Red Hat Data Grid 8.1

Running Data Grid on OpenShift

Configure and run Data Grid services on OpenShift

Last Updated: 2021-05-20

Red Hat Data Grid 8.1 Running Data Grid on OpenShift

Configure and run Data Grid services on OpenShift

Legal Notice

Abstract

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

Table of Contents

CHAPTER 1. 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.

1.1. DATA GRID DOCUMENTATION

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

- Data Grid 8.1 Documentation
- Data Grid 8.1 Component Details
- Supported Configurations for Data Grid 8.1
- Data Grid 8 Feature Support
- Data Grid Deprecated Features and Functionality

1.2. 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.

1.3. MAKING OPEN SOURCE MORE INCLUSIVE

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

CHAPTER 2. INSTALLING DATA GRID OPERATOR

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

2.1. 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 8.1.x.

   Approval Strategies

   Automatically install updates from the 8.1.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.

2.2. INSTALLING DATA GRID OPERATOR FROM THE COMMAND LINE

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

Prerequisites

- Have an **oc** client.

Procedure

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

```
$ oc new-project ${INSTALL_NAMESPACE} 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: 8.1.x
 installPlanApproval: Automatic 1
 name: datagrid
```

```
  source: redhat-operators
  sourceNamespace: openshift-marketplace
EOF
```

1 Specify **Manual** if you want to manually approve updates from the 8.1.x channel.

4. Verify the installation.

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

CHAPTER 3. GETTING STARTED WITH DATA GRID OPERATOR

Data Grid Operator lets you create, configure, and manage Data Grid clusters.

Prerequisites

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

3.1. INFINISPAN CUSTOM RESOURCE (CR)

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

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

The minimal **Infinispan** CR is as follows:

```
apiVersion: infinispan.org/v1 1
kind: Infinispan 2
metadata:
  name: example-infinispan 3
spec:
  replicas: 2 4
```

1. Declares the **Infinispan** API version.
2. Declares the **Infinispan** CR.
3. Names the Data Grid cluster.
4. Specifies the number of nodes in the Data Grid cluster.

3.2. CREATING DATA GRID CLUSTERS

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

Procedure

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

```
$ cat > cr_minimal.yaml<<EOF
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
```

```
spec:
  replicas: 2
EOF
```

2. Apply your **Infinispan** CR.

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

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

```
$ oc get pods -w

NAME                    READY  STATUS            RESTARTS   AGE
example-infinispan-1      0/1    ContainerCreating  0          4s
example-infinispan-2      0/1    ContainerCreating  0          4s
example-infinispan-3      0/1    ContainerCreating  0          5s
infinispan-operator-0     1/1    Running            0          3m
example-infinispan-3      1/1    Running            0          8s
example-infinispan-2      1/1    Running            0          8s
example-infinispan-1      1/1    Running            0          8s
```

Next Steps

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

3.3. VERIFYING DATA GRID CLUSTERS

Review log messages to ensure that Data Grid nodes receive clustered views.

Procedure

- Do either of the following:
  - Retrieve the cluster view from logs.

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

INFO  [org.infinispan.CLUSTER] (MSC service thread 1-2) \
ISPN000094: Received new cluster view for channel infinispan: \
[example-infinispan-0|0] (1) [example-infinispan-0]

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

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

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

    The response indicates that Data Grid pods have received clustered views:

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

```
        type: wellFormed
```

TIP

Use **oc wait** with the **wellFormed** condition for automated scripts.

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

CHAPTER 4. SETTING UP DATA GRID SERVICES

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

4.1. SERVICE TYPES

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

Cache service

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

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

IMPORTANT

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

Data Grid service

Use Data Grid service if you want to:

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

4.2. CREATING CACHE SERVICE NODES

By default, Data Grid Operator creates Data Grid clusters with Cache service nodes.

Procedure

1. Create an **Infinispan** CR.

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

```
replicas: 2
service:
  type: Cache 1
```

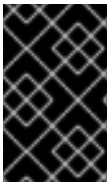
1 Creates nodes Cache service nodes. This is the default for the **Infinispan** CR.

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

4.2.1. Configuring Automatic Scaling

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

Data Grid Operator monitors default caches on Cache service nodes. As you add data to the cache, memory usage increases. When it detects that the cluster needs additional capacity, Data Grid Operator creates new nodes rather than eviciting entries. Likewise, if it detects that memory usage is below a certain threshold, Data Grid Operator shuts down nodes.



IMPORTANT

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

Procedure

1. Add the **spec.autoscale** resource to your **Infinispan** CR to enable automatic scaling.
2. Configure memory usage thresholds and number of nodes for your cluster with the **autoscale** field.

```
spec:
  ...
  service:
    type: Cache
  autoscale:
    maxMemUsagePercent: 70 1
    maxReplicas: 5 2
    minMemUsagePercent: 30 3
    minReplicas: 2 4
```

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

2 Defines the maximum number of number of nodes for the cluster.

3 Configures the minimum threshold, as a percentage, for memory usage across the cluster. When Data Grid Operator detects that memory usage falls below the minimum, it shuts down nodes.

4 Defines the minimum number of number of nodes for the cluster.

3. Apply the changes.

4.2.2. Configuring the Number of Owners

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

Procedure

1. Specify the number of owners with the **spec.service.replicationFactor** resource in your **Infinispan** CR as follows:

```
spec:
  ...
  service:
    type: Cache
    replicationFactor: 3 (1)
```

(1) Configures three replicas for each cache entry.

