Red Hat Data Grid 7.3

Red Hat Data Grid for OpenShift

在 OpenShift 中运行 Data Grid

Last Updated: 2023-03-03

Red Hat Data Grid 7.3 Red Hat Data Grid for OpenShift

在 OpenShift 中运行 Data Grid

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律通告

摘要

了解如何在 Red Hat OpenShift 中配置、自定义和运行红帽数据网格容器。

目录

第 1 章 RED HAT DATA GRID

数据网格为 Red Hat OpenShift 提供了一个弹性的可扩展内存数据存储。

无架构数据结构

灵活性将不同的对象存储为键值对。

基于网格的数据存储

设计为在集群间分发和复制数据。

弹性扩展

动态调整节点数，以在不中断服务的情况下满足需求。

数据互操作性

在来自不同端点的网格中存储、检索和查询数据。

1.1. DATA GRID 文档

Red Hat Data Grid 文档 可以通过红帽客户门户网站获得。

1.2. DATA GRID REPOSITORIES

- Data Grid 7 OpenShift Image 包含适用于 OpenShift 的 Data Grid 的容器镜像和资源。
- OpenShift Quickstart 教程 提供了代码示例，它包括了在 OpenShift 上运行 Data Grid 的方式和最佳实践演示。

1.3. DATA GRID IMAGE DETAILS

Red Hat Data Grid for OpenShift 镜像托管在 Red Hat Container Registry 上，您可以在其中找到镜像的健康索引以及各个标记版本的信息。

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

第 2 章 开始使用

2.1. 系统要求

要将 Red Hat Data Grid 用于 OpenShift，您需要：

- 正在运行的红帽 OpenShift 集群。
  如需支持的 Red Hat OpenShift Container Platform 版本，请参阅 Red Hat Data Grid 支持的配置。

  提示

  使用 Red Hat Container Development Kit 创建带有 minishift 的本地 OpenShift 集群。

- 您的 **$PATH** 中的 **oc** 客户端。

2.2. 为 OPENSHIFT 项目创建数据网格

设置 OpenShift 项目，您可以为 OpenShift Pod 运行 Data Grid。

1. 登录您的 OpenShift 集群。
   如果您刚开始 OpenShift，请尝试以下教程： 登录 OpenShift 集群。
2. 使用 **oc new-project** 命令创建 OpenShift 项目，例如：

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

2.3. 设置 REGISTRY 身份验证

您必须使用 Red Hat Container Catalog *registry.redhat.io* 进行身份验证，才能拉取 Data Grid 镜像。

使用以下之一：

- 红帽客户帐户用户名和密码。使用 **docker login** 命令从 *registry.redhat.io* 拉取资源。
- registry 服务帐户令牌。使用身份验证令牌配置多个主机。要做到这一点：
  1. 登录 *registry.redhat.io*。
  2. 创建或选择一个 **Registry Service Account**。
  3. 生成身份验证令牌。

2.3.1. 使用身份验证令牌配置主机

将身份验证令牌从 **Registry 服务帐户添加到** 主机，如下所示：

1. 选择 **Docker Login** 选项卡，再复制该命令。
2. 在从 *registry.redhat.io* 的每个主机上运行 **docker login** 命令。
3. 验证您的 Docker 配置。

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

2.3.2. 创建 Pull Secrets

要拉取 OpenShift 的内部 registry 上不可用的安全容器镜像，请执行以下操作：

1. 登录您的 OpenShift 集群。
2. 选择您的项目，例如：

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

3. 使用您的 Docker 配置创建通用 pull secret。

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

4. 将 pull secret 链接到您的服务帐户。

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

5. 挂载 secret。

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

如需更多信息，包括故障排除步骤，请参阅 Red Hat Container Registry 身份验证 。

第 3 章 配置身份验证和加密

只有在您使用自定义模板或希望将您自己的密钥存储与 Data Grid 部署配置搭配使用时，才需要配置身份验证和加密。

3.1. 在 SECRET 中添加密钥存储

配置身份验证和加密：

1. 创建含有可信证书的密钥存储(**.jks**)。
   HTTPS 和 Hot Rod 服务都可以使用相同的密钥存储，或者您可以创建单独的密钥存储。
2. 将密钥存储添加为 OpenShift 机密。
   a. 创建 secret。例如，若要从名为 **rhdg-https.jks** 的密钥存储创建一个名为 **rhdg-https-secret** 的 secret：

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

   b. 将 secret 链接到 default 服务帐户。

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

3.2. 配置部署

使用以下参数实例化一个安全模板：

1. 设置 HTTP 和 HTTPS 主机名：
   HOSTNAME_HTTP=my.example.hostname

   HOSTNAME_HTTPS=secure-my.example.hostname
2. 指定密钥存储的名称： **HTTPS_KEYSTORE=keystore.jks**
3. 指定密钥存储的路径： **HTTPS_KEYSTORE_DIR=/etc/datagrid-secret-volume**
4. 指定 secret 的名称： **HTTPS_SECRET=rhdg-https-secret**
5. 指定密钥存储的凭据：
   HTTPS_NAME=${USERNAME}

   HTTPS_PASSWORD=${PASSWORD}
6. 为用户设置 HTTP 安全域： **REST_SECURITY_DOMAIN=SecurityRealm**
7. 强制客户端证书身份验证： **ENCRYPTION_REQUIRE_SSL_CLIENT_AUTH=true**
8. 为 Hot Rod 协议启用验证和加密：IRC **ROD_AUTHENTICATION=true**

注意

如果您为 **HOSTNAME_HTTPS** 设置值，则模板会自动设置 **HOTROD_ENCRYPTION=true**。

3.3. 为 HOT ROD PROTOCOL 设置唯一密钥存储

将唯一密钥存储用于 Hot Rod 协议：

1. 指定密钥存储的路径： **SSL_KEYSTORE_PATH=hr_keystore.jks**
2. 指定密钥存储密码： **SSL_KEYSTORE_PASSWORD=${PASSWORD}**
3. 如有必要，请执行以下操作：
   a. 设置密钥存储的路径： **SSL_KEYSTORE_RELATIVE_TO=path/to/keystore/**
   b. 如果与密钥存储密码不同，请指定私钥密码： **SSL_KEY_PASSWORD=${PASSWORD}**
   c. 如果在密钥存储中设置正确的别名： **SSL_KEYSTORE_ALIAS=cert_alias**
4. 如果您还没有凭证，请指定授权凭证：
   USERNAME=${USERNAME}

   PASSWORD=${PASSWORD}

注意

Hot Rod 端点始终使用 **ApplicationRealm** 授权用户。如果要将独立密钥存储用于 Hot Rod 和 REST 端点，则必须使用 **USERNAME** 和 **PASSWORD** 参数设置凭据。然后，模板将 REST 端点配置为使用 **jdg-openshift** 安全域。在本例中，**REST_SECURITY_DOMAIN** 环境变量不会生效。

第 4 章 为 OPENSHIFT 服务设置数据网格

4.1. 用于 OPENSHIFT 服务的数据网格

数据网格服务是有状态的应用程序，您可以轻松地扩展或缩减而不丢失数据。

cache-service

适用于 OpenShift 集群的易于使用的数据网格，旨在通过高性能缓存加快应用程序响应时间。

- 内存中的数据在节点间平均分布。在创建服务时，您可以定义 Data Grid 集群的初始大小。发行版也是同步的。当向另一节点传播数据时，发送节点会等待操作在线程继续之前完成。
- 默认情况下，缓存条目的单一副本。如果 Pod 重启，则 pod 中的数据将会丢失。为了实现更加弹性的数据，您可以在创建服务时轻松启用复制。
- 缓存条目存储了 JVM 效率的不足堆。当缓存大小达到 pod 可用的内存量时，条目会被驱除。您可以选择更改驱除策略来引发 **ContainerFullException**。

datagrid-service

针对 OpenShift 的完整数据网格分布，可让您创建多个不同的缓存配置。为您提供索引和查询以及 Prometheus 监控等高级功能。

4.1.1. Container Storage

cache-service 和 **datagrid-service** 容器在 **/opt/datagrid/standalone/data** 上挂载了存储卷。

卷大小为 1GB。您可以使用 **datagrid-service** 来调整大小，但不使用 **cache-service**。

ephemeral 或 Permanent

当您远程创建缓存时，您可以控制它们是否为临时还是永久的。永久缓存会在容器重启后保留，因为缓存定义保存在存储卷中。默认缓存始终是永久的。

persistent

datagrid-service 可让您持久的缓存条目和索引到存储卷。如果您需要更多的数据保证，您可以选择性地通过缓存存储保留基于外部文件的存储或数据库。

4.1.2. 分区处理

默认情况下，OpenShift 服务的 Data Grid 使用分区处理配置来确保数据一致性。

- **DENY_READ_WRITES** 冲突解决策略，该策略拒绝缓存条目的读写操作，除非网段的所有所有者都位于同一分区。
- **REMOVE_ALL** merge 策略，在检测到冲突时从缓存中删除条目。

