The following compatible plugins are available:
/path/to/kubectl-infinispan
```

5. Use the **infinispan --help** command to view available commands.

```
oc infinispan --help
```

Additional resources

- Extending the OpenShift CLI with plug-ins

2.2. KUBECTL-INFINISPAN COMMAND REFERENCE

This topic provides some details about the **kubectl-infinispan** plugin for clients.

TIP

Use the **--help** argument to view the complete list of available options and descriptions for each command.

For example, **oc infinispan create cluster --help** prints all command options for creating Data Grid clusters.

Command	Description
oc infinispan install	Creates Data Grid Operator subscriptions and installs into the global namespace by default.
oc infinispan create cluster	Creates Data Grid clusters.
oc infinispan get clusters	Displays running Data Grid clusters.
oc infinispan shell	Starts an interactive remote shell session on a Data Grid cluster.
oc infinispan delete cluster	Removes Data Grid clusters.
oc infinispan uninstall	Removes Data Grid Operator installations and all managed resources.

CHAPTER 3. INSTALLING DATA GRID OPERATOR

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

3.1. INSTALLING DATA GRID OPERATOR ON RED HAT OPENSHIFT

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 8.3.x.

   Approval Strategies

   Automatically install updates from the 8.3.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.

3.2. INSTALLING DATA GRID OPERATOR WITH THE NATIVE CLI PLUGIN

Install Data Grid Operator with the native Data Grid CLI plugin, **kubectl-infinispan**.

Prerequisites

- Have **kubectl-infinispan** on your **PATH**.

Procedure

1. Run the **oc infinispan install** command to create Data Grid Operator subscriptions, for example:

```
oc infinispan install --channel=8.3.x
                      --source=redhat-operators
                      --source-namespace=openshift-marketplace
```

2. Verify the installation.

```
oc get pods -n openshift-operators | grep infinispan-operator
NAME                              READY   STATUS
infinispan-operator-<id>            1/1     Running
```

TIP

Use **oc infinispan install --help** for command options and descriptions.

3.3. INSTALLING DATA GRID OPERATOR WITH AN OPENSHIFT CLIENT

You can use the **oc** client to create Data Grid Operator subscriptions as an alternative to installing through the **OperatorHub** or with the native Data Grid CLI.

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} 1
oc new-project ${WATCH_NAMESPACE} 2
```

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
```

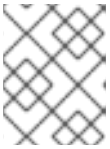
```
kind: OperatorGroup
metadata:
 name: datagrid
 namespace: ${INSTALL_NAMESPACE}
EOF
```

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: 8.3.x
 installPlanApproval: Automatic
 name: datagrid
 source: redhat-operators
 sourceNamespace: openshift-marketplace
EOF
```



NOTE

If you want to manually approve updates from the 8.3.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
```

CHAPTER 4. CREATING DATA GRID CLUSTERS

Create Data Grid clusters running on OpenShift with the **Infinispan** CR or with the native Data Grid CLI plugin for **oc** clients.

4.1. 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 listens for **Infinispan** Custom Resources (CR) that you use to instantiate and configure Data Grid clusters and manage OpenShift resources, such as StatefulSets and Services.

Infinispan CR

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

Field	Description
apiVersion	Declares the version of the **Infinispan** API.
kind	Declares the **Infinispan** CR.
metadata.name	Specifies a name for your Data Grid cluster.
spec.replicas	Specifies the number of pods in your Data Grid cluster.
spec.service.type	Specifies the type of Data Grid service to create.

4.2. CREATING DATA GRID CLUSTERS

Create Data Grid clusters with the native CLI plugin, **kubectl-infinispan**.

Prerequisites

- Install Data Grid Operator.
- Have **kubectl-infinispan** on your **PATH**.

Procedure

1. Run the **infinispan create cluster** command.

For example, create a Data Grid cluster with two pods as follows:

```
oc infinispan create cluster --replicas=3 -Pservice.type=DataGrid infinispan
```

2. Watch Data Grid Operator create the Data Grid pods.

```
oc get pods -w
```

Next steps

After you create a Data Grid cluster, use the **oc** to apply changes to **Infinispan** CR and configure your Data Grid service.

You can also delete Data Grid clusters with **kubectl-infinispan** and re-create them as required.

```
oc infinispan delete cluster infinispan
```

4.3. VERIFYING DATA GRID CLUSTER VIEWS

Confirm that Data Grid pods have successfully formed clusters.

Prerequisites

- Create at least one Data Grid cluster.

Procedure

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

```
oc get infinispan -o yaml
```

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

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

TIP

Do the following for automated scripts:

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

Retrieving cluster view from logs

You can also get the cluster view from Data Grid logs as follows:

```
oc logs infinispan-0 | grep ISPN000094
```

```
INFO  [org.infinispan.CLUSTER] (MSC service thread 1-2) \
```

```
ISPN000094: Received new cluster view for channel infinispan: \
[infinispan-0|0] (1) [infinispan-0]

INFO  [org.infinispan.CLUSTER] (jgroups-3,infinispan-0) \
ISPN000094: Received new cluster view for channel infinispan: \
[infinispan-0|1] (2) [infinispan-0, infinispan-1]
```

4.4. MODIFYING DATA GRID CLUSTERS

Configure Data Grid clusters by providing Data Grid Operator with a custom **Infinispan** CR.

Prerequisites

- Install Data Grid Operator.
- Create at least one Data Grid cluster.
- Have an **oc** client.

Procedure

1. Create a YAML file that defines your **Infinispan** CR.
   For example, create a **my_infinispan.yaml** file that changes the number of Data Grid pods to two:

```
cat > cr_minimal.yaml<<EOF
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
spec:
  replicas: 2
  service:
    type: DataGrid
EOF
```

2. Apply your **Infinispan** CR.

```
oc apply -f my_infinispan.yaml
```

3. Watch Data Grid Operator scale the Data Grid pods.

```
oc get pods -w
```

4.5. STOPPING AND STARTING DATA GRID CLUSTERS

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

Clusters of Data Grid service pods must restart with the same number of pods 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 pods.

Procedure

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

```
spec:
  replicas: 0
```

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

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

3. Change the **spec.replicas** field to the same number of pods to restart the Data Grid cluster.

```
spec:
  replicas: 6
```

CHAPTER 5. CONFIGURING DATA GRID CLUSTERS

Apply custom Data Grid configuration to clusters that Data Grid Operator manages.

5.1. APPLYING CUSTOM CONFIGURATION TO DATA GRID CLUSTERS

Add Data Grid configuration to a **ConfigMap** and make it available to Data Grid Operator. Data Grid Operator can then apply the custom configuration to your Data Grid cluster.

IMPORTANT

Data Grid Operator applies default configuration on top of your custom configuration to ensure it can continue to manage your Data Grid clusters.

Be careful when applying custom configuration outside the **cache-container** element or field. You can apply custom configuration to underlying Data Grid Server mechanisms such as endpoints, security realms, and cluster transport. Changing this configuration can result in error and result in service downtime for your Data Grid deployment.

TIP

Use the Data Grid Helm chart to deploy clusters of fully configurable Data Grid Server instances on OpenShift.

Prerequisites

- Have valid Data Grid configuration in XML, YAML, or JSON format.

Procedure

1. Add Data Grid configuration to a **infinispan-config.[xml|yaml|json]** key in the **data** field of your **ConfigMap**.

   XML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-config
  namespace: rhdg-namespace
data:
  infinispan-config.xml: >
    <infinispan>
      <!-- Custom configuration goes here. -->
    </infinispan>
```

   YAML

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-config
  namespace: rhdg-namespace
```

```
data:
  infinispan-config.yaml: >
    infinispan:
      # Custom configuration goes here.
```

JSON

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-config
  namespace: rhdg-namespace
data:
  infinispan-config.json: >
    {
      "infinispan": {
      }
    }
```

2. Create the **ConfigMap** from your YAML file.

```
oc apply -f cluster-config.yaml
```

3. Specify the name of the **ConfigMap** with the **spec.configMapName** field in your **Infinispan** CR and then apply the changes.

```
spec:
  configMapName: "cluster-config"
```

Next steps

If your cluster is already running Data Grid Operator restarts it to apply the configuration. Each time you modify the Data Grid configuration in the **ConfigMap**, Data Grid Operator detects the updates and restarts the cluster to apply the changes.

Additional resources

- Data Grid Helm chart

5.2. CUSTOM DATA GRID CONFIGURATION

You can add Data Grid configuration to a **ConfigMap** in XML, YAML, or JSON format.