2. Apply the changes.

4.2.3. Cache Service Resources

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  # Names the cluster.
  name: example-infinispan
spec:
  # Specifies the number of nodes in the cluster.
  replicas: 4
  service:
    # Configures the service type as Cache.
    type: Cache
    # Sets the number of replicas for each entry across the cluster.
    replicationFactor: 2
  # Enables and configures automatic scaling.
  autoscale:
    maxMemUsagePercent: 70
    maxReplicas: 5
    minMemUsagePercent: 30
    minReplicas: 2
  # Configures authentication and encryption.
  security:
    # Defines a secret with custom credentials.
    endpointSecretName: endpoint-identities
    # Adds a custom TLS certificate to encrypt client connections.
    endpointEncryption:
        type: Secret
        certSecretName: tls-secret
  # Sets container resources.
  container:
```

```
  extraJvmOpts: "-XX:NativeMemoryTracking=summary"
  cpu: "2000m"
  memory: 1Gi
# Configures logging levels.
logging:
  categories:
    org.infinispan: trace
    org.jgroups: trace
# Configures how the cluster is exposed on the network.
expose:
  type: LoadBalancer
affinity:
  podAntiAffinity:
    preferredDuringSchedulingIgnoredDuringExecution:
    - weight: 100
      podAffinityTerm:
        labelSelector:
          matchLabels:
            app: infinispan-pod
            clusterName: example-infinispan
            infinispan_cr: example-infinispan
        topologyKey: "kubernetes.io/hostname"
```

4.3. CREATING DATA GRID SERVICE NODES

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

Procedure

1. Specify **DataGrid** as the value for **spec.service.type** in your **Infinispan** CR.

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

NOTE

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

2. Configure nodes with any other Data Grid service resources.
3. Apply your **Infinispan** CR to create the cluster.

4.3.1. Data Grid service Resources

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  # Names the cluster.
  name: example-infinispan
spec:
  # Specifies the number of nodes in the cluster.
  replicas: 6
  service:
    # Configures the service type as Data Grid.
    type: DataGrid
    # Configures storage resources.
    container:
      storage: 2Gi
      storageClassName: my-storage-class
    # Configures cross-site replication.
    sites:
      local:
      name: azure
      expose:
        type: LoadBalancer
      locations:
      - name: azure
        url: openshift://api.azure.host:6443
        secretName: azure-token
      - name: aws
        url: openshift://api.aws.host:6443
        secretName: aws-token
  # Configures authentication and encryption.
  security:
    # Defines a secret with custom credentials.
    endpointSecretName: endpoint-identities
    # Adds a custom TLS certificate to encrypt client connections.
    endpointEncryption:
        type: Secret
        certSecretName: tls-secret
  # Sets container resources.
  container:
    extraJvmOpts: "-XX:NativeMemoryTracking=summary"
    cpu: "1000m"
    memory: 1Gi
  # Configures logging levels.
  logging:
    categories:
      org.infinispan: debug
      org.jgroups: debug
      org.jgroups.protocols.TCP: error
      org.jgroups.protocols.relay.RELAY2: fatal
  # Configures how the cluster is exposed on the network.
  expose:
    type: LoadBalancer
  # Configures affinity and anti-affinity strategies.
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100
```

```
    podAffinityTerm:
      labelSelector:
        matchLabels:
          app: infinispan-pod
          clusterName: example-infinispan
          infinispan_cr: example-infinispan
      topologyKey: "kubernetes.io/hostname"
```

4.4. ADDING LABELS TO DATA GRID RESOURCES

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

NOTE

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

Procedure

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

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  annotations:
    # Add labels that you want to attach to services.
    infinispan.org/targetLabels: svc-label1, svc-label2
    # Add labels that you want to attach to pods.
    infinispan.org/podTargetLabels: pod-label1, pod-label2
  labels:
    # Add values for your labels.
    svc-label1: svc-value1
    svc-label2: svc-value2
    pod-label1: pod-value1
    pod-label2: pod-value2
    # The operator does not attach these labels to resources.
    my-label: my-value
    environment: development
```

4. Apply your **Infinispan** CR.

Additional resources

- Labels and Selectors
- Labels: Kubernetes User Guide

CHAPTER 5. ADJUSTING CONTAINER SPECIFICATIONS

You can allocate CPU and memory resources, specify JVM options, and configure storage for Data Grid nodes.

5.1. JVM, CPU, AND MEMORY RESOURCES

```
spec:
  ...
  container:
    extraJvmOpts: "-XX:NativeMemoryTracking=summary" 1
    cpu: "1000m" 2
    memory: 1Gi 3
```

1. Specifies JVM options.
2. Allocates host CPU resources to node, measured in CPU units.
3. Allocates host memory resources to nodes, measured in bytes.

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

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

Garbage collection logging

By default, Data Grid Operator does not log garbage collection (GC) messages. You can optionally add the following JVM options to direct GC messages to stdout:

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

5.2. STORAGE RESOURCES

By default, Data Grid Operator allocates **1Gi** for storage for both Cache service and Data Grid service nodes. You can configure storage resources for Data Grid service nodes but not Cache service nodes.

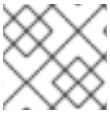
```
spec:
  ...
  service:
    type: DataGrid
    container:
      storage: 2Gi 1
      storageClassName: my-storage-class 2
```

1. Configures the storage size for Data Grid service nodes.
2. Specifies the name of a StorageClass object to use for the persistent volume claim. 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**.

Persistent Volume Claims

Data Grid Operator mounts persistent volumes at:
/opt/infinispan/server/data



NOTE

Persistent volume claims use the **ReadWriteOnce (RWO)** access mode.

CHAPTER 6. STOPPING AND STARTING DATA GRID CLUSTERS

Stop and start Data Grid clusters with Data Grid Operator.

Cache definitions

Both Cache service and Data Grid service store permanent cache definitions in persistent volumes so they are still available after cluster restarts.

Data

Data Grid service nodes can write all cache entries to persistent storage during cluster shutdown if you add a file-based cache store.

6.1. SHUTTING DOWN DATA GRID CLUSTERS

Shutting down Cache service nodes removes all data in the cache. For Data Grid service nodes, you should configure the storage size for Data Grid service nodes to ensure that the persistent volume can hold all your data.

If the available container storage is less than the amount of memory available to Data Grid service nodes, Data Grid writes the following exception to logs and data loss occurs during shutdown:

```
WARNING: persistent volume size is less than memory size. Graceful shutdown may not work.
```

Procedure

- Set the value of **replicas** to **0** and apply the changes.

