Red Hat build of Keycloak 24.0

高可用性指南

Last Updated: 2024-06-28

法律通告

摘要

本指南包含管理员用来配置和使用红帽构建的 Keycloak 24.0 来实现高可用性的信息。

目录

第 1 章 多站点部署

红帽构建的 Keycloak 支持由多个红帽构建的 Keycloak 实例组成的部署，它们通过其 Infinispan 缓存相互连接；负载均衡器可以在这些实例间平均分配负载。这些设置适用于单一站点上的透明网络。

红帽构建的 Keycloak 高可用性指南是进一步介绍多个站点之间的设置的步骤。虽然此设置增加了额外的复杂性，但有些环境中可能需要额外的高可用性。

不同的章节介绍了必要的概念和构建块。对于每个构建块，蓝图演示了如何设置完全功能的示例。在准备生产设置时，仍建议使用额外的性能调整和安全强化。

第 2 章 主动 - 被动部署的概念

本主题描述了高度可用的主动/被动设置，以及预期的行为。它概述了高可用性主动/被动架构的要求，并描述了优点和权衡。

2.1. 何时使用此设置

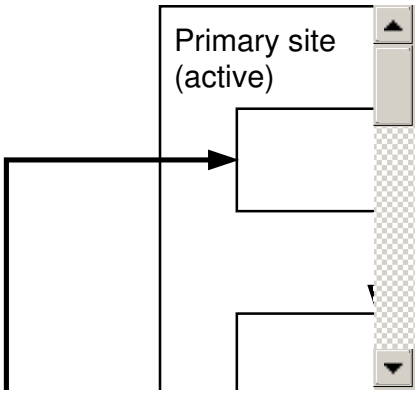
使用这个设置可以在站点失败时自动故障切换，这降低了丢失数据或会话的可能性。在故障转移后，需要手动交互来恢复冗余功能。

2.2. 部署、数据存储和缓存

在不同站点中运行的两个独立红帽构建的 Keycloak 部署都使用低延迟网络连接连接。用户、域、客户端、脱机会话和其他实体存储在在两个站点同步复制的数据库中。数据也缓存在红帽构建的 Keycloak Infinispan 缓存中作为本地缓存。当一个红帽构建的 Keycloak 实例更改数据时，该数据会在数据库中更新，并使用复制 **的工作** 缓存将无效的消息发送到其他站点。

会话相关的数据存储在红帽构建的 Keycloak 的 Infinispan 缓存的复制缓存中，并转发到外部数据网格，后者将信息转发到在其他站点同步运行的外部数据网格。由于外部数据网格的会话数据也缓存在 Infinispan 缓存中，因此验证复制 **工作** 缓存的消息需要失效。

在以下段落和图表中，部署 Data Grid 的引用适用于外部数据网格。



2.3. 数据丢失的原因

虽然此设置旨在实现高可用性，但以下情况仍然可以导致服务或数据丢失：

- 在检测到故障时，站点或组件故障之间的网络故障可能会导致缩短服务停机时间。该服务将自动恢复。系统会降级，直到检测到失败，并且备份集群被提升到服务请求。
- 在站点之间的通信时，需要手动步骤重新同步降级设置。
- 如果其他组件失败，降级的设置可能会导致服务或数据丢失。需要监控来检测降级的设置。

2.4. 此设置可以保留失败

失败	恢复	RPO[1]	RTO[2]

失败	恢复	RPO[1]	RTO[2]
数据库节点	如果 writer 实例失败，数据库可以将同一站点中的读取器实例提升为新的写入器。	没有数据丢失	分钟到秒（取决于数据库）
红帽构建的 Keycloak 节点	每个站点中运行多个红帽构建的 Keycloak 实例。如果一个实例失败，则其他节点需要几秒钟才能注意到更改，一些传入的请求可能会收到错误消息，或者延迟几秒钟。	没有数据丢失	少于一分钟
Data Grid 节点	多个 Data Grid 实例在每个站点中运行。如果一个实例失败，则其他节点需要几秒钟才能注意到更改。会话至少存储在两个 Data Grid 节点上，因此单个节点故障不会导致数据丢失。	没有数据丢失	少于一分钟
Data Grid 集群失败	如果在活跃站点中的 Data Grid 集群失败，红帽构建的 Keycloak 将无法与外部数据网格通信，红帽构建的 Keycloak 服务将不可用。loadbalancer 将检测情况为 **/lb-check** 返回错误，并将故障转移到其他站点。 设置会降级，直到 Data Grid 集群恢复，且会话数据被重新同步到主集群。	没有数据丢失[3]	分钟到秒（取决于负载均衡器设置）
连接数据网格	如果两个站点之间的连接丢失，则会话信息无法发送到其他站点。传入的请求可能会收到错误消息，或者延迟几秒钟。主站点将次要站点标记为离线，并将停止向次要站点发送数据。设置会降级，直到连接被恢复，且会话数据被重新同步到次站点。	没有数据丢失[3]	少于一分钟

失败	恢复	RPO[1]	RTO[2]
连接数据库	如果两个站点之间的连接丢失，同步复制将失败，并且主站点可能需要一些时间才能标记二级离线。有些请求可能会收到错误消息，或者延迟几秒钟。根据数据库，可能需要手动操作。	没有数据丢失[3]	分钟到秒（取决于数据库）
主站点	如果没有红帽构建的 Keycloak 节点可用，则 loadbalancer 将检测到中断并将流量重定向到二级站点。当 loadbalancer 没有检测到主站点失败时，一些请求可能会收到错误消息。设置将降级，直到主站点备份，并且从次要站点手动同步会话状态。	没有数据丢失[3]	少于一分钟
二级站点	如果次要站点不可用，则将需要稍等片刻，让主数据网格和数据库标记为离线次要站点。有些请求可能会在检测发生时收到错误消息。辅助站点再次启动后，需要将会话状态从主站点同步到次要站点。	没有数据丢失[3]	少于一分钟

表 footnotes:

[1] 恢复点目标（假设设置的所有部分在发生这种情况时都处于健康状态）。
恢复时间目标所需的[3] 个手动操作是恢复降级的设置。

语句"No 数据丢失"取决于设置不会因为之前失败而降级，其中包括完成任何待处理的手动操作来重新同步站点之间的状态。

2.5. 已知限制

升级

- 在红帽构建的 Keycloak 或 Data Grid 版本升级（主、次版本和补丁）中，所有会话数据（除离线会话除外）都会丢失，因为不支持零停机时间升级。

故障切换

- 成功故障转移需要设置不会因为以前的失败而降级。在以前的失败后，所有手动操作（如重新同步）都必须完成，以防止数据丢失。使用监控来确保及时检测和处理降级。

Switchovers

- 成功切换需要设置不会因为以前的失败而降级。在以前的失败后，所有手动操作（如重新同步）都必须完成，以防止数据丢失。使用监控来确保及时检测和处理降级。

不同步站点

- 当同步 Data Grid 请求失败时，站点可能会不同步。这个情况目前很难监控，需要完全手动重新同步 Data Grid 才能恢复。监控站点和红帽构建的 Keycloak 日志文件中的缓存条目数量，在需要重新同步时可能会显示。

手动操作

- 在站点间重新同步 Data Grid 状态的手动操作将发出完整的状态传输，这将对系统造成压力（网络、CPU、Java 堆在 Data Grid 中和红帽构建的 Keycloak）。

2.6. 问题和答案

为什么要同步数据库复制？

同步复制数据库可确保在故障切换时，在主站点中写入的数据始终可用，且不会丢失数据。

为什么要同步数据网格复制？

同步复制的数据网格确保主站点中创建、更新和删除的会话始终在故障切换的次要站点中可用，且不会丢失数据。

为什么在站点之间需要低延迟网络？

同步复制会延迟对调用者的响应，直到次要站点接收数据。对于同步数据库复制和同步数据网格复制，需要低延迟，因为当数据更新时，每个请求在站点之间可能存在多个交互，这会降低延迟。

为什么使用主动-被动？

有些数据库支持一个带有 reader 实例的单一写器实例，然后在原始写入器失败时将其提升到新的写入器。在这样的设置中，延迟与当前有效的 Keycloak 构建相同的站点有 writer 实例很有用。同步数据网格复制可能会导致同时修改两个站点中的条目时死锁。

此设置是否仅限于两个站点？

此设置可以扩展到多个站点，不需要修改三个站点。添加更多站点后，站点之间的总体延迟会增加，以及网络故障的类似线，因此还会增加停机时间。因此，此类部署应该具有更糟糕的性能和一个 inferior。现在，它已被测试，且只适用于两个站点的蓝图。

同步集群是否稳定低于异步集群？

异步设置将正常处理站点之间的网络故障，而同步设置会延迟请求，并将向调用者抛出错误，其中异步设置会将写入 Data Grid 或数据库延迟到次要站点。但是，由于二级站点永远不会与主站点完全更新，因此这个设置可能会导致在故障切换过程中丢失数据。这包括：

- 丢失的注销，这意味着会话会记录在次要站点，但它们在使用异步数据网格复制时在故障转移点登录到主站点。
- 丢失更改会导致用户可以使用旧密码登录，因为在使用异步数据库时，数据库更改不会复制到次要站点。
- 无效的缓存会导致用户可以使用旧密码登录，因为在使用异步数据网格复制时，故障转移时不会向二级站点传播无效的缓存。

因此，高可用性和一致性之间存在利弊。本主题的重点是，优先选择红帽构建的 Keycloak 的可用性的一致性。

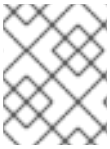
2.7. 后续步骤

继续阅读 构建块主动 - 被动部署 一章，以查找不同构建块的蓝图。

第 3 章 构建块主动 - 被动部署

使用同步复制设置主动 - 被动部署需要以下构建块。

构建块链接到带有示例配置的蓝图。它们按照需要安装的顺序列出。



注意

我们提供这些蓝图来显示最小功能的完整示例，以及常规安装的良好基准性能。您仍然需要根据您的环境以及您组织的标准和安全最佳实践进行调整。

3.1. 先决条件

- 了解用于 主动 - 被动部署 的概念介绍 章节。

3.2. 两个具有低延迟连接的站点

确保同步复制可用于数据库和外部数据网格。

建议设置： 同一 AWS 区域的两个 AWS Availablity 区域。

不考虑： 两个区域位于相同或不同的连续，因为它会增加延迟并出现网络故障的可能性。在 AWS 上将数据库同步复制为带有 Aurora 区域部署的服务，仅适用于同一区域。

3.3. 红帽构建的 KEYCLOAK 和 DATA GRID 环境

确保实例已根据需要部署并重启。

建议设置： 在每个可用区中部署 Red Hat OpenShift Service on AWS (ROSA)。

不考虑： 扩展 ROSA 集群跨越多个可用区，因为如果错误配置，这可能是单点故障。

3.4. 数据库

两个站点之间同步复制的数据库。

蓝图： 在多个可用区中部署 AWS Aurora。

3.5. DATA GRID

一个利用 Data Grid 的 Cross-DC 功能的 Data Grid 部署。

蓝图： 使用 Data Grid Operator 部署带有 Data Grid Operator 的 HA 的 Data Grid，并使用 Data Grid 的 Gossip Router 连接两个站点。

不考虑： 网络层上的 Kubernetes 集群之间的直接连接。未来可能会考虑它。

3.6. 红帽构建的 KEYCLOAK

每个站点中的红帽构建的 Keycloak 集群部署，连接到外部数据网格。

蓝图： 使用红帽构建的 Keycloak Operator 部署红帽构建的 Keycloak Operator 以及 Connect Red Hat build of Keycloak with an external Data Grid 和 Aurora 数据库。

3.7. 负载均衡器

一个负载均衡器，用于检查每个站点中红帽构建的 Keycloak 部署的 /**lb-check** URL。

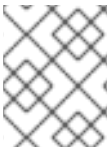
蓝图： 部署 AWS Route 53 loadbalancer 。

不考虑： AWS Global Accelerator 只支持权重流量路由，且不支持主动 - 被动故障转移。要支持主动 - 被动故障转移，例如，需要使用 AWS CloudWatch 和 AWS Lambda 的额外逻辑，以便在探测失败时调整权重来模拟主动 - 被动处理。