Cache template

XML

```
<infinispan>
  <cache-container>
    <distributed-cache-configuration name="base-template">
      <expiration lifespan="5000"/>
    </distributed-cache-configuration>
    <distributed-cache-configuration name="extended-template"
                                     configuration="base-template">
```

```
    <encoding media-type="application/x-protostream"/>
    <expiration lifespan="10000"
                max-idle="1000"/>
  </distributed-cache-configuration>
 </cache-container>
</infinispan>
```

YAML

```
infinispan:
  cacheContainer:
    caches:
      base-template:
        distributedCacheConfiguration:
          expiration:
            lifespan: "5000"
      extended-template:
        distributedCacheConfiguration:
          configuration: "base-template"
          encoding:
            mediaType: "application/x-protostream"
          expiration:
            lifespan: "10000"
            maxIdle: "1000"
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "caches" : {
        "base-template" : {
          "distributed-cache-configuration" : {
            "expiration" : {
              "lifespan" : "5000"
            }
          }
        },
        "extended-template" : {
          "distributed-cache-configuration" : {
            "configuration" : "base-template",
            "encoding": {
              "media-type": "application/x-protostream"
              },
            "expiration" : {
              "lifespan" : "10000",
              "max-idle" : "1000"
            }
          }
        }
      }
    }
  }
}
```

Multiple caches

XML

```
<infinispan
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="urn:infinispan:config:13.0 https://infinispan.org/schemas/infinispan-config-
13.0.xsd
                          urn:infinispan:server:13.0 https://infinispan.org/schemas/infinispan-server-13.0.xsd"
      xmlns="urn:infinispan:config:13.0"
      xmlns:server="urn:infinispan:server:13.0">
  <cache-container name="default"
                   statistics="true">
    <distributed-cache name="mycacheone"
                       mode="ASYNC"
                       statistics="true">
      <encoding media-type="application/x-protostream"/>
      <expiration lifespan="300000"/>
      <memory max-size="400MB"
              when-full="REMOVE"/>
    </distributed-cache>
    <distributed-cache name="mycachetwo"
                       mode="SYNC"
                       statistics="true">
      <encoding media-type="application/x-protostream"/>
      <expiration lifespan="300000"/>
      <memory max-size="400MB"
              when-full="REMOVE"/>
    </distributed-cache>
  </cache-container>
</infinispan>
```

YAML

```
infinispan:
  cacheContainer:
    name: "default"
    statistics: "true"
    caches:
      mycacheone:
       distributedCache:
          mode: "ASYNC"
          statistics: "true"
          encoding:
            mediaType: "application/x-protostream"
          expiration:
            lifespan: "300000"
          memory:
            maxSize: "400MB"
            whenFull: "REMOVE"
      mycachetwo:
        distributedCache:
          mode: "SYNC"
          statistics: "true"
          encoding:
```

```
        mediaType: "application/x-protostream"
      expiration:
        lifespan: "300000"
      memory:
        maxSize: "400MB"
        whenFull: "REMOVE"
```

JSON

```
{
  "infinispan" : {
    "cache-container" : {
      "name" : "default",
      "statistics" : "true",
      "caches" : {
        "mycacheone" : {
          "distributed-cache" : {
            "mode": "ASYNC",
            "statistics": "true",
            "encoding": {
              "media-type": "application/x-protostream"
            },
            "expiration" : {
              "lifespan" : "300000"
            },
            "memory": {
              "max-size": "400MB",
              "when-full": "REMOVE"
            }
          }
        },
        "mycachetwo" : {
          "distributed-cache" : {
            "mode": "SYNC",
            "statistics": "true",
            "encoding": {
              "media-type": "application/x-protostream"
            },
            "expiration" : {
              "lifespan" : "300000"
            },
            "memory": {
              "max-size": "400MB",
              "when-full": "REMOVE"
            }
          }
        }
      }
    }
  }
}
```

Logging configuration

You can also include Apache Log4j configuration in XML format as part of your **ConfigMap**.

NOTE

Use the **spec.logging.categories** field in your **Infinispan** CR to adjust logging levels for Data Grid clusters. Add Apache Log4j configuration only if you require advanced file-based logging capabilities.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: logging-config
  namespace: rhdg-namespace
data:
  infinispan-config.xml: >
    <infinispan>
      <!-- Add custom Data Grid configuration if required. -->
      <!-- You can provide either Data Grid configuration, logging configuration, or both. -->
    </infinispan>

  log4j.xml: >
    <?xml version="1.0" encoding="UTF-8"?>
    <Configuration name="ServerConfig" monitorInterval="60" shutdownHook="disable">
        <Appenders>
            <!-- Colored output on the console -->
            <Console name="STDOUT">
                <PatternLayout pattern="%d{HH:mm:ss,SSS} %-5p (%t) [%c] %m%throwable%n"/>
            </Console>
        </Appenders>

        <Loggers>
            <Root level="INFO">
                <AppenderRef ref="STDOUT" level="TRACE"/>
            </Root>
            <Logger name="org.infinispan" level="TRACE"/>
        </Loggers>
    </Configuration>
```

CHAPTER 6. UPGRADING DATA GRID CLUSTERS

Data Grid Operator handles Data Grid cluster upgrades when new versions become available.

IMPORTANT

Hot Rod rolling upgrades are available as a technology preview feature.

6.1. TECHNOLOGY PREVIEW FEATURES

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 .

6.2. DATA GRID CLUSTER UPGRADES

The **spec.upgrades.type** field controls how Data Grid Operator upgrades your Data Grid cluster when new versions become available. There are two types of cluster upgrade:

Shutdown

Upgrades Data Grid clusters with service downtime. This is the default upgrade type.

HotRodRolling

Upgrades Data Grid clusters without service downtime.

Shutdown upgrades

To perform a shutdown upgrade, Data Grid Operator does the following:

1. Gracefully shuts down the existing cluster.
2. Removes the existing cluster.
3. Creates a new cluster with the target version.

Hot Rod rolling upgrades

To perform a Hot Rod rolling upgrade, Data Grid Operator does the following:

1. Creates a new Data Grid cluster with the target version that runs alongside your existing cluster.
2. Creates a remote cache store to transfer data from the existing cluster to the new cluster.
3. Redirects all clients to the new cluster.
4. Removes the existing cluster when all data and client connections are transferred to the new cluster.

IMPORTANT

You should not perform Hot Rod rolling upgrades with caches that enable passivation with persistent cache stores. In the event that the upgrade does not complete successfully, passivation can result in data loss when Data Grid Operator rolls back the target cluster.

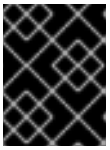
If your cache configuration enables passivation you should perform a shutdown upgrade.

6.3. UPGRADING DATA GRID CLUSTERS WITH DOWNTIME

Upgrading Data Grid clusters with downtime results in service disruption but does not require any additional capacity.

Prerequisites

- If required, configure a persistent cache store to preserve your data during the upgrade.



IMPORTANT

At the start of the upgrade process Data Grid Operator shuts down your existing cluster. This results in data loss if you do not configure a persistent cache store.

Procedure

1. Ensure that **Shutdown** is set as the value for the **spec.upgrades.type** field, which is the default.

```
spec:
  upgrades:
    type: Shutdown
```

2. Apply your changes, if necessary.

When it detects a new Data Grid version, Data Grid Operator automatically upgrades your cluster or prompts you to manually approve the upgrade before proceeding.

6.4. PERFORMING HOT ROD ROLLING UPGRADES FOR DATA GRID CLUSTERS

Performing Hot Rod rolling upgrades lets you move to a new Data Grid version without service disruption. However, this upgrade type requires additional capacity and temporarily results in two Data Grid clusters with different versions running concurrently.

Procedure

1. Specify **HotRodRolling** as the value for the **spec.upgrades.type** field.

```
spec:
  upgrades:
    type: HotRodRolling
```

2. Apply your changes.

When it detects a new Data Grid version, Data Grid Operator automatically upgrades your cluster or prompts you to manually approve the upgrade before proceeding.

CHAPTER 7. SETTING UP DATA GRID SERVICES

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

7.1. SERVICE TYPES

Services are stateful applications, based on the Data Grid Server image, that provide flexible and robust in-memory data storage. When you create Data Grid clusters you specify either **DataGrid** or **Cache** as the service type with the **spec.service.type** field.

DataGrid service type

Deploy Data Grid clusters with full configuration and capabilities.

Cache service type

Deploy Data Grid clusters with minimal configuration.

Red Hat recommends recommends the **DataGrid** service type for clusters because it lets you:

- 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.

IMPORTANT

The **Cache** service type was designed to provide a convenient way to create a low-latency data store with minimal configuration. Additional development on the **Infinispan** CRD has shown that the **Cache** CR offers a better approach to achieving this goal, ultimately giving users more choice and less deployment overhead. For this reason, the **Cache** service type is planned for removal in the next version of the **Infinispan** CRD and is no longer under active development.

The **DataGrid** service type continues to benefit from new features and improved tooling to automate complex operations such as cluster upgrades and data migration.

7.2. CREATING DATA GRID SERVICE PODS

To use custom cache definitions along with Data Grid capabilities such as cross-site replication, create clusters of Data Grid service pods.

Procedure

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

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
```

```
spec:
  replicas: 2
  service:
    type: DataGrid
```

IMPORTANT

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

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

7.2.1. Data Grid service CR

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

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
  annotations:
    infinispan.org/monitoring: 'true'
spec:
  replicas: 6
  upgrades:
    type: Shutdown
  service:
    type: DataGrid
    container:
      storage: 2Gi
      # The ephemeralStorage and storageClassName fields are mutually exclusive.
      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: infinispan
        namespace: rhdg-namespace
        url: openshift://api.aws.host:6443
        secretName: aws-token
  security:
    endpointSecretName: endpoint-identities
    endpointEncryption:
        type: Secret
        certSecretName: tls-secret
  container:
    extraJvmOpts: "-XX:NativeMemoryTracking=summary"
```