```
spec:
  replicas: 0
```

6.2. RESTARTING DATA GRID CLUSTERS

Complete the following procedure to restart Data Grid clusters after shutdown.

Prerequisites

For Data Grid service nodes, you must restart clusters with the same number of nodes before shutdown. For example, you shut down a cluster of 6 nodes. When you restart that cluster, you must specify 6 as the value for **spec.replicas**.

This allows Data Grid to restore the distribution of data across the cluster. When all nodes in the cluster are running, you can then add or remove nodes.

You can find the correct number of nodes for Data Grid clusters as follows:

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

Procedure

- Set the value of **spec.replicas** to the appropriate number of nodes for your cluster, for example:

```
spec:
  replicas: 6
```

CHAPTER 7. 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.

7.1. GETTING THE SERVICE FOR INTERNAL CONNECTIONS

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

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

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

Procedure

- Check that the internal service is available as follows:

```
$ oc get services

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

Additional resources

- Network Services

7.2. EXPOSING DATA GRID THROUGH LOAD BALANCERS

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

NOTE

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

Procedure

1. Include **spec.expose** in your **Infinispan** CR.
2. Specify **LoadBalancer** as the service type with **spec.expose.type**.

```
spec:
  ...
  expose:
    type: LoadBalancer (1)
    nodePort: 30000 (2)
```

(1) Exposes Data Grid on the network through a load balancer service on port **11222**.

2 Optionally defines a node port to which the load balancer service forwards traffic.

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

```
$ oc get services | grep external

NAME                        TYPE           CLUSTER-IP    EXTERNAL-IP   PORT(S)
example-infinispan-external  LoadBalancer   192.0.2.24    hostname.com  11222/TCP
```

7.3. EXPOSING DATA GRID THROUGH NODE PORTS

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

Procedure

1. Include **spec.expose** in your **Infinispan** CR.
2. Specify **NodePort** as the service type with **spec.expose.type**.

```
spec:
  ...
  expose:
    type: NodePort 1
    nodePort: 30000 2
```

1 Exposes Data Grid on the network through a node port service.

2 Defines the port where Data Grid is exposed. If you do not define a port, the platform selects one.

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

```
$ oc get services | grep external

NAME                        TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)
example-infinispan-external  NodePort       192.0.2.24      <none>        11222:30000/TCP
```

7.4. EXPOSING DATA GRID THROUGH ROUTES

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

Procedure

1. Include **spec.expose** in your **Infinispan** CR.
2. Specify **Route** as the service type with **spec.expose.type**.

3. Optionally add a hostname with **spec.expose.host**.

```
spec:
  ...
  expose:
    type: Route 1
    host: www.example.org 2
```

1. Exposes Data Grid on the network through an OpenShift Route.
2. Optionally specifies the hostname where Data Grid is exposed.

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

```
$ oc get routes

NAME                  CLASS    HOSTS   ADDRESS   PORTS   AGE
example-infinispan    <none>   *                 443     73s
```

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.

CHAPTER 8. SECURING DATA GRID CONNECTIONS

Secure client connections with authentication and encryption to prevent network intrusion and protect your data.

8.1. CONFIGURING AUTHENTICATION

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

8.1.1. Default Credentials

Data Grid Operator generates base64-encoded default credentials stored in an authentication secret named **example-infinispan-generated-secret**

Username	Description
developer	Default application user.
operator	Internal user that interacts with Data Grid clusters.

8.1.2. Retrieving Credentials

Get credentials from authentication secrets to access Data Grid clusters.

Procedure

- Retrieve credentials from authentication secrets, as in the following example:

```
$ oc get secret example-infinispan-generated-secret
```

Base64-decode credentials.

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

credentials:
- username: developer
  password: dIRs5cAAsHIeeRIL
- username: operator
  password: uMBo9CmEdEduYk24
```

8.1.3. Adding Custom Credentials

Configure access to Data Grid cluster endpoints with custom credentials.

Procedure

1. Create an **identities.yaml** file with the credentials that you want to add.

```
credentials:
```

```
- username: testuser
  password: testpassword
- username: operator
  password: supersecretoperatorpassword
```

IMPORTANT

identities.yaml must include the **operator** user.

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 (1)
```

(1) Specifies the name of the authentication secret that contains your credentials.

Modifying **spec.security.endpointSecretName** triggers a cluster restart. You can watch the Data Grid cluster as Data Grid Operator applies changes:

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

8.2. CONFIGURING ENCRYPTION

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

8.2.1. 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 (1)
      certSecretName: example-infinispan-cert-secret (2)
```

(1) Specifies the Red Hat OpenShift Service.

2 Names the secret that contains a service certificate, **tls.crt**, and key, **tls.key**, in PEM format. If you do not specify a name, Data Grid Operator uses **<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.

8.2.2. Retrieving TLS Certificates

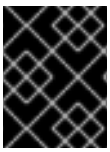
Get TLS certificates from encryption secrets to create client trust stores.

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

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

8.2.3. Disabling Encryption

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



IMPORTANT

Data Grid does not recommend disabling encryption in production environments where endpoints are exposed outside the OpenShift cluster via **spec.expose.type**.

Procedure

- Set **None** as the value for the **spec.security.endpointEncryption.type** field in your **Infinispan** CR and then apply the changes.

