# Red Hat build of Keycloak 24.0

## 高可用性ガイド

Last Updated: 2024-06-17

## 法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

## 概要

このガイドは、Red Hat build of Keycloak 24.0 を高可用性向けに設定および使用する方法に関する管理者向け情報が記載されています。

# 目次


**多様性を受け入れるオープンソースの強化** .......... **4**

**第1章 マルチサイトデプロイメント** .......... **5**

**第2章 アクティブ/パッシブデプロイメントの概念** .......... **6**
2.1. このセットアップを使用する場合 6
2.2. デプロイメント、データストレージ、キャッシュ 6
2.3. データとサービスの損失の原因 6
2.4. このセットアップが耐えられる障害 7
2.5. 既知の制限 10
2.6. 質問と回答 10
2.7. 次のステップ 11

**第3章 アクティブ/パッシブデプロイメントのビルディングブロック** .......... **12**
3.1. 前提条件 12
3.2. 低遅延接続で接続された 2 つのサイト 12
3.3. RED HAT BUILD OF KEYCLOAK と DATA GRID の環境 12
3.4. DATABASE 12
3.5. DATA GRID 12
3.6. RED HAT BUILD OF KEYCLOAK 13
3.7. ロードバランサー 13

**第4章 複数のアベイラビリティーゾーンへの AWS AURORA のデプロイ** .......... **14**
4.1. アーキテクチャー 14
4.2. 手順 14
4.3. 接続の検証 25
4.4. RED HAT BUILD OF KEYCLOAK のデプロイ 25

**第5章 RED HAT BUILD OF KEYCLOAK OPERATOR を使用した HA 用の RED HAT BUILD OF KEYCLOAK のデプロイ** .......... **26**
5.1. 前提条件 26
5.2. 手順 26
5.3. デプロイメントの確認 28
5.4. オプション: 負荷制限 28
5.5. オプション: スティッキーセッションの無効化 28

**第6章 DATA GRID OPERATOR を使用した HA 用の DATA GRID のデプロイ** .......... **29**
6.1. アーキテクチャー 29
6.2. 前提条件 29
6.3. 手順 29
6.4. デプロイメントの確認 36
6.5. 次のステップ 36

**第7章 外部 DATA GRID への RED HAT BUILD OF KEYCLOAK の接続** .......... **37**
7.1. アーキテクチャー 37
7.2. 前提条件 37
7.3. 手順 37
7.4. 関連するオプション 38

**第8章 AWS ROUTE 53 ロードバランサーのデプロイ** .......... **40**
8.1. アーキテクチャー 40
8.2. 前提条件 40
8.3. 手順 41
8.4. 検証 45

**第9章 セカンダリーサイトへのフェイルオーバー** ..... **46**
9.1. この手順を使用する状況 46
9.2. 手順 46

**第10章 セカンダリーサイトへのスイッチオーバー** ..... **47**
10.1. この手順を使用する状況 47
10.2. 手順 47
10.3. 関連資料 49

**第11章 同期が取れていないパッシブサイトからの回復** ..... **50**
11.1. この手順を使用する状況 50
11.2. 手順 50
11.3. 関連資料 55

**第12章 プライマリーサイトへのスイッチバック** ..... **56**
12.1. この手順を使用する状況 56
12.2. 手順 56
12.3. 関連資料 62

**第13章 スレッドプールの設定の概念** ..... **63**
13.1. 概念 63

**第14章 データベース接続プールの概念** ..... **65**
14.1. 概念 65

**第15章 CPU およびメモリーリソースのサイジングの概念** ..... **66**
15.1. パフォーマンスに関する推奨事項 66
15.2. リファレンスアーキテクチャー 67

**第16章 DATA GRID CLI コマンドを自動化するための概念** ..... **69**
16.1. 使用するタイミング 69
16.2. 例 69
16.3. 関連資料 69

# 多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

# 第1章 マルチサイトデプロイメント

Red Hat build of Keycloak は、Infinispan キャッシュを使用して相互に接続する複数の Red Hat build of Keycloak インスタンスで構成されるデプロイメントをサポートします。それらのインスタンス間でロードバランサーにより負荷を均等に分散できます。このようなセットアップは、1 つのサイトの透過的なネットワークを想定したものです。

Red Hat build of Keycloak 高可用性ガイドでは、さらに一歩進んで、複数のサイトにわたるセットアップについて説明します。このようなセットアップでは複雑さがさらに増し、環境によっては高可用性が必要になる場合があります。

それぞれの章で、必要な概念とビルディングブロックを紹介します。各ビルディングブロックのブループリントは、完全に機能するサンプルを設定する方法を示しています。ただし、実稼働環境のセットアップを準備する際には、追加のパフォーマンスチューニングとセキュリティーハードニングを実施することを推奨します。

# 第2章 アクティブ/パッシブデプロイメントの概念

このトピックでは、アクティブ/パッシブの高可用性セットアップと予期される動作を説明します。アクティブ/パッシブ高可用性アーキテクチャーの要件を概説し、利点とトレードオフについて説明します。

## 2.1. このセットアップを使用する場合

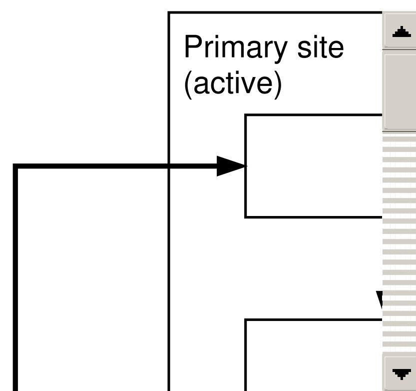
このセットアップを使用すると、サイトに障害が発生した場合に自動的にフェイルオーバーできるようになり、データやセッションが失われる可能性が低くなります。フェイルオーバー後に冗長性を復元するには、通常、手動による操作が必要です。

## 2.2. デプロイメント、データストレージ、キャッシュ

異なるサイトで実行されている 2 つの独立した Red Hat build of Keycloak デプロイメントを、低遅延のネットワーク接続で接続します。ユーザー、レルム、クライアント、オフラインセッション、およびその他のエンティティーを、2 つのサイト間で同期的にレプリケートされるデータベースに保存します。データは、ローカルキャッシュとして Red Hat build of Keycloak Infinispan キャッシュにもキャッシュされます。一方の Red Hat build of Keycloak インスタンスでデータが変更されると、そのデータがデータベース内で更新され、レプリケートされた **work** キャッシュを使用して無効化メッセージが他方のサイトに送信されます。

セッション関連のデータは、Red Hat build of Keycloak の Infinispan キャッシュのレプリケートされたキャッシュに保存され、外部 Data Grid に転送されます。外部 Data Grid は、他方のサイトで同期的に実行されている外部 Data Grid に情報を転送します。外部 Data Grid のセッションデータは、Infinispan キャッシュにもキャッシュされるため、無効化にはレプリケートされた **work** キャッシュの無効化メッセージが必要です。

以下の段落と図における Data Grid のデプロイへの言及は、外部 Data Grid を対象としています。




## 2.3. データとサービスの損失の原因

このセットアップは高可用性の実現を目的としたものですが、それでも次の状況ではサービスまたはデータの損失が発生する可能性があります。

- サイト間のネットワーク障害やコンポーネントの障害が発生すると、その障害が検出されるまでの間、短いサービスのダウンタイムが発生する可能性があります。サービスは自動的に復元されます。障害が検出され、バックアップクラスターがサービスリクエストにプロモートされるまで、システムの機能が低下します。
- サイト間の通信で障害が発生した場合、機能が低下したセットアップを再同期するために手動の手順が必要になります。

- セットアップの機能が低下すると、別のコンポーネントにさらに障害が発生した場合にサービスまたはデータの損失が発生する可能性があります。セットアップの機能低下を検出するには監視が必要です。

## 2.4. このセットアップが耐えられる障害

| 障害 | 復元 | RPO[1] | RTO[2] |
|---|---|---|---|
| データベースノード | ライターインスタンスに障害が発生した場合、データベースは同じサイトまたは別のサイトのリーダーインスタンスを新しいライターにプロモートすることができます。 | データ損失なし | 数秒から数分 (データベースによって異なる) |
| Red Hat build of Keycloak ノード | 複数の Red Hat build of Keycloak インスタンスが各サイトで実行されます。1 つのインスタンスに障害が発生した場合、他のノードが変化を認識するまでに数秒かかり、一部の受信リクエストがエラーメッセージを受信するか、数秒遅延する場合があります。 | データ損失なし | 1 分未満 |
| Data Grid ノード | 複数の Data Grid インスタンスが各サイトで実行されます。1 つのインスタンスに障害が発生した場合、他のノードが変化を認識するまでに数秒かかります。セッションが少なくとも 2 つの Data Grid ノードに保存されるため、1 つのノードの障害によってデータが失われることはありません。 | データ損失なし | 1 分未満 |

| 障害 | 復元 | RPO1 | RTO2 |
| --- | --- | --- | --- |
| Data Grid クラスターの障害 | アクティブサイトで Data Grid クラスターに障害が発生すると、Red Hat build of Keycloak が外部 Data Grid と通信できなくなり、Red Hat build of Keycloak サービスが使用できなくなります。**/lb-check** がエラーを返すと、ロードバランサーが状況を検出し、他のサイトにフェイルオーバーします。<br><br>Data Grid クラスターが復元され、セッションデータがプライマリーに再同期されるまで、セットアップの機能が低下します。 | データ損失なし [3] | 数秒から数分 (ロードバランサーの設定によって異なる) |
| 接続性 Data Grid | 2 つのサイト間の接続が失われると、セッション情報を他方のサイトに送信できなくなります。受信リクエストがエラーメッセージを受信するか、数秒遅延する場合があります。プライマリーサイトがセカンダリーサイトをオフラインとマークし、セカンダリーへのデータの送信を停止します。接続が復元され、セッションデータがセカンダリーサイトに再同期されるまで、セットアップの機能が低下します。 | データ損失なし [3] | 1 分未満 |

| 障害 | 復元 | RPO1 | RTO2 |
|---|---|---|---|
| 接続性データベース | 2 つのサイト間の接続が失われると、同期レプリケーションが失敗し、プライマリーサイトがセカンダリーサイトをオフラインとマークするまでに時間がかかることがあります。一部のリクエストがエラーメッセージを受信するか、数秒遅延する場合があります。データベースによっては手動操作が必要な場合があります。 | データ損失なし [3] | 数秒から数分 (データベースによって異なる) |
| プライマリーサイト | Red Hat build of Keycloak ノードが利用できない場合、ロードバランサーは停止を検出し、トラフィックをセカンダリーサイトにリダイレクトします。ロードバランサーがプライマリーサイトの障害を検出していないときに、一部のリクエストがエラーメッセージを受信する場合があります。プライマリーサイトがバックアップされ、セカンダリーサイトからプライマリーサイトにセッション状態が手動で同期されるまで、セットアップの機能が低下します。 | データ損失なし [3] | 1 分未満 |
| セカンダリーサイト | セカンダリーサイトが利用できない場合、プライマリーの Data Grid とデータベースがセカンダリーサイトをオフラインとマークするまでに少し時間がかかります。検出中、一部のリクエストがエラーメッセージを受信する場合があります。セカンダリーサイトが再び稼働したら、セッション状態をプライマリーサイトからセカンダリーサイトに手動で同期する必要があります。 | データ損失なし [3] | 1 分未満 |