第 4 章 在多个可用区中部署 AWS AURORA

本节论述了如何在多个可用区间部署 PostgreSQL 实例的 Aurora 区域，以便在给定的 AWS 区域中容忍一个或多个可用区失败。

此部署旨在与 概念中的主动 - 被动部署 章节中描述的设置一起使用。将此部署与构建块相关的其他构建块一起使用，如 构建块主动 - 被动部署 章节中。



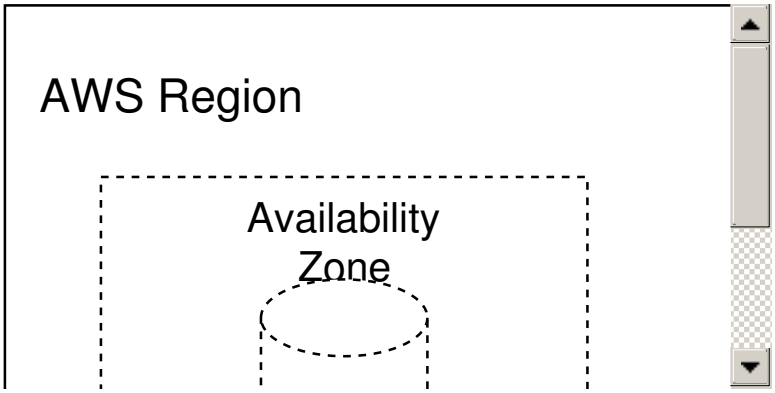
注意

我们提供这些蓝图来显示最小功能的完整示例，以及常规安装的良好基准性能。您仍然需要根据您的环境以及您组织的标准和安全最佳实践进行调整。

4.1. 架构

Aurora 数据库集群由多个 Aurora 数据库实例组成，一个实例被指定为主写器，所有其他实例都指定为备份读取器。为确保可用区失败时的高可用性，Aurora 允许在单个 AWS 区域中的多个区域间部署数据库实例。如果在托管主数据库实例的可用区上的故障，Aurora 会自动修复其自身，并将 reader 实例从非失败可用区提升为新的写器实例。

图 4.1. Aurora 多可用区部署



有关 Aurora 数据库提供的语义的更多详细信息，请参阅 AWS Aurora 文档。

本文档遵循 AWS 最佳实践，并创建一个不向互联网公开的私有 Aurora 数据库。要从 ROSA 集群访问数据库，请在 数据库和 ROSA 集群之间建立对等连接。

4.2. 流程

以下流程包含两个部分：

- 在 eu-west-1 中创建一个名为 "keycloak-aurora" 的 Aurora Multi-AZ 数据库集群。
- 在 ROSA 集群和 Aurora VPC 之间创建对等连接，以允许在 ROSA 集群上部署的应用程序与数据库建立连接。

4.2.1. 创建 Aurora 数据库集群

1. 为 Aurora 集群创建 VPC

   命令：

   ```
   aws ec2 create-vpc \
   ```

```
  --cidr-block 192.168.0.0/16 \
  --tag-specifications "ResourceType=vpc, Tags=[{Key=AuroraCluster,Value=keycloak-
aurora}]" \ 1
  --region eu-west-1
```

1 我们使用 Aurora 集群的名称添加可选标签，以便我们可以轻松检索 VPC。

输出：

```
{
    "Vpc": {
        "CidrBlock": "192.168.0.0/16",
        "DhcpOptionsId": "dopt-0bae7798158bc344f",
        "State": "pending",
        "VpcId": "vpc-0b40bd7c59dbe4277",
        "OwnerId": "606671647913",
        "InstanceTenancy": "default",
        "Ipv6CidrBlockAssociationSet": [],
        "CidrBlockAssociationSet": [
            {
                "AssociationId": "vpc-cidr-assoc-09a02a83059ba5ab6",
                "CidrBlock": "192.168.0.0/16",
                "CidrBlockState": {
                    "State": "associated"
                }
            }
        ],
        "IsDefault": false
    }
}
```

2. 使用新创建的 VPC 的 **VpcId**，为每个可用区创建一个子网。

注意

为每个可用区指定的 cidr-block 范围不得重叠。

a. 区 A

命令：

```
aws ec2 create-subnet \
  --availability-zone "eu-west-1a" \
  --vpc-id vpc-0b40bd7c59dbe4277 \
  --cidr-block 192.168.0.0/19 \
  --region eu-west-1
```

输出：

```
{
    "Subnet": {
        "AvailabilityZone": "eu-west-1a",
        "AvailabilityZoneId": "euw1-az3",
```

```
        "AvailableIpAddressCount": 8187,
        "CidrBlock": "192.168.0.0/19",
        "DefaultForAz": false,
        "MapPublicIpOnLaunch": false,
        "State": "available",
        "SubnetId": "subnet-0d491a1a798aa878d",
        "VpcId": "vpc-0b40bd7c59dbe4277",
        "OwnerId": "606671647913",
        "AssignIpv6AddressOnCreation": false,
        "Ipv6CidrBlockAssociationSet": [],
        "SubnetArn": "arn:aws:ec2:eu-west-1:606671647913:subnet/subnet-
0d491a1a798aa878d",
        "EnableDns64": false,
        "Ipv6Native": false,
        "PrivateDnsNameOptionsOnLaunch": {
            "HostnameType": "ip-name",
            "EnableResourceNameDnsARecord": false,
            "EnableResourceNameDnsAAAARecord": false
        }
    }
}
```


b. zone B

命令：

```
aws ec2 create-subnet \
  --availability-zone "eu-west-1b" \
  --vpc-id vpc-0b40bd7c59dbe4277 \
  --cidr-block 192.168.32.0/19 \
  --region eu-west-1
```

输出：


```
{
    "Subnet": {
        "AvailabilityZone": "eu-west-1b",
        "AvailabilityZoneId": "euw1-az1",
        "AvailableIpAddressCount": 8187,
        "CidrBlock": "192.168.32.0/19",
        "DefaultForAz": false,
        "MapPublicIpOnLaunch": false,
        "State": "available",
        "SubnetId": "subnet-057181b1e3728530e",
        "VpcId": "vpc-0b40bd7c59dbe4277",
        "OwnerId": "606671647913",
        "AssignIpv6AddressOnCreation": false,
        "Ipv6CidrBlockAssociationSet": [],
        "SubnetArn": "arn:aws:ec2:eu-west-1:606671647913:subnet/subnet-
057181b1e3728530e",
        "EnableDns64": false,
        "Ipv6Native": false,
        "PrivateDnsNameOptionsOnLaunch": {
            "HostnameType": "ip-name",
            "EnableResourceNameDnsARecord": false,
```

```
            "EnableResourceNameDnsAAAARecord": false
        }
    }
}
```

3. 获取 Aurora VPC 路由表 ID

命令：

```
aws ec2 describe-route-tables \
  --filters Name=vpc-id,Values=vpc-0b40bd7c59dbe4277 \
  --region eu-west-1
```

输出：

```
{
    "RouteTables": [
        {
            "Associations": [
                {
                    "Main": true,
                    "RouteTableAssociationId": "rtbassoc-02dfa06f4c7b4f99a",
                    "RouteTableId": "rtb-04a644ad3cd7de351",
                    "AssociationState": {
                        "State": "associated"
                    }
                }
            ],
            "PropagatingVgws": [],
            "RouteTableId": "rtb-04a644ad3cd7de351",
            "Routes": [
                {
                    "DestinationCidrBlock": "192.168.0.0/16",
                    "GatewayId": "local",
                    "Origin": "CreateRouteTable",
                    "State": "active"
                }
            ],
            "Tags": [],
            "VpcId": "vpc-0b40bd7c59dbe4277",
            "OwnerId": "606671647913"
        }
    ]
}
```

4. 关联 Aurora VPC 路由表每个可用区的子网

   a. 区 A

命令：

```
aws ec2 associate-route-table \
  --route-table-id rtb-04a644ad3cd7de351 \
  --subnet-id subnet-0d491a1a798aa878d \
  --region eu-west-1
```

b. zone B

命令：

```
aws ec2 associate-route-table \
  --route-table-id rtb-04a644ad3cd7de351 \
  --subnet-id subnet-057181b1e3728530e \
  --region eu-west-1
```

5. 创建 Aurora 子网组

命令：

```
aws rds create-db-subnet-group \
  --db-subnet-group-name keycloak-aurora-subnet-group \
  --db-subnet-group-description "Aurora DB Subnet Group" \
  --subnet-ids subnet-0d491a1a798aa878d subnet-057181b1e3728530e \
  --region eu-west-1
```

6. 创建 Aurora 安全组

命令：

```
aws ec2 create-security-group \
  --group-name keycloak-aurora-security-group \
  --description "Aurora DB Security Group" \
  --vpc-id vpc-0b40bd7c59dbe4277 \
  --region eu-west-1
```

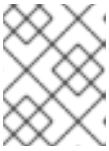
输出：

```
{
    "GroupId": "sg-0d746cc8ad8d2e63b"
}
```

7. 创建 Aurora DB 集群

命令：

```
aws rds create-db-cluster \
    --db-cluster-identifier keycloak-aurora \
    --database-name keycloak \
    --engine aurora-postgresql \
    --engine-version ${properties["aurora-postgresql.version"]} \
    --master-username keycloak \
    --master-user-password secret99 \
    --vpc-security-group-ids sg-0d746cc8ad8d2e63b \
    --db-subnet-group-name keycloak-aurora-subnet-group \
    --region eu-west-1
```



注意

您应该替换 **--master-username** 和 **--master-user-password** 值。在配置红帽构建的 Keycloak 数据库凭证时，必须使用此处指定的值。

输出：


```
{
    "DBCluster": {
        "AllocatedStorage": 1,
        "AvailabilityZones": [
            "eu-west-1b",
            "eu-west-1c",
            "eu-west-1a"
        ],
        "BackupRetentionPeriod": 1,
        "DatabaseName": "keycloak",
        "DBClusterIdentifier": "keycloak-aurora",
        "DBClusterParameterGroup": "default.aurora-postgresql15",
        "DBSubnetGroup": "keycloak-aurora-subnet-group",
        "Status": "creating",
        "Endpoint": "keycloak-aurora.cluster-clhthfqe0h8p.eu-west-1.rds.amazonaws.com",
        "ReaderEndpoint": "keycloak-aurora.cluster-ro-clhthfqe0h8p.eu-west-
1.rds.amazonaws.com",
        "MultiAZ": false,
        "Engine": "aurora-postgresql",
        "EngineVersion": "15.3",
        "Port": 5432,
        "MasterUsername": "keycloak",
        "PreferredBackupWindow": "02:21-02:51",
        "PreferredMaintenanceWindow": "fri:03:34-fri:04:04",
        "ReadReplicaIdentifiers": [],
        "DBClusterMembers": [],
        "VpcSecurityGroups": [
            {
                "VpcSecurityGroupId": "sg-0d746cc8ad8d2e63b",
                "Status": "active"
            }
        ],
        "HostedZoneId": "Z29XKXDKYMONMX",
        "StorageEncrypted": false,
        "DbClusterResourceId": "cluster-IBWXUWQYM3MS5BH557ZJ6ZQU4I",
        "DBClusterArn": "arn:aws:rds:eu-west-1:606671647913:cluster:keycloak-aurora",
        "AssociatedRoles": [],
        "IAMDatabaseAuthenticationEnabled": false,
        "ClusterCreateTime": "2023-11-01T10:40:45.964000+00:00",
        "EngineMode": "provisioned",
        "DeletionProtection": false,
        "HttpEndpointEnabled": false,
        "CopyTagsToSnapshot": false,
        "CrossAccountClone": false,
        "DomainMemberships": [],
        "TagList": [],
        "AutoMinorVersionUpgrade": true,
```

```
        "NetworkType": "IPV4"
    }
}
```