```
spec:
  ...
  security:
    endpointEncryption:
          type: None 1
```

1 Disables encryption for Data Grid endpoints.

8.2.4. 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. See:
- Certificate Secrets
- Keystore Secrets

Procedure

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

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

2. Specify the encryption secret with **spec.security.endpointEncryption** in your **Infinispan** CR and then apply the changes.

```
spec:
  ...
  security:
    endpointEncryption: 1
          type: Secret 2
          certSecretName: tls-secret 3
```

1. Encrypts traffic to and from Data Grid endpoints.
2. Configures Data Grid to use secrets that contain encryption certificates.
3. Names the encryption secret.

8.2.4.1. Certificate Secrets

```
apiVersion: v1
kind: Secret
metadata:
  name: tls-secret
type: Opaque
data:
    tls.key:  "LS0tLS1CRUdJTiBQUk ..." 1
    tls.crt: "LS0tLS1CRUdJTiBDRVI ..." 2
```

1. Adds a base64-encoded TLS key.
2. Adds a base64-encoded TLS certificate.

8.2.4.2. Keystore Secrets

```
apiVersion: v1
```

```
kind: Secret
metadata:
  name: tls-secret
type: Opaque
stringData:
    alias: server (1)
    password: password (2)
data:
    keystore.p12:  "MIIKDgIBAzCCCdQGCSqGSIb3DQEHA..." (3)
```

1. Specifies an alias for the keystore.
2. Specifies a password for the keystore.
3. Adds a base64-encoded keystore.

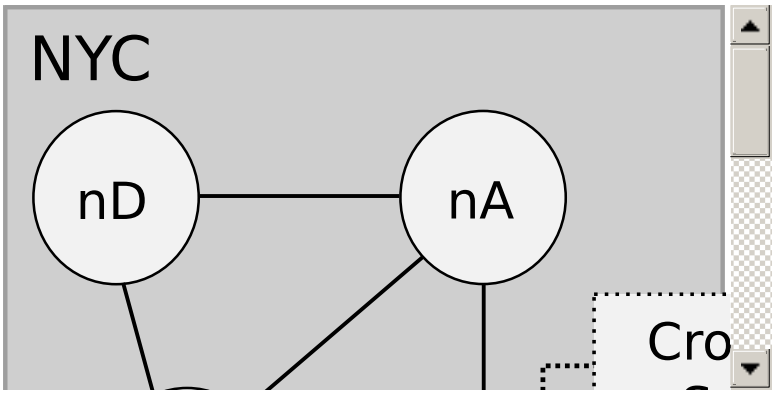
CHAPTER 9. CONFIGURING CROSS-SITE REPLICATION

Set up global Data Grid clusters to back up data across sites.

9.1. CROSS-SITE REPLICATION WITH DATA GRID OPERATOR

If you have Data Grid clusters running in separate locations, use Data Grid Operator to connect them so you can back up data across sites.

For example, in the following illustration, Data Grid Operator manages a Data Grid cluster at a data center in New York City, **NYC**. At another data center in London, **LON**, Data Grid Operator also manages a Data Grid cluster.



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

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

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

9.2. CREATING SERVICE ACCOUNT TOKENS

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

Procedure

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

```
$ 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. Repeat the preceding steps on your other OpenShift clusters.

Additional resources

Using service accounts in applications

9.3. 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...
```

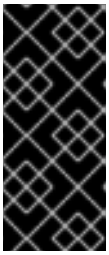
3. Repeat the preceding steps on your other OpenShift clusters.

9.4. CONFIGURING DATA GRID CLUSTERS FOR CROSS-SITE REPLICATION

Configure Data Grid clusters as backup locations so that they can communicate over a dedicated JGroups transport channel for replicating data.

Prerequisites

- Create secrets that contain service account tokens for each backup location.
- Ensure that all clusters are Data Grid service nodes.
- Ensure that OpenShift project names match.



IMPORTANT

To perform cross-site replication, Data Grid Operator requires Data Grid clusters to have the same name and run in matching namespaces.

For example, you create a cluster at **LON** in a project named **xsite-cluster**. The cluster at **NYC** must also run in a project named **xsite-cluster**.

Procedure

1. Create an **Infinispan** CR for each Data Grid cluster.
2. Specify a matching name for each Data Grid cluster with **metadata.name**.
3. Specify the name of the local site with **spec.service.sites.local.name**.
4. Set the expose service type for the local site with **spec.service.sites.local.expose.type**.
5. Provide the name, URL, and secret for each Data Grid cluster that acts as a backup location with **spec.service.sites.locations**.
   The following are example **Infinispan** CR definitions for **LON** and **NYC**:

   - LON

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: example-infinispan
spec:
  replicas: 3
  service:
    type: DataGrid
    sites:
      local:
        name: LON
        expose:
          type: LoadBalancer
      locations:
        - name: LON
          url: openshift://api.rhdg-lon.openshift-aws.myhost.com:6443
          secretName: lon-token
        - name: NYC
          url: openshift://api.rhdg-nyc.openshift-aws.myhost.com:6443
          secretName: nyc-token
```

   - NYC

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

```
    local:
      name: NYC
      expose:
        type: LoadBalancer
    locations:
      - name: NYC
        url: openshift://api.rhdg-nyc.openshift-aws.myhost.com:6443
        secretName: nyc-token
      - name: LON
        url: openshift://api.rhdg-lon.openshift-aws.myhost.com:6443
        secretName: lon-token
```

6. Adjust logging levels for cross-site replication as follows:

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

   The preceding configuration decreases logging for JGroups TCP and RELAY2 protocols to reduce excessive messages about cluster backup operations, which can result in a large number of log files that use container storage.