**表の脚注:**

[1] 目標復旧時点。この障害が発生した時点でセットアップのすべての部分が健全であると仮定した場合。

[2] 目標復旧時間。

[3] 機能が低下したセットアップを復元するには手動操作が必要です。

“データ損失なし” は、以前の障害によってセットアップの機能が低下していないことを条件としています。これには、サイト間の状態を再同期する保留中の手動操作が完了していることも含まれます。

## 2.5. 既知の制限

**Upgrades**

- Red Hat build of Keycloak または Data Grid のバージョンアップグレード (メジャー、マイナー、およびパッチ) は、どちらもゼロダウンタイムアップグレードをサポートしていないため、すべてのセッションデータ (オフラインセッションを除く) が失われます。

**フェイルオーバー**

- フェイルオーバーを正常に実行するには、以前の障害によってセットアップの機能が低下していない必要があります。データ損失を防ぐために、前回の障害後の再同期などの手動操作をすべて完了する必要があります。モニタリングを使用して、機能低下をタイムリーに検出して処理してください。

**スイッチオーバー**

- スイッチオーバーを正常に実行するには、以前の障害によってセットアップの機能が低下していない必要があります。データ損失を防ぐために、前回の障害後の再同期などの手動操作をすべて完了する必要があります。モニタリングを使用して、機能低下をタイムリーに検出して処理してください。

**同期していないサイト**

- Data Grid 同期リクエストが失敗すると、サイトの同期が取れなくなることがあります。この状況を監視することは現時点では困難であり、回復するには Data Grid をすべて手動で再同期する必要があります。両方のサイトのキャッシュエントリーの数と Red Hat build of Keycloak ログファイルを監視すると、再同期が必要なタイミングを把握できます。

**手動操作**

- サイト間で Data Grid の状態を再同期する手動操作を行うと、完全な状態の遷移が実行され、システム (ネットワーク、CPU、Data Grid の Java ヒープ、および Red Hat build of Keycloak) に負荷がかかります。

## 2.6. 質問と回答

**データベース同期レプリケーションを使用するのはなぜですか?**

データベースを同期的にレプリケートすると、プライマリーサイトに書き込まれたデータがフェイルオーバー時にセカンダリーサイトで常に利用可能になり、データが失われなくなります。

**Data Grid 同期レプリケーションを使用するのはなぜですか?**

Data Grid を同期的にレプリケートすると、プライマリーサイトで作成、更新、削除されたセッションがフェイルオーバー時にセカンダリーサイトで常に利用可能になり、データが失われなくなります。

**サイト間に低遅延ネットワークが必要なのはなぜですか?**

同期レプリケーションでは、セカンダリーサイトでデータが受信されるまで、呼び出し元への応答を延期します。データベース同期レプリケーションと Data Grid 同期レプリケーションでは、データの更新時に各リクエストでサイト間のやり取りが複数回発生し、遅延が増大する可能性があるため、低遅延が必要です。

**なぜアクティブ/パッシブなのですか?**

データベースによっては、リーダーインスタンスを備えた単一のライターインスタンスをサポートしています。その場合、元のライターで障害が発生すると、リーダーインスタンスが新しいライターにプロモートします。このようなセットアップでは、現在アクティブな Red Hat build of Keycloak と同じサイトにライターインスタンスがあると、レイテンシーの点で有利です。Data Grid 同期レプリケーションでは、両方のサイトのエントリーが同時に変更されるとデッドロックが発生する可能性があります。

**このセットアップを使用できるのは 2 つのサイトだけですか?**

このセットアップは複数のサイトに拡張できます。たとえば 3 つのサイトを使用する場合、基本的な変更を加える必要はありません。サイトを追加すると、サイト間の全体的な遅延が増加し、ネットワーク障害が発生する可能性、つまり短いダウンタイムが発生する可能性も増加します。したがって、そのようなデプロイメントではパフォーマンスが低下および劣化すると予想されます。現在は、2 つのサイト用のブループリントのみを使用してテストおよびドキュメント化されています。

**同期クラスターは非同期クラスターよりも安定性に劣りますか?**

非同期セットアップでは、サイト間のネットワーク障害が正常に処理されます。一方、同期セットアップでは、非同期セットアップならセカンダリーサイトの Data Grid またはデータベースへの書き込みが延期されるような状況でも、リクエストが遅延し、呼び出し元に対してエラーが出力されます。しかし、セカンダリーサイトがプライマリーサイトと完全に同期した状態になることはないため、このセットアップではフェイルオーバー中にデータ損失が発生する可能性があります。データ損失には次のものが含まれます。

- ログアウトの喪失。つまり、セッションの Data Grid 非同期レプリケーションを使用した場合、フェイルオーバーの時点で、セッションがプライマリーサイトからはログアウトしていますが、セカンダリーサイトにはログインしている状態になります。
- 非同期データベースを使用した場合、データベースの変更がフェイルオーバー時にセカンダリーサイトにレプリケートされないため、変更が失われ、ユーザーが古いパスワードでログインできます。
- Data Grid 非同期レプリケーションを使用した場合、セカンダリーサイトへのフェイルオーバー時に無効化キャッシュが伝播されないため、無効なキャッシュによりユーザーが古いパスワードでログインできます。

したがって、高可用性と整合性の間にはトレードオフが存在します。このトピックの焦点は、Red Hat build of Keycloak の可用性よりも整合性を優先することです。

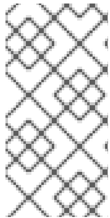
## 2.7. 次のステップ

アクティブ/パッシブデプロイメントのビルディングブロック の章を続けて読み、さまざまなビルディングブロックのブループリントを確認してください。

# 第3章 アクティブ/パッシブデプロイメントのビルディングブロック

同期レプリケーションを使用したアクティブ/パッシブデプロイメントをセットアップするには、次のビルディングブロックが必要です。

ビルディングブロックには、設定例を含むブループリントへのリンクがあります。ビルディングブロックはインストールする必要がある順序でリストされています。



**注記**

以下のブループリントは、機能的に完全な最小限の例を示すためのものであり、通常のインストールに適したベースラインのパフォーマンスを実現します。ただし、お使いの環境、組織の標準、セキュリティーのベストプラクティスに合わせて変更する必要があります。

## 3.1. 前提条件

- アクティブ/パッシブデプロイメントの概念 の章に記載されている概念を理解している。

## 3.2. 低遅延接続で接続された 2 つのサイト

同期レプリケーションをデータベースと外部 Data Grid の両方で使用できるようにします。

**推奨されるセットアップ:** 同じ AWS リージョン内の 2 つの AWS アベイラビリティーゾーン。

**考慮対象外:** 同じ大陸または異なる大陸の 2 つのリージョン。遅延が増加し、ネットワーク障害が発生する可能性が高くなります。AWS 上の Aurora リージョンデプロイメントを使用した、サービスとしてのデータベースの同期レプリケーションは、同じリージョン内でのみ利用できます。

## 3.3. RED HAT BUILD OF KEYCLOAK と DATA GRID の環境

インスタンスが必要に応じてデプロイおよび再起動されるようにします。

**推奨されるセットアップ:** 各アベイラビリティーゾーンにデプロイされた Red Hat OpenShift Service on AWS (ROSA)。

**考慮対象外:** 複数のアベイラビリティーゾーンにまたがる拡張された ROSA クラスター。設定を誤ると単一障害点になる可能性があります。

## 3.4. DATABASE

2 つのサイト間で同期レプリケートされるデータベース。

**ブループリント:** 複数のアベイラビリティーゾーンへの AWS Aurora のデプロイ

## 3.5. DATA GRID

Data Grid の Cross-DC 機能を活用した Data Grid のデプロイメント。

**ブループリント:** Data Grid Operator を使用して HA 用の Data Grid をデプロイ し、Data Grid の Gossip Router を使用して 2 つのサイトを接続します。

**考慮対象外:** ネットワーク層上の Kubernetes クラスター間の直接相互接続。将来的には考慮される可能性があります。

## 3.6. RED HAT BUILD OF KEYCLOAK

外部 Data Grid に接続された、各サイトの Red Hat build of Keycloak のクラスターデプロイメント。

**ブループリント:** Red Hat build of Keycloak Operator を使用して HA 用の Red Hat build of Keycloak をデプロイ し、Red Hat build of Keycloak を外部 Data Grid および Aurora データベースに接続 します。

## 3.7. ロードバランサー

各サイトの Red Hat build of Keycloak デプロイメントの **/lb-check** URL をチェックするロードバランサー。

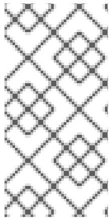
**ブループリント:** AWS Route 53 ロードバランサーをデプロイ します。

**考慮対象外:** AWS Global Accelerator は加重トラフィックルーティングのみをサポートし、アクティブ/パッシブフェイルオーバーはサポートしません。アクティブ/パッシブフェイルオーバーをサポートするには、プローブが失敗したときに重みを調整してアクティブ/パッシブ処理をシミュレートするために、AWS CloudWatch や AWS Lambda などを使用した追加のロジックが必要です。

# 第4章 複数のアベイラビリティーゾーンへの AWS AURORA のデプロイ

このトピックでは、特定の AWS リージョンにおける 1 つ以上のアベイラビリティーゾーンの障害に対応するために、複数のアベイラビリティーゾーンに PostgreSQL インスタンスの Aurora リージョンデプロイメントをデプロイする方法について説明します。

このデプロイメントは、アクティブ/パッシブデプロイメントの概念 の章で説明されているセットアップで使用することを想定しています。このデプロイメントは、アクティブ/パッシブデプロイメントのビルディングブロック の章で説明されている他のビルディングブロックとともに使用してください。



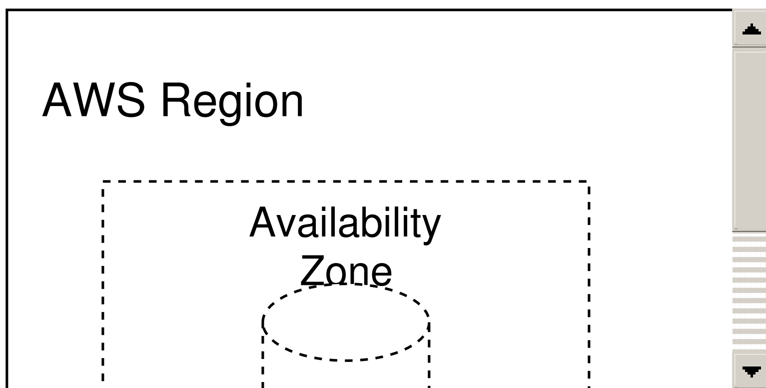
**注記**

以下のブループリントは、機能的に完全な最小限の例を示すためのものであり、通常のインストールに適したベースラインのパフォーマンスを実現します。ただし、お使いの環境、組織の標準、セキュリティーのベストプラクティスに合わせて変更する必要があります。