8. 创建 Aurora DB 实例

   a. 创建区 A Writer 实例

   命令：

   ```
   aws rds create-db-instance \
     --db-cluster-identifier keycloak-aurora \
     --db-instance-identifier "keycloak-aurora-instance-1" \
     --db-instance-class db.t4g.large \
     --engine aurora-postgresql \
     --region eu-west-1
   ```

   b. 创建区 B Reader 实例

   命令：

   ```
   aws rds create-db-instance \
     --db-cluster-identifier keycloak-aurora \
     --db-instance-identifier "keycloak-aurora-instance-2" \
     --db-instance-class db.t4g.large \
     --engine aurora-postgresql \
     --region eu-west-1
   ```

9. 等待所有 Writer 和 Reader 实例就绪

   命令：

   ```
   aws rds wait db-instance-available --db-instance-identifier keycloak-aurora-instance-1 --
   region eu-west-1
   aws rds wait db-instance-available --db-instance-identifier keycloak-aurora-instance-2 --
   region eu-west-1
   ```

10. Obtain the Writer 端点 URL 供 Keycloak 使用

    命令：

    ```
    aws rds describe-db-clusters \
      --db-cluster-identifier keycloak-aurora \
      --query 'DBClusters[*].Endpoint' \
      --region eu-west-1 \
      --output text
    ```

    输出：

    ```
    [
        "keycloak-aurora.cluster-clhthfqe0h8p.eu-west-1.rds.amazonaws.com"
    ]
    ```

4.2.2. 使用 ROSA 集群建立对等连接

对包含红帽构建的 Keycloak 部署的每个 ROSA 集群执行这些步骤。

1. 检索 Aurora VPC

   命令：

   ```
   aws ec2 describe-vpcs \
     --filters "Name=tag:AuroraCluster,Values=keycloak-aurora" \
     --query 'Vpcs[*].VpcId' \
     --region eu-west-1 \
     --output text
   ```

   输出：

   ```
   vpc-0b40bd7c59dbe4277
   ```

2. 检索 ROSA 集群 VPC

   a. 使用 **oc**登录到 ROSA 集群

   b. 检索 ROSA VPC

      命令：

      ```
      NODE=$(oc get nodes --selector=node-role.kubernetes.io/worker -o
      jsonpath='{.items[0].metadata.name}')
      aws ec2 describe-instances \
        --filters "Name=private-dns-name,Values=${NODE}" \
        --query 'Reservations[0].Instances[0].VpcId' \
        --region eu-west-1 \
        --output text
      ```

      输出：

      ```
      vpc-0b721449398429559
      ```

3. 创建 Peering 连接

   命令：

   ```
   aws ec2 create-vpc-peering-connection \
     --vpc-id vpc-0b721449398429559 \ 1
     --peer-vpc-id vpc-0b40bd7c59dbe4277 \ 2
     --peer-region eu-west-1 \
     --region eu-west-1
   ```

   1 ROSA 集群 VPC

   2 Aurora VPC

   输出：

```
{
    "VpcPeeringConnection": {
        "AccepterVpcInfo": {
            "OwnerId": "606671647913",
            "VpcId": "vpc-0b40bd7c59dbe4277",
            "Region": "eu-west-1"
        },
        "ExpirationTime": "2023-11-08T13:26:30+00:00",
        "RequesterVpcInfo": {
            "CidrBlock": "10.0.17.0/24",
            "CidrBlockSet": [
                {
                    "CidrBlock": "10.0.17.0/24"
                }
            ],
            "OwnerId": "606671647913",
            "PeeringOptions": {
                "AllowDnsResolutionFromRemoteVpc": false,
                "AllowEgressFromLocalClassicLinkToRemoteVpc": false,
                "AllowEgressFromLocalVpcToRemoteClassicLink": false
            },
            "VpcId": "vpc-0b721449398429559",
            "Region": "eu-west-1"
        },
        "Status": {
            "Code": "initiating-request",
            "Message": "Initiating Request to 606671647913"
        },
        "Tags": [],
        "VpcPeeringConnectionId": "pcx-0cb23d66dea3dca9f"
    }
}
```


4. 等待 Peering 连接存在

命令：

```
aws ec2 wait vpc-peering-connection-exists --vpc-peering-connection-ids pcx-0cb23d66dea3dca9f
```

5. 接受对等连接

命令：

```
aws ec2 accept-vpc-peering-connection \
  --vpc-peering-connection-id pcx-0cb23d66dea3dca9f \
  --region eu-west-1
```

输出：


```
{
    "VpcPeeringConnection": {
        "AccepterVpcInfo": {
```

```
            "CidrBlock": "192.168.0.0/16",
            "CidrBlockSet": [
                {
                    "CidrBlock": "192.168.0.0/16"
                }
            ],
            "OwnerId": "606671647913",
            "PeeringOptions": {
                "AllowDnsResolutionFromRemoteVpc": false,
                "AllowEgressFromLocalClassicLinkToRemoteVpc": false,
                "AllowEgressFromLocalVpcToRemoteClassicLink": false
            },
            "VpcId": "vpc-0b40bd7c59dbe4277",
            "Region": "eu-west-1"
        },
        "RequesterVpcInfo": {
            "CidrBlock": "10.0.17.0/24",
            "CidrBlockSet": [
                {
                    "CidrBlock": "10.0.17.0/24"
                }
            ],
            "OwnerId": "606671647913",
            "PeeringOptions": {
                "AllowDnsResolutionFromRemoteVpc": false,
                "AllowEgressFromLocalClassicLinkToRemoteVpc": false,
                "AllowEgressFromLocalVpcToRemoteClassicLink": false
            },
            "VpcId": "vpc-0b721449398429559",
            "Region": "eu-west-1"
        },
        "Status": {
            "Code": "provisioning",
            "Message": "Provisioning"
        },
        "Tags": [],
        "VpcPeeringConnectionId": "pcx-0cb23d66dea3dca9f"
    }
}
```


6. 更新 ROSA 集群 VPC 路由

命令：

```
ROSA_PUBLIC_ROUTE_TABLE_ID=$(aws ec2 describe-route-tables \
  --filters "Name=vpc-id,Values=vpc-0b721449398429559"
"Name=association.main,Values=true" \ 1
  --query "RouteTables[*].RouteTableId" \
  --output text \
  --region eu-west-1
)
aws ec2 create-route \
  --route-table-id ${ROSA_PUBLIC_ROUTE_TABLE_ID} \
```

```
  --destination-cidr-block 192.168.0.0/16 \ 2
  --vpc-peering-connection-id pcx-0cb23d66dea3dca9f \
  --region eu-west-1
```

1 ROSA 集群 VPC

2 这必须与创建 Aurora VPC 时使用的 cidr-block 相同

7. 更新 Aurora 安全组

命令：

```
AURORA_SECURITY_GROUP_ID=$(aws ec2 describe-security-groups \
  --filters "Name=group-name,Values=keycloak-aurora-security-group" \
  --query "SecurityGroups[*].GroupId" \
  --region eu-west-1 \
  --output text
)
aws ec2 authorize-security-group-ingress \
  --group-id ${AURORA_SECURITY_GROUP_ID} \
  --protocol tcp \
  --port 5432 \
  --cidr 10.0.17.0/24 \ 1
  --region eu-west-1
```

1 ROSA 集群的 "machine_cidr"

输出：

```
{
    "Return": true,
    "SecurityGroupRules": [
        {
            "SecurityGroupRuleId": "sgr-0785d2f04b9cec3f5",
            "GroupId": "sg-0d746cc8ad8d2e63b",
            "GroupOwnerId": "606671647913",
            "IsEgress": false,
            "IpProtocol": "tcp",
            "FromPort": 5432,
            "ToPort": 5432,
            "CidrIpv4": "10.0.17.0/24"
        }
    ]
}
```

4.3. 验证连接

验证在 ROSA 集群和 Aurora DB 集群之间可以连接的最简单方法是在 Openshift 集群上部署 **psql**，并尝试连接到 writer 端点。

以下命令在 default 命名空间中创建 pod，并在可能的情况下与 Aurora 集群建立 **psql** 连接。退出 pod shell 后，pod 会被删除。

```
USER=keycloak 1
PASSWORD=secret99 2
DATABASE=keycloak 3
HOST=$(aws rds describe-db-clusters \
  --db-cluster-identifier keycloak-aurora \ 4
  --query 'DBClusters[*].Endpoint' \
  --region eu-west-1 \
  --output text
)
oc run -i --tty --rm debug --image=postgres:15 --restart=Never -- psql
postgresql://${USER}:${PASSWORD}@${HOST}/${DATABASE}
```

1. Aurora DB 用户，这可能与创建 DB 时使用的 **--master-username** 相同。
2. Aurora DB 用户密码，这可能与创建 DB 时使用的 **--master-user-password** 相同。
3. Aurora DB 的名称，如 **--database-name**。
4. Aurora DB 集群的名称。

4.4. 部署红帽构建的 KEYCLOAK

现在，Aurora 数据库已建立并链接到所有 ROSA 集群，下一步是部署 Keycloak 的红帽构建 Keycloak，如部署带有红帽构建的 Keycloak Operator 的 Keycloak Operator 章节， 且将 JDBC url 配置为使用 Aurora 数据库写器端点所述。要做到这一点，使用以下调整创建一个 **Keycloak** CR：

1. 将 **spec.db.url** 更新为 **jdbc:aws-wrapper:postgresql://$HOST:5432/keycloak**，其中 **$HOST** 是 Aurora writer 端点 URL。
2. 确保 **spec.db.usernameSecret** 和 **spec.db.passwordSecret** 引用的 Secret 包含创建 Aurora 时定义的用户名和密码。

第 5 章 使用 RED HAT BUILD OF KEYCLOAK OPERATOR 部署红帽构建的 KEYCLOAK FOR HA

本指南描述了 Kubernetes 的高级红帽构建 Keycloak 配置，这些配置被测试，并从单个 Pod 失败中恢复。

这些说明旨在与 概念中用于主动 - 被动部署 一章中所述的设置一起使用。与构建块 的主动 - 被动部署章节中概述的其他构建块 一起使用。

5.1. 先决条件

- 正在运行的 OpenShift 或 Kubernetes 集群。
- 使用 Red Hat build of Keycloak Operator 了解红帽构建的 Keycloak 部署的基本红帽构建的 Keycloak 部署。

5.2. 流程

1. 使用 概念来确定部署的大小，以调整 CPU 和内存资源 大小 章节。
2. 安装红帽构建的 Keycloak Operator，如 红帽构建的 Keycloak Operator 安装 章节中所述。
3. 如在 多个可用区一章中的 Deploy AWS Aurora 所述，部署 Aurora AWS 。
4. 构建自定义红帽 Keycloak 镜像构建，该镜像 准备好用于 Amazon Aurora PostgreSQL 数据库。
5. 使用在第一步中计算的资源请求和限值，使用以下值部署红帽 Keycloak CR：

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  labels:
    app: keycloak
  name: keycloak
  namespace: keycloak
spec:
  hostname:
    hostname: <KEYCLOAK_URL_HERE>
  resources:
    requests:
      cpu: "2"
      memory: "1250M"
    limits:
      cpu: "6"
      memory: "2250M"
  db:
    vendor: postgres
    url: jdbc:aws-wrapper:postgresql://<AWS_AURORA_URL_HERE>:5432/keycloak
    poolMinSize: 30 (1)
    poolInitialSize: 30
    poolMaxSize: 30
    usernameSecret:
      name: keycloak-db-secret
      key: username
```

