



# Red Hat Single Sign-On 7.6

## サーバーインストールおよび設定ガイド

Red Hat Single Sign-On 7.6 向け



# Red Hat Single Sign-On 7.6 サーバーインストールおよび設定ガイド

---

Red Hat Single Sign-On 7.6 向け

## 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<sup>®</sup> is the registered trademark of Linus Torvalds in the United States and other countries.

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

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

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

Node.js<sup>®</sup> 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<sup>®</sup> 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

本ガイドは、Red Hat Single Sign-On 7.6 をインストールおよび設定するための情報で構成されています。

## 目次

多様性を受け入れるオープンソースの強化 .....	4
<b>第1章 ガイドの概要 .....</b>	<b>5</b>
1.1. その他の推奨される外部ドキュメント .....	5
<b>第2章 ソフトウェアのインストール .....</b>	<b>6</b>
2.1. インストールの要件 .....	6
2.2. ZIP ファイルからの RH-SSO のインストール .....	6
2.3. RPM からの RH-SSO のインストール .....	7
2.4. 重要なディレクトリー .....	9
<b>第3章 操作モードの使用 .....</b>	<b>10</b>
3.1. スタンドアロンモードの使用 .....	10
3.2. スタンドアロンクラスター化モードの使用 .....	13
3.3. ドメインクラスターモードの使用 .....	16
3.4. クロスサイトレプリケーションモードの使用 .....	25
<b>第4章 サブシステム設定の管理 .....</b>	<b>26</b>
4.1. SPI プロバイダーの設定 .....	26
4.2. JBOSS EAP CLI の起動 .....	27
4.3. CLI 組み込みモード .....	28
4.4. CLI GUI モードの使用 .....	28
4.5. CLI スクリプト .....	29
4.6. CLI レシピ .....	29
<b>第5章 プロファイル .....</b>	<b>32</b>
<b>第6章 リレーショナルデータベースの設定 .....</b>	<b>34</b>
6.1. データベース設定チェックリスト .....	34
6.2. JDBC ドライバーのパッケージ化 .....	34
6.3. JDBC ドライバーの宣言とロード .....	36
6.4. RED HAT SINGLE SIGN-ON データソースの変更 .....	37
6.5. データベースの設定 .....	38
6.6. データベースの UNICODE の考慮事項 .....	39
<b>第7章 パブリックホスト名の使用 .....</b>	<b>42</b>
7.1. デフォルトのプロバイダー .....	42
7.2. カスタムプロバイダー .....	43
<b>第8章 ネットワークの設定 .....</b>	<b>44</b>
8.1. バインドアドレス .....	44
8.2. ソケットポートバインディング .....	45
8.3. HTTPS/SSL .....	46
8.4. RED HAT SINGLE SIGN-ON サーバーの HTTPS/SSL の有効化 .....	46
8.5. 送信 HTTP 要求 .....	51
<b>第9章 RED HAT SINGLE SIGN-ON をクラスターで実行するための設定 .....</b>	<b>56</b>
9.1. 推奨されるネットワークアーキテクチャー .....	56
9.2. クラスタリングの例 .....	56
9.3. ロードバランサーまたはプロキシの設定 .....	56
9.4. ステイキセッション .....	61
9.5. マルチキャストネットワークの設定 .....	63
9.6. 安全なクラスター通信 .....	64
9.7. シリアル化されたクラスターの起動 .....	64

---

9.8. クラスターのブート	65
9.9. トラブルシューティング	65
<b>第10章 サーバーキャッシュ設定</b> .....	<b>66</b>
10.1. エビクションと有効期限	66
10.2. レプリケーションおよびフェイルオーバー	67
10.3. キャッシュの無効化	68
10.4. ランタイム時のキャッシュの消去	68
<b>第11章 RED HAT SINGLE SIGN-ON OPERATOR</b> .....	<b>70</b>
11.1. クラスターへの RED HAT SINGLE SIGN-ON OPERATOR のインストール	70
11.2. 実稼働環境での RED HAT SINGLE SIGN-ON OPERATOR の使用	74
11.3. カスタムリソースを使用した RED HAT SINGLE SIGN-ON のインストール	74
11.4. レルムカスタムリソースの作成	79
11.5. クライアントカスタムリソースの作成	82
11.6. ユーザーカスタムリソースの作成	84
11.7. 外部データベースへの接続	86
11.8. 外部の RED HAT SINGLE SIGN-ON への接続	89
11.9. データベースバックアップのスケジューリング	90
11.10. 拡張機能とテーマのインストール	91
11.11. カスタムリソースを管理するためのコマンドオプション	92
11.12. アップグレードストラテジー	92



## 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、[弊社](#) の CTO、Chris Wright の [メッセージ](#) を参照してください。



## 第1章 ガイドの概要

本ガイドでは、Red Hat Single Sign-On サーバーを初めて起動する前に完了が必要な手順を説明します。Red Hat Single Sign-On をテストするだけの場合は、独自の組み込みのローカル専用データベースを使用して、すぐに実行することができます。本番環境で実行される実際のデプロイメントでは、実行時のサーバー設定の管理方法 (スタンドアロンまたはドメインモード)、Red Hat Single Sign-On ストレージ用の共有データベースの設定方法、暗号化および HTTPS の設定方法、そして最後に Red Hat Single Sign-On をクラスターで実行する設定方法を決定する必要があります。本書では、サーバーをデプロイする前に実行する必要がある起動前の決定とセットアップのあらゆる側面について説明します。

特に注意すべき点の1つは、Red Hat Single Sign-On が JBoss EAP Application Server から派生していることです。Red Hat Single Sign-On の設定の多くの側面は、JBoss EAP 設定要素を中心に行われています。本書では、さらなる詳細の参照先として、外部のドキュメントを提示している場合が多々あります。

### 1.1. その他の推奨される外部ドキュメント

Red Hat Single Sign-On は、JBoss EAP アプリケーションサーバーと、Infinispan (キャッシュ用) や Hibernate (永続化用) などのサブプロジェクトの上に構築されています。本書では、インフラストラクチャーレベルの設定の基本のみを説明します。JBoss EAP とそのサブプロジェクトのドキュメントを熟読することを強く推奨します。以下は、ドキュメントへのリンクになります。

- [JBoss EAP 設定ガイド](#)

## 第2章 ソフトウェアのインストール

Red Hat Single Sign-On をインストールするには、ZIP ファイルをダウンロードして解凍するか、RPM を使用します。本章では、システム要件とディレクトリー構造を確認します。

### 2.1. インストールの要件

以下は Red Hat Single Sign-On サーバーをインストールするための前提条件です。

- Java 8 JRE または Java 11 JRE
- 選択した Java バージョンをサポートするオペレーティングシステム。[サポートされる設定](#) を参照してください。
- zip または gzip および tar
- 512M 以上の RAM
- 1G 以上のディスク領域
- PostgreSQL、MySQL、Oracle などの共有外部データベース。クラスターで実行する場合、Red Hat Single Sign-On には外部共有データベースが必要です。詳細は、このガイドの [本書のデータベースの設定](#) セクションを参照してください。
- マシンでのネットワークマルチキャストのサポート (クラスターで実行する必要がある場合)。Red Hat Single Sign-On は、マルチキャストなしでクラスター化できますが、これには多数の設定変更が必要です。詳細は、本ガイドの [クラスタリング](#) セクションを参照してください。
- Linux では、`/dev/random` の使用がセキュリティポリシーで義務付けられていない限り、利用可能なエントロピーの不足による Red Hat Single Sign-On のハングを防ぐために、ランダムデータのソースとして `/dev/urandom` を使用することを推奨します。これを行うには、Oracle JDK 8 および OpenJDK 8 で、システムの起動時に `java.security.egd` システムプロパティーを `file:/dev/urandom` に設定します。

### 2.2. ZIP ファイルからの RH-SSO のインストール

Red Hat Single Sign-On サーバーのダウンロード ZIP ファイルには、Red Hat Single Sign-On サーバーを実行するためのスクリプトとバイナリーが含まれています。最初に 7.6 サーバーをインストールし、次に 7.6.9 サーバーパッチをインストールします。

#### 手順

1. [Red Hat カスタマーポータル](#) に移動します。
2. Red Hat Single Sign-On 7.6 サーバーをダウンロードします。
3. unzip、tar、Expand-Archive などの適切な **unzip** ユーティリティーを使用して ZIP ファイルをデプロイメントします。
4. [Red Hat カスタマーポータル](#) に戻ります。
5. **Patches** タブをクリックします。
6. Red Hat Single Sign-On 7.6.9 サーバーパッチをダウンロードします。選択したディレクトリーにダウンロードした ZIP ファイルを配置します。

7. Red Hat Single Sign-On サーバーのルートディレクトリーに移動します。
8. JBoss EAP コマンドラインインターフェイスを起動します。

### Linux/Unix

```
┌ $ ./bin/jboss-cli.sh
```

### Windows

```
┌ > .\bin\jboss-cli.bat
```

9. パッチを適用します。

```
┌ $ patch apply <path-to-zip>/rh-sso-7.6.9-patch.zip
```

### 関連資料

パッチ適用の詳細は、[ZIP/インストーラーインストールのパッチ適用](#) を参照してください。

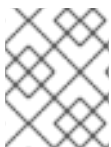
## 2.3. RPM からの RH-SSO のインストール



### 注記

Red Hat Enterprise Linux 7 および 8 では、チャンネル という用語はリポジトリーという用語に置き換えられました。これらの手順では、リポジトリーという用語のみが使用されています。

RPM から RH-SSO をインストールする前に、JBoss EAP 7.4 と RH-SSO 7.6 の両方のリポジトリーにサブスクライブする必要があります。



### 注記

EAP RPM へのアップグレードを引き続き受信することはできませんが、RH-SSO の更新の受信を停止することはできません。

### 2.3.1. JBoss EAP 7.4 リポジトリーへのサブスクライブ

#### 前提条件

1. Red Hat Subscription Manager を使用して、Red Hat Enterprise Linux システムがお使いのアカウントに登録されている必要があります。詳細は、[Red Hat Subscription Management のドキュメント](#) を参照してください。
2. すでに別の JBoss EAP リポジトリーにサブスクライブしている場合は、最初にそのリポジトリーからサブスクライブを解除する必要があります。

Red Hat Enterprise Linux 6、7 の場合: Red Hat Subscription Manager を使用して、以下のコマンドで JBoss EAP 7.4 リポジトリーにサブスクライブします。お使いの Red Hat Enterprise Linux のバージョンに応じて、<RHEL\_VERSION> を 6 または 7 に置き換えてください。

```
subscription-manager repos --enable=jb-eap-7.4-for-rhel-<RHEL_VERSION>-server-rpms --enable=rhel-<RHEL_VERSION>-server-rpms
```

Red Hat Enterprise Linux 8 の場合: Red Hat Subscription Manager を使用して、以下のコマンドで JBoss EAP 7.4 リポジトリにサブスクライブします。

```
subscription-manager repos --enable=jb-eap-7.4-for-rhel-8-x86_64-rpms --enable=rhel-8-for-x86_64-baseos-rpms --enable=rhel-8-for-x86_64-appstream-rpms
```

### 2.3.2. RH-SSO 7.6 リポジトリへのサブスクライブおよび RH-SSO 7.6 のインストール

#### 前提条件

1. Red Hat Subscription Manager を使用して、Red Hat Enterprise Linux システムがお使いのアカウントに登録されている必要があります。詳細は、[Red Hat Subscription Management のドキュメント](#) を参照してください。
2. JBoss EAP 7.4 リポジトリにサブスクライブしていることを確認してください。詳細は、[JBoss EAP 7.4 リポジトリへのサブスクライブ](#) を参照してください。

#### 手順

1. Red Hat Enterprise Linux 6、7 の場合: Red Hat Subscription Manager を使用して、以下のコマンドを使用して RH-SSO 7.6 リポジトリにサブスクライブします。お使いの Red Hat Enterprise Linux のバージョンに応じて、<RHEL\_VERSION> を 6 または 7 に置き換えてください。

```
subscription-manager repos --enable=rh-ss0-7.6-for-rhel-<RHEL-VERSION>-server-rpms
```

2. Red Hat Enterprise Linux 8 の場合: Red Hat Subscription Manager を使用して、以下のコマンドを使用して RH-SSO 7.6 リポジトリにサブスクライブします。

```
subscription-manager repos --enable=rh-ss0-7.6-for-rhel-8-x86_64-rpms
```

3. Red Hat Enterprise Linux 6 および 7 の場合は、以下のコマンドを使用して、サブスクライブしている RH-SSO 7.6 リポジトリから RH-SSO をインストールします。

```
yum groupinstall rh-ss07
```

4. Red Hat Enterprise Linux 8 の場合は、以下のコマンドを使用して、サブスクライブしている RH-SSO 7.6 リポジトリから RH-SSO をインストールします。

```
dnf groupinstall rh-ss07
```

インストールが完了します。RPM インストールのデフォルトの RH-SSO\_HOME パスは /opt/rh/rh-ss07/root/usr/share/keycloak です。

#### 関連情報

Red Hat Single Sign-On 用に 7.6.9 パッチをインストールする方法の詳細は、[RPM パッチ適用](#) を参照してください。

## 2.4. 重要なディレクトリー

以下は、サーバーディストリビューションのいくつかの重要なディレクトリーです。

### bin/

これには、サーバーを起動するスクリプト、またはサーバー上でその他の管理アクションを実行するスクリプトが含まれます。

### domain/

これには、Red Hat Single Sign-On を **ドメインモード** で実行している場合の設定ファイルと作業ディレクトリーが含まれます。

### modules/

これらはすべて、サーバーが使用する Java ライブラリーです。

### standalone/

**スタンドアロンモード** で Red Hat Single Sign-On を実行する場合の設定ファイルと作業ディレクトリーが含まれます。

### standalone/deployments/

Red Hat Single Sign-On の拡張機能を作成している場合は、ここに拡張機能を配置できます。詳細は、[サーバー開発者ガイド](#) を参照してください。

### themes/

このディレクトリーには、サーバーによって表示される UI 画面を表示するために使用されるすべての html、スタイルシート、JavaScript ファイル、およびイメージが含まれます。ここでは、既存のテーマを変更したり、独自のテーマを作成したりできます。詳細は、[サーバー開発者ガイド](#) を参照してください。

## 第3章 操作モードの使用

実稼働環境で Red Hat Single Sign-On をデプロイする前に、使用する操作モードを決定する必要があります。

- クラスター内で Red Hat Single Sign-On を実行しますか？
- サーバー設定を一元管理する方法が必要ですか？

操作モードを選択すると、データベースの設定方法、キャッシュの設定方法、およびサーバーの起動方法に影響を及ぼします。

### ヒント

Red Hat Single Sign-On は、JBoss EAP Application Server 上に構築されています。本ガイドでは、特定モードでのデプロイメントの基本についてのみ説明します。これに関する特定の情報が必要な場合は、JBoss EAP の [設定ガイド](#) を参照してください。

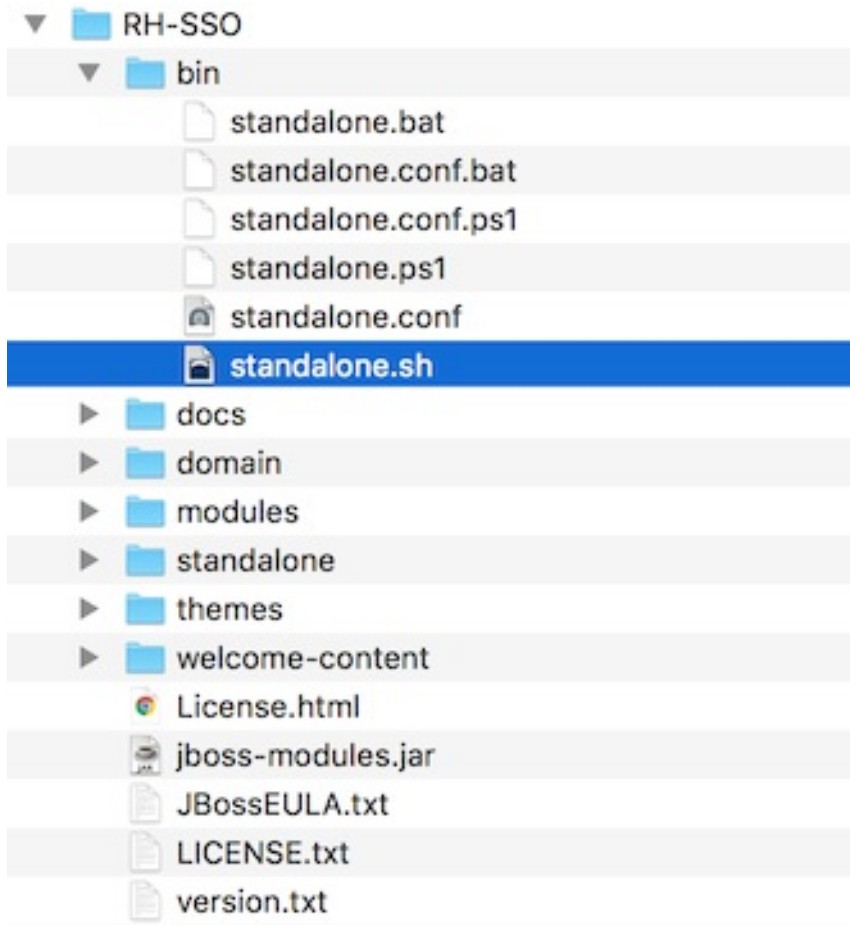
### 3.1. スタンドアロンモードの使用

スタンドアロン操作モードは、1つの Red Hat Single Sign-On サーバーインスタンスのみを実行する場合に便利です。クラスター化されたデプロイメントには使用できず、すべてのキャッシュは分散されず、ローカル専用となります。単一障害点が発生するため、本番環境でスタンドアロンモードを使用することは推奨しません。スタンドアロンモードサーバーがダウンした場合は、ユーザーはログインできなくなります。このモードは、Red Hat Single Sign-On の機能をテストしたり、試したりする場合にのみ便利です。

#### 3.1.1. スタンドアロンモードでの起動

スタンドアロンモードでサーバーを実行する場合は、オペレーティングシステムに応じて、サーバーを起動するために特定のスクリプトが必要です。このスクリプトは、サーバーディストリビューションの `bin/` ディレクトリーにあります。

#### スタンドアロン起動スクリプト



サーバーを起動するには、次のコマンドを実行します。

### Linux/Unix

```
$ ../bin/standalone.sh
```

### Windows

```
> ...\bin\standalone.bat
```



### 警告

Java SE 17 を使用してスタンドアロンモードで Red Hat Single Sign-On を実行するには、バンドルされたスクリプト **enable-elytron-se17.cli** の実行設定を変更する必要があります。

#### Linux/Unix

```
$ ./bin/jboss-cli.sh --file=docs/examples/enable-elytron-se17.cli
```

#### Windows

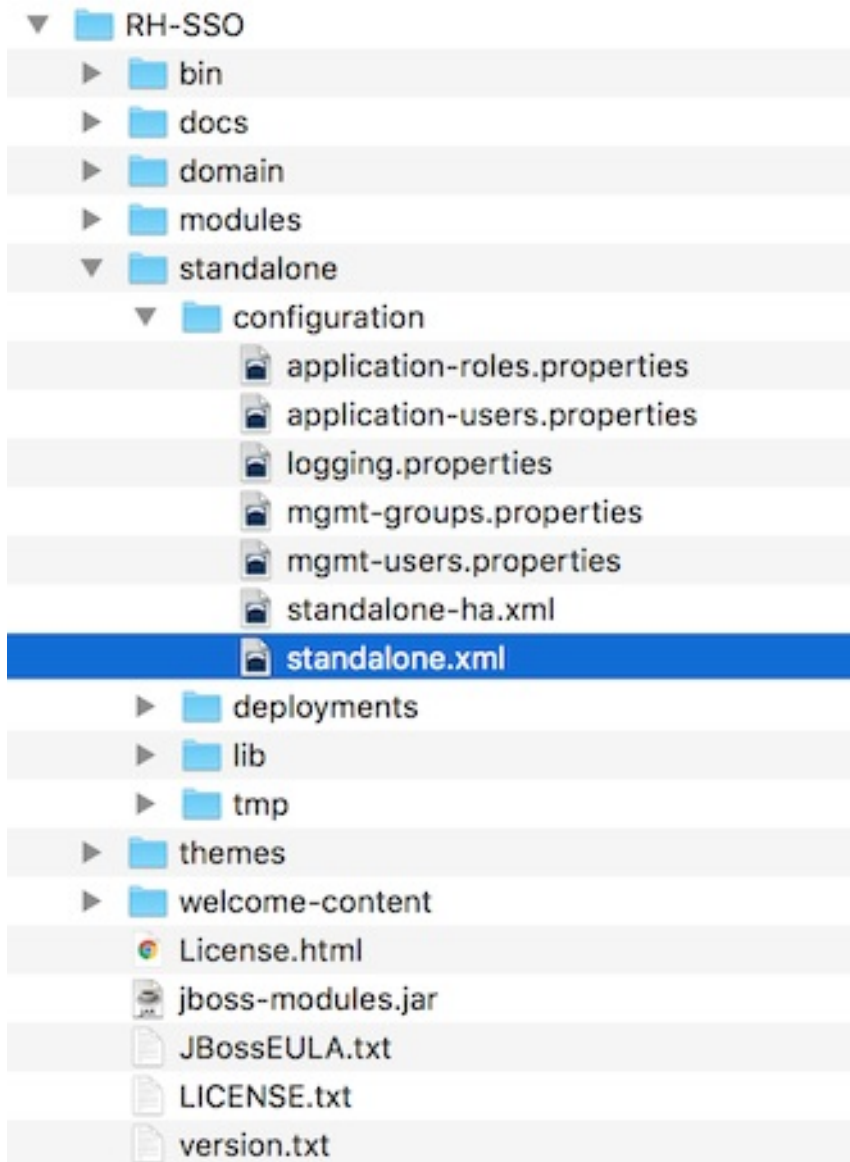
```
> .\bin\jboss-cli.bat --file=docs\examples\enable-elytron-se17.cli
```

## 3.1.2. スタンドアロン設定

このガイドの大部分では、Red Hat Single Sign-On のインフラストラクチャーレベルの側面を設定する方法について説明します。これらの側面は、Red Hat Single Sign-On が派生したアプリケーションサーバーに固有の設定ファイルで設定されます。スタンドアロン操作モードでは、このファイルは ... `/standalone/configuration/standalone.xml` にあります。このファイルは、Red Hat Single Sign-On コンポーネントに固有のインフラストラクチャー以外のレベルの設定にも使用されます。

### スタンドアロン設定ファイル





#### 警告

サーバーの実行中にこのファイルに加える変更は適用されず、サーバーによって上書きされる場合もあります。代わりに、JBoss EAP のコマンドラインスクリプトまたは Web コンソールを使用します。詳細は、JBoss EAP の [設定ガイド](#) を参照してください。

## 3.2. スタンドアロンクラスター化モードの使用

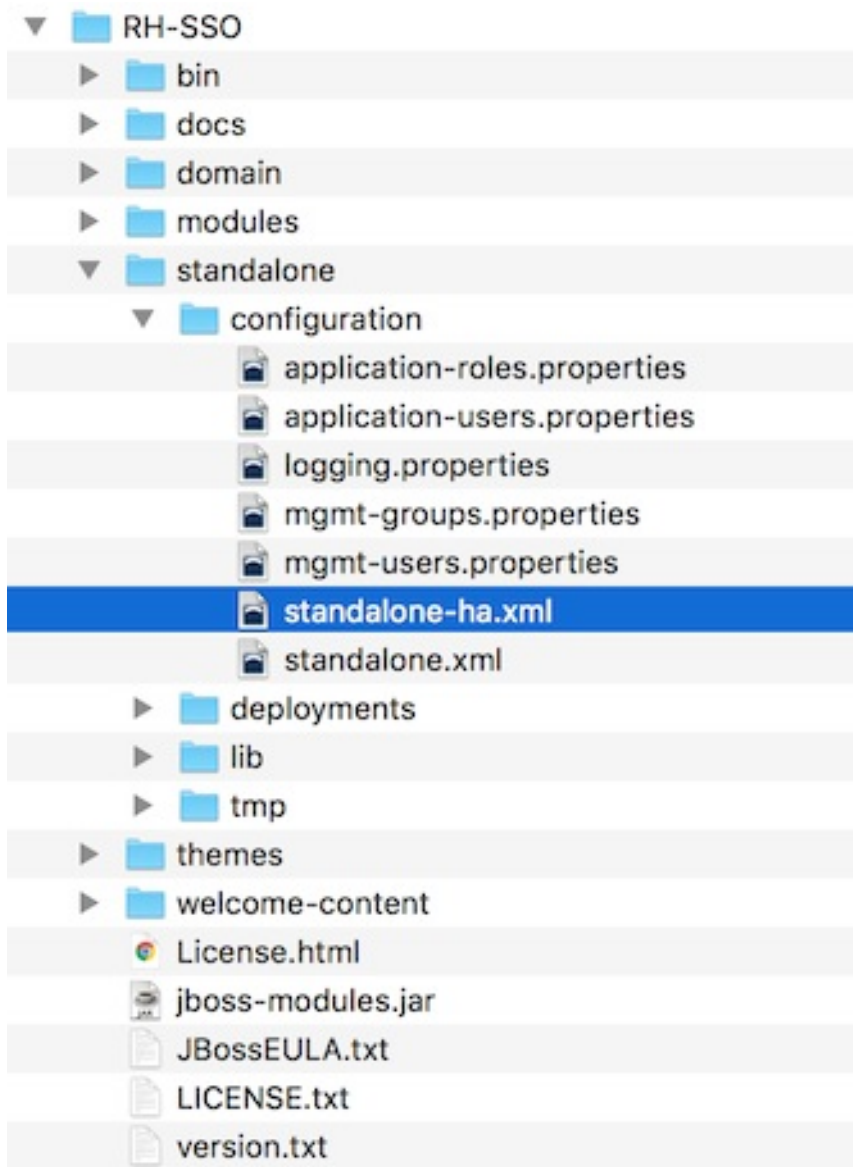
スタンドアロンクラスター化操作モードは、クラスター内で Red Hat Single Sign-On を実行する場合に適用します。このモードでは、サーバーインスタンスを実行する各マシンに Red Hat Single Sign-On ディストリビューションのコピーが必要です。このモードは、最初は簡単にデプロイできますが、非常に厄介となってくる場合があります。設定変更を行うには、各マシンの各ディストリビューションを変更します。大規模なクラスターの場合、このモードでは多大な時間がかかり、エラーが発生しやすくなります。

### 3.2.1. スタンドアロンクラスター化設定

ディストリビューションには、クラスター内で実行するための、ほとんどが事前設定されたアプリサーバー設定ファイルがあります。これには、ネットワーク、データベース、キャッシュ、検出用の特定のインフラストラクチャー設定がすべて含まれます。このファイルは ...