注意

只有在数据在集群间复制时，才会应用网络分区。

4.1.3. 确认服务可用性

用于 **cache-service** 和 **datagrid-service** 的模板可在 **openshift** 命名空间中的 Red Hat OpenShift Online 和 Red Hat OpenShift Container Platform 中找到。

运行以下命令，以验证服务模板是否可用：

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

4.1.3.1. 导入模板

如有必要，导入 **cache-service** 和 **datagrid-service**，如下所示：

1. 登录您的 OpenShift 集群。
2. 导入服务模板：

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

提示

使用 **oc replace --force** 覆盖现有模板。

4.2. 创建缓存服务

使用 **cache-service** 快速设置集群，以提供在最小配置中最佳性能和易用性。

1. 使用 **new-app** 命令创建服务。
2. 根据情况设置模板参数和环境变量。

例如：

- 使用最小配置创建 **cache-service** ：

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

- 创建带有三个节点和数据复制的 **cache-service** 集群：

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

模板参数

- **APPLICATION_NAME** 指定应用的名称。默认为 **cache-service**。

- **NUMBER_OF_INSTANCES** 设置用于 OpenShift 集群的 Data Grid 中的节点数量。默认值为 **1**。
- **TOTAL_CONTAINER_MEM** 配置容器可用的内存大小（以 MiB 为单位）。默认值为 **512**。
- **APPLICATION_USER** 创建一个用户以安全访问缓存。没有默认值。您必须始终创建用户。
- **APPLICATION_PASSWORD** 指定用户的密码。如果您没有设置密码，服务模板会随机生成一个，并将其存储为 secret。
- **REPLICATION_FACTOR** 指定每个缓存条目的副本数。默认值为 **1**。
- **EVICTION_POLICY** 定义在 **缓存大小** 达到 pod 可用内存量时缓存服务的行为。有两个值：
  - **驱除** 从缓存中删除条目。这是默认值。
  - **拒绝** 引发 **ContainerFullException**，而不是添加新条目。

环境变量

- **AB_PROMETHEUS_ENABLE** 允许您收集 JMX 指标来监控和分析 Data Grid，并具有以下值：

  false

  禁用使用默认 Prometheus 代理的监控。

  true

  使用默认的 Prometheus 代理启用监控。Prometheus Operator 必须已安装并正在运行。在创建服务后，还必须 设置 Monitoring。

- **AB_PROMETHEUS_JMX_EXPORTER_PORT** 定义数据网格发布 JMX 指标的端口。默认值为 **9779**。

验证应用程序

在创建 **cache-service** 时命令输出会显示参数值和资源，如下例所示：

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

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

    * With parameters:
       ...

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

提示

尝试 Hello World Quickstart 教程。

4.3. 创建数据网格服务

使用 **datagrid-service** 设置集群，可用于不同的缓存配置和更复杂的 Data Grid 功能。

1. 使用 **new-app** 命令创建服务。
2. 根据情况设置模板参数和环境变量。

例如：

- 使用最小配置创建 **datagrid-service** ：

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

- 创建启用了三个节点和监控的 **datagrid-service** 集群：

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

模板参数

- **APPLICATION_NAME** 指定应用的名称。默认为 **datagrid-service**。
- **NUMBER_OF_INSTANCES** 设置用于 OpenShift 集群的 Data Grid 中的节点数量。默认值为 1。
- **TOTAL_CONTAINER_MEM** 配置容器可用的内存大小（以 MiB 为单位）。默认值为 512。
- **APPLICATION_USER** 创建一个用户以安全访问缓存。没有默认值。您必须始终创建用户。
- **APPLICATION_PASSWORD** 指定用户的密码。如果您没有设置密码，服务模板会随机生成一个，并将其存储为 secret。
- **REPLICATION_FACTOR** 指定每个缓存条目的副本数。默认值为 2。
- **TOTAL_CONTAINER_STORAGE** 将配置基于文件的存储卷的大小（以 GiB 为单位）。默认值为 1。

环境变量

- **AB_PROMETHEUS_ENABLE** 允许您收集 JMX 指标来监控和分析 Data Grid，并具有以下值：

  false

  禁用使用默认 Prometheus 代理的监控。

  true

  使用默认的 Prometheus 代理启用监控。Prometheus Operator 必须已安装并正在运行。在创建服务后，还必须 设置 Monitoring。

- **AB_PROMETHEUS_JMX_EXPORTER_PORT** 定义数据网格发布 JMX 指标的端口。默认值为 **9779**。

验证应**用程序**

在创建 **datagrid-service** 时命令输出显示参数值和资源，如下例所示：

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

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

     * With parameters:
        ...

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

提示

尝试 Hello World Quickstart 教程。

第 5 章 调用 DATA GRID REST API

数据网格服务通过端口 **8443** 公开 REST 端点。

默认情况下，Data Grid 需要为客户端连接进行数据访问和加密进行用户身份验证。

身份验证

数据网格使用您使用 **APPLICATION_USER** 和 **APPLICATION_PASSWORD** 参数指定的凭证授权数据访问请求。

Encryption

当 Data Grid pod 启动时，它们生成 TLS 证书/密钥对，并将它们保存在 **service-certs** 机密中。TLS 证书由 OpenShift 证书颁发机构(CA)签名。

5.1. 创建到 REST API 的外部路由

使用重新加密 终止通过路由在 OpenShift 访问 Data Grid pod 之外运行的 REST 客户端。

流程

1. **使用重新加密** 终止创建路由。

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

例如：

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

2. 运行 **oc get routes** 来查找 HTTPS 路由主机名，例如：

```
$ oc get routes

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

5.2. 生成 REST 调用

前提条件

- 配置 REST 客户端以进行身份验证和加密。

对于 OpenShift

使用 pod 中挂载的 CA 捆绑包创建信任存储：
/var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt

在 OpenShift 外部

使用 OpenShift 环境的 CA 创建信任存储。

流程

- 根据情况调用 Data Grid REST API。
  例如，调用 **PUT** 调用来添加 key:value 对：

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

5.2.1. 使用 OpenShift CA 进行 REST 调用

如果 CA 证书无效，如本地 OpenShift 集群或 Red Hat OpenShift Container Platform 开发安装，您可以使用 **service-ca.crt** 进行 REST 调用。

流程

1. 从 Data Grid pod 获取 **service-ca.crt**。

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

2. 在调用 REST 调用时传递 **service-ca.crt**。

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

第 6 章 配置 HOT ROD 客户端

数据网格服务在端口 **11222** 上公开 Hot Rod 端点。

默认情况下，Data Grid 需要为客户端连接进行数据访问和加密进行用户身份验证。

身份验证

数据网格使用您使用 **APPLICATION_USER** 和 **APPLICATION_PASSWORD** 参数指定的凭证授权数据访问请求。

Encryption

当 Data Grid pod 启动时，它们生成 TLS 证书/密钥对，并将它们保存在 **service-certs** 机密中。TLS 证书由 OpenShift 证书颁发机构(CA)签名。

6.1. 使用 HOT ROD 配置信任存储

在 Hot Rod 客户端配置中，将 **trustStorePath** 设置为 PEM 格式的有效证书位置。Hot Rod Java 客户端使用路径中找到的所有证书构建内存 Java 密钥存储。

对于 OpenShift

- 指定 OpenShift 证书颁发机构(CA)捆绑包。
  /var/run/secrets/kubernetes.io/serviceaccount/service-ca.crt

在 OpenShift 外部

1. 从 **service-certs** secret 获取 **tls.crt**。

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

2. 指定客户端配置中的 **tls.crt** 的路径。

6.2. 客户端 INTELLIGENCE

客户端智能指的是 Hot Rod 协议提供的机制，以便客户端能够查找并发送请求到 Data Grid pod。

对于 OpenShift

客户端可以访问 pod 的内部 IP 地址，以便可以使用任何客户端智能。建议使用默认的智能 **HASH_DISTRIBUTION_AWARE**，因为它允许客户端将请求路由到主所有者，从而提高性能。

在 OpenShift 外部

仅使用 **BASIC** 智能。

参考

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

6.3. 为 HOT ROD 创建外部路由

在 OpenShift 访问数据网格 pod 的外部热 Rod 客户端通过带有 **passthrough** 终止的路由。

先决条件

- 配置 Data Grid Server 以加密客户端连接。

流程

1. 创建具有 **passthrough** 终止的路由。

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

例如：

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

2. 从 **.spec.host** 获取 Hot Rod 路由主机名。

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

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

6.4. DATA GRID SERVICES 的主机名

使用与您 Hot Rod 客户端的位置对应的 Data Grid 主机名。

在 Same OpenShift 命名空间中

使用 **APPLICATION_NAME**。

例如：

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

在不同的 OpenShift 命名空间上

使用以下格式的内部服务 DNS 名称：
APPLICATION_NAME.SERVICE_NAMESPACE.svc

例如：

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

在 OpenShift 外部

使用 Hot Rod 路由主机名。

例如：

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

6.5. 以方式配置热备份客户端

使用 **ConfigurationBuilder** 类，以编程方式配置 Hot Rod 客户端以访问数据网格集群。

1. 调用 **create（）** 方法以创建您可以传递给 **RemoteCacheManager** 的配置 bean。
2. 使用 **authentication（）** 和 **ssl（）** 方法配置身份验证和加密。

参考

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

6.5.1. OpenShift 上的热备份配置构建程序

配置为在 OpenShift 上运行的 Hot Rod 客户端的配置：

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

6.5.2. 热备份配置构建程序 OpenShift

配置为在 OpenShift 之外运行的 Hot Rod 客户端：