```
    passwordSecret:
      name: keycloak-db-secret
      key: password
  image: <KEYCLOAK_IMAGE_HERE> 2
  startOptimized: false 3
  features:
    enabled:
      - multi-site 4
  transaction:
    xaEnabled: false 5
  additionalOptions:
    - name: http-max-queued-requests
      value: "1000"
    - name: log-console-output
      value: json
    - name: metrics-enabled 6
      value: 'true'
    - name: http-pool-max-threads 7
      value: "66"
    - name: db-driver
      value: software.amazon.jdbc.Driver
  http:
    tlsSecret: keycloak-tls-secret
  instances: 3
```

1 数据库连接池 initial、max 和 min 大小应当相同，以允许数据库的声明缓存。调整这个数字以满足您的系统需求。由于红帽构建的 Keycloak 嵌入式缓存，大多数请求都不涉及数据库，因此这个更改可以每秒为几百个请求服务器。详情请参阅数据库连接池的概念 章节。

2 3 指定自定义红帽构建的 Keycloak 镜像的 URL。如果您的镜像经过优化，请将 **startOptimized** 标志设置为 **true**。

4 为多站点支持启用其他功能，如 loadbalancer probe **/lb-check**。

5 Amazon Web Services JDBC 驱动程序不支持 XA 事务 。

6 要能够分析负载下的系统，请启用指标端点。设置的缺点是，指标将在 Keycloak 端点的外部红帽构建中提供，因此您必须添加一个过滤器，以便外部无法使用端点。使用红帽构建的 Keycloak 之前的反向代理来过滤这些 URL。

7 内部 JGroup 线程池的默认设置最大为 200 个线程。StatefulSet 中所有红帽构建的 Keycloak 线程数量不应超过 JGroup 线程的数量，以避免 JGroup 线程池耗尽，这可能会使红帽构建 Keycloak 请求处理。您可以考虑进一步限制红帽构建的 Keycloak 线程数量，因为达到请求的 CPU 限制后，Kubernetes 将会导致多个并发线程节流。详情请参阅配置线程池的概念。

5.3. 验证部署

确认红帽构建的 Keycloak 部署已就绪。

```
oc wait --for=condition=Ready keycloaks.k8s.keycloak.org/keycloak
oc wait --for=condition=RollingUpdate=False keycloaks.k8s.keycloak.org/keycloak
```

5.4. 可选： LOAD SHEDDING

要启用负载她，请限制排队的请求数量。

使用最大排队的 http 请求加载她

```
spec:
  additionalOptions:
    - name: http-max-queued-requests
      value: "1000"
```

所有超过请求都通过 HTTP 503 提供。详情请查看配置线程池 的概念。

5.5. 可选：禁用粘性会话

当在 OpenShift 上运行以及由 Red Hat build of Keycloak Operator 提供的默认 passthrough Ingress 设置时，HAProxy 所做的负载均衡是通过根据源的 IP 地址使用粘性会话来实现的。在运行负载测试时，或者在 HAProxy 前面有一个反向代理时，您可能希望禁用此设置以避免在单个红帽构建的 Keycloak Pod 上接收所有请求。

在红帽构建的 Keycloak 自定义资源的 **spec** 下添加以下附加配置，以禁用粘性会话。

```
spec:
  ingress:
    enabled: true
    annotations:
      # When running load tests, disable sticky sessions on the OpenShift HAProxy router
      # to avoid receiving all requests on a single Red Hat build of Keycloak Pod.
      haproxy.router.openshift.io/balance: roundrobin
      haproxy.router.openshift.io/disable_cookies: 'true'
```

第 6 章 使用 DATA GRID OPERATOR 为 HA 部署 DATA GRID

本章描述了在多集群（跨站点）中部署 Data Grid 所需的步骤。为了简单起见，本主题使用最低配置，允许红帽构建 Keycloak 与外部 Data Grid 一起使用。

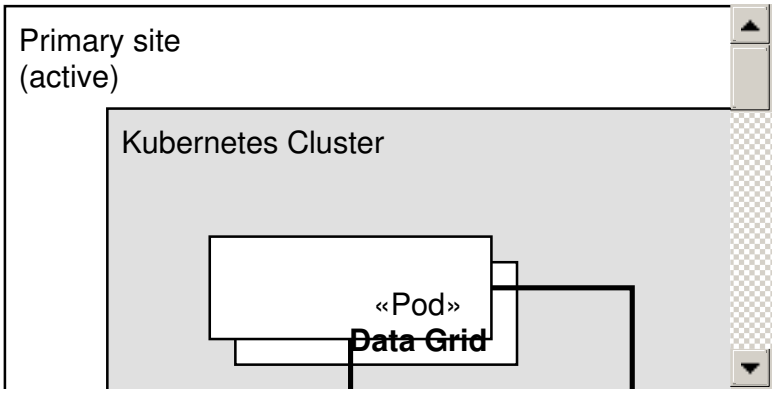
本章假定两个名为 **Site-A** 和 **Site-B** 的 OpenShift 集群。

这是一个构建块，遵循概念中 用于主动 - 被动部署章节中的概念。如需了解概述，请参阅 多站点部署 章节。

6.1. 架构

这个设置会在带有低延迟网络连接的两个站点中同步复制 Data Grid 集群。一个这种情况的示例可能是一个 AWS 区域中的两个可用区。

为了简单起见，红帽构建的 Keycloak、loadbalancer 和 数据库已从下图中删除。



6.2. 先决条件

- 正在运行的 OpenShift 或 Kubernetes 集群
- 了解 Data Grid Operator

6.3. 流程

1. 安装 Data Grid Operator
2. 配置凭据以访问 Data Grid 集群。
   红帽构建的 Keycloak 需要此凭证才能与 Data Grid 集群进行身份验证。以下 **identity.yaml** 文件使用 admin 权限设置用户名和密码

```
credentials:
  - username: developer
    password: strong-password
    roles:
      - admin
```

identity.yaml 可以在 secret 中设置为以下之一：

- 作为 Kubernetes 资源：

  凭证 Secret

```
apiVersion: v1
kind: Secret
type: Opaque
metadata:
  name: connect-secret
  namespace: keycloak
data:
  identities.yaml:
Y3JlZGVudGlhbHM6CiAgLSB1c2VybmFtZTogZGV2ZWxvcGVyCiAgICBwYXNzd29yZDog
c3Ryb25nLXBhc3N3b3JkCiAgICByb2xlczoKICAgICAgLSBhZG1pbgo= (1)
```

(1) 上例 base64 编码的 identity **.yaml**。

- 使用 CLI

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

如需了解更多详细信息，请参阅 配置身份验证文档。https://access.redhat.com/documentation/zh-cn/red_hat_data_grid/8.4/html-single/data_grid_operator_guide/index#configuring-authentication

这些命令必须在两个 OpenShift 集群上执行。

3. 创建一个服务帐户。
在集群间建立连接需要服务帐户。Data Grid Operator 使用它来从远程站点检查网络配置，并相应地配置本地 Data Grid 集群。

如需了解更多详细信息，请参阅管理跨站点连接 文档。

a. 按照如下所示，创建 **service-account-token** secret 类型：两个 OpenShift 集群中都可以使用相同的 YAML 文件。

xsite-sa-secret-token.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: ispn-xsite-sa-token (1)
  annotations:
    kubernetes.io/service-account.name: "xsite-sa" (2)
type: kubernetes.io/service-account-token
```

(1) 机密名称。

(2) 服务帐户名称。

b. 创建服务帐户并在两个 OpenShift 集群中生成访问令牌。

在 Site-A中创建服务帐户

```
oc create sa -n keycloak xsite-sa
oc policy add-role-to-user view -n keycloak -z xsite-sa
```

```
oc create -f xsite-sa-secret-token.yaml
oc get secrets ispn-xsite-sa-token -o jsonpath="{.data.token}" | base64 -d > Site-A-token.txt
```

在 Site-B中创建服务帐户

```
oc create sa -n keycloak xsite-sa
oc policy add-role-to-user view -n keycloak -z xsite-sa
oc create -f xsite-sa-secret-token.yaml
oc get secrets ispn-xsite-sa-token -o jsonpath="{.data.token}" | base64 -d > Site-B-token.txt
```

c. 下一步是将令牌从 **Site-A** 部署到 **Site-B**，反向部署：

将 Site-B 令牌部署到 Site-A

```
oc create secret generic -n keycloak xsite-token-secret \
  --from-literal=token="$(cat Site-B-token.txt)"
```

将 Site-A 令牌部署到 Site-B

```
oc create secret generic -n keycloak xsite-token-secret \
  --from-literal=token="$(cat Site-A-token.txt)"
```

4. 创建 TLS secret
在本章中，Data Grid 使用 OpenShift 路由进行跨站点通信。它使用 TLS 的 SNI 扩展将流量定向到正确的 Pod。为实现这一目标，JGroups 使用 TLS 套接字，这需要一个含有正确证书的密钥存储和 Truststore。

如需更多信息，请参阅 安全跨站点连接 文档或 红帽开发人员指南。

在 OpenShift Secret 中上传 Keystore 和 Truststore。secret 包含文件内容、访问它的密码，以及存储的类型。创建证书和存储的说明超出了本指南的范围。

要将密钥存储上传为 Secret，请使用以下命令：

部署密钥存储

```
oc -n keycloak create secret generic xsite-keystore-secret \
  --from-file=keystore.p12="./certs/keystore.p12" \ 1
  --from-literal=password=secret \ 2
  --from-literal=type=pkcs12 3
```

1 文件名和密钥存储的路径。

2 用于访问密钥存储的密码。

3 Keystore 类型。

要将 Truststore 作为一个 Secret 上传，请使用以下命令：

部署 Truststore

```
oc -n keycloak create secret generic xsite-truststore-secret \
      --from-file=truststore.p12="./certs/truststore.p12" \ 1
      --from-literal=password=caSecret \ 2
      --from-literal=type=pkcs12 3
```

1 文件名和 Truststore 的路径。

2 访问 Truststore 的密码。

3 Truststore 类型。

注意

密钥存储和 Truststore 必须在两个 OpenShift 集群中上传。

5. 为启用了 Cross-Site 的 Data Grid 创建集群
设置跨站点 文档提供了有关如何在启用跨站点（包括前面步骤）创建和配置 Data Grid 集群的所有信息。

本章中提供了一个基本示例，它使用从前面的步骤中创建的凭证、令牌和 TLS Keystore/Truststore。

Site-A的 Infinispan CR

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan 1
  namespace: keycloak
  annotations:
    infinispan.org/monitoring: 'true' 2
spec:
  replicas: 3
  security:
    endpointSecretName: connect-secret 3
  service:
    type: DataGrid
    sites:
      local:
        name: site-a 4
        expose:
          type: Route 5
        maxRelayNodes: 128
        encryption:
          transportKeyStore:
            secretName: xsite-keystore-secret 6
            alias: xsite 7
            filename: keystore.p12 8
          routerKeyStore:
            secretName: xsite-keystore-secret 9
            alias: xsite 10
            filename: keystore.p12 11
```

```
      trustStore:
        secretName: xsite-truststore-secret 12
        filename: truststore.p12 13
    locations:
      - name: site-b 14
        clusterName: infinispan
        namespace: keycloak 15
        url: openshift://api.site-b 16
        secretName: xsite-token-secret 17
```

1 集群名称

2 允许 Prometheus 监控集群。

3 如果使用自定义凭证，请在此处配置 secret 名称。

4 本地站点的名称，本例中为 **Site-A**。

5 使用 OpenShift 路由公开跨站点连接。

6 9 上一步中定义的 Keystore 的 secret 名称。

7 10 Keystore 中证书的别名。

8 11 上一步中定义的 Keystore 的 secret 密钥(filename)。

12 上一步中定义的 Truststore 的 secret 名称。

13 上一步中定义的 Keystore 的 Truststore 密钥(filename)。

14 远程站点的名称，本例中为 **Site-B**。

15 远程站点的 Data Grid 集群的命名空间。

16 远程站点的 OpenShift API URL。

17 有权在远程站点进行身份验证的 secret。

对于 **Site-B**，**Infinispan** CR 类似于以上内容。请注意点 4、11 和 13。

Site-B的 Infinispan CR

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan 1
  namespace: keycloak
  annotations:
    infinispan.org/monitoring: 'true' 2
spec:
  replicas: 3
  security:
    endpointSecretName: connect-secret 3
  service:
```

```
    type: DataGrid
    sites:
      local:
        name: site-b (4)
        expose:
          type: Route (5)
        maxRelayNodes: 128
        encryption:
          transportKeyStore:
            secretName: xsite-keystore-secret (6)
            alias: xsite (7)
            filename: keystore.p12 (8)
          routerKeyStore:
            secretName: xsite-keystore-secret (9)
            alias: xsite (10)
            filename: keystore.p12 (11)
          trustStore:
            secretName: xsite-truststore-secret (12)
            filename: truststore.p12 (13)
      locations:
        - name: site-a (14)
          clusterName: infinispan
          namespace: keycloak (15)
          url: openshift://api.site-a (16)
          secretName: xsite-token-secret (17)
```