`/standalone/configuration/standalone-ha.xml` にあります。この設定にはいくつか欠けているものがあります。共有データベース接続を設定せずに、クラスターで Red Hat Single Sign-On を実行することはできません。また、ある種のロードバランサーをクラスターの前にデプロイする必要もあります。本ガイドの [クラスタリング](#) および [データベース](#) セクションで、これらの項目について説明します。

#### スタンドアロン HA 設定



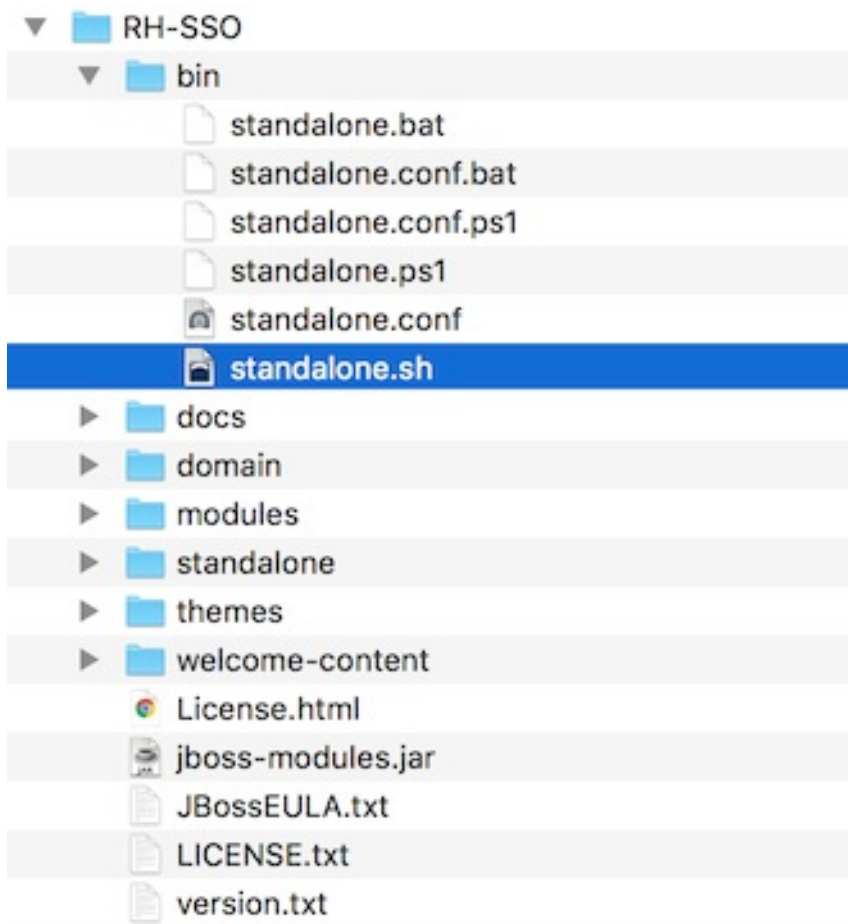
#### 警告

サーバーの実行中にこのファイルに加える変更は適用されず、サーバーによって上書きされる場合もあります。代わりに、JBoss EAP のコマンドラインスクリプトまたは Web コンソールを使用します。詳細は、JBoss EAP の [設定ガイド](#) を参照してください。

### 3.2.2. スタンドアロンクラスター化モードでの起動

スタンドアロンモードの場合と同じ起動スクリプトを使用して、Red Hat Single Sign-On を起動します。相違点は、追加のフラグを渡して HA 設定ファイルを参照することです。

#### スタンドアロンのクラスター化された起動スクリプト



サーバーを起動するには、次のコマンドを実行します。

#### Linux/Unix

```
$ ../bin/standalone.sh --server-config=standalone-ha.xml
```

#### Windows

```
> ...\bin\standalone.bat --server-config=standalone-ha.xml
```



### 警告

Java SE 17 を使用してスタンドアロンクラスターモードで Red Hat Single Sign-On を実行するには、バンドルされたスクリプト **enable-elytron-se17.cli** の実行設定を変更する必要があります。

#### Linux/Unix

```
$.bin/jboss-cli.sh --file=docs/examples/enable-elytron-se17.cli -
Dconfig=standalone-ha.xml
```

#### Windows

```
> .\bin\jboss-cli.bat --file=docs\examples\enable-elytron-se17.cli "-
Dconfig=standalone-ha.xml"
```

## 3.3. ドメインクラスターモードの使用

ドメインモードは、サーバーの設定を一元管理して公開する方法です。

クラスターを標準モードで実行すると、クラスターのサイズが大きくなるにつれてすぐに深刻化する可能性があります。設定変更が必要になるたびに、クラスター内の各ノードで実行します。ドメインモードは、設定を格納および公開するための中心的な場所を提供することで、この問題を解決します。設定は非常に複雑となる場合がありますが、それだけの価値があると言えます。この機能は、Red Hat Single Sign-On の派生元である JBoss EAP Application Server に組み込まれています。



### 注記

このガイドでは、ドメインモードの基本について説明します。クラスターでドメインモードを設定する方法は、JBoss EAP の [設定ガイド](#) を参照してください。

ドメインモードで実行するための基本的な概念のいくつかを以下に示します。

#### ドメインコントローラー

ドメインコントローラーは、クラスター内の各ノードの一般的な設定を保存、管理、および公開するプロセスです。このプロセスは、クラスター内のノードが設定を取得するための中心となるポイントです。

#### ホストコントローラー

ホストコントローラーは、特定のマシン上のサーバーインスタンスを管理します。1つ以上のサーバーインスタンスを実行するように設定します。ドメインコントローラーは、各マシンのホストコントローラーと対話してクラスターを管理することもできます。実行中のプロセスの数を減らすために、ドメインコントローラーは、実行されているマシンのホストコントローラーとしても機能します。

#### ドメインプロファイル

ドメインプロファイルは、サーバーが起動に使用できる名前付きの設定セットです。ドメインコントローラーは、異なるサーバーによって使用される複数のドメインプロファイルを定義できます。

#### サーバーグループ

サーバーグループとはサーバーの集合のことです。これらは1つとして管理および設定されます。ドメインプロファイルをサーバーグループに割り当てることができ、そのグループ内のすべてのサービスは、そのドメインプロファイルを設定として使用します。

ドメインモードでは、ドメインコントローラーがマスターノードで起動します。クラスターの設定は、ドメインコントローラーにあります。次に、クラスター内の各マシンでホストコントローラーが起動されます。各ホストコントローラーのデプロイメント設定は、そのマシンで開始する Red Hat Single Sign-On サーバーインスタンスの数を指定します。ホストコントローラーが起動すると、設定された数の Red Hat Single Sign-On サーバーインスタンスが起動します。これらのサーバーインスタンスは、ドメインコントローラーから設定をプルします。



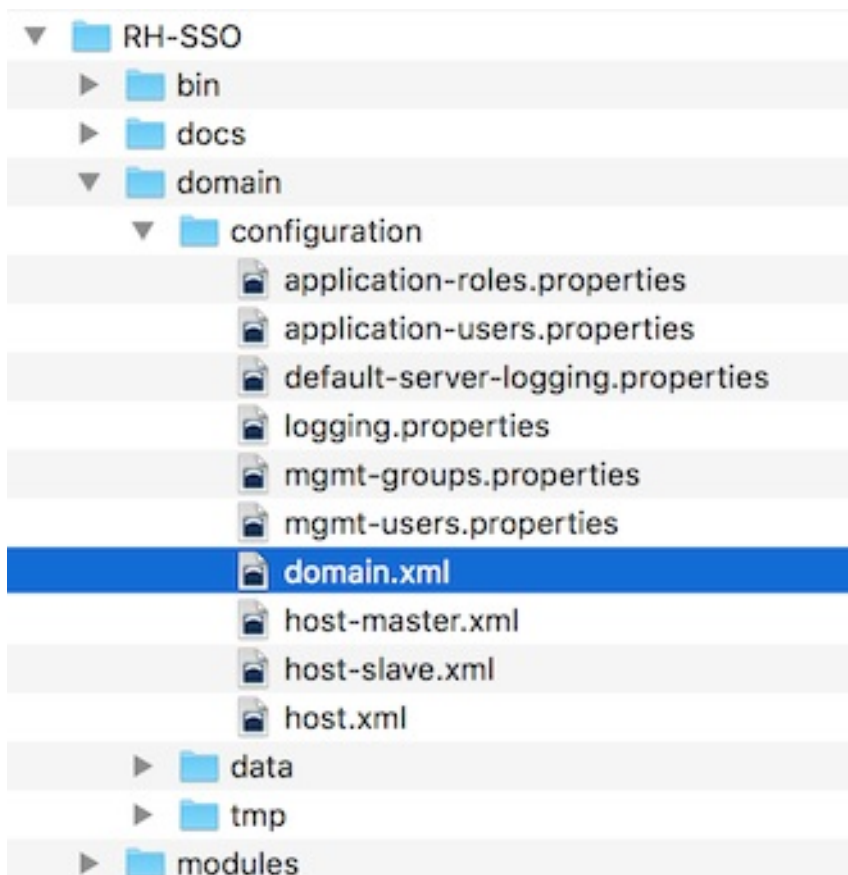
### 注記

Microsoft Azure などの一部の環境では、ドメインモードは適用されません。詳細は、JBoss EAP のドキュメントを参照してください。

## 3.3.1. ドメイン設定

本ガイドのさまざまな章では、データベース、HTTP ネットワーク接続、キャッシュ、およびその他のインフラストラクチャー関連のものなど、さまざまな側面の設定について説明します。スタンドアロンモードは `standalone.xml` ファイルを使用してこれらを設定しますが、ドメインモードは `/domain/configuration/domain.xml` 設定ファイルを使用します。ここで、Red Hat Single Sign-On サーバーのドメインプロファイルとサーバーグループが定義されます。

### domain.xml





### 警告

ドメインコントローラーの実行中にこのファイルに加える変更は適用されず、サーバーによって上書きされる場合もあります。代わりに、JBoss EAP のコマンドラインスクリプトまたは Web コンソールを使用します。詳細は、JBoss EAP の [設定ガイド](#) を参照してください。

この `domain.xml` ファイルのいくつかの側面を見てみましょう。**auth-server-standalone** および **auth-server-clustered profile** XML ブロックは、設定の決定を一括して行う場所です。ここでは、ネットワーク接続、キャッシュ、データベース接続などを設定します。

### auth-server プロファイル

```
<profiles>
  <profile name="auth-server-standalone">
    ...
  </profile>
  <profile name="auth-server-clustered">
    ...
  </profile>
```

**auth-server-standalone** プロファイルは、クラスター化されていないセットアップです。**auth-server-clustered** プロファイルはクラスター化されたセットアップです。

スクロールダウンすると、さまざまな **socket-binding-groups** が定義されていることが確認できます。

### socket-binding-groups

```
<socket-binding-groups>
  <socket-binding-group name="standard-sockets" default-interface="public">
    ...
  </socket-binding-group>
  <socket-binding-group name="ha-sockets" default-interface="public">
    ...
  </socket-binding-group>
  <!-- load-balancer-sockets should be removed in production systems and replaced with a better
software or hardware based one -->
  <socket-binding-group name="load-balancer-sockets" default-interface="public">
    ...
  </socket-binding-group>
</socket-binding-groups>
```

この設定は、各 Red Hat Single Sign-On サーバーインスタンスで開かれる各種コネクタのデフォルトのポートマッピングを定義します。`${...}` を含む値は、**-D** スイッチを使用してコマンドラインで上書きできる値です。つまり、以下ようになります。

```
$ domain.sh -Djboss.http.port=80
```

Red Hat Single Sign-On のサーバーグループの定義は、**server-groups** XML ブロックにあります。これは使用されるドメインプロファイル (**default**) と、ホストコントローラーがインスタンスの起動時に

Java 仮想マシンのデフォルトブート引数の一部を指定します。また、**socket-binding-group** をサーバーグループにバインドします。

## サーバーグループ

```
<server-groups>
  <!-- load-balancer-group should be removed in production systems and replaced with a better
  software or hardware based one -->
  <server-group name="load-balancer-group" profile="load-balancer">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="load-balancer-sockets"/>
  </server-group>
  <server-group name="auth-server-group" profile="auth-server-clustered">
    <jvm name="default">
      <heap size="64m" max-size="512m"/>
    </jvm>
    <socket-binding-group ref="ha-sockets"/>
  </server-group>
</server-groups>
```

### 3.3.2. ホストコントローラーの設定

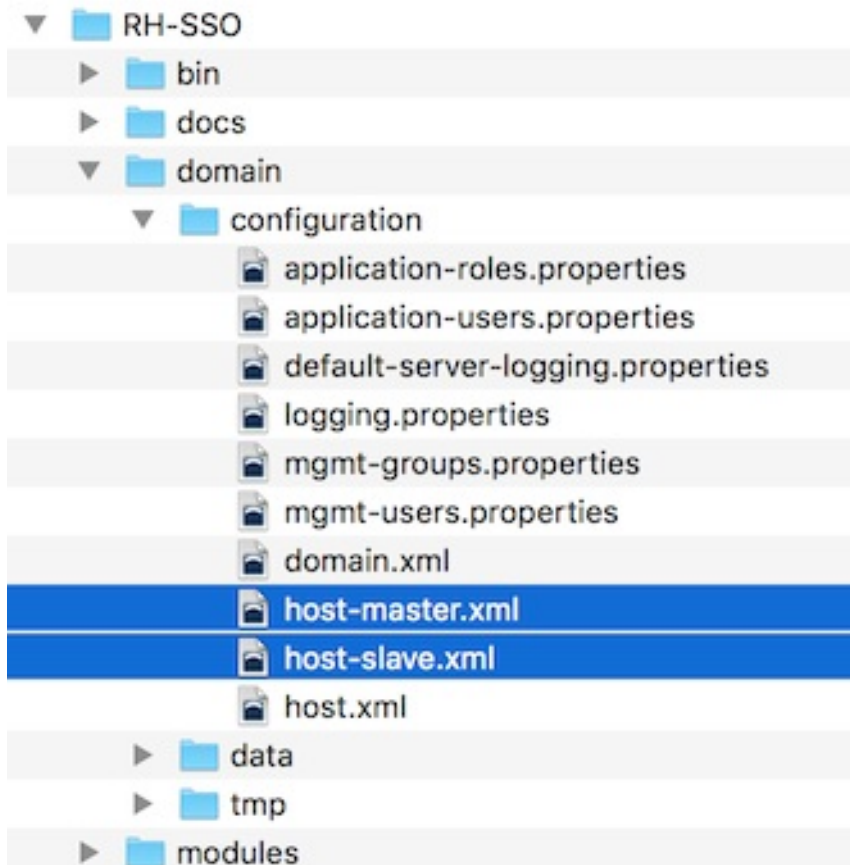
Red Hat Single Sign-On には、`.../domain/configuration/` ディレクトリーにある 2 つのホストコントローラー設定ファイル (`host-master.xml` および `host-slave.xml`) が同梱されています。`host-master.xml` は、ドメインコントローラー、ロードバランサー、および 1 つの Red Hat Single Sign-On サーバーインスタンスを起動するように設定されています。`host-slave.xml` は、ドメインコントローラーと通信し、1 つの Red Hat Single Sign-On サーバーインスタンスを起動するように設定されています。



#### 注記

ロードバランサーは必須サービスではありません。これは、開発マシンでクラスタリングを簡単にテストできるようにするために存在します。実稼働環境で使用できますが、使用するハードウェアまたはソフトウェアベースのロードバランサーが異なる場合は、これを置き換えるオプションがあります。

## ホストコントローラーの設定



ロードバランサーサーバーインスタンスを無効にするには、`host-master.xml` を編集し、"**load-balancer**" エントリーをコメントアウトまたは削除します。

```
<servers>
  <!-- remove or comment out next line -->
  <server name="load-balancer" group="loadbalancer-group"/>
  ...
</servers>
```

このファイルについて注意すべきもう1つの興味深い点は、認証サーバーインスタンスの宣言です。これには、**port-offset** が設定されています。`domain.xml` **socket-binding-group** またはサーバーグループで定義されたネットワークポートには、**port-offset** の値が追加されます。このサンプルドメイン設定では、ロードバランサーサーバーが開かれているポートが起動した認証サーバーインスタンスと競合しないように、これを行います。

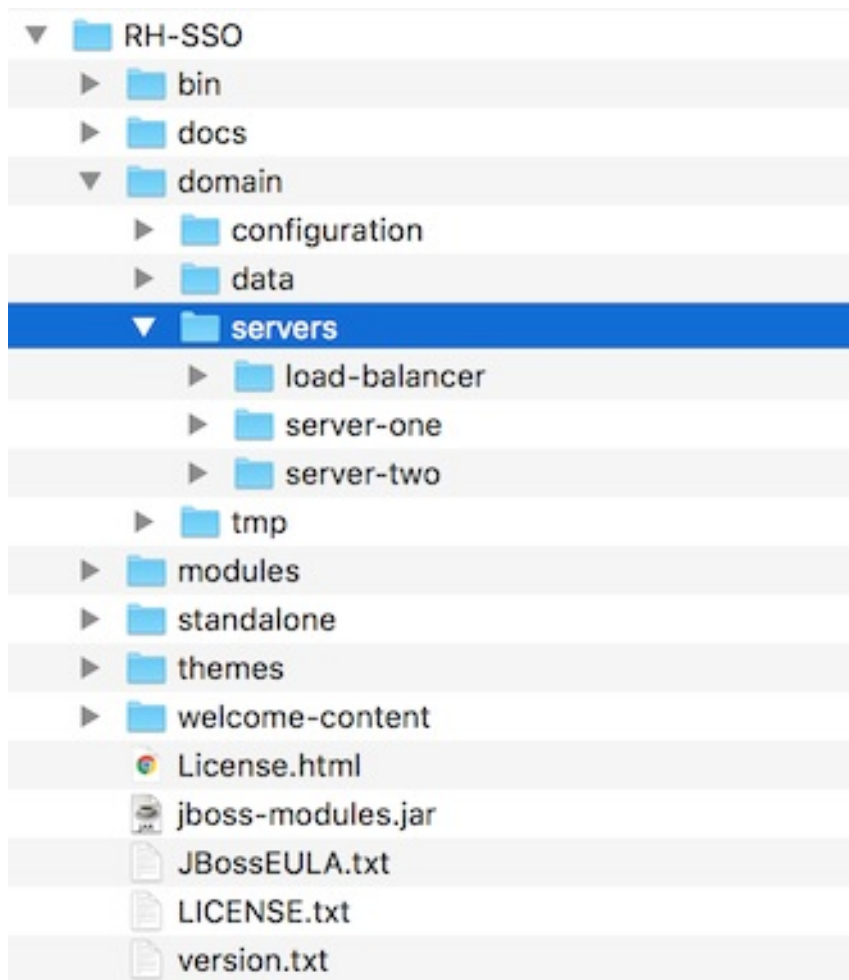
```
<servers>
  ...
  <server name="server-one" group="auth-server-group auto-start="true">
    <socket-bindings port-offset="150"/>
  </server>
</servers>
```

### 3.3.3. サーバーインスタンスの作業ディレクトリー

ホストファイルで定義された各 Red Hat Single Sign-On サーバーインスタンスは、... `/domain/servers/{SERVER NAME}` の下に作業ディレクトリーを作成します。追加の設定をそこに置くことができ、サーバーインスタンスが必要とする、または作成する一時ファイル、ログファイル、またはデータファイルもそこに配置されます。これらのサーバーディレクトリーごとの構造は、他の JBoss EAP の起動サーバーと同じようになります。



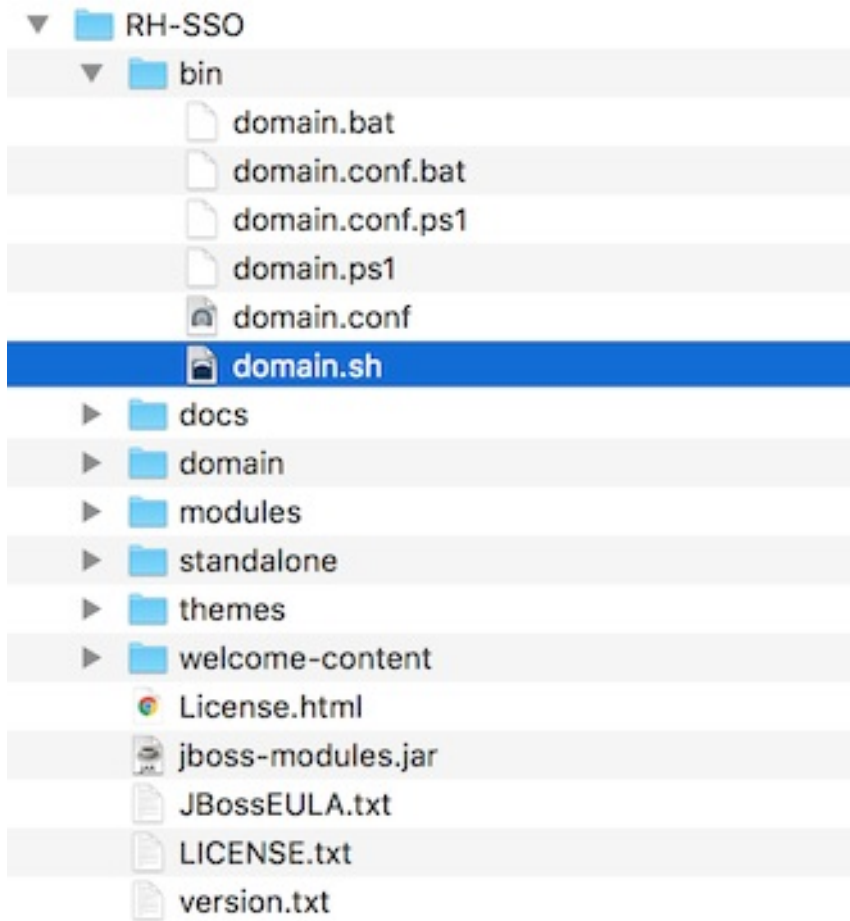
## 作業ディレクトリー



### 3.3.4. ドメインクラスターモードの起動

ドメインモードでサーバーを実行する場合は、オペレーティングシステムに応じて、サーバーを起動するために特定のスクリプトを実行する必要があります。このスクリプトは、サーバーディストリビューションの `bin/` ディレクトリーにあります。