7. Configure nodes with any other Data Grid service resources.

8. Apply the **Infinispan** CRs.

9. Check node logs to verify that Data Grid clusters form a cross-site view, for example:

```
$ oc logs example-infinispan-0 | grep x-site

INFO  [org.infinispan.XSITE] (jgroups-5,example-infinispan-0-<id>) ISPN000439: Received new x-site view: [NYC]
INFO  [org.infinispan.XSITE] (jgroups-7,example-infinispan-0-<id>) ISPN000439: Received new x-site view: [NYC, LON]
```

Next steps

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

Additional resources

- Cross-Site Replication Resources
- Adding Backup Locations to Caches
- Data Grid Guide to Cross-Site Replication

9.4.1. Cross-Site Replication Resources

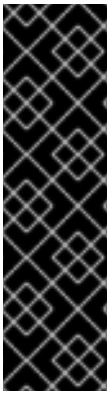
```
spec:
  ...
  service:
    type: DataGrid (1)
```

```
 sites:
   local:
     name: LON (2)
     expose:
       type: LoadBalancer (3)
   locations: (4)
   - name: LON (5)
     url: openshift://api.site-a.devcluster.openshift.com:6443 (6)
     secretName: lon-token (7)
   - name: NYC
     url: openshift://api.site-b.devcluster.openshift.com:6443
     secretName: nyc-token
logging:
  categories:
    org.jgroups.protocols.TCP: error (8)
    org.jgroups.protocols.relay.RELAY2: fatal (9)
```

1. Specifies Data Grid service. Data Grid supports cross-site replication with Data Grid service clusters only.
2. Names the local site for a Data Grid cluster.
3. Specifies **LoadBalancer** as the service that handles communication between backup locations.
4. Provides connection information for all backup locations.
5. Specifies a backup location that matches **spec.service.sites.local.name**.
6. Specifies the URL of the OpenShift API for the backup location.
7. Specifies the secret that contains the service account token for the backup site.
8. Logs error messages for the JGroups TCP protocol.
9. Logs fatal messages for the JGroups RELAY2 protocol.

CHAPTER 10. CREATING CACHES WITH DATA GRID OPERATOR

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



IMPORTANT

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

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 them for production. These features provide early access to upcoming product features, enabling customers to test functionality and provide feedback during the development process.

Red Hat Technology Preview Features Support Scope

When using **Cache** CRs, the following rules apply:

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

10.1. ADDING CREDENTIALS FOR DATA GRID OPERATOR

Data Grid Operator must authenticate with Data Grid service clusters to create caches. You add credentials to a secret so that Data Grid Operator can access your cluster when creating caches.

The following procedure explains how to add credentials to a new secret. If you already have a custom secret that contains credentials, you can use that instead of creating a new one.

Procedure

1. Define a **Secret** object type that provides valid user credentials for accessing Data Grid service clusters in a **StringData** map.
   For example, create an **basic-auth.yaml** file that provides credentials for the **developer** user as follows:

```
apiVersion: v1
stringData:
  username: developer (1)
  password: G8ZdJvSaY3lOOwfM (2)
```

```
kind: Secret
metadata:
  name: basic-auth 3
type: Opaque
```

1 Names a user that can create caches.

2 Specifies the password that corresponds to the user.

3 Specifies a name for the secret.

2. Create a secret from the file, as in the following example:

```
$ oc apply -f basic-auth.yaml
```

10.1.1. Using Custom Credentials Secrets

Data Grid Operator requires that credentials exist as values for the **username** and **password** keys in a secret. If you have a custom secret that contains Data Grid credentials, but uses different key names, you can override those names in your **Cache** CR.

For example, you have a secret named "my-credentials" that holds a list of Data Grid users and their passwords as follows:

```
stringData:
  app_user1: spock
  app_user1_pw: G8ZdJvSaY3lOOwfM
  app_user2: jim
  app_user2_pw: zTzz2gVyyF4JsYsH
```

Procedure

- In your **Cache** CR, override custom key names with **username** and **password** as follows:

```
spec:
  adminAuth:
    username:
      key: app_user1 1
      name: my-credentials 2
    password:
      key: app_user1_pw 3
      name: my-credentials
```

1 Overrides the **app_user1** key name with **username**.

2 Specifies the name of your custom credentials secret.

3 Overrides the **app_user1_pw** key name with **password**.

10.2. CREATING DATA GRID CACHES FROM XML

Complete the following steps to create caches on Data Grid service clusters using valid **infinispan.xml** cache definitions.

Prerequisites

- Create a secret that contains valid user credentials for accessing Data Grid clusters.

Procedure

1. Create a **Cache** CR that contains the XML cache definition you want to create.

```
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: mycachedefinition (1)
spec:
  adminAuth: (2)
    secretName: basic-auth
  clusterName: example-infinispan (3)
  name: mycache (4)
  template: <infinispan><cache-container><distributed-cache name="mycache"
mode="SYNC"><persistence><file-store/></persistence></distributed-cache></cache-
container></infinispan> (5)
```

(1) Names the **Cache** CR.

(2) Specifies a secret that provides credentials with **username** and **password** keys or an override for custom credentials secrets.

(3) Specifies the name of the target Data Grid cluster where you want Data Grid Operator to create the cache.

