Red Hat build of Keycloak 24.0

サーバーガイド

Last Updated: 2024-06-04

法律上の通知

概要

このガイドには、Red Hat build of Keycloak 24.0 の設定に関する管理者向け情報が記載されています。

目次

多様性を受け入れるオープンソースの強化

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

第1章 RED HAT BUILD OF KEYCLOAK の設定

この章では、Red Hat build of Keycloak の設定方法と、設定を開始して適用する方法について説明します。これには、Red Hat build of Keycloak を最適化して起動を高速化し、メモリー使用量を減らすための設定ガイドラインが含まれています。

1.1. RED HAT BUILD OF KEYCLOAK のソース設定

Red Hat build of Keycloak は、次の 4 つのソースから設定をロードします。ここでは適用順にリストされています。

1. コマンドラインパラメーター
2. 環境変数
3. **conf/keycloak.conf** ファイルまたはユーザーが作成した設定ファイルで定義されたオプション
4. ユーザーが作成した Java KeyStore ファイルで定義された機密オプション

オプションが複数のソースに設定されている場合、そのオプションの値はリストの最初にあるソースにより決定されます。たとえば、コマンドラインパラメーターにより設定されたオプションの値は、同じオプションの環境変数よりも優先されます。

1.1.1. 例: db-url-host パラメーターの設定

次の例は、**db-url** 値が 4 つの設定ソースでどのように設定されるかを示しています。

ソース	形式
コマンドラインパラメーター	**--db-url=cliValue**
環境変数	**KC_DB_URL=envVarValue**
設定ファイル	**db-url=confFileValue**
Java KeyStore ファイル	**kc.db-url=keystoreValue**

アプリケーションの優先順位に基づくと、最も優先順位が高いのはコマンドラインであるため、起動時に使用される値は **cliValue** になります。

--db-url=cliValue が使用されていない場合、適用される値は **KC_DB_URL=envVarValue** になります。値がコマンドラインまたは環境変数によって適用されていない場合は、**db-url=confFileValue** が使用されます。前述の値がいずれも適用されていない場合は、使用可能な設定ソースの中で優先順位が最も低い **kc.db-url=confFileValue** の値が使用されます。

1.2. 設定用の形式

この設定では、**ソースごとに統一された** 形式が使用されており、キー/値ペアの、ある設定ソースから別の設定ソースへの変換が簡素化されます。この形式は spi オプションにも適用されることに注意してください。

コマンドラインパラメーターの形式

コマンドラインの値には、**--<key-with-dashes>=<value>** の形式が使用されます。一部の値については、**-<abbreviation>=<value>** の省略表現もあります。

環境変数の形式

環境変数の値には、大文字の **KC_<key_with_underscores>=<value>** 形式が使用されます。

設定ファイルの形式

設定ファイルに格納される値には、**<key-with-dashes>=<value>** 形式が使用されます。

KeyStore 設定ファイルの形式

KeyStore 設定ファイルに格納される値には、**kc.<key-with-dashes>** 形式が使用されます。**<value>** は、KeyStore に保存されているパスワードです。

設定の各章の最後で、該当する設定形式を定義する **関連オプション** の見出しを探してください。すべての設定オプションについては、すべての設定 を参照してください。ユースケースに適用できる設定ソースと形式を選択します。

1.2.1. 例 - 設定ソースに基づく代替形式

次の例は、3 つの設定ソースにおける **db-url-host** の設定形式を示しています。

コマンドラインパラメーター

```
bin/kc.[sh|bat] start --db-url-host=mykeycloakdb
```

環境変数

```
export KC_DB_URL_HOST=mykeycloakdb
```

conf/keycloak.conf

```
db-url-host=mykeycloakdb
```

1.2.2. コマンドラインパラメーターの形式

Red Hat build of Keycloak には、設定用の多くのコマンドラインパラメーターが同梱されています。使用可能な設定形式を確認するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --help
```

もしくは、すべての設定 ですべてのサーバーオプションを確認できます。

1.2.3. 環境変数の形式

${ENV_VAR} 構文を使用することで、**keycloak.conf** ファイル内の環境変数から環境固有の値をプレースホルダーを使用して解決できます。

```
db-url-host=${MY_DB_HOST}
```

環境変数を解決できない場合は、フォールバック値を指定できます。ここで示すとおり、**mydb** の前に **:** (コロン) を使用します。

```
db-url-host=${MY_DB_HOST:mydb}
```

1.2.4. 特定の設定ファイルを含める形式

デフォルトでは、サーバーは常に **conf/keycloak.conf** ファイルから設定オプションを取得します。新規インストールの場合、このファイルには、実稼働環境で実行する際の設定案がコメントとして格納されているだけです。

次のコマンドを入力し、**[-cf|--config-file]** オプションを使用して設定ファイルの場所を明示的に指定することもできます。

```
bin/kc.[sh|bat] --config-file=/path/to/myconfig.conf start
```

このオプションを設定すると、Red Hat build of Keycloak は **conf/keycloak.conf** ではなく指定されたファイルから設定を読み取ります。

1.2.5. Java KeyStore ファイルを使用して機密オプションを設定する

KeyStore 設定ソースにより、**[--config-keystore]** および **[--config-keystore-password]** オプションを使用して Java KeyStore からプロパティーを直接ロードできます。必要に応じて、**[--config-keystore-type]** オプションを使用して KeyStore タイプを指定できます。デフォルトの KeyStore タイプは **PKCS12** です。

KeyStore 内のシークレットは、**PBE** (パスワードベースの暗号化) キーアルゴリズムを使用して保存する必要があります。この場合のキーは、KeyStore のパスワードから導出します。次の **keytool** コマンドを使用して、このような KeyStore を生成できます。

```
keytool -importpass -alias kc.db-password -keystore keystore.p12 -storepass keystorepass -
storetype PKCS12 -v
```

コマンドを実行すると、**Enter the password to be stored**というプロンプトが表示されます。これは、上記の **kc.db-password** プロパティーの値を表します。

KeyStore が作成されると、次のパラメーターを使用してサーバーを起動できます。

```
bin/kc.[sh|bat] start --config-keystore=/path/to/keystore.p12 --config-keystore-password=storepass --
config-keystore-type=PKCS12
```

1.2.6. raw Quarkus プロパティーの形式

ほとんどの場合、使用可能な設定オプションでサーバーを設定できます。ただし、Red Hat build of Keycloak 設定に欠けている特定の動作または機能については、基礎となる Quarkus フレームワークのプロパティーを使用できます。

可能であれば、Quarkus から直接プロパティーを使用することは避けてください。それらは Red Hat build of Keycloak でサポートされていません。どうしても必要な場合は、まず 機能拡張リクエスト を作成することを検討してください。このアプローチは、ニーズに合わせて Red Hat build of Keycloak の設定を改善するのに役立ちます。

拡張リクエストが不可能な場合は、raw Quarkus プロパティーを使用してサーバーを設定できます。

1. **conf** ディレクトリーに、**quarkus.properties** ファイルを作成します。
2. そのファイルで、必要なプロパティーを定義します。
   Quarkus ドキュメント で定義されている Quarkus 拡張機能の サブセット のみ使用できます。
   以下に示す、Quarkus プロパティーの違いにも注意してください。

- Quarkus ドキュメント に示される Quarkus プロパティーの鍵アイコンは、ビルド時のプロパティーを表しています。このプロパティーを適用するには、**build** コマンドを実行します。ビルドコマンドの詳細は、Red Hat build of Keycloak の最適化に関する後続セクションを参照してください。

- Quarkus ガイドの鍵アイコンがないプロパティーは、Quarkus および Red Hat build of Keycloak のランタイムプロパティーです。

3. **[-cf|--config-file]** コマンドラインパラメーターを使用して、そのファイルを含めます。

同様に、Quarkus プロパティーを Java KeyStore に保存することもできます。

quarkus.http.port や同様の必須プロパティーなど、一部の Quarkus プロパティーはすでに Red Hat build of Keycloak 設定にマップされていることに注意してください。プロパティーが Red Hat build of Keycloak によって使用されている場合、**quarkus.properties** でそのプロパティーキーを定義しても効果はありません。Red Hat build of Keycloak の設定値は、Quarkus のプロパティー値よりも優先されます。

1.2.7. 値に特殊文字を使用する

Red Hat build of Keycloak は、Quarkus と MicroProfile に依存して設定値を処理します。評価式がサポートされていることに注意してください。たとえば、**${some_key}** は **some_key** の値と評価されます。

式の評価を無効にするには、\ 文字をエスケープ文字として使用します。特に、**$** で式を定義する場合や $ が繰り返し現れる場合、\ によって $ の使用をエスケープする必要があります。たとえば、設定値 **my$$password** が必要な場合は、代わりに **my\$\$password** を使用します。ほとんどの Unix シェルを使用する場合、またはプロパティーファイルに現れる場合は、\ 文字を追加でエスケープするか、引用符で囲む必要があることに注意してください。たとえば、bash で一重引用符を使用すると、単一のバックスラッシュ **--db-password='my\$\$password'** が保持されます。また、bash で二重引用符を使用する場合は、バックスラッシュがもう 1 つ **--db-password="my\\$\\$password"** が必要です。同様に、プロパティーファイルでも、バックスラッシュ文字をエスケープする必要があります (**kc.db-password=my\\$\\$password**)。

1.3. RED HAT BUILD OF KEYCLOAK の起動

Red Hat build of Keycloak は、**development mode** または **production mode** で起動できます。各モードでは、対象となる環境に応じて異なるデフォルトが提供されます。

1.3.1. Red Hat build of Keycloak を開発モードで起動する

開発モードは、Red Hat build of Keycloak を初めて試す場合に、すばやく起動するために使用します。このモードは、Red Hat build of Keycloak の開発など、開発者にとって便利なデフォルト設定を提供します。

開発モードで起動するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start-dev
```

デフォルト

開発モードでは、次のデフォルト設定が適用されます。

- HTTP は有効

- 厳密なホスト名解決は無効
- キャッシュはローカルに設定 (高可用性のため、分散キャッシュメカニズムは使用されません)
- テーマとテンプレートのキャッシュは無効

1.3.2. Red Hat build of Keycloak を実稼働モードで起動する

Red Hat build of Keycloak を実稼働環境にデプロイするには、実稼働モードを使用します。このモードは、**セキュアバイデフォルト** (デフォルトでセキュア) の原則に従います。

実稼働モードで起動するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start
```

追加で設定を行わなければ、このコマンドを実行しても Red Hat build of Keycloak は起動せず、代わりにエラーが表示されます。Red Hat build of Keycloak は **セキュアバイデフォルト** の原則に従っているため、この応答は意図的なものです。実稼働モードでは、起動時にホスト名を設定し、HTTPS/TLS 設定を使用可能にすることが想定されています。

デフォルト

実稼働モードでは、次のデフォルトが設定されます。

- トランスポート層セキュリティー (HTTPS) が必須であるため、HTTP は無効になっています。
- ホスト名の設定が想定されています。
- HTTPS/TLS の設定が想定されています。

Red Hat build of Keycloak を実稼働環境にデプロイする前に、必ず Red Hat build of Keycloak を実稼働用に設定する に概説されている手順に従ってください。

デフォルトでは、実稼働モードの設定オプション例は、デフォルトの **conf/keycloak.conf** ファイル内でコメント化されています。これらのオプションは、実稼働環境で Red Hat build of Keycloak を実行する際に考慮すべき主要な設定についてアイデアを提供します。

1.4. 初期管理者ユーザーの作成

初期管理者ユーザーは、ローカル接続 (localhost) を使用してアクセスする Web フロントエンドを使用して作成できます。代わりに、環境変数を使用してこのユーザーを作成することもできます。初期管理者ユーザー名は **KEYCLOAK_ADMIN= <username>**、初期管理者パスワードは **KEYCLOAK_ADMIN_PASSWORD= <password>** を設定します。

Red Hat build of Keycloak は、初回起動時にこれらの値を解析して、管理者権限を持つ最初のユーザーを作成します。管理者権限を持つ最初のユーザーが存在する場合は、管理コンソールまたはコマンドラインツール **kcadm.[sh|bat]** を使用して追加のユーザーを作成できます。

初期管理者がすでに存在し、起動時に環境変数がまだ存在している場合は、初期管理者の作成が失敗したことを示すエラーメッセージがログに表示されます。Red Hat build of Keycloak はこの値を無視し、正しく起動します。

1.5. RED HAT BUILD OF KEYCLOAK の起動を最適化する

Red Hat build of Keycloak を実稼働環境にデプロイする前に、Red Hat build of Keycloak を最適化して

起動を高速化し、メモリー消費量を改善することが推奨されます。このセクションでは、最適なパフォーマンスと実行時の動作を実現するために、Red Hat build of Keycloak の最適化を適用する方法について説明します。

1.5.1. 最適化された Red Hat build of Keycloak ビルドの作成

デフォルトでは、**start** または **start-dev** コマンドを使用すると、Red Hat build of Keycloak は便宜上、内部で **build** コマンドを実行します。

この **build** コマンドは、起動時と実行時の動作に対して一連の最適化を実行します。ビルドプロセスには数秒かかる場合があります。特に、Kubernetes や OpenShift などのコンテナー化された環境で Red Hat build of Keycloak を実行する場合、起動時間は重要です。無駄な時間を発生させないために、起動前に **build** (CI/CD パイプラインの別のステップなど) を明示的に実行します。

1.5.1.1. 最初のステップ: build を明示的に実行する

build を実行するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build <build-options>
```

このコマンドは、入力した **build options** を表示します。Red Hat build of Keycloak は、 **build** コマンドの実行時に使用できる **ビルドオプション** と、サーバーの起動時に使用できる **設定オプション** を区別します。

Red Hat build of Keycloak の起動が最適化されていない場合、この区別には意味がありません。ただし、起動前に build を実行する場合、build コマンドではオプションのサブセットしか使用できません。この制限は、最適化された Red Hat build of Keycloak イメージに対して build オプションが永続化されることが原因です。たとえば、**db-password** (設定オプション) などの認証情報の設定は、セキュリティー上の理由から永続化することは禁止されています。

警告

すべてのビルドオプションは、プレーンテキストで保持されます。機密データは、ビルドオプションとして保存しないでください。これは、KeyStore Config Source を含む、使用可能なすべての設定ソースに適用されます。したがって、ビルドオプションを Java KeyStore に保存することも推奨されません。設定オプションに関しては、主に機密データの保存に KeyStore Config Source を使用することが推奨されます。機密性のないデータの場合は、残りの設定ソースを使用できます。

ビルドオプションは、All configuration でツールアイコンでマークされます。利用可能なビルドオプションを見つけるには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --help
```

例: 起動前に build を実行してデータベースを PostgreSQL に設定する

```
bin/kc.[sh|bat] build --db=postgres
```

1.5.1.2. 2 番目のステップ: --optimized を使用して Red Hat build of Keycloak を起動する

ビルドが成功すると、次のコマンドを入力して Red Hat build of Keycloak を起動し、デフォルトの起動動作をオフにできます。

```
bin/kc.[sh|bat] start --optimized <configuration-options>
```

--optimized パラメーターは、Red Hat build of Keycloak に対して、事前にビルドおよび最適化された Red Hat build of Keycloak の使用を前提とするように指示します。その結果、Red Hat build of Keycloak の起動時にビルドの直接確認と実行は行われず、時間が短縮されます。

起動時にすべての設定オプションを入力できます。これらのオプションは、All configuration でツールアイコンが **付いていない** オプションです。

- 起動時に、**build** の入力時に使用した値と同じ値のビルドオプションが見つかった場合、**--optimized** パラメーターを使用すると、そのオプションは暗黙的に無視されます。
- そのオプションの値が、ビルドの入力時に使用した値と異なる場合、ログに警告が表示され、以前にビルドした値が使用されます。この値を有効にするには、起動する前に新しい **build** を実行します。

最適化されたビルドを作成する

次の例は、Red Hat build of Keycloak の起動時に、**--optimized** パラメーターを使用して最適化されたビルドを作成する方法を示しています。

1. build コマンドを使用して、PostgreSQL データベースベンダーのビルドオプションを設定します。

```
bin/kc.[sh|bat] build --db=postgres
```

2. **conf/keycloak.conf** ファイルで、postgres の実行時設定オプションを設定します。

```
db-url-host=keycloak-postgres
db-username=keycloak
db-password=change_me
hostname=mykeycloak.acme.com
https-certificate-file
```

3. 最適化されたパラメーターでサーバーを起動します。