#### ドメインブートスクリプト



サーバーを起動するには、次のコマンドを実行します。

### Linux/Unix

```
$ ../bin/domain.sh --host-config=host-master.xml
```

### Windows

```
> ...\bin\domain.bat --host-config=host-master.xml
```

ブートスクリプトを実行する場合は、**--host-config** スイッチから使用するホスト制御設定ファイルを渡す必要があります。



### 警告

Java SE 17 を使用してドメインモードで Red Hat Single Sign-On を実行するには、バンドルされたスクリプト **enable-keycloak-se17-domain.cli** の実行で設定を変更する必要があります。

#### Linux/Unix

```
$ ./bin/jboss-cli.sh --file=docs/examples/enable-keycloak-se17-domain.cli
```

#### Windows

```
> .\bin\jboss-cli.bat --file=docs\examples\enable-keycloak-se17-domain.cli
```

### 3.3.5. サンプルクラスター化ドメインを使用したテスト

サンプルの **domain.xml** 設定を使用して、ドライブのクラスターリングをテストできます。このサンプルドメインは、1台のマシンで実行され、以下を起動することを目的としています。

- ドメインコントローラー
- HTTP ロードバランサー
- 2つの Red Hat Single Sign-On サーバーインスタンス

#### 手順

1. **domain.sh** スクリプトを2回実行して、2つの別々のホストコントローラーを起動します。1つ目は、ドメインコントローラー、HTTP ロードバランサー、および1つの Red Hat Single Sign-On 認証サーバーインスタンスを起動するマスターホストコントローラーです。2つ目は、認証サーバーインスタンスのみを起動するスレーブホストコントローラーです。
2. スレーブホストコントローラーを設定して、ドメインコントローラーとセキュアに通信できるようにします。以下の手順を実行します。これらの手順を実行しないと、スレーブホストはドメインコントローラーから集中設定を取得できません。
  - a. サーバー管理ユーザーと、マスターおよびスレーブの間で共有されるシークレットを作成して、セキュアな接続を設定します。  
**.../bin/add-user.sh** スクリプトを実行します。
  - b. スクリプトが追加するユーザーのタイプについて尋ねてきたら、**Management User** を選択します。  
この選択により、**.../domain/configuration/host-slave.xml** ファイルにカットアンドペーストするシークレットが生成されます。

#### アプリケーションサーバー管理者の追加

```
$ add-user.sh
What type of user do you wish to add?
```

```

a) Management User (mgmt-users.properties)
b) Application User (application-users.properties)
(a): a
Enter the details of the new user to add.
Using realm 'ManagementRealm' as discovered from the existing property files.
Username : admin
Password recommendations are listed below. To modify these restrictions edit the add-
user.properties configuration file.
- The password should not be one of the following restricted values {root, admin,
administrator}
- The password should contain at least 8 characters, 1 alphabetic character(s), 1
digit(s), 1 non-alphanumeric symbol(s)
- The password should be different from the username
Password :
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated list,
or leave blank for none)[ ]:
About to add user 'admin' for realm 'ManagementRealm'
Is this correct yes/no? yes
Added user 'admin' to file '/.../standalone/configuration/mgmt-users.properties'
Added user 'admin' to file '/.../domain/configuration/mgmt-users.properties'
Added user 'admin' with groups to file '/.../standalone/configuration/mgmt-
groups.properties'
Added user 'admin' with groups to file '/.../domain/configuration/mgmt-groups.properties'
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting connection for
server to server EJB calls.
yes/no? yes
To represent the user add the following to the server-identities definition <secret
value="bWdtdDEyMyE=" />

```



### 注記

add-user.sh スクリプトは、ユーザーを Red Hat Single Sign-On サーバーには追加せず、基礎となる JBoss Enterprise Application Platform に追加します。このスクリプトで使用および生成される認証情報は、デモンストレーションのみを目的としています。お使いのシステムで生成されたものを使用してください。

3. 以下のように秘密の値を `.../domain/configuration/host-slave.xml` ファイルにカットアンドペーストします。

```

<management>
  <security-realms>
    <security-realm name="ManagementRealm">
      <server-identities>
        <secret value="bWdtdDEyMyE=" />
      </server-identities>
    </security-realm>
  </security-realms>
</management>

```

4. 作成したユーザーの **ユーザー名** を `.../domain/configuration/host-slave.xml` ファイルに追加します。

```

<remote security-realm="ManagementRealm" username="admin">

```

5. 起動スクリプトを2回実行して、1台の開発マシンで2つのノードクラスターをシミュレートします。

#### マスターの起動

```
$ domain.sh --host-config=host-master.xml
```

#### スレーブの起動

```
$ domain.sh --host-config=host-slave.xml
```

6. これを試すには、ブラウザを開き、<http://localhost:8080/auth> に移動します。

### 3.4. クロスサイトレプリケーションモードの使用

Red Hat Single Sign-On 7.2 でテクノロジープレビュー機能として導入されたクロスサイトレプリケーションは、最新の RH-SSO 7.6 リリースを含む Red Hat SSO 7.x リリースでサポート機能として利用できなくなりました。Red Hat は、この機能がサポートされていないため、お使いの環境でこの機能を実装したり、使用したりすることは推奨しません。また、この機能のサポート例外は考慮されず、受け入れられなくなりました。

クロスサイトレプリケーションの新しいソリューションについて議論されており、Keycloak (RHBK) の Red Hat ビルドの将来のリリースで暫定的に検討されています。これは、Red Hat SSO 8 の代わりに導入される製品です。詳細はまもなく利用可能になります。

## 第4章 サブシステム設定の管理

Red Hat Single Sign-On の低レベル設定は、ディストリビューションの **standalone.xml** ファイル、**standalone-ha.xml** ファイル、または **domain.xml** ファイルを編集して行います。このファイルの場所は、**操作モード** によって異なります。

ここには調整可能な設定が無数にありますが、本セクションは **keycloak-server** サブシステムの設定に重点を置いています。使用している設定ファイルに関係なく、**keycloak-server** サブシステムの設定は同じです。

keycloak-server サブシステムは通常、以下のようにファイルの最後の方で宣言されます。

```
<subsystem xmlns="urn:jboss:domain:keycloak-server:1.2">
  <web-context>auth</web-context>
  ...
</subsystem>
```

このサブシステムで変更された変更は、サーバーが再起動されるまで反映されません。

### 4.1. SPI プロバイダーの設定

各設定の詳細については、その設定に関連する他の場所で説明します。ただし、SPI プロバイダーの設定を宣言するために使用される形式を理解しておくことが便利です。

Red Hat Single Sign-On は、優れた柔軟性を可能にする高度なモジュールシステムです。50 を超えるサービスプロバイダーインターフェイス (SPI) があり、各 SPI の実装を差し替えることができます。SPI の実装は **プロバイダー** と呼ばれます。

SPI 宣言のすべての要素はオプションですが、完全な SPI 宣言は次のようになります。

```
<spi name="myspi">
  <default-provider>myprovider</default-provider>
  <properties>
    <property name="spi-foo" value="spi-bar"/>
  </properties>
  <provider name="myprovider" enabled="true">
    <properties>
      <property name="foo" value="bar"/>
    </properties>
  </provider>
  <provider name="mysecondprovider" enabled="true">
    <properties>
      <property name="foo" value="foo"/>
    </properties>
  </provider>
</spi>
```

2つのプロバイダーが SPI **myspi** に定義されています。**default-provider** には **myprovider** が載せられています。ただし、この設定の処理方法は SPI が決定します。一部の SPI は複数のプロバイダーを許可し、一部は許可しません。そのため、**default-provider** は SPI の選択に役立ちます。

SPI **プロパティ** を使用すると、SPI 固有の設定プロパティを指定できます。たとえば、**ユーザー**、**クライアント**、および **ロール** SPI は、以下のように **storageProviderTimeout** プロパティを介してストレージプロバイダーのタイムアウトをミリ秒単位で設定できるようにします。

■

```
<spi name="user">
  <properties>
    <property name="storageProviderTimeout" value="10000"/>
  </properties>
</spi>
```

また、各プロバイダーは独自の設定プロパティセットを定義することにも注意してください。上記の両方のプロバイダーに **foo** という名前のプロパティがあるのは単なる偶然です。

各プロパティ値のタイプは、プロバイダーによって解釈されます。ただし、例外が1つあります。 **eventsStore** SPI の **jpa** プロバイダーについて考えてみましょう。

```
<spi name="eventsStore">
  <provider name="jpa" enabled="true">
    <properties>
      <property name="exclude-events" value="['EVENT1',
        'EVENT2']"/>
    </properties>
  </provider>
</spi>
```

値が角括弧で始まり、角括弧で終わることがわかります。これは、値がリストとしてプロバイダーに渡されることを意味します。この例では、システムは2つの要素値 **EVENT1** および **EVENT2** を含むリストをプロバイダーに渡します。リストに値をさらに追加するには、各リスト要素をコンマで区切ります。ただし、各リスト要素を囲む引用符は **&quot;** でエスケープする必要があります。

カスタムプロバイダーおよびプロバイダー設定についての詳細は、[サーバー開発者ガイド](#) の手順に従います。

## 4.2. JBOSS EAP CLI の起動

手動で設定を編集する以外に、**jboss-cli** ツールでコマンドを実行して設定を変更するオプションもあります。CLIを使用すると、サーバーをローカルまたはリモートで設定できます。また、スクリプトと組み合わせる場合に特に便利です。

JBoss EAP CLI を起動するには、**jboss-cli** を実行する必要があります。

### Linux/Unix

```
$ .../bin/jboss-cli.sh
```

### Windows

```
> ...\bin\jboss-cli.bat
```

これにより、以下のようなプロンプトが表示されます。

### Prompt

```
[disconnected /]
```

実行中のサーバーでコマンドを実行する場合は、最初に **connect** コマンドを実行します。

## connect

```
[disconnected /] connect
connect
[standalone@localhost:9990 /]
```

ユーザー名やパスワードを入力していません!とっているかもしれません。スタンドアロンサーバーまたはドメインコントローラーと同じマシンで **jboss-cli** を実行し、アカウントに適切なファイルパーミッションがある場合は、管理者のユーザー名とパスワードを設定する必要がなくなります。この設定に満足していない場合、内容をより安全にする方法に関する詳細は、JBoss EAP の [設定ガイド](#) を参照してください。

## 4.3. CLI 組み込みモード

スタンドアロンサーバーと同じマシン上にあり、サーバーがアクティブでないときにコマンドを発行する場合は、サーバーを CLI に組み込み、着信要求を許可しない特別なモードで変更を加えることができます。これを実行するには、まず、変更する設定ファイルを指定して **embed-server** コマンドを実行します。

### embed-server

```
[disconnected /] embed-server --server-config=standalone.xml
[standalone@embedded /]
```

## 4.4. CLI GUI モードの使用

CLI は GUI モードでも実行できます。GUI モードは、**実行中** のサーバーの管理モデル全体をグラフィカルに表示および編集できる Swing アプリケーションを起動します。GUI モードは、CLI コマンドのフォーマット化や、使用可能なオプションの学習でサポートが必要な場合に特に便利です。GUI は、ローカルまたはリモートサーバーからサーバーログを取得することもできます。

### 手順

1. GUI モードでの CLI の起動

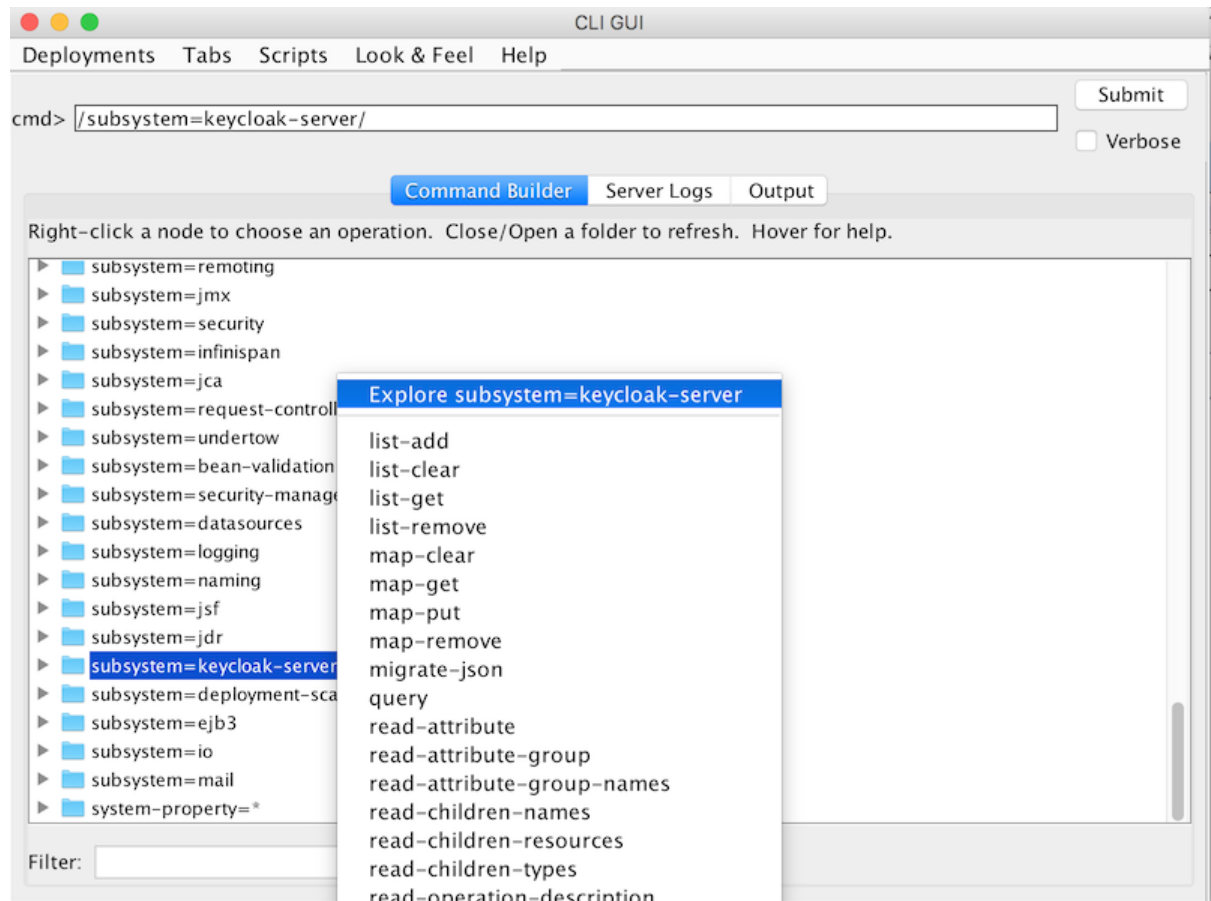
```
$ ../bin/jboss-cli.sh --gui
```

注記: リモートサーバーに接続するには、**--connect** オプションも渡します。詳細は、**--help** オプションを使用します。

2. 下にスクロールして、ノード **subsystem=keycloak-server** を見つけます。
3. ノードを右クリックして、**Explore subsystem=keycloak-server** を選択します。新しいタブには、keycloak-server サブシステムのみが表示されます。

### keycloak-server サブシステム





## 4.5. CLI スクリプト

CLI には広範なスクリプト機能があります。スクリプトは、CLI コマンドを含むテキストファイルに過ぎません。テーマとテンプレートのキャッシュをオフにする単純なスクリプトについて考えてみましょう。

### turn-off-caching.cli

```
/subsystem=keycloak-server/theme=defaults/:write-attribute(name=cacheThemes,value=false)
/subsystem=keycloak-server/theme=defaults/:write-attribute(name=cacheTemplates,value=false)
```

スクリプトを実行するには、CLI GUI の **Scripts** メニューに従うか、以下のようにコマンドラインからスクリプトを実行します。

```
$ ../bin/jboss-cli.sh --file=turn-off-caching.cli
```

## 4.6. CLI レシピ

いくつかの設定タスクと、CLI コマンドを使用してこれらを実行する方法を以下に説明します。最初を除く全ての例で、keycloak-serverサブシステムに置換する必要があることを表現するためにワイルドカードパス **\*\*** を使用します。

スタンドアロンの場合、これは単に次のことを意味します。

**\*\* = /subsystem=keycloak-server**

ドメインモードの場合、これは次のような意味になります。

```
** = /profile=auth-server-clustered/subsystem=keycloak-server
```

#### 4.6.1. サーバーの Web コンテキストの変更

```
/subsystem=keycloak-server/:write-attribute(name=web-context,value=myContext)
```

#### 4.6.2. グローバルデフォルトテーマの設定

```
**/theme=defaults/:write-attribute(name=default,value=myTheme)
```

#### 4.6.3. 新しい SPI とプロバイダーの追加

```
**/spi=mySPI/:add  
**/spi=mySPI/provider=myProvider/:add(enabled=true)
```

#### 4.6.4. プロバイダーの無効化

```
**/spi=mySPI/provider=myProvider/:write-attribute(name=enabled,value=false)
```

#### 4.6.5. SPI のデフォルトプロバイダーの変更

```
**/spi=mySPI/:write-attribute(name=default-provider,value=myProvider)
```

#### 4.6.6. SPI の単一プロパティ値の追加または変更

```
**/spi=mySPI/:map-put(name=properties, key=storageProviderTimeout, value=10000)
```

#### 4.6.7. SPI からの単一プロパティの削除

```
**/spi=mySPI/:map-remove(name=properties, key=storageProviderTimeout)
```

#### 4.6.8. dblock SPI の設定

```
**/spi=dblock/:add(default-provider=jpa)  
**/spi=dblock/provider=jpa/:add(properties={lockWaitTimeout => "900"},enabled=true)
```

#### 4.6.9. プロバイダーの単一プロパティ値の追加または変更

```
**/spi=dblock/provider=jpa/:map-put(name=properties,key=lockWaitTimeout,value=3)
```

#### 4.6.10. プロバイダーからの単一プロパティの削除

```
**/spi=dblock/provider=jpa/:map-remove(name=properties,key=lockRecheckTime)
```

#### 4.6.11. List タイプの provider プロパティへの値の設定

```
**/spi=eventsStore/provider=jpa/:map-put(name=properties,key=exclude-events,value=[EVENT1,EVENT2])
```

## 第5章 プロファイル

Red Hat Single Sign-On には、デフォルトでは有効にされていない機能があります。これには、完全にサポートされていない機能が含まれます。さらに、デフォルトで有効になっているが、無効にすることができる機能もいくつかあります。

有効および無効にできる機能は次のとおりです。

名前	説明	デフォルトでは有効	サポートレベル
account2	新規アカウント管理コンソール	はい	サポート対象
account_api	アカウント管理 REST API	はい	サポート対象
admin_fine_grained_authz	きめ細かい管理パーミッション	いいえ	プレビュー
ciba	OpenID Connect Client Initiated Backchannel Authentication (CIBA)	はい	サポート対象
client_policies	クライアント設定ポリシーの追加	はい	サポート対象
client_secret_rotation	機密クライアントのクライアントシークレットのローテーションを有効にします。	はい	プレビュー
par	OAuth 2.0 Pushed Authorization Requests (PAR)	はい	サポート対象
declarative_user_profile	宣言型スタイルを使用したユーザープロファイルの設定	いいえ	プレビュー
docker	Docker レジストリープロトコル	いいえ	サポート対象
impersonation	管理者がユーザーに成り代わる機能	はい	サポート対象
openshift_integration	OpenShift のセキュリティー保護を有効にするための拡張機能	いいえ	プレビュー
リカバリーコード	認証のリカバリーコード	いいえ	プレビュー

名前	説明	デフォルトでは有効	サポートレベル
scripts	JavaScript を使用したカスタムオーセンティケーターを作成	いいえ	プレビュー
step_up_authentication	ステップアップ認証	はい	サポート対象
token_exchange	トークン交換サービス	いいえ	プレビュー
upload_scripts	スクリプトのアップロード	いいえ	非推奨
web_authn	W3C Web Authentication (WebAuthn)	はい	サポート対象
update_email	電子メールワークフローの更新	いいえ	プレビュー

すべてのプレビュー機能を有効にするには、以下を使用してサーバーを起動します。

```
bin/standalone.sh|bat -Dkeycloak.profile=preview
```

これを永続的に設定するには、**standalone/configuration/profile.properties** ファイル (ドメインモードの **server-one** の場合は **domain/servers/server-one/configuration/profile.properties**) を作成します。以下をファイルに追加します。

```
profile=preview
```

特定の機能を有効にするには、以下を使用してサーバーを起動します。

```
bin/standalone.sh|bat -Dkeycloak.profile.feature.<feature name>=enabled
```

たとえば、Docker を有効にするには **-Dkeycloak.profile.feature.docker=enabled** を使用します。

以下を追加すると、**profile.properties** ファイルでこれを永続的に設定できます。

```
feature.docker=enabled
```

特定の機能を無効にするには、以下を使用してサーバーを起動します。

```
bin/standalone.sh|bat -Dkeycloak.profile.feature.<feature name>=disabled
```

たとえば、Impersonation を無効にするには、**-Dkeycloak.profile.feature.impersonation=disabled** を使用します。

以下を追加すると、**profile.properties** ファイルでこれを永続的に設定できます。

```
feature.impersonation=disabled
```

## 第6章 リレーショナルデータベースの設定

Red Hat Single Sign-On には、H2 と呼ばれる独自の組み込み型 Java ベースのリレーショナルデータベースが同梱されています。これは、Red Hat Single Sign-On がデータを永続化するために使用するデフォルトのデータベースであり、デフォルトで認証サーバーを実行できるようにするために存在します。

H2 データベースは、例示のみを目的としています。これはサポートされているデータベースではないため、データベースの移行についてはテストされていません。実稼働環境に対応した外部データベースに置き換えることを強く推奨します。H2 データベースは、同時並行性の高い状況ではあまり実行可能ではないため、クラスターでも使用しないでください。この章では、Red Hat Single Sign-On をより成熟したデータベースに接続する方法を示すことを目的としています。

Red Hat Single Sign-On は、2つの階層化テクノロジーを使用して、リレーショナルデータを永続化します。最下層にあるテクノロジーは JDBC です。JDBC は、RDBMS への接続に使用される Java API です。データベースベンダーが提供するデータベースのタイプごとに、異なる JDBC ドライバーがあります。この章では、これらのベンダー固有のドライバーの1つを使用するように Red Hat Single Sign-On を設定する方法について説明します。

永続性のための最上位の階層化テクノロジーは Hibernate JPA です。これは、Java オブジェクトをリレーショナルデータにマッピングするリレーショナルマッピング API のオブジェクトです。Red Hat Single Sign-On のほとんどのデプロイメントでは、Hibernate の設定要素を考慮する必要はありませんが、まれに実行する場合にこれがどのように実行されるかについて話していきます。



### 注記

データソース設定は、JBoss EAP 設定ガイドの [データソースの設定](#) の章で詳細に説明されています。

## 6.1. データベース設定チェックリスト

以下は、Red Hat Single Sign-On 用に RDBMS を設定するために実行する手順です。

1. データベースの JDBC ドライバーを見つけてダウンロードします。
2. ドライバー JAR をモジュールにパッケージ化し、このモジュールをサーバーにインストールします。
3. サーバーの設定プロファイルで JDBC ドライバーを宣言します。
4. データベースの JDBC ドライバーを使用するようにデータソース設定を変更します。
5. データソース設定を変更して、データベースへの接続パラメーターを定義します。