```
ConfigurationBuilder builder = new ConfigurationBuilder();
builder.addServer()
 // Connection
 .host("${HOTROD_ROUTE_HOSTNAME}").port(443)
 // Use BASIC client intelligence.
 .clientIntelligence(ClientIntelligence.BASIC)
 .security()
       // Authentication
       .authentication().enable()
       .username("${USERNAME}")
       .password("${PASSWORD}")
       .serverName("${APPLICATION_NAME}")
       .saslMechanism("DIGEST-MD5")
       .saslQop(SaslQop.AUTH)
       // Encryption
       .ssl()
       .sniHostName("${HOTROD_ROUTE_HOSTNAME}")
       .trustStorePath(path/to/tls.crt);
```

6.6. 设置 HOT ROD 客户端属性

使用 Hot Rod 客户端配置属性指定 Data Grid 主机名和端口、身份验证详情和 TLS 证书。

流程

1. 创建一个 **hotrod-client.properties** 文件，其中包含您的 Hot Rod 客户端配置。
2. 将 **hotrod-client.properties** 添加到 classpath 中。

参考

- org.infinispan.client.hotrod.configuration

6.6.1. OpenShift 上的热备份配置属性

在 OpenShift 上运行的 Hot Rod 客户端的配置属性：

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

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

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

6.6.2. OpenShift 热升级配置属性

在 OpenShift 外部运行的 Hot Rod 客户端的配置属性：

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

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

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

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

第 7 章 远程创建缓存

当您使用 **cache-service** 远程创建缓存时，您可以将缓存配置为临时或永久数据，并在集群间复制数据。

当您使用 **datagrid-service** 远程创建缓存时，您可以定义任何自定义配置。

通过 Hot Rod 协议使用 **cache-service** 和 **datagrid-service** 远程创建缓存定义，如下所示：

1. 实例化 **RemoteCacheManager** 类以连接到该服务。
2. 调用 **createCache（）** 方法以创建缓存，如下例所示：

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

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

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

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

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

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

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

注意

如果指定缓存已存在，则抛出异常。其它方法是：

- 在 **RemoteCacheManagerAdmin** 中调用 **getOrCreateCache** 方法，返回缓存名称而不是抛出异常。
- 在 **RemoteCacheManagerAdmin** 中调用 **removeCache** 方法，以销毁缓存，然后再次调用 **createCache**。

提示

试用 Quickstart 教程之一：

- 使用缓存服务创建缓存
- 使用 Data Grid Service 创建缓存

第 8 章 定义基于文件的缓存存储

使用 **datagrid-service** 定义基于文件的缓存存储，以便将数据持久化到外部存储。

使用 **XMLStringConfiguration** 类，通过 Hot Rod 接口将 XML 配置作为字符串提供。

- XML 必须使用 Data Grid 配置模式有效。
- 您文件存储的位置应该位于 **/opt/datagrid/standalone/data** 的存储卷中，因为 **data** 文件夹是一个 **PersistentVolume**，允许在容器重启时保留数据。

例如，以下 **主** 方法创建带有包含文件存储的分布式配置的缓存：

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

  ConfigurationBuilder cfg = ...

  RemoteCacheManager rcm = new RemoteCacheManager(build);

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

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

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

}
```

有关有效 **file-store** 配置选项的详情，请查看 Data Grid 配置模式。

如需更多信息，请参阅 Javadocs：

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

第 9 章 使用 DATA GRID DEPLOYMENT 配置模板

9.1. DATA GRID DEPLOYMENT 配置模板

Data Grid 提供了一组可帮助您为 OpenShift 部署有不同配置的模板。

重要

从 Data Grid 7.3 开始，这些部署配置模板已被弃用。您应该使用 **cache-service** 或 **datagrid-service** 服务模板。如需更多信息，请参阅 Red Hat Data Grid 支持的配置 。

模板	描述
datagrid73-basic	为 OpenShift 运行 Data Grid，无需身份验证或加密。
datagrid73-https	使用 HTTPS 路由为 OpenShift 运行 Data Grid，以安全访问缓存。需要 OpenShift 机密 来加密网络流量。
datagrid73-mysql	使用 MySQL 数据库作为临时缓存存储为 OpenShift 运行数据网格。需要 OpenShift 机密 来加密网络流量。
datagrid73-mysql-persistent	使用 MySQL 数据库作为持久缓存存储为 OpenShift 运行数据网格。需要 OpenShift 机密 来加密网络流量。
datagrid73-postgresql	使用 PostgreSQL 数据库为 OpenShift 运行 Data Grid，作为临时缓存存储。需要 OpenShift 机密 来加密网络流量。
datagrid73-postgresql-persistent	使用 PostgreSQL 数据库为 OpenShift 运行 Data Grid，作为持久缓存存储。需要 OpenShift 机密 来加密网络流量。
datagrid73-partition	使用分区数据目录为 OpenShift 运行 Data Grid，它会在 pod 重启时保留缓存条目元数据。

9.2. 导入部署配置模板

将 OpenShift 部署配置模板的 Data Grid 导入到 OpenShift 中，如下所示：

1. 登录您的 OpenShift 集群。
2. 导入特定模板或所有模板。
   - 导入特定模板：

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

- 导入所有模板：

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

提示

使用 **oc create** 导入新模板。使用 **oc replace --force** 覆盖现有模板。

使用 **-n** 选项指定要导入模板的命名空间。例如，**-n openshift** 将资源导入到全局 **openshift** 命名空间中，需要管理权限。

3. 导入 Data Grid 镜像。

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

4. 验证 OpenShift 上可用的模板。

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

9.3. 导入 OPENSHIFT SECRET

OpenShift 部署配置模板的一些 Data Grid 需要 HTTPS 和 JGroups 密钥存储。

用于 OpenShift 的数据网格提供 HTTPS 和 JGroups 密钥存储，您可以出于评估而导入。但是，不应在生产环境中使用这个 secret。

使用密钥存储将 secret 导入到项目命名空间中，如下所示：

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

如需更多信息，请参阅：

- 配置身份验证和加密
- 配置 JGroups 加密

9.4. 为 OPENSHIFT 部署数据网格

1. 使用 **new-app** 命令创建一个新部署。

2. 使用 **--template** 选项指定模板。

3. 设置环境变量以使用 **-e** 选项配置部署。
例如，若要使用 **datagrid73-basic** 模板创建部署，其包含名为 **mycache** 的缓存，可启动 eagerly，可运行以下命令：

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

有关支持的 环境变量 的详情，请参阅环境变量。

9.5. 为 OPENSHIFT 配置数据网格

为 OpenShift 部署创建了 Data Grid 后，您可以使用环境变量进行配置。

例如，您有一个名为 **datagrid-app** 的部署配置(**dc**)，其缓存名为 **mycache**。将 **mycache** 配置为启动 lazily，如下所示：

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

修改部署配置时，复制控制器会部署新版本。获取更新的部署配置，如下所示：

```
$ oc get pods

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

验证配置更改，如下所示：

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

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

第 10 章 设置监控

使用 Prometheus Operator 收集 JMX 指标，以监控事件并获取 Data Grid 集群的统计信息。

在高级别上，您可以设置监控功能，如下所示：

1. 将 **AB_PROMETHEUS_ENABLE** 环境变量设置为 **true** 来配置 Data Grid。
2. 安装 Prometheus Operator 并公开 Prometheus Web UI。
3. 将 Data Grid 指标导出到 Prometheus。
4. 启用 Prometheus 以监控用于指标数据的 Data Grid。

10.1. 部署 PROMETHEUS OPERATOR

要安装 Prometheus Operator，请参阅以下文档：

- Prometheus Operator
- 开始使用
- Prometheus RBAC

提示

尝试 监控数据网格快速入门教程.

10.2. 向 PROMETHEUS 公开 DATA GRID METRICS

添加从 Data Grid 向 Prometheus 公开 JMX 指标的服务。

1. 创建一个 **service-metrics.yaml** 文件来定义指标服务。

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

