Red Hat build of Keycloak 24.0

High Availability Guide

Last Updated: 2024-06-27

Legal Notice

Abstract

This guide consists of information for administrators to configure and use the Red Hat build of Keycloak 24.0 for high availability.

Table of Contents

CHAPTER 1. MULTI-SITE DEPLOYMENTS

Red Hat build of Keycloak supports deployments that consist of multiple Red Hat build of Keycloak instances that connect to each other using its Infinispan caches; load balancers can distribute the load evenly across those instances. Those setups are intended for a transparent network on a single site.

The Red Hat build of Keycloak high-availability guide goes one step further to describe setups across multiple sites. While this setup adds additional complexity, that extra amount of high availability may be needed for some environments.

The different chapters introduce the necessary concepts and building blocks. For each building block, a blueprint shows how to set a fully functional example. Additional performance tuning and security hardening are still recommended when preparing a production setup.

CHAPTER 2. CONCEPTS FOR ACTIVE-PASSIVE DEPLOYMENTS

This topic describes a highly available active/passive setup and the behavior to expect. It outlines the requirements of the high availability active/passive architecture and describes the benefits and tradeoffs.

2.1. WHEN TO USE THIS SETUP

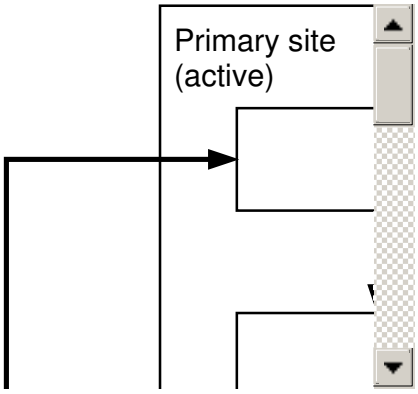
Use this setup to be able to fail over automatically in the event of a site failure, which reduces the likelihood of losing data or sessions. Manual interactions are usually required to restore the redundancy after the failover.

2.2. DEPLOYMENT, DATA STORAGE AND CACHING

Two independent Red Hat build of Keycloak deployments running in different sites are connected with a low latency network connection. Users, realms, clients, offline sessions, and other entities are stored in a database that is replicated synchronously across the two sites. The data is also cached in the Red Hat build of Keycloak Infinispan caches as local caches. When the data is changed in one Red Hat build of Keycloak instance, that data is updated in the database, and an invalidation message is sent to the other site using the replicated **work** cache.

Session-related data is stored in the replicated caches of the Infinispan caches of Red Hat build of Keycloak, and forwarded to the external Data Grid, which forwards information to the external Data Grid running synchronously in the other site. As session data of the external Data Grid is also cached in the Infinispan caches, invalidation messages of the replicated **work** cache are needed for invalidation.

In the following paragraphs and diagrams, references to deploying Data Grid apply to the external Data Grid.



2.3. CAUSES OF DATA AND SERVICE LOSS

While this setup aims for high availability, the following situations can still lead to service or data loss:

- Network failures between the sites or failures of components can lead to short service downtimes while those failures are detected. The service will be restored automatically. The system is degraded until the failures are detected and the backup cluster is promoted to service requests.
- Once failures occur in the communication between the sites, manual steps are necessary to re-synchronize a degraded setup.

- Degraded setups can lead to service or data loss if additional components fail. Monitoring is necessary to detect degraded setups.

2.4. FAILURES WHICH THIS SETUP CAN SURVIVE

Failure	Recovery	RPO[1]	RTO[2]
Database node	If the writer instance fails, the database can promote a reader instance in the same or other site to be the new writer.	No data loss	Seconds to minutes (depending on the database)
Red Hat build of Keycloak node	Multiple Red Hat build of Keycloak instances run in each site. If one instance fails, it takes a few seconds for the other nodes to notice the change, and some incoming requests might receive an error message or are delayed for some seconds.	No data loss	Less than one minute
Data Grid node	Multiple Data Grid instances run in each site. If one instance fails, it takes a few seconds for the other nodes to notice the change. Sessions are stored in at least two Data Grid nodes, so a single node failure does not lead to data loss.	No data loss	Less than one minute

Failure	Recovery	RPO[1]	RTO[2]
Data Grid cluster failure	If the Data Grid cluster fails in the active site, Red Hat build of Keycloak will not be able to communicate with the external Data Grid, and the Red Hat build of Keycloak service will be unavailable. The loadbalancer will detect the situation as **/lb-check** returns an error, and will fail over to the other site. The setup is degraded until the Data Grid cluster is restored and the session data is re-synchronized to the primary.	No data loss[3]	Seconds to minutes (depending on load balancer setup)
Connectivity Data Grid	If the connectivity between the two sites is lost, session information cannot be sent to the other site. Incoming requests might receive an error message or are delayed for some seconds. The primary site marks the secondary site offline, and will stop sending data to the secondary. The setup is degraded until the connection is restored and the session data is re-synchronized to the secondary site.	No data loss[3]	Less than one minute

Failure	Recovery	RPO[1]	RTO[2]
Connectivity database	If the connectivity between the two sites is lost, the synchronous replication will fail, and it might take some time for the primary site to mark the secondary offline. Some requests might receive an error message or be delayed for a few seconds. Manual operations might be necessary depending on the database.	No data loss[3]	Seconds to minutes (depending on the database)
Primary site	If none of the Red Hat build of Keycloak nodes are available, the loadbalancer will detect the outage and redirect the traffic to the secondary site. Some requests might receive an error message while the loadbalancer has not detected the primary site failure. The setup will be degraded until the primary site is back up and the session state has been manually synchronized from the secondary to the primary site.	No data loss[3]	Less than one minute
Secondary site	If the secondary site is not available, it will take a moment for the primary Data Grid and database to mark the secondary site offline. Some requests might receive an error message while the detection takes place. Once the secondary site is up again, the session state needs to be manually synced from the primary site to the secondary site.	No data loss[3]	Less than one minute

Table footnotes:

[1] Recovery point objective, assuming all parts of the setup were healthy at the time this occurred.

[2] Recovery time objective.

[3] Manual operations needed to restore the degraded setup.

The statement “No data loss” depends on the setup not being degraded from previous failures, which includes completing any pending manual operations to resynchronize the state between the sites.

2.5. KNOWN LIMITATIONS

Upgrades

- On Red Hat build of Keycloak or Data Grid version upgrades (major, minor and patch), all session data (except offline session) will be lost as neither supports zero downtime upgrades.

Failovers

- A successful failover requires a setup not degraded from previous failures. All manual operations like a re-synchronization after a previous failure must be complete to prevent data loss. Use monitoring to ensure degradations are detected and handled in a timely manner.

Switchovers

- A successful switchover requires a setup not degraded from previous failures. All manual operations like a re-synchronization after a previous failure must be complete to prevent data loss. Use monitoring to ensure degradations are detected and handled in a timely manner.

Out-of-sync sites

- The sites can become out of sync when a synchronous Data Grid request fails. This situation is currently difficult to monitor, and it would need a full manual re-sync of Data Grid to recover. Monitoring the number of cache entries in both sites and the Red Hat build of Keycloak log file can show when resynch would become necessary.

Manual operations

- Manual operations that re-synchronize the Data Grid state between the sites will issue a full state transfer which will put a stress on the system (network, CPU, Java heap in Data Grid and Red Hat build of Keycloak).

2.6. QUESTIONS AND ANSWERS

Why synchronous database replication?

A synchronously replicated database ensures that data written in the primary site is always available in the secondary site on failover and no data is lost.

Why synchronous Data Grid replication?

A synchronously replicated Data Grid ensures that sessions created, updated and deleted in the primary site are always available in the secondary site on failover and no data is lost.