本章では、そのすべての例に PostgreSQL を使用します。他のデータベースも同じ手順でインストールします。

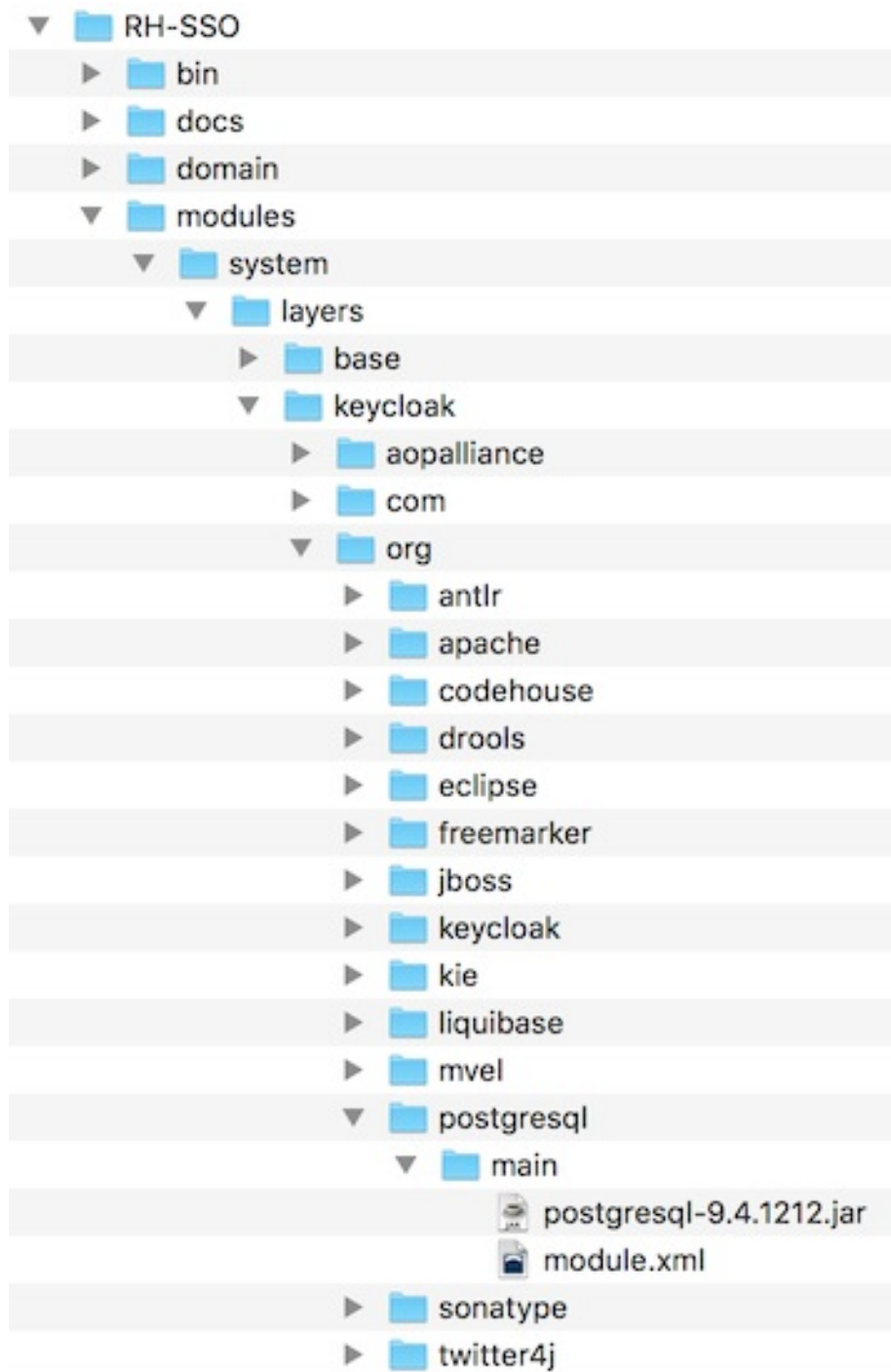
## 6.2. JDBC ドライバーのパッケージ化

RDBMS の JDBC ドライバー JAR を見つけてダウンロードします。このドライバーを使用する前に、モジュールにパッケージ化してサーバーにインストールする必要があります。モジュールは、Red Hat Single Sign-On クラスパスにロードされる JAR と、それらの JAR が他のモジュールに持つ依存関係を定義します。

## 手順

1. Red Hat Single Sign-On ディストリビューションの `.../modules/` ディレクトリー内に、モジュール定義を保持するためのディレクトリー構造を作成します。  
規則では、ディレクトリー構造の名前に JDBC ドライバーの Java パッケージ名を使用します。PostgreSQL の場合は、`org/postgresql/main` ディレクトリーを作成します。
2. データベースドライバーの JAR をこのディレクトリーにコピーし、その中に空の `module.xml` ファイルを作成します。

### モジュールディレクトリー



3. `module.xml` ファイルを開き、次の XML を作成します。

### モジュール XML

```
<?xml version="1.0" encoding="UTF-8"?>
<module xmlns="urn:jboss:module:1.3" name="org.postgresql">

  <resources>
    <resource-root path="postgresql-VERSION.jar"/>
  </resources>

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

- モジュール名は、モジュールのディレクトリー構造と一致する必要があります。そのため、`org/postgresql` は `org.postgresql` にマップします。
- `resource-root path` 属性は、ドライバーの JAR ファイル名を指定する必要があります。
- 残りは、JDBC ドライバー JAR が持つ通常の依存関係になります。

### 6.3. JDBC ドライバーの宣言とロード

JDBC をデプロイメントプロファイルに宣言すると、サーバーの起動時にロードされて使用可能になります。

#### 前提条件

JDBC ドライバーをパッケージ化している。

#### 手順

1. デプロイメントモードに基づいてこれらのファイルの1つを編集し、JDBC ドライバーを宣言します。
  - スタンドアロンモードの場合は、`.../standalone/configuration/standalone.xml` を編集します。
  - スタンドアロンのクラスタリングモードの場合は、`.../Standalone/configuration/standalone-ha.xml` を編集します。
  - ドメインモードの場合は、`.../domain/configuration/domain.xml` を編集します。ドメインモードでは、`auth-server-standalone` または `auth-server-clustered` のうち、使用している方のプロファイルを編集する必要があります。
2. プロファイルで `datasources` サブシステム内で `drivers` XML ブロックを検索します。H2 JDBC ドライバー用に宣言された事前定義されたドライバーが表示されるはずですが、ここで、外部データベース用の JDBC ドライバーを宣言します。

#### JDBC ドライバー

```
<subsystem xmlns="urn:jboss:domain:datasources:6.0">
  <datasources>
    ...
    <drivers>
      <driver name="h2" module="com.h2database.h2">
```



```

        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
      </driver>
    </drivers>
  </datasources>
</subsystem>

```

3. **drivers** XML ブロック内で、追加の JDBC ドライバーを宣言します。

- このドライバーに任意の **name** を割り当てます。
- ドライバー JAR について以前に作成した **module** パッケージを参照する **module** 属性を指定します。
- ドライバーの Java クラスを指定します。  
これは、この章で定義したモジュールの例に含まれる PostgreSQL ドライバーのインストール例です。

### JDBC ドライバーの宣言

```

<subsystem xmlns="urn:jboss:domain:datasources:6.0">
  <datasources>
    ...
    <drivers>
      <driver name="postgresql" module="org.postgresql">
        <xa-datasource-class>org.postgresql.xa.PGXADatasource</xa-datasource-
class>
      </driver>
      <driver name="h2" module="com.h2database.h2">
        <xa-datasource-class>org.h2.jdbcx.JdbcDataSource</xa-datasource-class>
      </driver>
    </drivers>
  </datasources>
</subsystem>

```

## 6.4. RED HAT SINGLE SIGN-ON データソースの変更

Red Hat Single Sign-On が新しい外部データベースに接続するために使用する既存のデータソース設定を変更します。これは、JDBC ドライバーを登録したものと同一設定ファイルと XML ブロック内で行います。新しいデータベースへの接続を設定する例を以下に示します。

### JDBC ドライバーの宣言

```

<subsystem xmlns="urn:jboss:domain:datasources:6.0">
  <datasources>
    ...
    <datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-name="KeycloakDS"
enabled="true" use-java-context="true">
      <connection-url>jdbc:postgresql://localhost/keycloak</connection-url>
      <driver>postgresql</driver>
      <pool>
        <max-pool-size>20</max-pool-size>
      </pool>
      <security>
        <user-name>William</user-name>
        <password>password</password>
      </security>
    </datasource>
  </datasources>
</subsystem>

```

```

    </security>
  </datasource>
  ...
</datasources>
</subsystem>

```

## 前提条件

- JDBC ドライバーはすでに宣言されていること。

## 手順

1. **KeycloakDS** の **datasource** 定義を検索します。  
最初に **connection-url** を変更する必要があります。ベンダーの JDBC 実装のドキュメントでは、この接続 URL 値の形式を指定する必要があります。

2. 使用する **driver** を定義します。  
これは、この章の前のセクションで宣言した JDBC ドライバーの論理名です。

トランザクションを実行するたびに、データベースへの新しい接続を開くにはコストがかかります。これを補うために、データソースの実装は開いている接続のプールを維持します。**max-pool-size** は、プールする接続の最大数を指定します。システムの負荷に応じて、この値を変更することを推奨します。

3. データベースへの接続に必要なデータベースのユーザー名とパスワードを定義します。この手順は、少なくとも PostgreSQL に必要です。この例では、クレデンシャルが明瞭なクリアテキストであることが懸念されるかもしれません。これらのクレデンシャルを難読化する方法は存在しますが、それはこのガイドの範囲を超えています。



### 注記

データソース機能の詳細は、JBoss EAP 設定ガイドの [データソースの設定](#) の章を参照してください。

## 6.5. データベースの設定

このコンポーネントの設定は、ディストリビューションの **standalone.xml** ファイル、**standalone-ha.xml** ファイル、または **domain.xml** ファイルにあります。このファイルの場所は、[操作モード](#) によって異なります。

### データベース設定

```

<subsystem xmlns="urn:jboss:domain:keycloak-server:1.2">
  ...
  <spi name="connectionsJpa">
    <provider name="default" enabled="true">
      <properties>
        <property name="dataSource" value="java:jboss/datasources/KeycloakDS"/>
        <property name="initializeEmpty" value="false"/>
        <property name="migrationStrategy" value="manual"/>
        <property name="migrationExport" value="\${jboss.home.dir}/keycloak-database-update.sql"/>
      </properties>
    </provider>
  </spi>

```

```

</spi>
...
</subsystem>

```

可能な設定オプションは以下のとおりです。

#### dataSource

dataSource の JNDI 名

#### jta

データソースが JTA 対応かどうかを指定するブール値プロパティ

#### driverDialect

データベース方言の値。ほとんどの場合、方言は Hibernate によって自動検出されるため、このプロパティを指定する必要はありません。

#### initializeEmpty

空の場合はデータベースを初期化します。false に設定する場合は、データベースを手動で初期化する必要があります。データベースを手動で初期化する場合は、migrationStrategy を **manual** に設定して、データベースを初期化する SQL コマンドを含むファイルを作成します。デフォルトは true です。

#### migrationStrategy

データベースの移行に使用するストラテジー。有効な値は、**update**、**manual**、および **validate** です。update は、データベーススキーマを自動的に移行します。manual は、データベースで手動で実行できる SQL コマンドを使用して、必要な変更をファイルにエクスポートします。validate は、データベースが最新であるかどうかを確認します。

#### migrationExport

手動のデータベース初期化/移行ファイルを書き込む場所のパス。

#### showSql

Hibernate がコンソールのすべての SQL コマンドを表示するかどうかを指定します (デフォルトは false)。これは非常に詳細です!

#### formatSql

Hibernate が SQL コマンドをフォーマットするかどうかを指定します (デフォルトは true)。

#### globalStatsInterval

実行された DB クエリーなどに関する Hibernate からのグローバル統計をログに記録します。統計は指定の間隔 (秒単位) でサーバーログに常に報告され、各レポートの後に消去されます。

#### schema

使用するデータベーススキーマを指定します。



#### 注記

これらの設定スイッチの詳細は、JBoss EAP の [開発ガイド](#) を参照してください。

## 6.6. データベースの UNICODE の考慮事項

Red Hat Single Sign-On のデータベーススキーマは、以下の特別なフィールドの Unicode 文字列のみを考慮します。

- レルム: 表示名、HTML 表示名、ローカリゼーションテキスト (キーと値)
- フェデレーションプロバイダー: 表示名

- ユーザー: ユーザー名、名、姓、属性名、および値
- グループ: 名前、属性名、値
- ロール: 名前
- オブジェクトの説明

それ以外の場合、文字は多くの場合 8 ビットであるデータベースエンコーディングに含まれるものに制限されます。ただし、データベースシステムによっては、Unicode 文字の UTF-8 エンコーディングを有効にし、すべてのテキストフィールドに完全な Unicode 文字セットを使用できます。多くの場合、8 ビットのエンコーディングの場合よりも文字列の最大長が短くなることで相殺されます。

データベースによっては、Unicode 文字を処理できるようにするには、データベースや JDBC ドライバーに特別な設定が必要になります。データベースの設定は、以下で確認してください。データベースがここにリストされている場合には、データベースと JDBC ドライバーのレベルの両方で UTF-8 エンコーディングを適切に処理できます。

技術的には、すべてのフィールドで Unicode がサポートされるかの主な基準は、データベースが **VARCHAR** フィールドおよび **CHAR** フィールドに Unicode 文字セットを設定することができるかどうかです。yes の場合、フィールドの長さとの引き換えに Unicode が使用できる可能性が高くなっています。**NVARCHAR** フィールドおよび **NCHAR** フィールドでのみ Unicode をサポートしているのであれば、Keycloak スキーマは **VARCHAR** フィールドおよび **CHAR** フィールドを広範囲に使用しているため、すべてのテキストフィールドで Unicode サポートしていることはおそらくありません。

### 6.6.1. Oracle データベース

データベースが **VARCHAR** フィールドおよび **CHAR** フィールドで Unicode サポートをサポートして作成されている場合 (**AL32UTF8** 文字セットをデータベースの文字セットとして使用するなど)、Unicode 文字は適切に処理されます。JDBC ドライバーには特別な設定は必要ありません。

データベース文字セットが Unicode でない場合、特殊フィールドに Unicode 文字を使用するには、接続プロパティ **oracle.jdbc.defaultNChar** を **true** に設定して JDBC ドライバーを設定する必要があります。厳密には必要ありませんが、**oracle.jdbc.convertNcharLiterals** 接続プロパティを **true** に設定することが適している場合があります。これらのプロパティはシステムプロパティまたは接続プロパティとして設定できます。**oracle.jdbc.defaultNChar** の設定は、パフォーマンスに悪い影響を及ぼす可能性があることに注意してください。詳細は、Oracle JDBC ドライバーの設定に関するドキュメントを参照してください。

### 6.6.2. Microsoft SQL Server データベース

Unicode 文字は、特別なフィールドに対してのみ適切に処理されます。JDBC ドライバーまたはデータベースの特別な設定は必要ありません。

### 6.6.3. MySQL データベース

データベースが、**CREATE DATABASE** コマンドの **VARCHAR** フィールドおよび **CHAR** フィールドで Unicode サポートをサポートして作成されている場合は、Unicode 文字が適切に処理されます (**utf8** 文字セットを MySQL 5.5 のデフォルトのデータベースの文字セットとして使用するなど。**utf8** 文字セットのストレージ要件が異なるため、**utf8mb4** 文字セットが機能しないことに注意してください<sup>[1]</sup>)。この場合、特定の文字数に対応するために列が作成されてバイトではなく、特別なフィールドへの長さの制限が適用されることに注意してください。データベースのデフォルト文字セットで Unicode を保存できない場合、特別なフィールドのみが Unicode 値を格納できます。

JDBC ドライバー設定で、接続プロパティ **characterEncoding=UTF-8** を JDBC 接続設定に追加する必要があります。

#### 6.6.4. PostgreSQL データベース

Unicode は、データベースの文字セットが **UTF8** の場合にサポートされます。この場合、Unicode 文字をいずれかのフィールドに使用できますが、特別なフィールド以外のフィールドの長さが削減されません。JDBC ドライバーの特別な設定は必要ありません。

PostgreSQL データベースの文字セットは、作成時に決定されます。SQL コマンドを使用して、PostgreSQL クラスターのデフォルトの文字セットを決定できます。

```
show server_encoding;
```

デフォルトの文字セットが UTF 8 ではない場合、以下のように UTF8 を文字セットとして使用してデータベースを作成できます。

```
create database keycloak with encoding 'UTF8';
```

---

[1] <https://issues.redhat.com/browse/KEYCLOAK-3873> で追跡されています。

## 第7章 パブリックホスト名の使用

Red Hat Single Sign-On では、パブリックのホスト名を使用します。たとえば、トークンの issuer フィールドおよびパスワードリセットメールで送信される URL で使用されます。

Hostname SPI は、リクエストに対するホスト名のカスタマイズ方法を提供します。初期状態のプロバイダーでは、フロントエンドリクエストの固定 URL を設定し、バックエンド要求をリクエスト URI を基にすることを許可します。組み込みプロバイダーが必要な機能を提供しない場合に、独自のプロバイダーを開発することもできます。

### 7.1. デフォルトのプロバイダー

デフォルトのホスト名プロバイダーは、設定された **frontendUrl** をフロントエンドリクエスト (ユーザーエージェントからの要求) のベース URL として使用し、バックエンドリクエスト (クライアントからの直接リクエスト) のベースとしてリクエスト URL を使用します。

frontend の要求は、Keycloak サーバーと同じ context-path を持つ必要はありません。これは、<https://auth.example.org> や <https://example.org/keycloak> などのように Keycloak を公開することができますが、内部的には、URL が <https://10.0.0.10:8080/auth> になる可能性があります。

これにより、ユーザーエージェント (ブラウザ) がパブリックドメイン名を介して Red Hat Single Sign-On にリクエストを送信できますが、内部クライアントは内部ドメイン名または IP アドレスを使用できます。

これは OpenID Connect Discovery エンドポイントに反映されます。たとえば、**authorization\_endpoint** はフロントエンド URL を使用し、**token\_endpoint** はバックエンド URL を使用します。ここでは、インスタンスのパブリッククライアントはパブリックエンドポイント経由で Keycloak と通信するため、**authorization\_endpoint** と **token\_endpoint** のベースが同じになります。

Keycloak の frontendUrl を設定するには、**-Dkeycloak.frontendUrl=https://auth.example.org** をスタートアップに渡すか、**standalone.xml** で設定できます。以下の例を参照してください。

```
<spi name="hostname">
  <default-provider>default</default-provider>
  <provider name="default" enabled="true">
    <properties>
      <property name="frontendUrl" value="https://auth.example.com"/>
      <property name="forceBackendUrlToFrontendUrl" value="false"/>
    </properties>
  </provider>
</spi>
```

jboss-cli で **frontendUrl** を更新するには、次のコマンドを使用します。

```
/subsystem=keycloak-server/spi=hostname/provider=default:write-attribute(name=properties.frontendUrl,value="https://auth.example.com")
```

すべてのリクエストがパブリックドメイン名を通過する必要がある場合は、**forceBackendUrlToFrontendUrl** を **true** に設定すると、バックエンドリクエストもフロントエンド URL を強制的に使用させることができます。

個々のレルムのデフォルトのフロントエンド URL を上書きすることもできます。これは管理コンソールで実行できます。

管理エンドポイントおよびコンソールをパブリックドメインに公開しない場合は、**adminUrl** プロパ

---

ティーを使用して管理コンソールの固定 URL を設定します。これは **frontendUrl** とは異なります。**/auth/admin** へのアクセスを外部でブロックする必要もあります。詳細は、[サーバー管理ガイド](#) を参照してください。

## 7.2. カスタムプロバイダー

カスタムホスト名プロバイダーを開発するには、**org.keycloak.urls.HostnameProviderFactory** および **org.keycloak.urls.HostnameProvider** を実装する必要があります。

カスタムプロバイダーの開発方法は、[サーバー開発者ガイド](#) のサービスプロバイダーインターフェイスのセクションを参照してください。

## 第8章 ネットワークの設定

Red Hat Single Sign-On のデフォルトインストールは、いくつかのネットワーク制限付きで実行できます。1つは、すべてのネットワークエンドポイントが **localhost** にバインドされるため、認証サーバーは実際には1つのローカルマシンでのみ使用可能です。HTTP ベースの接続では、80 や 443 などのデフォルトポートを使用しません。HTTPS/SSL は追加設定なしでは設定されず、設定しない場合、Red Hat Single Sign-On には多くのセキュリティー脆弱性があります。最後に、Red Hat Single Sign-On は、外部サーバーへのセキュアな SSL および HTTPS 接続を作成する必要があるため、エンドポイントを正しく検証できるようにトラストストアを設定する必要があります。本章では、これらすべてについて説明します。

### 8.1. バインドアドレス

デフォルトでは、Red Hat Single Sign-On は localhost ループバックアドレス **127.0.0.1** にバインドされます。これは、認証サーバーがネットワークで利用可能な場合に非常に便利なデフォルトではありません。通常、パブリックネットワークにリバースプロキシまたはロードバランサーをデプロイし、トラフィックをプライベートネットワーク上の個別の Red Hat Single Sign-On サーバーインスタンスにルーティングすることが推奨されます。ただし、いずれの場合も、**localhost** 以外のインターフェイスにバインドするようにネットワークインターフェイスを設定する必要があります。

バインドアドレスの設定は非常に簡単で、[操作モードでの操作](#) の章で説明されているとおり、コマンドラインでブートスクリプト **standalone.sh** または **domain.sh** のいずれかを使用して実行できます。

```
$ standalone.sh -b 192.168.0.5
```

**-b** スイッチは、任意のパブリックインターフェイスの IP バインドアドレスを設定します。

または、コマンドラインでバインドアドレスを設定したくない場合は、デプロイメントのプロファイル設定を編集することもできます。プロファイル設定ファイル ([操作モード](#) に応じて **standalone.xml** または **domain.xml**) を開き、**interfaces** XML ブロックを探します。

```
<interfaces>
  <interface name="management">
    <inet-address value="{jboss.bind.address.management:127.0.0.1}"/>
  </interface>
  <interface name="public">
    <inet-address value="{jboss.bind.address:127.0.0.1}"/>
  </interface>
</interfaces>
```

**public** インターフェイスは、公開されているソケットを作成するサブシステムに対応します。これらのサブシステムの1つが、Red Hat Single Sign-On の認証エンドポイントを提供する Web レイヤーです。**management** インターフェイスは、JBoss EAP の管理レイヤーによって開かれたソケットに対応します。具体的には、**jboss-cli.sh** コマンドラインインターフェイスと JBoss EAP Web コンソールを使用できるようにするソケットです。

**public** インターフェイスを確認すると、特別な文字列 **{jboss.bind.address:127.0.0.1}** があることを確認できます。この文字列は、Java システムプロパティを設定してコマンドラインで上書きできる値 **127.0.0.1** を示します。

```
$ domain.sh -Djboss.bind.address=192.168.0.5
```

**-b** は、このコマンドの簡単な表記です。そのため、バインドアドレス値をプロファイル設定で直接変更したり、起動時にコマンドラインで変更することができます。





## 注記

インターフェイスの定義を設定する際には、さらに多くのオプションを利用できます。詳細は、JBoss EAP [設定ガイド](#)の [ネットワークインターフェイス](#) を参照してください。

## 8.2. ソケットポートバインディング

各ソケットに対して開いているポートには、コマンドラインまたは設定内で上書きできる事前定義済みポートがあります。この設定を説明するために、[スタンドアロンモード](#) で実行していると仮定して、.../standalone/configuration/standalone.xml を開いてみましょう。 **socket-binding-group** を検索します。

```
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="{jboss.socket.binding.port-offset:0}">
  <socket-binding name="management-http" interface="management"
port="{jboss.management.http.port:9990}" />
  <socket-binding name="management-https" interface="management"
port="{jboss.management.https.port:9993}" />
  <socket-binding name="ajp" port="{jboss.ajp.port:8009}" />
  <socket-binding name="http" port="{jboss.http.port:8080}" />
  <socket-binding name="https" port="{jboss.https.port:8443}" />
  <socket-binding name="txn-recovery-environment" port="4712" />
  <socket-binding name="txn-status-manager" port="4713" />
  <outbound-socket-binding name="mail-smtp">
    <remote-destination host="localhost" port="25" />
  </outbound-socket-binding>
</socket-binding-group>
```

**socket-bindings** は、サーバーによって開かれるソケット接続を定義します。これらのバインディングは、使用する **インターフェイス** (バインドアドレス) と、開くポート番号を指定します。最も関心が高いと思われるものは、以下のとおりです。

### http

Red Hat Single Sign-On HTTP 接続に使用するポートを定義します。

### https

Red Hat Single Sign-On HTTPS 接続に使用されるポートを定義します。

### ajp

このソケットバインディングは、AJP プロトコルに使用されるポートを定義します。このプロトコルは、Apache HTTPD をロードバランサーとして使用する場合に **mod-cluster** とともに Apache HTTPD サーバーによって使用されます。

### management-http

JBoss EAP CLI および Web コンソールによって使用される HTTP 接続を定義します。

**ドメインモード** で実行している場合は、例の **domain.xml** ファイルに複数の **socket-binding-groups** が定義されているため、ソケット設定の設定は若干複雑になります。**server-group** 定義までスクロールダウンすると、各 **server-group** に使用される **socket-binding-group** を確認できます。

### ドメインソケットバインディング

```
<server-groups>
  <server-group name="load-balancer-group" profile="load-balancer">
    ...
    <socket-binding-group ref="load-balancer-sockets" />
  </server-group>
</server-groups>
```