6. 为红帽构建的 Keycloak 创建缓存。
   红帽构建的 Keycloak 需要存在以下缓存： **会话**,**actionTokens**,**authenticationSessions**,**offlineSessions**,**clientSessions**,**offlineClientSessions**,**loginFailures**, 和 **work**。

   Data Grid Cache CR 允许在 Data Grid 集群中部署缓存。需要为每个缓存启用跨站点，如 跨站点文档记录。文档包含有关本章使用的选项的更多详细信息。以下示例显示了 **Site-A** 的 **Cache** CR。

   Site-A中的会话

```
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: sessions
  namespace: keycloak
spec:
  clusterName: infinispan
  name: sessions
  template: |-
    distributedCache:
      mode: "SYNC"
      owners: "2"
      statistics: "true"
      remoteTimeout: 14000
      stateTransfer:
        chunkSize: 16
      backups:
```

```
      mergePolicy: ALWAYS_REMOVE (1)
      site-b: (2)
        backup:
          strategy: "SYNC" (3)
          timeout: 13000
          stateTransfer:
            chunkSize: 16
```

(1) 当存在写写冲突时，调用跨站点合并策略。把它设置为缓存 **会话**、**authenticationSessions**、**offlineSessions**、**clientSessions** 和 **offlineClientSessions**，且不会为所有其他缓存设置它。

(2) 远程站点名称。

(3) 跨站点通信，本例中为 **SYNC**。

对于 **Site-B**，**缓存** CR 类似于第 2 点。

Site-B中的会话

```
apiVersion: infinispan.org/v2alpha1
kind: Cache
metadata:
  name: sessions
  namespace: keycloak
spec:
  clusterName: infinispan
  name: sessions
  template: |-
    distributedCache:
      mode: "SYNC"
      owners: "2"
      statistics: "true"
      remoteTimeout: 14000
      stateTransfer:
        chunkSize: 16
      backups:
        mergePolicy: ALWAYS_REMOVE (1)
        site-a: (2)
          backup:
            strategy: "SYNC" (3)
            timeout: 13000
            stateTransfer:
              chunkSize: 16
```

6.4. 验证部署

确认 Data Grid 集群已形成，并且 OpenShift 集群之间建立了跨站点连接。

等待 Data Grid 集群形成

```
oc wait --for condition=WellFormed --timeout=300s infinispans.infinispan.org -n keycloak infinispan
```

等待 Data Grid 跨站点连接建立

```
oc wait --for condition=CrossSiteViewFormed --timeout=300s infinispans.infinispan.org -n keycloak infinispan
```

6.5. 后续步骤

部署并运行 Data Grid 后，使用 Connect Red Hat build of Keycloak with an external Data Grid 章节中的步骤将红帽构建的 Keycloak 集群与 Data Grid 集群连接。

第 7 章 将红帽构建的 KEYCLOAK 与外部数据网格连接

本主题描述了 Kubernetes 上红帽构建的 Keycloak 的高级数据网格配置。

7.1. 架构

这会将红帽构建的 Keycloak 与 Data Grid 连接使用由 TLS 1.3 保护的 TCP 连接。它使用红帽构建的 Keycloak 的信任存储来验证 Data Grid 的服务器证书。因为红帽构建的 Keycloak 会在以下先决条件中使用其 Operator 在 OpenShift 上部署，所以 Operator 已将 **service-ca.crt** 添加到用于为 Data Grid 的服务器证书签名的信任存储中。在其他环境中，将必要的证书添加到红帽构建的 Keycloak 的信任存储中。

7.2. 先决条件

- 使用 Red Hat build of Keycloak Operator 部署红帽构建的 Keycloak for HA，因为它将会被扩展。
- 使用 Data Grid Operator 部署用于 HA 的数据网格。

7.3. 流程

1. 使用用户名和密码创建 Secret 以连接到外部 Data Grid 部署：

```
apiVersion: v1
kind: Secret
metadata:
  name: remote-store-secret
  namespace: keycloak
type: Opaque
data:
  username: ZGV2ZWxvcGVy # base64 encoding for 'developer'
  password: c2VjdXJlX3Bhc3N3b3Jk # base64 encoding for 'secure_password'
```

2. 使用 **additionalOptions** 扩展红帽构建的 Keycloak 自定义资源，如下所示。

注意

以下所有内存、资源和数据库配置都会从以下 CR 跳过，因为它们已在 Red Hat build of Keycloak Operator 中部署红帽构建的 Keycloak for HA。管理员应使这些配置保持不变。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  labels:
    app: keycloak
  name: keycloak
  namespace: keycloak
spec:
  additionalOptions:
    - name: cache-remote-host (1)
      value: "infinispan.keycloak.svc"
    - name: cache-remote-port (2)
      value: "11222"
```

```
- name: cache-remote-username (3)
  secret:
    name: remote-store-secret
    key: username
- name: cache-remote-password (4)
  secret:
    name: remote-store-secret
    key: password
- name: spi-connections-infinispan-quarkus-site-name (5)
  value: keycloak
```

(1) (1) 远程 Data Grid 集群的主机名。

(2) (2) 远程 Data Grid 集群的端口。这是可选的，默认为 **11222**。

(3) (3) 带有 Data Grid 用户名凭证的 Secret **名称和 密钥**。

(4) 带有 Data Grid 密码凭证的 Secret **名称和 密钥**。

(5) **spi-connections-infinispan-quarkus-site-name** 是在使用远程存储时红帽构建的 Keycloak 需要的任意 Data Grid 站点名称。此 site-name 仅与 Infinispan 缓存相关，不需要与外部 Data Grid 部署中的任何值匹配。如果您在跨DC设置中为红帽构建的 Keycloak 使用多个站点，如使用 Data Grid Operator 部署用于 HA 的 Data Grid，则每个站点中的站点名称必须不同。

7.4. 相关选项

	value
cache-remote-host 远程存储配置的远程服务器的主机名。 它替换通过 XML 文件指定的配置的 **remote-server** 标签的 **host** 属性（请参阅 **cache-config-file** 选项）。如果指定了 选项，则需要 **cache-remote-username** 和 **cache-remote-password**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	
cache-remote-password 远程服务器对远程存储进行身份验证的密码。 它替换通过 XML 文件指定的配置的 **digest** 标签的 **password** 属性（请参阅 **cache-config-file** 选项）。如果指定了 选项，则需要 **cache-remote-host** 和 **cache-remote-username**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD**	

	value
cache-remote-port 远程服务器用于远程存储配置的端口。 它替换通过 XML 文件指定的配置的 **remote-server** 标签的 **port** 属性（请参阅 **cache-config-file** 选项）。 CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT**	**11222** （默认）
cache-remote-username 远程存储的远程服务器身份验证的用户名。 它替换通过 XML 文件指定的配置的 **digest** 标签的 **username** 属性（请参阅 **cache-config-file** 选项）。如果指定了 选项，则需要 **cache-remote-host** 和 **cache-remote-password**，并且 XML 文件中的相关配置不应存在。 CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME**	

第 8 章 部署 AWS ROUTE 53 LOADBALANCER

本主题描述了为 Multi-AZ Red Hat build of Keycloak 集群为主动/被动设置配置基于 DNS 的故障切换所需的步骤。这些说明旨在与 概念中用于主动 - 被动部署 一章中所述的设置一起使用。与构建块 的主动 - 被动部署章节中概述的其他构建块 一起使用。

注意

我们提供这些蓝图来显示最小功能的完整示例，以及常规安装的良好基准性能。您仍然需要根据您的环境以及您组织的标准和安全最佳实践进行调整。

8.1. 架构

所有红帽构建的 Keycloak 客户端请求都由 Route53 记录管理的 DNS 名称路由。Route53 被重新发送，以确保所有客户端请求在可用且健康时路由到主集群，或者在主可用区或红帽构建的 Keycloak 部署失败时路由到备份集群。

如果主站点失败，DNS 更改将需要传播到客户端。根据客户端的设置，传播可能需要一些时间，具体取决于客户端的配置。使用移动连接时，一些互联网供应商可能无法遵守 DNS 条目的 TTL，这可能会导致客户端连接到新站点前的延长时间。

图 8.1. AWS Global Accelerator Failover

在主和备份 ROSA 集群上公开两个 Openshift 路由。第一个路由使用 Route53 DNS 名称来服务客户端请求，而 Route53 使用第二个 Route 来监控红帽构建的 Keycloak 集群的健康状态。

8.2. 先决条件

- 部署红帽构建的 Keycloak，如在 AWS 一个区域的两个 AWS 可用区域中使用红帽构建的 Keycloak Operator 在 Red Hat build of Keycloak Operator 上部署。
- 用于要路由的客户端请求的拥有的域。

8.3. 流程

1. 使用您希望所有红帽构建的 Keycloak 客户端要连接的根域名创建一个 Route53 Hosted Zone。记录"托管区域 ID"，因为后续步骤中需要这个 ID。
2. 检索与每个 ROSA 集群关联的 "Hosted zone ID" 和 DNS 名称。
   对于 Primary 和 Backup 集群，请执行以下步骤：
   a. 登录到 ROSA 集群。

b. 检索集群 LoadBalancer Hosted Zone ID 和 DNS 主机名

命令：

```
HOSTNAME=$(oc -n openshift-ingress get svc router-default \
-o jsonpath='{.status.loadBalancer.ingress[].hostname}'
)
aws elbv2 describe-load-balancers \
--query "LoadBalancers[?DNSName=='${HOSTNAME}'].
{CanonicalHostedZoneId:CanonicalHostedZoneId,DNSName:DNSName}" \
--region eu-west-1 \ 1
--output json
```

1 托管您的 ROSA 集群的 AWS 区域

输出：

```
[
  {
    "CanonicalHostedZoneId": "Z2IFOLAFXWLO4F",
    "DNSName": "ad62c8d2fcffa4d54aec7ffff902c925-61f5d3e1cbdc5d42.elb.eu-west-
1.amazonaws.com"
  }
]
```

注意

运行 OpenShift 4.13 及更早版本的 ROSA 集群使用典型的负载均衡器，而不是应用程序负载均衡器。使用 **aws elb describe-load-balancers** 命令以及更新的查询字符串。

3. 创建 Route53 健康检查

命令：

```
function createHealthCheck() {
  # Creating a hash of the caller reference to allow for names longer than 64 characters
  REF=($(echo $1 | sha1sum ))
  aws route53 create-health-check \
  --caller-reference "$REF" \
  --query "HealthCheck.Id" \
  --no-cli-pager \
  --output text \
  --health-check-config '
  {
    "Type": "HTTPS",
    "ResourcePath": "/lb-check",
    "FullyQualifiedDomainName": "'$1'",
    "Port": 443,
    "RequestInterval": 30,
    "FailureThreshold": 1,
    "EnableSNI": true
  }
```

```
  '
}
CLIENT_DOMAIN="client.keycloak-benchmark.com" (1)
PRIMARY_DOMAIN="primary.${CLIENT_DOMAIN}" (2)
BACKUP_DOMAIN="backup.${CLIENT_DOMAIN}" (3)
createHealthCheck ${PRIMARY_DOMAIN}
createHealthCheck ${BACKUP_DOMAIN}
```

1. 红帽构建的 Keycloak 客户端应连接到的域。这应该是用于创建 Hosted Zone 的根域的子域或子域。
2. 用于主集群中的健康探测的子域
3. 用于备份集群中的健康探测的子域

输出：

```
233e180f-f023-45a3-954e-415303f21eab (1)
799e2cbb-43ae-4848-9b72-0d9173f04912 (2)
```

1. 主健康检查的 ID
2. Backup Health 检查的 ID

4. 创建 Route53 记录集

命令：