Why is a low-latency network between sites needed?

Synchronous replication defers the response to the caller until the data is received at the secondary site. For synchronous database replication and synchronous Data Grid replication, a low latency is necessary as each request can have potentially multiple interactions between the sites when data is updated which would amplify the latency.

Why active-passive?

Some databases support a single writer instance with a reader instance which is then promoted to be the new writer once the original writer fails. In such a setup, it is beneficial for the latency to have the writer instance in the same site as the currently active Red Hat build of Keycloak. Synchronous Data Grid replication can lead to deadlocks when entries in both sites are modified concurrently.

Is this setup limited to two sites?

This setup could be extended to multiple sites, and there are no fundamental changes necessary to have, for example, three sites. Once more sites are added, the overall latency between the sites increases, and the likeliness of network failures, and therefore short downtimes, increases as well. Therefore, such a deployment is expected to have worse performance and an inferior. For now, it has been tested and documented with blueprints only for two sites.

Is a synchronous cluster less stable than an asynchronous cluster?

An asynchronous setup would handle network failures between the sites gracefully, while the synchronous setup would delay requests and will throw errors to the caller where the asynchronous setup would have deferred the writes to Data Grid or the database to the secondary site. However, as the secondary site would never be fully up-to-date with the primary site, this setup could lead to data loss during failover. This would include:

- Lost logouts, meaning sessions are logged in the secondary site although they are logged out in to the primary site at the point of failover when using an asynchronous Data Grid replication of sessions.
- Lost changes leading to users being able to log in with an old password because database changes are not replicated to the secondary site at the point of failover when using an asynchronous database.
- Invalid caches leading to users being able to log in with an old password because invalidating caches are not propagated at the point of failover to the secondary site when using an asynchronous Data Grid replication.

Therefore, tradeoffs exist between high availability and consistency. The focus of this topic is to prioritize consistency over availability with Red Hat build of Keycloak.

2.7. NEXT STEPS

Continue reading in the Building blocks active-passive deployments chapter to find blueprints for the different building blocks.

CHAPTER 3. BUILDING BLOCKS ACTIVE-PASSIVE DEPLOYMENTS

The following building blocks are needed to set up an active-passive deployment with synchronous replication.

The building blocks link to a blueprint with an example configuration. They are listed in the order in which they need to be installed.

NOTE

We provide these blueprints to show a minimal functionally complete example with a good baseline performance for regular installations. You would still need to adapt it to your environment and your organization's standards and security best practices.

3.1. PREREQUISITES

- Understanding the concepts laid out in the Concepts for active-passive deployments chapter.

3.2. TWO SITES WITH LOW-LATENCY CONNECTION

Ensures that synchronous replication is available for both the database and the external Data Grid.

Suggested setup: Two AWS Availablity Zones within the same AWS Region.

Not considered: Two regions on the same or different continents, as it would increase the latency and the likelihood of network failures. Synchronous replication of databases as a services with Aurora Regional Deployments on AWS is only available within the same region.

3.3. ENVIRONMENT FOR RED HAT BUILD OF KEYCLOAK AND DATA GRID

Ensures that the instances are deployed and restarted as needed.

Suggested setup: Red Hat OpenShift Service on AWS (ROSA) deployed in each availability zone.

Not considered: A stretched ROSA cluster which spans multiple availability zones, as this could be a single point of failure if misconfigured.

3.4. DATABASE

A synchronously replicated database across two sites.

Blueprint: Deploy AWS Aurora in multiple availability zones .

3.5. DATA GRID

A deployment of Data Grid that leverages the Data Grid's Cross-DC functionality.

Blueprint: Deploy Data Grid for HA with the Data Grid Operator using the Data Grid Operator, and connect the two sites using Data Grid's Gossip Router.

Not considered: Direct interconnections between the Kubernetes clusters on the network layer. It might be considered in the future.

3.6. RED HAT BUILD OF KEYCLOAK

A clustered deployment of Red Hat build of Keycloak in each site, connected to an external Data Grid.

Blueprint: Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator together with Connect Red Hat build of Keycloak with an external Data Grid and the Aurora database.

3.7. LOAD BALANCER

A load balancer which checks the /**lb-check** URL of the Red Hat build of Keycloak deployment in each site.

Blueprint: Deploy an AWS Route 53 loadbalancer .

Not considered: AWS Global Accelerator as it supports only weighted traffic routing and not active-passive failover. To support active-passive failover, additional logic using, for example, AWS CloudWatch and AWS Lambda would be necessary to simulate the active-passive handling by adjusting the weights when the probes fail.

CHAPTER 4. DEPLOY AWS AURORA IN MULTIPLE AVAILABILITY ZONES

This topic describes how to deploy an Aurora regional deployment of a PostgreSQL instance across multiple availability zones to tolerate one or more availability zone failures in a given AWS region.

This deployment is intended to be used with the setup described in the Concepts for active-passive deployments chapter. Use this deployment with the other building blocks outlined in the Building blocks active-passive deployments chapter.

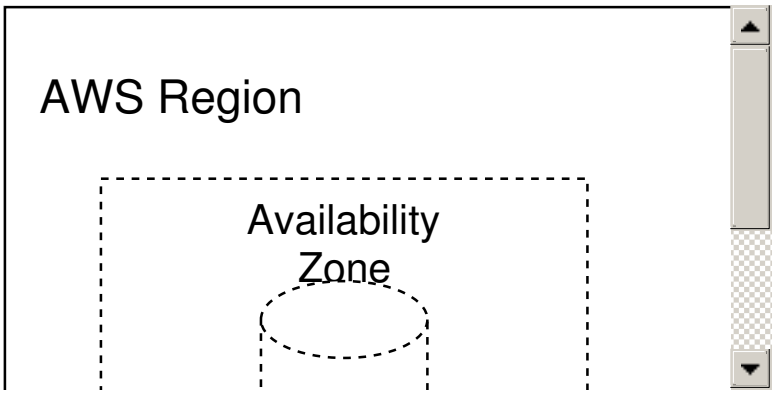
NOTE

We provide these blueprints to show a minimal functionally complete example with a good baseline performance for regular installations. You would still need to adapt it to your environment and your organization's standards and security best practices.

4.1. ARCHITECTURE

Aurora database clusters consist of multiple Aurora database instances, with one instance designated as the primary writer and all others as backup readers. To ensure high availability in the event of availability zone failures, Aurora allows database instances to be deployed across multiple zones in a single AWS region. In the event of a failure on the availability zone that is hosting the Primary database instance, Aurora automatically heals itself and promotes a reader instance from a non-failed availability zone to be the new writer instance.

Figure 4.1. Aurora Multiple Availability Zone Deployment



See the AWS Aurora documentation for more details on the semantics provided by Aurora databases.

This documentation follows AWS best practices and creates a private Aurora database that is not exposed to the Internet. To access the database from a ROSA cluster, establish a peering connection between the database and the ROSA cluster.

4.2. PROCEDURE

The following procedure contains two sections:

- Creation of an Aurora Multi-AZ database cluster with the name "keycloak-aurora" in eu-west-1.
- Creation of a peering connection between the ROSA cluster(s) and the Aurora VPC to allow applications deployed on the ROSA clusters to establish connections with the database.

4.2.1. Create Aurora database Cluster

1. Create a VPC for the Aurora cluster

Command:

```
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 We add an optional tag with the name of the Aurora cluster so that we can easily retrieve the VPC.

Output:

```
{
    "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. Create a subnet for each availability zone that Aurora will be deployed to, using the **VpcId** of the newly created VPC.