```

</server-group>
<server-group name="auth-server-group" profile="auth-server-clustered">
...
  <socket-binding-group ref="ha-sockets"/>
</server-group>
</server-groups>

```



### 注記

**socket-binding-group** 定義を設定する際に、さらに多くのオプションを利用できます。詳細は、JBoss EAP設定ガイドの [ソケットバインディンググループ](#) を参照してください。

## 8.3. HTTPS/SSL



### 警告

Red Hat Single Sign-On は、SSL/HTTPS を処理するようにデフォルトで設定されていません。Red Hat Single Sign-On サーバー自体で SSL を有効にするか、Red Hat Single Sign-On サーバーの前のリバースプロキシで SSL を有効にすることを強く推奨します。

このデフォルトの動作は、各 Red Hat Single Sign-On レルムの SSL/HTTPS モードによって定義されます。詳細は、[サーバー管理ガイド](#) で詳しく説明していますが、コンテキストとこれらのモードの概要を説明します。

#### external requests

**localhost**、**127.0.0.1**、**10.x.x.x**、**192.168.x.x**、**172.16.x.x** などのプライベート IP アドレスに限り、Red Hat Single Sign-On は SSL なしで追加できます。サーバーに SSL/HTTPS が設定されていない場合や、非プライベート IP アドレスから HTTP 経由で Red Hat Single Sign-On にアクセスしようとすると、エラーが発生します。

#### none

Red Hat Single Sign-On には SSL は必要ありません。これは、処理を実行し、この開発でのみ使用する必要があります。

#### all requests

Red Hat Single Sign-On では、すべての IP アドレスに SSL が必要です。

各レルムの SSL モードは、Red Hat Single Sign-On の管理コンソールで設定できます。

## 8.4. RED HAT SINGLE SIGN-ON サーバーの HTTPS/SSL の有効化

リバースプロキシまたはロードバランサーを使用して HTTPS トラフィックを処理する場合、Red Hat Single Sign-On サーバーの HTTPS を有効にする必要があります。以下が関与します。

1. SSL/HTTP トラフィック用の秘密鍵と証明書が含まれるキーストアの取得または生成
2. このキーペアと証明書を使用するように Red Hat Single Sign-On サーバーを設定します。

### 8.4.1. 証明書および Java キーストアの作成

HTTPS 接続を許可するには、Red Hat Single Sign-On サーバーをデプロイする Web コンテナで HTTPS を有効にする前に、自己署名またはサードパーティーの署名済み証明書を取得し、Java キーストアにインポートする必要があります。

#### 8.4.1.1. 自己署名証明書

開発時には、Red Hat Single Sign-On デプロイメントをテストするためのサードパーティーの署名済み証明書を持っていない場合がありますが、その場合、Java JDK に同梱される **keytool** ユーティリティを使用して自己署名証明書を生成する必要があります。

```
$ keytool -genkey -alias localhost -keyalg RSA -keystore keycloak.jks -validity 10950
Enter keystore password: secret
Re-enter new password: secret
What is your first and last name?
[Unknown]: localhost
What is the name of your organizational unit?
[Unknown]: Keycloak
What is the name of your organization?
[Unknown]: Red Hat
What is the name of your City or Locality?
[Unknown]: Westford
What is the name of your State or Province?
[Unknown]: MA
What is the two-letter country code for this unit?
[Unknown]: US
Is CN=localhost, OU=Keycloak, O=Test, L=Westford, ST=MA, C=US correct?
[no]: yes
```

**What is your first and last name ?** の質問が表示されたら、サーバーをインストールするマシンの DNS 名を提供します。テストの目的では、**localhost** を使用する必要があります。このコマンドを実行すると、**keycloak.jks** ファイルが **keytool** コマンドの実行と同じディレクトリーに生成されます。

サードパーティーの署名済み証明書が必要で、証明書がない場合は、[cacert.org](https://cacert.org) で無料の証明書を取得できます。ただし、最初に次の手順を使用する必要があります。

#### 手順

1. 証明書要求を生成します。

```
$ keytool -certreq -alias yourdomain -keystore keycloak.jks > keycloak.careq
```

**yourdomain** は、この証明書が生成される DNS 名に置き換えます。keytool は要求を生成します。

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIC2jCCAclCAQAwZTElMAkGA1UEBhMCVVMxGzAJBgNVBAGTAK1BMREwDwYDVQQHE
whXZXN0Zm9y
ZDEQMA4GA1UEChMHUmVkieHhdDEQMA4GA1UECxMHUmVkieHhdDESMBAGA1UEAxM
JbG9jYWVxob3N0
MIIIBjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA7kck2TaavIEOGbcpi9c0rncY4Hhd
zmY
Ax2nZfq1eZEalPqI5aTxwQZzzLDK9qbeAd8Ji79HzSqRDxNYaZu7mAYhFKHgixsolE3o5Yfzb
w1
```

```

29RvyeUVe+WZxv5oo9woIVVpdSINIMEL2LaFhtX/c1dqiqYVpfnvFshZQalg2nL8juzZcBjj4as
H98glS7khql/dkZKsw9NLvyxgJvp7PaXurX29fNf3ihG+oFrL22oFyV54BWWxXCKU/GPn61EGZ
Gw
Ft2qSIGLdctpMD1aJR2bcnlhEjZKDksjQZoQ5YMXaAGkcYkG6QkgrocDE2YXDbi7Gldf9MegV
J35
2DQMpwIDAQABoDAwLgYJKoZIhvcNAQkOMSEwHzAdBgNVHQ4EFgQUQwLZJBA+fjiDdiVz
aO9vrE/i
n2swDQYJKoZIhvcNAQELBQADggEBAC5FRvMkhal3q86tHPBYWBUtTmcSjs4qUm6V6f63frh
veWHf
PzRr1xH272XUleBk0gtzWo0nNZnf0mMCtUBbHhhDcG82xolikfqbZijoQQZCiGiedVjHJfTniDQ
9bMDUOXEMQ7gHZg5q6mJfNG9MbMpQaUVEEFvfGEQQxbiFK7hRWU8S23/d80e8nExgQx
dJWJ6vd0X
MzzFK6j4Dj55bJVuM7GFmfdNC52pNOD5vYe47Aqh8oajHX9XTycVtPXI45rrWAH33ftbrS8SrZ
2S
vqIFQeuLL3BaHwpl3t7j2IMWcK1p80laAxEASib/fAwrrRHplLHBXRcq6uALUOZI4Alt8=
-----END NEW CERTIFICATE REQUEST-----

```

- この CA 要求を認証局 (CA) に送信します。  
CA は署名済み証明書を発行して送信します。
- CA のルート証明書を取得してインポートします。  
CA (root.crt) から証明書をダウンロードし、以下のようにインポートすることができます。

```
$ keytool -import -keystore keycloak.jks -file root.crt -alias root
```

- 新しい CA が生成した証明書をキーストアにインポートします。

```
$ keytool -import -alias yourdomain -keystore keycloak.jks -file your-certificate.cer
```

## 8.4.2. キーストアを使用するための Red Hat Single Sign-On の設定

適切な証明書を持つ Java キーストアを利用したので、Red Hat Single Sign-On インストールを使用するように設定する必要があります。インストールに適用される設定手順を使用します。

- [JBoss セキュリティーレガシー](#)
- [Elytron TLS v1.2](#)
- [Elytron TLS v1.3](#)

### 8.4.2.1. JBoss セキュリティーレガシー

#### 手順

- キーストアを使用し、HTTPS を有効にするには、**standalone.xml** ファイル、**standalone-ha.xml** ファイル、または **host.xml** ファイルを編集する必要があります。
- キーストアファイルをデプロイメントの **設定** ディレクトリーまたは選択した場所にあるファイルに移動し、そのファイルへの絶対パスを指定します。  
絶対パスを使用している場合は、設定から任意の **relative-to** パラメーターを削除します ([操作モード](#) を参照してください)。
- JBoss EAP の **bin** ディレクトリーに **sso\_legacy.cli** という名前のバッチファイルを作成します。

4. 次の内容をバッチファイルに追加します。

```
# Start batching commands

batch

/core-service=management/security-realm=UndertowRealm:add()
/core-service=management/security-realm=UndertowRealm/server-identity=ssl:add(keystore-
path=keycloak.jks, keystore-relative-to=jboss.server.config.dir, keystore-password=secret)
/subsystem=undertow/server=default-server/https-listener=https:write-
attribute(name=security-realm, value=UndertowRealm)

# Run the batch commands

run-batch
```

5. Red Hat Single Sign-On サーバーを起動します。
6. JBoss EAP の **bin** ディレクトリーに移動します。
7. 次のスクリプトを実行します。

```
$ sh jboss-cli.sh --connect --file=sso_legacy.cli

The batch executed successfully
process-state: reload-required
```

8. **sso\_legacy.cli** の変更を有効にするために、Red Hat Single Sign-On サーバーを再起動しま
す。

#### 8.4.2.2. Elytron TLS v1.2

##### 手順

1. JBoss EAP の **bin** ディレクトリーに **sso.cli** という名前のバッチファイルを作成します。
2. 次の内容をバッチファイルに追加します。

```
# Start batching commands

batch

# Add the keystore, key manager and ssl context configuration in the elytron subsystem

/subsystem=elytron/key-store=httpsKS:add(relative-
to=jboss.server.config.dir,path=keycloak.jks,credential-reference={clear-
text=secret},type=JKS)
/subsystem=elytron/key-manager=httpsKM:add(key-store=httpsKS,credential-reference=
{clear-text=secret})
/subsystem=elytron/server-ssl-context=httpsSSC:add(key-manager=httpsKM,protocols=
["TLSv1.2"])

# Change the undertow subsystem configuration to use the ssl context defined in the
previous step for https
```

```

/subsystem=undertow/server=default-server/https-listener=https:undefine-
attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context, value=httpsSSC)

# Run the batch commands

run-batch

```

3. Red Hat Single Sign-On サーバーを起動します。
4. JBoss EAP の **bin** ディレクトリーに移動します。
5. 次のスクリプトを実行します。

```

$ sh jboss-cli.sh --connect --file=sso.cli

The batch executed successfully
process-state: reload-required

```

6. **sso.cli** の変更を有効にするために、Red Hat Single Sign-On サーバーを再起動します。

TLS の設定の詳細については、[WildFly のドキュメント](#) を参照してください。

### 8.4.2.3. Elytron TLS 1.3

#### 手順

1. JBoss EAP の **bin** ディレクトリーに **sso.cli** という名前のバッチファイルを作成します。
2. 次の内容をバッチファイルに追加します。

```

batch

# Add the keystore, key manager and ssl context configuration in the elytron subsystem
/subsystem=elytron/key-store=httpsKS:add(relative-
to=jboss.server.config.dir,path=keycloak.jks,credential-reference={clear-
text=secret},type=JKS)
/subsystem=elytron/key-manager=httpsKM:add(key-store=httpsKS,credential-reference=
{clear-text=secret})

/subsystem=elytron/server-ssl-context=httpsSSC:add(key-manager=httpsKM,protocols=
["TLSv1.3"])
/subsystem=elytron/server-ssl-context=httpsSSC:write-attribute(name=cipher-suite-
names,value=TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AE
S_128_GCM_SHA256)

# Change the undertow subsystem configuration to use the ssl context defined in the
previous step for https

/subsystem=undertow/server=default-server/https-listener=https:undefine-
attribute(name=security-realm)
/subsystem=undertow/server=default-server/https-listener=https:write-attribute(name=ssl-
context, value=httpsSSC)

```

```
# Run the batch commands
```

```
run-batch
```

3. Red Hat Single Sign-On サーバーを起動します。
4. JBoss EAP の **bin** ディレクトリーに移動します。
5. 次のスクリプトを実行します。

```
$ sh jboss-cli.sh --connect --file=sso.cli
```

```
The batch executed successfully
process-state: reload-required
```

6. **sso.cli** の変更を有効にするために、Red Hat Single Sign-On サーバーを再起動します。

TLS の設定の詳細については、[WildFly のドキュメント](#) を参照してください。

## 8.5. 送信 HTTP 要求

Red Hat Single Sign-On サーバーは、それが保護するサービスやアプリケーションに対してブラウザーのものではない HTTP 要求を送信する必要があります。認証サーバーは、HTTP クライアント接続プールを維持し、これらの発信接続を管理します。**standalone.xml**、**standalone-ha.xml**、または **domain.xml** で設定する必要がある設定があります。このファイルの場所は、[操作モード](#) によって異なります。

### HTTP クライアント設定例

```
<spi name="connectionsHttpClient">
  <provider name="default" enabled="true">
    <properties>
      <property name="connection-pool-size" value="256"/>
    </properties>
  </provider>
</spi>
```

可能な設定オプションは以下のとおりです。

#### establish-connection-timeout-millis

ソケット接続の確立のタイムアウト。

#### socket-timeout-millis

発信リクエストがこの期間のデータを受信しない場合は、接続をタイムアウトします。

#### connection-pool-size

プールで使用できる接続数 (デフォルトは 128)。

#### max-pooled-per-route

ホストごとにプールできる接続の数 (デフォルトでは 64)。

#### connection-ttl-millis

最大接続時間 (ミリ秒単位)。デフォルトでは設定されません。

#### max-connection-idle-time-millis

接続プールで接続がアイドル状態でいられる最大期間 (デフォルトでは 900 秒)。Apache HTTP クライアントのバックグラウンドクリーナースレッドを開始します。このチェックとバックグラウンドスレッドを無効にするには、**-1** に設定します。

#### disable-cookies

デフォルトは **true** です。true に設定すると、クッキーキャッシングは無効になります。

#### client-keystore

これは、Java キーストアファイルへのパスです。このキーストアには双方向 SSL のクライアント証明書が含まれます。

#### client-keystore-password

クライアントキーストアのパスワード。これは、**client-keystore** が設定されている場合は **必須** になります。

#### client-key-password

クライアントのキーのパスワードこれは、**client-keystore** が設定されている場合は **REQUIRED** になります。

#### proxy-mappings

送信 HTTP 要求のプロキシ設定を示します。詳細は、[HTTP リクエストの送信のプロキシマッピング](#) のセクションを参照してください。

#### disable-trust-manager

発信要求に HTTPS が必要で、この設定オプションが **true** に設定されている場合は、トラストストアを指定する必要がありません。この設定は開発時のみ使用してください。これは SSL 証明書の検証を無効にするため、実稼働環境では **使用しないで** ください。これは **任意** です。デフォルト値は **false** です。

### 8.5.1. HTTP 要求の送信プロキシマッピング

Red Hat Single Sign-On によって送信される送信 HTTP 要求は、任意でプロキシマッピングのコンマ区切りリストに基づいてプロキシサーバーを使用できます。プロキシマッピングは、**hostnamePattern;proxyUri** の形式で、正規表現ベースのホスト名パターンとプロキシ URI の組み合わせを示します。以下に例を示します。

```
.*\.(google|googleapis)\.com;http://www-proxy.acme.com:8080
```

送信 HTTP リクエストのプロキシを決定するには、ターゲットのホスト名が、設定されたホスト名パターンと照合されます。最初のマッチングパターンは、使用する proxy-uri を決定します。指定のホスト名に対して設定されたパターンのいずれも一致しない場合は、プロキシは使用されません。

プロキシサーバーに認証が必要な場合は、**username:password@** 形式でプロキシユーザーの認証情報を含めます。以下は例になります。

```
.*\.(google|googleapis)\.com;http://user01:pas2w0rd@www-proxy.acme.com:8080
```

proxy-uri の特別な値 **NO\_PROXY** は、関連付けられたホスト名パターンに一致するホストにプロキシを使用すべきではないことを示すために使用できます。proxy-mappings の最後に catch-all パターンを指定して、すべての送信リクエストにデフォルトのプロキシを定義することができます。

proxy-mapping の設定の例を以下に示します。

```
# All requests to Google APIs should use http://www-proxy.acme.com:8080 as proxy
.*\.(google|googleapis)\.com;http://www-proxy.acme.com:8080
```



```
# All requests to internal systems should use no proxy
.*\acme\.com;NO_PROXY

# All other requests should use http://fallback:8080 as proxy
.*;http://fallback:8080
```

これは、以下の **jboss-cli** コマンドで設定できます。以下のように regex-pattern を適切にエスケープする必要があります。

```
echo SETUP: Configure proxy routes for HttpClient SPI

# In case there is no connectionsHttpClient definition yet
/subsystem=keycloak-server/spi=connectionsHttpClient/provider=default:add(enabled=true)

# Configure the proxy-mappings
/subsystem=keycloak-server/spi=connectionsHttpClient/provider=default:write-attribute(name=properties.proxy-mappings,value=[".*\.(google|googleapis)\.com;http://www-proxy.acme.com:8080",".*\acme\.com;NO_PROXY",".*;http://fallback:8080"])
```

**jboss-cli** コマンドを実行すると、以下のサブシステムが設定されます。&quot; で " 文字をエンコードする必要がありますことに注意してください。

```
<spi name="connectionsHttpClient">
  <provider name="default" enabled="true">
    <properties>
      <property
        name="proxy-mappings"
        value="[&quot;.*\.(google|googleapis)\.com;http://www-
proxy.acme.com:8080&quot;;&quot;.*\acme\.com;NO_PROXY&quot;;&quot;.*;http://fallback:8080&qu
ot;]">
    </properties>
  </provider>
</spi>
```

### 8.5.2. 標準の環境変数の使用

または、標準の環境変数を使用して、プロキシマッピング (**HTTP\_PROXY**、**HTTPS\_PROXY**、および **NO\_PROXY** 変数) を設定できます。

**HTTP\_PROXY** および **HTTPS\_PROXY** 変数は、すべての送信 HTTP 要求に使用されるプロキシサーバーを表します。Red Hat Single Sign-On は、この 2 つの変数とは変わりません。両方が指定されている場合には、**HTTPS\_PROXY** はプロキシサーバーが使用する実際のスキームに関係なく優先されます。

**NO\_PROXY** 変数は、プロキシを使用しないホスト名のコンマ区切りリストを定義するために使用されます。ホスト名が指定されている場合、その接頭辞 (subdomains) もプロキシの使用から除外されます。

以下の例を参照してください。

```
HTTPS_PROXY=https://www-proxy.acme.com:8080
NO_PROXY=google.com,login.facebook.com
```

この例では、すべての送信 HTTP リクエスト

は、[login.google.com](https://login.google.com)、[google.com](https://google.com)、[auth.login.facebook.com](https://auth.login.facebook.com) などのリクエストを除き、<https://www-proxy.acme.com:8080> プロキシサーバーを使用します。たとえば、[groups.facebook.com](https://groups.facebook.com) はプロキシ経由でルーティングされます。



### 注記

環境変数は小文字または大文字にすることができます。小文字が優先されます。たとえば、`HTTP_PROXY` と `http_proxy` の両方が定義されている場合、`http_proxy` が使用されます。

(上記のように) プロキシマッピングがサブシステム設定を使用して定義される場合に、Red Hat Single Sign-On では環境変数は考慮されません。`HTTP_PROXY` 環境変数が定義されていても、プロキシサーバーを使用しない場合は、このシナリオが適用されます。これを実行するには、以下のように汎用のプロキシルートを指定します。

```
<spi name="connectionsHttpClient">
  <provider name="default" enabled="true">
    <properties>
      <property name="proxy-mappings" value=".*;NO_PROXY"/>
    </properties>
  </provider>
</spi>
```

### 8.5.3. 送信 HTTPS リクエストトラストストア

Red Hat Single Sign-On がリモート HTTPS エンドポイントで呼び出される場合、信頼できるサーバーに接続するためにリモートサーバーの証明書を検証する必要があります。これは、中間者攻撃を防ぐために必要です。これらの証明書を署名したこれらのリモートサーバーまたは CA の証明書はトラストストアに配置する必要があります。このトラストストアは、Red Hat Single Sign-On サーバーによって管理されます。

トラストストアの設定は、常に Red Hat Single Sign-On トラストストア SPI によって行われます。このセクションの手順は、キーストアが JBoss Security Legacy または Elytron TLS によって設定された場合に適用されます。

トラストストアは、アイデンティティブローカー、LDAP アイデンティティプロバイダーに安全に接続する際に使用され、電子メールの送信時やクライアントアプリケーションとのバックチャネル通信に使用されます。



### 警告

デフォルトでは、トラストストアプロバイダーは設定されず、https 接続は [Java の JSSE リファレンスガイド](#) で説明されているように、標準の Java トラストストア設定にフォールバックします。信頼が確立されていない場合、これらの発信 HTTPS リクエストは失敗します。

`keytool` を使用して新しいトラストストアファイルを作成したり、信頼されるホスト証明書を既存のホスト証明書に追加したりできます。

```
$ keytool -import -alias HOSTDOMAIN -keystore truststore.jks -file host-certificate.cer
```

トラストストアは、ディストリビューションの **standalone.xml** ファイル、**standalone-ha.xml** ファイル、または **domain.xml** ファイル内で設定されます。このファイルの場所は、**操作モード** によって異なります。以下のテンプレートを使用して、トラストストア設定を追加できます。

```
<spi name="truststore">
  <provider name="file" enabled="true">
    <properties>
      <property name="file" value="path to your .jks file containing public certificates"/>
      <property name="password" value="password"/>
      <property name="hostname-verification-policy" value="WILDCARD"/>
    </properties>
  </provider>
</spi>
```

この設定の可能な設定オプションは以下のとおりです。

#### file

Java キーストアファイルへのパス。HTTPS 要求は、通信しているサーバーのホストを確認する方法が必要です。これは、トラストストアが行なうことです。キーストアには、1つ以上の信頼できるホスト証明書または認証局が含まれます。このトラストストアファイルには、セキュアなホストのパブリック証明書のみを含める必要があります。これは、上記のプロパティのいずれかが定義されている場合には **必須** です。

#### password

KeyStore のパスワード。これは、上記のプロパティのいずれかが定義されている場合には **必須** です。

#### hostname-verification-policy

デフォルトでは **WILDCARD** です。HTTPS 要求の場合、これによりサーバーの証明書のホスト名が検証されます。**ANY** は、ホスト名が検証されていないことを意味します。**WILDCARD \*.foo.com** などのサブドメイン名のワイルドカードを許可します。**STRICT CN** はホスト名に完全に一致する必要があります。

## 第9章 RED HAT SINGLE SIGN-ON をクラスターで実行するための設定

クラスターで実行するように Red Hat Single Sign-On を設定するには、以下のアクションを実行します。

- [操作モードの選択](#)
- [共有外部データベースの設定](#)
- [ロードバランサーの設定](#)
- [IP マルチキャストをサポートするプライベートネットワークの指定](#)

オペレーションモードの選択および共有データベースの設定については、本ガイドで前述しています。この章では、ロードバランサーの設定、プライベートネットワークの提供、およびクラスター内のホストの起動について説明します。



### 注記

IP マルチキャストなしで Red Hat Single Sign-On をクラスター化できますが、本トピックでは本ガイドの対象外となります。詳細は、[JBoss EAP 設定ガイドの JGroups](#) の章を参照してください。

### 9.1. 推奨されるネットワークアーキテクチャー

Red Hat Single Sign-On のデプロイに推奨されるネットワークアーキテクチャーは、Red Hat Single Sign-On サーバーにリクエストをルーティングするパブリック IP アドレスに HTTP/HTTPS ロードバランサーを設定する方法です。これにより、すべてのクラスタリング接続を分離され、サーバーを保護するための適切な手段が提供されます。



### 注記

デフォルトでは、認可されていないノードがクラスターに参加してマルチキャストメッセージをブロードキャストするのを防ぐためには何也没有什么ありません。このため、クラスターノードはプライベートネットワークに配置され、ファイアウォールは外部攻撃から保護します。