```
    cpu: "2000m:1000m"
    memory: "2Gi:1Gi"
  logging:
    categories:
      org.infinispan: debug
      org.jgroups: debug
      org.jgroups.protocols.TCP: error
      org.jgroups.protocols.relay.RELAY2: error
  expose:
    type: LoadBalancer
  configMapName: "my-cluster-config"
  configListener:
    enabled: true
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchLabels:
              app: infinispan-pod
              clusterName: infinispan
              infinispan_cr: infinispan
          topologyKey: "kubernetes.io/hostname"
```

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 pods in your cluster.
spec.upgrades.type	Controls how Data Grid Operator upgrades your Data Grid cluster when new versions become available.
spec.service.type	Configures the type Data Grid service. A value of **DataGrid** creates a cluster with Data Grid service pods.
spec.service.container	Configures the storage resources for Data Grid service pods.
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.

Field	Description
spec.container	Specifies JVM, CPU, and memory resources for Data Grid pods.
spec.logging	Configures Data Grid logging categories.
spec.expose	Controls how Data Grid endpoints are exposed on the network.
spec.configMapName	Specifies a **ConfigMap** that contains Data Grid configuration.
spec.configListener	Creates a **listener** pod in each Data Grid cluster that allows Data Grid Operator to reconcile server-side modifications with Data Grid resources such as the **Cache** CR. The **listener** pod consumes minimal resources and is enabled by default. Setting a value of **false** removes the **listener** pod and disables bi-directional reconciliation. You should do this only if you do not need declarative Kubernetes representations of Data Grid resources created through the Data Grid Console, CLI, or client applications.
spec.affinity	Configures anti-affinity strategies that guarantee Data Grid availability.

7.3. ALLOCATING STORAGE RESOURCES

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

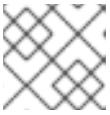
By default, Data Grid Operator allocates **1Gi** for the persistent volume claim. However you should adjust the amount of storage available to Data Grid service pods so that Data Grid can preserve cluster state during shutdown.

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. Configure either the **ephemeralStorage** field or the **storageClassName** field as required.



NOTE

These fields are mutually exclusive. Add only one of them to your **Infinispan** CR.

3. Apply the changes.

Ephemeral storage

```
spec:
  service:
    type: DataGrid
    container:
      storage: 2Gi
      ephemeralStorage: true
```

Name of a **StorageClass** object

```
spec:
  service:
    type: DataGrid
    container:
      storage: 2Gi
      storageClassName: my-storage-class
```

Field	Description
spec.service.container.storage	Specifies the amount of storage for Data Grid service pods.
spec.service.container.ephemeralStorage	Defines whether storage is ephemeral or permanent. Set the value to **true** to use ephemeral storage, which means all data in storage is deleted when clusters shut down or restart. The default value is **false**, which means storage is permanent.
spec.service.container.storageClassName	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 **storageclass.kubernetes.io/is-default-class** annotation set to **true**.

7.3.1. 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 pods.

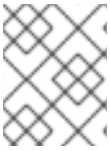
Data

Data is always volatile in clusters of Cache service pods. 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 pods to persist data during cluster shutdown.

7.4. ALLOCATING CPU AND MEMORY

Allocate CPU and memory resources to Data Grid pods with the **Infinispan** CR.



NOTE

Data Grid Operator requests **1Gi** of memory from the OpenShift scheduler when creating Data Grid pods. CPU requests are unbounded by default.

Procedure

1. Allocate the number of CPU units with the **spec.container.cpu** field.
2. Allocate the amount of memory, in bytes, with the **spec.container.memory** field.
   The **cpu** and **memory** fields have values in the format of **<limit>:<requests>**. For example, **cpu: "2000m:1000m"** limits pods to a maximum of **2000m** of CPU and requests **1000m** of CPU for each pod at startup. Specifying a single value sets both the limit and request.
3. Apply your **Infinispan** CR.
   If your cluster is running, Data Grid Operator restarts the Data Grid pods so changes take effect.

```
spec:
  container:
    cpu: "2000m:1000m"
    memory: "2Gi:1Gi"
```

7.5. SETTING JVM OPTIONS

Pass additional JVM options to Data Grid pods at startup.

Procedure

1. Configure JVM options with the **spec.container** filed in your **Infinispan** CR.
2. Apply your **Infinispan** CR.
   If your cluster is running, Data Grid Operator restarts the Data Grid pods so changes take effect.

JVM options

```
spec:
  container:
```

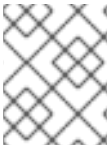
```
extraJvmOpts: "-<option>=<value>"
routerExtraJvmOpts: "-<option>=<value>"
cliExtraJvmOpts: "-<option>=<value>"
```

Field	Description
spec.container.extraJvmOpts	Specifies additional JVM options for the Data Grid Server.
spec.container.routerExtraJvmOpts	Specifies additional JVM options for the Gossip router.
spec.container.cliExtraJvmOpts	Specifies additional JVM options for the Data Grid CLI.

7.6. DISABLING FIPS MODE IN YOUR INFINISPAN CR

The Red Hat OpenShift Container Platform can use certain Federal Information Processing Standards (FIPS) components that ensure OpenShift clusters meet the requirements of a FIPS compliance audit. This might cause issues when you want your Data Grid instance to run on any OpenShift cluster that has FIPS mode enabled. Data Grid 8.3 does not support FIPS mode, so you must disable FIPS mode in your **Infinispan** CR.

After you disable FIPS mode in your **Infinispan** CR, any component that uses a JVM, such as Data Grid Server, Data Grid CLI, or Gossip router, ignores FIPS mode. This happens, because the JVM no longer loads FIPS-related cryptographic libraries on implementation startup.



NOTE

You need to explicitly disable FIPS mode in your **Infinispan** CR configuration only if FIPS mode is enabled for an OpenShift cluster.

Prerequisites

- Created **Infinispan** CR on OpenShift, so that your Data Grid Operator can interact with OpenShift.

Procedure

1. Configure JVM options with the **spec.container** field in your **Infinispan** CR.

```
spec:
  container:
    extraJvmOpts: "-Dcom.redhat.fips=false"
    cliExtraJvmOpts: "-Dcom.redhat.fips=false"
    routerExtraJvmOpts: "-Dcom.redhat.fips=false"
```

2. Apply your **Infinispan** CR.

Additional resources

- Support for FIPS cryptography Red Hat OpenShift Container Platform
- Setting JVM options

7.7. 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.

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

2. Apply the changes.
3. Retrieve logs from Data Grid pods as required.

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

7.7.1. Logging reference

Find information about log categories and levels.

Table 7.1. Log categories

Root category	Description	Default level
org.infinispan	Data Grid messages	**info**
org.jgroups	Cluster transport messages	**info**

Table 7.2. Log levels

Log level	Description
trace	Provides detailed information about running state of applications. This is the most verbose log level.
debug	Indicates the progress of individual requests or activities.
info	Indicates overall progress of applications, including lifecycle events.

Log level	Description
warn	Indicates circumstances that can lead to error or degrade performance.
error	Indicates error conditions that might prevent operations or activities from being successful but do not prevent applications from running.

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"
```

7.8. CREATING CACHE SERVICE PODS

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

IMPORTANT

Cache service pods provide volatile storage only, which means you lose all data when you modify your **Infinispan** CR or update the version of your Data Grid cluster.

Procedure

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

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

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

7.8.1. Cache service CR

This topic describes the **Infinispan** CR for Cache service pods.

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
```

```
  annotations:
    infinispan.org/monitoring: 'true'
spec:
  replicas: 2
  upgrades:
    type: Shutdown
  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:1000m"
    memory: "2Gi:1Gi"
  logging:
    categories:
      org.infinispan: trace
      org.jgroups: trace
  expose:
    type: LoadBalancer
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
        podAffinityTerm:
          labelSelector:
            matchLabels:
              app: infinispan-pod
              clusterName: infinispan
              infinispan_cr: infinispan
          topologyKey: "kubernetes.io/hostname"
```

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 pods in your cluster. If you enable autoscaling capabilities, this field specifies the initial number of pods.

Field	Description
spec.upgrades.type	Controls how Data Grid Operator upgrades your Data Grid cluster when new versions become available.
spec.service.type	Configures the type Data Grid service. A value of **Cache** creates a cluster with Cache service pods.
spec.service.replicationFactor	Sets the number of copies for each entry across the cluster. The default for Cache service pods is two, which replicates each cache entry to avoid data loss.
spec.autoscale	Enables and configures automatic scaling.
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 pods.
spec.logging	Configures Data Grid logging categories.
spec.expose	Controls how Data Grid endpoints are exposed on the network.
spec.affinity	Configures anti-affinity strategies that guarantee Data Grid availability.

7.9. AUTOMATIC SCALING

Data Grid Operator can monitor the default cache on Cache service pods to automatically scale clusters up or down, by creating or deleting pods based on memory usage.

IMPORTANT

Automatic scaling is available for clusters of Cache service pods only. Data Grid Operator does not perform automatic scaling for clusters of Data Grid service pods.

When you enable automatic scaling, you define memory usage thresholds that let Data Grid Operator determine when it needs to create or delete pods. Data Grid Operator monitors statistics for the default cache and, when memory usage reaches the configured thresholds, scales your clusters up or down.

Maximum threshold

This threshold sets an upper boundary for the amount of memory that pods in your cluster can use before scaling up or performing eviction. When Data Grid Operator detects that any node reaches the

maximum amount of memory that you configure, 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.

Minimum threshold

This threshold sets a lower boundary for memory usage across your Data Grid cluster. When Data Grid Operator detects that memory usage falls below the minimum, it shuts down pods.

Default cache only

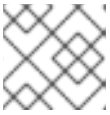
Autoscaling capabilities work with the default cache only. If you plan to add other caches to your cluster, you should not include the **autoscale** field in your **Infinispan** CR. In this case you should use eviction to control the size of the data container on each node.

7.9.1. Configuring automatic scaling

If you create clusters with Cache service pods, you can configure Data Grid Operator to automatically scale clusters.

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 scaling.

2. Configure thresholds for automatic scaling with the following fields:

Field	Description
spec.autoscale.maxMemUsagePercent	Specifies a maximum threshold, as a percentage, for memory usage on each node.
spec.autoscale.maxReplicas	Specifies the maximum number of Cache service pods for the cluster.
spec.autoscale.minMemUsagePercent	Specifies a minimum threshold, as a percentage, for cluster memory usage.
spec.autoscale.minReplicas	Specifies the minimum number of Cache service pods for the cluster.

For example, add the following to your **Infinispan** CR:

```
spec:
  service:
    type: Cache
  autoscale:
    disabled: false
    maxMemUsagePercent: 70
```

```
    maxReplicas: 5
    minMemUsagePercent: 30
    minReplicas: 2
```

3. Apply the changes.

7.10. ADDING LABELS AND ANNOTATIONS TO DATA GRID RESOURCES

Attach key/value labels and annotations to pods and services that Data Grid Operator creates and manages. Labels help you identify relationships between objects to better organize and monitor Data Grid resources. Annotations are arbitrary non-identifying metadata for client applications or deployment and management tooling.

NOTE

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

Procedure

1. Open your **Infinispan** CR for editing.
2. Attach labels and annotations to Data Grid resources in the **metadata.annotations** section.
   - Define values for annotations directly in the **metadata.annotations** section.
   - Define values for labels with the **metadata.labels** field.
3. Apply your **Infinispan** CR.

Custom annotations

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  annotations:
    infinispan.org/targetAnnotations: service-annotation1, service-annotation2
    infinispan.org/podTargetAnnotations: pod-annotation1, pod-annotation2
    service-annotation1: value
    service-annotation2: value
    pod-annotation1: value
    pod-annotation2: value
```

Custom labels

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  annotations:
    infinispan.org/targetLabels: service-label1, service-label2
    infinispan.org/podTargetLabels: pod-label1, pod-label2
  labels:
    service-label1: value
    service-label2: value
```

```
    pod-label1: value
    pod-label2: value
    # The operator does not attach these labels to resources.
    my-label: my-value
    environment: development
```

7.11. ADDING LABELS AND ANNOTATIONS WITH ENVIRONMENT VARIABLES

Set environment variables for Data Grid Operator to add labels and annotations that automatically propagate to all Data Grid pods and services.

Procedure

Add labels and annotations to your Data Grid Operator subscription with the **spec.config.env** field in one of the following ways:

- Use the **oc edit subscription** command.

```
oc edit subscription {subscription_name} -n openshift-operators
```

- Use the Red Hat OpenShift Console.
  1. Navigate to **Operators** > **Installed Operators** > **Subscription**.
  2. Select **Edit Subscription** from the **Actions** menu.

Labels and annotations with environment variables

```
spec:
  config:
    env:
      - name: INFINISPAN_OPERATOR_TARGET_LABELS
        value: |
         {"service-label1":"value",
         service-label1":"value"}
      - name: INFINISPAN_OPERATOR_POD_TARGET_LABELS
        value: |
         {"pod-label1":"value",
         "pod-label2":"value"}
      - name: INFINISPAN_OPERATOR_TARGET_ANNOTATIONS
        value: |
         {"service-annotation1":"value",
         "service-annotation2":"value"}
      - name: INFINISPAN_OPERATOR_POD_TARGET_ANNOTATIONS
        value: |
         {"pod-annotation1":"value",
         "pod-annotation2":"value"}
```

CHAPTER 8. CONFIGURING AUTHENTICATION

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

8.1. DEFAULT CREDENTIALS

Data Grid Operator generates base64-encoded credentials for the following users:

User	Secret name	Description
developer	**infinispan-generated-secret**	Credentials for the default application user.
operator	**infinispan-generated-operator-secret**	Credentials that Data Grid Operator uses to interact with Data Grid resources.

8.2. RETRIEVING CREDENTIALS

Get credentials from authentication secrets to access Data Grid clusters.

Procedure

- Retrieve credentials from authentication secrets.

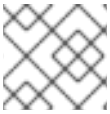
```
oc get secret infinispan-generated-secret
```

Base64-decode credentials.

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

8.3. 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.

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

8.4. 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 **infinispan-generated-operator-secret** secret as follows:

```
oc patch secret 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.

8.5. 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 9. 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).

9.1. 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).

9.2. 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**.

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.

```
spec:
  security:
    endpointEncryption:
      type: Secret
      certSecretName: tls-secret
      clientCert: Validate
      clientCertSecretName: 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.

9.3. 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.

NOTE

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

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.

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

4. Apply the changes.

9.4. PROVIDING CLIENT CERTIFICATES

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

Procedure

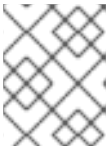
1. Specify the name of the secret that contains the client trust store as the value of the **metadata.name** field.

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.

```
apiVersion: v1
kind: Secret
metadata:
  name: 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 10. CONFIGURING ENCRYPTION

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

10.1. 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:

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

Field	Description
spec.security.endpointEncryption.certServiceName	Specifies the service that provides TLS certificates.
spec.security.endpointEncryption.certSecretName	Specifies a secret with a service certificate and key in PEM format. Defaults to **<cluster_name>-cert-secret**.

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.

10.2. RETRIEVING TLS CERTIFICATES

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

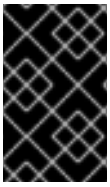
Procedure

- Retrieve **tls.crt** from encryption secrets as follows:

```
oc get secret infinispan-cert-secret -o jsonpath='{.data.tls\.crt}' | base64 --decode > tls.crt
```

10.3. 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.

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

2. Apply the changes.

10.4. 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:

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

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

```
spec:
```

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

3. Apply the changes.

10.4.1. Custom encryption secrets

Custom encryption secrets that add keystores or certificate/key pairs to secure Data Grid connections must contain specific fields.

Keystore secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: Opaque
stringData:
  alias: server
  password: changeme
data:
  keystore.p12:  "MIIKDgIBAzCCCdQGCSqGSIb3DQEHA..."
```

Field	Description
stringData.alias	Specifies an alias for the keystore.
stringData.password	Specifies the keystore password.
data.keystore.p12	Adds a base64-encoded keystore.

Certificate secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: Opaque
data:
  tls.key:  "LS0tLS1CRUdJTiBQUk ..."
  tls.crt: "LS0tLS1CRUdJTiBDRVI ..."
```

Field	Description
data.tls.key	Adds a base64-encoded TLS key.
data.tls.crt	Adds a base64-encoded TLS certificate.

CHAPTER 11. 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.

11.1. 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.

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

2. Apply the changes.

11.2. USER ROLES AND PERMISSIONS

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

Table 11.1. Default roles and permissions

Role	Permissions	Description
admin	ALL	Superuser with all permissions including control of the Cache Manager lifecycle.
deployer	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR, CREATE	Can create and delete Data Grid resources in addition to **application** permissions.
application	ALL_READ, ALL_WRITE, LISTEN, EXEC, MONITOR	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.
observer	ALL_READ, MONITOR	Has read access to Data Grid resources in addition to **monitor** permissions.
monitor	MONITOR	Can view statistics for Data Grid clusters.

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*).

11.3. 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.

```
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.

```
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.

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

11.4. 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.

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

3. Apply the changes.

CHAPTER 12. CONFIGURING NETWORK ACCESS TO DATA GRID

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

12.1. 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: infinispan
```

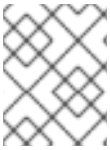
Procedure

- Check that the internal service is available as follows:

```
oc get services
```

12.2. EXPOSING DATA GRID THROUGH A LOADBALANCER SERVICE

Use a **LoadBalancer** 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 **LoadBalancer** 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 the network port where the service is exposed with the **spec.expose.port** field.

```
spec:
  expose:
    type: LoadBalancer
    port: 65535
```

4. Apply the changes.
5. Verify that the **-external** service is available.

```
oc get services | grep external
```

12.3. EXPOSING DATA GRID THROUGH A NODEPORT SERVICE

Use a **NodePort** 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. Configure the port where Data Grid is exposed with the **spec.expose.nodePort** field.

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

4. Apply the changes.
5. Verify that the **-external** service is available.