```
  selector:
    deploymentConfig: datagrid-service
  sessionAffinity: None
```

2. 应用 **service-metrics.yaml**。

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

10.3. 启用 PROMETHEUS 以监控 DATA GRID

服务监控器可让 Prometheus 连接到 Data Grid 指标服务。

1. 创建一个 **service-monitor.yaml** 文件，其中包含 **ServiceMonitor** 对象的定义。

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

2. 应用 **service-monitor.yaml**。

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

第 11 章 为 OPENSHIFT 集群配置 DATA GRID

11.1. 配置集群发现

用于 OpenShift 的数据网格可使用 Kubernetes 或 DNS 发现机制来集群。这些发现机制可让镜像自动加入集群。

OpenShift 模板和服务的 data Grid 默认使用 DNS。如果直接从镜像或自定义模板为 OpenShift 部署 Data Grid，您必须配置适当的发现机制。

11.1.1. 配置 DNS_PING

要为集群配置 DNS 发现机制，请执行以下操作：

1. 将 **openshift.DNS_PING** 设置为 **JGROUPS_PING_PROTOCOL** 环境变量的值。

```
JGROUPS_PING_PROTOCOL=openshift.DNS_PING
```

2. 指定集群的 ping 服务名称，作为 **OPENSHIFT_DNS_PING_SERVICE_NAME** 环境变量的值。

```
OPENSHIFT_DNS_PING_SERVICE_NAME=${PING_SERVICE_NAME}
```

3. 指定公开 ping 服务的端口号，作为 **OPENSHIFT_DNS_PING_SERVICE_PORT** 环境变量的值。默认值为 **8888**。

```
OPENSHIFT_DNS_PING_SERVICE_PORT=${PING_SERVICE_NAME}
```

4. 定义公开 ping 端口的 ping 服务，如下例所示：

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

重要

您应该配置 **clusterIP: None**，以便服务处于无头状态。同样，ping 端口必须被命名并包含 **service.alpha.kubernetes.io/tolerate-unready-endpoints: 'true'** 注解。

11.1.2. 配置 KUBE_PING

要为集群配置 Kubernetes 发现机制，请执行以下操作：

1. 将 **openshift.KUBE_PING** 设置为 **JGROUPS_PING_PROTOCOL** 环境变量的值。

```
JGROUPS_PING_PROTOCOL=openshift.KUBE_PING
```

2. 将 OpenShift 项目名称指定为 **OPENSHIFT_KUBE_PING_NAMESPACE** 环境变量的值。如果没有设置此变量，服务器的行为与单节点集群类似。

```
OPENSHIFT_KUBE_PING_NAMESPACE=${PING_NAMESPACE}
```

3. 使用 **OPENSHIFT_KUBE_PING_LABELS** 环境变量指定集群标签。如果没有设置此变量，则应用程序以外的 pod 会尝试加入同一命名空间。

```
OPENSHIFT_KUBE_PING_LABELS=labelKey=labelValue
```

4. 为 pod 运行的服务帐户授予授权，以便它能够访问 Kubernetes REST API。例如，为 ***datagrid-service-account*** 授予授权，如下所示：

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

5. 确保 **8888** 定义为 pod 容器上的 ping 端口，如下所示：

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

11.2. 配置 JGROUPS 加密

用于 OpenShift 的数据网格使用 JGroups 技术保护集群服务器间的流量，并带有以下选项：

身份验证

使用需要节点在加入集群时使用密码进行身份验证的Groups **AUTH** 协议。
您可以使用 **JGROUPS_CLUSTER_PASSWORD** 环境变量配置身份验证。此环境变量设置在加入集群时要使用的节点密码。整个集群中的密码必须相同。

对称加密

使用 JGroups **SYM_ENCRYPT** 协议保护来自 JGroups 密钥存储的流量(**.jceks**)。这是默认的加密协议。
JGroups **AUTH** 协议具有对称加密的可选功能。

JGroups 密钥存储包含集群中各个节点用于安全通信的凭据。

非对称加密

使用 JGroups **ASYM_ENCRYPT** 协议保护带有公钥/私钥加密的流量。
利用非对称加密，需要 JGroups **AUTH** 协议。

协调器节点会生成 secret 密钥。当节点加入集群时，它会从协调器请求 secret 密钥并提供其公钥。协调器使用公钥加密 secret 密钥并将其返回到节点。然后，节点会解密并安装 secret，以便它可以安全地与集群中的其他节点通信。

11.2.1. 设置对称加密

要使用对称加密，请执行以下操作：

1. 创建一个Groups 密钥存储(**.jceks**)，其中包含用于加密流量的凭据。
   您可以使用 Java keytool 生成 JGroups 密钥存储。
2. 将 JGroups 密钥存储作为机密部署到 OpenShift。
   a. 登录您的 OpenShift 集群。
   b. 为 JGroups 密钥存储创建机密。例如，若要从名为 **jgroups.jceks** 的密钥存储创建名为 **jgroups-secret** 的 secret，请执行以下操作：

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

   c. 将 secret 链接到 default 服务帐户。

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

   d. 将 secret 挂载到容器。

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

3. 将集群中每个节点的 **JGROUPS_ENCRYPT_PROTOCOL** 环境变量设置为 **SYM_ENCRYPT**。
4. 配置集群中的每个节点，使其使用具有以下环境变量的 JGroups 密钥存储：

JGROUPS_ENCRYPT_KEYSTORE
Specifes the JGroups 密钥存储用于加密集群流量。

JGROUPS_ENCRYPT_KEYSTORE_DIR
指定 JGroups 密钥存储所在的目录。

JGROUPS_ENCRYPT_SECRET
匹配密钥存储的 OpenShift 机密。

JGROUPS_ENCRYPT_NAME
匹配密钥存储的用户名。

JGROUPS_ENCRYPT_PASSWORD
匹配密钥存储密码。

5. 如果需要，在使用 **JGROUPS_CLUSTER_PASSWORD** 环境变量加入集群时，为节点设置密码。

11.2.2. 设置 Asymmetric Encryption

要使用非对称加密，请执行以下操作：

1. 使用 **JGROUPS_CLUSTER_PASSWORD** 环境变量配置身份验证。
2. 将集群中每个节点的 **JGROUPS_ENCRYPT_PROTOCOL** 环境变量设置为 **ASYM_ENCRYPT**。

第 12 章 为 OPENSHIFT 自定义数据网格

将 Data Grid 镜像与自定义配置一起使用，可通过 Source-to-Image(S2I)流程或通过 **ConfigMap** API 使用。

注意

红帽鼓励您使用 Data Grid for OpenShift 镜像和 **datagrid-service** 和 **cache-service** 模板来创建 Data Grid 集群。虽然可以使用自定义配置创建 Data Grid pod，但 **datagrid-service** 和 **cache-service** 是高性能设计的，它适用于不需要配置的各种用例。

请参阅为 OpenShift 服务设置数据网格， 了解如何快速创建数据网格集群。

Source-to-Image(S2I)

使用 S2I 流程和二进制构建来创建自定义 Data Grid 镜像。
在 **openshift-clustered.xml** 中添加缓存定义和端点配置，然后使用 S2I 功能构建使用该配置文件的自定义 Data Grid 镜像。然后，您可以根据需要创建带有镜像的 Data Grid pod。

要修改配置，您必须构建新镜像。但是，当您重建自定义镜像时，Data Grid pod 会自动使用新配置更改重新部署。

ConfigMap API

使用 Red Hat OpenShift **ConfigMap** API 动态配置 Data Grid pod。
在 **standalone.xml** 中定义自定义配置，该配置映射到命名空间中的 **ConfigMap** 对象，作为 Data Grid pod。您可以修改 Data Grid 配置，然后重新部署 pod 以加载配置更改。但是，当修改 **standalone.xml** 时，Data Grid pod 不会自动重新部署。您必须手动重新部署 pod 才能加载配置更改。

- 在创建 Data Grid pod 前，您应该在 **standalone.xml** 中显式定义所有缓存和端点配置。除非在部署前存在占位符，否则缓存和端点配置的环境变量不会生效。例如，以下是 **JGROUPS_PING_PROTOCOL** 的占位符：

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

  请参阅 **cluster-openshift.xml** 查看可用的占位符。

- 要加密客户端到服务器流量，您必须在 **standalone.xml** 中配置服务器身份。
- 对于通过 **ConfigMap** API 自定义的 OpenShift pod， **DATAGRID_SPLIT** 环境变量不会与 Data Grid 生效。这些 pod 无法将共享持久性卷用于 **DATAGRID_SPLIT**。

12.1. 克隆数据网格示例

1. 克隆用于 OpenShift 镜像存储库的 Data Grid。

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

2. 查看 **docs/examples** 目录的内容，例如：

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

$ tree
```