NOTE

The cidr-block range specified for each of the availability zones must not overlap.

a. Zone A

Command:

```
aws ec2 create-subnet \
  --availability-zone "eu-west-1a" \
  --vpc-id vpc-0b40bd7c59dbe4277 \
```

```
  --cidr-block 192.168.0.0/19 \
  --region eu-west-1
```

Output:


```
{
    "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

Command:

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

Output:


```
{
    "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. Obtain the ID of the Aurora VPC route-table

Command:

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

Output:


```
{
    "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. Associate the Aurora VPC route-table each availability zone's subnet

   a. Zone A

   Command:

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

   b. Zone B

   Command:

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

5. Create Aurora Subnet Group

   Command:

```
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. Create Aurora Security Group

   Command:

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

   Output:

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

7. Create the Aurora DB Cluster

   Command:

```
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
```

NOTE

You should replace the **--master-username** and **--master-user-password** values. The values specified here must be used when configuring the Red Hat build of Keycloak database credentials.

Output:


```
{
    "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. Create Aurora DB instances

    a. Create Zone A Writer instance

    Command:

    ```
    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. Create Zone B Reader instance

    Command:

    ```
    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. Wait for all Writer and Reader instances to be ready

    Command:

    ```
    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 endpoint URL for use by Keycloak

    Command:

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

Output:

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

4.2.2. Establish Peering Connections with ROSA clusters

Perform these steps once for each ROSA cluster that contains a Red Hat build of Keycloak deployment.

1. Retrieve the Aurora VPC

   Command:

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

   Output:

   ```
   vpc-0b40bd7c59dbe4277
   ```

2. Retrieve the ROSA cluster VPC

   a. Login to the ROSA cluster using **oc**

   b. Retrieve the ROSA VPC

      Command:

      ```
      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
      ```

      Output:

      ```
      vpc-0b721449398429559
      ```

3. Create Peering Connection

   Command:

   ```
   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 cluster VPC

2 Aurora VPC

Output:


```
{
    "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. Wait for Peering connection to exist

   Command:

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

5. Accept the peering connection

   Command:

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

Output:


```
{
    "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. Update ROSA cluster VPC route-table

Command:

```
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 cluster VPC

2 This must be the same as the cidr-block used when creating the Aurora VPC

7. Update the Aurora Security Group

Command:

```
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 The "machine_cidr" of the ROSA cluster

Output:

```
{
    "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. VERIFYING THE CONNECTION

The simplest way to verify that a connection is possible between a ROSA cluster and an Aurora DB cluster is to deploy **psql** on the Openshift cluster and attempt to connect to the writer endpoint.

The following command creates a pod in the default namespace and establishes a **psql** connection with the Aurora cluster if possible. Upon exiting the pod shell, the pod is deleted.

```
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 user, this can be the same as **--master-username** used when creating the DB.
2. Aurora DB user-password, this can be the same as **--master—user-password** used when creating the DB.
3. The name of the Aurora DB, such as **--database-name**.
4. The name of your Aurora DB cluster.

4.4. DEPLOYING RED HAT BUILD OF KEYCLOAK

Now that an Aurora database has been established and linked with all of your ROSA clusters, the next step is to deploy Red Hat build of Keycloak as described in the Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator chapter with the JDBC url configured to use the Aurora database writer endpoint. To do this, create a **Keycloak** CR with the following adjustments:

1. Update **spec.db.url** to be **jdbc:aws-wrapper:postgresql://$HOST:5432/keycloak** where **$HOST** is the Aurora writer endpoint URL.
2. Ensure that the Secrets referenced by **spec.db.usernameSecret** and **spec.db.passwordSecret** contain usernames and passwords defined when creating Aurora.

CHAPTER 5. DEPLOY RED HAT BUILD OF KEYCLOAK FOR HA WITH THE RED HAT BUILD OF KEYCLOAK OPERATOR

This guide describes advanced Red Hat build of Keycloak configurations for Kubernetes which are load tested and will recover from single Pod failures.

These instructions are intended for use with the setup described in the Concepts for active-passive deployments chapter. Use it together with the other building blocks outlined in the Building blocks active-passive deployments chapter.

5.1. PREREQUISITES

- OpenShift or Kubernetes cluster running.
- Understanding of a Basic Red Hat build of Keycloak deployment of Red Hat build of Keycloak with the Red Hat build of Keycloak Operator.

5.2. PROCEDURE

1. Determine the sizing of the deployment using the Concepts for sizing CPU and memory resources chapter.
2. Install the Red Hat build of Keycloak Operator as described in the Red Hat build of Keycloak Operator installation chapter.
3. Deploy Aurora AWS as described in the Deploy AWS Aurora in multiple availability zones chapter.
4. Build a custom Red Hat build of Keycloak image which is prepared for usage with the Amazon Aurora PostgreSQL database.
5. Deploy the Red Hat build of Keycloak CR with the following values with the resource requests and limits calculated in the first step:

```
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) The database connection pool initial, max and min size should be identical to allow statement caching for the database. Adjust this number to meet the needs of your system. As most requests will not touch the database due to the Red Hat build of Keycloak embedded cache, this change can server several hundreds of requests per second. See the Concepts for database connection pools chapter for details.

(2) (3) Specify the URL to your custom Red Hat build of Keycloak image. If your image is optimized, set the **startOptimized** flag to **true**.

(4) Enable additional features for multi-site support like the loadbalancer probe **/lb-check**.

(5) XA transactions are not supported by the Amazon Web Services JDBC Driver.

(6) To be able to analyze the system under load, enable the metrics endpoint. The disadvantage of the setting is that the metrics will be available at the external Red Hat build of Keycloak endpoint, so you must add a filter so that the endpoint is not available from the outside. Use a reverse proxy in front of Red Hat build of Keycloak to filter out those URLs.

(7) The default setting for the internal JGroup thread pools is 200 threads maximum. The number of all Red Hat build of Keycloak threads in the StatefulSet should not exceed the number of JGroup threads to avoid a JGroup thread pool exhaustion which could stall Red Hat build of Keycloak request processing. You might consider limiting the number of Red Hat build of Keycloak threads further because multiple concurrent threads will lead to

throttling by Kubernetes once the requested CPU limit is reached. See the Concepts for configuring thread pools chapter for details.

5.3. VERIFYING THE DEPLOYMENT

Confirm that the Red Hat build of Keycloak deployment is ready.

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

5.4. OPTIONAL: LOAD SHEDDING

To enable load shedding, limit the number of queued requests.

Load shedding with max queued http requests

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

All exceeding requests are served with an HTTP 503. See the Concepts for configuring thread pools chapter about load shedding for details.

5.5. OPTIONAL: DISABLE STICKY SESSIONS

When running on OpenShift and the default passthrough Ingress setup as provided by the Red Hat build of Keycloak Operator, the load balancing done by HAProxy is done by using sticky sessions based on the IP address of the source. When running load tests, or when having a reverse proxy in front of HAProxy, you might want to disable this setup to avoid receiving all requests on a single Red Hat build of Keycloak Pod.

Add the following supplementary configuration under the **spec** in the Red Hat build of Keycloak Custom Resource to disable sticky sessions.