## 4.1. アーキテクチャー

Aurora データベースクラスターは、複数の Aurora データベースインスタンスで構成されます。1 つのインスタンスがプライマリーライターとして指定され、他のすべてのインスタンスがバックアップリーダーとして指定されます。アベイラビリティーゾーンに障害が発生した場合に高可用性を確保するために、Aurora では、単一の AWS リージョン内の複数のゾーンにデータベースインスタンスをデプロイできます。プライマリーデータベースインスタンスをホストしているアベイラビリティーゾーンで障害が発生した場合、Aurora は自動的に自己修復し、障害が発生していないアベイラビリティーゾーンのリーダーインスタンスを新しいライターインスタンスにプロモートします。

**図4.1 Aurora の複数アベイラビリティーゾーンのデプロイメント**




Aurora データベースが提供するセマンティクスの詳細は、AWS Aurora のドキュメント を参照してください。

このドキュメントは AWS のベストプラクティスに従い、インターネットに公開されないプライベート Aurora データベースを作成します。ROSA クラスターからデータベースにアクセスするには、データベースと ROSA クラスターの間にピアリング接続を確立 します。

## 4.2. 手順

次の手順には 2 つのセクションがあります。

- eu-west-1 に "keycloak-aurora" という名前の Aurora Multi-AZ データベースクラスターを作成します。

- ROSA クラスターと Aurora VPC の間にピアリング接続を作成し、ROSA クラスターにデプロイされたアプリケーションがデータベースとの接続を確立できるようにします。

### 4.2.1. Aurora データベースクラスターの作成

1. Aurora クラスターの VPC を作成します。

   **コマンド:**

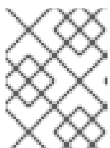
```
aws ec2 create-vpc \
  --cidr-block 192.168.0.0/16 \
  --tag-specifications "ResourceType=vpc, Tags=[{Key=AuroraCluster,Value=keycloak-aurora}]" \ 1
  --region eu-west-1
```

   1 VPC を簡単に取得できるように、Aurora クラスターの名前を含むオプションのタグを追加します。

   **出力:**

```
{
    "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. 新しく作成した VPC の **VpcId** を使用して、Aurora をデプロイする各アベイラビリティーゾーンのサブネットを作成します。



   **注記**

   各アベイラビリティーゾーンに指定した cidr ブロック範囲が重複しないようにしてください。

   a. ゾーン A

      **コマンド:**

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

**出力:**


```
{
    "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. ゾーン B

**コマンド:**

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

**出力:**


```
{
    "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. Aurora VPC ルートテーブルの ID を取得します。

**コマンド:**

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

**出力:**

```
{
    "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. Aurora VPC ルートテーブルを各アベイラビリティーゾーンのサブネットに関連付けます。

   a. ゾーン A

      **コマンド:**

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

   b. ゾーン B

      **コマンド:**

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

5. Aurora サブネットグループを作成します。

   **コマンド:**

   ```
   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. Aurora セキュリティーグループを作成します。

   **コマンド:**

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

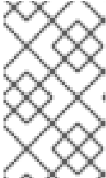
   **出力:**

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

7. Aurora DB クラスターを作成します。

   **コマンド:**

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



**注記**

**--master-username** と **--master-user-password** の値は置き換える必要があります。ここで指定した値は、Red Hat build of Keycloak データベース認証情報を設定するときに使用する必要があります。

**出力:**


```
{
  "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. Aurora DB インスタンスを作成します。

    a. ゾーン A ライターインスタンスを作成します。

    **コマンド:**

    ```
    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. ゾーン B リーダーインスタンスを作成します。

    **コマンド:**

    ```
    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. すべてのライターインスタンスとリーダーインスタンスの準備が完了するまで待ちます。

    **コマンド:**

    ```
    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. Keycloak で使用するライターエンドポイント URL を取得します。

    **コマンド:**

    ```
    aws rds describe-db-clusters \
    ```

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

**出力:**

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

### 4.2.2. ROSA クラスターとのピアリング接続の確立

Red Hat build of Keycloak デプロイメントを含む ROSA クラスターごとに、以下の手順を 1 回実行します。

1. Aurora VPC を取得します。

   **コマンド:**

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

   **出力:**

   ```
   vpc-0b40bd7c59dbe4277
   ```

2. ROSA クラスター VPC を取得します。

   a. **oc** を使用して ROSA クラスターにログインします。

   b. ROSA VPC を取得します。

      **コマンド:**

      ```
      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
      ```

      **出力:**

      ```
      vpc-0b721449398429559
      ```

3. ピアリング接続を作成します。

   **コマンド:**

```
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 クラスター VPC
2. Aurora VPC

**出力:**



```
{
    "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. ピアリング接続の存在が確認されるまで待機します。

**コマンド:**

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

5. ピアリング接続を承認します。

**コマンド:**

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

**出力:**


```
{
    "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. ROSA クラスター VPC ルートテーブルを更新します。

   **コマンド:**

```
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 クラスター VPC
2. これは Aurora VPC の作成時に使用した cidr ブロックと同じである必要があります。

7. Aurora セキュリティーグループを更新します。

**コマンド:**

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

1. ROSA クラスターの "machine_cidr"

**出力:**

```
{
    "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. 接続の検証

ROSA クラスターと Aurora DB クラスターの間で接続が可能であることを確認する最も簡単な方法は、Openshift クラスターに **psql** をデプロイし、ライターエンドポイントへの接続を試みることです。

次のコマンドは、デフォルトの namespace に Pod を作成し、可能であれば Aurora クラスターとの **psql** 接続を確立します。Pod シェルを終了すると、Pod は削除されます。

```
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 ユーザー。これは DB の作成時に使用した **--master-username** と同じものにすることができます。
2. Aurora DB ユーザーパスワード。これは DB の作成時に使用した **--master—user-password** と同じものにすることができます。
3. Aurora DB の名前 (**--database-name** など)。
4. Aurora DB クラスターの名前。

## 4.4. RED HAT BUILD OF KEYCLOAK のデプロイ

Aurora データベースを設定し、すべての ROSA クラスターにリンクしたら、次は Aurora データベースのライターエンドポイントを使用するように設定した JDBC の URL を使用して Red Hat build of Keycloak をデプロイします (Red Hat build of Keycloak Operator を使用した HA 用の Red Hat build of Keycloak のデプロイ の章を参照)。これを行うには、次のように調整した **Keycloak** CR を作成します。

1. **spec.db.url** を **jdbc:aws-wrapper:postgresql://$HOST:5432/keycloak** に更新します。**$HOST** は Aurora ライターのエンドポイント URL です。
2. **spec.db.usernameSecret** および **spec.db.passwordSecret** によって参照されるシークレットに、Aurora の作成時に定義したユーザー名とパスワードが含まれていることを確認します。

# 第5章 RED HAT BUILD OF KEYCLOAK OPERATOR を使用した HA 用の RED HAT BUILD OF KEYCLOAK のデプロイ

ここでは、1 つの Pod の障害から回復する、負荷テスト実施済みの Kubernetes 用の高度な Red Hat build of Keycloak の設定について説明します。

以下の手順は、アクティブ/パッシブデプロイメントの概念 の章で説明されているセットアップで使用することを想定しています。これは、アクティブ/パッシブデプロイメントのビルディングブロック の章で説明されている他のビルディングブロックとともに使用してください。

## 5.1. 前提条件

- OpenShift または Kubernetes クラスターが実行中である。
- Red Hat build of Keycloak Operator を使用した 基本的な Red Hat build of Keycloak のデプロイメント についての理解。