```
oc get services | grep external
```

12.4. EXPOSING DATA GRID THROUGH A ROUTE

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.

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

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

```
oc get routes
```

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.

Port	Description
80	Encryption is disabled.
443	Encryption is enabled.

12.5. NETWORK SERVICES

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

Service	Port	Protocol	Description
<cluster_name>	**11222**	TCP	Access to Data Grid endpoints within the OpenShift cluster or from an OpenShift **Route**.
<cluster_name>-ping	**8888**	TCP	Cluster discovery for Data Grid pods.
<cluster_name>-external	**11222**	TCP	Access to Data Grid endpoints from a **LoadBalancer** or **NodePort** service.
<cluster_name>-site	**7900**	TCP	JGroups RELAY2 channel for cross-site communication.

CHAPTER 13. SETTING UP CROSS-SITE REPLICATION

Ensure availability with Data Grid Operator by configuring geographically distributed clusters as a unified service.

You can configure clusters to perform cross-site replication with:

- Connections that Data Grid Operator manages.
- Connections that you configure and manage.

NOTE

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

13.1. CROSS-SITE REPLICATION EXPOSE TYPES

You can use a **NodePort** service, a **LoadBalancer** service, or an OpenShift **Route** to handle network traffic for backup operations between Data Grid clusters. Before you start setting up cross-site replication you should determine what expose type is available for your Red Hat OpenShift cluster. In some cases you may require an administrator to provision services before you can configure an expose type.

NodePort

A **NodePort** is a service that accepts network traffic at a static port, in the **30000** to **32767** range, on an IP address that is available externally to the OpenShift cluster.

To use a **NodePort** as the expose type for cross-site replication, an administrator must provision external IP addresses for each OpenShift node. In most cases, an administrator must also configure DNS routing for those external IP addresses.

LoadBalancer

A **LoadBalancer** is a service that directs network traffic to the correct node in the OpenShift cluster.

Whether you can use a **LoadBalancer** as the expose type for cross-site replication depends on the host platform. AWS supports network load balancers (NLB) while some other cloud platforms do not. To use a **LoadBalancer** service, an administrator must first create an ingress controller backed by an NLB.

Route

An OpenShift **Route** allows Data Grid clusters to connect with each other through a public secure URL.

Data Grid uses TLS with the SNI header to send backup requests between clusters through an OpenShift **Route**. To do this you must add a keystore with TLS certificates so that Data Grid can encrypt network traffic for cross-site replication.

When you specify **Route** as the expose type for cross-site replication, Data Grid Operator creates a route with TLS passthrough encryption for each Data Grid cluster that it manages. You can specify a hostname for the **Route** but you cannot specify a **Route** that you have already created.

Additional resources

- Configuring ingress cluster traffic overview

13.2. MANAGED CROSS-SITE REPLICATION

Data Grid Operator can discover Data Grid clusters running in different data centers to form global clusters.

When you configure managed cross-site connections, Data Grid Operator creates router pods in each Data Grid cluster. Data Grid pods use the **<cluster_name>-site** service to connect to these router pods and send backup requests.

Router pods maintain a record of all pod IP addresses and parse RELAY message headers to forward backup requests to the correct Data Grid cluster. If a router pod crashes then all Data Grid pods start using any other available router pod until OpenShift restores it.

IMPORTANT

To manage cross-site connections, Data Grid Operator uses the Kubernetes API. Each OpenShift cluster must have network access to the remote Kubernetes API and a service account token for each backup cluster.

NOTE

Data Grid clusters do not start running until Data Grid Operator discovers all backup locations that you configure.

13.2.1. Creating service account tokens for managed cross-site connections

Generate service account tokens on OpenShift clusters that allow Data Grid Operator to automatically discover Data Grid clusters and manage cross-site connections.

Prerequisites

- Ensure all OpenShift clusters have access to the Kubernetes API.
  Data Grid Operator uses this API to manage cross-site connections.

NOTE

Data Grid Operator does not modify remote Data Grid clusters. The service account tokens provide read only access through the Kubernetes API.

Procedure

1. Log in to an OpenShift cluster.
2. Create a service account.
   For example, create a service account at **LON**:

```
oc create sa lon
serviceaccount/lon created
```

3. Add the view role to the service account with the following command:

```
oc policy add-role-to-user view system:serviceaccount:<namespace>:lon
```

4. If you use a **NodePort** service to expose Data Grid clusters on the network, you must also add the **cluster-reader** role to the service account:

```
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.
6. Exchange service account tokens on each OpenShift cluster.

Additional resources

- Using service accounts in applications

13.2.2. 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:

```
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:

```
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.

13.2.3. Configuring managed cross-site connections

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

Prerequisites

- Determine a suitable expose type for cross-site replication.
  If you use an OpenShift **Route** you must add a keystore with TLS certificates and secure cross-site connections.
- Create and exchange Red Hat OpenShift service account tokens for each Data Grid 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. Configure the expose type for cross-site replication.
   a. Set the value of the **spec.service.sites.local.expose.type** field to one of the following:
      - **NodePort**
      - **LoadBalancer**
      - **Route**
   b. Optionally specify a port or custom hostname with the following fields:
      - **spec.service.sites.local.expose.nodePort** if you use a **NodePort** service.
      - **spec.service.sites.local.expose.port** if you use a **LoadBalancer** service.
      - **spec.service.sites.local.expose.routeHostName** if you use an OpenShift **Route**.
4. Specify the number of pods that can send RELAY messages with the **service.sites.local.maxRelayNodes** field.

   TIP

   Configure all pods in your cluster to send **RELAY** messages for better performance. If all pods send backup requests directly, then no pods need to forward backup requests.

5. Provide the name, URL, and secret for each Data Grid cluster that acts as a backup location with **spec.service.sites.locations**.
6. 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

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
spec:
  replicas: 3
  service:
    type: DataGrid
    sites:
```

```
      local:
        name: LON
        expose:
          type: LoadBalancer
          port: 65535
        maxRelayNodes: 1
      locations:
        - name: NYC
          clusterName: <nyc_cluster_name>
          namespace: <nyc_cluster_namespace>
          url: openshift://api.rhdg-nyc.openshift-aws.myhost.com:6443
          secretName: nyc-token
  logging:
    categories:
      org.jgroups.protocols.TCP: error
      org.jgroups.protocols.relay.RELAY2: error
```

- NYC

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: nyc-cluster
spec:
  replicas: 2
  service:
    type: DataGrid
    sites:
      local:
        name: NYC
        expose:
          type: LoadBalancer
          port: 65535
        maxRelayNodes: 1
      locations:
        - name: LON
          clusterName: infinispan
          namespace: rhdg-namespace
          url: openshift://api.rhdg-lon.openshift-aws.myhost.com:6443
          secretName: lon-token
  logging:
    categories:
      org.jgroups.protocols.TCP: error
      org.jgroups.protocols.relay.RELAY2: error
```

IMPORTANT

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 uses container storage.

```
spec:
  logging:
    categories:
      org.jgroups.protocols.TCP: error
      org.jgroups.protocols.relay.RELAY2: error
```

7. Configure your **Infinispan** CRs with any other Data Grid service resources and then apply the changes.
8. 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.

Additional resources

- Data Grid guide to cross-site replication

13.3. MANUALLY CONFIGURING CROSS-SITE CONNECTIONS

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.

Prerequisites

- Determine a suitable expose type for cross-site replication.
  If you use an OpenShift **Route** you must add a keystore with TLS certificates and secure cross-site connections.
- Ensure you have the correct host names and ports for each Data Grid cluster and each **<cluster-name>-site** service.
  Manually connecting Data Grid clusters to form cross-site views requires predictable network locations for Data Grid services, which means you need to know the network locations before they are created.

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. Configure the expose type for cross-site replication.
   a. Set the value of the **spec.service.sites.local.expose.type** field to one of the following:
      - **NodePort**
      - **LoadBalancer**
      - **Route**
   b. Optionally specify a port or custom hostname with the following fields:
      - **spec.service.sites.local.expose.nodePort** if you use a **NodePort** service.
      - **spec.service.sites.local.expose.port** if you use a **LoadBalancer** service.
      - **spec.service.sites.local.expose.routeHostName** if you use an OpenShift **Route**.
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

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
spec:
  replicas: 3
  service:
    type: DataGrid
    sites:
      local:
        name: LON
        expose:
          type: LoadBalancer
          port: 65535
        maxRelayNodes: 1
      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

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

```
    local:
      name: NYC
      expose:
        type: LoadBalancer
        port: 65535
      maxRelayNodes: 1
    locations:
      - name: LON
        url: infinispan+xsite://infinispan-lon.myhost.com
logging:
  categories:
    org.jgroups.protocols.TCP: error
    org.jgroups.protocols.relay.RELAY2: error
```

IMPORTANT

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 uses container storage.

```
spec:
  logging:
    categories:
      org.jgroups.protocols.TCP: error
      org.jgroups.protocols.relay.RELAY2: error
```

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.

Additional resources

- Data Grid guide to cross-site replication

13.4. RESOURCES FOR CONFIGURING CROSS-SITE REPLICATION

The following tables provides fields and descriptions for cross-site resources.

Table 13.1. service.type

Field	Description
service.type: DataGrid	Data Grid supports cross-site replication with Data Grid service clusters only.

Table 13.2. service.sites.local

Field	Description
service.sites.local.name	Names the local site where a Data Grid cluster runs.
service.sites.local.expose.type	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**, **LoadBalancer**, or **Route**.
service.sites.local.expose.nodePort	Specifies a static port within the default range of **30000** to **32767** if you expose Data Grid through a **NodePort** service. If you do not specify a port, the platform selects an available one.
service.sites.local.expose.port	Specifies the network port for the service if you expose Data Grid through a **LoadBalancer** service. The default port is **7900**.
service.sites.local.expose.routeHostName	Specifies a custom hostname if you expose Data Grid through an OpenShift **Route**. If you do not set a value then OpenShift generates a hostname.
service.sites.local.maxRelayNodes	Specifies the maximum number of pods that can send RELAY messages for cross-site replication. The default value is **1**.

Table 13.3. service.sites.locations

Field	Description
service.sites.locations	Provides connection information for all backup locations.
service.sites.locations.name	Specifies a backup location that matches **.spec.service.sites.local.name**.