```
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'
```

CHAPTER 6. DEPLOY DATA GRID FOR HA WITH THE DATA GRID OPERATOR

This chapter describes the procedures required to deploy Data Grid in a multiple-cluster environment (cross-site). For simplicity, this topic uses the minimum configuration possible that allows Red Hat build of Keycloak to be used with an external Data Grid.

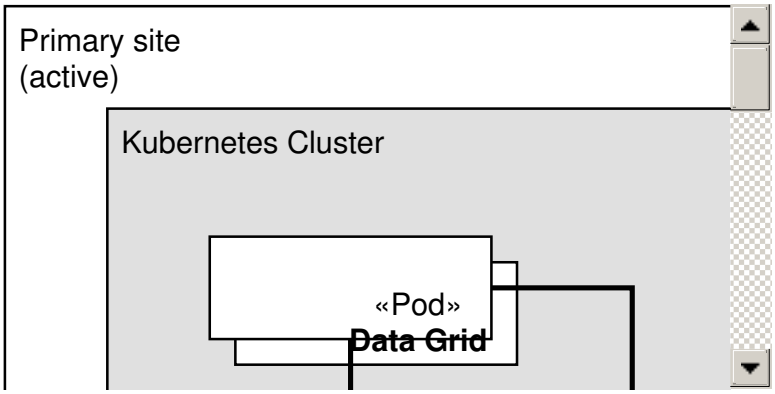
This chapter assumes two OpenShift clusters named **Site-A** and **Site-B**.

This is a building block following the concepts described in the Concepts for active-passive deployments chapter. See the Multi-site deployments chapter for an overview.

6.1. ARCHITECTURE

This setup deploys two synchronously replicating Data Grid clusters in two sites with a low-latency network connection. An example of this scenario could be two availability zones in one AWS region.

Red Hat build of Keycloak, loadbalancer and database have been removed from the following diagram for simplicity.



6.2. PREREQUISITES

- OpenShift or Kubernetes cluster running
- Understanding of the Data Grid Operator

6.3. PROCEDURE

1. Install the Data Grid Operator
2. Configure the credential to access the Data Grid cluster.
   Red Hat build of Keycloak needs this credential to be able to authenticate with the Data Grid cluster. The following **identities.yaml** file sets the username and password with admin permissions

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

The **identities.yaml** could be set in a secret as one of the following:

- As a Kubernetes Resource:

  Credential Secret

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

  (1) The **identities.yaml** from the previous example base64 encoded.

- Using the CLI

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

  Check the Configuring Authentication documentation for more details.

  These commands must be executed on both OpenShift clusters.

3. Create a service account.
   A service account is required to establish a connection between clusters. The Data Grid Operator uses it to inspect the network configuration from the remote site and to configure the local Data Grid cluster accordingly.

   For more details, see the Managing Cross-Site Connections documentation.

   a. Create a **service-account-token** secret type as follows. The same YAML file can be used in both OpenShift clusters.

      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) The secret name.

      (2) The service account name.

   b. Create the service account and generate an access token in both OpenShift clusters.

Create the service account in Site-A

```
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
```

Create the service account in Site-B

```
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. The next step is to deploy the token from **Site-A** into **Site-B** and the reverse:

Deploy Site-B token into Site-A

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

Deploy Site-A token into Site-B

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

4. Create TLS secrets
In this chapter, Data Grid uses an OpenShift Route for the cross-site communication. It uses the SNI extension of TLS to direct the traffic to the correct Pods. To achieve that, JGroups use TLS sockets, which require a Keystore and Truststore with the correct certificates.

For more information, see the Securing Cross Site Connections documentation or this Red Hat Developer Guide.

Upload the Keystore and the Truststore in an OpenShift Secret. The secret contains the file content, the password to access it, and the type of the store. Instructions for creating the certificates and the stores are beyond the scope of this guide.

To upload the Keystore as a Secret, use the following command:

Deploy a Keystore

```
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 The filename and the path to the Keystore.

2 The password to access the Keystore.

3 The Keystore type.

To upload the Truststore as a Secret, use the following command:

Deploy a 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 The filename and the path to the Truststore.

2 The password to access the Truststore.

3 The Truststore type.

NOTE

Keystore and Truststore must be uploaded in both OpenShift clusters.

5. Create a Cluster for Data Grid with Cross-Site enabled
The Setting Up Cross-Site documentation provides all the information on how to create and configure your Data Grid cluster with cross-site enabled, including the previous steps.

A basic example is provided in this chapter using the credentials, tokens, and TLS Keystore/Truststore created by the commands from the previous steps.

The Infinispan CR for Site-A

```
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. The cluster name
2. Allows the cluster to be monitored by Prometheus.
3. If using a custom credential, configure here the secret name.
4. The name of the local site, in this case **Site-A**.
5. Exposing the cross-site connection using OpenShift Route.
6. 9 The secret name where the Keystore exists as defined in the previous step.
7. 10 The alias of the certificate inside the Keystore.
8. 11 The secret key (filename) of the Keystore as defined in the previous step.
12. The secret name where the Truststore exists as defined in the previous step.
13. The Truststore key (filename) of the Keystore as defined in the previous step.
14. The remote site's name, in this case **Site-B**.
15. The namespace of the Data Grid cluster from the remote site.
16. The OpenShift API URL for the remote site.
17. The secret with the access toke to authenticate into the remote site.

For **Site-B**, the **Infinispan** CR looks similar to the above. Note the differences in point 4, 11 and 13.

The Infinispan CR for Site-B

```
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. Creating the caches for Red Hat build of Keycloak.
   Red Hat build of Keycloak requires the following caches to be present: **sessions**, **actionTokens**, **authenticationSessions**, **offlineSessions**, **clientSessions**, **offlineClientSessions**, **loginFailures**, and **work**.

   The Data Grid Cache CR allows deploying the caches in the Data Grid cluster. Cross-site needs to be enabled per cache as documented by Cross Site Documentation. The documentation contains more details about the options used by this chapter. The following example shows the **Cache** CR for **Site-A**.

   sessions in Site-A

```
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) The cross-site merge policy, invoked when there is a write-write conflict. Set this for the caches **sessions**, **authenticationSessions**, **offlineSessions**, **clientSessions** and **offlineClientSessions**, and do not set it for all other caches.

(2) The remote site name.

(3) The cross-site communication, in this case, **SYNC**.

For **Site-B**, the **Cache** CR is similar except in point 2.

session in Site-B

```
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. VERIFYING THE DEPLOYMENT

Confirm that the Data Grid cluster is formed, and the cross-site connection is established between the OpenShift clusters.

Wait until the Data Grid cluster is formed

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

Wait until the Data Grid cross-site connection is established

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

6.5. NEXT STEPS

After Data Grid is deployed and running, use the procedure in the Connect Red Hat build of Keycloak with an external Data Grid chapter to connect your Red Hat build of Keycloak cluster with the Data Grid cluster.

CHAPTER 7. CONNECT RED HAT BUILD OF KEYCLOAK WITH AN EXTERNAL DATA GRID

This topic describes advanced Data Grid configurations for Red Hat build of Keycloak on Kubernetes.

7.1. ARCHITECTURE

This connects Red Hat build of Keycloak to Data Grid using TCP connections secured by TLS 1.3. It uses the Red Hat build of Keycloak's truststore to verify Data Grid's server certificate. As Red Hat build of Keycloak is deployed using its Operator on OpenShift in the prerequisites listed below, the Operator already added the **service-ca.crt** to the truststore which is used to sign Data Grid's server certificates. In other environments, add the necessary certificates to Red Hat build of Keycloak's truststore.

7.2. PREREQUISITES

- Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator as it will be extended.
- Deploy Data Grid for HA with the Data Grid Operator .

7.3. PROCEDURE

1. Create a Secret with the username and password to connect to the external Data Grid deployment:

```
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. Extend the Red Hat build of Keycloak Custom Resource with **additionalOptions** as shown below.