## 5.2. 手順

1. CPU およびメモリーリソースのサイジングの概念 の章を使用して、デプロイメントのサイジングを決定します。
2. Red Hat build of Keycloak Operator のインストール の章の説明に従って、Red Hat build of Keycloak Operator をインストールします。
3. 複数のアベイラビリティーゾーンへの AWS Aurora のデプロイ の章の説明に従って、Aurora AWS をデプロイします。
4. Amazon Aurora PostgreSQL データベースで使用するために準備 したカスタムの Red Hat build of Keycloak イメージをビルドします。
5. ステップ 1 で計算したリソース要求および制限を含む次の値を使用して、Red Hat build of Keycloak CR をデプロイします。

```
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. データベースのステートメントキャッシュを許可するには、データベース接続プールの初期サイズ、最大サイズ、最小サイズが同一である必要があります。この数値はシステムのニーズに合わせて調整してください。Red Hat build of Keycloak の組み込みキャッシュにより、ほとんどのリクエストはデータベースに影響を与えないため、このように変更すると、1 秒あたり数百件のリクエストを処理できます。詳細は、データベース接続プールの概念 の章を参照してください。

2. 3. Red Hat build of Keycloak イメージへの URL を指定します。イメージが最適化されている場合は、**startOptimized** フラグを **true** に設定します。

4. ロードバランサープローブ **/lb-check** などのマルチサイトサポートの追加機能を有効にします。

5. XA トランザクションは、Amazon Web Services JDBC Driver ではサポートされていません。

6. 負荷がかかっているシステムを分析できるように、メトリクスエンドポイントを有効にします。この設定の欠点は、外部の Red Hat build of Keycloak エンドポイントでメトリクスが利用可能になるため、エンドポイントが外部から利用できないようにフィルターを追加する必要があることです。Red Hat build of Keycloak の前にリバースプロキシーを使用して、これらの URL をフィルタリングして除外します。

7. 

内部 JGroup スレッドプールのデフォルト設定は、最大 200 スレッドです。StatefulSet 内の Red Hat build of Keycloak スレッドの総数が、JGroup スレッドの数を超えないよう

## 5.3. デプロイメントの確認

Red Hat build of Keycloak デプロイメントの準備ができていることを確認します。

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

## 5.4. オプション: 負荷制限

負荷制限を有効にするには、キューに入れられるリクエストの数を制限します。

**キューに入れられる HTTP リクエストの最大数による負荷制限**

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

超過したリクエストはすべて HTTP 503 で処理されます。詳細は、負荷制限に関する スレッドプールの設定の概念 の章を参照してください。

## 5.5. オプション: スティッキーセッションの無効化

HAProxy によって実行される負荷分散は、OpenShift および Red Hat build of Keycloak Operator が提供するデフォルトのパススルー Ingress セットアップで実行される場合、ソースの IP アドレスに基づくスティッキーセッションを使用して実行されます。負荷テストを実行するとき、または HAProxy の前にリバースプロキシーがあるときは、1 つの Red Hat build of Keycloak Pod ですべてのリクエストを受信しないように、この設定を無効にすることができます。

Red Hat build of Keycloak カスタムリソースの **spec** に次の補足設定を追加して、スティッキーセッションを無効にします。

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

# 第6章 DATA GRID OPERATOR を使用した HA 用の DATA GRID のデプロイ

この章では、Data Grid を複数クラスター環境 (クロスサイト) にデプロイするために必要な手順について説明します。わかりやすくするために、このトピックでは、Red Hat build of Keycloak を外部 Data Grid で使用できる最小限の設定を使用します。

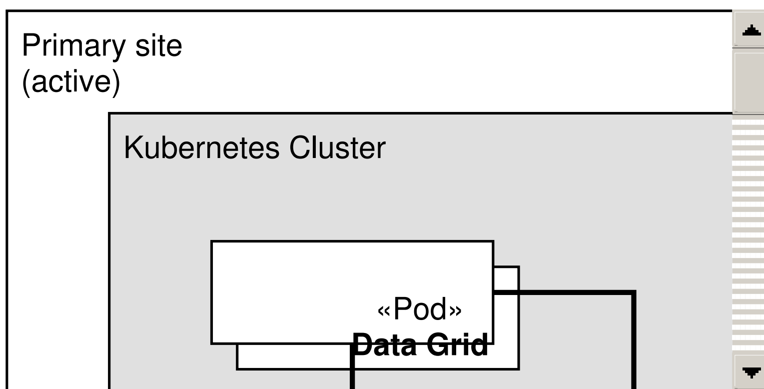
この章では、**Site-A** および **Site-B** という名前の 2 つの OpenShift クラスターを想定しています。

これは アクティブ/パッシブデプロイメントの概念の 章で説明されている概念に沿ったビルディングブロックです。概要は、マルチサイトデプロイメント の章を参照してください。

## 6.1. アーキテクチャー

このセットアップでは、低遅延のネットワーク接続で接続された 2 つのサイトに、同期的にレプリケートする 2 つの Data Grid クラスターがデプロイされています。このようなシナリオの例としては、2 つのアベイラビリティーゾーンがある 1 つの AWS リージョンが考えられます。

わかりやすくするために、Red Hat build of Keycloak、ロードバランサー、およびデータベースは次の図から削除されています。




## 6.2. 前提条件

- OpenShift または Kubernetes クラスターが実行中である。
- Data Grid Operator について理解している。

## 6.3. 手順

1. Data Grid Operator をインストールします。
2. Data Grid クラスターにアクセスするための認証情報を設定します。
   Red Hat build of Keycloak が Data Grid クラスターで認証できるようにするには、この認証情報が必要です。次の **identities.yaml** ファイルで、管理者権限を持つユーザー名とパスワードを設定します。

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

**identities.yaml** は、次のいずれかの方法でシークレットに設定できます。

- Kubernetes リソースとして:

  **認証情報シークレット**

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

  (1) Base64 でエンコードした上記の例の **identities.yaml**。

- CLI の使用

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

  詳細は、認証の設定 ドキュメントを参照してください。

  これらのコマンドは両方の OpenShift クラスターで実行する必要があります。

3. サービスアカウントを作成します。
   クラスター間の接続を確立するには、サービスアカウントが必要です。Data Grid Operator は、これを使用してリモートサイトからネットワーク設定を検査し、それに応じてローカル Data Grid クラスターを設定します。

   詳細は、マネージドのクロスサイトレプリケーション ドキュメントを参照してください。

   a. 次のように **service-account-token** シークレットタイプを作成します。同じ YAML ファイルを両方の OpenShift クラスターで使用できます。

      **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) シークレットの名前。

      (2) サービスアカウントの名前。

   b. サービスアカウントを作成し、両方の OpenShift クラスターでアクセストークンを生成します。

**Site-A でサービスアカウントを作成する**

```
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
```

**Site-B でサービスアカウントを作成する**

```
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. 次に、**Site-A** から **Site-B** にトークンをデプロイしてから、その逆を行います。

**Site-B のトークンを Site-A にデプロイする**

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

**Site-A のトークンを Site-B にデプロイする**

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

4. TLS シークレットを作成します。
この章では、Data Grid でクロスサイト通信に OpenShift ルートを使用します。この OpenShift ルートは、TLS の SNI 拡張を使用してトラフィックを正しい Pod に送信します。これを実現するために、JGroups は TLS ソケットを使用します。これには、正しい証明書を備えたキーストアとトラストストアが必要です。

詳細は、クロスサイト接続のセキュリティー保護 ドキュメントまたはこちらの Red Hat Developer Guide を参照してください。

キーストアとトラストストアは、OpenShift シークレットでアップロードします。シークレットには、ファイルの内容、ファイルにアクセスするためのパスワード、およびストアのタイプを含めます。証明書とストアを作成する手順については、このガイドの範囲外です。

キーストアをシークレットとしてアップロードするには、次のコマンドを使用します。

**キーストアのデプロイ**

```
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) ファイル名とキーストアへのパス。

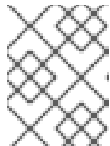
2. キーストアにアクセスするためのパスワード。
3. キーストアのタイプ。

トラストストアをシークレットとしてアップロードするには、次のコマンドを使用します。

**トラストストアのデプロイ**

```
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. ファイル名とトラストストアへのパス。
2. トラストストアにアクセスするためのパスワード。
3. トラストストアのタイプ。



**注記**

キーストアとトラストストアは、両方の OpenShift クラスターにアップロードする必要があります。

5. クロスサイトを有効にして Data Grid のクラスターを作成する
   クロスサイトの設定 ドキュメントに、上記の手順を含め、クロスサイトを有効にして Data Grid クラスターを作成および設定する方法に関するすべての情報が記載されています。

   この章では、上記の手順のコマンドによって作成された認証情報、トークン、および TLS キーストア/トラストストアを使用した基本的な例を示します。

   **Site-A の Infinispan CR**

```
apiVersion: infinispan.org/v1
kind: Infinispan
metadata:
  name: infinispan 1
  namespace: keycloak
  annotations:
    infinispan.org/monitoring: 'true' 2
spec:
  replicas: 3
  security:
    endpointSecretName: connect-secret 3
  service:
    type: DataGrid
    sites:
      local:
        name: site-a 4
        expose:
          type: Route 5
        maxRelayNodes: 128
```

```
        encryption:
          transportKeyStore:
            secretName: xsite-keystore-secret (6)
            alias: xsite (7)
            filename: keystore.p12 (8)
          routerKeyStore:
            secretName: xsite-keystore-secret (9)
            alias: xsite (10)
            filename: keystore.p12 (11)
          trustStore:
            secretName: xsite-truststore-secret (12)
            filename: truststore.p12 (13)
        locations:
        - name: site-b (14)
          clusterName: infinispan
          namespace: keycloak (15)
          url: openshift://api.site-b (16)
          secretName: xsite-token-secret (17)
```

1. クラスター名。
2. Prometheus によるクラスターの監視を許可します。
3. カスタムの認証情報を使用する場合は、ここでシークレット名を設定します。
4. ローカルサイトの名前。この場合は **Site-A** です。
5. OpenShift ルートを使用したクロスサイト接続の公開。
6. 9 前のステップで定義したキーストアが存在するシークレット名。
7. 10 キーストア内の証明書のエイリアス。
8. 11 前のステップで定義したキーストアの秘密鍵 (ファイル名)。
12. 前のステップで定義したトラストストアが存在するシークレット名。
13. 前のステップで定義したキーストアのトラストストアキー (ファイル名)。
14. リモートサイトの名前 (この場合は **Site-B**)。
15. リモートサイトの Data Grid クラスターの namespace。
16. リモートサイトの OpenShift API URL。
17. リモートサイトで認証するためのアクセストークンを含むシークレット。

**Site-B** の **Infinispan** CR も上記と同様です。ポイント 4、11、13 の違いに注意してください。

**Site-B の Infinispan CR**

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

```
  name: infinispan (1)
  namespace: keycloak
  annotations:
    infinispan.org/monitoring: 'true' (2)
spec:
  replicas: 3
  security:
    endpointSecretName: connect-secret (3)
  service:
    type: DataGrid
    sites:
      local:
        name: site-b (4)
        expose:
          type: Route (5)
        maxRelayNodes: 128
        encryption:
          transportKeyStore:
            secretName: xsite-keystore-secret (6)
            alias: xsite (7)
            filename: keystore.p12 (8)
          routerKeyStore:
            secretName: xsite-keystore-secret (9)
            alias: xsite (10)
            filename: keystore.p12 (11)
          trustStore:
            secretName: xsite-truststore-secret (12)
            filename: truststore.p12 (13)
      locations:
        - name: site-a (14)
          clusterName: infinispan
          namespace: keycloak (15)
          url: openshift://api.site-a (16)
          secretName: xsite-token-secret (17)
```

6. Red Hat build of Keycloak 用のキャッシュを作成します。
Red Hat build of Keycloak
は、**sessions**、**actionTokens**、**authenticationSessions**、**offlineSessions**、**clientSessions**、**offlineClientSessions**、**loginFailures**、および **work** キャッシュの存在を必要とします。

Data Grid Cache CR を使用すると、Data Grid クラスターにキャッシュをデプロイできます。クロスサイトのドキュメント に記載されているように、クロスサイトはキャッシュごとに有効にする必要があります。このドキュメントには、この章で使用するオプションの詳細が記載されています。次の例は、**Site-A** の **Cache** CR を示しています。

**Site-A のセッション**

```
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) クロスサイトマージポリシー。書き込みと書き込みの競合が発生した場合に呼び出されます。これは、**sessions**、**authenticationSessions**、**offlineSessions**、**clientSessions**、および **offlineClientSessions** キャッシュに設定します。他のすべてのキャッシュには設定しないでください。

(2) リモートサイト名。

(3) クロスサイト通信 (この場合は **SYNC**)。

**Site-B** の **Cache** CR もポイント 2 を除いて同様です。

**Site-B のセッション**

```
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. デプロイメントの確認

Data Grid クラスターが形成され、OpenShift クラスター間にクロスサイト接続が確立されていることを確認します。

**Data Grid クラスターが形成されるまで待機する**

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

**Data Grid のクロスサイト接続が確立されるまで待機する**

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

## 6.5. 次のステップ

Data Grid がデプロイされて実行されたら、Red Hat build of Keycloak と外部 Data Grid の接続 の章の手順を使用して、Red Hat build of Keycloak クラスターを Data Grid クラスターに接続します。

# 第7章 外部 DATA GRID への RED HAT BUILD OF KEYCLOAK の接続

このトピックでは、Kubernetes 上の Red Hat build of Keycloak の高度な Data Grid 設定について説明します。

## 7.1. アーキテクチャー

この設定では、TLS 1.3 で保護された TCP 接続を使用して、Red Hat build of Keycloak を Data Grid に接続します。Red Hat build of Keycloak のトラストストアを使用して、Data Grid のサーバー証明書を検証します。Red Hat build of Keycloak は、下記の前提条件に基づいて OpenShift 上の Operator を使用してデプロイされます。そのため、Data Grid のサーバー証明書への署名に使用されるトラストストアに **service-ca.crt** が Operator によってすでに追加されています。その他の環境では、Red Hat build of Keycloak のトラストストアに必要な証明書を追加してください。