```
HOSTED_ZONE_ID="Z09084361B6LKQQRCVBEY" (1)
PRIMARY_LB_HOSTED_ZONE_ID="Z2IFOLAFXWLO4F"
PRIMARY_LB_DNS=ad62c8d2fcffa4d54aec7ffff902c925-61f5d3e1cbdc5d42.elb.eu-west-1.amazonaws.com
PRIMARY_HEALTH_ID=233e180f-f023-45a3-954e-415303f21eab
BACKUP_LB_HOSTED_ZONE_ID="Z2IFOLAFXWLO4F"
BACKUP_LB_DNS=a184a0e02a5d44a9194e517c12c2b0ec-1203036292.elb.eu-west-1.amazonaws.com
BACKUP_HEALTH_ID=799e2cbb-43ae-4848-9b72-0d9173f04912
aws route53 change-resource-record-sets \
 --hosted-zone-id Z09084361B6LKQQRCVBEY \
 --query "ChangeInfo.Id" \
 --output text \
 --change-batch '
 {
  "Comment": "Creating Record Set for '${CLIENT_DOMAIN}'",
  "Changes": [{
   "Action": "CREATE",
   "ResourceRecordSet": {
    "Name": "'${PRIMARY_DOMAIN}'",
    "Type": "A",
     "AliasTarget": {
      "HostedZoneId": "'${PRIMARY_LB_HOSTED_ZONE_ID}'",
      "DNSName": "'${PRIMARY_LB_DNS}'",
      "EvaluateTargetHealth": true
```

```
        }
      }
    }, {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "'${BACKUP_DOMAIN}'",
        "Type": "A",
          "AliasTarget": {
            "HostedZoneId": "'${BACKUP_LB_HOSTED_ZONE_ID}'",
            "DNSName": "'${BACKUP_LB_DNS}'",
            "EvaluateTargetHealth": true
          }
      }
    }, {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "'${CLIENT_DOMAIN}'",
        "Type": "A",
          "SetIdentifier": "client-failover-primary-'${SUBDOMAIN}'",
          "Failover": "PRIMARY",
          "HealthCheckId": "'${PRIMARY_HEALTH_ID}'",
          "AliasTarget": {
            "HostedZoneId": "'${HOSTED_ZONE_ID}'",
            "DNSName": "'${PRIMARY_DOMAIN}'",
            "EvaluateTargetHealth": true
          }
      }
    }, {
      "Action": "CREATE",
      "ResourceRecordSet": {
        "Name": "'${CLIENT_DOMAIN}'",
        "Type": "A",
          "SetIdentifier": "client-failover-backup-'${SUBDOMAIN}'",
          "Failover": "SECONDARY",
          "HealthCheckId": "'${BACKUP_HEALTH_ID}'",
          "AliasTarget": {
            "HostedZoneId": "'${HOSTED_ZONE_ID}'",
            "DNSName": "'${BACKUP_DOMAIN}'",
            "EvaluateTargetHealth": true
          }
      }
    }]
  }
  '
```

1 之前创建的 Hosted 区的 ID

输出：

```
/change/C053410633T95FR9WN3YI
```

5. 等待 Route53 记录更新

命令：

```
aws route53 wait resource-record-sets-changed --id /change/C053410633T95FR9WN3YI
```

6. 更新或创建红帽构建的 Keycloak 部署
   对于 Primary 和 Backup 集群，请执行以下步骤：

   a. 登录到 ROSA 集群

   b. 确保 **Keycloak** CR 有以下配置

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: keycloak
spec:
  hostname:
    hostname: ${CLIENT_DOMAIN} (1)
```

   (1) 用于连接到红帽构建的 Keycloak 的域客户端

   为确保请求转发正常工作，请编辑红帽构建的 Keycloak CR，以指定客户端将访问红帽 Keycloak 实例的主机名。此主机名必须是 Route53 配置中使用的 **$CLIENT_DOMAIN**。

   c. 创建健康检查路由

   命令：

```
cat <<EOF | oc apply -n $NAMESPACE -f - (1)
apiVersion: route.openshift.io/v1
kind: Route
metadata:
  name: aws-health-route
spec:
  host: $DOMAIN (2)
  port:
    targetPort: https
  tls:
    insecureEdgeTerminationPolicy: Redirect
    termination: passthrough
  to:
    kind: Service
    name: keycloak-service
    weight: 100
  wildcardPolicy: None

EOF
```

   (1) **$NAMESPACE** 应该替换为红帽构建的 Keycloak 部署的命名空间

   (2) 如果当前集群分别是 Backup 集群，则 **$DOMAIN** 应该替换为 **PRIMARY_DOMAIN** 或 **BACKUP_DOMAIN**。

8.4. 验证

导航到本地浏览器中所选的 CLIENT_DOMAIN，并登录到红帽构建的 Keycloak 控制台。

要测试故障转移可以正常工作，请登录到 Primary 集群，并将 Keycloak 部署的红帽构建扩展到零个 Pod。扩展会导致主健康检查失败，Route53 应该开始将流量路由到备份集群中的红帽 Keycloak Pod。

第 9 章 故障转移到二级站点

本章论述了在设置中从主站点切换到次要站点的步骤，如 构建块 主动-被动部署中的 蓝图中所述。

9.1. 何时使用步骤

从主站点故障转移到次要站点，将根据 loadbalancer 中配置的检查自动进行。

当主站点丢失了 Data Grid 或网络分区（阻止同步）时，需要手动步骤在可以再次处理流量前恢复主站点，请参阅 切换回主站点 章节。

要在执行这些手动步骤前防止自动回退到主站点，请按照如下所述配置 loadbalancer，以防止自动发生这种情况。

对于安全切换到二级站点，请按照 切换到次要站点 章节中的说明进行操作。

有关不同的操作步骤，请参阅 多站点部署 章节。

9.2. 流程

按照以下步骤手动强制故障转移。

9.2.1. Route53

要强制 Route53 将主站点标记为永久不可用，并防止自动回退，在 AWS 中编辑健康检查以指向不存在的路由(**健康/关闭**)。

第 10 章 切换到二级站点

在将设置用于主动 - 被动部署与 构建块 主动 - 被动部署中的 蓝图中所述，这个流程从主站点切换到次要站点。

10.1. 何时使用此流程

使用这个流程正常使主离线。

主站点恢复在线后，使用章节 恢复自同步被动站点，并 切回到主站点 以返回到原始状态，同时主站点处于活动状态。

有关不同的操作步骤，请参阅 多站点部署 章节。

10.2. 流程

10.2.1. Data Grid 集群

对于本章的上下文中，site **-A** 是主站点，site **-B** 是次要站点。

当您准备离线站点时，最好禁用该站点的复制。当频道在主站点和次站点之间断开连接时，此操作可防止错误或延迟。

10.2.1.1. 将状态从次要站点传输到主站点的流程

1. 登录到您的次要站点
2. 使用 Data Grid CLI 工具连接到 Data Grid 集群：

   命令：

   ```
   oc -n keycloak exec -it pods/infinispan-0 -- ./bin/cli.sh --trustall --connect
   https://127.0.0.1:11222
   ```

   它要求提供 Data Grid 集群的用户名和密码。这些凭证是在配置凭证部分的 带有 Data Grid Operator 的 Deploy Data Grid for HA 一章中设置的。

   输出：

   ```
   Username: developer
   Password:
   [infinispan-0-29897@ISPN//containers/default]>
   ```

   注意

   pod 名称取决于 Data Grid CR 中定义的集群名称。该连接可以通过 Data Grid 集群中的任何 pod 来完成。

3. 运行以下命令，禁用到主站点的复制：

   命令：

```
site take-offline --all-caches --site=site-a
```

输出：

```
{
  "offlineClientSessions" : "ok",
  "authenticationSessions" : "ok",
  "sessions" : "ok",
  "clientSessions" : "ok",
  "work" : "ok",
  "offlineSessions" : "ok",
  "loginFailures" : "ok",
  "actionTokens" : "ok"
}
```

4. 检查复制状态是否为 **offline**。

命令：

```
site status --all-caches --site=site-a
```

输出：

```
{
  "status" : "offline"
}
```

如果状态未 **脱机**，请重复上一步。

二级站点中的 Data Grid 集群已准备好处理请求，而无需尝试复制到主站点。

10.2.2. AWS Aurora 数据库

假设区域 multi-AZ Aurora 部署，当前写入器实例应与有效的红帽构建的 Keycloak 集群位于同一个区域，以避免跨可用区的延迟和通信。

切换 Aurora 的写器实例会导致短暂的停机时间。在某些部署中，其他站点中的写入器实例可能会接受延迟稍长。因此，这个情况可能会延迟到维护窗口或跳过，具体取决于部署的情况。

要更改写器实例，请运行故障转移。此更改将使数据库在短时间内不可用。红帽构建的 Keycloak 需要重新建立数据库连接。

要将 writer 实例切换到其他 AZ，请运行以下命令：

```
aws rds failover-db-cluster  --db-cluster-identifier ...
```

10.2.3. 红帽构建的 Keycloak 集群

不需要任何操作。

10.2.4. Route53

要强制 Route53 将主站点标记为不可用，请编辑 AWS 中的健康检查以指向不存在的路由(**健康/关闭**)。几分钟后，客户端会注意到更改，流量将逐渐移到次要站点。

10.3. 进一步阅读

请参阅 概念来自动执行 Data Grid CLI 命令，以自动执行 Infinispan CLI 命令。

第 11 章 从同步被动站点中恢复

本章论述了将二级站点与设置中的主站点同步的步骤，如 构建块 主动-被动部署中的 蓝图中所述。

11.1. 何时使用步骤

在 Data Grid 断开连接且缓存内容不同步的站点之间使用该连接。

在流程结束时，次要站点上的会话内容已被丢弃，并替换为主站点的会话内容。二级站点中的所有缓存都已清除，以防止无效的缓存内容。

有关不同的操作步骤，请参阅 多站点部署 章节。

11.2. 流程

11.2.1. Data Grid 集群

对于本章的上下文中，site **-A** 是主站点且处于活动状态，site **-B** 是二级站点，是被动。

站点和 Data Grid 集群之间的复制可能会在网络分区之间停止。这些流程使两个站点重新同步。

警告

通过增加响应时间和/或资源使用量，传输完整状态可能会影响 Data Grid 集群性能。

第一个步骤是从次要站点中删除过时的数据。

1. 登录到您的次要站点。
2. 关闭红帽构建的 Keycloak。这将清除所有红帽构建的 Keycloak 缓存，并可防止红帽构建的 Keycloak 状态与数据网格不同步。
   当使用红帽构建的 Keycloak Operator 部署红帽 Keycloak 时，将红帽构建的 Keycloak 实例的数量改为 0。
3. 使用 Data Grid CLI 工具连接到 Data Grid 集群：

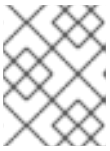
   命令：

   ```
   oc -n keycloak exec -it pods/infinispan-0 -- ./bin/cli.sh --trustall --connect
   https://127.0.0.1:11222
   ```

   它要求提供 Data Grid 集群的用户名和密码。这些凭证是在配置凭证部分的 带有 Data Grid Operator 的 Deploy Data Grid for HA 一章中设置的。

   输出：

```
Username: developer
Password:
[infinispan-0-29897@ISPN//containers/default]>
```



注意

pod 名称取决于 Data Grid CR 中定义的集群名称。该连接可以通过 Data Grid 集群中的任何 pod 来完成。

4. 运行以下命令，禁用从次要站点到主站点的复制。它可防止清除请求访问主站点并删除所有正确的缓存数据。

命令：

```
site take-offline --all-caches --site=site-a
```

输出：

```
{
  "offlineClientSessions" : "ok",
  "authenticationSessions" : "ok",
  "sessions" : "ok",
  "clientSessions" : "ok",
  "work" : "ok",
  "offlineSessions" : "ok",
  "loginFailures" : "ok",
  "actionTokens" : "ok"
}
```

5. 检查复制状态是否为 **offline**。

命令：

```
site status --all-caches --site=site-a
```

输出：

```
{
  "status" : "offline"
}
```

如果状态未 **脱机**，请重复上一步。

警告

确保复制 **离线**，否则清除数据将清除这两个站点。

6. 使用以下命令清除二级站点中的所有缓存数据：

   命令：

   ```
   clearcache actionTokens
   clearcache authenticationSessions
   clearcache clientSessions
   clearcache loginFailures
   clearcache offlineClientSessions
   clearcache offlineSessions
   clearcache sessions
   clearcache work
   ```