Field	Description
service.sites.locations.url	Specifies the URL of the Kubernetes API for managed connections or a static URL for manual connections. Use **openshift:**// to specify the URL of the Kubernetes API for an OpenShift cluster. Note that the **openshift:**// URL must present a valid, CA-signed certificate. You cannot use self-signed certificates. Use the **infinispan+xsite://<hostname>:<port>** format for static hostnames and ports. The default port is **7900**.
service.sites.locations.secretName	Specifies the secret that contains the service account token for the backup site.
service.sites.locations.clusterName	Specifies the cluster name at the backup location if it is different to the cluster name at the local site.
service.sites.locations.namespace	Specifies the namespace of the Data Grid cluster at the backup location if it does not match the namespace at the local site.

Managed cross-site connections

```
spec:
  service:
    type: DataGrid
    sites:
      local:
        name: LON
        expose:
          type: LoadBalancer
        maxRelayNodes: 1
      locations:
      - name: NYC
        clusterName: <nyc_cluster_name>
        namespace: <nyc_cluster_namespace>
        url: openshift://api.site-b.devcluster.openshift.com:6443
        secretName: nyc-token
```

Manual cross-site connections

```
spec:
  service:
    type: DataGrid
    sites:
      local:
```

```
    name: LON
    expose:
      type: LoadBalancer
      port: 65535
    maxRelayNodes: 1
  locations:
  - name: NYC
    url: infinispan+xsite://infinispan-nyc.myhost.com:7900
```

13.5. SECURING CROSS-SITE CONNECTIONS

Add keystores and trust stores so that Data Grid clusters can secure cross-site replication traffic.

You must add a keystore to use an OpenShift **Route** as the expose type for cross-site replication. Securing cross-site connections is optional if you use a **NodePort** or **LoadBalancer** as the expose type.

Prerequisites

- Have a PKCS12 keystore that Data Grid can use to encrypt and decrypt RELAY messages.
  You must provide a keystore for relay pods and router pods to secure cross-site connections.
  The keystore can be the same for relay pods and router pods or you can provide separate keystores for each.
  You can also use the same keystore for each Data Grid cluster or a unique keystore for each cluster.

- Optionally have a trust store that contains part of the certificate chain or root CA certificate that verifies public certificates for Data Grid relay pods and router pods.
  By default, Data Grid uses the Java trust store to verify public certificates.

Procedure

1. Create cross-site encryption secrets.

   a. Create keystore secrets.

   b. Create trust store secrets if you do not want to use the default Java trust store.

2. Modify the **Infinispan** CR for each Data Grid cluster to specify the secret name for the **encryption.transportKeyStore.secretName** and **encryption.routerKeyStore.secretName** fields.

3. Configure any other fields to encrypt RELAY messages as required and then apply the changes.

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
spec:
  replicas: 2
  expose:
    type: LoadBalancer
  service:
    type: DataGrid
    sites:
      local:
```

```
      name: SiteA
      # ...
      encryption:
        protocol: TLSv1.3
        transportKeyStore:
          secretName: transport-tls-secret
          alias: transport
          filename: keystore.p12
        routerKeyStore:
          secretName: router-tls-secret
          alias: router
          filename: keystore.p12
        trustStore:
          secretName: truststore-tls-secret
          filename: truststore.p12
    locations:
      # ...
```

13.5.1. Resources for configuring cross-site encryption

The following tables provides fields and descriptions for encrypting cross-site connections.

Table 13.4. service.type.sites.local.encryption

Field	Description
service.type.sites.local.encryption.protocol	Specifies the TLS protocol to use for cross-site connections. The default value is **TLSv1.2** but you can set **TLSv1.3** if required.
service.type.sites.local.encryption.transportKeyStore	Configures a keystore secret for relay pods.
service.type.sites.local.encryption.routerKeyStore	Configures a keystore secret for router pods.
service.type.sites.local.encryption.trustStore	Configures an optional trust store secret for relay pods and router pods.

Table 13.5. service.type.sites.local.encryption.transportKeyStore

Field	Description
secretName	Specifies the secret that contains a keystore that relay pods can use to encrypt and decrypt RELAY messages. This field is required.
alias	Optionally specifies the alias of the certificate in the keystore. The default value is **transport**.

Field	Description
filename	Optionally specifies the filename of the keystore. The default value is **keystore.p12**.

Table 13.6. service.type.sites.local.encryption.routerKeyStore

Field	Description
secretName	Specifies the secret that contains a keystore that router pods can use to encrypt and decrypt RELAY messages. This field is required.
alias	Optionally specifies the alias of the certificate in the keystore. The default value is **router**.
filename	Optionally specifies the filename of the keystore. The default value is **keystore.p12**.

Table 13.7. service.type.sites.local.encryption.trustStore

Field	Description
secretName	Optionally specifies the secret that contains a trust store to verify public certificates for relay pods and router pods. The default value is **<cluster-name>-truststore-site-tls-secret**.
filename	Optionally specifies the filename of the trust store. The default value is **truststore.p12**.

13.5.2. Cross-site encryption secrets

Cross-site replication encryption secrets add keystores and optional trust stores for securing cross-site connections.

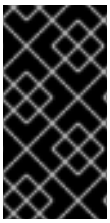
Cross-site encryption secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: Opaque
stringData:
  password: changeme
  type: pkcs12
data:
  <file-name>: "MIIKDgIBAzCCCdQGCSqGSIb3DQEHA..."
```

Field	Description
stringData.password	Specifies the password for the keystore or trust store.
stringData.type	Optionally specifies the keystore or trust store type. The default value is **pkcs12**.
data.<file-name>	Adds a base64-encoded keystore or trust store.

13.6. CONFIGURING SITES IN THE SAME OPENSHIFT CLUSTER

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



IMPORTANT

Using **ClusterIP** as the expose type for cross-site replication is intended for demonstration purposes only. It would be appropriate to use this expose type only to perform a temporary proof-of-concept deployment on a laptop or something of that nature.

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**.

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-clustera
spec:
  replicas: 1
  expose:
    type: LoadBalancer
  service:
    type: DataGrid
    sites:
      local:
        name: SiteA
        expose:
          type: ClusterIP
```

```
    maxRelayNodes: 1
  locations:
    - name: SiteB
      clusterName: example-clusterb
      namespace: cluster-namespace
```

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.

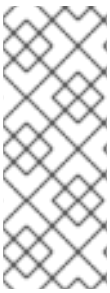
   a. Retrieve the **Infinispan** CR.

   ```
   oc get infinispan -o yaml
   ```

   b. Check for the **type: CrossSiteViewFormed** condition.

CHAPTER 14. 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.

14.1. 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 Operator detects an **Infinispan** CR with the monitoring annotation set to **true**, which is the default, Data Grid Operator does the following:

- Creates a **ServiceMonitor** named **<cluster_name>-monitor**.
- Adds the **infinispan.org/monitoring: 'true'** annotation to your **Infinispan** CR metadata, if the value is not already explicitly set:

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: 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

14.1.1. 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.

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

2. Apply the changes.

14.2. 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.

14.3. 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.

```
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
```

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

```
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
      url: 'https://thanos-querier.openshift-monitoring.svc.cluster.local:9091'
```

3. Apply the **GrafanaDataSource**.

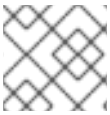
```
oc apply -f grafana-datasource.yaml
```

Next steps

Enable Grafana dashboards with the Data Grid Operator configuration properties.

14.4. 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.

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: infinispan-operator-config
data:
  grafana.dashboard.namespace: 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.

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

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

```
oc get routes grafana-route -o jsonpath=https://"{.spec.host}"
```

CHAPTER 15. GUARANTEEING AVAILABILITY WITH ANTI-AFFINITY

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

15.1. 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

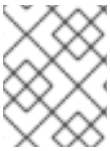
Data Grid replica pods are scheduled on different OpenShift nodes.

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

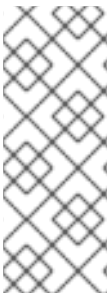
Provides tolerance of **x** node failures for the following types of cache:

- Replicated: **x = spec.replicas - 1**
- Distributed: **x = 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 = spec.replicas - 1**
- Distributed: **x = 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.

15.2. 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.

15.2.1. Anti-affinity strategy configurations

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

Topology keys	Description
topologyKey: "topology.kubernetes.io/zone"	Schedules Data Grid replica pods across multiple zones.
topologyKey: "kubernetes.io/hostname"	Schedules Data Grid replica pods on different OpenShift nodes.

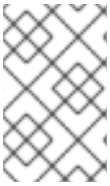
Schedule pods on different OpenShift nodes
The following is the anti-affinity strategy that Data Grid Operator uses if you do not configure the **spec.affinity** field in your **Infinispan** CR:

```
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:

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

```
        clusterName: <cluster_name>
        infinispan_cr: <cluster_name>
      topologyKey: "topology.kubernetes.io/hostname"
```



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 16. 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.

16.1. 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.

Cache CRs

- **Cache** CRs apply to Data Grid service pods only.
- Each **Cache** CR corresponds to a single cache on the Data Grid cluster.

16.2. CREATING CACHES WITH THE CACHE CR

Complete the following steps to create caches on Data Grid service clusters using valid configuration in XML or YAML format.