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

S2I

将 **clustered-openshift.xml** 注入 **${JBOSS_HOME}/standalone/configuration/** 用于 OpenShift 镜像的 Data Grid。

提示

将自定义 **logging.properties** 和 **application-role.properties** 添加到 **配置** 目录中，以便在构建自定义镜像时包含它们。

ConfigMap

将 **standalone.xml** 映射到 OpenShift 镜像的 Data Grid 内部的 **/opt/datagrid/standalone/configuration/user** 目录中。

12.2. 使用自定义配置创建 S2I 构建

12.2.1. 设置数据网格镜像

您需要 Data Grid for OpenShift 镜像作为自定义来源。

1. 确认适用于 OpenShift 镜像的 Data Grid 是否可用。

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

2. 如果镜像不可用，请创建一个镜像流并导入它。

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

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

12.2.2. 创建自定义数据网格镜像

1. 使用 Data Grid 为 OpenShift 镜像创建一个新的二进制构建。

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

2. 导航到 **s2i** 目录，以便您可以传递 **--from-dir="."**。您必须包含 **配置目录**，以便 S2I 进程可以检测您的自定义配置。
要使用配置示例，请执行以下操作：

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

3. 使用自定义配置为 OpenShift 镜像构建 Data Grid。

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

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

...
Push successful
```

4. 检查您的镜像是否可用。

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

12.2.3. 验证自定义数据网格镜像

1. 确认构建 Pod 正在运行。

```
$ oc get pods

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

2. 使用您的自定义配置创建 Data Grid 应用程序。

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

3. 等待您的自定义数据网格应用程序开始运行。

```
$ oc get pods -w

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

4. 远程访问自定义 Data Grid pod 的 bash shell。

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

5. 查看 **cluster-openshift.xml** 以验证配置，例如：

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

如果 **clustered-openshift.xml** 包含您的自定义配置，则 Data Grid pod 使用它。您可以使用 Data Grid 命令行界面验证您的配置，例如：

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

6. 验证配置后，结束远程会话。

```
$ exit
```

12.3. 使用 CONFIGMAP API 为 OPENSHIFT POD 创建自定义 DATA GRID

1. 为您的 Data Grid 创建自定义模板，以用于 OpenShift pod。
   a. 在您的环境中公开所需的端口和服务。
   b. 在自定义模板中添加 **configMap** 对象。
   c. 为容器添加位于 **/opt/datagrid/standalone/configuration/user** 的配置卷。
   d. 将您的自定义模板导入到 OpenShift。
   要使用示例模板，请执行以下操作：

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

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

2. 在 OpenShift 项目中创建 ConfigMap，例如：

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

3. 使用自定义配置创建 Data Grid pod。

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

其中：

- **APPLICATION_NAME** 是示例模板中的必需参数，默认为 **custom-datagrid**。
- **USER_CONFIG_MAP=true** 将 ConfigMap 应用到 Data Grid pod。这在示例模板中设置，如下所示：

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

12.3.1. 使用 ConfigMap API 验证 OpenShift Pod 的自定义 Data Grid

1. 等待您的自定义数据网格应用程序开始运行。

```
$ oc get pods -w

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

2. 检查容器日志。

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

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

提示

尝试自定义 Data Grid Service Deployment Quickstart 教程。

12.4. 配置持久数据源

通过数据网格，您可以将存储在缓存中的数据保存到数据源。Red Hat Data Grid 有两个类型的数据源：

- 在 OpenShift 上运行的内部数据源。这些数据源可通过 Red Hat Container Registry 提供，且不需要配置额外的环境文件。

注意

内部数据源包括 PostgreSQL、MySQL 和 MongoDB。但是，适用于 OpenShift 的 Red Hat Data Grid 目前仅支持 PostgreSQL 和 MySQL。

- 未在 OpenShift 上运行的外部数据源。您必须使用添加到 OpenShift Secret 的环境文件来配置这些外部数据源。

12.4.1. 配置内部数据源

DB_SERVICE_PREFIX_MAPPING 环境变量为内部数据源定义 JNDI 映射。

您可以将多个 JNDI 映射定义为 **DB_SERVICE_PREFIX_MAPPING** 环境变量的逗号分隔值。当您为 OpenShift 镜像运行 Data Grid 时，启动脚本会为每个 JNDI 映射创建一个独立数据源。OpenShift 的 Data Grid 会自动发现每个数据源。

要定义 JNDI 映射，以以下格式指定环境变量的值：

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

- **${POOL_NAME}** 是数据源的 **pool-name** 属性。使用有意义的、易于识别的任何字母数字值。该值不能包含特殊字符。同样，该值必须包含小写字母字符。
- **${DATABASE_TYPE}** 指定要使用的数据库驱动程序。该值必须仅包含小写字符。

注意

只有 **mysql** 和 **postgresql** 是 **{$DATABASE_TYPE}** 的支持值。

- **${PREFIX}** 用于配置数据源的环境变量的名称。

12.4.1.1. 单一数据源示例

如果您将 **test-postgresql=TEST** 指定为 **DB_SERVICE_PREFIX_MAPPING** 环境变量的值，它会创建一个具有以下名称的数据源：

java:jboss/datasources/test_postgresql

在为数据源指定其他环境变量时，您必须使用 **TEST** 前缀。例如，若要设置用户名和密码，可使用 **TEST_USERNAME** 和 **TEST_PASSWORD** 作为环境变量。

12.4.1.2. 多个数据源示例

如果您指定了 **cloud-postgresql=CLOUD,test-mysql=TEST_MYSQL** 作为 **DB_SERVICE_PREFIX_MAPPING** 环境变量的值，它会使用以下名称创建两个数据源：

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

为数据源指定其他环境变量时，您必须使用 **TEST_MYSQL** 前缀来配置 MySQL 数据源。例如，使用 **TEST_MYSQL_USERNAME** 作为环境变量来指定用户名。

同样，您必须使用 **CLOUD** 前缀来配置 PostgreSQL 数据源。例如，使用 **CLOUD_USERNAME** 作为环境变量来指定用户名。

12.4.2. 配置外部数据源

要使用外部数据源，您可以定义自定义镜像模板，然后使用 Source-to-Image(S2I)构建工具来创建镜像。S2I 是一个框架，将应用程序源代码用作输入，并生成一个运行编译的应用程序的新镜像作为输出。

以下高级别步骤提供进程概述：

1. 指定镜像模板 JSON 中的 **CUSTOM_INSTALL_DIRECTORIES** 环境变量。此变量定义 S2I 工件所在的位置，如下例所示：

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

2. 在该目录中创建一个 **install.sh** 脚本。此脚本为镜像中外部数据源安装模块和驱动程序。以下是 **install.sh** 脚本示例：

```
#!/bin/bash

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

# Directory where this script is located
injected_dir=$1

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

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

3. 包括一个 **模块** 子目录，其中包含 **module.xml** 文件和数据源的驱动程序。生成的镜像使用模块来加载类并定义依赖项。
   例如，您计划将 Derby 用作外部数据源。您需要获取一个驱动，如 **derby-10.12.1.1.jar**，并将其放置在以下目录中： **modules/org/apache/derby/main/**

   在同一目录中，您还需要创建一个 **module.xml** 文件，该文件将驱动程序定义为资源并声明依赖项。

   以下是 **module.xml** 文件示例：

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

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

4. 在 driver **.property 环境变量文件中定义驱动程序** 配置属性。
以下是 **驱动程序.property 文件示例** ：

```
#DRIVERS
DRIVERS=DERBY

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

5. 在构建和部署镜像后，为数据源指定环境变量。
以下示例显示了带有 **DATASOURCES** 环境变量的数据源定义：

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

12.4.3. 数据源环境变量

DB_SERVICE_PREFIX_MAPPING

定义要配置的数据源的逗号分隔列表。
例如： **DB_SERVICE_PREFIX_MAPPING=test-mysql=TEST_MYSQL**.如需更多信息，请参阅配置 Persistent Datasources。