## 7.2. 前提条件

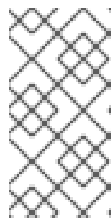
- 拡張する予定の HA 用の Red Hat build of Keycloak を、Red Hat build of Keycloak Operator を使用してデプロイ している。
- Data Grid Operator を使用して HA 用の Data Grid をデプロイ している。

## 7.3. 手順

1. 外部 Data Grid デプロイメントに接続するためのユーザー名とパスワードを使用してシークレットを作成します。

```
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. 以下に示すように、**additionalOptions** を使用して Red Hat build of Keycloak カスタムリソースを拡張します。



**注記**

メモリー、リソース、およびデータベース設定は、Red Hat build of Keycloak Operator を使用した HA 用の Red Hat build of Keycloak のデプロイ の章ですでに説明したため、すべて以下の CR から省略されています。管理者はこれらの設定をそのままにしておく必要があります。

```
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 リモート Data Grid クラスターのホスト名。

2 2 リモート Data Grid クラスターのポート。これは任意で、デフォルトは **11222** です。

3 3 Data Grid のユーザー名認証情報を含むシークレットの **name** および **key**。

4 Data Grid のパスワード認証情報を含むシークレットの **name** および **key**。

5 **spi-connections-infinispan-quarkus-site-name** は、リモートストアの使用時に Red Hat build of Keycloak が Infinispan キャッシュのデプロイメント用に必要とする任意の Data Grid サイト名です。このサイト名は、Infinispan キャッシュにのみ関連しており、外部 Data Grid デプロイメントの値と一致させる必要はありません。Data Grid Operator を使用した HA 用の Data Grid のデプロイ など、クロス DC セットアップで Red Hat build of Keycloak に複数のサイトを使用している場合、サイト名は各サイトで異なる必要があります。

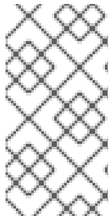
## 7.4. 関連するオプション

| | 値 |
|---|---|
| **cache-remote-host**<br><br>リモートストア設定のリモートサーバーのホスト名。<br><br>これは、XML ファイルで指定された設定の **remote-server** タグの **host** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-username** と **cache-remote-password** も必要となり、XML ファイル内に関連する設定を含めることができません。<br><br>CLI: **--cache-remote-host**<br>Env: **KC_CACHE_REMOTE_HOST** | |

| | 値 |
|---|---|
| **cache-remote-password**<br><br>リモートストアのリモートサーバーへの認証に使用するパスワード。<br><br>これは、XML ファイルで指定された設定の **digest** タグの **password** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-host** と **cache-remote-username** も必要となり、XML ファイル内に関連する設定を含めることができません。<br><br>CLI: **--cache-remote-password**<br>Env: **KC_CACHE_REMOTE_PASSWORD** | |
| **cache-remote-port**<br><br>リモートストア設定用のリモートサーバーのポート。<br><br>これは、XML ファイルで指定された設定の **remote-server** タグの **port** 属性を置き換えます (**cache-config-file** オプションを参照)。<br><br>CLI: **--cache-remote-port**<br>Env: **KC_CACHE_REMOTE_PORT** | **11222** (デフォルト) |
| **cache-remote-username**<br><br>リモートストアのリモートサーバーへの認証に使用するユーザー名。<br><br>これは、XML ファイルで指定された設定の **digest** タグの **username** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-host** と **cache-remote-password** も必要となり、XML ファイル内に関連する設定を含めることができません。<br><br>CLI: **--cache-remote-username**<br>Env: **KC_CACHE_REMOTE_USERNAME** | |

# 第8章 AWS ROUTE 53 ロードバランサーのデプロイ

このトピックでは、アクティブ/パッシブセットアップ用に AWS Route53 を使用して、Multi-AZ Red Hat build of Keycloak クラスターに DNS ベースのフェイルオーバーを設定するために必要な手順について説明します。以下の手順は、アクティブ/パッシブデプロイメントの概念 の章で説明されているセットアップで使用することを想定しています。これは、アクティブ/パッシブデプロイメントのビルディングブロック の章で説明されている他のビルディングブロックとともに使用してください。



**注記**

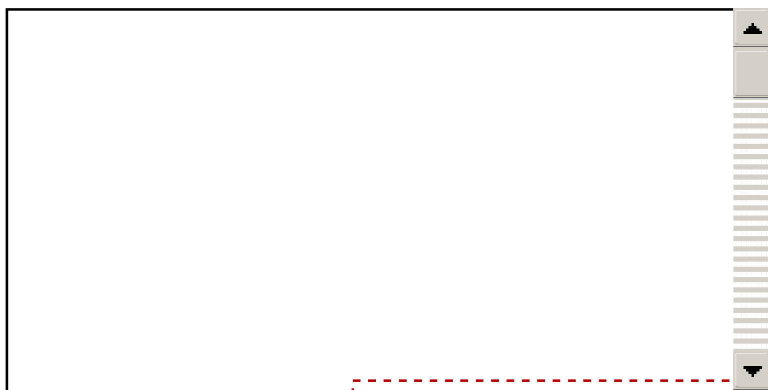
以下のブループリントは、機能的に完全な最小限の例を示すためのものであり、通常のインストールに適したベースラインのパフォーマンスを実現します。ただし、お使いの環境、組織の標準、セキュリティーのベストプラクティスに合わせて変更する必要があります。

## 8.1. アーキテクチャー

Red Hat build of Keycloak クライアントのすべてのリクエストは、Route53 レコードによって管理される DNS 名によってルーティングされます。Route53 は、すべてのクライアントリクエストを、プライマリークラスターが利用可能で健全な場合はプライマリークラスターにルーティングし、プライマリーアベイラビリティーゾーンまたは Red Hat build of Keycloak デプロイメントに障害が発生した場合はバックアップクラスターにルーティングされるようにします。

プライマリーサイトに障害が発生した場合、DNS の変更をクライアントに伝播する必要があります。クライアントの設定によっては、伝播に数分かかる場合があります。モバイル接続を使用する場合、一部のインターネットプロバイダーは DNS エントリーの TTL を順守しない場合があります。そのため、クライアントが新しいサイトに接続できるようになるまでに時間がかかる可能性があります。

**図8.1 AWS Global Accelerator のフェイルオーバー**



2 つの Openshift ルートが、プライマリー ROSA クラスターとバックアップ ROSA クラスターの両方で公開されます。1 つ目のルートは Route53 DNS 名を使用してクライアントのリクエストにサービスを提供します。2 つ目のルートは Route53 によって Red Hat build of Keycloak クラスターの健全性を監視するために使用されます。

## 8.2. 前提条件

- AWS の 1 つのリージョン内の 2 つの AWS アベイラビリティーゾーンにある、OpenShift 4.14 以降を実行している ROSA クラスター上の Red Hat build of Keycloak のデプロイメント。これについては、Red Hat build of Keycloak Operator を使用した HA 用の Red Hat build of Keycloak のデプロイ で説明されています。
- クライアント要求がルーティングされる所有ドメイン。

## 8.3. 手順

1. すべての Red Hat build of Keycloak クライアントが接続時に経由する必要があるルートドメイン名を使用して Route53 ホストゾーン を作成します。
"Hosted zone ID" は後の手順で必要になるため、メモしておきます。

2. 各 ROSA クラスターに関連付けられた "Hosted zone ID" と DNS 名を取得します。
プライマリークラスターとバックアップクラスターの両方で、次の手順を実行します。

   a. ROSA クラスターにログインします。

   b. クラスターのロードバランサーのホストゾーン ID と DNS ホスト名を取得します。

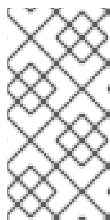
   **コマンド:**

```
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 ROSA クラスターをホストする AWS リージョン

   **出力:**

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



**注記**

OpenShift 4.13 以前を実行している ROSA クラスターは、アプリケーションロードバランサーの代わりにクラシックロードバランサーを使用します。**aws elb description-load-balancers** コマンドと更新されたクエリー文字列を使用してください。

3. Route53 ヘルスチェックを作成します。

**コマンド:**

```
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. Red Hat build of Keycloak クライアントが接続するドメイン。これは ホストゾーン の作成に使用したルートドメインと同じか、そのサブドメインである必要があります。
2. プライマリークラスター上のヘルスプローブに使用するサブドメイン
3. バックアップクラスター上のヘルスプローブに使用するサブドメイン

**出力:**

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

1. プライマリーヘルスチェックの ID
2. バックアップヘルスチェックの ID

4. Route53 レコードセットを作成します。

**コマンド:**

```
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. 前に作成した ホストゾーン の ID

**出力:**

```
/change/C053410633T95FR9WN3YI
```

5. Route53 レコードが更新されるまで待ちます。

**コマンド:**

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

6. Red Hat build of Keycloak デプロイメントを更新または作成します。
プライマリークラスターとバックアップクラスターの両方で、次の手順を実行します。

a. ROSA クラスターにログインします。

b. **Keycloak** CR が次の設定になっていることを確認します。

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

1. Red Hat build of Keycloak に接続するために使用するドメインクライアント

リクエスト転送を確実に機能させるために、Red Hat build of Keycloak CR を編集して、クライアントが Red Hat build of Keycloak インスタンスにアクセスするときに経由するホスト名を指定します。このホスト名は、Route53 の設定で使用される **$CLIENT_DOMAIN** である必要があります。

c. ヘルスチェックルートを作成します。

**コマンド:**

```
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. **$NAMESPACE** は Red Hat build of Keycloak デプロイメントの namespace に置き換えてください。
2. 現在のクラスターがバックアップクラスターのプライマリーである場合、**$DOMAIN** は **PRIMARY_DOMAIN** または **BACKUP_DOMAIN** にそれぞれ置き換えてください。

## 8.4. 検証

ローカルブラウザーで、選択した CLIENT_DOMAIN に移動し、Red Hat build of Keycloak コンソールにログインします。

フェイルオーバーが期待どおりに機能することをテストするには、プライマリークラスターにログインし、Red Hat build of Keycloak デプロイメントの Pod をゼロにスケーリングします。スケーリングによりプライマリーのヘルスチェックが失敗するため、Route53 がバックアップクラスター上の Red Hat build of Keycloak Pod へのトラフィックのルーティングを開始するはずです。

# 第9章 セカンダリーサイトへのフェイルオーバー

この章では、アクティブ/パッシブデプロイメントの概念 で説明されているセットアップと、アクティブ/パッシブデプロイメントのビルディングブロック で説明されているブループリントを使用した場合に、プライマリーサイトからセカンダリーサイトにフェイルオーバーする手順について説明します。