```
bin/kc.[sh|bat] start --optimized
```

build コマンドを使用すると、起動時と実行時の動作のほとんどを最適化できます。また、**keycloak.conf** ファイルを設定ソースとして使用すると、CLI 自体の初期化など、コマンドラインパラメーターが必要となる起動時の一部の手順を回避できます。その結果、サーバーの起動時間がさらに短縮されます。

1.6. 基礎となる概念

このセクションでは、Red Hat build of Keycloak が使用する基礎となる概念、特に起動の最適化にかかわる概念について説明します。

Red Hat build of Keycloak は、Quarkus フレームワークと再拡張/mutable-jar アプローチを内部で使用します。このプロセスは、**build** コマンドが実行されると開始されます。

以下は、**build** コマンドにより実行される最適化の一部です。

- インストールされているプロバイダーに関する閉世界仮説が作成されます。つまり、Red Hat build of Keycloak の起動ごとに、レジストリーを再作成してファクトリーを初期化する必要がなくなります。
- 設定ファイルは、サーバー起動時の I/O を削減するために事前に解析されます。
- データベース固有のリソースは、特定のデータベースベンダーに対して実行するように設定および準備されています。
- ビルドオプションをサーバーイメージに対して永続化すると、サーバーは設定オプションを解釈して自身を (再) 設定するための追加の手順を実行しません。

詳細は、該当する Quarkus ガイド を参照してください。

第2章 RED HAT BUILD OF KEYCLOAK を実稼働用に設定する

Red Hat build of Keycloak の実稼働環境は、数千人のユーザーをサポートするオンプレミスのデプロイメントから数百万のユーザーにサービスを提供するデプロイメントまで、幅広いデプロイメントに対してセキュアな認証と認可を提供します。

この章では、実稼働に対応した Red Hat build of Keycloak 環境に必要な設定に関する一般的な情報を提供します。この情報は、環境に応じて異なる実際の実装ではなく、一般的な概念に重点を置いています。この章で説明する重要な側面は、コンテナー化、オンプレミス、GitOps、Ansible のいずれかにかかわらず、すべての環境に当てはまります。

2.1. セキュアな通信のための SSL/TLS

Red Hat build of Keycloak は、継続的に機密データを交換します。つまり、Red Hat build of Keycloak との間のすべての通信には、セキュアな通信チャネルが必要です。さまざまな攻撃ベクトルを防ぐには、そのチャネルに対して HTTP over TLS (HTTPS) を有効にします。

Red Hat build of Keycloak のためにセキュアな通信チャネルを設定するには、TLS の設定 および 送信 HTTP 要求の設定 を参照してください。

Red Hat build of Keycloak のキャッシュ通信を保護するには、分散キャッシュの設定 を参照してください。

2.2. RED HAT BUILD OF KEYCLOAK のホスト名

通常、実稼働環境では、Red Hat build of Keycloak インスタンスはプライベートネットワークで実行されます。しかし、保護すべきアプリケーションと通信するために、Red Hat build of Keycloak は特定のパブリック向けエンドポイントを公開する必要があります。

エンドポイントカテゴリーの詳細と、それらのパブリックホスト名を設定する方法については、ホスト名の設定 を参照してください。

2.3. 分散環境でのリバースプロキシー

通常、ホスト名の設定 とは別に、実稼働環境にはリバースプロキシー/ロードバランサーコンポーネントが含まれています。これは、会社や組織が使用するネットワークへのアクセスの分離や統合を行います。Red Hat build of Keycloak の実稼働環境では、このコンポーネントが推奨されます。

Red Hat build of Keycloak でプロキシー通信モードを設定する方法について、詳細は リバースプロキシーの使用 を参照してください。Red Hat build of Keycloak がアプリケーションを保護するために、パブリックアクセスから隠すべきパスと、公開すべきパスに関する推奨も記載されています。

2.4. キューに入れる要求の数の制限

実稼働環境では、過負荷状態から保護して、可能な限り多くの有効な要求に応答し、状況が正常に戻ったときに通常の操作を継続できるようにする必要があります。これを実現する 1 つの方法は、特定のしきい値に達したときに追加の要求を拒否することです。

負荷制限は、環境内のロードバランサーを含むすべてのレベルで実装する必要があります。さらに、Red Hat build of Keycloak には、すぐに処理できないためキューに入れる必要がある要求の数を制限する機能があります。デフォルトでは制限は設定されていません。オプション **http-max-queued-requests** を設定すると、キューに入れる要求の数を、環境に合わせて特定のしきい値に制限できます。この制限を超える要求には、即時に **503 Server not Available** 応答が返されます。

2.5. 実稼働グレードのデータベース

Red Hat build of Keycloak で使用されるデータベースは、Red Hat build of Keycloak の全体的なパフォーマンス、可用性、信頼性、整合性において非常に重要です。サポート対象データベースの設定方法について、詳細は データベースの設定 を参照してください。

2.6. クラスター内での RED HAT BUILD OF KEYCLOAK のサポート

Red Hat build of Keycloak インスタンスがダウンした場合もユーザーのログインを継続するために、一般的な実稼働環境には Red Hat build of Keycloak インスタンスが 2 つ以上含まれています。

Red Hat build of Keycloak は、 JGroups および Infinispan 上で実行されます。これらは、クラスター化されている場合に高い信頼性と可用性を提供します。クラスターにデプロイされている場合は、組み込み Infinispan サーバーの通信を保護する必要があります。この通信を保護するには、認証と暗号化を有効にするか、クラスター通信に使用されるネットワークを分離します。

複数のノード、さまざまなキャッシュ、環境に適したスタックを使用する場合の詳細は、分散キャッシュの設定 を参照してください。

2.7. RED HAT BUILD OF KEYCLOAK サーバーで IPV4 または IPV6 を設定する

システムプロパティーである **java.net.preferIPv4Stack** と **java.net.preferIPv6Addresses** を使用して、IPv4 または IPv6 アドレスを使用するように JVM を設定できます。

デフォルトでは、Red Hat build of Keycloak は、同時に IPv4 アドレスと IPv6 アドレスを介してアクセスできます。IPv4 アドレスのみで実行する場合は、プロパティー **java.net.preferIPv4Stack=true** を指定する必要があります。そうすることで、ホスト名から IP アドレスへの変換では必ず IPv4 アドレスのバリアントが返されるようになります。

これらのシステムプロパティーは、**JAVA_OPTS_APPEND** 環境変数を使用して容易に設定できます。たとえば、IP スタック設定を IPv4 に変更するには、次のように環境変数を設定します。

```
export JAVA_OPTS_APPEND="-Djava.net.preferIPv4Stack=true"
```

第3章 RED HAT BUILD OF KEYCLOAK をコンテナー内で実行する

この章では、コンテナーを実行する際に最適なエクスペリエンスを実現するために、Red Hat build of Keycloak コンテナーイメージを最適化して実行する方法について説明します。

警告

この章は、OpenShift 環境で実行するイメージのビルドにのみ適用されます。このイメージは OpenShift 環境のみをサポートします。他の Kubernetes ディストリビューションで実行する場合はサポートされません。

3.1. カスタマイズおよび最適化されたコンテナーイメージの作成

デフォルトの Red Hat build of Keycloak コンテナーイメージは、すぐに設定および最適化できる状態で出荷されます。

Red Hat build of Keycloak コンテナーを最適に起動するには、コンテナーのビルド中に **build** ステップを実行してイメージをビルドします。この手順を実行することで、後に続くコンテナーイメージの各起動フェーズで時間を節約できます。

3.1.1. 最適化された Red Hat build of Keycloak Dockerfile を記述する

次の **Dockerfile** は、健全性およびメトリクスのエンドポイントとトークン交換機能を有効にし、PostgreSQL データベースを使用する、事前設定済みの Red Hat build of Keycloak イメージを作成します。

Dockerfile:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24 as builder

# Enable health and metrics support
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true

# Configure a database vendor
ENV KC_DB=postgres

WORKDIR /opt/keycloak
# for demonstration purposes only, please make sure to use proper certificates in production instead
RUN keytool -genkeypair -storepass password -storetype PKCS12 -keyalg RSA -keysize 2048 -dname "CN=server" -alias server -ext "SAN:c=DNS:localhost,IP:127.0.0.1" -keystore conf/server.keystore
RUN /opt/keycloak/bin/kc.sh build

FROM registry.redhat.io/rhbk/keycloak-rhel9:24
COPY --from=builder /opt/keycloak/ /opt/keycloak/

# change these values to point to a running postgres instance
```

```
ENV KC_DB=postgres
ENV KC_DB_URL=<DBURL>
ENV KC_DB_USERNAME=<DBUSERNAME>
ENV KC_DB_PASSWORD=<DBPASSWORD>
ENV KC_HOSTNAME=localhost
ENTRYPOINT ["/opt/keycloak/bin/kc.sh"]
```

ビルドプロセスには複数の段階が含まれます。

- **build** コマンドを実行してサーバーのビルドオプションを設定し、最適化されたイメージを作成します。
- **build** 段階で生成されたファイルが、新しいイメージにコピーされます。
- 最後のイメージで、ホスト名とデータベースの追加の設定オプションが設定されているため、コンテナーの実行時にそれらを再度設定する必要はありません。
- エントリーポイントで、**kc.sh** により、すべてのディストリビューションのサブコマンドがアクセス可能になります。

カスタムプロバイダーは、JAR ファイルを **/opt/keycloak/providers** ディレクトリーに含めるステップを定義するだけでインストールできます。このステップは、以下のように、**build** コマンドを **RUNs** 行の前に配置する必要があります。

```
# A example build step that downloads a JAR file from a URL and adds it to the providers directory
FROM registry.redhat.io/rhbk/keycloak-rhel9:24 as builder

...

# Add the provider JAR file to the providers directory
ADD --chown=keycloak:keycloak --chmod=644 <MY_PROVIDER_JAR_URL>
/opt/keycloak/providers/myprovider.jar

...

# Context: RUN the build command
RUN /opt/keycloak/bin/kc.sh build
```

3.1.2. 追加の RPM パッケージをインストールする

FROM registry.redhat.io/rhbk/keycloak-rhel9 段階で新しいソフトウェアをインストールしようとすると、**microdnf**、**dnf**、さらには **rpm** がインストールされていないことがわかります。また、利用できるパッケージは非常に少なく、**bash** シェルと Red Hat build of Keycloak 自体の実行に必要なものしかありません。これは、Red Hat build of Keycloak の攻撃対象領域を減らすセキュリティー強化対策によるものです。

まず、ユースケースを別の方法で実装できるかどうかを検討し、なるべく最終的なコンテナーへの新規 RPM のインストールを回避します。

- Dockerfile 内の **RUN curl** 命令は、リモート URL をネイティブにサポートしているため、**ADD** に置き換えることができます。
- 一部の一般的な CLI ツールは、Linux ファイルシステムを創造的に使用することで置き換えることができます。たとえば、**ip addr show tap0** は **cat/sys/class/net/tap0/address** になります。

- RPM を必要とするタスクはイメージビルドの前の段階に移動し、代わりに結果をコピーできます。

以下は例です。前のビルド段階で **update-ca-trust** を実行し、後の段階に結果をコピーします。

```
FROM registry.access.redhat.com/ubi9 AS ubi-micro-build
COPY mycertificate.crt /etc/pki/ca-trust/source/anchors/mycertificate.crt
RUN update-ca-trust

FROM registry.redhat.io/rhbk/keycloak-rhel9
COPY --from=ubi-micro-build /etc/pki /etc/pki
```

絶対に必要な場合は、ubi-micro で確立された次の 2 段階のパターンに従い、新しい RPM をインストールできます。

```
FROM registry.access.redhat.com/ubi9 AS ubi-micro-build
RUN mkdir -p /mnt/rootfs
RUN dnf install --installroot /mnt/rootfs <package names go here> --releasever 9 --setopt
install_weak_deps=false --nodocs -y && \
    dnf --installroot /mnt/rootfs clean all && \
    rpm --root /mnt/rootfs -e --nodeps setup

FROM registry.redhat.io/rhbk/keycloak-rhel9
COPY --from=ubi-micro-build /mnt/rootfs /
```

このアプローチでは chroot (**/mnt/rootfs)** を使用するため、指定したパッケージとその依存関係のみがインストールされます。その結果、推測を必要とせずに第 2 段階に簡単にコピーできます。

警告

一部のパッケージには、依存関係の大きなツリーがあります。新しい RPM をインストールすると、コンテナーの攻撃対象領域が意図せず増大する可能性があります。インストールされているパッケージのリストを慎重に確認してください。

3.1.3. コンテナーイメージの構築

実際の container イメージをビルドするには、Dockerfile を含むディレクトリーから次のコマンドを実行します。

```
podman build . -t mykeycloak
```

3.1.4. 最適化された Red Hat build of Keycloak コンテナーイメージの起動

イメージを起動するには、以下を実行します。

```
podman run --name mykeycloak -p 8443:8443 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      mykeycloak \
      start --optimized
```

Red Hat build of Keycloak は、セキュアな HTTPS 通信のみを使用して実稼働モードで開始され、**https://localhost:8443** で使用できます。

ヘルスチェックエンドポイントは、、**https://localhost:8443/health**、**https://localhost:8443/health/ready**、および **https://localhost:8443/health/live** で使用できます。

https://localhost:8443/metrics を開くと、モニタリングソリューションで使用できる運用メトリクスを含むページが表示されます。

3.2. コンテナーを別のポートに公開する

デフォルトで、サーバーはポート **8080** と **8443** を使用して、それぞれ **http** 要求と **https** 要求をリッスンします。

別のポートを使用してコンテナーを公開する場合は、それに応じて **hostname-port** を設定する必要があります。

1. デフォルトポート以外のポートを使用してコンテナーを公開する

```
podman run --name mykeycloak -p 3000:8443 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      mykeycloak \
      start --optimized --hostname-port=3000
```

hostname-port オプションを設定すると、**https://localhost:3000** のサーバーにアクセスできるようになります。

3.3. 開発モードで RED HAT BUILD OF KEYCLOAK を試用する

開発またはテスト目的でコンテナーから Red Hat build of Keycloak を試用する場合、開発モードが最適です。**start-dev** コマンドを使用します。

```
podman run --name mykeycloak -p 8080:8080 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      registry.redhat.io/rhbk/keycloak-rhel9:24 \
      start-dev
```

このコマンドを呼び出すと、Red Hat build of Keycloak サーバーが開発モードで起動します。

このモードはデフォルトがセキュアではないため、実稼働環境では絶対に使用しないでください。Red Hat build of Keycloak を実稼働環境で実行する方法の詳細は、Red Hat build of Keycloak を実稼働用に設定する を参照してください。

3.4. 標準の RED HAT BUILD OF KEYCLOAK コンテナーを実行する

イミュータブルインフラストラクチャーなどの概念に従い、コンテナーは定期的に再プロビジョニングする必要があります。これらの環境では、素早く起動するコンテナーが必要なため、前のセクションで説明したように、最適化されたイメージを作成する必要があります。ただし、環境に異なる要件がある場合は、**start** コマンドを実行するだけで標準の Red Hat build of Keycloak イメージを実行できます。以下に例を示します。

```
podman run --name mykeycloak -p 8080:8080 \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
```

```
    registry.redhat.io/rhbk/keycloak-rhel9:24 \
    start \
    --db=postgres --features=token-exchange \
    --db-url=<JDBC-URL> --db-username=<DB-USER> --db-password=<DB-PASSWORD> \
    --https-key-store-file=<file> --https-key-store-password=<password>
```

このコマンドを実行すると、最初にビルドオプションを検出して適用する Red Hat build of Keycloak サーバーが起動します。この例では、**--db=postgres --features=token-exchange** の行でデータベースベンダーが PostgreSQL に設定され、トークン交換機能が有効になります。

その後、Red Hat build of Keycloak が起動し、設定が環境に適用されます。このアプローチでは起動時間が大幅に増加し、ミュータブルなイメージが作成されますが、これはベストプラクティスではありません。

3.5. コンテナー内での実行時に初期の管理者認証情報を入力する

Red Hat build of Keycloak では、ローカルネットワーク接続からのみ初期管理ユーザーを作成できます。コンテナー内で実行する場合、これは当てはまりません。そのため、イメージ実行時に次の環境変数を指定する必要があります。

```
# setting the admin username
-e KEYCLOAK_ADMIN=<admin-user-name>

# setting the initial password
-e KEYCLOAK_ADMIN_PASSWORD=change_me
```

3.6. 起動時にレルムをインポートする

Red Hat build of Keycloak コンテナーには、ディレクトリー **/opt/keycloak/data/import** があります。ボリュームマウントまたはその他の手段でこのディレクトリーに 1 つ以上のインポートファイルを配置し、起動引数 **--import-realm** を追加すると、Red Hat build of Keycloak コンテナーが起動時にそのデータをインポートします。これは、開発モードでのみの使用が合理的です。

```
podman run --name keycloak_unoptimized -p 8080:8080 \
    -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
    -v /path/to/realm/data:/opt/keycloak/data/import \
    registry.redhat.io/rhbk/keycloak-rhel9:24 \
    start-dev --import-realm
```

管理者ブートストラッププロセスの機能強化については、誰でも参加できる GitHub ディスカッションがあります。お気軽にご参加ください。

3.7. 異なるメモリー設定を指定する

Red Hat build of Keycloak コンテナーでは、初期ヒープサイズと最大ヒープサイズにハードコード値を指定せずに、コンテナーの合計メモリーに対する相対値を使用します。この動作は、JVM オプション **-XX:MaxRAMPercentage=70** および **-XX:InitialRAMPercentage=50** によって実現されます。

-XX:MaxRAMPercentage オプションは、最大ヒープサイズをコンテナーメモリーの合計の 70% として表します。**-XX:InitialRAMPercentage** オプションは、初期ヒープサイズをコンテナーメモリー全体の 50% として表します。これらの値は、Red Hat build of Keycloak メモリー管理の詳細な分析に基づいて選択されています。

ヒープサイズはコンテナーの合計メモリーに基づいて動的に計算されるため、コンテナーの **メモリー制限を必ず設定** してください。以前は、最大ヒープサイズは 512 MB に設定されていましたが、同じ値に近づけるには、メモリー制限を少なくとも 750 MB に設定する必要があります。小規模な実稼働環境対応のデプロイメントの場合、推奨されるメモリー制限は 2 GB です。

ヒープに関連する JVM オプションは、環境変数 **JAVA_OPTS_KC_HEAP** を設定することによってオーバーライドされる可能性があります。**JAVA_OPTS_KC_HEAP** のデフォルト値は、**kc.sh** または **kc.bat** スクリプトのソースコードにあります。

たとえば、環境変数とメモリー制限を次のように指定できます。

```
podman run --name mykeycloak -p 8080:8080 -m 1g \
      -e KEYCLOAK_ADMIN=admin -e KEYCLOAK_ADMIN_PASSWORD=change_me \
      -e JAVA_OPTS_KC_HEAP="-XX:MaxHeapFreeRatio=30 -XX:MaxRAMPercentage=65" \
      registry.redhat.io/rhbk/keycloak-rhel9:24 \
      start-dev
```

警告

メモリー制限が設定されていない場合、ヒープサイズがコンテナーの合計メモリーの最大 70% まで増加する可能性があるため、メモリー消費量が急激に増加します。JVM がメモリーを割り当てると、現在の Red Hat build of Keycloak の GC 設定により、そのメモリーが消極的に OS に返されます。

3.8. 関連するオプション

	値
db データベースベンダー CLI: **--db** Env: **KC_DB**	**dev-file** (デフォルト)、**dev-mem**、**mariadb**、**mssql**、**mysql**、**oracle**、**postgres**
db-password データベースユーザーアカウントのパスワード CLI: **--db-password** Env: **KC_DB_PASSWORD**	

	値
db-url データベースの完全な JDBC URL 指定しない場合、選択したデータベースベンダーに基づきデフォルト URL が設定されます。たとえば、**postgres** を使用している場合、デフォルトの JDBC URL は **jdbc:postgresql://localhost/keycloak** になります。 CLI: **--db-url** Env: **KC_DB_URL**	
db-username データベースユーザーのユーザー名 CLI: **--db-username** Env: **KC_DB_USERNAME**	
features ▮ 1 つ以上の機能セットを有効にします。 CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account2[:v1]**, **account3[:v1]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1]**, **admin2[:v1]**, **authorization[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v1]**, **impersonation[:v1]**, **js-adapter[:v1]**, **kerberos[:v1]**, **linkedin-oauth[:v1]**, **login2[:v1]**, **multi-site[:v1]**, **offline-session-preloading[:v1]**, **oid4vc-vci[:v1]**, **par[:v1]**, **preview**, **recovery-codes[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **web-authn[:v1]**

	値
health-enabled ▮ サーバーがヘルスチェックエンドポイントを公開する必要があるかどうか。 有効にすると、**/health**、**/health/ready**、および **/health/live** エンドポイントでヘルスチェックが利用可能になります。 CLI: **--health-enabled** Env: **KC_HEALTH_ENABLED**	**true**、**false** (デフォルト)
hostname Keycloak サーバーのホスト名 CLI: **--hostname** Env: **KC_HOSTNAME**	

	値
https-key-store-file 個別のファイルを指定する代わりに、証明書情報を保持するキーストア。 CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password キーストアファイルのパスワード CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**password** (デフォルト)
metrics-enabled サーバーがメトリクスを公開する必要があるかどうか。 有効にすると、**/metrics** エンドポイントでメトリクスを利用できるようになります。 CLI: **--metrics-enabled** Env: **KC_METRICS_ENABLED**	**true**、**false** (デフォルト)

第4章 TLS の設定

Transport Layer Security (略称: TLS) は、保護されたチャネル上でデータを交換するために重要です。実稼働環境では、Red Hat build of Keycloak が他のアプリケーションと交換する内容の中核に機密データが含まれるため、HTTP 経由で Red Hat build of Keycloak エンドポイントを公開しないでください。この章では、HTTPS/TLS を使用するように Red Hat build of Keycloak を設定する方法を説明します。

4.1. RED HAT BUILD OF KEYCLOAK で TLS を設定する

Red Hat build of Keycloak は、PEM 形式のファイルを使用するか、Java Keystore から、必要な証明書インフラストラクチャーをロードするように設定できます。両方の手段が設定されている場合、PEM ファイルが Java Keystore よりも優先されます。