Procedure

1. Create a **Cache** CR with a unique value in the **metadata.name** field.
2. Specify the target Data Grid cluster with the **spec.clusterName** field.
3. Name your cache with the **spec.name** field.

NOTE

The **name** attribute in the cache configuration does not take effect. If you do not specify a name with the **spec.name** field then the cache uses the value of the **metadata.name** field.

4. Add a cache configuration with the **spec.template** field.
5. Apply the **Cache** CR, for example:

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

Cache CR examples

XML

```
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: mycachedefinition
spec:
  clusterName: infinispan
  name: myXMLcache
  template: <distributed-cache mode="SYNC" statistics="true"><encoding media-type="application/x-protostream"/><persistence><file-store/></persistence></distributed-cache>
```

YAML

```
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: mycachedefinition
spec:
  clusterName: infinispan
  name: myYAMLcache
  template: |
  distributedCache:
    mode: "SYNC"
    owners: "2"
    statistics: "true"
    encoding:
      mediaType: "application/x-protostream"
    persistence:
      fileStore: ~
```

16.3. ADDING PERSISTENT CACHE STORES

You can add persistent cache stores to Data Grid service pods to save data to the persistent volume.

Data Grid creates a Single File cache store, **.dat** file, in the **/opt/infinispan/server/data** directory.

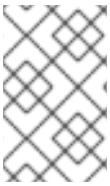
Procedure

- 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>
```

16.4. ADDING CACHES TO CACHE SERVICE PODS

Cache service pods 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 pods instead of Cache service pods.

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:

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

16.4.1. Default cache configuration

This topic describes default cache configuration for Cache service pods.

```
<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 pods.
- Evict cache entries to control the size of the data container. You can enable automatic scaling so that Data Grid Operator adds pods 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.

CHAPTER 17. 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.

17.1. 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: 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.

17.2. 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
```

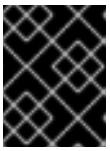
- Create a Data Grid cache configuration.

```
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
```

Procedure

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/mycache.xml
put --cache=mycache hello world
put --cache=mycache 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
```

17.3. RUNNING BATCH OPERATIONS WITH CONFIGMAPS

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.

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

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

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.

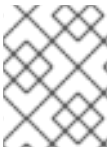
17.4. BATCH STATUS MESSAGES

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

Phase	Description
Succeeded	All batch operations have completed successfully.
Initializing	Batch operations are queued and resources are initializing.
Initialized	Batch operations are ready to start.
Running	Batch operations are in progress.
Failed	One or more batch operations were not successful.

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.

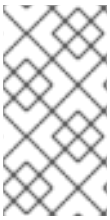


NOTE

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

17.5. 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.

17.5.1. Caches

- Create multiple caches from configuration files.

```
echo "creating caches..."
create cache sessions --file=/etc/batch/infinispan-prod-sessions.xml
create cache tokens --file=/etc/batch/infinispan-prod-tokens.xml
create cache people --file=/etc/batch/infinispan-prod-people.xml
create cache books --file=/etc/batch/infinispan-prod-books.xml
create cache authors --file=/etc/batch/infinispan-prod-authors.xml
echo "list caches in the cluster"
ls caches
```

- Create a template from a file and then create caches from the template.

```
echo "creating caches..."
create cache mytemplate --file=/etc/batch/mycache.xml
create cache sessions --template=mytemplate
create cache tokens --template=mytemplate
echo "list caches in the cluster"
ls caches
```

17.5.2. 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
echo "list counters in the cluster"
ls counters
```

17.5.3. 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
echo "list Protobuf schema"
ls schemas
```

17.5.4. 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"
ls tasks
```

Additional resources

- Data Grid CLI Guide

CHAPTER 18. BACKING UP AND RESTORING DATA GRID CLUSTERS

Data Grid Operator lets you back up and restore Data Grid cluster state for disaster recovery and to migrate Data Grid resources between clusters.

18.1. BACKUP AND RESTORE CRS

Backup and **Restore** CRs save in-memory data at runtime so you can easily recreate Data Grid clusters.

Applying a **Backup** or **Restore** CR creates a new pod that joins the Data Grid cluster as a zero-capacity member, which means it does not require cluster rebalancing or state transfer to join.

For backup operations, the pod iterates over cache entries and other resources and creates an archive, a **.zip** file, in the **/opt/infinispan/backups** directory on the persistent volume (PV).

NOTE

Performing backups does not significantly impact performance because the other pods in the Data Grid cluster only need to respond to the backup pod as it iterates over cache entries.

For restore operations, the pod retrieves Data Grid resources from the archive on the PV and applies them to the Data Grid cluster.

When either the backup or restore operation completes, the pod leaves the cluster and is terminated.

Reconciliation

Data Grid Operator does not reconcile **Backup** and **Restore** CRs which mean that backup and restore operations are "one-time" events.

Modifying an existing **Backup** or **Restore** CR instance does not perform an operation or have any effect. If you want to update **.spec** fields, you must create a new instance of the **Backup** or **Restore** CR.

18.2. BACKING UP DATA GRID CLUSTERS

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

Prerequisites

- Create an **Infinispan** CR with **spec.service.type: DataGrid**.
- Ensure there are no active client connections to the Data Grid cluster.
  Data Grid backups do not provide snapshot isolation and data modifications are not written to the archive after the cache is backed up.
  To archive the exact state of the cluster, you should always disconnect any clients before you back it up.

Procedure

1. Name the **Backup** CR with the **metadata.name** field.
2. Specify the Data Grid cluster to backup with the **spec.cluster** field.

3. Configure the persistent volume claim (PVC) that adds the backup archive to the persistent volume (PV) with the **spec.volume.storage** and **spec.volume.storage.storageClassName** fields.

```
apiVersion: infinispan.org/v2alpha1
kind: Backup
metadata:
  name: my-backup
spec:
  cluster: source-cluster
  volume:
    storage: 1Gi
    storageClassName: my-storage-class
```

4. Optionally include **spec.resources** fields to specify which Data Grid resources you want to back up.
   If you do not include any **spec.resources** fields, the **Backup** CR creates an archive that contains all Data Grid resources. If you do specify **spec.resources** fields, the **Backup** CR creates an archive that contains those resources only.

```
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 as in the following example:

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

5. Apply your **Backup** CR.

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

Verification

1. Check that the **status.phase** field has a status of **Succeeded** in the **Backup** CR and that Data Grid logs have the following message:

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

2. Run the following command to check that the backup is successfully created:

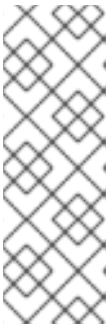
```
oc describe Backup my-backup
```

18.3. RESTORING DATA GRID CLUSTERS

Restore Data Grid cluster state from a backup archive.

Prerequisites

- Create a **Backup** CR on a source cluster.
- Create a target Data Grid cluster of Data Grid service pods.



NOTE

If you restore an existing cache, the operation overwrites the data in the cache but not the cache configuration.

For example, you back up a distributed cache named **mycache** on the source cluster. You then restore **mycache** on a target cluster where it already exists as a replicated cache. In this case, the data from the source cluster is restored and **mycache** continues to have a replicated configuration on the target cluster.

- Ensure there are no active client connections to the target Data Grid cluster you want to restore. Cache entries that you restore from a backup can overwrite more recent cache entries. For example, a client performs a **cache.put(k=2)** operation and you then restore a backup that contains **k=1**.

Procedure

1. Name the **Restore** CR with the **metadata.name** field.
2. Specify a **Backup** CR to use with the **spec.backup** field.
3. Specify the Data Grid cluster to restore with the **spec.cluster** field.

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

4. Optionally add the **spec.resources** field to restore specific resources only.

```
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
```

5. Apply your **Restore** CR.

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

Verification

- Check that the **status.phase** field has a status of **Succeeded** in the **Restore** CR and that Data Grid logs have the following message:

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

You should then open the Data Grid Console or establish a CLI connection to verify data and Data Grid resources are restored as expected.

18.4. BACKUP AND RESTORE STATUS

Backup and **Restore** CRs include a **status.phase** field that provides the status for each phase of the operation.

Status	Description
Initializing	The system has accepted the request and the controller is preparing the underlying resources to create the pod.
Initialized	The controller has prepared all underlying resources successfully.
Running	The pod is created and the operation is in progress on the Data Grid cluster.
Succeeded	The operation has completed successfully on the Data Grid cluster and the pod is terminated.

Status	Description
Failed	The operation did not successfully complete and the pod is terminated.
Unknown	The controller cannot obtain the status of the pod or determine the state of the operation. This condition typically indicates a temporary communication error with the pod.

18.4.1. Handling failed backup and restore operations

If the **status.phase** field of the **Backup** or **Restore** CR is **Failed**, you should examine pod logs to determine the root cause before you attempt the operation again.

Procedure

1. Examine the logs for the pod that performed the failed operation.
   Pods are terminated but remain available until you delete the **Backup** or **Restore** CR.

   ```
   oc logs <backup|restore_pod_name>
   ```

2. Resolve any error conditions or other causes of failure as indicated by the pod logs.
3. Create a new instance of the **Backup** or **Restore** CR and attempt the operation again.

CHAPTER 19. DEPLOYING CUSTOM CODE TO DATA GRID

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

Before you can deploy custom code to Data Grid clusters, you need to make it available. To do this you can copy artifacts from a persistent volume (PV), download artifacts from an HTTP or FTP server, or use both methods.