${NAME}_${DATABASE_TYPE}_SERVICE_HOST

定义数据源 **connection_url** 属性的数据库服务器主机名或 IP。
例如： **EXAMPLE_MYSQL_SERVICE_HOST=192.0.2.0**

${NAME}_${DATABASE_TYPE}_SERVICE_PORT

定义数据库服务器端口。

${PREFIX}_USERNAME

定义数据源的用户。

${PREFIX}_PASSWORD

定义数据源的密码。

${PREFIX}_DATABASE

定义数据源的数据库名称。
例如，**CLOUD_DATABASE=myDatabase**。

${PREFIX}_DRIVER

为数据源定义 Java 数据库驱动程序。
例如，**CLOUD_DRIVER=postgresql**

${PREFIX}_BACKGROUND_VALIDATION

指定后台线程是否在使用前验证数据库连接。值为 **true** 或 **false** （默认）。默认情况下启用 < **validate-on-match&** gt; 方法。

${PREFIX}_BACKGROUND_VALIDATION_MILLIS

如果您将 **${PREFIX}_BACKGROUND_VALIDATION** 环境变量设置为 **true**，则指定以毫秒为单位验证的频率。默认值为 **10000**。

${PREFIX}_CONNECTION_CHECKER

指定用于验证与数据库的连接的连接检查器类。
例
如，**CLOUD_CONNECTION_CHECKER=org.jboss.jca.adapters.jdbc.extensions.postgres.PostgreSQLValidConnectionChecker**

${PREFIX}_EXCEPTION_SORTER

指定在致命数据库连接异常时检测和清理异常分类器类。
例
如，**CLOUD_EXCEPTION_SORTER=org.jboss.jca.adapters.jdbc.extensions.mysql.MySQLExceptionSorter**

${PREFIX}_JNDI

定义数据源的 JNDI 名称。
默认为 **java:jboss/datasources/<name>_<database_type>**。启动脚本自动从 **DB_SERVICE_PREFIX_MAPPING** 环境变量生成值。

例如，**CLOUD_JNDI=java:jboss/datasources/test-postgresql**

${PREFIX}_JTA

定义非 XA 数据源的 Java 事务 API(JTA)选项。值为 **true** （默认）或 **false**。

${PREFIX}_MAX_POOL_SIZE

定义数据源的最大池大小。

${PREFIX}_MIN_POOL_SIZE

定义数据源的最小池大小。

${PREFIX}_NONXA

将数据源定义为非 XA 数据源。值为 **true** 或 **false** （默认）。

${PREFIX}_TX_ISOLATION

定义数据库的 java.sql.Connection 事务隔离级别。
例如，**CLOUD_TX_ISOLATION=TRANSACTION_READ_UNCOMMITTED**

${PREFIX}_URL

定义非 XA 数据源的连接 URL。
如果没有指定连接 URL，启动脚本会自动从其他环境变量生成它，如下所示：
url="jdbc:${DRIVER}://${HOST}:${PORT}/${DATABASE}".

但是，启动脚本只为 PostgreSQL 和 MySQL 等内部数据源构建正确的连接 URL。如果使用任何其他非 XA 数据源，则必须指定连接 URL。

例如，**CLOUD_URL=jdbc:postgresql://localhost:5432/postgresdb**

${PREFIX}_XA_CONNECTION_PROPERTY_<PROPERTY_NAME>

定义 XA 数据源的连接属性。
请参考相应的驱动程序文档，以使数据源查找可在连接上设置哪些 XA 属性。

例
如，**CLOUD_XA_CONNECTION_PROPERTY_DatabaseName=/opt/eap/standalone/data/databases/db/accounts**

这个示例在配置中添加以下内容：

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

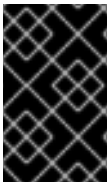
第 13 章 嵌入式数据网格(INDEX MODE)

在自定义 OpenShift 应用程序中嵌入数据网格或以库模式运行，仅用于特定用途：

- 在自定义 Java 应用程序中使用本地或分布式缓存来保持对缓存生命周期的完全控制。另外，在使用仅适用于嵌入式数据网格（如分布式流）的功能时。
- 减少网络延迟以提高缓存操作速度。值得注意的是，Hot Rod 协议提供满足标准客户端-服务器架构同等性能的近缓存功能。

嵌入数据网格也有以下限制：

- 目前不支持对缓存存储进行持久性。您只能内嵌用于内存缓存的 Data Grid。
- 集群只支持 **DNS_PING**。
- ping 端口支持 **TCP**。
- 嵌入式数据网格不支持 **UDP**。



重要

红帽强烈不建议嵌入了 Data Grid，以构建自定义缓存服务器来处理远程客户端请求。使用 **datagrid-service** 服务器实现。它通过性能改进和安全修复，对定期更新获得全面支持和优势。

提示

尝试 嵌入式数据网格快速入门。

第 14 章 针对 OPENSHIFT 的 DATA GRID 故障排除

14.1. 启动命令行界面(CLI)

访问 Data Grid Management CLI 以对 OpenShift 的 Data Grid 进行故障排除，如下所示：

1. 打开与正在运行的 pod 的远程 shell 会话。

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

2. 从远程 shell 启动 Data Grid CLI。

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

注意

Data Grid Management CLI 与运行它的 pod 绑定。如果容器重启，您通过 CLI 进行的更改不会保存。

14.2. 查看 POD 日志

运行以下命令来查看正在运行的 pod 的日志信息：

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

第 15 章 参考

15.1. 探测

用于 OpenShift 的数据网格提供存活度探测和就绪度探测，以执行容器健康检查。

存活度探测

存活度探测位于 **/opt/datagrid/bin/livenessProbe.sh** 的容器中。
存活度探测测试服务器状态并在发生以下事件时重启 pod：

- 用于 OpenShift 引导的数据网格出错。
- 自定义部署配置无法成功部署。
- 一个或多个缓存无法实例化，这通常在缓存配置无效时发生。

就绪度探测

就绪度探测位于 **/opt/datagrid/bin/readinessProbe.sh** 的容器中。
就绪度探测(Readiness probe)决定 pod 是否准备好接收请求并检查 Data Grid 缓存级别 **MBeans** 以确保：

- 所有缓存实例都会被初始化。
- 如果使用分布式缓存模式，所有缓存实例都加入了集群。
- 初始状态转移已完成。如果状态转移正在进行，则 pod 不会标记为 ready。
- 缓存管理器中的所有缓存实例都在运行。

要配置自定义部署以使用存活度探测和就绪度探测，请运行以下命令：

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

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

15.2. 端口

用于 OpenShift 的数据网格使用以下端口：

端口号	协议	使用
8080	TCP	HTTP 访问
8443	TCP	HTTPS 访问

端口号	协议	使用
8888	TCP	JGroups Ping
11222	TCP	热访问

data Grid 部署配置模板还使用以下端口：

端口号	协议	使用
11211	TCP	memcached 访问
11333	TCP	外部 Hot Rod 访问
8778	TCP	远程 JMX 访问

注意

如果您使用部署配置模板设置 **GPOROD_SERVICE_NAME** 环境变量，则 Hot Rod 外部连接器为端点返回 **${service_name}:11333**。

15.3. 管理控制台

Red Hat OpenShift 不支持 Data Grid 管理控制台。

要监控事件并获取 OpenShift 集群的 Data Grid 统计信息，您应该使用 Prometheus。请参阅 设置监控。

您还可以使用管理 CLI 对 OpenShift pod 的 Data Grid 进行故障排除。请参阅 启动命令行界面。

15.4. 环境变量

15.4.1. 监控和日志记录

AB_PROMETHEUS_ENABLE

允许您收集 JMX 指标来监控和分析数据网格。默认值为 **false**。将值设为 **true** 来启用使用默认 Prometheus 代理的监控。
Prometheus Operator 必须已安装并正在运行。您还必须 设置 Monitoring。

AB_PROMETHEUS_JMX_EXPORTER_PORT

定义数据网格发布 JMX 指标的端口。默认值为 **9779**。

LOGGING_CATEGORIES

调整 Data Grid 捕获日志消息的类别和级别，例如：

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

日志类别与 Java 软件包名称对应，并使用标准日志级别：
TRACE、**DEBUG**、**INFO**、**WARN**、**ERROR** 和 **FATAL**。

重要

如果指定了 **LOGGING_CATEGORIES**，Data Grid 不会设置以下默认日志记录器：对于您没有通过 **LOGGING_CATEGORIES** 明确指定的软件包的所有软件包，Data Grid 会使用默认的 **INFO** 级别。