(4) Names the cache on the Data Grid cluster.

(5) Specifies the XML cache definition to create the cache. Note that the **name** attribute is ignored. Only **spec.name** applies to the resulting cache.

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

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

10.3. CREATING DATA GRID CACHES FROM TEMPLATES

Complete the following steps to create caches on Data Grid service clusters using cache configuration templates.

Prerequisites

- Create a secret that contains valid user credentials for accessing Data Grid clusters.
- Identify the cache configuration template you want to use for your cache. You can find a list of available configuration templates in Data Grid Console.

Procedure

1. Create a **Cache** CR that specifies the name of the template you want to use.
   For example, the following CR creates a cache named "mycache" that uses the **org.infinispan.DIST_SYNC** cache configuration template:

```
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: mycachedefinition (1)
spec:
  adminAuth: (2)
    secretName: basic-auth
  clusterName: example-infinispan (3)
  name: mycache (4)
  templateName: org.infinispan.DIST_SYNC (5)
```

   (1) Names the **Cache** CR.

   (2) Specifies a secret that provides credentials with **username** and **password** keys or an override for custom credentials secrets.

   (3) Specifies the name of the target Data Grid cluster where you want Data Grid Operator to create the cache.

   (4) Names the Data Grid cache instance.

   (5) Specifies the **infinispan.org** cache configuration template to create the cache.

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

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

10.4. ADDING BACKUP LOCATIONS TO CACHES

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

Procedure

1. Create cache configurations with identical names for each site.
   Cache configurations at each site can use different cache modes and backup strategies. Data Grid replicates data based on cache names.

2. Configure backup locations to go offline automatically with the **take-offline** element.

   a. Set the amount of time, in milliseconds, before backup locations go offline with the **min-wait** attribute.

3. Define any other valid cache configuration.

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

The following configuration examples show backup locations for caches:

- NYC

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

- LON

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

Additional resources

- Configuring Clusters for Cross-Site Replication
- Data Grid Guide to Cross-Site Replication

10.4.1. Performance Considerations with Taking Backup Locations Offline

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

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

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

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

10.5. ADDING PERSISTENT CACHE STORES

You can add Single File cache stores to Data Grid service nodes to save data to the persistent volume.

You configure cache stores as part of your Data Grid cache definition with the **persistence** element as follows:

```
<persistence>
  <file-store/>
</persistence>
```

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

Procedure

- Add a cache store to your cache configurations as follows:

```
<infinispan>
 <cache-container>
  <distributed-cache name="customers" mode="SYNC">
   <encoding media-type="application/x-protostream"/>
   <persistence>
    <file-store/>
   </persistence>
  </distributed-cache>
 </cache-container>
</infinispan>
```

Additional resources

- Storage Resources

CHAPTER 11. ESTABLISHING REMOTE CLIENT CONNECTIONS

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

11.1. CLIENT CONNECTION DETAILS

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

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

Service hostnames

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

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

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

Ports

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

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

Additional resources

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

11.2. CREATING DATA GRID CACHES

To create caches when running Data Grid on OpenShift, you can:

- Use **Cache** CR.
- Create multiple caches at a time with Data Grid CLI if you do not use **Cache** CR.
- Access Data Grid Console and create caches in XML or JSON format as an alternative to **Cache** CR or Data Grid CLI.

- Use Hot Rod clients to create caches either programmatically or through per cache properties only if required.

Additional resources

Creating Caches with Data Grid Operator

11.3. CONNECTING WITH THE DATA GRID CLI

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

The CLI is available as part of the server distribution, which you can run on your local host to establish remote connections to Data Grid clusters on OpenShift.

NOTE

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

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

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

11.3.1. Creating Caches with Data Grid CLI

Add caches to your Data Grid cluster with the CLI.

Prerequisites

- Download the server distribution so you can run the CLI.
- Retrieve the necessary client connection details.

Procedure

1. Create a file with a cache configuration in XML or JSON format, for example:

```
cat > infinispan.xml<<EOF
<infinispan>
  <cache-container>
    <distributed-cache name="mycache">
      <encoding>
        <key media-type="application/x-protostream"/>
        <value media-type="application/x-protostream"/>
      </encoding>
    </distributed-cache>
  </cache-container>
</infinispan>
EOF
```

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

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

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

3. Enter your Data Grid credentials when prompted.
4. Add the cache with the **create cache** command and the **--file** option.

```
[//containers/default]> create cache --file=infinispan.xml mycache
```

5. Verify the cache exists with the **ls** command.

```
[//containers/default]> ls caches
mycache
```

6. Optionally retrieve the cache configuration with the **describe** command.

```
[//containers/default]> describe caches/mycache
```

Additional resources

- Downloading Server Distributions
- Using the Data Grid Command Line Interface

11.3.2. Creating Caches in Batches

Add multiple caches with batch operations with the Data Grid CLI.

Prerequisites

- Download the server distribution so you can run the CLI.
- Retrieve the necessary client connection details.

Procedure

1. Create at least one file with a cache configuration in XML or JSON format.
2. Create a batch file, for example:

```
cat > caches.batch<<EOF
echo "connecting"
connect --username=developer --password=dIRs5cAAsHIeeRIL
echo "creating caches..."
create cache firstcache --file=infinispan-one.xml
create cache secondcache --file=infinispan-two.xml
create cache thirdcache --file=infinispan-three.xml
create cache fourthcache --file=infinispan-four.xml
echo "verifying caches"
ls caches
EOF
```

3. Create the caches with the CLI.

```
$ bin/cli.sh -c https://$SERVICE_HOSTNAME:$PORT --trustall -f /tmp/caches.batch
```

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

Additional resources

- Downloading Server Distributions
- Using the Data Grid Command Line Interface

11.4. 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 load balancer service or create a route.