## 9.1. この手順を使用する状況

プライマリーサイトからセカンダリーサイトへのフェイルオーバーは、ロードバランサーで設定されたチェックに基づいて自動的に実行されます。

プライマリーサイトの状態が Data Grid で失われた場合や、同期を妨げるネットワークパーティションが発生した場合は、プライマリーサイトが再びトラフィックを処理できるように、手動の手順でプライマリーサイトを回復する必要があります。プライマリーサイトへのスイッチバック の章を参照してください。

この手動の手順が実行する前にプライマリーサイトへの自動フォールバックが行われるのを防ぐには、次の説明に従ってロードバランサーを設定して、自動フォールバックが自動的に行われないようにします。

セカンダリーサイトへの正常な切り替えについては、セカンダリーサイトへのスイッチオーバー の章の手順を参照してください。

各操作手順については、マルチサイトデプロイメント の章を参照してください。

## 9.2. 手順

フェイルオーバーを手動で強制的に実行するには、次の手順に従います。

### 9.2.1. Route53

Route53 にプライマリーサイトを永続的に利用不可とマークさせ、自動フォールバックを防ぐために、存在しないルート (**health/down**) を参照するように AWS のヘルスチェックを編集します。

# 第10章 セカンダリーサイトへのスイッチオーバー

ここでは、アクティブ/パッシブデプロイメントの概念 で説明されているセットアップと、アクティブ/パッシブデプロイメントのビルディングブロック で説明されているブループリントを使用した場合に、プライマリーサイトからセカンダリーサイトに切り替える手順について説明します。

## 10.1. この手順を使用する状況

プライマリーを正常にオフラインにするには、この手順を使用します。

プライマリーサイトがオンラインに戻ったら、非同期パッシブサイトからの回復 および プライマリーサイトへのスイッチバック の章を使用して、元の状態に戻してプライマリーサイトをアクティブにします。

各操作手順については、マルチサイトデプロイメント の章を参照してください。

## 10.2. 手順

### 10.2.1. Data Grid クラスター

この章の文脈では、**Site-A** がプライマリーサイト、**Site-A** がセカンダリーサイトとなります。

サイトをオフラインにする準備ができたら、そのサイトへのレプリケーションを無効にすることを推奨します。この操作により、プライマリーサイトとセカンダリーサイトの間でチャネルが切断されたときのエラーや遅延が防止されます。

#### 10.2.1.1. 状態をセカンダリーサイトからプライマリーサイトに転送する手順

1. セカンダリーサイトにログインします。

2. Data Grid CLI ツールを使用して Data Grid クラスターに接続します。

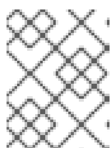
   **コマンド:**

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

   Data Grid クラスターのユーザー名とパスワードが要求されます。これらの認証情報は、Data Grid Operator を使用した HA 用の Data Grid のデプロイ の章にある認証情報の設定セクションで設定したものです。

   **出力:**

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



**注記**

Pod 名は、Data Grid CR で定義したクラスター名によって異なります。接続は、Data Grid クラスター内の任意の Pod で行うことができます。

3. 次のコマンドを実行して、プライマリーサイトへのレプリケーションを無効にします。

   **コマンド:**

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

   **出力:**

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

4. レプリケーションのステータスが **offline** であることを確認します。

   **コマンド:**

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

   **出力:**

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

   ステータスが **offline** でない場合は、前のステップを繰り返します。

セカンダリーサイトの Data Grid クラスターが、プライマリーサイトへのレプリケーションを試行せずにリクエストを処理できるようになります。

### 10.2.2. AWS Aurora データベース

リージョン内のマルチ AZ Aurora デプロイメントの場合、アベイラビリティーゾーン間の遅延と通信を回避するために、現在のライターインスタンスが、アクティブな Red Hat build of Keycloak クラスターと同じリージョン内にあるはずです。

Aurora のライターインスタンスを切り替えると、短いダウンタイムが発生します。他のサイトのライターインスタンスは、遅延が若干長くても、デプロイメントによっては許容可能な場合があります。したがって、このような状況は、デプロイメントの環境によっては、メンテナンス期間まで保留されるか、無視されることがあります。

ライターインスタンスを変更するには、フェイルオーバーを実行します。この変更により、データベースが短期間利用できなくなります。Red Hat build of Keycloak は、データベース接続を再確立する必要があります。

ライターインスタンスを他の AZ にフェイルオーバーするには、次のコマンドを発行します。

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

### 10.2.3. Red Hat build of Keycloak クラスター

アクションは不要です。

### 10.2.4. Route53

Route53 にプライマリーサイトを利用不可とマークさせるために、存在しないルート (**health/down**) を参照するように AWS のヘルスチェックを編集します。数分後、クライアントが変更を認識し、トラフィックが徐々にセカンダリーサイトに移動します。

## 10.3. 関連資料

Infinispan CLI コマンドを自動化する方法については、Data Grid CLI コマンドを自動化するための概念を参照してください。

# 第11章 同期が取れていないパッシブサイトからの回復

この章では、アクティブ/パッシブデプロイメントの概念 で説明されているセットアップと、アクティブ/パッシブデプロイメントのビルディングブロック で説明されているブループリントを使用した場合に、セカンダリーサイトをプライマリーサイトと同期するために必要な手順について説明します。

## 11.1. この手順を使用する状況

この手順は、Data Grid が切断され、キャッシュのコンテンツの同期がずれたサイト間の一時的な切断後に使用します。

手順の終了時に、セカンダリーサイトのセッションコンテンツが破棄され、プライマリーサイトのセッションコンテンツに置き換えられます。無効なキャッシュコンテンツを防ぐために、セカンダリーサイトのすべてのキャッシュがクリアされます。

各操作手順については、マルチサイトデプロイメント の章を参照してください。

## 11.2. 手順

### 11.2.1. Data Grid クラスター

この章の文脈では、**Site-A** はプライマリーサイトでアクティブ、**Site-B** はセカンダリーサイトでパッシブです。

サイト間でネットワークパーティションが発生すると、Data Grid クラスター間のレプリケーションが停止します。この手順により、両方のサイトの同期が戻ります。

**警告**

完全な状態の遷移を実行すると、応答時間やリソースの使用量が増加し、Data Grid クラスターのパフォーマンスに影響が及ぶ可能性があります。

最初の手順は、セカンダリーサイトから古いデータを削除することです。

1. セカンダリーサイトにログインします。
2. Red Hat build of Keycloak をシャットダウンします。これにより、Red Hat build of Keycloak のキャッシュをすべてクリアし、Red Hat build of Keycloak の状態と Data Grid との同期ずれを防ぎます。
   Red Hat build of Keycloak Operator を使用して Red Hat build of Keycloak をデプロイした場合、Red Hat build of Keycloak カスタムリソース内の Red Hat build of Keycloak インスタンスの数を 0 に変更します。
3. Data Grid CLI ツールを使用して Data Grid クラスターに接続します。

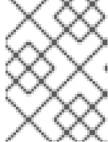
   **コマンド:**

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

Data Grid クラスターのユーザー名とパスワードが要求されます。これらの認証情報は、Data Grid Operator を使用した HA 用の Data Grid のデプロイ の章にある認証情報の設定セクションで設定したものです。

**出力:**

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



**注記**

Pod 名は、Data Grid CR で定義したクラスター名によって異なります。接続は、Data Grid クラスター内の任意の Pod で行うことができます。

4. 次のコマンドを実行して、セカンダリーサイトからプライマリーサイトへのレプリケーションを無効にします。これにより、クリアリクエストがプライマリーサイトに到達してキャッシュされた正しいデータがすべて削除されるのを防ぎます。

**コマンド:**

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

**出力:**

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

5. レプリケーションのステータスが **offline** であることを確認します。

**コマンド:**

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

**出力:**

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

ステータスが **offline** でない場合は、前のステップを繰り返します。

**警告**

レプリケーションが **offline** あることを確認してください。そうでない場合、クリアデータによって両方のサイトがクリアされます。

6. 次のコマンドを使用して、セカンダリーサイトのキャッシュデータをすべてクリアします。

 **コマンド:**

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

 これらのコマンドは何も出力しません。

7. セカンダリーサイトからプライマリーサイトへのクロスサイトレプリケーションを再度有効にします。

 **コマンド:**

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

 **出力:**

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

8. レプリケーションのステータスが **online** であることを確認します。

 **コマンド:**

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

 **出力:**

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

これで、プライマリーサイトからセカンダリーサイトに状態を転送する準備が整いました。

1. プライマリーサイトにログインします。
2. Data Grid CLI ツールを使用して Data Grid クラスターに接続します。

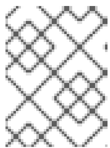
**コマンド:**

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

Data Grid クラスターのユーザー名とパスワードが要求されます。これらの認証情報は、Data Grid Operator を使用した HA 用の Data Grid のデプロイ の章にある認証情報の設定セクションで設定したものです。

**出力:**

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



**注記**

Pod 名は、Data Grid CR で定義したクラスター名によって異なります。接続は、Data Grid クラスター内の任意の Pod で行うことができます。

3. プライマリーサイトからセカンダリーサイトへの状態転送をトリガーします。

**コマンド:**

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

**出力:**

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

4. すべてのキャッシュのレプリケーションステータスが **online** であることを確認します。

**コマンド:**

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

**出力:**

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

5. すべてのキャッシュに対する **push-site-status** コマンドの出力を確認して、状態転送が完了するまで待ちます。

**コマンド:**

```
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
```

**出力:**

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

考えられるステータス値については、クロスサイトのドキュメントにあるこちらのセクションの表を参照してください。

エラーが報告された場合は、その特定のキャッシュに対して状態転送を再度実行します。

**コマンド:**

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

6. 以下のコマンドで状態転送ステータスをクリア/リセットします。

   **コマンド:**

```
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
```

   **出力:**

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

状態がセカンダリーサイトで利用できるようになったので、Red Hat build of Keycloak を再度起動できます。

1. セカンダリーサイトにログインします。
2. Red Hat build of Keycloak を起動します。
   Red Hat build of Keycloak Operator を使用して Red Hat build of Keycloak をデプロイした場合、Red Hat build of Keycloak カスタムリソース内の Red Hat build of Keycloak インスタンスの数を元の値に変更します。

### 11.2.2. AWS Aurora データベース

アクションは不要です。

### 11.2.3. Route53

アクションは不要です。

## 11.3. 関連資料

Infinispan CLI コマンドを自動化する方法については、Data Grid CLI コマンドを自動化するための概念を参照してください。

# 第12章 プライマリーサイトへのスイッチバック

ここでは、セカンダリーサイトへのフェイルオーバーまたはスイッチオーバー後に、プライマリーサイトにスイッチバックする手順について説明します。アクティブ/パッシブデプロイメントの概念 で説明されているセットアップと、アクティブ/パッシブデプロイメントのビルディングブロック で説明されているブループリントを使用します。

## 12.1. この手順を使用する状況

この手順は、セカンダリーサイトがすべてのトラフィックを処理しているときに、プライマリーサイトを動作状態に復旧するものです。この章の最後で、プライマリーサイトが再びオンラインになり、トラフィックを処理します。