4.1.1. PEM 形式の証明書を指定する

PEM 形式の証明書ファイルと秘密鍵ファイルのペアを使用する場合は、次のコマンドを実行して、それらを使用するように Red Hat build of Keycloak を設定します。

```
bin/kc.[sh|bat] start --https-certificate-file=/path/to/certfile.pem --https-certificate-key-file=/path/to/keyfile.pem
```

Red Hat build of Keycloak は、メモリー内のこれらのファイルからキーストアを作成し、以降はこのキーストアを使用します。

4.1.2. Java Keystore を指定する

キーストアファイルが明示的に設定されておらず、**http-enabled** が false に設定されている場合、Red Hat build of Keycloak は **conf/server.keystore** ファイルを探します。

代わりに、次のコマンドを実行して既存のキーストアを使用することもできます。

```
bin/kc.[sh|bat] start --https-key-store-file=/path/to/existing-keystore-file
```

4.1.2.1. Keystore のパスワードを設定する

https-key-store-password オプションを使用して、キーストアにセキュアなパスワードを設定できます。

```
bin/kc.[sh|bat] start --https-key-store-password=<value>
```

パスワードが設定されていない場合は、デフォルトのパスワードである **password** が使用されます。

4.2. TLS プロトコルの設定

デフォルトでは、Red Hat build of Keycloak は非推奨の TLS プロトコルを有効にしません。クライアントが非推奨のプロトコルのみをサポートしている場合は、クライアントのアップグレードを検討してください。一時的な回避策として、次のコマンドを実行し、非推奨のプロトコルを有効にできます。

```
bin/kc.[sh|bat] start --https-protocols=<protocol>[,<protocol>]
```

TLSv1.2 も許可するには、**kc.sh start --https-protocols=TLSv1.3,TLSv1.2** などのコマンドを使用します。

4.3. HTTPS ポートの切り替え

Red Hat build of Keycloak は、ポート **8443** で HTTPS トラフィックをリッスンします。このポートを変更するには、次のコマンドを使用します。

```
bin/kc.[sh|bat] start --https-port=<port>
```

4.4. トラストストアの使用

クライアント証明書を適切に検証し、TLS 相互認証 (mTLS) などの特定の認証方法を有効にするために、サーバーが信頼する必要があるすべての証明書 (および証明書チェーン) を含むトラストストアを設定できます。多くの機能は、このトラストストアに依存して、次のような証明書でクライアントを適切に認証します。

- 相互 TLS クライアント認証
- エンドユーザー X.509 ブラウザー認証

次のコマンドを実行して、このトラストストアの場所を設定できます。

```
bin/kc.[sh|bat] start --https-trust-store-file=/path/to/file
```

注記

これは、Red Hat build of Keycloak がサーバーとして機能するクライアントを認証するためのトラストストアです。Red Hat build of Keycloak が TLS を介して外部サービスへのクライアントとして機能するトラストストアの設定については、信頼済み証明書の設定を参照してください。

4.4.1. トラストストアのパスワードを設定する

https-trust-store-password オプションを使用して、トラストストアのセキュアなパスワードを設定できます。

```
bin/kc.[sh|bat] start --https-trust-store-password=<value>
```

パスワードが設定されていない場合は、デフォルトのパスワードである **password** が使用されます。

4.5. 認証情報の保護

CLI を使用するか、**conf/keycloak.conf** ファイルにパスワードを追加するなどして、プレーンテキストでのパスワード設定を回避してください。代わりに、vault/マウントされたシークレットを使用するなどの適切な方法を使用してください。詳細は、vault の使用 および Red Hat build of Keycloak を実稼働用に設定する を参照してください。

4.6. 相互 TLS の有効化

mTLS を使用した認証は、デフォルトで無効になっています。Red Hat build of Keycloak がサーバーであり、Red Hat build of Keycloak エンドポイントに対する要求からの証明書を検証する必要がある場合に mTLS 証明書の処理を有効にするには、Red Hat build of Keycloak トラストストアに適切な証明書を配置し、次のコマンドを使用して mTLS を有効にします。

```
bin/kc.[sh|bat] start --https-client-auth=<none|request|required>
```

値に **required** を使用すると、必ず Red Hat build of Keycloak が証明書を要求し、要求内に証明書がない場合は失敗するように設定されます。値を **request** に設定すると、Red Hat build of Keycloak は証明書がない要求も受け入れ、証明書が存在する場合にのみ証明書の正確性を検証します。

これは、Red Hat build of Keycloak がサーバーとして機能する mTLS ユースケースの、基本的な証明書設定であることに注意してください。Red Hat build of Keycloak がクライアントとして機能する場合 (たとえば Red Hat build of Keycloak が mTLS によって保護されているブローカーアイデンティティープロバイダーのトークンエンドポイントからトークンを取得しようとする場合)、キーストア内の適切な証明書を送信要求に提供するように、HttpClient を設定する必要があります。このようなシナリオで mTLS を設定するには、送信 HTTP 要求の設定 を参照してください。

4.7. 関連するオプション

	値
http-enabled HTTP リスナーを有効にします。 CLI: **--http-enabled** Env: **KC_HTTP_ENABLED**	**true**、**false** (デフォルト)
https-certificate-file PEM 形式のサーバー証明書または証明書チェーンへのファイルパス。 CLI: **--https-certificate-file** Env: **KC_HTTPS_CERTIFICATE_FILE**	
https-certificate-key-file PEM 形式の秘密鍵へのファイルパス。 CLI: **--https-certificate-key-file** Env: **KC_HTTPS_CERTIFICATE_KEY_FILE**	
https-cipher-suites 使用する暗号スイート。 指定されていない場合、適切なデフォルトが選択されます。 CLI: **--https-cipher-suites** Env: **KC_HTTPS_CIPHER_SUITES**	
https-client-auth クライアント認証を必要とする、または要求するようにサーバーを設定します。 CLI: **--https-client-auth** Env: **KC_HTTPS_CLIENT_AUTH**	**none** (デフォルト)、**request**、**required**

	値
https-key-store-file 個別のファイルを指定する代わりに、証明書情報を保持するキーストア。 CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password キーストアファイルのパスワード CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**password** (デフォルト)
https-key-store-type キーストアファイルの型。 指定されていない場合、ファイル名に基き自動的に型が検出されます。**fips-mode** が **strict** に設定され、値が設定されていない場合、デフォルトの **BCFKS** になります。 CLI: **--https-key-store-type** Env: **KC_HTTPS_KEY_STORE_TYPE**	
https-port 使用される HTTPS ポート。 CLI: **--https-port** Env: **KC_HTTPS_PORT**	**8443** (デフォルト)
https-protocols 明示的に有効にするプロトコルのリスト。 CLI: **--https-protocols** Env: **KC_HTTPS_PROTOCOLS**	**[TLSv1.3, TLSv1.2]** (デフォルト)
https-trust-store-file 信頼する証明書の証明書情報を保持するトラストストア。 CLI: **--https-trust-store-file** Env: **KC_HTTPS_TRUST_STORE_FILE** **非推奨。**代わりにシステムトラストストアを使用してください。詳細はドキュメントを参照してください。	

	値
https-trust-store-password トラストストアファイルのパスワード。 CLI: **--https-trust-store-password** Env: **KC_HTTPS_TRUST_STORE_PASSWORD** **非推奨。**代わりにシステムトラストストアを使用してください。詳細はドキュメントを参照してください。	
https-trust-store-type トラストストアファイルの型。 指定されていない場合、ファイル名に基き自動的に型が検出されます。**fips-mode** が **strict** に設定され、値が設定されていない場合、デフォルトの **BCFKS** になります。 CLI: **--https-trust-store-type** Env: **KC_HTTPS_TRUST_STORE_TYPE** **非推奨。**代わりにシステムトラストストアを使用してください。詳細はドキュメントを参照してください。	

第5章 ホスト名の設定

5.1. サーバーエンドポイント

Red Hat build of Keycloak は、アプリケーションと対話したり、管理コンソールへのアクセスを許可したりするための、さまざまなエンドポイントを公開します。これらのエンドポイントは、次の 3 つの主要なグループに分類できます。

- フロントエンド
- バックエンド
- 管理コンソール

各グループのベース URL は、トークンの発行方法と検証方法、ユーザーを Red Hat build of Keycloak にリダイレクトする必要があるアクション (メールリンクを通じてパスワードをリセットする場合など) のリンクの作成方法、さらにはアプリケーションが **realms/{realm-name}/.well-known/openid-configuration** から OpenID Connect Discovery Document を取得する際にこれらのエンドポイントを検出する方法に重要な影響を与えます。

5.1.1. フロントエンド

フロントエンドのエンドポイントは、パブリックドメインを通じてアクセスでき、通常はフロントチャネルを通じて発生する認証/認可フローに関連します。たとえば、SPA がユーザーを認証する必要がある場合、SPA はユーザーを **authorization_endpoint** にリダイレクトし、ユーザーがフロントチャネル経由でブラウザーを使用して認証できるようにします。

デフォルトでは、ホスト名設定が行われていない場合、これらのエンドポイントのベース URL は受信要要求に基づくため、HTTP スキーム、ホスト、ポート、パスは要求と同じになります。発行者もフロントエンドエンドポイントに設定された URL に基づくため、デフォルトの動作は、サーバーがトークンを発行する方法にも直接影響します。ホスト名設定が行われていない場合、トークン発行者も受信要求に基づくことになり、クライアントが異なる URL を使用してトークンを要求している場合は一貫性が失われます。

通常、実稼働環境にデプロイする場合、要求の構築方法に関係なく、フロントエンドエンドポイントとトークン発行者の一貫した URL が必要になります。この一貫性を実現するために、**hostname** または **hostname-url** オプションを設定できます。

ほとんどの場合、フロントエンド URL の **ホスト** のみを変更するには、**hostname** オプションを設定するだけで十分です。

```
bin/kc.[sh|bat] start --hostname=<host>
```

hostname オプションを使用している場合、以下を達成するために、サーバーは HTTP スキーム、ポート、およびパスを自動的に解決します。

- **hostname-strict-https=false** を設定しない限り、**https** スキームが使用されます。
- **proxy-headers** オプションが設定されている場合、プロキシーがデフォルトのポート (80 と 443) を使用します。プロキシーで別のポートを使用する場合は、**hostname-url** 設定オプションで指定する必要があります。

ただし、ホストだけでなくスキーム、ポート、パスも設定する必要がある場合は、**hostname-url** オプションを設定できます。

```
bin/kc.[sh|bat] start --hostname-url=<scheme>://<host>:<port>/<path>
```

このオプションを使用すると、1つのオプションで URL のさまざまな部分を設定できるため、柔軟性が向上します。**hostname** と **hostname-url** は、相互に排他的であることに注意してください。

注記

hostname と **proxy-headers** の設定オプションは、静的リソースの URL、リダイレクト URI、OIDC well-known エンドポイントなどにのみ影響します。サーバーが実際にどこでどのポートをリッスンするかを変更するには、**http/tls** 設定オプション (例: **http-host**、**https-port** など) を使用する必要があります。詳細は、TLS の設定 および すべての設定 を参照してください。

5.1.2. バックエンド

バックエンドエンドポイントは、パブリックドメインまたはプライベートネットワークを通じてアクセスできるエンドポイントです。これらは、サーバーとクライアント間の、仲介のないプレーン HTTP 要求による直接通信に使用されます。たとえば、ユーザーが認証された後、SPA はトークン要求を **token_endpoint** に送信することで、サーバーが送信した **code** をトークンセットと交換する必要があるとします。

デフォルトでは、バックエンドエンドポイントの URL も受信要求に基づいています。この動作をオーバーライドするには、次のコマンドを入力して **hostname-strict-backchannel** 設定オプションを設定します。

```
bin/kc.[sh|bat] start --hostname=<value> --hostname-strict-backchannel=true
```

hostname-strict-backchannel オプションを設定すると、バックエンドエンドポイントの URL はフロントエンドエンドポイントとまったく同じになります。

Red Hat build of Keycloak に接続されているすべてのアプリケーションがパブリック URL 経由で通信する場合は、**hostname-strict-backchannel** を **true** に設定します。それ以外の場合は、このパラメーターを **false** のままにして、プライベートネットワークを介したクライアント/サーバー通信を許可します。

5.1.3. 管理コンソール

サーバーは、特定の URL を使用して管理コンソールと静的リソースを公開します。

デフォルトでは、管理コンソールの URL も受信要求に基づいています。ただし、特定の URL を使用して管理コンソールへのアクセスを制限する場合は、特定のホストまたはベース URL を設定できます。フロントエンド URL の設定方法と同様に、**hostname-admin** および **hostname-admin-url** オプションを使用してこれを実現できます。HTTPS が有効になっている場合 (実稼働モードのデフォルト設定として **http-enabled** 設定オプションは false に設定されています)、Red Hat build of Keycloak サーバーは HTTPS URL の使用を自動的に想定することに注意してください。次に、管理コンソールは HTTPS 経由で Red Hat build of Keycloak に接続しようとします。HTTPS URL は、設定されたリダイレクト/Web オリジン URL にも使用されます。実稼働環境には推奨されませんが、HTTP URL を **hostname-admin-url** として使用して、この動作をオーバーライドできます。

ほとんどの場合、管理コンソール URL の **ホスト** のみを変更するのであれば、**hostname-admin** オプションを設定するだけで十分です。

```
bin/kc.[sh|bat] start --hostname-admin=<host>
```

ただし、ホストだけでなくスキーム、ポート、パスも設定する必要がある場合は、**hostname-admin-url** オプションを設定できます。

```
bin/kc.[sh|bat] start --hostname-admin-url=<scheme>://<host>:<port>/<path>
```

hostname-admin と **hostname-admin-url は** 相互に排他的であることに注意してください。

攻撃対象領域を減らすために、Red Hat build of Keycloak の管理エンドポイントと管理コンソールは、パブリックにアクセスできないようにする必要があります。つまり、リバースプロキシーを使用して保護できます。リバースプロキシーを使用して公開するパスの詳細は、リバースプロキシーの使用 を参照してください。

5.2. シナリオ例

以下に、ホスト名を設定する別のサンプルシナリオと、対応するコマンドを示します。

start コマンドには TLS の設定が必要であることに注意してください。対応するオプションは例として示されていません。詳細は、TLS の設定 を参照してください。

5.2.1. TLS Termination プロキシーの背後にあるサーバーを公開する

この例では、サーバーは TLS Termination プロキシーの背後で実行されており、**https://mykeycloak** から公開されています。

設定:

```
bin/kc.[sh|bat] start --hostname=mykeycloak --http-enabled=true --proxy-
headers=forwarded|xforwarded
```

5.2.2. プロキシーを使用せずにサーバーを公開する

この例では、サーバーはプロキシーなしで実行されており、HTTPS を使用して URL で公開されています。

Red Hat build of Keycloak の設定:

```
bin/kc.[sh|bat] start --hostname-url=https://mykeycloak
```

セキュリティーと可用性の観点から、サーバーの前で TLS Termination プロキシーを使用することが強く推奨されます。詳細は、リバースプロキシーの使用 を参照してください。

5.2.3. バックエンドのエンドポイントに、サーバーが公開されているのと同じ URL を使用するように強制する

この例では、サーバーが使用するのと同じ URL を使用してバックエンドエンドポイントが公開されているため、クライアントは要求の送信元に関係なく常に同じ URL を取得します。

Red Hat build of Keycloak の設定:

```
bin/kc.[sh|bat] start --hostname=mykeycloak --hostname-strict-backchannel=true
```

5.2.4. デフォルトポート以外のポートを使用してサーバーを公開する

この例では、デフォルトポート以外のポートを使用してサーバーにアクセスできます。

Red Hat build of Keycloak の設定:

```
bin/kc.[sh|bat] start --hostname-url=https://mykeycloak:8989
```

5.2.5. 異なるポートを使用して、TLS 再暗号化プロキシーの背後で Red Hat build of Keycloak を公開する

この例では、サーバーはプロキシーの背後で実行されており、サーバーとプロキシーの両方が独自の証明書を使用しているため、Red Hat build of Keycloak とプロキシー間の通信は暗号化されます。リバースプロキシーは **Forwarded** ヘッダーを使用し、**X-Forwarded-*** ヘッダーを設定しません。プロキシー設定オプション (およびホスト名設定オプション) は、サーバーが実際にリッスンしているポートを変更するものではないことに留意する必要があります (JavaScript および CSS リンク、OIDC well-known エンドポイント、リダイレクト URI など、静的リソースのポートのみが変更されます)。したがって、HTTP 設定オプションを使用して、別のポート (例: 8543) を内部でリッスンするように Red Hat build of Keycloak サーバーを変更する必要があります。プロキシーはポート 8443 (ブラウザー経由でコンソールにアクセスしている際に表示されるポート) をリッスンします。サンプルのホスト名 **my-keycloak.org** がサーバーに使用され、同様に管理コンソールには **admin.my-keycloak.org** サブドメイン経由でアクセスできます。

Red Hat build of Keycloak の設定:

```
bin/kc.[sh|bat] start --proxy-headers=forwarded --https-port=8543 --hostname-url=https://my-keycloak.org:8443 --hostname-admin-url=https://admin.my-keycloak.org:8443
```

警告

proxy-headers オプションの使用は、それぞれ **Forwarded** ヘッダーと **X-Forwarded-*** ヘッダーに依存します。これらのヘッダーは、リバースプロキシーによって設定および上書きされる必要があります。設定を誤ると、Red Hat build of Keycloak がセキュリティー上の問題にさらされる可能性があります。詳細は、リバースプロキシーの使用 を参照してください。

5.3. トラブルシューティング

ホスト名の設定に対してトラブルシューティングを行うには、次のように有効にできる専用のデバッグツールを使用します。

Red Hat build of Keycloak の設定:

```
bin/kc.[sh|bat] start --hostname=mykeycloak --hostname-debug=true
```

Red Hat build of Keycloak が適切に起動したら、ブラウザーを開いて次の場所に移動します。

http://mykeycloak:8080/realms/<your-realm>/hostname-debug

5.4. 関連するオプション

表5.1 デフォルトでは、このエンドポイントは無効になっています (--hostname-debug=false)

	値
hostname Keycloak サーバーのホスト名 CLI: **--hostname** Env: **KC_HOSTNAME**	
hostname-admin 管理コンソールにアクセスするためのホスト名。 **hostname** オプションに設定された値以外のホスト名を使用して管理コンソールを公開する場合は、このオプションを使用します。 CLI: **--hostname-admin** Env: **KC_HOSTNAME_ADMIN**	
hostname-admin-url 管理コンソールにアクセスするためのベース URL (スキーム、ホスト、ポート、パスなどを含む) を設定します。 CLI: **--hostname-admin-url** Env: **KC_HOSTNAME_ADMIN_URL**	
hostname-debug /realms/master/hostname-debug でアクセスできるホスト名のデバッグページを切り替えます。 CLI: **--hostname-debug** Env: **KC_HOSTNAME_DEBUG**	**true**、**false** (デフォルト)
hostname-path プロキシーが Keycloak のために異なるコンテキストパスを使用する場合は、これを設定する必要があります。 CLI: **--hostname-path** Env: **KC_HOSTNAME_PATH**	
hostname-port ホスト名を公開する際に、プロキシーが使用するポート。 プロキシーが、デフォルトの HTTP および HTTPS ポート以外のポートを使用する場合は、このオプションを設定します。 CLI: **--hostname-port** Env: **KC_HOSTNAME_PORT**	**-1** (デフォルト)