Procedure

1. Access the console from any browser at **$SERVICE_HOSTNAME:$PORT**. Replace **$SERVICE_HOSTNAME:$PORT** with the hostname and port where Data Grid is available on the network.
2. Enter your Data Grid credentials when prompted.

11.5. HOT ROD CLIENTS

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

Client intelligence

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

Hot Rod clients running on OpenShift can access internal IP addresses for Data Grid nodes so you can use any client intelligence. The default intelligence, **HASH_DISTRIBUTION_AWARE**, is recommended because it allows clients to route requests to primary owners, which improves performance.

Hot Rod clients running outside OpenShift must use **BASIC** intelligence.

11.5.1. Hot Rod Configuration API

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

NOTE

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

On OpenShift

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

```
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("$SERVICE_HOSTNAME")
            .port(ConfigurationProperties.DEFAULT_HOTROD_PORT)
           .security().authentication()
            .username("username")
            .password("password")
            .realm("default")
            .saslQop(SaslQop.AUTH)
            .saslMechanism("SCRAM-SHA-512")
           .ssl()
            .sniHostName("$SERVICE_HOSTNAME")
            .trustStorePath("/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt");
```

Outside OpenShift

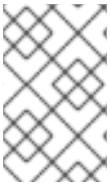
Hot Rod clients running outside OpenShift can use the following configuration:

```
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("$SERVICE_HOSTNAME")
            .port("$PORT")
           .security().authentication()
            .username("username")
            .password("password")
            .realm("default")
            .saslQop(SaslQop.AUTH)
            .saslMechanism("SCRAM-SHA-512")
           .ssl()
            .sniHostName("$SERVICE_HOSTNAME")
            .trustStorePath("/path/to/tls.crt");
    builder.clientIntelligence(ClientIntelligence.BASIC);
```

11.5.2. Hot Rod Client Properties

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



NOTE

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

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

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

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

# Encryption
infinispan.client.hotrod.sni_host_name=$SERVICE_HOSTNAME
# Path to the TLS certificate.
# Clients automatically generate trust stores from certificates.
infinispan.client.hotrod.trust_store_path=/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt
```

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

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

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

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

# Encryption
infinispan.client.hotrod.sni_host_name=$SERVICE_HOSTNAME
# Path to the TLS certificate.
# Clients automatically generate trust stores from certificates.
infinispan.client.hotrod.trust_store_path=tls.crt
```

11.5.3. Creating Caches with 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("<infinispan>" +
                              "<cache-container>" +
                              "<distributed-cache name=\"%s\" mode=\"SYNC\">" +
                                "<encoding media-type=\"application/x-protostream\"/>" +
                                "<locking isolation=\"READ_COMMITTED\"/>" +
                                "<transaction mode=\"NON_XA\"/>" +
                                "<expiration lifespan=\"60000\" interval=\"20000\"/>" +
                              "</distributed-cache>" +
                              "</cache-container>" +
                            "</infinispan>"
                            , 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 Hot Rod 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
```

11.6. ACCESSING THE REST API

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

Prerequisites

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

Procedure

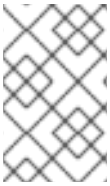
- Access the REST API with any HTTP client at **$SERVICE_HOSTNAME:$PORT/rest/v2**.
  Replace **$SERVICE_HOSTNAME:$PORT** with the hostname and port where Data Grid is available on the network.

Additional resources

- Data Grid REST API

11.7. ADDING CACHES TO CACHE SERVICE NODES

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



NOTE

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

Procedure

- Access the Data Grid Console and provide a copy of the default configuration in XML or JSON format.
- Use the Data Grid CLI to create a copy from the default cache as follows:

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

11.7.1. Default Cache Configuration

The default cache for Cache service nodes is as follows:

```
<infinispan>
```

```
<cache-container>
  <distributed-cache name="default" 1
                     mode="SYNC" 2
                     owners="2"> 3
    <memory storage="OFF_HEAP" 4
            max-size="<maximum_size_in_bytes>" 5
            when-full="REMOVE" /> 6
    <partition-handling when-split="ALLOW_READ_WRITES" 7
                        merge-policy="REMOVE_ALL"/> 8
  </distributed-cache>
 </cache-container>
</infinispan>
```

1. Names the cache instance as "default".
2. Uses synchronous distribution for storing data across the cluster.
3. Configures two replicas of each cache entry on the cluster.
4. Stores cache entries as bytes in native memory (off-heap).
5. Defines the maximum size for the data container in bytes. Data Grid Operator calculates the maximum size when it creates nodes.
6. Evicts cache entries to control the size of the data container. You can enable automatic scaling so that Data Grid Operator adds nodes when memory usage increases instead of removing entries.
7. Names a conflict resolution strategy that allows read and write operations for cache entries, even if segment owners are in different partitions.
8. Specifies a merge policy that removes entries from the cache when Data Grid detects conflicts.

CHAPTER 12. MONITORING DATA GRID WITH PROMETHEUS

Data Grid exposes a metrics endpoint that provides statistics and events to Prometheus.

12.1. CREATING A PROMETHEUS SERVICE MONITOR

Define a service monitor instances that configures Prometheus to monitor your Data Grid cluster.

Prerequisites

- Set up a Prometheus stack on your OpenShift cluster.

Procedure

1. Create an authentication secret that contains Data Grid credentials so that Prometheus can authenticate with your Data Grid cluster.