   这些命令不会打印任何输出。

7. 重新启用从次要站点到主站点的跨站点复制。

   命令：

   ```
   site bring-online --all-caches --site=site-a
   ```

   输出：

   ```
   {
     "offlineClientSessions" : "ok",
     "authenticationSessions" : "ok",
     "sessions" : "ok",
     "clientSessions" : "ok",
     "work" : "ok",
     "offlineSessions" : "ok",
     "loginFailures" : "ok",
     "actionTokens" : "ok"
   }
   ```

8. 检查复制状态是否为 **在线**。

   命令：

   ```
   site status --all-caches --site=site-a
   ```

   输出：

   ```
   {
     "status" : "online"
   }
   ```

现在，我们已准备好将状态从主站点传输到次要站点。

1. 登录到您的主站点
2. 使用 Data Grid CLI 工具连接到 Data Grid 集群：

   命令：

```
oc -n keycloak exec -it pods/infinispan-0 -- ./bin/cli.sh --trustall --connect
https://127.0.0.1:11222
```

它要求提供 Data Grid 集群的用户名和密码。这些凭证是在配置凭证部分的 带有 Data Grid Operator 的 Deploy Data Grid for HA 一章中设置的。

输出：

```
Username: developer
Password:
[infinispan-0-29897@ISPN//containers/default]>
```

注意

pod 名称取决于 Data Grid CR 中定义的集群名称。该连接可以通过 Data Grid 集群中的任何 pod 来完成。

3. 触发从主站点传输到次要站点的状态。

命令：

```
site push-site-state --all-caches --site=site-b
```

输出：

```
{
  "offlineClientSessions" : "ok",
  "authenticationSessions" : "ok",
  "sessions" : "ok",
  "clientSessions" : "ok",
  "work" : "ok",
  "offlineSessions" : "ok",
  "loginFailures" : "ok",
  "actionTokens" : "ok"
}
```

4. 检查所有缓存的复制状态是否 **在线**。

命令：

```
site status --all-caches --site=site-b
```

输出：

```
{
  "status" : "online"
}
```

5. 检查所有缓存的 **push-site-status** 命令的输出，以等待状态传输完成。

命令：

```
```

```
site push-site-status --cache=actionTokens
site push-site-status --cache=authenticationSessions
site push-site-status --cache=clientSessions
site push-site-status --cache=loginFailures
site push-site-status --cache=offlineClientSessions
site push-site-status --cache=offlineSessions
site push-site-status --cache=sessions
site push-site-status --cache=work
```

输出：

```
{
  "site-b" : "OK"
}
{
  "site-b" : "OK"
}
{
  "site-b" : "OK"
}
{
  "site-b" : "OK"
}
{
  "site-b" : "OK"
}
{
  "site-b" : "OK"
}
{
  "site-b" : "OK"
}
```

检查 本节中的表，以查看 Cross-Site 文档 以了解可能的状态值。

如果报告错误，请为该特定缓存重复状态传输。

命令：

```
site push-site-state --cache=<cache-name> --site=site-b
```

6. 使用以下命令清除/重置状态传输状态

命令：

```
site clear-push-site-status --cache=actionTokens
site clear-push-site-status --cache=authenticationSessions
site clear-push-site-status --cache=clientSessions
site clear-push-site-status --cache=loginFailures
site clear-push-site-status --cache=offlineClientSessions
```

```
site clear-push-site-status --cache=offlineSessions
site clear-push-site-status --cache=sessions
site clear-push-site-status --cache=work
```

输出：

```
"ok"
"ok"
"ok"
"ok"
"ok"
"ok"
"ok"
"ok"
```

现在，状态在二级站点中可用，因此 Red Hat build of Keycloak 可以再次启动：

1. 登录到您的次要站点。
2. 启动红帽构建的 Keycloak。
   当使用红帽构建的 Keycloak Operator 部署红帽 Keycloak 时，将红帽构建的 Keycloak 实例的数量改为原始值。

11.2.2. AWS Aurora 数据库

不需要任何操作。

11.2.3. Route53

不需要任何操作。

11.3. 进一步阅读

请参阅 概念来自动执行 Data Grid CLI 命令，以自动执行 Infinispan CLI 命令。

第 12 章 切回到主站点

这些流程在故障转移或切换到二级站点后切换回主站点。在设置中，如 主动 - 被动部署概念 中所述，以及 构建块主动-被动部署 中概述的蓝图。

12.1. 何时使用此流程

当二级站点处理所有流量时，这些步骤会将主站点恢复为操作。在本章结束时，主站点再次在线并处理流量。

当主站点丢失 Data Grid 状态时，需要这个过程，在主站点和次站点活跃时发生网络分区，或者复制被禁用，如 切换到次要站点 章节中所述。

如果两个站点的 Data Grid 中的数据仍然同步，则可以跳过 Data Grid 的步骤。

有关不同的操作步骤，请参阅 多站点部署 章节。

12.2. 流程

12.2.1. Data Grid 集群

对于本章的上下文中，site **-A** 是主站点，恢复为操作，site **-B** 是生产中运行的次要站点。

在主站点中的 Data Grid 返回后，并已加入跨站点频道（请参阅 带有 Data Grid Operator Deploy Data Griding-the-deployment on the Data Griddeployment）的 Deploy Data Griding-the-deployment on the Data Grid deployment，必须从二级站点手动启动。

在清除主站点中的状态后，它会将完整状态从次要站点传输到主站点，并且必须在主站点开始处理传入请求之前完成。

警告

通过增加响应时间和/或资源使用量，传输完整状态可能会影响 Data Grid 集群执行。

第一个步骤是从主站点中删除任何过时的数据。

1. 登录到主站点。
2. 关闭红帽构建的 Keycloak。此操作将清除所有红帽构建的 Keycloak 缓存，并防止红帽构建的 Keycloak 状态与数据网格不同步。
   当使用红帽构建的 Keycloak Operator 部署红帽 Keycloak 时，将红帽构建的 Keycloak 实例的数量改为 0。
3. 使用 Data Grid CLI 工具连接到 Data Grid 集群：

   命令：

   ```
   oc -n keycloak exec -it pods/infinispan-0 -- ./bin/cli.sh --trustall --connect
   https://127.0.0.1:11222
   ```

它要求提供 Data Grid 集群的用户名和密码。这些凭证是在配置凭证部分的 带有 Data Grid Operator 的 Deploy Data Grid for HA 一章中设置的。

输出：

```
Username: developer
Password:
[infinispan-0-29897@ISPN//containers/default]>
```

注意

pod 名称取决于 Data Grid CR 中定义的集群名称。该连接可以通过 Data Grid 集群中的任何 pod 来完成。

4. 运行以下命令，禁用从主站点到次要站点的复制。它可防止清除请求访问二级站点，并删除所有正确的缓存数据。

命令：

```
site take-offline --all-caches --site=site-b
```

输出：

```
{
  "offlineClientSessions" : "ok",
  "authenticationSessions" : "ok",
  "sessions" : "ok",
  "clientSessions" : "ok",
  "work" : "ok",
  "offlineSessions" : "ok",
  "loginFailures" : "ok",
  "actionTokens" : "ok"
}
```

5. 检查复制状态是否为 **offline**。

命令：

```
site status --all-caches --site=site-b
```

输出：

```
{
  "status" : "offline"
}
```

如果状态未 **脱机**，请重复上一步。

警告

确保复制 **离线**，否则清除数据将清除这两个站点。

6. 使用以下命令清除主站点中的所有缓存数据：

命令：

```
clearcache actionTokens
clearcache authenticationSessions
clearcache clientSessions
clearcache loginFailures
clearcache offlineClientSessions
clearcache offlineSessions
clearcache sessions
clearcache work
```

这些命令不会打印任何输出。

7. 重新启用从主站点到次要站点的跨站点复制。

命令：

```
site bring-online --all-caches --site=site-b
```

输出：

```
{
  "offlineClientSessions" : "ok",
  "authenticationSessions" : "ok",
  "sessions" : "ok",
  "clientSessions" : "ok",
  "work" : "ok",
  "offlineSessions" : "ok",
  "loginFailures" : "ok",
  "actionTokens" : "ok"
}
```

8. 检查复制状态是否为 **在线**。

命令：

```
site status --all-caches --site=site-b
```

输出：

```
{
  "status" : "online"
}
```

现在，我们已准备好将状态从次要站点传输到主站点。

1. 登录到您的次要站点。
2. 使用 Data Grid CLI 工具连接到 Data Grid 集群：

   命令：

   ```
   oc -n keycloak exec -it pods/infinispan-0 -- ./bin/cli.sh --trustall --connect
   https://127.0.0.1:11222
   ```

   它要求提供 Data Grid 集群的用户名和密码。这些凭证是在配置凭证部分的 带有 Data Grid Operator 的 Deploy Data Grid for HA 一章中设置的。

   输出：

   ```
   Username: developer
   Password:
   [infinispan-0-29897@ISPN//containers/default]>
   ```

   注意

   pod 名称取决于 Data Grid CR 中定义的集群名称。该连接可以通过 Data Grid 集群中的任何 pod 来完成。

3. 触发从次要站点传输到主站点的状态。

   命令：

   ```
   site push-site-state --all-caches --site=site-a
   ```

   输出：

   ```
   {
     "offlineClientSessions" : "ok",
     "authenticationSessions" : "ok",
     "sessions" : "ok",
     "clientSessions" : "ok",
     "work" : "ok",
     "offlineSessions" : "ok",
     "loginFailures" : "ok",
     "actionTokens" : "ok"
   }
   ```

4. 检查所有缓存的复制状态是否 **在线**。

   命令：

   ```
   site status --all-caches --site=site-a
   ```

   输出：

```
{
  "status" : "online"
}
```

5. 检查所有缓存的 **push-site-status** 命令的输出，以等待状态传输完成。

   命令：

```
site push-site-status --cache=actionTokens
site push-site-status --cache=authenticationSessions
site push-site-status --cache=clientSessions
site push-site-status --cache=loginFailures
site push-site-status --cache=offlineClientSessions
site push-site-status --cache=offlineSessions
site push-site-status --cache=sessions
site push-site-status --cache=work
```

   输出：

```
{
  "site-a" : "OK"
}
{
  "site-a" : "OK"
}
{
  "site-a" : "OK"
}
{
  "site-a" : "OK"
}
{
  "site-a" : "OK"
}
{
  "site-a" : "OK"
}
{
  "site-a" : "OK"
}
{
  "site-a" : "OK"
}
```

   检查 本节中的表，以查看 Cross-Site 文档 以了解可能的状态值。

   如果报告错误，请为该特定缓存重复状态传输。

   命令：

```
site push-site-state --cache=<cache-name> --site=site-a
```

6. 使用以下命令清除/重置状态传输状态

   命令：

命令：

```
site clear-push-site-status --cache=actionTokens
site clear-push-site-status --cache=authenticationSessions
site clear-push-site-status --cache=clientSessions
site clear-push-site-status --cache=loginFailures
site clear-push-site-status --cache=offlineClientSessions
site clear-push-site-status --cache=offlineSessions
site clear-push-site-status --cache=sessions
site clear-push-site-status --cache=work
```

输出：

```
"ok"
"ok"
"ok"
"ok"
"ok"
"ok"
"ok"
"ok"
```

7. 登录到主站点。

8. 启动红帽构建的 Keycloak。
   当使用红帽构建的 Keycloak Operator 部署红帽 Keycloak 时，将红帽构建的 Keycloak 实例的数量改为原始值。

两个 Data Grid 集群都同步，可以执行从次要到主站点的切换。

12.2.2. AWS Aurora 数据库

假设区域 multi-AZ Aurora 部署，当前写入器实例应与有效的红帽构建的 Keycloak 集群位于同一个区域，以避免跨可用区的延迟和通信。

切换 Aurora 的写器实例会导致短暂的停机时间。在某些部署中，其他站点中的写入器实例可能会接受延迟稍长。因此，这个情况可能会延迟到维护窗口或跳过，具体取决于部署的情况。

要更改写器实例，请运行故障转移。此更改将使数据库在短时间内不可用。红帽构建的 Keycloak 需要重新建立数据库连接。

要将 writer 实例切换到其他 AZ，请运行以下命令：

```
aws rds failover-db-cluster  --db-cluster-identifier ...
```