	値
hostname-strict 要求ヘッダーからのホスト名の動的解決を無効にします。 プロキシーが Host ヘッダーを検証しない限り、実稼働環境では常に true に設定する必要があります。 CLI: **--hostname-strict** Env: **KC_HOSTNAME_STRICT**	**true** (デフォルト)、**false**
hostname-strict-backchannel デフォルトでは、バックチャネル URL は要求ヘッダーから動的に解決され、内部および外部アプリケーションが許可されます。 すべてのアプリケーションがパブリック URL を使用する場合、このオプションを有効にする必要があります。 CLI: **--hostname-strict-backchannel** Env: **KC_HOSTNAME_STRICT_BACKCHANNEL**	**true**、**false** (デフォルト)
hostname-url フロントエンド URL のベース URL (スキーム、ホスト、ポート、パスなどを含む) を設定します。 CLI: **--hostname-url** Env: **KC_HOSTNAME_URL**	
proxy サーバーがリバースプロキシーの背後にある場合のプロキシーアドレス転送モード。 CLI: **--proxy** Env: **KC_PROXY** **非推奨。proxy-headers** を使用してください。	**none** (デフォルト)、**edge**、**reencrypt**、**passthrough**

第6章 リバースプロキシーの使用

分散環境では、頻繁にリバースプロキシーの使用が必要になります。Red Hat build of Keycloak は、このような環境とのセキュアな統合を実現するためのオプションをいくつか備えています。

6.1. リバースプロキシーヘッダーの設定

Red Hat build of Keycloak は、**proxy-headers** オプションに基づいてリバースプロキシーヘッダーを解析します。このオプションは、次のいくつかの値を受け入れます。

- デフォルトでは、オプションが指定されていない場合、リバースプロキシーヘッダーは解析されません。
- **forwarded** は、RFC7239 に従って **Forwarded** ヘッダーの解析を有効にします。
- **xforwarded** は、**X-Forwarded-For**、**X-Forwarded-Proto**、**X-Forwarded-Host**、**X-Forwarded-Port** などの非標準の **X-Forwarded-*** ヘッダーの解析を有効にします。

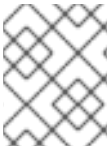
以下に例を示します。

```
bin/kc.[sh|bat] start --proxy-headers forwarded
```

警告

forwarded または **xforwarded** のいずれかを選択した場合は、リバースプロキシーによって **Forwarded** または **X-Forwarded-*** ヘッダーが適切に設定して上書きされることを確認してください。これらのヘッダーを設定するには、リバースプロキシーのドキュメントを参照してください。設定を誤ると、Red Hat build of Keycloak がセキュリティー上の脆弱性にさらされることになります。

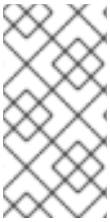
クライアントアドレスが、リバースプロキシーにより **Forwarded** ヘッダーまたは **X-Forwarded-For** ヘッダーを介して適切に設定されていることを、特に注意して確認してください。このヘッダーが正しく設定されていない場合、不正なクライアントがこのヘッダーを設定し、クライアントが実際のアドレスとは異なる IP アドレスから接続しているという誤った認識を Red Hat build of Keycloak が持つ可能性があります。IP アドレスの拒否リストまたは許可リストを作成する場合、このような注意はさらに重要です。



注記

xforwarded 設定を使用する場合、**X-Forwarded-Port** は **X-Forwarded-Host** に含まれるポートよりも優先されます。

6.2. プロキシーモード



注記

プロキシーモードの設定のサポートは非推奨であり、Red Hat build of Keycloak の今後のリリースで削除される予定です。代わりに、上記の章の説明に従って、受け入れられるリバースプロキシーヘッダーを設定することを検討してください。移行手順については、アップグレードガイド を参照してください。

Red Hat build of Keycloak の場合、プロキシーモードの選択は環境の TLS Termination により異なります。次のプロキシーモードを使用できます。

edge

プロキシーと Red Hat build of Keycloak 間で、HTTP 経由の通信を有効にします。このモードは、HTTP を使用して Red Hat build of Keycloak と通信する間、リバースプロキシーがクライアントとのセキュアな接続 (HTTP over TLS) を維持する、安全性の高い内部ネットワークを使用するデプロイメントに適しています。

reencrypt

プロキシーと Red Hat build of Keycloak の間で HTTPS を介した通信が必要です。このモードは、リバースプロキシーと Red Hat build of Keycloak 間の内部通信も保護する必要があるデプロイメントに適しています。リバースプロキシーと Red Hat build of Keycloak では、異なる鍵と証明書が使用されます。

passthrough

プロキシーは、TLS を終了せずに、HTTPS 接続を Red Hat build of Keycloak に転送します。サーバーとクライアント間のセキュアな接続は、Red Hat build of Keycloak サーバーで使用される鍵と証明書に基づいています。

edge または **reencrypt** プロキシーモードの場合、Red Hat build of Keycloak は次のヘッダーを解析し、それらがリバースプロキシーにより設定されることを想定します。

- RFC7239 に準じた **Forwarded**
- 非標準の **X-Forwarded-*** (**X-Forwarded-For**、**X-Forwarded-Proto**、**X-Forwarded-Host**、**X-Forwarded-Port** など)

6.2.1. Red Hat build of Keycloak でプロキシーモードを設定する

プロキシーモードを選択するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --proxy <mode>
```

6.3. リバースプロキシー上のさまざまなコンテキストパス

Red Hat build of Keycloak では、Red Hat build of Keycloak が設定されているのと同じコンテキストパスで、リバースプロキシーを通じて公開されることが想定されています。デフォルトでは、Red Hat build of Keycloak はルート (/) を通じて公開されます。これは、/ 上のリバースプロキシーを通じての公開が想定されていることを意味します。このような場合は、**hostname-path** または **hostname-url** を使用できます。たとえば、Red Hat build of Keycloak が **/auth** 上のリバースプロキシーを通じて公開されている場合は **--hostname-path=/auth** を使用します。

あるいは、**http-relative-path** オプションを使用して、Red Hat build of Keycloak 自体のコンテキストパスをリバースプロキシーのコンテキストパスに一致するように変更することもできます。これにより、リバースプロキシーが使用するコンテキストパスと一致するように、Red Hat build of Keycloak 自体のコンテキストパスが変更されます。

6.4. ホスト名を設定するためにプロキシーを信頼する

デフォルトでは、Red Hat build of Keycloak は、どのホスト名で呼び出されるかを認識する必要があります。リバースプロキシーが正しいホスト名をチェックするように設定されている場合は、任意のホスト名を受け入れるように Red Hat build of Keycloak を設定できます。

```
bin/kc.[sh|bat] start --proxy-headers=forwarded|xforwarded --hostname-strict=false
```

6.5. スティッキーセッションの有効化

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 (リバースプロキシー) および 2 つ以上の Red Hat build of Keycloak サーバーで構成されます。パフォーマンスの観点からは、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要求を同じ Red Hat build of Keycloak バックエンドノードに転送すると便利です。

なぜなら、Red Hat build of Keycloak 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裏で、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 2 つの所有者で設定されます。つまり、特定のセッションは 2 つのクラスターノードに保存され、そのセッションにアクセスする必要がある場合は、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ーを返すために、ネットワーク経由で要求を 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ー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 build of Keycloak 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。

- ユーザーは、Red Hat build of Keycloak のログイン画面を表示するために、最初の要求を送信します。
- 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 (逆引きプロキシー) によって異なります。
- Red Hat build of Keycloak 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ー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、このセッションの所有者として node2 を割り当てたとします。
- Red Hat build of Keycloak 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 build of Keycloak ログイン画面と cookie (AUTH_SESSION_ID) を持つユーザーに応答が返されます。

この観点から、ロードバランサーが次の要求をすべて node2 に転送することは有用です。なぜなら、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からです。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。そ

の場合も ID 123 と同じになるため、node2 に保存されます。

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 をスティッキーにするように、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。

プロキシーがバックエンドノードからの cookie を処理せずにセッションアフィニティーをサポートする場合は、ノードを cookie にアタッチせず、リバースプロキシー機能に依存するのみにするために、**spi-sticky-session-encoder-infinispan-should-attach-route** オプションを **false** に設定する必要があります。

```
bin/kc.[sh|bat] start --spi-sticky-session-encoder-infinispan-should-attach-route=false
```

デフォルトでは、**spi-sticky-session-encoder-infinispan-Should-attach-route** オプションの値は **true** であるため、ノード名は cookie にアタッチされ、リバースプロキシーには後続の要求の送信先であるノードが示されます。

6.5.1. 管理コンソールを公開する

デフォルトでは、管理コンソール URL は、適切なスキーム、ホスト名、およびポートを解決する要求に基づく場合にのみ作成されます。たとえば、**edge** プロキシーモードを使用しており、プロキシーが正しく設定されていない場合、TLS Termination プロキシーからのバックエンド要求ではプレーン HTTP が使用されます。その場合、URL は **http** スキームを使用して作成され、プロキシーがプレーン HTTP をサポートしないため、管理コンソールにアクセスできなくなる可能性があります。

管理コンソールを適切に公開するには、プロキシーにより公開されたスキーム、ホスト名、ポートを使用して URL を作成するために、ここで説明されている **X-Forwarded-*** ヘッダーをプロキシーが設定していることを確認する必要があります。

6.5.2. 公開されたパスに関する推奨事項

リバースプロキシーを使用する場合、Red Hat build of Keycloak では特定のパスのみ公開する必要があります。次の表は、公開が推奨されるパスを示しています。

Red Hat build of Keycloak のパス	リバースプロキシーパス	公開	理由
/	-	いいえ	すべてのパスを公開すると、管理パスが不必要に公開されます。
/admin/	-	いいえ	管理パスが公開されると、不要な攻撃ベクトルが発生します。
/js/	-	はい (以下の注記を参照)	"内部" クライアント (アカウントコンソールなど) に必要な keycloak.js へのアクセス
/welcome/	-	いいえ	初回インストール後に welcome ページを公開する必要はありません。

Red Hat build of Keycloak のパス	リバースプロキシーパス	公開	理由
/realms/	/realms/	はい	このパスは、たとえば OIDC エンドポイントなどで正しく機能するために必要です。
/resources/	/resources/	はい	このパスは、アセットを正しく提供するために必要です。Red Hat build of Keycloak のパスではなく、CDN から提供される場合があります。
/robots.txt	/robots.txt	はい	検索エンジンのルール
/metrics	-	いいえ	メトリクスが公開されると、不要な攻撃ベクトルが発生します。
/health	-	いいえ	ヘルスチェックが公開されると、不要な攻撃ベクトルが発生します。

注記

アカウントコンソールなどの内部クライアントには **js** パスが必要なため、外部クライアントには npm や yarn などの JavaScript パッケージマネージャーから **keycloak.js** を使用することが推奨されます。

Red Hat build of Keycloak をリバースプロキシー/ゲートウェイのパブリック API のルートパス / で実行していることを前提としています。そうでない場合は、パスの前に任意の接頭辞を追加します。

6.5.3. クライアント証明書ルックアップを有効にする

プロキシーが TLS Termination プロキシーとして設定されている場合、クライアント証明書情報は特定の HTTP 要求ヘッダーを通じてサーバーに転送され、クライアントの認証に使用されます。使用しているプロキシーに応じて、サーバーがクライアント証明書情報を取得する方法を設定できます。

サーバーは、次のような最も一般的な TLS Termination プロキシーのいくつかをサポートしています。

Proxy	Provider
Apache HTTP サーバー	apache
HAProxy	haproxy
NGINX	nginx

要求からクライアント証明書を取得する方法を設定するには、以下を実行する必要があります。

対応するプロキシープロバイダーを有効にする

```
bin/kc.[sh|bat] build --spi-x509cert-lookup-provider=<provider>
```

HTTP ヘッダーを設定する

```
bin/kc.[sh|bat] start --spi-x509cert-lookup-<provider>-ssl-client-cert=SSL_CLIENT_CERT --spi-x509cert-lookup-<provider>-ssl-cert-chain-prefix=CERT_CHAIN --spi-x509cert-lookup-<provider>-certificate-chain-length=10
```

HTTP ヘッダーを設定する際には、使用している値が、プロキシーによってクライアント証明書情報とともに転送されるヘッダーの名前に対応していることを確認する必要があります。

プロバイダーの設定に使用できるオプションは次のとおりです。

オプション	説明
ssl-client-cert	クライアント証明書を保持するヘッダーの名前
ssl-cert-chain-prefix	チェーン内の追加の証明書を保持するヘッダーの接頭辞。チェーンの長さに応じて個々の証明書を取得するために使用されます。たとえば **CERT_CHAIN** の値は、**certificate-chain-length** が **10** に設定されている場合に、ヘッダー **CERT_CHAIN_0** から **CERT_CHAIN_9** まで追加の証明書をロードするようにサーバーに指示します。
certificate-chain-length	証明書チェーンの最大長。
trust-proxy-verification	証明書を Red Hat build of Keycloak に転送して Red Hat build of Keycloak で検証するのではなく、信頼している NGINX プロキシー証明書の検証を有効にします。

6.5.3.1. NGINX プロバイダーを設定する

NGINX SSL/TLS モジュール は、クライアント証明書チェーンを公開しません。Red Hat build of Keycloak の NGINX 証明書ルックアッププロバイダーは、Red Hat build of Keycloak トラストストアを使用してそれを再構築します。

このプロバイダーを使用している場合は、Red Hat build of Keycloak トラストストアを設定する方法について、信頼済み証明書の設定 を参照してください。

6.6. 関連するオプション

	値
hostname-path プロキシーが Keycloak のために異なるコンテキストパスを使用する場合は、これを設定する必要があります。 CLI: **--hostname-path** Env: **KC_HOSTNAME_PATH**	
hostname-url フロントエンド URL のベース URL (スキーム、ホスト、ポート、パスなどを含む) を設定します。 CLI: **--hostname-url** Env: **KC_HOSTNAME_URL**	
http-relative-path ▮ リソースの提供に使用する、/ に相対するパスを設定します。 パスは / で始まる必要があります。 CLI: **--http-relative-path** Env: **KC_HTTP_RELATIVE_PATH**	/ (デフォルト)
proxy サーバーがリバースプロキシーの背後にある場合のプロキシーアドレス転送モード。 CLI: **--proxy** Env: **KC_PROXY** **非推奨。proxy-headers** を使用してください。	**none** (デフォルト)、**edge**、**reencrypt**、**passthrough**
proxy-headers サーバーが受け入れる必要があるプロキシーヘッダー。 設定を誤ると、サーバーがセキュリティー上の脆弱性にさらされる可能性があります。非推奨のプロキシーオプションよりも優先されます。 CLI: **--proxy-headers** Env: **KC_PROXY_HEADERS**	**forwarded**、**xforwarded**

第7章 データベースの設定

この章では、データをリレーショナルデータベースに保存するために、Red Hat build of Keycloak サーバーを設定する方法について説明します。

7.1. サポートされているデータベース

サーバーには、各種データベースのサポートが組み込まれています。**db** 設定オプションの期待値を表示することで、使用可能なデータベースをクエリーできます。次の表は、サポート対象のデータベースとそのテスト済みバージョンを示しています。

データベース	オプションの値	テスト済みバージョン
MariaDB Server	**mariadb**	10.11
Microsoft SQL Server	**mssql**	2022
MySQL	**mysql**	8.0
Oracle データベース	**oracle**	19.3
PostgreSQL	**postgres**	15
Amazon Aurora PostgreSQL	**postgres**	15.3

デフォルトでは、サーバーは **dev-file** データベースを使用します。これはサーバーがデータを保持するために使用するデフォルトのデータベースであり、開発ユースケースに限定されています。**dev-file** データベースは実稼働環境のユースケースには適していないため、実稼働環境にデプロイする前に置き換える必要があります。

7.2. データベースドライバーのインストール

データベースドライバーは、個別にインストールする必要がある Oracle Database ドライバーと Micrsoft SQL Server ドライバーを除き、Red Hat build of Keycloak に同梱されます。

これらのデータベースのいずれかに接続する場合は、必要なドライバーをインストールします。データベースドライバーがすでに含まれている別のデータベースに接続する場合は、このセクションをスキップしてください。

7.2.1. Oracle データベースドライバーをインストールする

Red Hat build of Keycloak 用の Oracle Database ドライバーをインストールするには、以下を実行します。

1. 次のいずれかのソースから、**ojdbc11** および **orai18n** JAR ファイルをダウンロードします。
   a. Oracle ドライバーのダウンロードページ の **圧縮された JDBC ドライバーと Companion Jars** バージョン 23.3.0.23.09。
   b. **ojdbc11** および **orai18n** 経由で Maven Central。

c. 使用しているデータベースのデータベースベンダーが推奨するインストールメディア。

2. 展開済みのディストリビューションを実行する場合: **ojdbc11** および **orai18n** JAR ファイルを、Red Hat build of Keycloak の **providers** ファイルに配置します。

3. コンテナーを実行する場合: カスタムの Red Hat build of Keycloak イメージをビルドし、**providers** フォルダーに JAR を追加します。Operator 用のカスタムイメージをビルドする場合、そのイメージは Red Hat build of Keycloak セットのすべてのビルド時オプションを使用して最適化されたイメージである必要があります。
Red Hat build of Keycloak Operator で使用でき、Maven Central からダウンロードした Oracle Database JDBC ドライバーを含むイメージをビルドするための最小限の Dockerfile は、次のようになります。

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/oracle/database/jdbc/ojdbc11/23.3.0.23.09/ojdbc11-23.3.0.23.09.jar /opt/keycloak/providers/ojdbc11.jar
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/oracle/database/nls/orai18n/23.3.0.23.09/orai18n-23.3.0.23.09.jar /opt/keycloak/providers/orai18n.jar
# Setting the build parameter for the database:
ENV KC_DB=oracle
# Add all other build parameters needed, for example enable health and metrics:
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true
# To be able to use the image with the Red Hat build of Keycloak Operator, it needs to be optimized, which requires Red Hat build of Keycloak's build step:
RUN /opt/keycloak/bin/kc.sh build
```

最適化されたイメージをビルドする方法の詳細は、コンテナーで Red Hat build of Keycloak を実行する の章を参照してください。

次のセクションの説明に従い、引き続きデータベースを設定します。

7.2.2. Microsoft SQL Server ドライバーをインストールする

Red Hat build of Keycloak 用の Microsoft SQL Server ドライバーをインストールするには、以下を実行します。

1. 次のいずれかのソースから、**mssql-jdbc** JAR ファイルをダウンロードします。

   a. Microsoft JDBC Driver for SQL Server ページから、バージョンをダウンロードする。

   b. **mssql-jdbc** 経由の Maven Central。

   c. 使用しているデータベースのデータベースベンダーが推奨するインストールメディア。

2. 展開済みのディストリビューションを実行している場合: **mssql-jdbc** を、Red Hat build of Keycloak の **providers** フォルダーに配置します。

3. コンテナーを実行する場合: カスタムの Red Hat build of Keycloak イメージをビルドし、**providers** フォルダーに JAR を追加します。Red Hat build of Keycloak Operator のカスタムイメージをビルドする場合、それらのイメージは Red Hat build of Keycloak セットのすべてのビルド時オプションを使用して最適化されたイメージである必要があります。

Red Hat build of Keycloak Operator で使用でき、Maven Central からダウンロードした Microsoft SQL Server JDBC ドライバーを含むイメージをビルドするための最小限の Dockerfile は、次のようになります。