表 15.1. 默认日志记录器

类别	级别
com.arjuna	**WARN**
sun.rmi	**WARN**
org.jboss.as.config	**DEBUG**

15.4.2. 容器

用户名

在被授权访问数据的安全域中创建用户。

注意

默认情况下，Hot Rod 端点使用 **ApplicationRealm** 安全域，而 REST 端点则使用 **jdg-openshift** 安全域。

密码

指定用户的密码。

DATAGRID_SPLIT

确定每个节点的数据目录应该被分割到网格中。值为 **true** 或 **false** （默认）。
如果将值设为 **true**，还必须配置在 **/opt/datagrid/standalone/partitioned_data** 上挂载的持久性卷。

JAVA_OPTS_APPEND

在启动时将选项附加到 **JAVA_OPTS** 环境变量中。
例如： **JAVA_OPTS_APPEND=-Dfoo=bar**

OPENSHIFT_KUBE_PING_LABELS

指定集群标签选择器。
例如，**OPENSHIFT_KUBE_PING_LABELS=application=eap-app**

OPENSHIFT_KUBE_PING_NAMESPACE

指定集群项目命名空间。

TRANSPORT_LOCK_TIMEOUT

设定等待获取分布式锁定的时间。默认值为 **240000**。
数据网格使用分布式锁定在状态传输或重新哈希期间保持一致性的事务日志，这意味着一次只有一个缓存才能执行状态传输或重新哈希。这个约束就位，因为可以涉及到多个缓存。

15.4.3. 缓存

使用 cache-service 和 datagrid-service创建和配置缓存

不要使用环境变量来使用 **cache-service** 或 **datagrid-service** 创建并配置缓存。

这些环境变量仅用于部署配置模板，且已被弃用。

您应该通过 Hot Rod 端点动态使用 **cache-service** 和 **datagrid-service** 进行远程创建缓存。如需更多信息，请参阅 远程创建缓存。

CACHE_NAMES

在配置中定义缓存实例。
如果您使用 Data Grid 部署配置模板，且您不定义任何缓存实例，则启动脚本会在 **SYNC** 模式中添加了一个默认的分布式缓存。

提示

为您的配置中的每个缓存实例指定一个唯一的名称。使用下划线字符(_)和描述性标签可帮助您区分缓存实例。这样可确保在应用特定于缓存的配置时没有冲突。

例如，**CACHE_NAMES=addressbook,addressbook_indexed**

CACHE_CONTAINER_START

配置缓存容器启动方式。指定以下之一：

- 在服务或部署请求时，**LAZY** 启动 cache 容器。这是默认值。
- **EAGER** 在服务器启动时启动缓存容器。

CACHE_CONTAINER_STATISTICS

配置缓存容器以收集统计信息。值为 **true** （默认）或 **false**。您可以将值设为 **false** 以提高性能。

DEFAULT_CACHE

为缓存容器设置默认缓存。

15.4.3.1. 缓存容器安全配置

CONTAINER_SECURITY_CUSTOM_ROLE_MAPPER_CLASS

指定角色映射程序的自定义主体类。
例
如，**CONTAINER_SECURITY_CUSTOM_ROLE_MAPPER_CLASS=com.acme.CustomRoleMapper**

CONTAINER_SECURITY_ROLE_MAPPER

使用以下值为此缓存容器设置 role mapper：

- **identity-role-mapper** 将 Principal name 用作角色名称。如果您没有指定服务并使用 **CONTAINER_SECURITY_ROLES** 环境变量来定义角色名称，则这是默认角色映射程序。
- 如果主体名称为可辨识的名称(DN)，**common-name-role-mapper** 会将 Common Name(CN)用作角色名称。例如，DN **cn=managers、ou=body,dc=example,dc=com** 映射到 **manager** 角色名称。

- **cluster-role-mapper** 使用 **ClusterRegistry** 来存储主体名称到角色映射。
- **custom-role-mapper** 获取 **org.infinispan.security.impl.PrincipalRoleMapper** 接口的完全限定类名称。

CONTAINER_SECURITY_ROLES

定义角色名称并为它们分配权限。
例如，**CONTAINER_SECURITY_ROLES=admin=ALL,reader=READ,writer=WRITE**

15.4.3.2. 指定缓存配置

以大写字母（所有大写）中的环境变量的前缀指定缓存名称，否则配置不会生效。

例如，您创建两个单独的缓存实例： **MyCache** 和 **MYCACHE**。然后，您将 **MyCache_CACHE_TYPE=replicated** 设置为配置 **MyCache** 实例。这个配置不会生效。但是，如果您设置了 **MYCACHE_CACHE_TYPE=replicated**，则配置对 **MyCache** 和 **MYCACHE** 实例都生效。

${CACHE_NAME}_CACHE_TYPE

决定此缓存应该是分布式还是复制。您可以指定 **分布式** （默认） **或复制**。

${CACHE_NAME}_CACHE_START

配置缓存的启动方式。指定以下之一：

- 在服务或部署请求时，**LAZY** 启动缓存。这是默认值。
- **EAGER** 在服务器启动时启动缓存。

${CACHE_NAME}_CACHE_BATCHING

启用这个缓存的调用批处理。值为 **true** 或 **false** （默认）。

${CACHE_NAME}_CACHE_STATISTICS

配置缓存以收集统计信息。值为 **true** （默认）或 **false**。您可以将值设为 **false** 以提高性能。

${CACHE_NAME}_CACHE_MODE

设置集群缓存模式。指定以下之一：

- 用于异步操作的 **ASYNC**。
- 用于同步操作的 **SYNC**。

${CACHE_NAME}_CACHE_QUEUE_SIZE

设置当缓存为 **ASYNC** 模式时清空复制队列的阈值。默认值为 **0**（禁用lushing）。

${CACHE_NAME}_CACHE_QUEUE_FLUSH_INTERVAL

指定在 **ASYNC** 模式中清除复制队列的线程的 wakeup 时间（以毫秒为单位）。默认值为 **10**。

${CACHE_NAME}_CACHE_REMOTE_TIMEOUT

以毫秒为单位指定超时，以便在在 **SYNC** 模式中进行远程调用时等待确认。如果达到超时，则远程调用将中止并抛出异常。默认值为 **17500**。

${CACHE_NAME}_CACHE_OWNERS

指定每个缓存条目的集群范围副本数。默认值为 **2**。

${CACHE_NAME}_CACHE_SEGMENTS

指定每个集群的哈希空间片段数量。建议的值是 **10 * 集群大小**。默认值为 **80**。

${CACHE_NAME}_CACHE_L1_LIFESPAN

指定 L1 缓存中的最大生命周期span（以毫秒为单位）。默认值为 **0**（禁用L1）。

${CACHE_NAME}_CACHE_MEMORY_EVICTION_TYPE

定义缓存中的条目的最大限值。您可以设置以下值：

- **COUNT** 测量缓存中的条目数量。当计数超过最大值时，Data Grid 会驱除未使用的条目。
- **MEMORY** 测量缓存中所有条目的内存量。当内存量超过最大值时，Data Grid 会驱除未使用的条目。

${CACHE_NAME}_CACHE_MEMORY_STORAGE_TYPE

定义数据网格如何在缓存中存储条目。您可以设置以下值：

存储类型	描述	驱除类型	策略
对象	将条目作为对象存储在 Java 堆中。这是默认的存储类型。	**数量**	TinyLFU
二进制（二进制）	将条目作为 **bytes[]** 存储在 Java 堆中。	**COUNT** 或 **MEMORY**	TinyLFU
off-heap	将条目作为 **bytes[]** 存储在 Java 之外的原生内存中。	**COUNT** 或 **MEMORY**	LRU

${CACHE_NAME}_CACHE_MEMORY_EVICTION_SIZE

配置驱除启动前缓存的大小。将值设置为大于零的数字。

- 对于 **COUNT**，大小是缓存在驱除开始前可以容纳的最大条目数。
- 对于 **MEMORY**，其大小是缓存在驱除启动前可以从内存中获取的最大字节数。例如，值设为 **10000000000** 为 10 GB。
  尝试不同的缓存大小以确定最佳设置。缓存大小过大可能会导致 Data Grid 耗尽内存。同时，缓存大小太小了可用内存。

注意

如果您配置 JDBC 存储，当您将驱除大小设置为大于零的值时，会自动启用 passivation。

${CACHE_NAME}_CACHE_MEMORY_EVICTION_STRATEGY

控制数据网格执行驱除的方式。您可以设置以下值：

策略	描述
NONE	数据网格不会驱除条目。这是默认设置，除非您配置驱除。

策略	描述
删除	数据网格从内存中删除条目，以便缓存不会超过配置的大小。这是配置驱除时的默认设置。
手动	数据网格不执行驱除。通过从 **Cache** API 调用 **evict（）** 方法手动进行驱除。
例外	如果这样做超过配置的大小，则 data Grid 不会将新条目写入缓存中。Data Grid 丢弃了 **ContainerFullException**，而不是向缓存写入新条目。