この手順は、Data Grid でプライマリーサイトの状態が失われた場合、セカンダリーサイトがアクティブである間にプライマリーサイトとセカンダリーサイトの間でネットワークパーティションが発生した場合、または セカンダリーサイトへのスイッチオーバー の章で説明されているようにレプリケーションを無効にした場合に必要です。

両方のサイトの Data Grid のデータがまだ同期している場合は、Data Grid の手順をスキップできます。

各操作手順については、マルチサイトデプロイメント の章を参照してください。

## 12.2. 手順

### 12.2.1. Data Grid クラスター

この章の文脈では、**Site-A** は動作状態に回復しているプライマリーサイトであり、**Site-B** は実稼働環境で実行されているセカンダリーサイトです。

プライマリーサイトの Data Grid がオンラインに戻り、クロスサイトチャネルに参加したら (Data Grid デプロイメントを検証する方法については、Data Grid Operator を使用した HA 用の Data Grid のデプロイ#verifying-the-deployment を参照)、セカンダリーサイトから状態転送を手動で開始する必要があります。

プライマリーサイトの状態をクリアした後、セカンダリーサイトからプライマリーサイトへの完全な状態の遷移を実行します。プライマリーサイトが受信リクエストの処理を開始するには、この遷移が完了している必要があります。

**警告**

完全な状態の遷移を実行すると、応答時間やリソースの使用量が増加し、Data Grid クラスターのパフォーマンスに影響を与える可能性があります。

最初の手順は、プライマリーサイトから古いデータを削除することです。

1. プライマリーサイトにログインします。

2. Red Hat build of Keycloak をシャットダウンします。この操作により、Red Hat build of Keycloak のキャッシュをすべてクリアし、Red Hat build of Keycloak の状態と Data Grid との同期ずれを防ぎます。
Red Hat build of Keycloak Operator を使用して Red Hat build of Keycloak をデプロイした場合、Red Hat build of Keycloak カスタムリソース内の Red Hat build of Keycloak インスタンスの数を 0 に変更します。

3. Data Grid CLI ツールを使用して Data Grid クラスターに接続します。

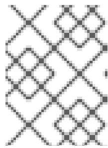
**コマンド:**

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

Data Grid クラスターのユーザー名とパスワードが要求されます。これらの認証情報は、Data Grid Operator を使用した HA 用の Data Grid のデプロイ の章にある認証情報の設定セクションで設定したものです。

**出力:**

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



**注記**

Pod 名は、Data Grid CR で定義したクラスター名によって異なります。接続は、Data Grid クラスター内の任意の Pod で行うことができます。

4. 次のコマンドを実行して、プライマリーサイトからセカンダリーサイトへのレプリケーションを無効にします。これにより、クリアリクエストがセカンダリーサイトに到達してキャッシュされた正しいデータがすべて削除されるのを防ぎます。

**コマンド:**

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

**出力:**

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

5. レプリケーションのステータスが **offline** であることを確認します。

**コマンド:**

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

**出力:**

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

ステータスが **offline** でない場合は、前のステップを繰り返します。

**警告**

レプリケーションが **offline** あることを確認してください。そうでない場合、クリアデータによって両方のサイトがクリアされます。

6. 次のコマンドを使用して、プライマリーサイトのキャッシュデータをすべてクリアします。

   **コマンド:**

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

   これらのコマンドは何も出力しません。

7. プライマリーサイトからセカンダリーサイトへのクロスサイトレプリケーションを再度有効にします。

   **コマンド:**

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

   **出力:**

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

```
  "loginFailures" : "ok",
  "actionTokens" : "ok"
}
```

8. レプリケーションのステータスが **online** であることを確認します。

**コマンド:**

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

**出力:**

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

これで、セカンダリーサイトからプライマリーサイトに状態を転送する準備が整いました。

1. セカンダリーサイトにログインします。
2. Data Grid CLI ツールを使用して Data Grid クラスターに接続します。

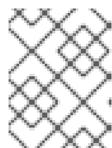
**コマンド:**

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

Data Grid クラスターのユーザー名とパスワードが要求されます。これらの認証情報は、Data Grid Operator を使用した HA 用の Data Grid のデプロイ の章にある認証情報の設定セクションで設定したものです。

**出力:**

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



**注記**

Pod 名は、Data Grid CR で定義したクラスター名によって異なります。接続は、Data Grid クラスター内の任意の Pod で行うことができます。

3. セカンダリーサイトからプライマリーサイトへの状態転送をトリガーします。

**コマンド:**

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

**出力:**

```
{
  "offlineClientSessions" : "ok",
```

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

4. すべてのキャッシュのレプリケーションステータスが **online** であることを確認します。

   **コマンド:**

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

   **出力:**

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

5. すべてのキャッシュに対する **push-site-status** コマンドの出力を確認して、状態転送が完了するまで待ちます。

   **コマンド:**

   ```
   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
   ```

   **出力:**

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

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

考えられるステータス値については、クロスサイトのドキュメントにあるこちらのセクションの表を参照してください。

エラーが報告された場合は、その特定のキャッシュに対して状態転送を再度実行します。

**コマンド:**

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

6. 以下のコマンドで状態転送ステータスをクリア/リセットします。

**コマンド:**

```
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
```

**出力:**

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

7. プライマリーサイトにログインします。

8. Red Hat build of Keycloak を起動します。
Red Hat build of Keycloak Operator を使用して Red Hat build of Keycloak をデプロイした場合、Red Hat build of Keycloak カスタムリソース内の Red Hat build of Keycloak インスタンスの数を元の値に変更します。

両方の Data Grid クラスターが同期し、セカンダリーサイトからプライマリーサイトへのスイッチオーバーを実行できるようになります。

### 12.2.2. AWS Aurora データベース

リージョン内のマルチ AZ Aurora デプロイメントの場合、アベイラビリティーゾーン間の遅延と通信を回避するために、現在のライターインスタンスが、アクティブな Red Hat build of Keycloak クラスターと同じリージョン内にあるはずです。

Aurora のライターインスタンスを切り替えると、短いダウンタイムが発生します。他のサイトのライターインスタンスは、遅延が若干長くても、デプロイメントによっては許容可能な場合があります。したがって、このような状況は、デプロイメントの環境によっては、メンテナンス期間まで保留されるか、無視されることがあります。

ライターインスタンスを変更するには、フェイルオーバーを実行します。この変更により、データベースが短期間利用できなくなります。Red Hat build of Keycloak は、データベース接続を再確立する必要があります。

ライターインスタンスを他の AZ にフェイルオーバーするには、次のコマンドを発行します。

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

### 12.2.3. Route53

ヘルスエンドポイントの変更によってセカンダリーサイトへのスイッチオーバーがトリガーされた場合は、正しいエンドポイント (**health/live**) を参照するように AWS のヘルスチェックを編集します。数分後、クライアントが変更を認識し、トラフィックが徐々にセカンダリーサイトに移動します。

## 12.3. 関連資料

Infinispan CLI コマンドを自動化する方法については、Data Grid CLI コマンドを自動化するための概念を参照してください。

# 第13章 スレッドプールの設定の概念

このセクションは、Red Hat build of Keycloak 用にスレッドプール接続プールを設定する方法に関する考慮事項とベストプラクティスを説明することを目的としています。これが適用される設定については、Red Hat build of Keycloak Operator を使用した HA 用の Red Hat build of Keycloak のデプロイ を参照してください。

## 13.1. 概念

### 13.1.1. Quarkus エグゼキュータープール

Red Hat build of Keycloak のリクエストとブロッキングプローブは、エグゼキュータープールによって処理されます。利用可能な CPU コア数に応じて、プールのサイズは最大 200 スレッド以上になります。スレッドは必要に応じて作成され、不要になると終了するため、システムは自動的にスケールアップおよびスケールダウンします。Red Hat build of Keycloak では、**http-pool-max-threads** 設定オプションによって最大スレッドプールサイズを設定できます。例については、Red Hat build of Keycloak Operator を使用した HA 用の Red Hat build of Keycloak のデプロイ を参照してください。

Kubernetes で実行する場合は、Pod に許可された CPU 制限を超える負荷が発生しないようにワーカースレッドの数を調整して、輻輳を引き起こすスロットリングを回避します。物理マシンで実行する場合は、ノードが処理できる以上の負荷が発生しないようにワーカースレッドの数を調整して、輻輳を回避します。輻輳が発生すると、応答時間が長くなり、メモリー使用量が増加し、最終的にはシステムが不安定になります。

可能であれば、下限のスレッド数から始めて、目標のスループットと応答時間に応じて数を調整してください。負荷とスレッド数が増加すると、データベース接続もボトルネックになる可能性があります。リクエストが 5 秒以内にデータベース接続を取得できないと、そのリクエストは失敗し、**Unable to acquire JDBC Connection** のようなメッセージがログに記録されます。呼び出し元は、サーバー側のエラーを示す 5xx HTTP ステータスコードを含む応答を受け取ります。

データベース接続数とスレッド数を増やしすぎると、高負荷時にリクエストがキューに蓄積されてシステムが輻輳し、パフォーマンスが低下します。データベース接続の数は、それぞれ **Database** 設定の **db-pool-initial-size**、**db-pool-min-size**、および **db-pool-max-size** によって設定されます。数値が低いと、負荷が急増したときにリクエストが失敗することがありますが、すべてのクライアントの応答時間が速くなります。

### 13.1.2. JGroups 接続プール

**org.jgroups.util.ThreadPool: thread pool is full** エラーを避けるために、クラスター内のすべての Red Hat build of Keycloak ノードのエグゼキュータースレッドの合計数が、JGroups スレッドプールで使用可能なスレッドの数を超えないようにしてください。初めてエラーが発生したときにエラーを確認するには、システムプロパティー **jgroups.thread_dumps_threshold** を **1** に設定する必要があります。そうしないと、10000 個のスレッドが拒否されるまで、メッセージが表示されません。

JGroup スレッドの数はデフォルトで **200** です。これは、Java システムプロパティー **jgroups.thread_pool.max_threads** を使用して設定できますが、この値を維持することを推奨します。実験によると、JGroups の通信のデッドロックを回避するには、クラスター内の Quarkus ワーカースレッドの合計数が、各ノードの JGroup スレッドプール内のスレッド数 (200) 以下である必要があります。4 つの Pod を持つ Red Hat build of Keycloak クラスターの場合、各 Pod の Quarkus ワーカースレッド数が 50 個である必要があります。Quarkus ワーカースレッドの最大数は、Red Hat build of Keycloak 設定オプション **http-pool-max-threads** を使用して設定します。

メトリクスの **vendor_jgroups_tcp_get_thread_pool_size** を使用してプール内の JGroup スレッドの総数を監視し、**vendor_jgroups_tcp_get_thread_pool_size_active** を使用してプール内のアクティブなスレッドを確認します。これは、Quarkus スレッドプールサイズの制限により、アクティブな