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24
ADD --chown=keycloak:keycloak --chmod=644
https://repo1.maven.org/maven2/com/microsoft/sqlserver/mssql-jdbc/12.4.2.jre11/mssql-jdbc-12.4.2.jre11.jar /opt/keycloak/providers/mssql-jdbc.jar
# Setting the build parameter for the database:
ENV KC_DB=mssql
# Add all other build parameters needed, for example enable health and metrics:
ENV KC_HEALTH_ENABLED=true
ENV KC_METRICS_ENABLED=true
# To be able to use the image with the Red Hat build of Keycloak Operator, it needs to be optimized, which requires Red Hat build of Keycloak's build step:
RUN /opt/keycloak/bin/kc.sh build
```

最適化されたイメージをビルドする方法の詳細は、コンテナーで Red Hat build of Keycloak を実行する の章を参照してください。

次のセクションの説明に従い、引き続きデータベースを設定します。

7.3. データベースを設定する

サーバーでは、各サポート対象データベース用に、データベース設定を簡略化するための独自のデフォルトがいくつか提供されています。データベースホストや認証情報などの重要な設定を指定すると、設定が完了します。

1. サーバーを起動し、データベースを設定するための基本オプションを設定します。

```
bin/kc.[sh|bat] start --db postgres --db-url-host mypostgres --db-username myuser --db-password change_me
```

このコマンドには、データベースに接続するために必要な最小限の設定が含まれています。

デフォルトのスキーマは **keycloak** ですが、**db-schema** 設定オプションを使用して変更できます。

警告

特定の DB (H2 を除く) を使用する場合は、**start** コマンドに **--optimized** フラグを使用しないでください。サーバーインスタンスを起動する前にビルドフェーズを実行する必要があります。そのためには、**--optimized** フラグを指定せずにインスタンスを起動するか、最適化された起動の前に **build** コマンドを実行します。詳細は、Red Hat build of Keycloak の設定 を参照してください。

7.4. デフォルトの接続設定をオーバーライドする

サーバーは、データベースとの通信の基盤となるテクノロジーとして JDBC を使用します。デフォルトの接続設定が不十分な場合は、**db-url** 設定オプションを使用して JDBC URL を指定できます。

以下は、PostgreSQL データベースのサンプルコマンドです。

```
bin/kc.[sh|bat] start --db postgres --db-url jdbc:postgresql://mypostgres/mydatabase
```

; などの特殊なシェル文字を含むコマンドを呼び出す場合は、CLI を使用して文字をエスケープする必要があることに注意してください。代わりに設定ファイルにそれを設定することが推奨されます。

7.5. デフォルトの JDBC ドライバーをオーバーライドする

サーバーは、選択したデータベースに応じてデフォルトの JDBC ドライバーを使用します。

別のドライバーを設定する場合は、JDBC ドライバーの完全修飾クラス名を使用して **db-driver** を設定できます。

```
bin/kc.[sh|bat] start --db postgres --db-driver=my.Driver
```

設定したドライバーにかかわらず、デフォルトのドライバーは実行時に常に使用できます。

本当に必要な場合にのみ、このプロパティーを設定してください。たとえば、特定のクラウドデータベースサービス用に JDBC Driver Wrapper の機能を利用する場合などです。

7.6. データベースの UNICODE サポートを設定する

すべてのフィールドの Unicode サポートは、データベースが VARCHAR および CHAR フィールドで Unicode 文字セットの使用を許可するかどうかによって異なります。

- これらのフィールドが設定可能な場合、通常、Unicode はフィールド長を犠牲にして機能します。
- データベースが NVARCHAR フィールドと NCHAR フィールドでのみ Unicode をサポートしている場合、サーバースキーマは VARCHAR フィールドと CHAR フィールドを広範囲に使用するため、すべてのテキストフィールドの Unicode サポートは機能しない可能性があります。

データベーススキーマは、次の特殊フィールドでのみ Unicode 文字列のサポートを提供します。

- **Realms**: 表示名、HTML 表示名、ローカリゼーションテキスト (キーと値)
- **Federation** プロバイダー: 表示名
- **Users**: ユーザー名、名、姓、属性名、値
- **Groups**: 名前、属性名、値
- **Roles**: 名前
- オブジェクトの説明

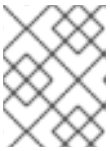
それ以外の場合、データベースエンコーディングに含まれる文字に制限され、多くの場合それは 8 ビットです。ただし、データベースシステムによっては、Unicode 文字の UTF-8 エンコーディングを有効にし、すべてのテキストフィールドで完全な Unicode 文字セットを使用できます。特定のデータベースでは、この選択により、最大文字列長が 8 ビットエンコーディングでサポートされる最大文字列長よりも短くなる可能性があります。

7.6.1. Oracle データベースの Unicode サポートを設定する

Oracle データベースが VARCHAR フィールドおよび CHAR フィールドで Unicode をサポートするように作成されている場合、そのデータベースでは Unicode 文字がサポートされます。たとえば、AL32UTF8 をデータベース文字セットとして設定したとします。この場合、JDBC ドライバーに特別な設定は必要ありません。

データベースが Unicode をサポートするように作成されていない場合、特殊フィールドで Unicode 文字をサポートするように JDBC ドライバーを設定する必要があります。2 つのプロパティーを設定します。これらのプロパティーは、システムプロパティーまたは接続プロパティーとして設定できることに注意してください。

1. **oracle.jdbc.defaultNChar** を **true** に設定します。
2. 必要に応じて、**oracle.jdbc.convertNcharLiterals** を **true** に設定します。



注記

これらのプロパティーとパフォーマンスへの影響について、詳しくは Oracle JDBC ドライバーの設定ドキュメントを参照してください。

7.6.2. Microsoft SQL Server データベースの Unicode サポート

Unicode 文字は、Microsoft SQL Server データベースの特殊フィールドでのみサポートされます。データベースには特別な設定は必要ありません。

パフォーマンスを大幅に向上させるには、JDBC ドライバーの **sendStringParametersAsUnicode** プロパティーを **false** に設定する必要があります。このパラメーターがないと、Microsoft SQL Server はインデックスを使用できない可能性があります。

7.6.3. MySQL データベースの Unicode サポートを設定する

CREATE DATABASE コマンドの使用時に VARCHAR フィールドと CHAR フィールドで Unicode サポートを指定してデータベースが作成されている場合、MySQL データベースで Unicode 文字がサポートされます。

utf8 文字セットとはストレージ要件が異なるため、utf8mb4 文字セットはサポートされていないことに注意してください。詳細は、MySQL のドキュメントを参照してください。この状況では、バイト数ではなく文字数を収容するように列が作成されるため、非特殊フィールド以外の長さ制限は適用されません。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、特殊フィールドのみが Unicode 値を保存できます。

1. Start MySQL Server.
2. JDBC ドライバー設定で、**JDBC 接続設定** を見つけます。
3. 接続プロパティー **characterEncoding=UTF-8** を追加します。

7.6.4. PostgreSQL データベースの Unicode サポートを設定する

データベースの文字セットが UTF8 の場合、Unicode は PostgreSQL データベースでサポートされます。任意のフィールドで Unicode 文字を使用でき、非特殊フィールドでフィールド長の削減はありません。JDBC ドライバーに特別な設定は必要ありません。文字セットは、PostgreSQL データベースの作成時に決定されます。

1. 次の SQL コマンドを入力して、PostgreSQL クラスターのデフォルトの文字セットを確認します。

```
-
```

```
show server_encoding;
```

2. デフォルトの文字セットが UTF 8 ではない場合は、次のようなコマンドを使用して、デフォルトの文字セットが UTF8 のデータベースを作成します。

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

7.7. AMAZON AURORA POSTGRESQL の準備

Amazon Aurora PostgreSQL を使用する場合は、Amazon Web Services JDBC ドライバー で、マルチ AZ セットアップでライターインスタンスが変更されたときのデータベース接続の転送など、追加機能を利用できます。このドライバーはディストリビューションの一部ではないため、使用する前にインストールする必要があります。

このドライバーをインストールするには、次の手順に従います。

1. 展開したディストリビューションを実行する場合: Amazon Web Services JDBC ドライバーリリースページ から JAR ファイルをダウンロードし、Red Hat build of Keycloak の **providers** フォルダーに配置します。

2. コンテナーを実行する場合: カスタムの Red Hat build of Keycloak イメージをビルドし、**providers** フォルダーに JAR を追加します。
Red Hat build of Keycloak Operator で使用できるイメージをビルドするための最小限の Dockerfile は、次のようになります。

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24
ADD --chmod=0666 https://github.com/awslabs/aws-advanced-jdbc-wrapper/releases/download/2.3.1/aws-advanced-jdbc-wrapper-2.3.1.jar /opt/keycloak/providers/aws-advanced-jdbc-wrapper.jar
```

最適化されたイメージを構築する方法の詳細は、Red Hat build of Keycloak をコンテナー内で実行する の章を参照してください。また、Red Hat build of Keycloak Operator を使用して最適化されたイメージと最適化されていないイメージを実行する方法の詳細は、カスタム Red Hat build of Keycloak イメージの使用 の章を参照してください。

3. 次のパラメーターを使用して Red Hat build of Keycloak を実行するように設定します。

db-url

通常の PostgreSQL JDBC URL に **aws-wrapper** を挿入すると、**jdbc:aws-wrapper:postgresql://...** のような URL になります。

db-driver

AWS JDBC ラッパーを使用するには、**software.amazon.jdbc.Driver** に設定します。

transaction-xa-enabled

Amazon Web Services JDBC ドライバーは XA トランザクションをサポートしていないため、**false** に設定します。

7.8. MYSQL サーバーの準備

MySQL 8.0.30 以降、MySQL は、明示的なプライマリーキーなしで作成されたすべての InnoDB テーブルに対して生成された非表示のプライマリーキーをサポートしています (詳細は、こちら を参照してください)。この機能を有効にすると、データベーススキーマの初期化と移行が失敗し、エラーメッセー

ジ **Multiple primary key defined (1068)** が表示されます。その場合、Red Hat build of Keycloak をインストールまたはアップグレードする前に、MySQL サーバー設定でパラメーター **sql_generate_invisible_primary_key** を **OFF** に設定して無効にする必要があります。

7.9. クラスター設定でデータベースロックタイムアウトを変更する

クラスターノードは同時に起動できるため、データベースのアクションに余分な時間がかかります。たとえば、起動中のサーバーインスタンスは、データベースの移行、インポート、または初回の初期化を実行する場合があります。データベースロックは、クラスターノードが同時に起動する際に起動アクションが相互に競合するのを防ぎます。

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ロックを待機すると、起動は失敗します。デフォルト値を変更する必要はほぼありませんが、変更する場合は次のコマンドを入力します。

```
bin/kc.[sh|bat] start --spi-dblock-jpa-lock-wait-timeout 900
```

7.10. XA トランザクションサポートなしでデータベースベンダーを使用する

Red Hat build of Keycloak は、デフォルトで XA トランザクションと適切なデータベースドライバーを使用します。Azure SQL や MariaDB Galera などの特定のベンダーは、XA トランザクションメカニズムをサポートしていないか、それらに依存していません。適切な JDBC ドライバーを使用して、XA トランザクションサポートなしで Red Hat build of Keycloak を使用するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --db=<vendor> --transaction-xa-enabled=false
```

Red Hat build of Keycloak は、ベンダーに適した JDBC ドライバーを自動的に選択します。

7.11. MIGRATIONSTRATEGY の JPA プロバイダー設定オプションを設定する

JPA migrationStrategy (手動/更新/検証) を設定するには、次のように JPA プロバイダーを設定する必要があります。

Connections-jpa SPI の quarkus プロバイダーの migration-strategy を設定する

```
bin/kc.[sh|bat] start --spi-connections-jpa-quarkus-migration-strategy=manual
```

DB 初期化用の SQL ファイルも取得する場合は、次の SPI initializeEmpty (true/false) を追加する必要があります。

Connections-jpa SPI の quarkus プロバイダーの Initialize-empty を設定する

```
bin/kc.[sh|bat] start --spi-connections-jpa-quarkus-initialize-empty=false
```

同様に、migrationExport が特定のファイルと場所を指すようにします。

Connections-jpa SPI の quarkus プロバイダーの migration-export を設定する

```
bin/kc.[sh|bat] start --spi-connections-jpa-quarkus-migration-export=<path>/<file.sql>
```

7.12. 関連するオプション

	値
db▮ データベースベンダー CLI: **--db** Env: **KC_DB**	**dev-file** (デフォルト)、**dev-mem**、**mariadb**、**mssql**、**mysql**、**oracle**、**postgres**
db-driver▮ JDBC ドライバーの完全修飾名。 設定されていない場合、選択したデータベースに応じてデフォルトのドライバーが設定されます。 CLI: **--db-driver** Env: **KC_DB_DRIVER**	
db-password データベースユーザーアカウントのパスワード CLI: **--db-password** Env: **KC_DB_PASSWORD**	
db-pool-initial-size 接続プールの初期サイズ。 CLI: **--db-pool-initial-size** Env: **KC_DB_POOL_INITIAL_SIZE**	
db-pool-max-size 接続プールの最大サイズ。 CLI: **--db-pool-max-size** Env: **KC_DB_POOL_MAX_SIZE**	**100** (デフォルト)
db-pool-min-size 接続プールの最小サイズ。 CLI: **--db-pool-min-size** Env: **KC_DB_POOL_MIN_SIZE**	

	値
db-schema 使用するデータベーススキーマ CLI: **--db-schema** Env: **KC_DB_SCHEMA**	
db-url データベースの完全な JDBC URL 指定しない場合、選択したデータベースベンダーに基づきデフォルト URL が設定されます。たとえば、**postgres** を使用している場合、デフォルトの JDBC URL は **jdbc:postgresql://localhost/keycloak** になります。 CLI: **--db-url** Env: **KC_DB_URL**	
db-url-database 選択したベンダーのデフォルト JDBC URL のデータベース名を設定します。 **db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-database** Env: **KC_DB_URL_DATABASE**	
db-url-host 選択したベンダーのデフォルト JDBC URL のホスト名を設定します。 **db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-host** Env: **KC_DB_URL_HOST**	
db-url-port 選択したベンダーのデフォルト JDBC URL のポートを設定します。 **db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-port** Env: **KC_DB_URL_PORT**	

	値
db-url-properties 選択したベンダーのデフォルト JDBC URL のプロパティーを設定します。 データベースベンダーが想定する形式に従ってプロパティーを設定し、そのプロパティー値の先頭に正しい文字を追加してください。**db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-properties** Env: **KC_DB_URL_PROPERTIES**	
db-username データベースユーザーのユーザー名 CLI: **--db-username** Env: **KC_DB_USERNAME**	
transaction-xa-enabled false に設定すると、データベースが XA トランザクションをサポートしない場合に備えて、Keycloak は非 XA データソースを使用します。 CLI: **--transaction-xa-enabled** Env: **KC_TRANSACTION_XA_ENABLED**	**true** (デフォルト)、**false**

第8章 分散キャッシュの設定

Red Hat build of Keycloak は、高可用性とマルチノードのクラスター化セットアップ向けに設計されています。現在の分散キャッシュ実装は、高性能で分散可能なインメモリーデータグリッドである Infinispan の上にビルドされています。

8.1. 分散キャッシュを有効にする

start コマンドを使用して Red Hat build of Keycloak を実稼働モードで開始すると、キャッシュが有効になり、ネットワーク上の Red Hat build of Keycloak ノードがすべて検出されます。

デフォルトでは、キャッシュは UDP トランスポートスタックを使用するため、UDP に基づく IP マルチキャストトランスポートを使用してノードが検出されます。ほとんどの実稼働環境では、UDP に代わるより優れた検出手段が利用可能です。この章で後述するとおり、Red Hat build of Keycloak では、事前定義されたデフォルトのトランスポートスタックのセットから選択することも、独自のカスタムスタックを定義することもできます。

分散 Infinispan キャッシュを明示的に有効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --cache=ispn
```

start-dev コマンドを使用して Red Hat build of Keycloak を開発モードで開始すると、Red Hat build of Keycloak はローカルキャッシュのみを使用し、分散キャッシュは **--cache=local** オプションを暗黙的に設定することによって完全に無効になります。**local** キャッシュモードは、開発目的およびテスト目的に限定されています。

8.2. キャッシュを設定する

Red Hat build of Keycloak は 、**conf/cache-ispn.xml** に置かれた適切なデフォルトを含むキャッシュ設定ファイルを提供します。

キャッシュ設定は、通常の nfinispan configuration file です。

次の表は、Red Hat build of Keycloak が使用する特定のキャッシュの概要を示しています。これらのキャッシュは、**conf/cache-ispn.xml** で設定します。

キャッシュ名	キャッシュタイプ	説明
realms	Local	永続化されたレルムデータをキャッシュします
ユーザー	Local	永続化されたユーザーデータをキャッシュします
authorization	Local	永続化された認可データをキャッシュします
keys	Local	外部公開鍵をキャッシュします
work	Replicated (レプリケート)	無効化メッセージをノード間で伝播します

キャッシュ名	キャッシュタイプ	説明
authenticationSessions	Distributed (分散)	認証プロセス中に作成/破棄された、または期限切れになった認証セッションをキャッシュします。
sessions	Distributed (分散)	認証の成功時に作成され、ログアウト、トークンの失効中、または有効期限切れが原因で破棄されたユーザーセッションをキャッシュします。
clientSessions	Distributed (分散)	特定のクライアントに対する認証の成功時に作成され、ログアウト、トークンの失効中、または有効期限切れが原因で破棄されたクライアントセッションをキャッシュします。
offlineSessions	Distributed (分散)	認証の成功時に作成され、ログアウト、トークンの失効中、または有効期限切れが原因で破棄されたオフラインユーザーセッションをキャッシュします。
offlineClientSessions	Distributed (分散)	特定のクライアントに対する認証の成功時に作成され、ログアウト、トークンの失効中、または有効期限切れが原因で破棄されたクライアントセッションをキャッシュします。
loginFailures	Distributed (分散)	失敗したログイン (不正の検知) を追跡します
actionTokens	Distributed (分散)	アクショントークンをキャッシュします

8.2.1. キャッシュタイプとデフォルト

ローカルキャッシュ

Red Hat build of Keycloak は、データベースへの不必要なラウンドトリップを回避するために、永続データをローカルにキャッシュします。

次のデータは、ローカルキャッシュを使用して、クラスター内の各ノードのローカルに保持されます。

- **レルム** と、クライアント、ロール、グループなどの関連データ。
- **ユーザー** と、付与されたロールやグループメンバーシップなどの関連データ。
- **認可** と、リソース、権限、ポリシーなどの関連データ。

- keys

レルム、ユーザー、認可のローカルキャッシュは、デフォルトで最大 10,000 エントリーを保持するように設定されています。デフォルトで、ローカルキーキャッシュは最大 1,000 エントリーを保持でき、1 時間ごとに期限切れになります。したがって、外部クライアントまたはアイデンティティープロバイダーからキーを定期的にダウンロードする必要があります。

最適な実行時間を実現し、データベースへの追加のラウンドトリップを回避するには、各キャッシュの設定で、エントリーの最大数がデータベースのサイズと一致していることを確認する必要があります。キャッシュできるエントリーが増えると、サーバーがデータベースからデータを取得しなければならない回数が少なくなります。メモリーの使用率とパフォーマンスの間で何が犠牲になるかを評価する必要があります。