${CACHE_NAME}_CACHE_MEMORY_OFF_HEAP_ADDRESS_COUNT

指定哈希映射中可用的指针数量，以防止在使用 **OFFHEAP** 存储时发生冲突。防止哈希映射中的冲突提高了性能。
将值设为大于缓存条目数的数字。默认情况下，**address-count** 为 2^20，或 1048576。参数始终向上取为 2 的电源。

${CACHE_NAME}_CACHE_EXPIRATION_LIFESPAN

指定条目在集群范围过期后的最大生命周期span（以毫秒为单位）。默认值为 **-1（** 条目永不过期）。

${CACHE_NAME}_CACHE_EXPIRATION_MAX_IDLE

指定缓存条目在缓存中维护的最大空闲时间（以毫秒为单位）。如果超过空闲时间，则条目会在整个集群到期。默认值为 **-1（** 禁用过期）。

${CACHE_NAME}_CACHE_EXPIRATION_INTERVAL

指定间隔（以毫秒为单位），以毫秒为单位，从内存和任何缓存存储中清除过期的条目。默认值为 **5000**。设置 **-1** 可禁用过期。

${CACHE_NAME}_JDBC_STORE_TYPE

设置要配置的 JDBC 存储的类型。您可以设置以下值：

- **字符串**
- **二进制（二进制）**

${CACHE_NAME}_JDBC_STORE_DATASOURCE

定义数据源的 jndiname。
例如： **MYCACHE_ JDBC_STORE_DATASOURCE=java:jboss/datasources/ExampleDS**

${CACHE_NAME}_KEYED_TABLE_PREFIX

定义在生成缓存条目表名称时使用的缓存名称前的前缀。defaule 值为 **ispn_entry**。

${CACHE_NAME}_CACHE_INDEX

设置缓存的索引模式。您可以设置以下值：

- **NONE** 这是默认值。
- **本地**

- **ALL**

${CACHE_NAME}_INDEXING_PROPERTIES

指定要传递给索引系统的属性列表。
例如： **MYCACHE_INDEXING_PROPERTIES=default.directory_provider=ram**

${CACHE_NAME}_CACHE_SECURITY_AUTHORIZATION_ENABLED

为此缓存启用授权检查。值为 **true** 或 **false** （默认）。

${CACHE_NAME}_CACHE_SECURITY_AUTHORIZATION_ROLES

设置访问此缓存所需的角色。
例如： **MYCACHE_CACHE_SECURITY_AUTHORIZATION_ROLES=admin, reader, writer**

${CACHE_NAME}_CACHE_PARTITION_HANDLING_WHEN_SPLIT

配置策略，以便在网络事件相互隔离节点时处理集群中节点间分区的策略。分区作为独立集群运行，直到 Data Grid 合并缓存条目以调整单个集群。您可以设置以下值：

分区处理策略	描述
ALLOW_READ_WRITES	来自任何分区的节点都可以读取或写入缓存条目。这是默认值。
DENY_READ_WRITES	如果出现以下情况，节点进入 degraded 模式： * 分区中的一个或者多个散列空间片段没有所有者。所有者是缓存条目的集群范围副本数。 *** 分区从最新的稳定集群拓扑中少于一半的节点。** **在降级模式中，只有同一分区中的节点才能读取或写入缓存条目。缓存条目的所有所有者或副本必须存在于同一分区中，否则，读写操作将失败，并显示 AvailabilityException。**

分区处理策略	描述
ALLOW_READS	节点与 **DENY_READ_WRITES** 策略类似。来自任何分区的节点都可以读取缓存条目。 在降级模式中，只有同一分区中的节点才能写入缓存条目。缓存条目的所有所有者或副本必须存在于同一分区中，否则写入操作会失败并显示 **AvailabilityException**。

${CACHE_NAME}_CACHE_PARTITION_MERGE_POLICY

配置 Data Grid 如何在合并分区时解析缓存条目间的冲突。您可以设置以下值：

合并策略	描述
NONE	合并分区时不会解决冲突。这是默认值。
PREFERRED_ALWAYS	始终使用 **preferredEntry**。**preferredEntry** 是缓存条目的主副本，它位于包含最多节点的分区中。如果节点数量相等，则 **preferredEntry** 是分区中带有最高拓扑 ID 的缓存条目，这意味着拓扑更为近期。
PREFERRED_NON_NULL	如果值为(non-null)，请使用 **preferredEntry**。如果 **preferredEntry** 没有值，请使用 **otherEntries** 中定义的第一个条目。
REMOVE_ALL	如果存在冲突，从缓存中删除条目（键和值）。

${CACHE_NAME}_STATE_TRANSFER_TIMEOUT

设置时间（以毫秒为单位），以等待集群中其他缓存实例将状态转移到缓存。如果其他缓存实例在超时发生前没有传输状态，应用程序会抛出异常并中止启动。默认值为 **240000** （4 分钟）。
您必须使用自定义模板来设置此环境变量。如果您在 OpenShift 模板的默认 Data Grid 中设置状态传输超时，则不会生效。

15.4.4. 安全域

SECDOMAIN_NAME

定义其他安全域。例如： **SECDOMAIN_NAME=myDomain**

SECDOMAIN_PASSWORD_STACKING

启用密码 staking 模块并设置 **useFirstPass** 选项。值为 **true** 或 **false** （默认）。

SECDOMAIN_LOGIN_MODULE

指定要使用的登录模块。默认值为 **UsersRoles**。

SECDOMAIN_USERS_PROPERTIES

指定包含用户定义的属性文件。默认值为 **users.properties**。

SECDOMAIN_ROLES_PROPERTIES

指定包含角色定义的属性文件。默认值为 **roles.properties**。

15.4.5. Endpoints

客户端可以通过您在缓存配置中定义的 REST、Hot Rod 和 Memcached 端点访问数据网格。

在与 OpenShift Data Grid 相同的项目中运行的客户端可以通过 Hot Rod 访问缓存并接收完整的集群视图。这些客户端也可以使用一致的哈希功能。

但是，当客户端在不同项目中为 OpenShift 的 Data Grid 中运行到 Data Grid 时，他们需要使用一个 OpenShift 服务从外部公开 Hot Rod 端点来访问 Data Grid 集群。根据您的网络配置，客户端可能无法访问某些 pod，且必须使用 **BASIC** 客户端智能。在这些情况下，客户端可能需要额外的网络跃点访问数据，这会增加网络延迟。

对 OpenShift 中运行的客户端的外部访问权限需要使用 passthrough 加密终止的路由。客户端还必须使用 **BASIC** 客户端智能和完全限定域名作为 TLS/SNI 主机名。另外，您还可以在外部提供的 Load Balancer 服务后公开 Data Grid 集群。

使用以下环境变量配置端点：

INFINISPAN_CONNECTORS

定义要配置的连接器的逗号分隔列表。默认为热线 **,memcached** ， **剩余** 部分。如果在缓存上启用了授权或身份验证，则您应该删除 **memcached**，因为此协议本质上是不安全的。

MEMCACHED_CACHE

为 Memcached 连接器设置缓存名称。如果您没有使用 **CACHE_NAMES** 环境变量指定缓存名称，则默认为 **memcached**。

HOTROD_SERVICE_NAME

为外部 Hot Rod 连接器定义 OpenShift 服务的名称。
只有在定义此环境变量时，只有部署配置模板可以使用外部 Hot Rod 连接器。**cache-service** 和 **datagrid-service** 不使用外部 Hot Rod 连接器。

例如，如果您设置了 **HOTROD_SERVICE_NAME=DATAGRID_APP_HOTROD**，则 Hot Rod 外部连接器返回 **DATAGRID_APP_HOTROD:11333**。

REST_STORE_AS_STRING

指定在通过 REST API 写入缓存时，如果 Data Grid 将条目保存为 Java 字符串。值为 **true** 或 **false**（默认）。
如果您要从上一版本升级镜像并计划读取持久的缓存条目，则将值设为 **true**。

注意

Data Grid 版本 7.1 及更早的版本： 当您通过 REST 端点将条目写入缓存时，Data Grid 会将它们存储为 Java 字符串。

Data Grid version 7.2 及更新的版本： Data Grid 将缓存条目存储为 **bytes[]**，以启用客户端和协议间的数据互操作性。

如果您将 OpenShift 镜像的 Data Grid 升级到 7.2 或更高版本，当您试图读取一直到数据存储的缓存条目时，Data Grid 会返回 null 值。要解析 null 值，请设置 **REST_STORE_AS_STRING=true**。