```
apiVersion: v1
stringData:
  username: developer (1)
  password: dIRs5cAAsHIeeRIL (2)
kind: Secret
metadata:
  name: basic-auth
type: Opaque
```

(1) Specifies an application user. **developer** is the default.

(2) Specifies the corresponding password.

2. Add the authentication secret to your Prometheus namespace.

```
$ oc apply -f basic-auth.yaml
```

3. Create a service monitor that configures Prometheus to monitor your Data Grid cluster.

```
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    k8s-app: prometheus
  name: datagrid-monitoring (1)
  namespace: infinispan-monitoring (2)
spec:
  endpoints:
    - targetPort: 11222 (3)
      path: /metrics (4)
      honorLabels: true
      basicAuth:
        username:
          key: username
          name: basic-auth (5)
```

```
      password:
        key: password
        name: basic-auth
    interval: 30s
    scrapeTimeout: 10s
    scheme: https (6)
    tlsConfig:
      insecureSkipVerify: true
      serverName: example-infinispan (7)
  namespaceSelector:
    matchNames:
      - infinispan (8)
  selector:
    matchLabels:
      app: infinispan-service
      clusterName: example-infinispan (9)
```

1. Names the service monitor instances.
2. Specifies the namespace of your Prometheus stack.
3. Sets the port of **11222** for the Data Grid metrics endpoint.
4. Sets the path where Data Grid exposes metrics.
5. Specifies the authentication secret with Data Grid credentials.
6. Specifies that Data Grid clusters use endpoint encryption.
7. Specifies the Common Name (CN) of the TLS certificate for Data Grid encryption. If you use an OpenShift service certificate, the CN matches the **metadata.name** resource for your Data Grid cluster.
8. Specifies the namespace of your Data Grid cluster.
9. Specifies the name of your Data Grid cluster.

4. Add the service monitor instance to your Prometheus namespace.

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

Additional resources

- Prometheus Operator
- Monitoring Services with the OpenShift Cluster Monitoring Stack

CHAPTER 13. GUARANTEEING AVAILABILITY WITH ANTI-AFFINITY

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

13.1. 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.

13.2. 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.

Additional resources

- Anti-Affinity Strategy Configurations

13.3. ANTI-AFFINITY STRATEGY CONFIGURATIONS

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

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 (1)
        podAffinityTerm:
          labelSelector:
            matchLabels:
              app: infinispan-pod
              clusterName: <cluster_name>
              infinispan_cr: <cluster_name>
          topologyKey: "kubernetes.io/hostname" (2)
```

(1) Sets the hostname strategy as most preferred.

(2) Schedules Data Grid replica pods on different OpenShift nodes.

Requiring different nodes

```
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution: (1)
      - labelSelector:
          matchLabels:
            app: infinispan-pod
            clusterName: <cluster_name>
            infinispan_cr: <cluster_name>
        topologyKey: "topology.kubernetes.io/hostname"
```

1 OpenShift does not schedule Data Grid pods if there are no different nodes available.

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:

```
spec:
  affinity:
    podAntiAffinity:
      preferredDuringSchedulingIgnoredDuringExecution:
      - weight: 100 (1)
        podAffinityTerm:
          labelSelector:
            matchLabels:
              app: infinispan-pod
              clusterName: <cluster_name>
              infinispan_cr: <cluster_name>
          topologyKey: "topology.kubernetes.io/zone" (2)
      - weight: 90 (3)
        podAffinityTerm:
          labelSelector:
            matchLabels:
              app: infinispan-pod
              clusterName: <cluster_name>
              infinispan_cr: <cluster_name>
          topologyKey: "kubernetes.io/hostname" (4)
```

1 Sets the zone strategy as most preferred.

2 Schedules Data Grid replica pods across multiple zones.

3 Sets the hostname strategy as next preferred.

4 Schedules Data Grid replica pods on different OpenShift nodes if it is not possible to schedule across zones.

Requiring multiple zones

```
spec:
  affinity:
    podAntiAffinity:
      requiredDuringSchedulingIgnoredDuringExecution: (1)
      - labelSelector:
          matchLabels:
            app: infinispan-pod
```

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

1 Uses the zone strategy only when scheduling Data Grid replica pods.

CHAPTER 14. MONITORING DATA GRID LOGS

Set logging categories to different message levels to monitor, debug, and troubleshoot Data Grid clusters.

14.1. CONFIGURING DATA GRID LOGGING

Procedure

1. Specify logging configuration with **spec.logging** in your **Infinispan** CR and then apply the changes.

```
spec:
  ...
  logging: 1
    categories: 2
      org.infinispan: debug 3
      org.jgroups: debug
```

1. Configures Data Grid logging.
2. Adds logging categories.
3. Names logging categories and levels.

NOTE

The root logging category is **org.infinispan** and is **INFO** by default.

2. Retrieve logs from Data Grid nodes as required.

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

14.2. LOG LEVELS

Log levels indicate the nature and severity of messages.

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.

CHAPTER 15. REFERENCE

Find information about Data Grid services and clusters that you create with Data Grid Operator.

15.1. NETWORK SERVICES

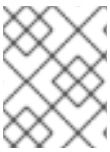
Internal service

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

Service	Port	Protocol	Description
<cluster_name>	**11222**	TCP	Internal access to Data Grid endpoints
<cluster_name>-ping	**8888**	TCP	Cluster discovery

External service

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



NOTE

You must create the external service with Data Grid Operator. It is not available by default.

Service	Port	Protocol	Description
<cluster_name>-external	**11222**	TCP	External access to Data Grid endpoints.

Cross-site service

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

Service	Port	Protocol	Description
<cluster_name>-site	**7900**	TCP	JGroups RELAY2 channel for cross-site communication.

Additional resources

Creating Network Services