ローカルキャッシュの無効化

ローカルキャッシュによりパフォーマンスは向上しますが、マルチノードセットアップでは課題が発生します。

1 つの Red Hat build of Keycloak ノードが共有データベース内のデータを更新すると、他のすべてのノードはそれを認識する必要があるため、キャッシュからそのデータを無効にします。

work キャッシュはレプリケートされたキャッシュであり、これらの無効化メッセージの送信に使用されます。このキャッシュのエントリー/メッセージは有効期限が非常に短いため、このキャッシュのサイズが時間の経過とともに増加することを想定する必要はありません。

認証セッション

ユーザーが認証を試みるたびに、認証セッションが作成されます。これらは、認証プロセスが完了するか、有効期限に達すると自動的に破棄されます。

authenticationSessions 分散キャッシュは、認証プロセス中に、認証セッションとそれに関連する他のデータを保存するために使用されます。

分散可能キャッシュに依存すると、クラスター内のどのノードでも認証セッションを利用できるため、ユーザーは認証状態を失うことなく任意のノードにリダイレクトできます。ただし、実稼働環境に対応したデプロイメントでは、常にセッションアフィニティーを考慮し、セッションが最初に作成されたノードにユーザーをリダイレクトすることを優先する必要があります。これにより、ノード間の不要な状態遷移が回避され、CPU、メモリー、ネットワークの使用率が向上します。

ユーザーセッション

ユーザーが認証されると、ユーザーセッションが作成されます。ユーザーセッションはアクティブユーザーとその状態を追跡するため、認証情報を再度要求されることなく、あらゆるアプリケーションに対してシームレスに認証できるようになります。アプリケーションごとに、ユーザーは作成されたクライアントセッションでも認証されるため、サーバーは、ユーザーが認証されたアプリケーションとその状態をアプリケーションごとに追跡できます。

ユーザーセッションとクライアントセッションは、ユーザーがログアウトを実行するとき、クライアントがトークンの取り消しを実行するとき、または有効期限に達したときに自動的に破棄されます。

次のキャッシュは、ユーザーセッションおよびクライアントセッションを保存するために使用されます。

- sessions
- clientSessions

分散可能キャッシュに依存すると、クラスター内のどのノードでもユーザーセッションとクライアント

セッションを利用できるため、状態遷移を発生させることなくユーザーを任意のノードにリダイレクトできます。ただし、実稼働環境に対応したデプロイメントでは、常にセッションアフィニティーを考慮し、セッションが最初に作成されたノードにユーザーをリダイレクトすることを優先する必要があります。これにより、ノード間の不要な状態遷移が回避され、CPU、メモリー、ネットワークの使用率が向上します。

サーバーは、OpenID Connect プロバイダーとしてユーザーを認証し、オフライントークンを発行することもできます。通常のユーザーセッションやクライアントセッションと同様に、サーバーは認証成功時にオフライントークンを発行すると、オフラインユーザーセッションとオフラインクライアントセッションも作成します。ただし、オフライントークンの性質上、オフラインセッションの有効期間は長く、クラスターを完全にシャットダウンしても存続するため、オフラインセッションの処理は異なります。そのため、データベースでも永続化されます。

次のキャッシュは、オフラインセッションを保存するために使用されます。

- offlineSessions
- offlineClientSessions

クラスターを再起動すると、オフラインセッションはデータベースから遅延してロードされ、上記の 2 つのキャッシュを使用して共有キャッシュに保持されます。

パスワードのブルートフォース検出

loginFailures 分散キャッシュは、失敗したログイン試行に関するデータを追跡するために使用されます。このキャッシュは、マルチノード Red Hat build of Keycloak セットアップでブルートフォース保護機能が動作するために必要です。

アクショントークン

アクショントークンは、たとえばパスワードを忘れた場合のフローで送信されるメールなど、アクションを非同期で確認する必要がある場合に使用されます。**actionTokens** 分散キャッシュは、アクショントークンに関するメタデータを追跡するために使用されます。

8.2.2. 可用性のためのキャッシュ設定

分散キャッシュは、クラスター内のノードのサブセットでキャッシュエントリーをレプリケートし、エントリーを固定所有者ノードに割り当てます。

デフォルトでは、各分散キャッシュには 2 人の所有者がいます。これは、2 つのノードが特定のキャッシュエントリーのコピーを持つことを意味します。非所有者ノードは、データを取得するために、特定のキャッシュの所有者にクエリーを実行します。両方の所有者ノードがオフラインになると、すべてのデータが失われます。通常この状況では、次の要求時にユーザーはログアウトされ、再度ログインする必要があります。

デフォルトの所有者数は、3 つ以上のノードを持つクラスターセットアップで 1 つのノード (所有者) で障害が発生しても継続できる数です。所有者の数は、可用性要件に合わせて自由に変更できます。所有者の数を変更するには、**conf/cache-ispn.xml** を開き、分散キャッシュの **owner=<value>** 値を任意の値に変更します。

8.2.3. 独自のキャッシュ設定ファイルを指定する

独自のキャッシュ設定ファイルを指定するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --cache-config-file=my-cache-file.xml
```

設定ファイルは **conf/** ディレクトリーに相対的です。

8.2.4. リモートサーバーの CLI オプション

高可用性用およびマルチノードのクラスター化セットアップ用に Red Hat build of Keycloak サーバーを設定するために、CLI オプション **cache-remote-host**、**cache-remote-port**、**cache-remote-username**、**cache-remote-password** が導入され、XML ファイル内の設定が簡素化されました。上記の CLI パラメーターのいずれかが存在する場合、XML ファイル内にリモートストアに関連する設定が存在しないしないものと想定されます。

8.3. トランスポートスタック

トランスポートスタックにより、クラスター内の分散キャッシュノードは信頼できる方法で通信できます。Red Hat build of Keycloak は、幅広いトランスポートスタックをサポートしています。

- tcp
- udp
- kubernetes
- ec2
- azure
- google

特定のキャッシュスタックを適用するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --cache-stack=<stack>
```

分散キャッシュが有効な場合、デフォルトのスタックが **udp** に設定されます。

8.3.1. 利用可能なトランスポートスタック

次の表は、**--cache-stack** ビルドオプションを使用する以外の設定を行わずに使用できるトランスポートスタックを示しています。

スタック名	トランスポートプロトコル	検出
tcp	TCP	MPING (UDP マルチキャストを使用)。
udp	UDP	UDP マルチキャスト

次の表は、**--cache-stack** ビルドオプションと最小の設定で使用できるトランスポートスタックを示しています。

スタック名	トランスポートプロトコル	検出

スタック名	トランスポートプロトコル	検出
kubernetes	TCP	DNS_PING (**-Djgroups.dns.query=<headless-service-FQDN>** を JAVA_OPTS または JAVA_OPTS_APPEND 環境変数に追加する必要があります)。

8.3.2. 追加のトランスポートスタック

次の表は、Red Hat build of Keycloak でサポートされているトランスポートスタックを示しています。ただし、動作するには追加の手順が必要です。これらのスタックは、いずれも Kubernetes/OpenShift スタックでは **ありません**。そのため、Google Kubernetes エンジン上で Red Hat build of Keycloak を実行する場合は、**google** スタックを有効にする必要はありません。その場合は、**kubernetes** スタックを使用します。AWS EC2 インスタンスで実行されている分散キャッシュセットアップがある場合は、スタックを **ec2** に設定する必要があります。これは、ec2 が UDP などのデフォルトの検出メカニズムをサポートしていないためです。

スタック名	トランスポートプロトコル	検出
ec2	TCP	NATIVE_S3_PING
google	TCP	GOOGLE_PING2
azure	TCP	AZURE_PING

クラウドベンダー固有のスタックには、Red Hat build of Keycloak に対する追加の依存関係があります。これらの依存関係に関する詳細とリポジトリーへのリンクについては、Infinispan のドキュメントを参照してください。

Red Hat build of Keycloak の依存関係を提供するには、次のコマンドを入力して、それぞれの JAR を **providers** ディレクトリーに配置し、Red Hat build of Keycloak をビルドします。

```
bin/kc.[sh|bat] build --cache-stack=<ec2|google|azure>
```

8.3.3. カスタムのトランスポートスタック

使用可能なトランスポートスタックがデプロイメントに適切ではない場合、キャッシュ設定ファイルを変更して、独自のトランスポートスタックを定義できます。

詳細は、Using inline JGroups stacks を参照してください。

カスタムのトランスポートスタックを定義する

```
<jgroups>
    <stack name="my-encrypt-udp" extends="udp">
    <SSL_KEY_EXCHANGE keystore_name="server.jks"
        keystore_password="password"
        stack.combine="INSERT_AFTER"
```

```
        stack.position="VERIFY_SUSPECT2"/>
        <ASYM_ENCRYPT asym_keylength="2048"
        asym_algorithm="RSA"
        change_key_on_coord_leave = "false"
        change_key_on_leave = "false"
        use_external_key_exchange = "true"
        stack.combine="INSERT_BEFORE"
        stack.position="pbcast.NAKACK2"/>
    </stack>
</jgroups>

<cache-container name="keycloak">
    <transport lock-timeout="60000" stack="my-encrypt-udp"/>
    ...
</cache-container>
```

デフォルトでは、**cache-stack** オプションに設定された値は、キャッシュ設定ファイルで定義したトランスポートスタックよりも優先されます。カスタムスタックを定義する場合は、カスタムの変更を有効にするために **cache-stack** オプションを使用していないことを確認してください。

8.4. キャッシュ通信を保護する

現行の Infinispan キャッシュ実装は、RBAC、ACL、トランスポートスタック暗号化などのさまざまなセキュリティー対策により保護されているはずです。

JGroups は、Red Hat build of Keycloak サーバー間のすべての通信を処理し、TCP 通信用の Java SSL ソケットをサポートします。Red Hat build of Keycloak は、CLI オプションを使用して TLS 通信を設定します。カスタマイズした JGroups スタックを作成したり、キャッシュ XML ファイルを変更したりする必要はありません。

TLS を有効にするには、**cache-embedded-mtls-enabled** を **true** に設定する必要があります。使用する証明書を含むキーストアが必要です。**cache-embedded-mtls-key-store-file** でキーストアへのパスを設定し、**cache-embedded-mtls-key-store-password** でそれを復号するためのパスワードを設定します。トラストストアに、接続を受け入れるための有効な証明書を格納します。トラストストアは、**cache-embedded-mtls-trust-store-file** (トラストストアへのパス) と **cache-embedded-mtls-trust-store-password** (それを復号するためのパスワード) を使用して設定できます。不正アクセスを制限するには、各 Red Hat build of Keycloak に自己署名証明書を使用します。

UDP または **TCP_NIO2** を使用した JGroups スタックの場合、プロトコルスタックの設定方法について、JGroups 暗号化ドキュメント を参照してください。

キャッシュ通信の保護の詳細は、Infinispan Security Guide を参照してください。

8.5. キャッシュからのメトリクスを公開する

デフォルトでは、メトリクスが有効になっている場合、キャッシュからのメトリクスは自動的に公開されません。メトリクスを有効にする方法について、詳細は Red Hat build of Keycloak のメトリクスを有効にする を参照してください。

cache-container 内のすべてのキャッシュに対してグローバルメトリクスを有効にするには、キャッシュ設定ファイル (例: **conf/cache-ispn.xml**) を次のように変更して **cache-container** レベルで **statistics** を有効にする必要があります。

すべてのキャッシュのメトリクスを有効にする

```
<cache-container name="keycloak" statistics="true">
    ...
</cache-container>
```

同様に、次のように **statistics** を有効にすることで、各キャッシュのメトリクスを個別に有効にできます。

特定のキャッシュのメトリクスを有効にする

```
<local-cache name="realms" statistics="true">
    ...
</local-cache>
```

8.6. 関連するオプション

	値
cache▮ 高可用性のためのキャッシュメカニズムを定義します。 実稼働モードのデフォルトでは、複数のサーバーノード間のクラスターは **ispn** キャッシュを使用して作成されます。開発モードのデフォルトでは、**local** キャッシュはクラスタリングを無効にし、開発とテスト目的で使用されます。 CLI: **--cache** Env: **KC_CACHE**	**ispn** (デフォルト)、**local**
cache-config-file▮ キャッシュ設定のロード元となるファイルを定義します。 設定ファイルは **conf/** ディレクトリーに相対です。 CLI: **--cache-config-file** Env: **KC_CACHE_CONFIG_FILE**	
cache-embedded-mtls-enabled Keycloak サーバー間のネットワーク通信を暗号化します。 CLI: **--cache-embedded-mtls-enabled** Env: **KC_CACHE_EMBEDDED_MTLS_ENABLED**	**true**、**false** (デフォルト)

	値
cache-embedded-mtls-key-store-file キーストアファイルのパス。 キーストアには、TLS プロトコルが使用する証明書が含まれている必要があります。デフォルトでは、conf/ ディレクトリーの下にある **cache-mtls-keystore.p12** が検索されます。 CLI: **--cache-embedded-mtls-key-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_FILE**	
cache-embedded-mtls-key-store-password キーストアにアクセスするためのパスワード。 CLI: **--cache-embedded-mtls-key-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_PASSWORD**	
cache-embedded-mtls-trust-store-file トラストストアファイルのパス。 トラストストアには、信頼済み証明書または証明書に署名した認証局が含まれている必要があります。デフォルトでは、conf/ ディレクトリーの下にある **cache-mtls-truststore.p12** が検索されます。 CLI: **--cache-embedded-mtls-trust-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_FILE**	
cache-embedded-mtls-trust-store-password トラストストアにアクセスするためのパスワード。 CLI: **--cache-embedded-mtls-trust-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_PASSWORD**	
cache-remote-host リモートストア設定のリモートサーバーのホスト名。 これは、XML ファイルで指定された設定の **remote-server** タグの **host** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-username** と **cache-remote-password** も必要となり、XML ファイル内に関連する設定を含めることができません。 CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	

	値
cache-remote-password リモートストアのリモートサーバーへの認証に使用するパスワード。 これは、XML ファイルで指定された設定の **digest** タグの **password** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-host** と **cache-remote-username** も必要となり、XML ファイル内に関連する設定を含めることができません。 CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD**	
cache-remote-port リモートストア設定用のリモートサーバーのポート。 これは、XML ファイルで指定された設定の **remote-server** タグの **port** 属性を置き換えます (**cache-config-file** オプションを参照)。 CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT**	**11222** (デフォルト)
cache-remote-username リモートストアのリモートサーバーへの認証に使用するユーザー名。 これは、XML ファイルで指定された設定の **digest** タグの **username** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-host** と **cache-remote-password** も必要となり、XML ファイル内に関連する設定を含めることができません。 CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME**	
cache-stack ▮ クラスター通信とノード検出に使用するデフォルトのスタックを定義します。 このオプションは、**cache** が **ispn** に設定されている場合にのみ有効です。デフォルト: udp. CLI: **--cache-stack** Env: **KC_CACHE_STACK**	**tcp**, **udp**, **kubernetes**, **ec2**, **azure**, **google**

第9章 送信 HTTP 要求を設定する

Red Hat build of Keycloak では、保護するアプリケーションやサービスに対して頻繁に要求を行う必要があります。Red Hat build of Keycloak は、HTTP クライアントを使用してこれらの送信接続を管理します。この章では、クライアント、接続プール、プロキシー環境設定、タイムアウトなどの設定方法を説明します。

9.1. クライアント設定コマンド

Red Hat build of Keycloak が送信通信に使用する HTTP クライアントは、高度な設定が可能です。Red Hat build of Keycloak の送信 HTTP クライアントを設定するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-<configurationoption>=<value>
```

コマンドのオプションは次のとおりです。

establish-connection-timeout-millis

接続の確立がタイムアウトになるまでの最大時間 (ミリ秒)。デフォルト: 設定されていません。

socket-timeout-millis

2 つのデータパケット間で、ソケット接続がタイムアウトになるまでの非アクティブな最大時間 (ミリ秒単位)。デフォルト: 5000ms

connection-pool-size

送信接続の接続プールのサイズ。デフォルト: 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: 64

connection-ttl-millis

最大接続時間 (ミリ秒単位)。デフォルト: 設定されていません。

max-connection-idle-time-millis

アイドル状態の接続が接続プール内に留まる最大時間 (ミリ秒単位)。アイドル状態の接続は、バックグラウンドクリーナースレッドによってプールから削除されます。このチェックを無効にするには、このオプションを -1 に設定します。デフォルト: 900000

disable-cookies

cookie のキャッシュを有効または無効にします。デフォルト: true

client-keystore

Java キーストアファイルへのパス。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。**client-keystore が** 設定されている場合は必須。

client-key-password

クライアントの秘密鍵のパスワード。client-keystore が設定されている場合は必須。

proxy-mappings

送信 HTTP 要求のプロキシー設定を指定します。詳細は、「HTTP 要求の送信プロキシーマッピング」 を参照してください。

disable-trust-manager

送信要求に 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、トラストストアを指定する必要はありません。この設定は SSL 証明書の検証を無効にするため、開発時にのみ使用し、**実稼働環境では絶対に使用しないでください**。デフォルト: false

9.2. HTTP 要求の送信プロキシーマッピング

プロキシーを使用するように送信要求を設定するには、標準プロキシー環境変数である **HTTP_PROXY**、**HTTPS_PROXY**、**NO_PROXY** を使用してプロキシーマッピングを設定します。

- **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ーサーバーを表します。Red Hat build of Keycloak では、この 2 つの変数は区別されません。両方の変数を定義すると、プロキシーサーバーが使用する実際のスキームに関係なく、**HTTPS_PROXY** が優先されます。
- **NO_PROXY** 変数は、プロキシーを使用しないホスト名のコンマ区切りリストを定義します。指定したホスト名ごとに、そのすべてのサブドメインもプロキシーの使用から除外されます。

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** の両方を定義した場合、**http_proxy** が使用されます。

プロキシーマッピングと環境変数の例

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

この例では、次のような結果になります。

- google.com または google.com のサブドメイン (例: auth.google.com) への要求を除き、すべての送信要求はプロキシー **https://www-proxy.acme.com:8080** を使用します。
- login.facebook.com とそのすべてのサブドメインは、定義されたプロキシーを使用しません。ただし、groups.facebook.com は login.facebook.com のサブドメインではないため、除外されます。

9.3. 正規表現を使用したプロキシーマッピング

プロキシーマッピングに環境変数を使用する代わりに、Red Hat build of Keycloak が送信する送信要求の proxy-mappings のコンマ区切りリストを設定できます。proxy-mapping は、**hostname-pattern;proxy-uri** 形式を使用する、正規表現ベースのホスト名パターンとプロキシー URI で構成されます。

たとえば、次の正規表現があります。

```
.*\.(google|googleapis)\.com
```

次のコマンドを入力して、正規表現ベースのホスト名パターンを適用します。

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-proxy-mappings="'*\\\.
(google|googleapis)\\\.com;http://www-proxy.acme.com:8080'"
```

送信 HTTP 要求を決定するために、以下が実行されます。

- ターゲットのホスト名を、設定されているすべてのホスト名パターンと照合します。
- 最初に一致したパターンの proxy-uri が使用されます。
- ホスト名と一致する設定済みパターンがない場合、プロキシーは使用されません。

プロキシーサーバーに認証が必要な場合は、**username:password@** 形式でプロキシーユーザーの認証情報を含めます。以下に例を示します。

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

プロキシーマッピングの正規表現の例:

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

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

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

この例では、以下が実行されます。

- proxy-uri の特別な値である NO_PROXY が使用されます。これは、関連付けられたホスト名パターンに一致するホストにはプロキシーが使用されないことを意味します。
- catch-all パターンはプロキシーマッピングを終了し、すべての送信要求にデフォルトのプロキシーを提供します。