### 9.2. クラスタリングの例

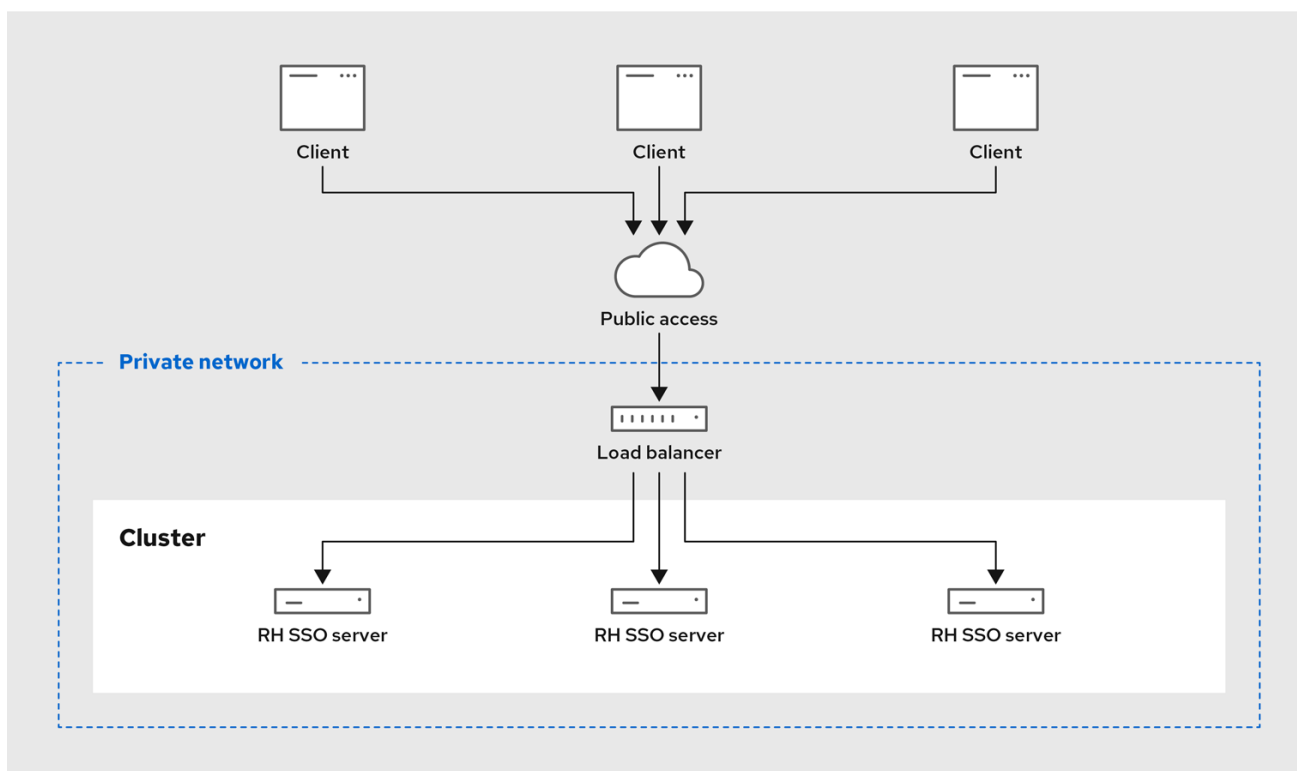
Red Hat Single Sign-On には、すぐに使用可能なドメインモードを活用するクラスタリングデモが同梱されています。詳細は、[クラスタ化ドメインの例](#) の章を参照してください。

### 9.3. ロードバランサーまたはプロキシの設定

このセクションでは、クラスター化された Red Hat Single Sign-On デプロイメントの前にリバースプロキシまたはロードバランサーを配置する前に設定する必要があるいくつかの点について説明します。また、[クラスタ化されたドメインの例](#) であった組み込みロードバランサーの設定についても説明します。

以下の図は、ロードバランサーの使用を示しています。この例では、ロードバランサーは3つのクライアントと3つの Red Hat Single Sign-On サーバーのクラスターとの間のリバースプロキシとして機能します。

## ロードバランサーダイアグラムの例



70\_RHSSO\_0320

## 9.3.1. クライアント IP アドレスの特定

Red Hat Single Sign-On のいくつかの機能は、認証サーバーに接続する HTTP クライアントのリモートアドレスがクライアントマシンの実際の IP アドレスであるという事実に依存します。たとえば、以下のとおりです。

- イベントログ - 失敗したログイン試行が誤ったソース IP アドレスでログに記録される
- SSL 必須: SSL が外部 (デフォルト) に設定されている場合、外部リクエストすべてに SSL が必要になります。
- 認証フロー: IP アドレスを使用して外部リクエストにのみ OTP を示すカスタム認証フロー
- 動的クライアント登録

これは、Red Hat Single Sign-On 認証サーバーの前にリバースプロキシまたはロードバランサーがある場合に問題が発生することがあります。通常の設定とは、プライベートネットワークにあるバックエンドの Red Hat Single Sign-On サーバーインスタンスに負荷を分散し、要求を転送するパブリックネットワークにフロントエンドプロキシがあることです。この場合、実際のクライアントの IP アドレスが Red Hat Single Sign-On サーバーインスタンスへ転送され、処理されるようにするため、いくつかの追加設定が必要です。具体的には以下を実行します。

- HTTP ヘッダーの **X-Forwarded-For** および **X-Forwarded-Proto** を適切に設定するように、リバースプロキシまたはロードバランサーを設定します。
- 元の 'Host' HTTP ヘッダーを保持するようにリバースプロキシまたはロードバランサーを設定します。

- クライアントの IP アドレスを **X-Forwarded-For** ヘッダーから読み取る認証サーバーを設定します。

HTTP ヘッダーの **X-Forwarded-For** および **X-Forwarded-Proto** を生成し、元の **Host** HTTP ヘッダーを保持するようにプロキシを設定することは本ガイドの対象外となります。**X-Forwarded-For** ヘッダーがプロキシによって設定されていることを確認します。プロキシが正しく設定されていないと、不正なクライアントはこのヘッダー自体を設定し、Red Hat Single Sign-On に、クライアントが実際とは異なる IP アドレスから接続していると思わせることができます。IP アドレスのブラックリストまたはホワイトリストを実行する場合は、このことが重要になります。

プロキシ自体以外では、Red Hat Single Sign-On で設定する必要があるいくつかの点があります。プロキシが HTTP プロトコル経由で要求を転送している場合は、ネットワークパケットからではなく、**X-Forwarded-For** ヘッダーからクライアントの IP アドレスをプルするように Red Hat Single Sign-On を設定する必要があります。これを行うには、プロファイル設定ファイル (動作モードに応じて `standalone.xml`、`standalone-ha.xml`、または `domain.xml`) を開き、XML ブロック `urn:jboss:domain:undertow:12.0` を探します。

## HTTP 設定 X-Forwarded-For

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
  <buffer-cache name="default"/>
  <server name="default-server">
    <ajp-listener name="ajp" socket-binding="ajp"/>
    <http-listener name="default" socket-binding="http" redirect-socket="https"
      proxy-address-forwarding="true"/>
    ...
  </server>
  ...
</subsystem>
```

`proxy-address-forwarding` 属性を `http-listener` 要素に追加します。この値は `true` に設定します。

プロキシがリクエストの転送に HTTP ではなく AJP プロトコルを使用している場合 (Apache HTTPD + `mod-cluster` など)、少し異なる設定を行う必要があります。`http-listener` を変更する代わりに、フィルターを追加して AJP パケットからこの情報をプルする必要があります。

## AJP 設定 X-Forwarded-For

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
  <buffer-cache name="default"/>
  <server name="default-server">
    <ajp-listener name="ajp" socket-binding="ajp"/>
    <http-listener name="default" socket-binding="http" redirect-socket="https"/>
    <host name="default-host" alias="localhost">
      ...
      <filter-ref name="proxy-peer"/>
    </host>
  </server>
  ...
  <filters>
    ...
    <filter name="proxy-peer"
      class-name="io.undertow.server.handlers.ProxyPeerAddressHandler"
      module="io.undertow.core" />
  </filters>
</subsystem>
```

### 9.3.2. リバースプロキシによる HTTPS/SSL の有効化

リバースプロキシが SSL にポート 8443 を使用しない場合は、HTTPS トラフィックのリダイレクト先となるポートを設定する必要があります。

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
...
<http-listener name="default" socket-binding="http"
  proxy-address-forwarding="true" redirect-socket="proxy-https"/>
...
</subsystem>
```

#### 手順

1. **http-listener** 要素に **redirect-socket** 属性を追加します。この値は、定義する必要のあるソケットバインディングを参照する **proxy-https** である必要があります。
2. 新しい **socket-binding** 要素を **socket-binding-group** 要素に追加します。

```
<socket-binding-group name="standard-sockets" default-interface="public"
  port-offset="${jboss.socket.binding.port-offset:0}">
...
<socket-binding name="proxy-https" port="443"/>
...
</socket-binding-group>
```

### 9.3.3. 設定の確認

リバースプロキシまたはロードバランサーの設定を確認できます

#### 手順

1. リバースプロキシを介してパス **/auth/realms/master/.well-known/openid-configuration** を開きます。  
たとえば、リバースプロキシアドレスが **https://acme.com/** の場合は、URL **https://acme.com/auth/realms/master/.well-known/openid-configuration** を開きます。これにより、JSON ドキュメントに Red Hat Single Sign-On の多数のエンドポイントがリストされます。
2. エンドポイントがリバースプロキシまたはロードバランサーのアドレス (スキーム、ドメイン、ポート) で始まることを確認します。これを実行することで、Red Hat Single Sign-On が正しいエンドポイントを使用していることを確認します。
3. Red Hat Single Sign-On が要求の正しいソース IP アドレスを確認することを確認する必要があります。  
これを確認するには、無効なユーザー名とパスワードを使用して管理コンソールにログインします。これにより、以下のような警告がサーバーログに記録されます。

```
08:14:21,287 WARN XNIO-1 task-45 [org.keycloak.events] type=LOGIN_ERROR,
realmId=master, clientId=security-admin-console, userId=8f20d7ba-4974-4811-a695-
242c8fbd1bf8, ipAddress=X.X.X.X, error=invalid_user_credentials, auth_method=openid-
```

```
connect, auth_type=code, redirect_uri=http://localhost:8080/auth/admin/master/console/?
redirect_fragment=%2Frealms%2Fmaster%2Fevents-settings, code_id=a3d48b67-a439-
4546-b992-e93311d6493e, username=admin
```

4. **ipAddress** の値が、リバースプロキシまたはロードバランサーの IP アドレスではなく、ログインを試みたマシンの IP アドレスであることを確認します。

### 9.3.4. 組み込みロードバランサーの使用

このセクションでは、[クラスター化ドメインの例](#) で説明されている組み込みロードバランサーの設定を説明します。

[クラスター化されたドメインの例](#) は、1台のマシンでのみ実行されるように設計されています。別のホストでスレーブを起動するには、

1. **domain.xml** ファイルを編集して、新規ホストのスレーブを指定します。
2. サーバーのディストリビューションをコピーします。**domain.xml** ファイル、**host.xml** ファイル、または **host-master.xml** ファイルは必要ありません。また、**standalone/** ディレクトリーも必要ありません。
3. **host-slave.xml** ファイルを編集して、コマンドラインで使用するバインドアドレスを変更するか、上書きします。

#### 手順

1. **domain.xml** を開いて、新しいホストスレーブをロードバランサー設定に登録できるようにします。
2. **load-balancer** プロファイルの **undertow** 設定に移動します。**reverse-proxy** XML ブロック内に **remote-host3** という名前の新規 **host** 定義を追加します。

#### domain.xml reverse-proxy config

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
  ...
  <handlers>
    <reverse-proxy name="lb-handler">
      <host name="host1" outbound-socket-binding="remote-host1" scheme="ajp" path="/"
instance-id="myroute1"/>
      <host name="host2" outbound-socket-binding="remote-host2" scheme="ajp" path="/"
instance-id="myroute2"/>
      <host name="remote-host3" outbound-socket-binding="remote-host3" scheme="ajp"
path="/" instance-id="myroute3"/>
    </reverse-proxy>
  </handlers>
  ...
</subsystem>
```

**output-socket-binding** は、**domain.xml** ファイルの後半に設定された **socket-binding** を示す論理名です。**instance-id** 属性は、負荷分散時にスティッキーセッションを有効にするために cookie によってこの値が使用されるため、新規ホストに一意である必要があります。

3. **load-balancer-sockets socket-binding-group** に移動し、**remote-host3** に **outbound-socket-binding** を追加します。



この新しいバインディングは新規ホストのホストおよびポートを参照する必要があります。

### domain.xml outbound-socket-binding

```
<socket-binding-group name="load-balancer-sockets" default-interface="public">
  ...
  <outbound-socket-binding name="remote-host1">
    <remote-destination host="localhost" port="8159"/>
  </outbound-socket-binding>
  <outbound-socket-binding name="remote-host2">
    <remote-destination host="localhost" port="8259"/>
  </outbound-socket-binding>
  <outbound-socket-binding name="remote-host3">
    <remote-destination host="192.168.0.5" port="8259"/>
  </outbound-socket-binding>
</socket-binding-group>
```

#### 9.3.4.1. マスターバインドアドレス

次に、マスターホストの **public** および **management** バインドアドレスを変更する必要があります。[バインドアドレス](#) の章で説明されているように **domain.xml** ファイルを編集するか、以下のようにコマンドラインでこれらのバインドアドレスを指定します。

```
$ domain.sh --host-config=host-master.xml -Djboss.bind.address=192.168.0.2 -
Djboss.bind.address.management=192.168.0.2
```

#### 9.3.4.2. ホストスレーブバインドアドレス

次に、**public**、**management**、およびドメインコントローラーのバインドアドレス (**jboss.domain.master.address**) を変更する必要があります。**host-slave.xml** ファイルを編集するか、以下のようにコマンドラインで指定します。

```
$ domain.sh --host-config=host-slave.xml
-Djboss.bind.address=192.168.0.5
-Djboss.bind.address.management=192.168.0.5
-Djboss.domain.master.address=192.168.0.2
```

ホストのスレーブの IP アドレスに関連する **jboss.bind.address** 値および **jboss.bind.address.management** の値です。**jboss.domain.master.address** の値は、マスターホストの管理アドレスであるドメインコントローラーの IP アドレスである必要があります。

#### 関連情報

- 他のソフトウェアベースのロードバランサーの使用方法は、[JBoss EAP 設定ガイドの 負荷分散](#) を参照してください。

## 9.4. スティックセッション

通常のクラスターデプロイメントは、プライベートネットワークにあるロードバランサー (逆引きプロキシ) および 2 つ以上の Red Hat Single Sign-On サーバーで構成されます。パフォーマンスの観点で、ロードバランサーが特定のブラウザーセッションに関連するすべてのリクエストを同じ Red Hat Single Sign-On バックエンドノードに転送すると良いでしょう。

その理由は、Red Hat Single Sign-On は、現在の認証セッションとユーザーセッションに関連するデータを保存するために、Infinispan の分散キャッシュを使用しているためです。Infinispan の分散キャッシュは、デフォルトで1所有者で設定されます。つまり、特定のセッションは1つのクラスターノードにのみ保存され、他のノードはそれにアクセスする場合はリモートでセッションを検索する必要があります。

たとえば、ID **123** の認証セッションが **node1** の Infinispan キャッシュに保存され、**node2** がこのセッションを検索する必要がある場合は、特定のセッションエンティティを返すために、ネットワーク経由でリクエストを **node1** に送信する必要があります。

特定のセッションエンティティが常にローカルで利用可能な場合に利点があります。これは、スティッキーセッションを使用して実行できます。パブリックフロントエンドロードバランサーと2つのバックエンド Red Hat Single Sign-On ノードを持つクラスター環境のワークフローは次のようになります。

- ユーザーが最初のリクエストを送信して、Red Hat Single Sign-On のログイン画面を表示する
- この要求はフロントエンドロードバランサーにより提供されます。このロードバランサーは、これをランダムなノード (例: node1) に転送します。厳密に言及されているので、ノードはランダムにする必要はありませんが、他の基準 (クライアント IP アドレスなど) に従って選択できます。これらはすべて、基盤のロードバランサーの実装および設定 (逆引きプロキシ) によって異なります。
- Red Hat Single Sign-On は、ランダム ID (例: 123) で認証セッションを作成し、Infinispan キャッシュに保存します。
- Infinispan の分散キャッシュは、セッション ID のハッシュに基づいてセッションのプライマリ所有者を割り当てます。詳細は、[Infinispan のドキュメント](#) を参照してください。Infinispan が、**node2** をこのセッションの所有者として割り当てたとしましょう。
- Red Hat Single Sign-On は、**<session-id>.<owner-node-id>** のような形式で cookie **AUTH\_SESSION\_ID** を作成します。この例では、**123.node2** になります。
- ブラウザーで Red Hat Single Sign-On ログイン画面と AUTH\_SESSION\_ID クッキーを持つユーザーに対して応答が返されます。

この観点から、ロードバランサーが次のリクエストをすべて **node2** に転送する利点があります。これは ID **123** の認証セッションの所有者であるノードであり、Infinispan がこのセッションをローカルで検索できるためです。認証が終了すると、認証セッションはユーザーセッションに変換されます。これは、ID **123** が同じであるため、**node2** にも保存されます。

クラスターの設定にスティッキーセッションは必須ではありませんが、上記の理由でパフォーマンスの観点から推奨されます。**AUTH\_SESSION\_ID** cookie をスティッキーするようにロードバランサーを設定する必要があります。これは、ロードバランサーによって異なります。

起動時にシステムプロパティ **jboss.node.name** を使用するには、Red Hat Single Sign-On で、ルートの名前に対応する値を使用することが推奨されます。たとえば、**-Djboss.node.name=node1** は、**node1** を使用してルートを特定します。このルートは Infinispan キャッシュによって使用され、ノードが特定のキーの所有者である場合に AUTH\_SESSION\_ID クッキーに割り当てられます。このシステムプロパティを使用した起動コマンドの例を以下に示します。

```
cd $RHSSO_NODE1
./standalone.sh -c standalone-ha.xml -Djboss.socket.binding.port-offset=100 -
Djboss.node.name=node1
```

実稼働環境では、ルート名はバックエンドホストと同じ名前を使用する必要がありますが、必須ではありません。別のルート名を使用できます。たとえば、プライベートネットワーク内で Red Hat Single Sign-On サーバーのホスト名を非表示にする場合などです。

### 9.4.1. ルートの追加を無効化

一部のロードバランサーは、バックエンド Red Hat Single Sign-On ノードに依存する代わりに、ルート情報を追加するように設定できます。ただし、上記で説明されているように、Red Hat Single Sign-On によるルートの追加が推奨されます。Red Hat Single Sign-On は特定のセッションの所有者で、そのノード (必ずしもローカルノードではない) にルーティングできるエンティティを認識するため、この方法によりパフォーマンスが向上します。

以下を Red Hat Single Sign-On サブシステム設定の

**RHSSO\_HOME/standalone/configuration/standalone-ha.xml** ファイルに追加すると、Red Hat Single Sign-On による AUTH\_SESSION\_ID cookie へのルート情報の追加を無効にできます。

```
<subsystem xmlns="urn:jboss:domain:keycloak-server:1.2">
...
  <spi name="stickySessionEncoder">
    <provider name="infinispan" enabled="true">
      <properties>
        <property name="shouldAttachRoute" value="false"/>
      </properties>
    </provider>
  </spi>
</subsystem>
```

## 9.5. マルチキャストネットワークの設定

デフォルトのクラスタリングサポートには、IP マルチキャストが必要です。マルチキャストはネットワークブロードキャストプロトコルです。このプロトコルは、起動時にクラスターを検出し、参加するために使用されます。また、Red Hat Single Sign-On が使用する分散キャッシュのレプリケーションおよび無効化のメッセージをブロードキャストするために使用されます。

Red Hat Single Sign-On のクラスタリングサブシステムは、JGroups スタックで実行されます。クラスタリングのバインドアドレスは、デフォルトの IP アドレスとして 127.0.0.1 を使用して、プライベートネットワークインターフェイスにすぐにバインドされます。

### 手順

1. [BIND アドレス](#) の章で説明した、**standalone-ha.xml** または **domain.xml** セクションを編集します。

#### プライベートネットワークの設定

```
<interfaces>
...
  <interface name="private">
    <inet-address value="${jboss.bind.address.private:127.0.0.1}"/>
  </interface>
</interfaces>
<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
```

```

...
<socket-binding name="jgroups-mping" interface="private" port="0" multicast-
address="{jboss.default.multicast.address:230.0.0.4}" multicast-port="45700"/>
<socket-binding name="jgroups-tcp" interface="private" port="7600"/>
<socket-binding name="jgroups-tcp-fd" interface="private" port="57600"/>
<socket-binding name="jgroups-udp" interface="private" port="55200" multicast-
address="{jboss.default.multicast.address:230.0.0.4}" multicast-port="45688"/>
<socket-binding name="jgroups-udp-fd" interface="private" port="54200"/>
<socket-binding name="modcluster" port="0" multicast-address="224.0.1.105" multicast-
port="23364"/>
...
</socket-binding-group>

```

2. **jboss.bind.address.private** および **jboss.default.multicast.address** と、クラスタリングスタックのサービスのポートを設定します。



### 注記

IP マルチキャストなしで Red Hat Single Sign-On をクラスター化できますが、本トピックでは本ガイドの対象外となります。詳細は、[JBoss EAP 設定ガイド](#)の [JGroups](#) を参照してください。

## 9.6. 安全なクラスター通信

クラスターノードがプライベートネットワーク上に分離されている場合は、プライベートネットワークにアクセスしたり、クラスター内の通信を表示できるようにする必要があります。さらに、クラスター通信の認証および暗号化を有効にすることもできます。プライベートネットワークがセキュアである限り、認証および暗号化を有効にする必要はありません。Red Hat Single Sign-On は、いずれの場合でもクラスターに非常に機密情報を送信しません。

クラスタリング通信の認証および暗号化を有効にする場合は、[JBoss EAP 設定ガイド](#)の [クラスターのセキュア化](#) を参照してください。

## 9.7. シリアル化されたクラスターの起動

Red Hat Single Sign-On クラスターノードは、同時に起動することができます。Red Hat Single Sign-On サーバーインスタンスが起動すると、データベースの移行、インポート、または初回の初期化が行われる場合があります。DB ロックは、クラスターノードが同時に起動する場合に、起動アクションが相互に競合しないようにするために使用されます。

デフォルトでは、このロックの最大タイムアウトは 900 秒です。ノードがタイムアウトを超えてこのロックで待機している場合、ノードは起動に失敗します。通常、デフォルト値を増減する必要はありませんが、ディストリビューションの **standalone.xml** ファイル、**standalone-ha.xml** ファイル、または **domain.xml** ファイルで設定できます。このファイルの場所は、[操作モード](#) によって異なります。

```

<spi name="dblock">
  <provider name="jpa" enabled="true">
    <properties>
      <property name="lockWaitTimeout" value="900"/>
    </properties>
  </provider>
</spi>

```

## 9.8. クラスターのブート

クラスターでの Red Hat Single Sign-On の起動は、[操作モード](#) によって異なります。

### スタンドアロンモード

```
$ bin/standalone.sh --server-config=standalone-ha.xml
```

### ドメインモード

```
$ bin/domain.sh --host-config=host-master.xml  
$ bin/domain.sh --host-config=host-slave.xml
```

追加のパラメーターまたはシステムプロパティを使用する必要がある場合があります。たとえば、[スティーキーセッション](#) のセクションで説明されているように、バインディングホストまたはシステムプロパティ **jboss.node.name** パラメーターでルートの名前を指定するためのパラメーター **-b** です。

## 9.9. トラブルシューティング

- クラスターの実行時に、両方のクラスターノードのログに以下のようなメッセージが表示されることに注意してください。

```
INFO [org.infinispan.remoting.transport.jgroups.JGroupsTransport] (Incoming-  
10,shared=udp)  
ISPN000094: Received new cluster view: [node1/keycloak|1] (2) [node1/keycloak,  
node2/keycloak]
```

上記のノードが1つのみであれば、クラスターホストが結合していない可能性があります。

通常、これらの通信にファイアウォールなしでクラスターノードをプライベートネットワークに配置することがベストプラクティスになります。ファイアウォールは、お使いのネットワークへのパブリックアクセスポイントでのみ有効にできます。何らかの理由でクラスターノードでファイアウォールを有効にしておく必要がある場合は、一部のポートを開く必要があります。デフォルト値は UDP ポート 55200 で、マルチキャストポート 45688 とマルチキャストアドレス 230.0.0.4 です。JGroups スタックの診断などの追加機能を有効にする場合は、より多くのポートを開放する必要があることに注意してください。Red Hat Single Sign-On は、クラスタリングの大半を Infinispan/JGroups に委任します。詳細は、[JBoss EAP 設定ガイドの JGroups](#) を参照してください。

- フェイルオーバーのサポート (高可用性)、エビクション、有効期限、およびキャッシュの調整を検討する場合は、[10章 サーバーキャッシュ設定](#) を参照してください。

## 第10章 サーバーキャッシュ設定

Red Hat Single Sign-On には、2種類のキャッシュがあります。DBの負荷を減らして、データをメモリーに維持することで、データベースの前にあるキャッシュのタイプが1つあり、全体的に応答時間を削減します。レルム、クライアント、ロール、およびユーザーメタデータはこの種類のキャッシュに保存されます。このキャッシュはローカルキャッシュです。より Red Hat Single Sign-On サーバーがあるクラスター内にある場合でも、ローカルキャッシュはレプリケーションを使用しません。代わりに、ローカルにコピーのみを保持し、エントリーが更新されても無効化メッセージが残りのクラスターに送信され、エントリーはエビクトされます。別個のレプリケートされたキャッシュ **work** があります。この作業では、ローカルキャッシュから削除されるエントリーについて、無効化メッセージをクラスター全体に送信します。これにより、ネットワークトラフィックが大幅に削減され、効率的な状態になり、機密メタデータをネットワーク経由で送信しないようにします。