19.1. COPYING CODE ARTIFACTS TO DATA GRID CLUSTERS

Adding your artifacts to a persistent volume (PV) and then copy them to Data Grid pods.

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 pods in a cluster being scheduled on the same OpenShift node.

Procedure

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

```
oc project rhdg-namespace
```

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

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

3. Apply your PVC.

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

4. Create a pod that mounts the PVC, for example:

```
apiVersion: v1
kind: Pod
metadata:
  name: datagrid-libs-pod
spec:
  securityContext:
    fsGroup: 2000
  volumes:
    - name: lib-pv-storage
      persistentVolumeClaim:
        claimName: datagrid-libs
  containers:
    - name: lib-pv-container
      image: registry.redhat.io/datagrid/datagrid-8-rhel8:8.3
      volumeMounts:
        - mountPath: /tmp/libs
          name: lib-pv-storage
```

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:

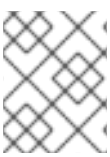
```
oc cp --no-preserve=true libs datagrid-libs-pod:/tmp/
```

7. Delete the pod.

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

Specify the persistent volume with **spec.dependencies.volumeClaimName** in your **Infinispan** CR and then apply the changes.

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: 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.

Additional resources

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

19.2. DOWNLOADING CODE ARTIFACTS

Add your artifacts to an HTTP or FTP server so that Data Grid Operator downloads them to the **{lib_path}** directory on each Data Grid node.

When downloading files, Data Grid Operator can automatically detect the file type. Data Grid Operator also extracts archived files, such as **zip** or **tgz**, to the filesystem after the download completes.

NOTE

Each time Data Grid Operator creates a Data Grid node it downloads the artifacts to the node. The download also occurs when Data Grid Operator recreates pods after terminating them.

Prerequisites

- Host your code artifacts on an HTTP or FTP server.

Procedure

1. Add the **spec.dependencies.artifacts** field to your **Infinispan** CR.
   a. Specify the location of the file to download via **HTTP** or **FTP** as the value of the **spec.dependencies.artifacts.url** field.
   b. Optionally specify a checksum to verify the integrity of the download with the **spec.dependencies.artifacts.hash** field.
   The **hash** field requires a value is in the format of **<algorithm>:<checksum>** where **<algorithm>** is **sha1|sha224|sha256|sha384|sha512|md5**.
   c. Set the file type, if necessary, with the **spec.dependencies.artifacts.type** field.
   You should explicitly set the file type if it is not included in the URL or if the file type is actually different to the extension in the URL.

NOTE

If you set **type: file**, Data Grid Operator downloads the file as-is without extracting it to the filesystem.

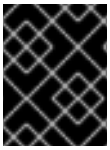
```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan
```

```
spec:
  replicas: 2
  dependencies:
    artifacts:
      - url: http://example.com:8080/path
        hash:
sha256:596408848b56b5a23096baa110cd8b633c9a9aef2edd6b38943ade5b4edcd686
        type: zip
  service:
    type: DataGrid
```

2. Apply the changes.

CHAPTER 20. SENDING CLOUD EVENTS FROM DATA GRID CLUSTERS

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



IMPORTANT

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

20.1. TECHNOLOGY PREVIEW FEATURES

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 .

20.2. 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:

```
{
   "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"
   }
}
```

Field	Description
type	Prefixes events for Data Grid cache entries with **org.infinispan.entry**.
data	Entry value.
subject	Entry key, converted to string.
id	Generated identifier for the event.

20.3. 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.

```
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 21. ESTABLISHING REMOTE CLIENT CONNECTIONS

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

21.1. CLIENT CONNECTION DETAILS

Client connections to Data Grid require the following information:

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

Hostnames

The hostname you use depends on whether clients are running on the same OpenShift cluster as Data Grid.

Client applications running on the same OpenShift cluster use the internal service name for the Data Grid cluster.

```
metadata:
  name: infinispan
```

Client applications running on a different OpenShift, or outside OpenShift, use a hostname that depends on how Data Grid is exposed on the network.

A **LoadBalancer** service uses the URL for the load balancer. A **NodePort** service uses the node hostname. An Red Hat OpenShift **Route** uses either a custom hostname that you define or a hostname that the system generates.

Ports

Client connections on OpenShift and a through **LoadBalancer** service use port **11222**.

NodePort 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

21.2. CONNECTING TO DATA GRID CLUSTERS WITH REMOTE SHELLS

Start a remote shell session to Data Grid clusters and use the command line interface (CLI) to work with Data Grid resources and perform administrative operations.

Prerequisites

- Have **kubectl-infinispan** on your **PATH**.
- Have valid Data Grid credentials.

Procedure

1. Run the **infinispan shell** command to connect to your Data Grid cluster.

```
oc infinispan shell <cluster_name>
```

NOTE

If you have access to authentication secrets and there is only one Data Grid user the **kubectl-infinispan** plugin automatically detects your credentials and authenticates to Data Grid. If your deployment has multiple Data Grid credentials, specify a user with the **--username** argument and enter the corresponding password when prompted.

2. Perform CLI operations as required.

TIP

Press the tab key or use the **--help** argument to view available options and help text.

3. Use the **quit** command to end the remote shell session.

Additional resources

- Using the Data Grid Command Line Interface

21.3. ACCESSING DATA GRID CONSOLE

Access the console to create caches, perform adminstrative 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 **LoadBalancer** service or create a **Route**.

Procedure

- Access the console from any browser at **$HOSTNAME:$PORT**.
  Replace **$HOSTNAME:$PORT** with the network location where Data Grid is available.

21.4. 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

The Hot Rod protocol includes a mechanism that provides clients with an up-to-date view of the cache topology. Client intelligence improves performance by reducing the number of network hops for read and write operations.

Clients running in the same OpenShift cluster can access internal IP addresses for Data Grid pods so you can use any client intelligence.

HASH_DISTRIBUTION_AWARE is the default intelligence mechanism and enables clients to route requests to primary owners, which provides the best performance for Hot Rod clients.

BASIC intelligence

Hot Rod clients must use **BASIC** intelligence in the following situations:

- Connecting to Data Grid through a **LoadBalancer**, **NodePort**, or OpenShift **Route**.
- Failing over to a different OpenShift cluster when using cross-site replication.

OpenShift cluster administrators can define network policies that restrict traffic to Data Grid. In some cases network isolation policies can require you to use **BASIC** intelligence even when clients are running in the same OpenShift cluster but a different namespace.

21.4.1. Hot Rod client configuration API

You can programmatically configure Hot Rod client connections with the **ConfigurationBuilder** interface.

NOTE

Replace **$SERVICE_HOSTNAME** in the following examples with the internal service name of your Data Grid cluster.

```
metadata:
  name: infinispan
```

On OpenShift

ConfigurationBuilder

```
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("$HOSTNAME")
           .port(ConfigurationProperties.DEFAULT_HOTROD_PORT)
          .security().authentication()
           .username("username")
           .password("changeme")
           .realm("default")
           .saslQop(SaslQop.AUTH)
```

```
            .saslMechanism("SCRAM-SHA-512")
          .ssl()
            .sniHostName("$SERVICE_HOSTNAME")
            .trustStoreFileName("/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt")
            .trustStoreType("pem");
```

hotrod-client.properties

```
# Connection
infinispan.client.hotrod.server_list=$HOSTNAME:$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

ConfigurationBuilder

```
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("$HOSTNAME")
            .port("$PORT")
          .security().authentication()
            .username("username")
            .password("changeme")
            .realm("default")
            .saslQop(SaslQop.AUTH)
            .saslMechanism("SCRAM-SHA-512")
          .ssl()
            .sniHostName("$SERVICE_HOSTNAME")
            //Create a client trust store with tls.crt from your project.
            .trustStoreFileName("/path/to/truststore.pkcs12")
            .trustStorePassword("trust_store_password")
            .trustStoreType("PCKS12");
    builder.clientIntelligence(ClientIntelligence.BASIC);
```

hotrod-client.properties

```
# Connection
infinispan.client.hotrod.server_list=$HOSTNAME:$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
```

21.4.2. Configuring Hot Rod clients for certificate authentication

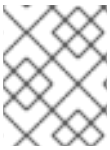
If you enable client certificate authentication, clients must present valid certificates when negotiating connections with Data Grid.

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:

```
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");
```

21.4.3. 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**:

```
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")
           .configuration("<infinispan><cache-container><distributed-cache name=\"another-cache\">
<encoding media-type=\"application/x-protostream\"/></distributed-cache></cache-container>
</infinispan>");
    try (RemoteCacheManager cacheManager = new RemoteCacheManager(builder.build())) {
    // Get a remote cache that does not exist.
    // Rather than return null, create the cache from a template.
    RemoteCache<String, String> cache = cacheManager.getCache("my-cache");
    // Store a value.
    cache.put("hello", "world");
    // Retrieve the value and print it.
    System.out.printf("key = %s\n", cache.get("hello"));
```

This example shows how to create a cache named CacheWithXMLConfiguration using the **XMLStringConfiguration()** method to pass the cache configuration as XML:

```
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_uri=file:/path/to/configuration.xml
```

21.5. 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 **LoadBalancer** service or create a **Route**.

Procedure

- Access the REST API with any HTTP client at **$HOSTNAME:$PORT/rest/v2**.
  Replace **$HOSTNAME:$PORT** with the network location where Data Grid listens for client connections.

Additional resources

- Data Grid REST API