9.4. TLS 接続の信頼済み証明書を設定する

Red Hat build of Keycloak が TLS を使用して送信要求を実行できるように Red Hat build of Keycloak のトラストストアを設定する方法については、信頼済み証明書の設定 を参照してください。

第10章 信頼済み証明書の設定

Red Hat build of Keycloak が外部サービスと通信する場合、または TLS 経由で着信接続を行う場合、信頼済みサーバーに接続していることを確認するために、リモート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。

これらのクライアントまたはサーバーの証明書、またはこれらの証明書に署名した CA を、トラストストアに配置する必要があります。その後、このトラストストアを、Red Hat build of Keycloak で使用するために設定します。

10.1. システムトラストストアの設定

既存の Java デフォルトトラストストア証明書は、常に信頼されます。追加の証明書が必要な場合 (JRE によって認識されない自己署名認証局または内部認証局がある場合など) は、それを **conf/truststores** ディレクトリーまたはサブディレクトリーに追加することができます。証明書は、PEM ファイルか、拡張子が **.p12** または **.pfx** の PKCS12 ファイル内に含めます。PKCS12 の場合、証明書が暗号化されていない必要があります。つまり、パスワードは必要ありません。

別のパスが必要な場合は、**--truststore-paths** オプションを使用して、PEM または PKCS12 ファイルが配置されている追加のファイルまたはディレクトリーを指定します。パスは、Red Hat build of Keycloak を起動した場所に対する相対パスであるため、代わりに絶対パスを使用することを推奨します。ディレクトリーが指定されている場合、そのディレクトリでトラストストアファイルが再帰的にスキャンされます。

該当するすべての証明書を追加すると、トラストストアは、**javax.net.ssl** プロパティーを介してシステムのデフォルトのトラストストアとして、また Red Hat build of Keycloak 内部で使用するデフォルトのトラストストアとして使用されます。

以下に例を示します。

```
bin/kc.[sh|bat] start --truststore-paths=/opt/truststore/myTrustStore.pfx,/opt/other-truststore/myOtherTrustStore.pem
```

独自の **javax.net.ssl** トラストストアシステムプロパティーを直接設定することも可能ですが、代わりに **--truststore-paths** を使用することを推奨します。

10.2. ホスト名検証ポリシー

tls-hostname-verifier プロパティーを使用して、TLS 接続によるホスト名の検証方法を調整できます。

- **WILDCARD** (デフォルト) では、サブドメイン名にワイルドカードを使用できます (例: *.foo.com)。
- **ANY** に設定すると、ホスト名が検証されません。
- **STRICT** を使用する場合、コモンネーム (CN) がホスト名と完全に一致する必要があります。この設定は、厳密なホスト名チェックを必要とする LDAP セキュア接続には適用されないことに注意してください。

10.3. 関連するオプション

	値
tls-hostname-verifier 送信 HTTPS および SMTP 要求の TLS ホスト名検証ポリシー。 CLI: **--tls-hostname-verifier** Env: **KC_TLS_HOSTNAME_VERIFIER**	**ANY**、**WILDCARD** (デフォルト)、**STRICT**
truststore-paths システムトラストストアとして使用される pkcs12 (p12 または pfx ファイル拡張子)、PEM ファイル、またはそれらのファイルを含むディレクトリーのリスト。 CLI: **--truststore-paths** Env: **KC_TRUSTSTORE_PATHS**	

第11章 機能の有効化と無効化

Red Hat build of Keycloak には、テクノロジープレビューや非推奨機能などの無効化された機能を含め、いくつかの機能が組み込まれています。他の機能はデフォルトで有効になっていますが、Red Hat build of Keycloak のユースケースに適さない場合は無効にできます。

11.1. 機能を有効にする

サポートされている一部の機能とすべてのプレビュー機能は、デフォルトで無効になっています。機能を有効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --features="<name>[,<name>]"
```

たとえば、**docker** と **token-exchange** を有効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --features="docker,token-exchange"
```

すべてのプレビュー機能を有効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --features="preview"
```

有効な機能には、バージョン管理されているものも、バージョン管理されていないものもあります。バージョン付きの機能名 (例: feature:v1) を使用すると、その機能バージョンがランタイム内に存在する限り有効になります。代わりにバージョンなしの名前 (例: feature) を使用すると、サポートされる特定の機能バージョンの選択が、次の優先順位によってリリースごとに変わる可能性があります。

1. サポートされている最新のデフォルトバージョン
2. サポートされている最新のデフォルトでないバージョン
3. 最新の非推奨のバージョン
4. 最新のプレビューバージョン
5. 最新の実験バージョン

11.2. 機能を無効にする

デフォルトで有効になっている機能を無効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --features-disabled="<name>[,<name>]"
```

たとえば、**impersonation** を無効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] build --features-disabled="impersonation"
```

次のコマンドを入力すると、デフォルトの機能をすべて無効にできます。

```
bin/kc.[sh|bat] build --features-disabled="default"
```

このコマンドを **features** と組み合わせて使用すると、どの機能を使用可能にするかを明示的に設定できます。ある機能を **features-disabled** リストと **features** リストの両方に指定することはできません。

機能を無効にすると、その機能のすべてのバージョンが無効になります。

11.3. サポートされる機能

次のリストには、デフォルトで有効になっているサポート対象機能が含まれています。これらの機能は、必要がない場合は無効にできます。

account-api
: アカウント管理 REST API

account3
: アカウントコンソールバージョン 3

admin-api
: 管理者 API

admin2
: 新しい管理コンソール

authorization
: 認可サービス

ciba
: OpenID Connect Client Initiated Backchannel Authentication (CIBA)

client-policies
: クライアント設定ポリシー

device-flow
: OAuth 2.0 Device Authorization Grant

hostname-v1
: ホスト名オプション V1

impersonation
: 管理者がユーザーに成り代わる機能

js-adapter
: Keycloak サーバー経由で keycloak.js と keycloak-authz.js をホストします

Kerberos
: Kerberos

par
: OAuth 2.0 Pushed Authorization Requests (PAR)

step-up-authentication
: ステップアップ認証

web-authn
: W3C Web Authentication (WebAuthn)

11.3.1. デフォルトでは無効になっています。

次のリストには、デフォルトで無効になっているサポート対象機能が含まれています。これらの機能は、必要に応じて有効にできます。

docker
: Docker レジストリープロトコル

fips

FIPS 140-2 モード

multi-site

マルチサイトサポート

11.4. プレビュー機能

プレビュー機能はデフォルトでは無効になっており、実稼働環境での使用は推奨されません。これらの機能は、今後のリリースで変更または削除される可能性があります。

admin-fine-grained-authz

きめ細かい管理パーミッション

client-secret-rotation

クライアントのシークレットローテーション

dpop

OAuth 2.0 Demonstrating Proof-of-Possession at the Application Layer

recovery-codes

リカバリーコード

scripts

JavaScript を使用したカスタムオーセンティケーターの作成

token-exchange

トークン交換サービス

update-email

メールアクションを更新します

11.5. 非推奨の機能

次のリストには、今後のリリースで削除される予定の非推奨機能が含まれています。これらの機能は、デフォルトで無効になっています。

account2

アカウントコンソールバージョン 2

linkedin-oauth

OAuth に基づく LinkedIn ソーシャルアイデンティティープロバイダー

offline-session-preloading

オフラインセッションのプリロード

11.6. 関連するオプション

	値
features ▮ 1 つ以上の機能セットを有効にします。 CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account2[:v1]**, **account3[:v1]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1]**, **admin2[:v1]**, **authorization[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v1]**, **impersonation[:v1]**, **js-adapter[:v1]**, **kerberos[:v1]**, **linkedin-oauth[:v1]**, **login2[:v1]**, **multi-site[:v1]**, **offline-session-preloading[:v1]**, **oid4vc-vci[:v1]**, **par[:v1]**, **preview**, **recovery-codes[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **web-authn[:v1]**

	値
features-disabled ▮ 1 つ以上の機能のセットを無効にします。 CLI: **--features-disabled** Env: **KC_FEATURES_DISABLED**	**account-api**, **account2**, **account3**, **admin-api**, **admin-fine-grained-authz**, **admin2**, **authorization**, **ciba**, **client-policies**, **client-secret-rotation**, **client-types**, **declarative-ui**, **device-flow**, **docker**, **dpop**, **dynamic-scopes**, **fips**, **impersonation**, **js-adapter**, **kerberos**, **linkedin-oauth**, **login2**, **multi-site**, **offline-session-preloading**, **oid4vc-vci**, **par**, **preview**, **recovery-codes**, **scripts**, **step-up-authentication**, **token-exchange**, **transient-users**, **update-email**, **web-authn**

第12章 プロバイダーの設定

サーバーは拡張性を念頭に置いて構築されています。そのために多数のサービスプロバイダーインターフェイス (SPI) が提供されており、それぞれがサーバーに特定の機能を提供します。この章では、SPI とそれぞれのプロバイダーの設定に関する中核的な概念を理解します。

この章を読むと、説明された概念と手順を使用して、プロバイダーをインストール、アンインストール、有効化、無効化、設定するための概念と手順を使用できるようになります。そこには、それぞれの要件を満たすことを目的として、サーバー機能を拡張するために実装したプロバイダーも含まれます。

12.1. 設定オプションの形式

プロバイダーは、特定の設定形式を使用して設定できます。形式は、以下で構成されています。

```
spi-<spi-id>-<provider-id>-<property>=<value>
```

<spi-id> は、設定する SPI の名前です。

<provider-id> は、設定するプロバイダーの ID です。これは、対応するプロバイダーファクトリー実装に設定された ID です。

<property> は、特定のプロバイダーに設定するプロパティーの実際の名前です。

これらの名前 (spi、プロバイダー、プロパティー) はすべて小文字とし、**myKeycloakProvider** のようなキャメルケースの名前は、**my-keycloak-provider** のように大文字の前にダッシュ (**-**) が必要です。

たとえば **HttpClientSpi** SPI の場合、SPI の名前は **connectionsHttpClient** で、使用可能なプロバイダー実装の 1 つは、**default** という名前です。**connectionPoolSize** プロパティーを設定するには、次のように設定オプションを使用します。

```
spi-connections-http-client-default-connection-pool-size=10
```

12.2. プロバイダー設定オプションを設定する

プロバイダー設定オプションは、サーバーの起動時に指定します。Red Hat build of Keycloak の設定 のオプションで、サポートされているすべての設定ソースと形式を参照してください。たとえば、コマンドラインオプションを使用して以下のように指定します。

Connections-http-client SPI の default プロバイダーの connection-pool-size を設定する

```
bin/kc.[sh|bat] start --spi-connections-http-client-default-connection-pool-size=10
```

12.3. デフォルトプロバイダーを設定する

SPI によっては、複数のプロバイダー実装が同時に存在することも可能ですが、実行時に使用されるのはそのうちの 1 つだけです。これらの SPI では、デフォルトプロバイダーが主要な実装となり、実行時にアクティブになって使用されます。

プロバイダーをデフォルトとして設定するには、次のように **build** コマンドを実行する必要があります。

mycustomprovider プロバイダーを email-template SPI のデフォルトプロバイダーとしてマークする

```
bin/kc.[sh|bat] build --spi-email-template-provider=mycustomprovider
```

上記の例では、**provider** プロパティーを使用して、デフォルトとしてマークするプロバイダーの ID を設定しています。

12.4. プロバイダーの有効化と無効化

プロバイダーを有効または無効にするには、次のように **build** コマンドを実行する必要があります。

プロバイダーを無効にする

```
bin/kc.[sh|bat] build --spi-email-template-mycustomprovider-enabled=true
```

プロバイダーを無効にするには、同じコマンドを使用し、**enabled** プロパティーを **false** に設定します。

12.5. プロバイダーのインストールとアンインストール

カスタムプロバイダーは、Java アーカイブ (JAR) ファイルにパッケージ化して、ディストリビューションの **providers** ディレクトリーにコピーする必要があります。その後、**build** コマンドを実行して、サーバーのプロバイダーレジストリーを JAR ファイルの実装で更新する必要があります。

この手順は、サーバーランタイムを最適化するために必要です。そうすることで、サーバーの起動時や実行時にプロバイダーを検出するのではなく、事前にすべてのプロバイダーが既知になるようになります。

プロバイダーをアンインストールするには、**providers** ディレクトリーから JAR ファイルを削除し、**build** コマンドを再度実行する必要があります。

12.6. サードパーティーの依存関係を使用する

プロバイダーを実装する場合、サーバーディストリビューションからは利用できないサードパーティーの依存関係を使用する必要があることもあります。

この場合、追加の依存関係を **providers** ディレクトリーにコピーし、**build** コマンドを実行する必要があります。これを実行すると、サーバーはこれらの追加の依存関係を、それに依存するプロバイダーが実行時に使用できるようにします。

12.7. 参考資料

- Red Hat build of Keycloak の設定
- Server Developer Documentation

第13章 ロギングの設定

Red Hat build of Keycloak は、JBoss Logging フレームワークを使用します。以下は、使用可能なログハンドラーの概要です。

- root
  - console (**デフォルト**)
  - file

13.1. ロギング設定

ロギングは、Red Hat build of Keycloak でカテゴリーごとに行われます。ロギングは、ルートログレベル、または **org.hibernate** や **org.keycloak** などのより具体的なカテゴリーで設定できます。この章では、ロギングの設定方法について説明します。

13.1.1. ログレベル

次の表は、使用可能なログレベルを定義しています。

レベル	説明
FATAL	いかなる種類の要求にもまったく対応できない致命的な障害。
ERROR	要求を処理できなくなる重大なエラーまたは問題。
WARN	即時の修正を必要としない場合もある、致命的ではないエラーまたは問題。
INFO	Red Hat build of Keycloak のライフサイクルイベントまたは重要な情報。低頻度で発生。
DEBUG	データベースログなど、デバッグ目的の詳細情報。高頻度で発生。
TRACE	最も詳細なデバッグ情報。非常に高い頻度で発生。
ALL	すべてのログメッセージ向けの特別なレベル。
OFF	ログを完全にオフにする特別なレベル (推奨されません)。

13.1.2. ルートログレベルを設定する

より具体的なカテゴリーロガーのログレベル設定が存在しない場合は、代わりにそれを含むカテゴリーが使用されます。それを含むカテゴリーがない場合は、ルートロガーレベルが使用されます。

ルートログレベルを設定するには、次のコマンドを入力します。

■

```
bin/kc.[sh|bat] start --log-level=<root-level>
```

このコマンドには、次のガイドラインを使用してください。

- **<root-level>** には、前述の表で定義されたレベルを指定します。
- ログレベルで大文字と小文字は区別されません。たとえば、**DEBUG** または **debug** を使用できます。
- 誤ってログレベルを 2 回設定してしまった場合、リストの最後に指定されたログレベルになります。たとえば、**--log-level="info,...,DEBUG,..."** の構文を含めた場合、ルートロガーは **DEBUG** になります。

13.1.3. カテゴリー固有のログレベルを設定する

Red Hat build of Keycloak では、特定の領域に異なるログレベルを設定できます。このコマンドを使用すると、別のログレベルが必要なカテゴリーをコンマ区切りリストで指定できます。

```
bin/kc.[sh|bat] start --log-level="<root-level>,<org.category1>:<org.category1-level>"
```

カテゴリーに適用される設定は、より具体的な一致するサブカテゴリーを含めない限り、そのサブカテゴリーにも適用されます。

例

```
bin/kc.[sh|bat] start --log-level="INFO,org.hibernate:debug,org.hibernate.hql.internal.ast:info"
```

この例では、次のようにログレベルを設定します。

- すべてのロガーのルートログレベルは INFO に設定されます。
- 通常、ハイバーネートログレベルは DEBUG に設定されます。
- SQL 抽象構文ツリーが詳細なログ出力を作成しないようにするために、特定のサブカテゴリー **org.hibernate.hql.internal.ast** が INFO に設定されます。その結果、SQL 抽象構文ツリーは **debug** レベルでは表示されずに省略されます。

13.2. ログハンドラーを有効にする

ログハンドラーを有効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --log="<handler1>,<handler2>"
```

使用可能なハンドラーは、**console** と **file** です。後述する、より具体的なハンドラー設定は、ハンドラーがこのコンマ区切りリストに追加された場合にのみ有効になります。

13.3. コンソールログハンドラー

コンソールログハンドラーはデフォルトで有効になっており、コンソール用に構造化されていないログメッセージを提供します。

13.3.1. コンソールログ形式を設定する

Red Hat build of Keycloak は、デフォルトで人間が判読できるテキストログを生成するパターンベースのロギングフォーマッターを使用します。

これらの行のログ形式テンプレートは、ルートレベルで適用できます。デフォルトの形式テンプレートは次のとおりです。

- **%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n**

この形式の文字列では、下表の記号がサポートされています。

記号	概要	説明
%%	%	単純な % 文字をレンダリングします。
%c	カテゴリー	ログカテゴリー名をレンダリングします。
%d{xxx}	日付	指定された日付形式の文字列で日付をレンダリングします。**java.text.SimpleDateFormat** で定義された文字列構文。
%e	例外	出力された例外をレンダリングします。
%h	Hostname	単純なホスト名をレンダリングします。
%H	修飾ホスト名	完全修飾ホスト名をレンダリングします。OS 設定によっては、単純なホスト名と同じになる場合があります。
%i	プロセス ID	現在のプロセスの PID をレンダリングします。
%m	フルメッセージ	出力された場合は、ログメッセージと例外をレンダリングします。
%n	改行	プラットフォーム固有の行区切り文字列をレンダリングします。
%N	プロセス名	現在のプロセスの名前をレンダリングします。
%p	レベル	メッセージのログレベルをレンダリングします。

記号	概要	説明
%r	相対時間	アプリケーションログの開始からの相対時間 (ミリ秒単位) をレンダリングします。
%s	単純なメッセージ	例外トレースのないログメッセージをレンダリングします。
%t	スレッド名	スレッド名をレンダリングします。
%t{id}	スレッド ID	スレッド ID をレンダリングします。
%z{<zone name>}	タイムゾーン	ログ出力のタイムゾーンを <zone name> に設定します。
%L	行番号	ログメッセージの行番号をレンダリングします。

13.3.2. ロギング形式を設定する

ログに記録される行のログ形式を設定するには、次の手順を実行します。

1. 前述の表を使用して、形式テンプレートを作成します。
2. 以下のコマンドを入力します。

```
bin/kc.[sh|bat] start --log-console-format="'<format>'"
```

; などの特殊なシェル文字を含むコマンドを呼び出す場合は、CLI を使用して文字をエスケープする必要があります。代わりに、設定ファイルで設定することを検討してください。

例: 完全修飾カテゴリー名を短縮する

```
bin/kc.[sh|bat] start --log-console-format="'%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n'"
```

この例では、テンプレートでデフォルトの **[%c]** の代わりに **[%c{3.}]** を設定することで、カテゴリー名を 3 文字に短縮します。

13.3.3. JSON またはプレーンコンソールのロギングを設定する

デフォルトでは、コンソールログハンドラーはプレーンな非構造化データをコンソールに記録します。代わりに構造化された JSON ログ出力を使用するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --log-console-output=json
```

ログメッセージの例

{"timestamp":"2022-02-25T10:31:32.452+01:00","sequence":8442,"loggerClassName":"org.jboss.logging.Logger","loggerName":"io.quarkus","level":"INFO","message":"Keycloak 18.0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.2.Final) started in 3.253s. Listening on: http://0.0.0.0:8080","threadName":"main","threadId":1,"mdc":{},"ndc":"","hostName":"host-name","processName":"QuarkusEntryPoint","processId":36946}

JSON 出力を使用する場合、色は無効になり、**--log-console-format** で設定された形式設定は適用されません。

非構造化ロギングを使用するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --log-console-output=default
```