2つ目のキャッシュは、ユーザーセッション、オフライントークンの管理、およびログイン失敗の追跡を処理し、サーバーがパスワードのシャッキングやその他の攻撃を検出できるようにします。これらのキャッシュに保持されるデータは一時的なものですが、メモリーのみはクラスター全体に複製される可能性があります。

本章では、クラスター化されたデプロイメントと非クラスターデプロイメント向けのこれらのキャッシュの設定オプションについて説明します。



### 注記

これらのキャッシュの高度な設定については、JBoss EAP 設定ガイドの [Infinispan](#) セクションを参照してください。

## 10.1. エビクションと有効期限

Red Hat Single Sign-On には複数の異なるキャッシュが設定されます。セキュアなアプリケーション、一般的なセキュリティーデータ、設定オプションに関する情報を保持するレルムキャッシュがあります。また、ユーザーメタデータを含むユーザーキャッシュもあります。両方のキャッシュを10000 エントリーに制限し、最も最近使用されたエビクションストラテジーを使用します。それぞれがクラスター設定のエビクションを制御するオブジェクトリビジョンキャッシュにも関連付けられます。このキャッシュは暗黙的に作成され、設定されたサイズの2倍になります。認可データを保持する **authorization** キャッシュにも同様のものが適用されます。**keys** キャッシュは外部キーについてのデータを保持し、専用のリビジョンキャッシュを持つ必要はありません。むしろ、**有効期限** が明示的に宣言されているため、キーは定期的に期限切れになり、外部クライアントまたはIDプロバイダーから定期的にダウンロードされます。

これらのキャッシュのエビクションポリシーおよび最大エントリーは、[操作モード](#) に応じて、`standalone.xml`、`standalone-ha.xml`、または `domain.xml` で設定できます。設定ファイルには、以下のような `infinispan` サブシステムの一部があります。

```
<subsystem xmlns="urn:jboss:domain:infinispan:12.0">
  <cache-container name="keycloak">
    <local-cache name="realms">
      <object-memory size="10000"/>
    </local-cache>
    <local-cache name="users">
      <object-memory size="10000"/>
    </local-cache>
    ...
    <local-cache name="keys">
      <object-memory size="1000"/>
      <expiration max-idle="3600000"/>
    </local-cache>
  </cache-container>
</subsystem>
```

```
</local-cache>
...
</cache-container>
```

許可されるエントリーの数制限するか、拡張するには、**object** 要素または特定のキャッシュ設定の **expiration** 要素を追加または編集します。

さらに、個別のキャッシュ

**sessions**、**clientSessions**、**offlineSessions**、**offlineClientSessions**、**loginFailures**、および **actionTokens** もあります。これらのキャッシュはクラスター環境で配布され、デフォルトでバインドされないサイズに設定されます。バインドされている場合、一部のセッションが失われる可能性があります。期限切れのセッションは、制限なしでこれらのキャッシュのサイズを増やさないように、Red Hat Single Sign-On が内部でクリアされます。多数のセッションが原因でメモリーの問題が発生する場合は、以下を試行できます。

- クラスターのサイズを増やします (より多くのノードは、セッションがノード間で均等に分散されることを意味します)
- Red Hat Single Sign-On サーバープロセスのメモリーを増やす。
- 所有者の数を減らし、1箇所にキャッシュが1度保存されるようにします。詳細は、「[レプリケーションおよびフェイルオーバー](#)」を参照してください。
- 分散キャッシュの L1 ライフスパンを無効にします。詳細は、Infinispan ドキュメントを参照してください。
- セッションのタイムアウトを減らします。これは、Red Hat Single Sign-On の管理コンソールの各レームに個別に実行できます。ただし、エンドユーザーのユーザービリティに影響する可能性があります。詳細は [タイムアウト](#) について参照してください。

他にもレプリケートされたキャッシュ (**work**) があり、大半はクラスターノード間でメッセージを送信するために使用されます。また、デフォルトではバインドされません。ただし、このキャッシュのエントリーは非常に短くなるため、このキャッシュでメモリーの問題は発生しません。

## 10.2. レプリケーションおよびフェイルオーバー

**sessions**、**authenticationSessions**、**offlineSessions**、**loginFailures** などのキャッシュがあります (詳細は「[エビクションと有効期限](#)」を参照)。これらは、クラスター化されたセットアップを使用する場合に分散キャッシュとして設定されます。エントリーは、すべての単一ノードに複製されませんが、代わりに1つ以上のノードがそのデータの所有者として選択されます。ノードが特定のキャッシュエントリーの所有者ではない場合、そのノードはクラスターに対してクエリーを実行し、これを取得します。フェイルオーバーの意味は、一部のデータを所有するすべてのノードがダウンした場合に、そのデータは永久に失われることを意味します。デフォルトでは、Red Hat Single Sign-On は、データの所有者を1つだけ指定します。したがって、1つのノードがそのデータを失う場合、これは通常、ユーザーをログアウトし、再度ログインする必要があることを意味します。

**distributed-cache** 宣言の **owners** 属性を変更すると、データを複製するノード数を変更できます。

**owners**

```
<subsystem xmlns="urn:jboss:domain:infinispan:12.0">
  <cache-container name="keycloak">
    <distributed-cache name="sessions" owners="2"/>
  ...
```

ここでは、少なくとも2つのノードが1つの特定ユーザーログインセッションを複製するように変更されました。

## ヒント

推奨される所有者の数は、実際のデプロイメントによって異なります。ノードがダウンしたときにユーザーがログアウトしても問題がない場合は、所有者が十分であるため、レプリケーションは回避できます。

## ヒント

通常、スティッキーセッションでロードバランサーを使用するように環境を設定します。特定の要求が提供される Red Hat Single Sign-On サーバーのパフォーマンスには、通常分散キャッシュからのデータの所有者であるため、データをローカルで検索できます。詳細は、「[スティッキーセッション](#)」を参照してください。

## 10.3. キャッシュの無効化

レルムまたはユーザーキャッシュを無効にできます。

### 手順

1. ディストリビューションの **standalone.xml** ファイル、**standalone-ha.xml** ファイル、または **domain.xml** ファイルを編集します。  
このファイルの場所は、[操作モード](#) によって異なります。これが設定ファイルのサンプルです。

```
<spi name="userCache">
  <provider name="default" enabled="true"/>
</spi>

<spi name="realmCache">
  <provider name="default" enabled="true"/>
</spi>
```

2. 無効にするキャッシュの **enabled** 属性を `false` に設定します。
3. この変更を有効にするには、サーバーを再起動してください。

## 10.4. ランタイム時のキャッシュの消去

レルムキャッシュ、ユーザーキャッシュ、または外部公開鍵を消去できます。

### 手順

1. 管理コンソールにログインします。
2. **Realm Settings** をクリックします。
3. **Cache** タブをクリックします。
4. レルムキャッシュ、ユーザーキャッシュ、または外部公開鍵のキャッシュを消去します。



**注記**

すべてのレルムに対してキャッシュが消去されます!

## 第11章 RED HAT SINGLE SIGN-ON OPERATOR

Red Hat Single Sign-On Operator は、OpenShift での Red Hat Single Sign-On の管理を自動化します。この Operator を使用して、管理タスクを自動化するカスタムリソース (CR) を作成します。たとえば、Red Hat Single Sign-On 管理コンソールでクライアントまたはユーザーを作成する代わりに、カスタムリソースを作成して、これらのタスクを実行することができます。カスタムリソースは、管理タスクのパラメーターを定義する YAML ファイルです。

カスタムリソースを作成して、以下のタスクを実行できます。

- [Red Hat Single Sign-On のインストール](#)
- [レルムの作成](#)
- [クライアントの作成](#)
- [ユーザーの作成](#)
- [外部データベースへの接続](#)
- [データベースのバックアップのスケジュール](#)
- [拡張機能およびテーマのインストール](#)



### 注記

レルム、クライアント、およびユーザーにカスタムリソースを作成したら、Red Hat Single Sign-On 管理コンソールを使用するか、**oc** コマンドを使用したカスタムリソースとして管理できます。ただし、Operator は変更するカスタムリソースに対して1つの方法を同期するため、両方の方法を使用することはできません。たとえば、レルムカスタムリソースを変更すると、管理コンソールで変更が表示されます。ただし、管理コンソールを使用してレルムを変更する場合、これらの変更はカスタムリソースには影響を与えません。

Operator の使用を開始するには、[Red Hat Single Sign-On Operator をクラスターにインストール](#) します。

### 11.1. クラスターへの RED HAT SINGLE SIGN-ON OPERATOR のインストール

Red Hat Single Sign-On Operator をインストールするには、以下を使用できます。

- [Operator Lifecycle Manager \(OLM\)](#)
- [コマンドラインのインストール](#)

#### 11.1.1. Operator Lifecycle Manager を使用したインストール

##### 前提条件

- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

##### 手順

OpenShift クラスターで以下の手順を実行します。

1. OpenShift Container Platform Web コンソールを開きます。
2. 左側の列で、**Operators, OperatorHub** をクリックします。
3. Red Hat Single Sign-On Operator を検索します。

### OpenShift の OperatorHub タブ

The screenshot shows the OpenShift OperatorHub interface. On the left is a dark sidebar with navigation options: Administrator, Home, Operators (selected), OperatorHub, Installed Operators, Workloads, Networking, Storage, Builds, Monitoring, Compute, User Management, and Administration. The main area is titled 'OperatorHub' and includes a search bar with 'single sign-on operator' entered. Below the search bar, a card for 'Red Hat Single Sign-On Operator' is shown, provided by Red Hat, with a 'Custom' provider type. The card description reads: 'An Operator for installing and managing Red Hat Single Sign-On'.

4. Red Hat Single Sign-On Operator アイコンをクリックします。  
Install ページが開きます。

### OpenShift への Operator Install ページ

The screenshot shows the 'Red Hat Single Sign-On Operator' install page. The title is 'Red Hat Single Sign-On Operator 7.4.0 provided by Red Hat'. There is a blue 'Install' button. Below the button, the 'Operator Version' is 7.4.0. The 'Capability Level' is set to 'Basic Install', 'Seamless Upgrades', and 'Full Lifecycle'. The 'Provider Type' is 'Custom'. The page describes the operator as a Kubernetes Operator based on the Operator SDK for installing and managing Red Hat Single Sign-On. It also lists supported features: 'Install Keycloak to a namespace', 'Import Keycloak Realms', 'Import Keycloak Clients', 'Import Keycloak Users', and 'Create scheduled backups of the database'.

5. **Install** をクリックします。
6. namespace を選択し、Subscribe をクリックします。  
**stable** チャネルにいることを確認してください。

### OpenShift での namespace 選択

#### Installation Mode \*

- All namespaces on the cluster (default)  
This mode is not supported by this Operator
- A specific namespace on the cluster  
Operator will be available in a single namespace only.

#### Installed Namespace \*

#### Update Channel \*

- stable

#### Approval Strategy \*

- Automatic
- Manual

Operator はインストールを開始します。

#### 関連資料

- Operator のインストールが完了したら、最初のカスタムリソースを作成することができます。[カスタムリソースを使用した Red Hat Single Sign-On インストール](#) を参照してください。
- OpenShift Operator の詳細は、[OpenShift Operators guide](#) を参照してください。

### 11.1.2. コマンドラインからのインストール

コマンドラインから Red Hat Single Sign-On Operator をインストールできます。

## 別添実行

- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

## 手順

1. プロジェクトを作成します。

```
$ oc new-project <namespace>
```

2. **rhssso-operator-olm.yaml** という名前のファイルを作成し、YAML グループとサブスクリプション Operator を定義します。

**targetNamespaces** を RH-SSO の **namespace** に更新します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: rhssso-operator-group
spec:
  targetNamespaces:
  - <namespace> # change this to the namespace you will use for RH-SSO
---
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: rhssso-operator
spec:
  channel: stable
  installPlanApproval: Manual
  name: rhssso-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  # Here you can specify a specific Operator version, otherwise it will use the latest
  # startingCSV: rhssso-operator.7.6.0-opr-001
```

3. Operator グループおよびサブスクリプションをインストールします。

```
$ oc apply -f rhssso-operator-olm.yaml
```

4. インストール計画を承認し、適切な **namespace** を入力します。

```
oc patch installplan $(oc get ip -n <namespace> -o=jsonpath='{.items[?(@.spec.approved==false)].metadata.name}') -n <namespace> --type merge --patch '{"spec":{"approved":true}}'
```

5. Operator が実行されていることを確認します。

```
$ oc get pods
NAME                                READY STATUS RESTARTS AGE
rhssso-operator-558876f75c-m25mt  1/1   Running  0      28s
```

## 関連情報

- Operator のインストールが完了したら、最初のカスタムリソースを作成することができます。 [カスタムリソースを使用した Red Hat Single Sign-On インストール](#) を参照してください。
- OpenShift Operator の詳細は、 [OpenShift Operators guide](#) を参照してください。

## 11.2. 実稼働環境での RED HAT SINGLE SIGN-ON OPERATOR の使用

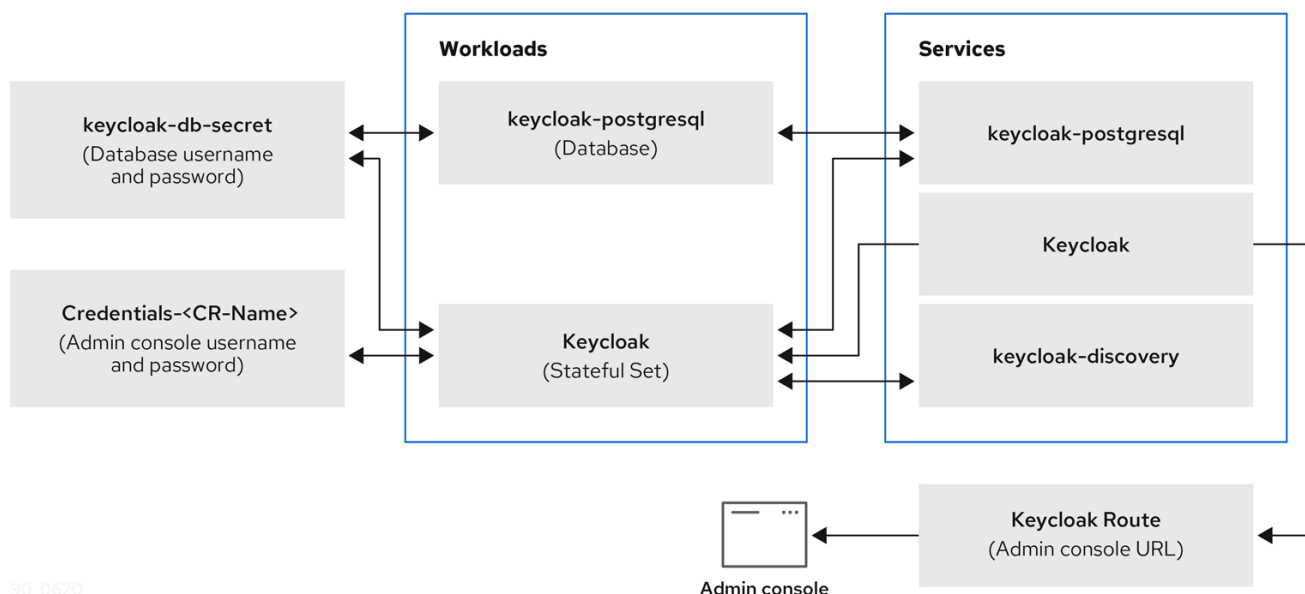
- 組み込み DB の使用は、実稼働環境ではサポートされていません。
- バックアップ CRD は非推奨であり、実稼働環境ではサポートされていません。
- Keycloak CR の **podDisruptionBudget** フィールドは非推奨となり、Operator が Kubernetes バージョン 1.25 以降にデプロイされると無視されます。
- それ以外の CRD については、**v1alpha1** バージョンにかかわらず本番環境で完全にサポートされています。この CRD バージョンで重大な変更を加える予定はありません。

## 11.3. カスタムリソースを使用した RED HAT SINGLE SIGN-ON のインストール

Operator を使用して Keycloak カスタムリソースを作成して、Red Hat Single Sign-On のインストールを自動化できます。カスタムリソースを使用して Red Hat Single Sign-On をインストールする際に、ここに説明されているコンポーネントおよびサービスを作成し、続いて表示されるグラフで示唆します。

- **keycloak-db-secret** - データベースのユーザー名、パスワード、および外部アドレスなどのプロパティを保存します (外部データベースに接続する場合)。
- **credentials-<CR-Name>** - Red Hat Single Sign-On 管理コンソールにログインする管理者のユーザー名とパスワードです (<CR-Name> は **Keycloak** カスタムリソース名に基づいています)。
- **keycloak** - 高可用性サポートのある StatefulSet として実装される Keycloak デプロイメント仕様
- **keycloak-postgresql** - PostgreSQL データベースインストールを開始します。
- **keycloak-discovery** サービス - **JDBC\_PING** 検出を実行します。
- **keycloak** サービス - HTTPS 経由で Red Hat Single Sign-On への接続 (HTTP はサポートされていません)
- **keycloak-postgresql** サービス - 内部および外部 (使用されている場合) のデータベースインスタンスを接続します。
- **keycloak** ルート - OpenShift から Red Hat Single Sign-On 管理コンソールにアクセスするための URL

Operator コンポーネントおよびサービスがどのように対話するか



90\_0620

### 11.3.1. Keycloak カスタムリソース

Keycloak カスタムリソースは、インストールのパラメーターを定義する YAML ファイルです。このファイルには、3つのプロパティが含まれます。

- **instances** - 高可用性モードで実行中のインスタンス数を制御します。
- **externalAccess** - **enabled** が **True** の場合、Operator は Red Hat Single Sign-On クラスターの OpenShift のルートを作成します。Route に自動選択されるホスト名を上書きするように **host** を設定できます。
- **externalDatabase** - 外部でホストされているデータベースに接続するため。このトピックは、本ガイドの [外部データベース](#) のセクションで説明しています。false に設定すると、テスト目的でのみ使用され、組み込み PostgreSQL データベースがインストールされます。  
externalDatabase:false は実稼働環境ではサポートされていないことに注意してください。

### Keycloak カスタムリソースの YAML ファイルのサンプル

```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: example-sso
  labels:
    app: sso
spec:
  instances: 1
  externalAccess:
    enabled: True
```



#### 注記

YAML ファイルを更新しても、Red Hat Single Sign-On 管理コンソールに表示される変更は更新できますが、管理コンソールへの変更はカスタムリソースを更新しません。

### 11.3.2. OpenShift での Keycloak カスタムリソースの作成

OpenShift では、カスタムリソースを使用して管理コンソールの URL であるルートを作成し、管理コンソールのユーザー名とパスワードを保持するシークレットを検索します。

## 前提条件

- このカスタムリソースの YAML ファイルがある。
- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

## 手順

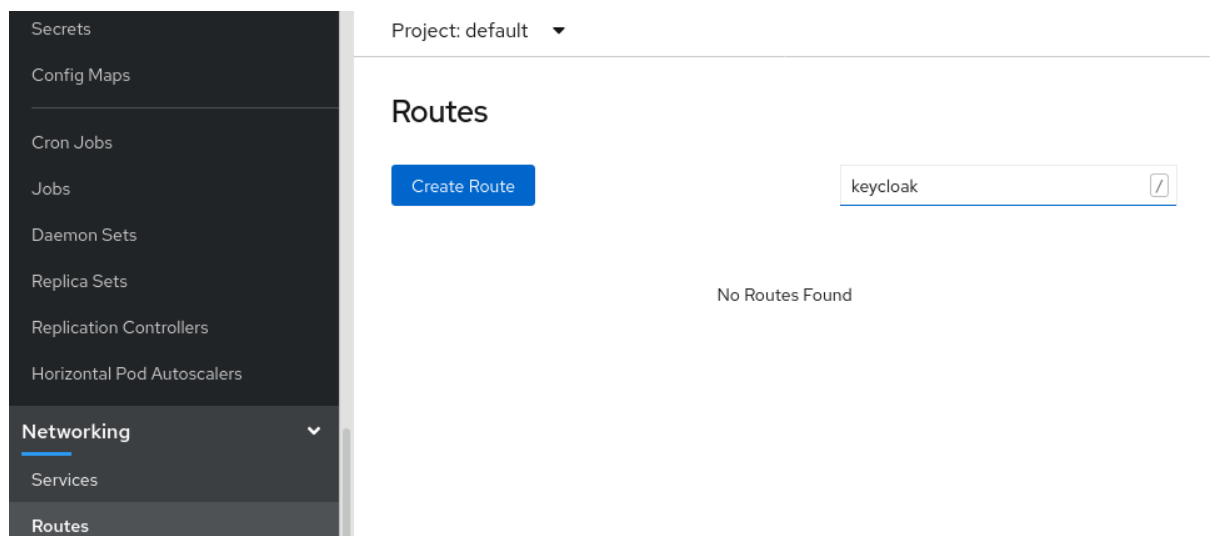
1. YAML ファイルを使用してルートを作成します (`oc create -f <filename>.yaml -n <namespace>`)。以下は例になります。

```
$ oc create -f sso.yaml -n sso
keycloak.keycloak.org/example-sso created
```

ルートは OpenShift に作成されます。

2. OpenShift Web コンソールにログインします。
3. **Networking**、**Routes** を選択し、Keycloak を検索します。

### OpenShift Web コンソールのルート画面



4. Keycloak ルートのある画面で、**Location** の下にある URL をクリックします。Red Hat Single Sign-On の管理コンソールのログイン画面が表示されます。

### 管理コンソールのログイン画面



Username or email

Password

Remember me

Log In

5. OpenShift Web コンソールで管理コンソールのユーザー名およびパスワードを確認します。**Workloads** で **Secrets** をクリックし、Keycloak を検索します。

### OpenShift Web コンソールのシークレット画面

Workloads  
Pods  
Deployments  
Deployment Configs  
Stateful Sets  
**Secrets**  
Config Maps  
Cron Jobs

Project: default

### Secrets

Create

keycloak

4 Image 0 Source 0 TLS 8 Service Account Token 0 Opaque

Select all filters 12 Items

6. 管理コンソールのログイン画面に、ユーザー名とパスワードを入力します。

### 管理コンソールのログイン画面

## Username or email

admin

## Password

●●●●●●●●

 Remember me

Log In

次に、Keycloak カスタムリソースによってインストールされた Red Hat Single Sign-On のインスタンスにログインしている。レルム、クライアント、およびユーザーのカスタムリソースを作成できます。

## Red Hat Single Sign-On マスターレルム

The screenshot shows the configuration page for the 'Master' realm. The left sidebar contains a navigation menu with options like 'Configure', 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', 'Authentication', 'Manage', 'Groups', 'Users', and 'Sessions'. The main content area is titled 'Master' and has tabs for 'General', 'Login', 'Keys', 'Email', 'Themes', 'Cache', 'Tokens', and 'Client Registration'. The 'General' tab is active, showing the following configuration:

- Name:** master
- Display name:** (empty)
- HTML Display name:** (empty)
- Frontend URL:** (empty)
- Enabled:** ON
- User-Managed Access:** OFF
- Endpoints:** OpenID Endpoint Configuration, SAML 2.0 Identity Provider Metadata

Buttons for 'Save' and 'Cancel' are visible at the bottom of the configuration area.

7. カスタムリソースのステータスを確認します。

```
$ oc describe keycloak <CR-name>
```

## 結果

Operator がカスタムリソースを処理した後に、以下のコマンドでステータスを表示します。

```
$ oc describe keycloak <CR-name>
```

## Keycloak カスタムリソースのステータス

```

Name:      example-keycloak
Namespace: keycloak
Labels:    app=sso
Annotations: <none>
API Version: keycloak.org/v1alpha1
Kind:      Keycloak
Spec:
  External Access:
    Enabled: true
    Instances: 1
Status:
  Credential Secret: credential-example-keycloak
  Internal URL:      https://<External URL to the deployed instance>
  Message:
  Phase:             reconciling
  Ready:             true
  Secondary Resources:
    Deployment:
      keycloak-postgresql
    Persistent Volume Claim:
      keycloak-postgresql-claim
    Prometheus Rule:
      keycloak
    Route:
      keycloak
    Secret:
      credential-example-keycloak
      keycloak-db-secret
    Service:
      keycloak-postgresql
      keycloak
      keycloak-discovery
    Service Monitor:
      keycloak
    Stateful Set:
      keycloak
  Version:
  Events:

```

### 関連資料

- Red Hat Single Sign-On のインストールが完了すると、[レルムカスタムリソースを作成](#) する準備が整います。
- 外部データベースはサポートされているオプションであり、Keycloak カスタムリソースで有効にする必要があります。このオプションはテストでのみ無効にし、実稼働環境に切り替えるときに有効にすることができます。[外部データベースへの接続](#) を参照してください。

## 11.4. レルムカスタムリソースの作成

Operator を使用して、カスタムリソースで定義されているように Red Hat Single Sign-On でレルムを作成できます。YAML ファイルで、レルムカスタムリソースのプロパティを定義します。



## 注記

レルムを作成または削除する唯一の方法は、YAML ファイルを削除または削除することで、変更内容が Red Hat Single Sign-On 管理コンソールに表示されます。ただし、管理コンソールへの変更は反映されず、レルムの作成後の CR の更新はサポートされません。

## Realm カスタムリソースの YAML ファイルの例

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
  name: test
  labels:
    app: sso
spec:
  realm:
    id: "basic"
    realm: "basic"
    enabled: True
    displayName: "Basic Realm"
  instanceSelector:
    matchLabels:
      app: sso
```