12.2.3. Route53

如果通过更改健康端点触发了切换到二级站点，请编辑 AWS 中的健康检查以指向正确的端点 (**health/live**)。几分钟后，客户端会注意到更改，流量将逐渐移到次要站点。

12.3. 进一步阅读

请参阅 概念来自动执行 Data Grid CLI 命令，以自动执行 Infinispan CLI 命令。

第 13 章 配置线程池的概念

本节旨在帮助您了解如何为红帽构建的 Keycloak 配置线程池的注意事项和最佳实践。对于应用此功能的配置，请访问红帽构建的 Keycloak Operator 的 Deploy Red Hat build of HA 。

13.1. 概念

13.1.1. Quarkus executor 池

红帽构建的 Keycloak 请求以及阻塞探测由 executor 池处理。根据可用的 CPU 内核，最大为 200 或多个线程。线程根据需要创建，并在不再需要时结束，因此系统将自动缩放和缩减。红帽构建的 Keycloak 允许通过 **http-pool-max-threads** 配置选项配置最大线程池大小。请参阅 使用红帽构建的 Keycloak Operator 部署红帽 Keycloak for HA。

在 Kubernetes 上运行时，调整 worker 线程数量，以避免创建比 Pod 允许的 CPU 限值更多的负载，从而避免节流，从而导致拥塞。在物理机上运行时，调整 worker 线程数量，以避免创建比节点可以处理的负载更多的负载以避免拥塞。拥塞会导致响应时间较长，增加内存用量，最终会导致系统不稳定。

理想情况下，您应该从低线程限制开始，并将其相应地调整到目标吞吐量和响应时间。当负载和线程数量增加时，数据库连接也可以成为瓶颈。当请求在 5 秒内无法获取数据库连接后，它将失败，并显示 **Unable to acquire JDBC Connection** 的消息。调用者将收到显示服务器端错误的 5xx HTTP 状态代码的响应。

如果您增加数据库连接的数量和线程数量过多，则系统将处于高负载下，请求排队，从而导致性能不良。数据库连接的数量通过 **Database** 设置 **db-pool-initial-size**、**db-pool-min-size** 和 **db-pool-max-size** 分别进行配置。低数字可确保所有客户端快速响应时间，即使存在负载激增时偶尔会出现请求失败。

13.1.2. JGroups 连接池

集群中所有红帽构建的 Keycloak 节点合并的 executor 线程数量不应超过 JGroups 线程池中提供的线程数量，以避免错误 **org.jgroups.util.ThreadPool: thread pool is full**。要查看第一次发生的错误，需要把系统属性 **jgroups.thread_dumps_threshold** 设置为 **1**，因为否则消息仅在 10000 个线程被拒绝后出现。

JGroup 线程的数量默认为 **200**。虽然可以使用属性 Java 系统属性 **jgroups.thread_pool.max_threads** 进行配置，但我们建议将其保持在这个值中。如试验中所示，集群中 Quarkus worker 线程总数不能超过每个节点 200 的 JGroup 线程池中的线程数量，以避免 JGroups 通信中的死锁。鉴于带有四个 Pod 的红帽构建的 Keycloak 集群，则每个 Pod 应该有 50 Quarkus worker 线程。使用红帽构建的 Keycloak 配置选项 **http-pool-max-threads** 来配置最大 Quarkus worker 线程数。

使用指标 **vendor_jgroups_tcp_get_thread_pool_size** 监控池中活跃的线程的总 JGroup 线程和 **vendor_jgroups_tcp_get_thread_pool_size_active**。这对于监控限制 Quarkus 线程池大小会保持在最大 JGroup 线程池大小下的活跃 JGroup 线程数量。

13.1.3. Load Shedding

默认情况下，红帽构建的 Keycloak 将无限地排队所有传入的请求，即使请求处理停滞也是如此。这将在 Pod 中使用额外的内存，可能会耗尽负载均衡器中的资源，请求最终会在客户端超时，而无需客户端了解请求是否已被处理。要限制红帽构建的 Keycloak 中排队的请求数，请设置额外的 Quarkus 配置选项。

配置 **http-max-queued-requests** 以指定最大队列长度，以便在超过此队列大小后实现有效的负载。假设红帽构建的 Keycloak Pod 每秒处理 200 个请求，队列 1000 会导致最多等待时间 5 秒。

当此设置处于活动状态时，超过排队请求数量的请求将返回 HTTP 503 错误。Red Hat build of Keycloak 会在日志中记录错误信息。

13.1.4. probes

红帽构建的 Keycloak 存活度探测不是阻止的，以避免在高负载下重启 Pod。

总体健康探测和就绪度探测在某些情况下可以探测到数据库，因此它们可能会在高负载下失败。因此，Pod 可能会在高负载下变得未就绪。

13.1.5. OS 资源

为了让 Java 创建线程，在 Linux 上运行时需要文件句柄。因此，打开的文件数量（作为 **ulimit -n** 在 Linux 上检索）需要为红帽构建的 Keycloak 提供头空间，以增加所需的线程数量。每个线程也会消耗内存，容器内存限值需要设置为允许此值或 Pod 由 Kubernetes 终止的值。

第 14 章 数据库连接池的概念

本节旨在帮助您了解如何为红帽构建的 Keycloak 配置数据库连接池的注意事项和最佳实践。对于应用此功能的配置，请访问红帽构建的 Keycloak Operator 的 Deploy Red Hat build of HA 。

14.1. 概念

创建新数据库连接的成本为需要时间。当请求到达时创建它们会延迟响应，因此最好在请求到达前创建它们。它还有助于在短时间内创建很多连接使系统和块线程减慢速度更糟糕。https://en.wikipedia.org/wiki/Cache_stampede关闭连接也会使该连接的所有服务器端语句缓存无效。

为获得最佳性能，初始的值最小和最大数据库连接池大小应相等。这可避免在新请求发生时创建新的数据库连接，而成本较高。

只要可能允许将服务器端声明缓存绑定到连接，就可以使数据库连接保持打开。对于 PostgreSQL，若要使用服务器端准备语句，需要至少执行（默认为 ）查询。

如需更多信息，请参阅准备的语句中的 PostgreSQL 文档。

第 15 章 CPU 和内存资源大小的概念

将此用作调整产品环境大小的起点。根据您的负载测试，根据需要调整您的环境的值。

15.1. 性能建议

警告

- 当扩展到更多 Pod （由于额外的开销）和使用跨数据中心设置（因为额外的流量和操作）时，性能会降低。
- 当红帽构建 Keycloak 实例时，增加的缓存大小可以提高性能。这将减少响应时间并减少数据库的 IOPS。仍然需要在实例重启时填充这些缓存，因此不要根据缓存填充后测量的稳定状态设置资源。
- 使用这些值作为起点，并在进入生产前执行您自己的负载测试。

Summary：

- 使用的 CPU 会线性扩展，请求数量最高为以下测试的限制。
- 使用的内存会线性扩展，且活跃会话数量达到以下测试的限制。

建议：

- 非活动 Pod 的基本内存用量为 1000 MB RAM。
- 对于每个 100,000 个活跃的用户会话，在三节点集群中添加 500 MB 的每个 Pod （测试最多 200,000 个会话）。
  这假定每个用户仅连接到一个客户端。内存要求随着每个用户会话的客户端会话数量（尚未测试）增加。
- 在容器中，Keycloak 为基于堆的内存分配 70% 的内存限值。它还将使用大约 300 MB 的非堆内存。要计算请求的内存，请使用上面的计算。作为内存限制，从上面的值中减去非堆内存，并将结果分为 0.7。
- 对于每 8 个基于密码的用户登录，在三节点集群中，每个 Pod 的每个 Pod 1 个 vCPU （每秒测试最多 300 个）。
  红帽构建的 Keycloak 将大多数 CPU 时间哈希处理了用户提供的密码，它与哈希迭代数量成比例。
- 对于每个 450 客户端凭证，每个 Pod 在三个节点集群中每秒授予 1 个 vCPU （每秒测试最多 2000 个）。
  大多数 CPU 时间都要创建新的 TLS 连接，因为每个客户端仅运行单个请求。
- 对于每个 350 刷新令牌请求，三个集群中每个 Pod 的 1 个 vCPU （测试一次为 435 个刷新令牌请求）。

- 将 200% 的额外头空间留给 CPU 使用量，以处理负载中的高峰。这样可确保节点的快速启动，并有足够的容量来处理故障转移任务，例如在一个节点失败时重新平衡 Infinispan 缓存。当我们的测试中节流时，红帽构建的 Keycloak 的性能会显著下降。

15.1.1. 计算示例

目标大小：

- 50,000 个活跃的用户会话
- 每秒 24 次登录
- 450 客户端凭证每秒授予
- 350 刷新令牌请求每秒

计算的限制：

- 请求的 CPU：5 个 vCPU
  （每秒的24 登录 = 3 个 vCPU，450 客户端凭据会每秒授予 1 个 vCPU，350 刷新令牌 = 1 个 vCPU）
- CPU 限制：15 个 vCPU
  （允许 CPU 请求处理峰值、启动和故障转移任务的三倍。）
- 请求内存：1250 MB
  (1000 MB 基础内存，以及 50,000 个活动会话的 250 MB RAM)
- 内存限制：1360 MB
  (1250 MB 预期内存使用量减去 300 个非堆使用，除以 0.7 为单位)

15.2. 参考架构

以下设置用于检索以上设置，以便在不同的场景中运行大约 10 分钟的测试：

- 通过 ROSA 在 AWS 上部署的 OpenShift 4.14.x。
- 带有 **m5.4xlarge** 实例的 MachinePool。
- 红帽在高可用性设置中使用 Operator 和 3 个 pod 部署的 Keycloak 构建，它有两个站点处于主动/被动模式。
- 在 passthrough 模式下运行 OpenShift 的反向代理是在 Pod 上终止客户端的 TLS 连接。
- 多AZ 设置中的数据库 Amazon Aurora PostgreSQL，在主站点的可用区中有 writer 实例。
- 带有 PBKDF2 (SHA512) 210,000 哈希迭代的默认用户密码散列迭代，这是 OWASP 推荐的 默认用户密码。
- 客户端凭据授予不使用刷新令牌（这是默认设置）。
- 数据库已看到有 20,000 个用户和 20,000 个客户端。
- Infinispan 本地缓存的默认 10,000 条目，因此并非所有客户端和用户都适合缓存，有些请求需要从数据库获取数据。

- 默认情况下，分布式缓存中的所有会话都有两个所有者，每个条目有两个所有者，允许一个失败 Pod，而不会丢失数据。

第 16 章 用于自动化 DATA GRID CLI 命令的概念

当与 Kubernetes 中的外部 Data Grid 交互时，**Batch** CR 允许您使用标准 **oc** 命令自动化此操作。

16.1. 何时使用它

在自动化 Kubernetes 的交互时，请使用此选项。这可避免提供用户名和密码，并检查 shell 脚本输出及其状态。

对于人类交互，CLI shell 可能仍是一个更适合的。

16.2. EXAMPLE

以下 **Batch** CR 使用站点离线，如操作流程 切换到次要站点 中所述。

```
apiVersion: infinispan.org/v2alpha1
kind: Batch
metadata:
  name: take-offline
  namespace: keycloak (1)
spec:
  cluster: infinispan (2)
  config: | (3)
    site take-offline --all-caches --site=site-a
    site status --all-caches --site=site-a
```

1. **Batch** CR 必须与 Data Grid 部署在同一命名空间中创建。
2. Infinispan CR 的名称。
3. 包含一个或多个 Data Grid CLI 命令的多行字符串。

创建 CR 后，等待状态显示完成。

```
oc -n keycloak wait --for=jsonpath='{.status.phase}'=Succeeded Batch/take-offline
```

注意

修改 **批处理** CR 实例无效。批处理操作是修改 Infinispan 资源的"一次性"事件。要为 CR 更新 **.spec** 字段，或者在批处理操作失败时，您必须创建 **Batch** CR 的新实例。

16.3. 进一步阅读

如需更多信息，请参阅 Data Grid Operator **Batch** CR 文档。