NOTE

All the memory, resource and database configurations are skipped from the CR below as they have been described in Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator chapter already. Administrators should leave those configurations untouched.

```
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) The hostname of the remote Data Grid cluster.

(2) (2) The port of the remote Data Grid cluster. This is optional and it default to **11222**.

(3) (3) The Secret **name** and **key** with the Data Grid username credential.

(4) The Secret **name** and **key** with the Data Grid password credential.

(5) The **spi-connections-infinispan-quarkus-site-name** is an arbitrary Data Grid site name which Red Hat build of Keycloak needs for its Infinispan caches deployment when a remote store is used. This site-name is related only to the Infinispan caches and does not need to match any value from the external Data Grid deployment. If you are using multiple sites for Red Hat build of Keycloak in a cross-DC setup such as Deploy Data Grid for HA with the Data Grid Operator, the site name must be different in each site.

7.4. RELEVANT OPTIONS

	Value
cache-remote-host The hostname of the remote server for the remote store configuration. It replaces the **host** attribute of **remote-server** tag of the configuration specified via XML file (see **cache-config-file** option.). If the option is specified, **cache-remote-username** and **cache-remote-password** are required as well and the related configuration in XML file should not be present. CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	

	Value
cache-remote-password The password for the authentication to the remote server for the remote store. It replaces the **password** attribute of **digest** tag of the configuration specified via XML file (see **cache-config-file** option.). If the option is specified, **cache-remote-host** and **cache-remote-username** are required as well and the related configuration in XML file should not be present. CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD**	
cache-remote-port The port of the remote server for the remote store configuration. It replaces the **port** attribute of **remote-server** tag of the configuration specified via XML file (see **cache-config-file** option.). CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT**	**11222** (default)
cache-remote-username The username for the authentication to the remote server for the remote store. It replaces the **username** attribute of **digest** tag of the configuration specified via XML file (see **cache-config-file** option.). If the option is specified, **cache-remote-host** and **cache-remote-password** are required as well and the related configuration in XML file should not be present. CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME**	

CHAPTER 8. DEPLOY AN AWS ROUTE 53 LOADBALANCER

This topic describes the procedure required to configure DNS based failover for Multi-AZ Red Hat build of Keycloak clusters using AWS Route53 for an active/passive setup. These instructions are intended to be used with the setup described in the Concepts for active-passive deployments chapter. Use it together with the other building blocks outlined in the Building blocks active-passive deployments chapter.

NOTE

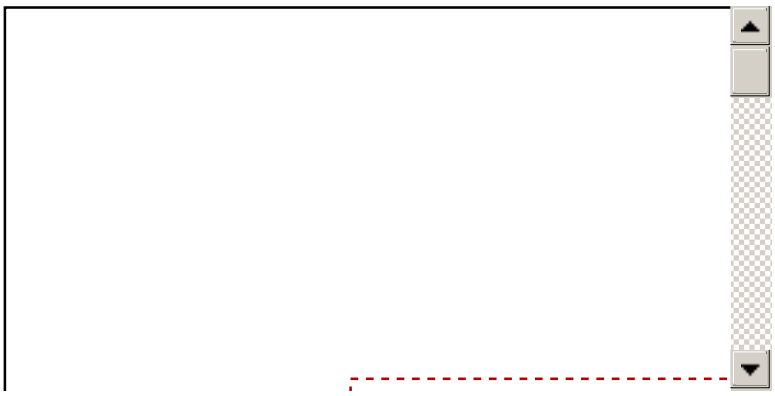
We provide these blueprints to show a minimal functionally complete example with a good baseline performance for regular installations. You would still need to adapt it to your environment and your organization's standards and security best practices.

8.1. ARCHITECTURE

All Red Hat build of Keycloak client requests are routed by a DNS name managed by Route53 records. Route53 is responsibile to ensure that all client requests are routed to the Primary cluster when it is available and healthy, or to the backup cluster in the event of the primary availability-zone or Red Hat build of Keycloak deployment failing.

If the primary site fails, the DNS changes will need to propagate to the clients. Depending on the client's settings, the propagation may take some minutes based on the client's configuration. When using mobile connections, some internet providers might not respect the TTL of the DNS entries, which can lead to an extended time before the clients can connect to the new site.

Figure 8.1. AWS Global Accelerator Failover



Two Openshift Routes are exposed on both the Primary and Backup ROSA cluster. The first Route uses the Route53 DNS name to service client requests, whereas the second Route is used by Route53 to monitor the health of the Red Hat build of Keycloak cluster.

8.2. PREREQUISITES

- Deployment of Red Hat build of Keycloak as described in Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator on a ROSA cluster running OpenShift 4.14 or later in two AWS availability zones in AWS one region.
- An owned domain for client requests to be routed through.

8.3. PROCEDURE

1. Create a Route53 Hosted Zone using the root domain name through which you want all Red Hat build of Keycloak clients to connect.
   Take note of the "Hosted zone ID", because this ID is required in later steps.

2. Retrieve the "Hosted zone ID" and DNS name associated with each ROSA cluster.
   For both the Primary and Backup cluster, perform the following steps:

   a. Log in to the ROSA cluster.

   b. Retrieve the cluster LoadBalancer Hosted Zone ID and DNS hostname

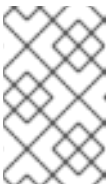
   Command:

```
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) The AWS region hosting your ROSA cluster

   Output:

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



NOTE

ROSA clusters running OpenShift 4.13 and earlier use classic load balancers instead of application load balancers. Use the **aws elb describe-load-balancers** command and an updated query string instead.

3. Create Route53 health checks

   Command:

```
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. The domain which Red Hat build of Keycloak clients should connect to. This should be the same, or a subdomain, of the root domain used to create the Hosted Zone.
2. The subdomain that will be used for health probes on the Primary cluster
3. The subdomain that will be used for health probes on the Backup cluster

Output:

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

1. The ID of the Primary Health check
2. The ID of the Backup Health check

4. Create the Route53 record set

Command:

```
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 The ID of the Hosted Zone created earlier

Output:

```
/change/C053410633T95FR9WN3YI
```

5. Wait for the Route53 records to be updated

Command:

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

6. Update or create the Red Hat build of Keycloak deployment
For both the Primary and Backup cluster, perform the following steps:

   a. Log in to the ROSA cluster

   b. Ensure the **Keycloak** CR has the following configuration

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

1 The domain clients used to connect to Red Hat build of Keycloak

To ensure that request forwarding works, edit the Red Hat build of Keycloak CR to specify the hostname through which clients will access the Red Hat build of Keycloak instances. This hostname must be the **$CLIENT_DOMAIN** used in the Route53 configuration.