## 前提条件

- このカスタムリソースの YAML ファイルがある。
- YAML ファイルでは、**instanceSelector** の下にある **app** が、Keycloak カスタムリソースのラベルと一致します。これらの値に一致させることで、Red Hat Single Sign-On の適切なインスタンスでレルムを作成することができます。
- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

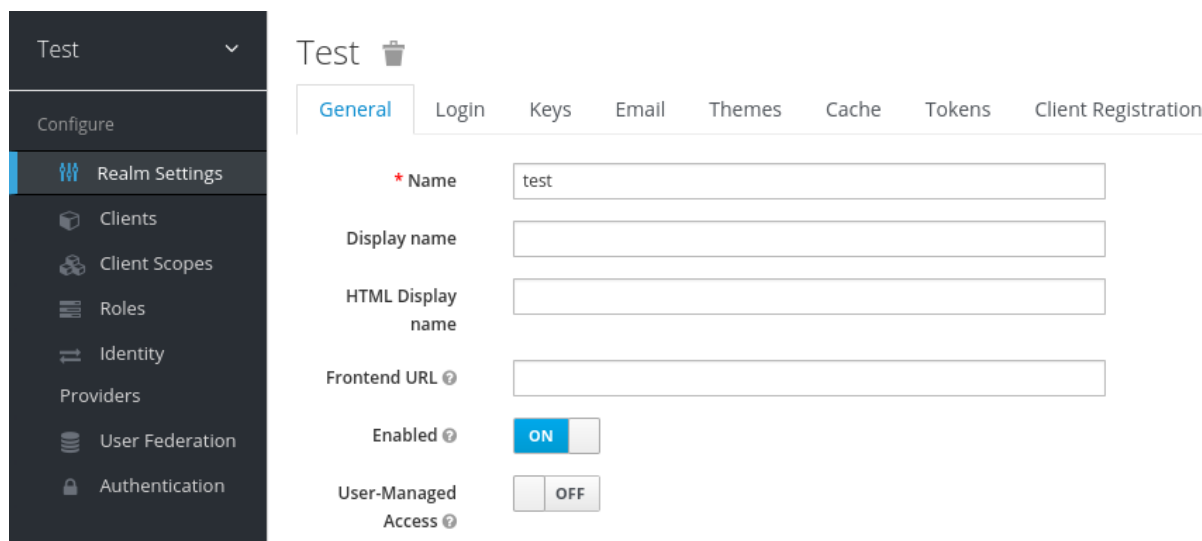
## 手順

1. このコマンドは、作成した YAML ファイルで使われます (**oc create -f <realm-name>.yaml**)。以下は例になります。

```
$ oc create -f initial_realm.yaml
keycloak.keycloak.org/test created
```

2. Red Hat Single Sign-On の関連インスタンス向けに、管理コンソールにログインします。
3. Select Realm をクリックし、作成したレルムを見つけます。新しいレルムが開きます。

## 管理コンソールのマスターレルム



## 結果

Operator がカスタムリソースを処理した後に、以下のコマンドでステータスを表示します。

```
$ oc describe keycloak <CR-name>
```

## レムカスタムリソースのステータス

```
Name:      example-keycloakrealm
Namespace: keycloak
Labels:    app=sso
Annotations: <none>
API Version: keycloak.org/v1alpha1
Kind:      KeycloakRealm
Metadata:
  Creation Timestamp: 2019-12-03T09:46:02Z
  Finalizers:
    realm.cleanup
  Generation: 1
  Resource Version: 804596
  Self Link: /apis/keycloak.org/v1alpha1/namespaces/keycloak/keycloakrealms/example-keycloakrealm
  UID:      b7b2f883-15b1-11ea-91e6-02cb885627a6
Spec:
  Instance Selector:
    Match Labels:
      App: sso
  Realm:
    Display Name: Basic Realm
    Enabled: true
    Id: basic
    Realm: basic
Status:
  Login URL:
  Message:
  Phase: reconciling
  Ready: true
  Events: <none>
```

## 関連資料

- レルムの作成が完了すると、[クライアントカスタムリソースを作成する](#) 準備が整います。

## 11.5. クライアントカスタムリソースの作成

Operator を使用して、カスタムリソースで定義されているように Red Hat Single Sign-On でクライアントを作成できます。レルムのプロパティを YAML ファイルに定義します。



## 注記

YAML ファイルを更新しても、Red Hat Single Sign-On 管理コンソールに表示される変更は更新できませんが、管理コンソールへの変更はカスタムリソースを更新しません。

## クライアントカスタムリソースの YAML ファイルの例

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakClient
metadata:
  name: example-client
  labels:
    app: sso
spec:
  realmSelector:
    matchLabels:
      app: <matching labels for KeycloakRealm custom resource>
  client:
    # auto-generated if not supplied
    #id: 123
    clientId: client-secret
    secret: client-secret
    # ...
    # other properties of Keycloak Client
```

## 前提条件

- このカスタムリソースの YAML ファイルがある。
- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

## 手順

1. このコマンドは、作成した YAML ファイルで使います (**oc create -f <client-name>.yaml**)。以下は例になります。

```
$ oc create -f initial_client.yaml
keycloak.keycloak.org/example-client created
```

2. Red Hat Single Sign-On の関連インスタンス用に、Red Hat Single Sign-On 管理コンソールにログインします。
3. Clients をクリックします。

クライアントのリストに新しいクライアントが表示されます。

Client ID	Enabled	Base URL	Actions		
account	True	http://localhost:8080/auth/realms/master/account/	Edit	Export	Delete
account-console	True	http://localhost:8080/auth/realms/master/account/	Edit	Export	Delete
admin-cli	True	Not defined	Edit	Export	Delete
broker	True	Not defined	Edit	Export	Delete
master-realm	True	Not defined	Edit	Export	Delete
security-admin-console	True	http://localhost:8080/auth/admin/master/console/	Edit	Export	Delete

## 結果

クライアントの作成後に、Operator は **keycloak-client-secret-*<custom resource name>*** という命名パターンを使用して **Client ID** とクライアントのシークレットが含まれる Secret を作成します。以下に例を示します。

### クライアントのシークレット

```
apiVersion: v1
data:
  CLIENT_ID: <base64 encoded Client ID>
  CLIENT_SECRET: <base64 encoded Client Secret>
kind: Secret
```

Operator がカスタムリソースを処理した後に、以下のコマンドでステータスを表示します。

```
$ oc describe keycloak <CR-name>
```

### クライアントのカスタムリソースのステータス

```
Name:      client-secret
Namespace: keycloak
Labels:    app=sso
API Version: keycloak.org/v1alpha1
Kind:      KeycloakClient
Spec:
  Client:
    Client Authenticator Type: client-secret
    Client Id:                 client-secret
    Id:                        keycloak-client-secret
  Realm Selector:
    Match Labels:
      App: sso
Status:
  Message:
  Phase: reconciling
  Ready: true
  Secondary Resources:
    Secret:
      keycloak-client-secret-client-secret
Events: <none>
```

## 関連資料

- クライアントの作成が完了すると、[ユーザーカスタムリソースを作成](#)する準備が整います。

## 11.6. ユーザーカスタムリソースの作成

Operator を使用して、カスタムリソースで定義されているように Red Hat Single Sign-On でユーザーを作成できます。YAML ファイルで、ユーザーカスタムリソースのプロパティを定義します。



### 注記

YAML ファイルのプロパティを更新すると、Red Hat Single Sign-On 管理コンソールに変更が表示されますが、管理コンソールを変更しても、カスタムリソースは更新されません。

同じことが認証情報にも当てはまることに注意してください。**credentials** フィールドが定義されている場合、ユーザーの認証情報は常に CR で設定された値と一致します。つまり、ユーザーがアカウントコンソールを介してパスワードを変更すると、Operator は CR で設定された値と一致するように、パスワードをリセットします。

### ユーザーカスタムリソースのサンプル YAML ファイル

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakUser
metadata:
  name: example-user
spec:
  user:
    username: "realm_user"
    firstName: "John"
    lastName: "Doe"
    email: "user@example.com"
    enabled: True
    emailVerified: False
    credentials:
      - type: "password"
        value: "12345"
    realmRoles:
      - "offline_access"
    clientRoles:
      account:
        - "manage-account"
      realm-management:
        - "manage-users"
    realmSelector:
      matchLabels:
        app: sso
```

### 前提条件

- このカスタムリソースの YAML ファイルがある。
- **realmSelector** は、既存のレルムカスタムリソースのラベルに一致する。



- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

## 手順

1. このコマンドは作成した YAML ファイルで使します (`oc create -f <user_cr>.yaml`)。以下は例になります。

```
$ oc create -f initial_user.yaml
keycloak.keycloak.org/example-user created
```

2. Red Hat Single Sign-On の関連インスタンス向けに、管理コンソールにログインします。
3. Users をクリックします。
4. YAML ファイルで定義したユーザーを検索します。  
ユーザーを検索するには、別のレルムに切り替える必要がある場合があります。

ID	Username	Email	Last Name	First Name	Actions
d6b907cd-e...	leiazuki	lazuki@wes...	Azuki	Lei	Edit Impersonate Delete
2a26e1b2-8...	nemethjabaz	njabaz@we...	Jabaz	Nemeth	Edit Impersonate Delete
7c1f0c04-0f...	realm_user	user@exam...	Doe	John	Edit Impersonate Delete

## 結果

ユーザーの作成後、Operator は `credential-<realm name>-<username>-<namespace>` という命名パターンを使用してシークレットを作成します。このシークレットには、ユーザー名と、CR `credentials` 属性で指定されている場合はパスワードが含まれます。

以下に例を示します。

### KeycloakUser シークレット

```
kind: Secret
apiVersion: v1
data:
  password: <base64 encoded password>
  username: <base64 encoded username>
type: Opaque
```

Operator がカスタムリソースを処理した後に、以下のコマンドでステータスを表示します。

```
$ oc describe keycloak <CR-name>
```

### ユーザーカスタムリソースのステータス

```
Name:      example-realm-user
Namespace: keycloak
Labels:    app=ssso
```

```

API Version: keycloak.org/v1alpha1
Kind:      KeycloakUser
Spec:
  Realm Selector:
    Match Labels:
      App: sso
  User:
    Email:      realm_user@redhat.com
  Credentials:
    Type:      password
    Value:     <user password>
  Email Verified: false
  Enabled:    true
  First Name: John
  Last Name:  Doe
  Username:   realm_user
Status:
  Message:
  Phase:     reconciled
Events:    <none>

```

## 関連資料

- 外部データベースがある場合は、Keycloak カスタムリソースを変更してサポートすることができます。[外部データベースへの接続](#) を参照してください。
- カスタムリソースを使用してデータベースのバックアップを作成するには、[データベースのバックアップのスケジュール](#) を参照してください。

## 11.7. 外部データベースへの接続

Operator を使用して、**keycloak-db-secret** YAML ファイルを作成し、Keycloak CR externalDatabase プロパティを有効に設定することで、外部 PostgreSQL データベースに接続できます。値は Base64 でエンコードされていることに注意してください。

### keycloak-db-secret の YAML ファイルサンプル

```

apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-secret
  namespace: keycloak
stringData:
  POSTGRES_DATABASE: <Database Name>
  POSTGRES_EXTERNAL_ADDRESS: <External Database URL (resolvable by K8s)>
  POSTGRES_EXTERNAL_PORT: <External Database Port>
  POSTGRES_PASSWORD: <Database Password>
  # Required for AWS Backup functionality
  POSTGRES_SUPERUSER: "true"
  POSTGRES_USERNAME: <Database Username>
  SSLMODE: <TLS configuration for the Database connection>
type: Opaque

```

以下のプロパティは、データベースのホスト名または IP アドレスとポートを設定します。

- **POSTGRES\_EXTERNAL\_ADDRESS** - 外部データベースのホスト名。ホスト名ではなくデータベースの IP のみがある場合は、サービスと対応する **EndpointSlice** または **Endpoint** を作成してホスト名を指定します。
- **POSTGRES\_EXTERNAL\_PORT** - (必要に応じて) データベースポート。

他のプロパティは、ホストされたデータベースまたは外部データベースと同じ方法で機能します。以下のように設定します。

- **POSTGRES\_DATABASE** - 使用されるデータベース名。
- **POSTGRES\_USERNAME** - データベースのユーザー名
- **POSTGRES\_PASSWORD** - データベースのパスワード
- **POSTGRES\_SUPERUSER** - バックアップがスーパーユーザーとして実行されるかどうかを示します。通常は **true** です。
- **SSLMODE** - 外部 PostgreSQL データベースへの接続で TLS を使用するかどうかを示します。使用できる **値**を確認します。

**SSLMODE** が有効になっている場合、Operator は、PostgreSQL データベースで使用されている **root.crt** を含む **keycloak-db-ssl-cert-secret** と呼ばれるシークレットを検索します。シークレットの作成は任意で、データベースの証明書を検証する場合にのみシークレットが使用されます (例: **SSLMODE: verify-ca**)。以下は例です。

Operator によって使用される TLS Secret の YAML ファイルのサンプル。

```
apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-ssl-cert-secret
  namespace: keycloak
type: Opaque
data:
  root.crt: {root.crt base64}
```

Operator は **keycloak-postgresql** という名前のサービスを作成します。このサービスは、**POSTGRES\_EXTERNAL\_ADDRESS** のコンテンツに基づいて外部データベースを公開するように、Operator によって設定されます。Red Hat Single Sign-On は、このサービスを使用してデータベースに接続します。つまり、データベースに直接接続するのではなく、このサービスを介して接続します。

Keycloak カスタムリソースには、外部データベースのサポートを有効にするために更新が必要です。

外部データベースをサポートする Keycloak カスタムリソースの YAML ファイルの例

```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  labels:
    app: sso
  name: example-keycloak
  namespace: keycloak
spec:
```

```
externalDatabase:
  enabled: true
  instances: 1
```

## 前提条件

- **keycloak-db-secret** の YAML ファイルがある。
- Keycloak カスタムリソースを変更して **externalDatabase** を **true** に設定している。
- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

## 手順

1. PostgreSQL データベースのシークレットを検索します (**oc get secret <secret\_for\_db> -o yaml**)。以下は例になります。

```
$ oc get secret keycloak-db-secret -o yaml
apiVersion: v1
data
  POSTGRES_DATABASE: cm9vdA==
  POSTGRES_EXTERNAL_ADDRESS: MTcyLjE3LjAuMw==
  POSTGRES_EXTERNAL_PORT: NTQzMg==
```

**POSTGRES\_EXTERNAL\_ADDRESS** は Base64 形式です。

2. シークレットの値をデコードします (**echo "<encoded\_secret>" | base64 -decode**)。以下は例になります。

```
$ echo "MTcyLjE3LjAuMw==" | base64 -decode
192.0.2.3
```

3. デコードされた値がデータベースの IP アドレスと一致していることを確認します。

```
$ oc get pods -o wide
NAME                READY STATUS  RESTARTS  AGE  IP
keycloak-0          1/1  Running  0        13m  192.0.2.0
keycloak-postgresql-c8vv27m 1/1  Running  0        24m  192.0.2.3
```

4. 実行中のサービスのリストに **keycloak-postgresql** が表示されることを確認します。

```
$ oc get svc
NAME                TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)  AGE
keycloak            ClusterIP   203.0.113.0   <none>       8443/TCP 27m
keycloak-discovery  ClusterIP   None          <none>       8080/TCP 27m
keycloak-postgresql ClusterIP   203.0.113.1   <none>       5432/TCP 27m
```

**keycloak-postgresql** サービスは、バックエンドの IP アドレスのセットにリクエストを送信します。これらの IP アドレスはエンドポイントと呼ばれます。

5. **keycloak-postgresql** サービスで使用されるエンドポイントを表示して、それらがデータベースの IP アドレスを使用していることを確認します。

```
$ oc get endpoints keycloak-postgresql
NAME                ENDPOINTS    AGE
keycloak-postgresql 192.0.2.3.5432 27m
```

6. Red Hat Single Sign-On が外部データベースで実行されていることを確認します。この例は、すべてが実行されていることを示しています。

```
$ oc get pods
NAME                READY STATUS  RESTARTS  AGE  IP
keycloak-0          1/1   Running  0         26m  192.0.2.0
keycloak-postgresql-c8vv27m 1/1   Running  0         36m  192.0.2.3
```

## 11.8. 外部の RED HAT SINGLE SIGN-ON への接続

この Operator を使用して、外部の Red Hat Single Sign-On インスタンスを部分的に管理できます。現在の状態では、クライアントのみを作成できます。

これには、ターゲットおよび設定に使用する **Keycloak** および **KeycloakRealm** CRD のマネージドバージョンを作成する必要があります。

### external-keycloakの YAML ファイルサンプル

```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: external-ref
  labels:
    app: external-sso
spec:
  unmanaged: true
  external:
    enabled: true
    url: https://some.external.url
```

この keycloak に対して認証するため、Operator は CRD 名に **credential-** の接頭辞を追加して CRD からシークレット名を推測します。

### credential-external-ref の YAML ファイルの例

```
apiVersion: v1
kind: Secret
metadata:
  name: credential-external-ref
type: Opaque
data:
  ADMIN_USERNAME: YWRtaW4=
  ADMIN_PASSWORD: cGFzcw==
```

### external-realm の YAML ファイルの例

```
apiVersion: keycloak.org/v1alpha1
kind: KeycloakRealm
metadata:
```

```

name: external-realm
labels:
  app: external-sso
spec:
  unmanaged: true
  realm:
    id: "basic"
    realm: "basic"
  instanceSelector:
    matchLabels:
      app: external-sso

```

通常どおりクライアントでレム参照を使用できるようになり、外部の Red Hat Single Sign-On インスタンスにクライアントが作成されます。

## 11.9. データベースバックアップのスケジューリング



### 警告

バックアップ CR は**非推奨**であり、将来のリリースで削除される可能性があります。

Operator を使用して、カスタムリソースで定義されるデータベースの自動バックアップをスケジュールできます。カスタムリソースは、バックアップジョブをトリガーし、そのステータスを報告します。

Operator を使用して、ローカルの永続ボリュームへのワンタイムバックアップを実行するバックアップジョブを作成できます。

### バックアップカスタムリソースの YAML ファイルの例

```

apiVersion: keycloak.org/v1alpha1
kind: KeycloakBackup
metadata:
  name: test-backup

```

### 前提条件

- このカスタムリソースの YAML ファイルがある。
- Red Hat Single Sign-On Operator によって作成される **PersistentVolumeClaim** についてのみ予約する **claimRef** の **PersistentVolume** が必要です。

### 手順

1. バックアップジョブ (**oc create -f <backup\_crname>**) を作成します。以下は例になります。

```

$ oc create -f one-time-backup.yaml
keycloak.keycloak.org/test-backup

```

Operator は、**Keycloak-backup-<CR-name>** という命名スキームを使用して **PersistentVolumeClaim** を作成します。

2. ボリュームのリストを表示します。

```
$ oc get pvc
NAME                                STATUS VOLUME
keycloak-backup-test-backup         Bound  pvc-e242-ew022d5-093q-3134n-41-adff
keycloak-postgresql-claim          Bound  pvc-e242-vs29202-9bcd7-093q-31-zadj
```

3. バックアップジョブのリストを表示します。

```
$ oc get jobs
NAME          COMPLETIONS  DURATION  AGE
test-backup  0/1           6s        6s
```

4. 実行したバックアップジョブのリストを表示します。

```
$ oc get pods
NAME                                READY STATUS RESTARTS AGE
test-backup-5b4rf                   0/1   Completed 0      24s
keycloak-0                           1/1   Running  0      52m
keycloak-postgresql-c824c6-vv27m    1/1   Running  0      71m
```

5. 完了したバックアップジョブログを表示します。

```
$ oc logs test-backup-5b4rf
==> Component data dump completed
.
.
.
.
```

## 関連資料

- 永続ボリュームの詳細は、[Understanding persistent storage](#) を参照してください。

## 11.10. 拡張機能とテーマのインストール

Operator を使用して、エクステンションおよび会社または組織に必要な拡張機能をインストールできます。エクステンションまたはテーマは、Red Hat Single Sign-On が使用できる任意のものになります。たとえば、メトリック拡張を追加できます。エクステンションまたはテーマを Keycloak カスタムリソースに追加します。

### Keycloak カスタムリソースの YAML ファイルのサンプル

```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: example-keycloak
  labels:
    app: sso
spec:
```

```
instances: 1
extensions:
  - <url_for_extension_or_theme>
externalAccess:
  enabled: True
```

他の拡張と同じ方法でパッケージ化し、デプロイできます。詳細は、[テーマの導入](#) マニュアルエントリーを参照してください。

### 前提条件

- Keycloak カスタムリソースの YAML ファイルがある。
- cluster-admin パーミッション、または管理者によって付与される同等のレベルのパーミッションがある。

### 手順

1. Keycloak カスタムリソースの YAML ファイルを編集します (`oc edit <CR-name>`)。
2. **instances** の行に **extensions:** という行を追加します。
3. カスタムエクステンションまたはテーマの JAR ファイルに URL を追加します。
4. ファイルを保存します。

Operator は拡張またはテーマをダウンロードし、これをインストールします。

## 11.11. カスタムリソースを管理するためのコマンドオプション

カスタムリクエストの作成後に、**oc** コマンドを使用してこれを編集するか、削除することができます。

- カスタム要求を編集するには、**oc edit <CR-name>** コマンドを使用します。
- カスタム要求を削除するには、コマンド **oc delete <CR-name>** を使用します。

たとえば、**test-realm** という名前のレルムカスタム要求を編集するには、次のコマンドを使用します。

```
$ oc edit test-realm
```

変更を実行できるウィンドウが開きます。



### 注記

Keycloak CR YAML ファイルを更新して、変更がデプロイメントに適用されます。

他のリソースへの更新は制限されます。

Keycloak Realm CR は、同期オプションなしでの基本的な作成および削除のみをサポートします。Keycloak User および Client CR は一方向更新をサポートします (CR に変更は Keycloak に反映されますが、Keycloak で実行された変更は CR で更新されません)。

## 11.12. アップグレードストラテジー



Operator による Red Hat Single Sign-On のアップグレード方法を設定できます。以下のアップグレードストラテジーを選択できます。

- **recreate**: これはデフォルトのストラテジーです。オペレーターは、すべての Red Hat Single Sign-On レプリカを削除し、オプションでバックアップを作成し、新しい Red Hat Single Sign-On イメージに基づいてレプリカを作成します。このストラテジーは、単一の Red Hat Single Sign-On バージョンとして、基礎となるデータベースにアクセスするのに適しています。マイナスイメージは、アップグレード中に Red Hat Single Sign-On をシャットダウンする必要があります。
- **rolling**: オペレーターは一度に1つのレプリカを削除し、新しい Red Hat Single Sign-On イメージに基づいて再び作成します。これにより、ゼロダウンタイムのアップグレードが確保されますが、データベースの移行を必要としないマイナーバージョンアップグレードには、複数の Red Hat Single Sign-On バージョンが同時にアクセスされるため、データベースの移行は必要ありません。このストラテジーでは、自動バックアップはサポートされていません。

### Keycloak カスタムリソースの YAML ファイルのサンプル

```
apiVersion: keycloak.org/v1alpha1
kind: Keycloak
metadata:
  name: example-keycloak
  labels:
    app: sso
spec:
  instances: 2
  migration:
    strategy: recreate
  backups:
    enabled: True
  externalAccess:
    enabled: True
```

#### 注記

以前のバージョンの Operator で発生した [バグ](#) が原因で、Red Hat Single Sign-On StatefulSet の **Selector** フィールドは、設定によっては、誤設定される可能性があります。このような状態が Operator によって検出され、さらに **recreate** ストラテジーを使用している場合には、適切な **Selector** を使用して **StatefulSet** が削除および再作成されます。これは、**Selector** フィールドが変更できないため、必要になります。

削除の操作を行うと、数は少ないですが、危険な結果に陥る可能性があります。たとえば、Operator で認識されていないカスタムの機能を StatefulSet 定義に追加した場合に、代わりに StatefulSet を手動で削除できます。