JGroup スレッドの数が JGroup スレッドプールの最大サイズ内に収まっていることを確認するのに役立ちます。

### 13.1.3. 負荷制限

デフォルトでは、Red Hat build of Keycloak は、リクエストの処理が停止した場合でも、すべての受信リクエストを無限にキューに入れます。これにより、Pod でさらにメモリーが使用され、ロードバランサーのリソースが枯渇する可能性があります。最終的には、リクエストが処理されたかどうかをクライアントが認識することなく、リクエストがクライアント側でタイムアウトになります。Red Hat build of Keycloak でキューに入れられるリクエストの数を制限するには、追加の Quarkus 設定オプションを設定します。

**http-max-queued-requests** を設定して最大キュー長を指定し、このキューサイズを超えた場合に効果的な負荷制限を適用できるようにします。Red Hat build of Keycloak Pod が 1 秒あたり約 200 のリクエストを処理すると仮定すると、キューが 1000 の場合、最大待機時間は約 5 秒になります。

この設定がアクティブな場合、キューに入れられたリクエストの数を超えるリクエストに対して、HTTP 503 エラーが返されます。Red Hat build of Keycloak は、エラーメッセージをログに記録します。

### 13.1.4. プローブ

Red Hat build of Keycloak の liveness プローブは、高負荷時の Pod の再起動を回避するために、ノンブロッキングです。

全体的なヘルスプローブと readiness プローブは、場合によってはデータベースへの接続を確認するためにブロックすることがあるため、高負荷時に失敗する可能性があります。このため、高負荷時には Pod が準備完了状態にならない可能性があります。

### 13.1.5. OS リソース

Linux 上で Java を実行する場合、Java がスレッドを作成するには、使用可能なファイルハンドルが必要です。したがって、オープンファイルの数 (Linux で **ulimit -n** で取得される数) に余裕を確保し、Red Hat build of Keycloak が必要なスレッドの数を増やせるようにする必要があります。各スレッドはメモリーも消費するため、コンテナーのメモリー制限を、これを許容する値に設定する必要があります。そうしないと、Pod が Kubernetes によって強制終了されます。

# 第14章 データベース接続プールの概念

このセクションは、Red Hat build of Keycloak 用にデータベース接続プールを設定する方法に関する考慮事項とベストプラクティスを説明することを目的としています。これが適用される設定については、Red Hat build of Keycloak Operator を使用した HA 用の Red Hat build of Keycloak のデプロイ　を参照してください。

## 14.1. 概念

新しいデータベース接続の作成には時間がかかるため、コストがかかります。リクエストが到着したときにデータベース接続を作成すると、レスポンスが遅れるため、リクエストが到着する前に作成しておくことをお勧めします。また、これは スタンピード現象 の原因となる可能性があります。つまり、短期間に大量の接続が作成されると、システムの速度が低下し、スレッドがブロックされるため、さらに状況が悪化する可能性があります。接続を閉じると、その接続のサーバー側ステートメントのキャッシュもすべて無効になります。

最適なパフォーマンスを得るには、データベース接続プールの初期サイズ、最小サイズ、最大サイズの値をすべて等しくする必要があります。これにより、新しいリクエスト受信時の新しいデータベース接続の作成とそのコストが回避されます。

データベース接続をできるだけ長く開いたままにすることで、接続にバインドされたサーバー側のステートメントキャッシュが可能になります。PostgreSQL の場合、サーバー側のプリペアドステートメントを使用するには、クエリーを (デフォルトで) 5 回以上実行する必要があります。

詳細は、プリペアドステートメントに関する PostgreSQL ドキュメント を参照してください。

# 第15章 CPU およびメモリーリソースのサイジングの概念

この章は、実稼働環境のサイジングの開始点として使用してください。負荷テストに基づいて、必要に応じて環境に合わせて値を調整してください。

## 15.1. パフォーマンスに関する推奨事項

**警告**

- より多くの Pod にスケーリングする場合 (オーバーヘッドの追加のため)、および複数のデータセンターにまたがるセットアップを使用する場合 (トラフィックと運用の拡大のため)、パフォーマンスが低下します。
- Red Hat build of Keycloak インスタンスを長時間実行する場合、キャッシュサイズを増やすと、パフォーマンスが向上する可能性があります。これにより、応答時間が短縮され、データベースの IOPS が減少します。ただし、このキャッシュはインスタンスの再起動時にいっぱいにする必要があるため、キャッシュがいっぱいになった後に測定した安定状態に基づいてリソースを厳しく設定しないでください。
- 以下の値は開始点として使用し、実稼働に移行する前に独自の負荷テストを実行してください。

概要:

- 使用される CPU は、以下のテスト済みの上限まで、リクエストの数に比例して増加します。
- 使用されるメモリーは、以下のテスト済みの上限まで、アクティブなセッションの数に比例して増加します。

推奨事項:

- 非アクティブな Pod のベースメモリー使用量は、1000 MB の RAM です。
- 100,000 のアクティブユーザーセッションにつき、3 ノードクラスター内の Pod ごとに 500 MB を追加します (200,000 セッションまでテスト済み)。
  これは、各ユーザーが 1 つのクライアントのみに接続することを前提としています。メモリー要件は、ユーザーセッションあたりのクライアントセッションの数に応じて増加します (これはまだテストされていません)。
- コンテナーでは、Keycloak はメモリー制限の 70% をヒープベースのメモリーに割り当てます。また、ヒープベース以外のメモリーも約 300 MB 使用します。要求されるメモリーを計算するには、上記の計算を使用してください。メモリー制限を求めるには、上記の値からヒープ以外のメモリーを減算し、その結果を 0.7 で割ります。
- 1 秒あたり 8 件のパスワードベースのユーザーログインにつき、3 ノードクラスター内の Pod ごとに 1 vCPU (1 秒あたり 300 までテスト済み)。
  Red Hat build of Keycloak は、ユーザーが指定したパスワードのハッシュ化にほとんどの CPU 時間を費やします。CPU 時間はハッシュ化の反復回数に比例します。

- 1 秒あたり 450 件のクライアントクレデンシャルグラントにつき、3 ノードクラスター内の Pod ごとに 1 vCPU (1 秒あたり 2000 までテスト済み)。
  各クライアントは 1 つのリクエストのみを実行するため、ほとんどの CPU 時間は新しい TLS 接続の作成に費やされます。

- 1 秒あたり 350 件のリフレッシュトークンリクエストにつき、3 ノードクラスター内の Pod ごとに 1 vCPU (1 秒あたり 435 のリフレッシュトークンリクエストまでテスト済み)。

- 負荷の急増に対処するために、CPU 使用率に 200% の余裕を残しておきます。これにより、ノードが高速で起動するようになるほか、1 つのノードに障害が発生した際のフェイルオーバータスク (Infinispan キャッシュのリバランスなど) を処理するための十分な容量が確保されます。当社のテストでは、Pod がスロットリングされたときに Red Hat build of Keycloak のパフォーマンスが大幅に低下しました。

### 15.1.1. 計算例

ターゲットサイズ:

- 50,000 のアクティブユーザーセッション
- 1 秒あたり 24 件のログイン
- 1 秒あたり 450 件のクライアントクレデンシャルグラント
- 1 秒あたり 350 件のリフレッシュトークンリクエスト

制限の算定:

- 要求される CPU: 5 vCPU
  (1 秒あたり 24 件のログイン = 3 vCPU、1 秒あたり 450 のクライアントクレデンシャルグラント = 1 vCPU、350 のリフレッシュトークン = 1 vCPU)

- CPU 制限: 15 vCPU
  (ピーク、起動、フェイルオーバータスクを処理するために、要求される CPU の 3 倍を確保)

- 要求されるメモリー: 1250 MB
  (1000 MB のベースメモリーと、50,000 のアクティブセッションに対する 250 MB RAM)

- メモリー制限: 1360 MB
  (1250 MB の予想メモリー使用量から 300 MB (ヒープ以外のメモリー使用量) を引き、0.7 で割った値)

## 15.2. リファレンスアーキテクチャー

次のセットアップを使用して上記の設定を反映し、さまざまなシナリオで約 10 分間のテストを実行しました。

- ROSA 経由で AWS にデプロイした OpenShift 4.14.x。
- **m5.4xlarge** インスタンスを含むマシンプール。
- Operator と 3 つの Pod を使用して、アクティブ/パッシブモードの 2 つのサイトを持つ高可用性セットアップでデプロイした Red Hat build of Keycloak。
- クライアントの TLS 接続を Pod で終端する、パススルーモードで実行中の OpenShift のリバースプロキシー。

- マルチ AZ セットアップのデータベースの Amazon Aurora PostgreSQL。ライターインスタンスはプライマリーサイトのアベイラビリティーゾーンにあります。
- PBKDF2(SHA512) による 210,000 回のハッシュイテレーションを使用したデフォルトのユーザーパスワードハッシュ化。これは OWASP が推奨 するデフォルトです。
- クライアントクレデンシャルグラントでリフレッシュトークンを使用しません (デフォルト)。
- 20,000 のユーザーと 20,000 のクライアントを使用してシードしたデータベース。
- デフォルトの 10,000 エントリーの Infinispan ローカルキャッシュ。そのため、すべてのクライアントとユーザーがキャッシュに収まるわけではありません。一部のリクエストはデータベースからデータを取得する必要があります。
- すべてのセッションはデフォルト設定に従って分散キャッシュにあります。1 つの Pod で障害が発生してもデータ損失が起きないように、エントリーごとに 2 人の所有者が存在します。

# 第16章 DATA GRID CLI コマンドを自動化するための概念

Kubernetes で外部 Data Grid と対話する場合、**Batch** CR を使用すると、標準の **oc** コマンドを使用して対話を自動化できます。

## 16.1. 使用するタイミング

Kubernetes での対話を自動化するときに使用します。これにより、ユーザー名とパスワードの入力や、シェルスクリプトの出力とそのステータスの確認が不要になります。

手動操作の場合は、CLI シェルの方が適している場合があります。

## 16.2. 例

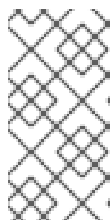
次の **Batch** CR は、セカンダリーサイトへの切り替え 操作手順で説明されているように、サイトをオフラインにします。

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

1. **Batch** CR は、Data Grid デプロイメントと同じ namespace に作成する必要があります。
2. Infinispan CR の名前。
3. 1 つ以上の Data Grid CLI コマンドを含む複数行の文字列。

CR を作成したら、ステータスに完了と表示されるまで待ちます。

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



**注記**

**Batch** CR インスタンスを変更しても効果はありません。バッチ操作は、Infinispan リソースを変更する "1 回限り" のイベントです。CR の **.spec** フィールドを更新するには、またはバッチ操作が失敗した場合には、**Batch** CR の新規インスタンスを作成する必要があります。

## 16.3. 関連資料

詳細は、Data Grid Operator **Batch** CR ドキュメント を参照してください。