   c. Create health check Route

Command:

```
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. **$NAMESPACE** should be replaced with the namespace of your Red Hat build of Keycloak deployment
2. **$DOMAIN** should be replaced with either the **PRIMARY_DOMAIN** or **BACKUP_DOMAIN**, if the current cluster is the Primary of Backup cluster, respectively.

8.4. VERIFY

Navigate to the chosen CLIENT_DOMAIN in your local browser and log in to the Red Hat build of Keycloak console.

To test failover works as expected, log in to the Primary cluster and scale the Red Hat build of Keycloak deployment to zero Pods. Scaling will cause the Primary's health checks to fail and Route53 should start routing traffic to the Red Hat build of Keycloak Pods on the Backup cluster.

CHAPTER 9. FAIL OVER TO THE SECONDARY SITE

This chapter describes the steps to fail over from primary site to secondary site in a setup as outlined in Concepts for active-passive deployments together with the blueprints outlined in Building blocks active-passive deployments.

9.1. WHEN TO USE PROCEDURE

A failover from the primary site to the secondary site will happen automatically based on the checks configured in the loadbalancer.

When the primary site loses its state in Data Grid or a network partition occurs that prevents the synchronization, manual procedures are necessary to recover the primary site before it can handle traffic again, see the Switch back to the primary site chapter.

To prevent an automatic fallback to the primary site before those manual steps have been performed, configure the loadbalancer as described following to prevent this from happening automatically.

For a graceful switch to the secondary site, follow the instructions in the Switch over to the secondary site chapter.

See the Multi-site deployments chapter for different operational procedures.

9.2. PROCEDURE

Follow these steps to manually force a failover.

9.2.1. Route53

To force Route53 to mark the primary site as permanently not available and prevent an automatic fallback, edit the health check in AWS to point to a non-existent route (**health/down**).

CHAPTER 10. SWITCH OVER TO THE SECONDARY SITE

This procedure switches from the primary site to the secondary site when using a setup as outlined in Concepts for active-passive deployments together with the blueprints outlined in Building blocks active-passive deployments.

10.1. WHEN TO USE THIS PROCEDURE

Use this procedure to gracefully take the primary offline.

Once the primary site is back online, use the chapters Recover from an out-of-sync passive site and Switch back to the primary site to return to the original state with the primary site being active.

See the Multi-site deployments chapter for different operational procedures.

10.2. PROCEDURES

10.2.1. Data Grid Cluster

For the context of this chapter, **Site-A** is the primary site and **Site-B** is the secondary site.

When you are ready to take a site offline, a good practice is to disable the replication towards it. This action prevents errors or delays when the channels are disconnected between the primary and the secondary site.

10.2.1.1. Procedures to transfer state from secondary to primary site

1. Log in into your secondary site
2. Connect into Data Grid Cluster using the Data Grid CLI tool:

   Command:

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

   It asks for the username and password for the Data Grid cluster. Those credentials are the one set in the Deploy Data Grid for HA with the Data Grid Operator chapter in the configuring credentials section.

   Output:

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

   NOTE

   The pod name depends on the cluster name defined in the Data Grid CR. The connection can be done with any pod in the Data Grid cluster.

3. Disable the replication to the primary site by running the following command:

Command:

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

Output:

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

4. Check the replication status is **offline**.

Command:

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

Output:

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

If the status is not **offline**, repeat the previous step.

The Data Grid cluster in the secondary site is ready to handle requests without trying to replicate to the primary site.

10.2.2. AWS Aurora Database

Assuming a Regional multi-AZ Aurora deployment, the current writer instance should be in the same region as the active Red Hat build of Keycloak cluster to avoid latencies and communication across availability zones.

Switching the writer instance of Aurora will lead to a short downtime. The writer instance in the other site with a slightly longer latency might be acceptable for some deployments. Therefore, this situation might be deferred to a maintenance window or skipped depending on the circumstances of the deployment.

To change the writer instance, run a failover. This change will make the database unavailable for a short time. Red Hat build of Keycloak will need to re-establish database connections.

To fail over the writer instance to the other AZ, issue the following command:

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

10.2.3. Red Hat build of Keycloak Cluster

No action required.

10.2.4. Route53

To force Route53 to mark the primary site as not available, edit the health check in AWS to point to a non-existent route (**health/down**). After some minutes, the clients will notice the change and traffic will gradually move over to the secondary site.

10.3. FURTHER READING

See Concepts to automate Data Grid CLI commands on how to automate Infinispan CLI commands.

CHAPTER 11. RECOVER FROM AN OUT-OF-SYNC PASSIVE SITE

This chapter describes the procedures required to synchronize the secondary site with the primary site in a setup as outlined in Concepts for active-passive deployments together with the blueprints outlined in Building blocks active-passive deployments.

11.1. WHEN TO USE PROCEDURE

Use this after a temporary disconnection between sites where Data Grid was disconnected and the contents of the caches are out-of-sync.

At the end of the procedure, the session contents on the secondary site have been discarded and replaced by the session contents of the primary site. All caches in the secondary site have been cleared to prevent invalid cached contents.

See the Multi-site deployments chapter for different operational procedures.

11.2. PROCEDURES

11.2.1. Data Grid Cluster

For the context of this chapter, **Site-A** is the primary site and is active, and **Site-B** is the secondary site and is passive.

Network partitions may happen between the site and the replication between the Data Grid cluster will stop. These procedures bring both sites back in sync.

WARNING

Transferring the full state may impact the Data Grid cluster performance by increasing the response time and/or resources usage.

The first procedure is to delete the stale data from the secondary site.

1. Login into your secondary site.
2. Shutdown Red Hat build of Keycloak. This will clear all Red Hat build of Keycloak caches, and it prevents the state of Red Hat build of Keycloak from being out-of-sync with Data Grid. When deploying Red Hat build of Keycloak using the Red Hat build of Keycloak Operator, change the number of Red Hat build of Keycloak instances in the Red Hat build of Keycloak Custom Resource to 0.
3. Connect into Data Grid Cluster using the Data Grid CLI tool:

   Command:

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

It asks for the username and password for the Data Grid cluster. Those credentials are the one set in the Deploy Data Grid for HA with the Data Grid Operator chapter in the configuring credentials section.

Output:

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

NOTE

The pod name depends on the cluster name defined in the Data Grid CR. The connection can be done with any pod in the Data Grid cluster.

4. Disable the replication from secondary site to the primary site by running the following command. It prevents the clear request to reach the primary site and delete all the correct cached data.

Command:

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

Output:

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

5. Check the replication status is **offline**.

Command:

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

Output:

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

If the status is not **offline**, repeat the previous step.

WARNING

Make sure the replication is **offline** otherwise the clear data will clear both sites.

6. Clear all the cached data in secondary site using the following commands:

   Command:

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

   These commands do not print any output.

7. Re-enable the cross-site replication from secondary site to the primary site.

   Command:

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

   Output:

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

8. Check the replication status is **online**.

   Command:

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

   Output:

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

Now we are ready to transfer the state from the primary site to the secondary site.

1. Login into your primary site
2. Connect into Data Grid Cluster using the Data Grid CLI tool:

   Command:

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

   It asks for the username and password for the Data Grid cluster. Those credentials are the one set in the Deploy Data Grid for HA with the Data Grid Operator chapter in the configuring credentials section.

   Output:

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

   NOTE

   The pod name depends on the cluster name defined in the Data Grid CR. The connection can be done with any pod in the Data Grid cluster.

3. Trigger the state transfer from the primary site to the secondary site.

   Command:

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

   Output:

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

4. Check the replication status is **online** for all caches.

   Command:

   -

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

Output:

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

5. Wait for the state transfer to complete by checking the output of **push-site-status** command for all caches.

Command:

```
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
```

Output:

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

Check the table in this section for the Cross-Site Documentation for the possible status values.

If an error is reported, repeat the state transfer for that specific cache.

Command:

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

6. Clear/reset the state transfer status with the following command

Command:

```
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
```

Output:

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

As now the state is available in the secondary site, Red Hat build of Keycloak can be started again:

1. Login into your secondary site.
2. Startup Red Hat build of Keycloak.
   When deploying Red Hat build of Keycloak using the Red Hat build of Keycloak Operator, change the number of Red Hat build of Keycloak instances in the Red Hat build of Keycloak Custom Resource to the original value.

11.2.2. AWS Aurora Database

No action required.

11.2.3. Route53

No action required.

11.3. FURTHER READING

See Concepts to automate Data Grid CLI commands on how to automate Infinispan CLI commands.

CHAPTER 12. SWITCH BACK TO THE PRIMARY SITE

These procedures switch back to the primary site back after a failover or switchover to the secondary site. In a setup as outlined in Concepts for active-passive deployments together with the blueprints outlined in Building blocks active-passive deployments.

12.1. WHEN TO USE THIS PROCEDURE

These procedures bring the primary site back to operation when the secondary site is handling all the traffic. At the end of the chapter, the primary site is online again and handles the traffic.

This procedure is necessary when the primary site has lost its state in Data Grid, a network partition occurred between the primary and the secondary site while the secondary site was active, or the replication was disabled as described in the Switch over to the secondary site chapter.

If the data in Data Grid on both sites is still in sync, the procedure for Data Grid can be skipped.

See the Multi-site deployments chapter for different operational procedures.

12.2. PROCEDURES

12.2.1. Data Grid Cluster

For the context of this chapter, **Site-A** is the primary site, recovering back to operation, and **Site-B** is the secondary site, running in production.

After the Data Grid in the primary site is back online and has joined the cross-site channel (see Deploy Data Grid for HA with the Data Grid Operator#verifying-the-deployment on how to verify the Data Grid deployment), the state transfer must be manually started from the secondary site.

After clearing the state in the primary site, it transfers the full state from the secondary site to the primary site, and it must be completed before the primary site can start handling incoming requests.

WARNING

Transferring the full state may impact the Data Grid cluster perform by increasing the response time and/or resources usage.

The first procedure is to delete any stale data from the primary site.

1. Log in to the primary site.
2. Shutdown Red Hat build of Keycloak. This action will clear all Red Hat build of Keycloak caches and prevents the state of Red Hat build of Keycloak from being out-of-sync with Data Grid. When deploying Red Hat build of Keycloak using the Red Hat build of Keycloak Operator, change the number of Red Hat build of Keycloak instances in the Red Hat build of Keycloak Custom Resource to 0.
3. Connect into Data Grid Cluster using the Data Grid CLI tool:

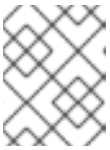
Command:

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

It asks for the username and password for the Data Grid cluster. Those credentials are the one set in the Deploy Data Grid for HA with the Data Grid Operator chapter in the configuring credentials section.

Output:

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



NOTE

The pod name depends on the cluster name defined in the Data Grid CR. The connection can be done with any pod in the Data Grid cluster.

4. Disable the replication from primary site to the secondary site by running the following command. It prevents the clear request to reach the secondary site and delete all the correct cached data.

Command:

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

Output:

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

5. Check the replication status is **offline**.

Command:

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

Output:

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

If the status is not **offline**, repeat the previous step.

WARNING

Make sure the replication is **offline** otherwise the clear data will clear both sites.

6. Clear all the cached data in primary site using the following commands:

Command:

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

These commands do not print any output.

7. Re-enable the cross-site replication from primary site to the secondary site.

Command:

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

Output:

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

8. Check the replication status is **online**.

Command:

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

Output:

```
-
```

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

Now we are ready to transfer the state from the secondary site to the primary site.

1. Log in into your secondary site.
2. Connect into Data Grid Cluster using the Data Grid CLI tool:

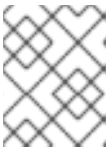
   Command:

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

   It asks for the username and password for the Data Grid cluster. Those credentials are the one set in the Deploy Data Grid for HA with the Data Grid Operator chapter in the configuring credentials section.

   Output:

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

   NOTE

   The pod name depends on the cluster name defined in the Data Grid CR. The connection can be done with any pod in the Data Grid cluster.

3. Trigger the state transfer from the secondary site to the primary site.

   Command:

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

   Output:

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

4. Check the replication status is **online** for all caches.

   Command:

   -

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

Output:

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

5. Wait for the state transfer to complete by checking the output of **push-site-status** command for all caches.

Command:

```
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
```

Output:

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

Check the table in this section for the Cross-Site Documentation for the possible status values.

If an error is reported, repeat the state transfer for that specific cache.

Command:

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

6. Clear/reset the state transfer status with the following command

   Command:

```
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
```

   Output:

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

7. Log in to the primary site.

8. Start Red Hat build of Keycloak.
   When deploying Red Hat build of Keycloak using the Red Hat build of Keycloak Operator, change the number of Red Hat build of Keycloak instances in the Red Hat build of Keycloak Custom Resource to the original value.

Both Data Grid clusters are in sync and the switchover from secondary back to the primary site can be performed.

12.2.2. AWS Aurora Database

Assuming a Regional multi-AZ Aurora deployment, the current writer instance should be in the same region as the active Red Hat build of Keycloak cluster to avoid latencies and communication across availability zones.

Switching the writer instance of Aurora will lead to a short downtime. The writer instance in the other site with a slightly longer latency might be acceptable for some deployments. Therefore, this situation might be deferred to a maintenance window or skipped depending on the circumstances of the deployment.

To change the writer instance, run a failover. This change will make the database unavailable for a short time. Red Hat build of Keycloak will need to re-establish database connections.

To fail over the writer instance to the other AZ, issue the following command:

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

12.2.3. Route53

If switching over to the secondary site has been triggered by changing the health endpoint, edit the health check in AWS to point to a correct endpoint (**health**/**live**). After some minutes, the clients will notice the change and traffic will gradually move over to the secondary site.

12.3. FURTHER READING

See Concepts to automate Data Grid CLI commands on how to automate Infinispan CLI commands.

CHAPTER 13. CONCEPTS FOR CONFIGURING THREAD POOLS

This section is intended when you want to understand the considerations and best practices on how to configure thread pools connection pools for Red Hat build of Keycloak. For a configuration where this is applied, visit Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator .

13.1. CONCEPTS

13.1.1. Quarkus executor pool

Red Hat build of Keycloak requests, as well as blocking probes, are handled by an executor pool. Depending on the available CPU cores, it has a maximum size of 200 or more threads. Threads are created as needed, and will end when no longer needed, so the system will scale up and down automatically. Red Hat build of Keycloak allows configuring the maximum thread pool size by the **http-pool-max-threads** configuration option. See Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator for an example.

When running on Kubernetes, adjust the number of worker threads to avoid creating more load than what the CPU limit allows for the Pod to avoid throttling, which would lead to congestion. When running on physical machines, adjust the number of worker threads to avoid creating more load than the node can handle to avoid congestion. Congestion would result in longer response times and an increased memory usage, and eventually an unstable system.

Ideally, you should start with a low limit of threads and adjust it accordingly to the target throughput and response time. When the load and the number of threads increases, the database connections can also become a bottleneck. Once a request cannot acquire a database connection within 5 seconds, it will fail with a message in the log like **Unable to acquire JDBC Connection**. The caller will receive a response with a 5xx HTTP status code indicating a server side error.

If you increase the number of database connections and the number of threads too much, the system will be congested under a high load with requests queueing up, which leads to a bad performance. The number of database connections is configured via the **Database** settings **db-pool-initial-size**, **db-pool-min-size** and **db-pool-max-size** respectively. Low numbers ensure fast response times for all clients, even if there is an occasionally failing request when there is a load spike.

13.1.2. JGroups connection pool

The combined number of executor threads in all Red Hat build of Keycloak nodes in the cluster should not exceed the number of threads available in JGroups thread pool to avoid the error **org.jgroups.util.ThreadPool: thread pool is full**. To see the error the first time it happens, the system property **jgroups.thread_dumps_threshold** needs to be set to **1**, as otherwise the message appears only after 10000 threads have been rejected.

The number of JGroup threads is **200** by default. While it can be configured using the property Java system property **jgroups.thread_pool.max_threads**, we advise keeping it at this value. As shown in experiments, the total number of Quarkus worker threads in the cluster must not exceed the number of threads in the JGroup thread pool of 200 in each node to avoid deadlocks in the JGroups communication. Given a Red Hat build of Keycloak cluster with four Pods, each Pod should then have 50 Quarkus worker threads. Use the Red Hat build of Keycloak configuration option **http-pool-max-threads** to configure the maximum number of Quarkus worker threads.

Use the metrics **vendor_jgroups_tcp_get_thread_pool_size** to monitor the total JGroup threads in the pool and **vendor_jgroups_tcp_get_thread_pool_size_active** for the threads active in the pool.

This is useful to monitor that limiting the Quarkus thread pool size keeps the number of active JGroup threads below the maximum JGroup thread pool size.

13.1.3. Load Shedding

By default, Red Hat build of Keycloak will queue all incoming requests infinitely, even if the request processing stalls. This will use additional memory in the Pod, can exhaust resources in the load balancers, and the requests will eventually time out on the client side without the client knowing if the request has been processed. To limit the number of queued requests in Red Hat build of Keycloak, set an additional Quarkus configuration option.

Configure **http-max-queued-requests** to specify a maximum queue length to allow for effective load shedding once this queue size is exceeded. Assuming a Red Hat build of Keycloak Pod processes around 200 requests per second, a queue of 1000 would lead to maximum waiting times of around 5 seconds.

When this setting is active, requests that exceed the number of queued requests will return with an HTTP 503 error. Red Hat build of Keycloak logs the error message in its log.

13.1.4. Probes

Red Hat build of Keycloak's liveness probe is non-blocking to avoid a restart of a Pod under a high load.

The overall health probe and the readiness can probe in some cases block to check the connection to the database, so they might fail under a high load. Due to this, a Pod can become non-ready under a high load.

13.1.5. OS Resources

In order for Java to create threads, when running on Linux it needs to have file handles available. Therefore, the number of open files (as retrieved as **ulimit -n** on Linux) need to provide head-space for Red Hat build of Keycloak to increase the number of threads needed. Each thread will also consume memory, and the container memory limits need to be set to a value that allows for this or the Pod will be killed by Kubernetes.

CHAPTER 14. CONCEPTS FOR DATABASE CONNECTION POOLS

This section is intended when you want to understand considerations and best practices on how to configure database connection pools for Red Hat build of Keycloak. For a configuration where this is applied, visit Deploy Red Hat build of Keycloak for HA with the Red Hat build of Keycloak Operator .

14.1. CONCEPTS

Creating new database connections is expensive as it takes time. Creating them when a request arrives will delay the response, so it is good to have them created before the request arrives. It can also contribute to a stampede effect where creating a lot of connections in a short time makes things worse as it slows down the system and blocks threads. Closing a connection also invalidates all server side statements caching for that connection.

For the best performance, the values for the initial, minimal and maximum database connection pool size should all be equal. This avoids creating new database connections when a new request comes in which is costly.

Keeping the database connection open for as long as possible allows for server side statement caching bound to a connection. In the case of PostgreSQL, to use a server-side prepared statement, a query needs to be executed (by default) at least five times.

See the PostgreSQL docs on prepared statements for more information.

CHAPTER 15. CONCEPTS FOR SIZING CPU AND MEMORY RESOURCES

Use this as a starting point to size a product environment. Adjust the values for your environment as needed based on your load tests.

15.1. PERFORMANCE RECOMMENDATIONS

WARNING

- Performance will be lowered when scaling to more Pods (due to additional overhead) and using a cross-datacenter setup (due to additional traffic and operations).
- Increased cache sizes can improve the performance when Red Hat build of Keycloak instances running for a longer time. This will decrease response times and reduce IOPS on the database. Still, those caches need to be filled when an instance is restarted, so do not set resources too tight based on the stable state measured once the caches have been filled.
- Use these values as a starting point and perform your own load tests before going into production.

Summary:

- The used CPU scales linearly with the number of requests up to the tested limit below.
- The used memory scales linearly with the number of active sessions up to the tested limit below.

Recommendations:

- The base memory usage for an inactive Pod is 1000 MB of RAM.
- For each 100,000 active user sessions, add 500 MB per Pod in a three-node cluster (tested with up to 200,000 sessions).
  This assumes that each user connects to only one client. Memory requirements increase with the number of client sessions per user session (not tested yet).
- In containers, Keycloak allocates 70% of the memory limit for heap based memory. It will also use approximately 300 MB of non-heap-based memory. To calculate the requested memory, use the calculation above. As memory limit, subtract the non-heap memory from the value above and divide the result by 0.7.
- For each 8 password-based user logins per second, 1 vCPU per Pod in a three-node cluster (tested with up to 300 per second).
  Red Hat build of Keycloak spends most of the CPU time hashing the password provided by the user, and it is proportional to the number of hash iterations.
- For each 450 client credential grants per second, 1 vCPU per Pod in a three node cluster (tested with up to 2000 per second).

Most CPU time goes into creating new TLS connections, as each client runs only a single request.

- For each 350 refresh token requests per second, 1 vCPU per Pod in a three-node cluster (tested with up to 435 refresh token requests per second).
- Leave 200% extra head-room for CPU usage to handle spikes in the load. This ensures a fast startup of the node, and sufficient capacity to handle failover tasks like, for example, re-balancing Infinispan caches, when one node fails. Performance of Red Hat build of Keycloak dropped significantly when its Pods were throttled in our tests.

15.1.1. Calculation example

Target size:

- 50,000 active user sessions
- 24 logins per seconds
- 450 client credential grants per second
- 350 refresh token requests per second

Limits calculated:

- CPU requested: 5 vCPU
  (24 logins per second = 3 vCPU, 450 client credential grants per second = 1 vCPU, 350 refresh token = 1 vCPU)
- CPU limit: 15 vCPU
  (Allow for three times the CPU requested to handle peaks, startups and failover tasks)
- Memory requested: 1250 MB
  (1000 MB base memory plus 250 MB RAM for 50,000 active sessions)
- Memory limit: 1360 MB
  (1250 MB expected memory usage minus 300 non-heap-usage, divided by 0.7)

15.2. REFERENCE ARCHITECTURE

The following setup was used to retrieve the settings above to run tests of about 10 minutes for different scenarios:

- OpenShift 4.14.x deployed on AWS via ROSA.
- Machinepool with **m5.4xlarge** instances.
- Red Hat build of Keycloak deployed with the Operator and 3 pods in a high-availability setup with two sites in active/passive mode.
- OpenShift's reverse proxy running in passthrough mode were the TLS connection of the client is terminated at the Pod.
- Database Amazon Aurora PostgreSQL in a multi-AZ setup, with the writer instance in the availability zone of the primary site.

- Default user password hashing with PBKDF2(SHA512) 210,000 hash iterations which is the default as recommended by OWASP .
- Client credential grants don't use refresh tokens (which is the default).
- Database seeded with 20,000 users and 20,000 clients.
- Infinispan local caches at default of 10,000 entries, so not all clients and users fit into the cache, and some requests will need to fetch the data from the database.
- All sessions in distributed caches as per default, with two owners per entries, allowing one failing Pod without losing data.

CHAPTER 16. CONCEPTS TO AUTOMATE DATA GRID CLI COMMANDS

When interacting with an external Data Grid in Kubernetes, the **Batch** CR allows you to automate this using standard **oc** commands.

16.1. WHEN TO USE IT

Use this when automating interactions on Kubernetes. This avoids providing usernames and passwords and checking shell script outputs and their status.

For human interactions, the CLI shell might still be a better fit.

16.2. EXAMPLE

The following **Batch** CR takes a site offline as described in the operational procedure Switch over to the secondary site.

```
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. The **Batch** CR must be created in the same namespace as the Data Grid deployment.
2. The name of the Infinispan CR.
3. A multiline string containing one or more Data Grid CLI commands.

Once the CR has been created, wait for the status to show the completion.

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

NOTE

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

16.3. FURTHER READING

For more information, see the Data Grid Operator **Batch** CR documentation.