ログメッセージの例

```
2022-03-02 10:36:50,603 INFO  [io.quarkus] (main) Keycloak 18.0.0-SNAPSHOT on JVM (powered by Quarkus 2.7.2.Final) started in 3.615s. Listening on: http://0.0.0.0:8080
```

13.3.4. 色

非構造化ログの色付きコンソールログ出力は、デフォルトでは無効になっています。色を使用すると読みやすくなりますが、ログを外部のログ集約システムに送信する際に問題が発生する可能性があります。色分けされたコンソールログ出力を有効または無効にするには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --log-console-color=<false|true>
```

13.4. ファイルロギング

コンソールにログを記録する代わりに、ファイルへの非構造化ロギングを使用できます。

13.4.1. ファイルロギングを有効にする

ファイルへのロギングは、デフォルトで無効になっています。有効にするには、以下のコマンドを入力します。

```
bin/kc.[sh|bat] start --log="console,file"
```

keycloak.log という名前のログファイルが、Red Hat build of Keycloak の **data/log** ディレクトリー内に作成されます。

13.4.2. ログファイルの場所と名前を設定する

ログファイルの作成場所とファイル名を変更するには、次の手順を実行します。

1. ログファイルを保存するための書き込み可能なディレクトリーを作成します。
   ディレクトリーが書き込み可能でない場合、Red Hat build of Keycloak は正しく起動しますが、エラーが発生してログファイルは作成されません。
2. 以下のコマンドを入力します。

   ```
   bin/kc.[sh|bat] start --log="console,file" --log-file=<path-to>/<your-file.log>
   ```

13.4.3. ファイルハンドラーの形式を設定する

ファイルログハンドラーに別のログ形式を設定するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --log-file-format="<pattern>"
```

詳細と利用可能なパターン設定の表は、「コンソールログ形式を設定する」 を参照してください。

13.5. 関連するオプション

	値
log コンマ区切りリストで 1 つ以上のログハンドラーを有効にします。 CLI: **--log** Env: **KC_LOG**	**console**、**file**
log-console-color コンソールへのログイン時に、色を有効または無効にします。 CLI: **--log-console-color** Env: **KC_LOG_CONSOLE_COLOR**	**true**、**false** (デフォルト)
log-console-format 非構造化コンソールログエントリーの形式。 形式にスペースが含まれている場合は、"<format>" を使用して値をエスケープします。 CLI: **--log-console-format** Env: **KC_LOG_CONSOLE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (デフォルト)
log-console-output ログ出力を、JSON またはデフォルトの (プレーン) 非構造化ロギングに設定します。 CLI: **--log-console-output** Env: **KC_LOG_CONSOLE_OUTPUT**	**default** (デフォルト)、**json**
log-file ログファイルのパスとファイル名を設定します。 CLI: **--log-file** Env: **KC_LOG_FILE**	**data/log/keycloak.log** (デフォルト)

	値
log-file-format ファイルログエントリーに固有の形式を設定します。 CLI: **--log-file-format** Env: **KC_LOG_FILE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (デフォルト)
log-file-output ログ出力を、JSON またはデフォルトの (プレーン) 非構造化ロギングに設定します。 CLI: **--log-file-output** Env: **KC_LOG_FILE_OUTPUT**	**default** (デフォルト)、**json**
log-level ルートカテゴリーのログレベル、または個々のカテゴリーとそのレベルのコンマ区切りリスト。 ルートカテゴリーの場合、カテゴリーを指定する必要はありません。 CLI: **--log-level** Env: **KC_LOG_LEVEL**	**[info]** (デフォルト)

第14章 FIPS 140-2 サポート

Federal Information Processing Standard Publication 140-2 (FIPS 140-2) は、暗号化モジュールを承認するために使用される米国政府のコンピューターセキュリティー標準です。Red Hat build of Keycloak は FIPS 140-2 準拠モードでの実行をサポートしています。この場合、Red Hat build of Keycloak は、その機能に FIPS で承認されている暗号化アルゴリズムのみを使用します。

FIPS 140-2 で実行するには、Red Hat build of Keycloak が FIPS 140-2 対応システム上で実行されていなければなりません。この要件は通常、インストール時に FIPS が有効化された RHEL または Fedora を前提としています。詳細は、RHEL のドキュメント を参照してください。システムが FIPS モードの場合、基盤となる OpenJDK も FIPS モードであることが確認され、FIPS 対応のセキュリティープロバイダー のみが使用されます。

システムが FIPS モードであることを確認するには、コマンドラインから次のコマンドを使用します。

```
fips-mode-setup --check
```

システムが FIPS モードではない場合、次のコマンドを使用して有効にできます。ただし、この方法で後から有効にするのではなく、インストール時からシステムを FIPS モードにすることが推奨されます。

```
fips-mode-setup --enable
```

14.1. BOUNCYCASTLE ライブラリー

Red Hat build of Keycloak の内部では、多くの暗号化ユーティリティーに BouncyCastle ライブラリーが使用されています。Red Hat build of Keycloak に同梱されている BouncyCastle ライブラリーのデフォルトバージョンは FIPS に準拠していないことに注意してください。ただし、BouncyCastle には FIPS 検証済みバージョンのライブラリーもあります。FIPS 検証済みの BouncyCastle ライブラリーは、ライセンスの制約により Red Hat build of Keycloak に同梱できず、Red Hat build of Keycloak がその公式サポートを提供することもできません。したがって、FIPS 準拠モードで実行するには、BouncyCastle-FIPS ビットをダウンロードし、それを Red Hat build of Keycloak のディストリビューションに追加する必要があります。Red Hat build of Keycloak を FIPS モードで実行すると、デフォルトの BouncyCastle ビットの代わりに BCFIPS ビットが使用されます。これにより、FIPS 準拠が実現します。

14.1.1. BouncyCastle FIPS ビット

BouncyCastle FIPS は、BouncyCastle の公式ページ からダウンロードできます。その後、使用しているディストリビューションの **KEYCLOAK_HOME/providers** ディレクトリーにそれらを追加できます。Red Hat build of Keycloak の BouncyCastle 依存関係と互換性のある適切なバージョンを使用してください。サポートされている、必要な BCFIPS ビットは次のとおりです。

- **bc-fips-1.0.2.3.jar**
- **bctls-fips-1.0.18.jar**
- **bcpkix-fips-1.0.7.jar**

14.2. キーストアを生成する

Red Hat build of Keycloak サーバーの SSLで使用する **pkcs12** または **bcfks** キーストアを作成できます。

14.2.1. PKCS12 キーストア

p12 (または **pkcs12**) キーストア (および/またはトラストストア) は、BCFIPS 非承認モードで適切に機能します。

PKCS12 キーストアは、RHEL 9 上の OpenJDK 17 Java を使用して、標準的な方法で生成できます。たとえば、次のコマンドを使用してキーストアを生成できます。

```
keytool -genkeypair -sigalg SHA512withRSA -keyalg RSA -storepass passwordpassword \
  -keystore $KEYCLOAK_HOME/conf/server.keystore \
  -alias localhost \
  -dname CN=localhost -keypass passwordpassword
```

システムが FIPS モードの場合、FIPS 対応のセキュリティープロバイダーを使用するためにデフォルトの **java.security** ファイルが変更されるため、追加の設定は必要ありません。さらに、PKCS12 キーストアでは、keytool コマンドを使用して簡単に PBE (パスワードベースの暗号化) キーを保存できます。そのため、このキーストアは、Red Hat build of Keycloak KeyStore Vault とともに使用したり、KeyStore 設定ソースに設定プロパティーを保存するために使用するのに最適です。詳細は、Red Hat build of Keycloak の設定 および vault を使用する を参照してください。

14.2.2. BCFKS キーストア

BCFKS キーストアを生成するには、BouncyCastle FIPS ライブラリーとカスタムセキュリティーファイルを使用する必要があります。

まず、**/tmp/kc.keystore-create.java.security** などのヘルパーファイルを作成します。ファイルの内容としては、次のプロパティーのみ必要です。

```
securerandom.strongAlgorithms=PKCS11:SunPKCS11-NSS-FIPS
```

次に、次のようなコマンドを入力してキーストアを生成します。

```
keytool -keystore $KEYCLOAK_HOME/conf/server.keystore \
  -storetype bcfks \
  -providername BCFIPS \
  -providerclass org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider \
  -provider org.bouncycastle.jcajce.provider.BouncyCastleFipsProvider \
  -providerpath $KEYCLOAK_HOME/providers/bc-fips-*.jar \
  -alias localhost \
  -genkeypair -sigalg SHA512withRSA -keyalg RSA -storepass passwordpassword \
  -dname CN=localhost -keypass passwordpassword \
  -J-Djava.security.properties=/tmp/kc.keystore-create.java.security
```

警告

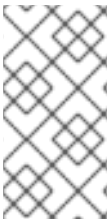
自己署名付き証明書はデモンストレーション目的に限定して使用しているため、実稼働環境に移行する際にこれらの証明書を適切な証明書に置き換えてください。

bcfks タイプのキーストア/トラストストアを使用して他の操作を行う場合も、同様のオプションが必要です。

14.3. サーバーを実行する

非承認モードで BCFIPS を使用してサーバーを実行するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --features=fips --hostname=localhost --https-key-store-
password=passwordpassword --log-
level=INFO,org.keycloak.common.crypto:TRACE,org.keycloak.crypto:TRACE
```



注記

非承認モードでは、デフォルトのキーストアタイプ (およびデフォルトのトラストストアタイプ) は PKCS12 です。したがって、上記のように BCFKS キーストアを生成した場合は、コマンド **--https-key-store-type=bcfks** も使用する必要があります。トラストストアを使用する場合も、同様のコマンドが必要になる場合があります。

注記

すべてが期待どおりに動作する場合は、実稼働環境でのログインを無効にできます。

14.4. STRICT モード

fips-mode オプションがあります。**fips** 機能が有効になっている場合、これは自動的に **non-strict** に設定されます。これは、BCFIPS が "非承認モード" で実行されることを意味します。よりセキュアな代替方法として、**--features=fips --fips-mode=strict** を使用できます。この場合、BouncyCastle FIPS は "承認モード" を使用します。このオプションを使用すると、暗号化とセキュリティーアルゴリズムに対するセキュリティー要件が厳しくなります。

注記

strict モードでは、デフォルトのキーストアタイプ (およびデフォルトのトラストストアタイプ) は BCFKS です。別のキーストアタイプを使用する場合は、オプション **--https-key-store-type** を使用して適切なタイプを指定する必要があります。トラストストアを使用する場合も、同様のコマンドが必要になる場合があります。

サーバーを起動すると、起動ログに次のような **Approved Mode** に関する注釈とともに **KC** プロバイダーが含まれていることを確認できます。

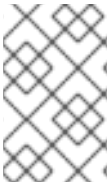
```
KC(BCFIPS version 1.000203 Approved Mode, FIPS-JVM: enabled) version 1.0 - class
org.keycloak.crypto.fips.KeycloakFipsSecurityProvider,
```

14.4.1. strict モードでの暗号化の制限

- 前のセクションで説明したように、strict モードは **pkcs12** キーストアでは機能しない可能性があります。前述のように、別のキーストア (**bcfks** など) を使用する必要があります。また、strict モードを使用している場合、**jks** および **pkcs12** キーストアは Red Hat build of Keycloak ではサポートされません。たとえば、管理コンソールの OIDC または SAML クライアントのキーストア、もしくはレルムキーの **ava-keystore** プロバイダーのキーストアのインポートや生成などです。

- ユーザーパスワードは、14 文字以上でなければなりません。Red Hat build of Keycloak は、デフォルトで PBKDF2 ベースのパスワードエンコーディングを使用します。BCFIPS 承認モードでは、PBKDF2 アルゴリズムを使用した 112 ビット (実質的には 14 文字) 以上のパスワードが必要です。それよりも短いパスワードを許可する場合は、SPI **password-hashing** のプロバイダー **pbkdf2-sha256** のプロパティー **max-padding-length** で値を 14 に設定して、このアルゴリズムによって作成されたハッシュの検証時に追加のパディングを提供します。この設定は、以前に保存されたパスワードとの下位互換性もあります。たとえば、ユーザーのデータベースが非 FIPS 環境にあり、パスワードが短く、承認モードで BCFIPS を使用して Red Hat build of Keycloak でパスワードを検証する場合、そのパスワードは機能するはずです。したがって、サーバーの起動時に次のようなオプションを効果的に使用できます。

```
--spi-password-hashing-pbkdf2-sha256-max-padding-length=14
```



注記

上記のオプションを使用しても、FIPS 準拠は損なわれません。いずれにせよ、パスワードは長くすることが推奨されます。たとえば、最新のブラウザーで自動生成されるパスワードは 14 文字を超えるため、この要件に一致します。

- 1024 ビットの RSA キーは機能しません (最小は 2048)。これは、Red Hat build of Keycloak レルム自体が使用するキー (管理コンソールの **Keys** のレルムキー) だけでなく、クライアントキーと IDP キーにもあてはまります。

- HMAC SHA-XXX キーは、112 ビット (または 14 文字) 以上でなければなりません。たとえば、OIDC クライアントをクライアント認証 **Signed Jwt with Client Secret** (OIDC 表記では **client-secret-jwt**) で使用する場合、クライアントシークレットの長さは 14 文字以上である必要があります。優れたセキュリティーを確保するには、この要件が必ず満たされるように、Red Hat build of Keycloak サーバーによって生成されたクライアントシークレットを使用することを推奨します。

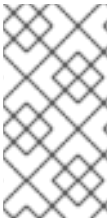
14.5. その他の制限

SAML が機能するためには、**XMLDSig** セキュリティープロバイダーがセキュリティープロバイダーで利用できることを確認する必要があります。Kerberos が機能するためには、**SunJGSS** セキュリティープロバイダーが利用できることを確認する必要があります。OpenJDK 17.0.6 の FIPS 対応 RHEL 9 では、これらのセキュリティープロバイダーは **java.security** に存在しないため、事実上は機能しません。

SAML が機能するためには、プロバイダーを **JAVA_HOME/conf/security/java.security** FIPS プロバイダーリストに手動で追加します。たとえば、次のような行を追加します。

```
fips.provider.7=XMLDSig
```

セキュリティープロバイダーを追加すると、正常に機能するはずです。実際、これは FIPS に準拠しており、今後の OpenJDK 17 マイクロバージョンではデフォルトで追加される可能性があります。詳細は、Bugzilla を参照してください。



注記

JAVA_HOME/conf/security/java.security で、設定済みのすべてのプラバイダーを確認し、番号が一致することを確認することが推奨されます。言い換えると、**fips.provider.7** は、このファイル内に **fips.provider.N** のような接頭辞が設定されたプロバイダーがすでに 6 つあることを前提としています。

Java 内の **java.security** ファイルを編集しない場合は、カスタム Java セキュリティーファイル (たとえば、**kc.java.security** という名前で) を作成し、そのファイルに XMLDSig プロバイダーを追加するための上記のプロパティーを 1 つだけ追加できます。その後、このプロパティーファイルをアタッチして Red Hat build of Keycloak サーバーを起動します。

```
-Djava.security.properties=/location/to/your/file/kc.java.security
```

Kerberos/SPNEGO の場合、セキュリティープロバイダー **SunJGSS** はまだ完全には FIPS に準拠していません。したがって、FIPS に準拠する必要がある場合は、それをセキュリティープロバイダーリストに追加することは推奨されません。FIPS プラットフォームで実行され、セキュリティープロバイダーが使用できない場合、Red Hat build of Keycloak ではデフォルトで **KERBEROS** 機能が無効になっています。詳細は、Bugzilla を参照してください。

14.6. FIPS ホストで CLI を実行する

クライアント登録 CLI (**kcreg.sh|bat** スクリプト) または管理 CLI (**kcadm.sh|bat** スクリプト) を実行する場合、CLI はプレーンな BouncyCastle 依存関係の代わりに BouncyCastle FIPS 依存関係も使用する必要があります。そのために必要なのは、jar を CLI ライブラリーフォルダーにコピーすることだけです。CLI ツールは、対応する BCFIPS jar が存在することを検出すると、プレーン BC の代わりに BCFIPS 依存関係を自動的に使用します (使用されるバージョンについては上記を参照)。たとえば、CLI を実行する前に次のようなコマンドを使用します。

```
cp $KEYCLOAK_HOME/providers/bc-fips-*.jar $KEYCLOAK_HOME/bin/client/lib/
cp $KEYCLOAK_HOME/providers/bctls-fips-*.jar $KEYCLOAK_HOME/bin/client/lib/
```

注記

CLI で BCFKS トラストストア/キーストアを使用しようとすると、このトラストストアがデフォルトの Java キーストアタイプではないために問題が発生することがあります。その場合は、Java セキュリティープロパティーでデフォルトとして指定できます。たとえば、kcadm|kcreg クライアントで操作を行う前に、unix ベースのシステムで次のコマンドを実行します。

```
echo "keystore.type=bcfks
fips.keystore.type=bcfks" > /tmp/kcadm.java.security
export KC_OPTS="-Djava.security.properties=/tmp/kcadm.java.security"
```

14.7. コンテナー内での FIPS モードの RED HAT BUILD OF KEYCLOAK サーバー

コンテナー内で Red Hat build of Keycloak を FIPS モードで 実行する場合は、"ホスト" も FIPS モードを使用する必要があります。その後、コンテナーは親ホストから FIPS モードを "継承" します。詳細は、RHEL ドキュメントの このセクション を参照してください。

Red Hat build of Keycloak のコンテナーイメージは、FIPS モードのホストから実行されると自動的に FIPS モードになります。ただし、Red Hat build of Keycloak コンテナーも起動時に (BC jar ではなく) BCFIPS jar と適切なオプションを使用していることを確認してください。

これに関しては、コンテナー内で Red Hat build of Keycloak を実行する で説明されているように独自のコンテナーイメージをビルドし、BCFIPS などを使用するように調整することが最適です。

たとえば、現在のディレクトリーにサブディレクトリー **files** を作成し、以下を追加できます。

- 前述の BC FIPS jar ファイル
- カスタムキーストアファイル (名前の例: **keycloak-fips.keystore.bcfks**)
- SAML のプロバイダーが追加された **kc.java.security** という名前のセキュリティーファイル

次に、現在のディレクトリーに次のような **Dockerfile** を作成します。

Dockerfile:

```
FROM registry.redhat.io/rhbk/keycloak-rhel9:24 as builder

ADD files /tmp/files/

WORKDIR /opt/keycloak
RUN cp /tmp/files/*.jar /opt/keycloak/providers/
RUN cp /tmp/files/keycloak-fips.keystore.* /opt/keycloak/conf/server.keystore
RUN cp /tmp/files/kc.java.security /opt/keycloak/conf/

RUN /opt/keycloak/bin/kc.sh build --features=fips --fips-mode=strict

FROM registry.redhat.io/rhbk/keycloak-rhel9:24
COPY --from=builder /opt/keycloak/ /opt/keycloak/

ENTRYPOINT ["/opt/keycloak/bin/kc.sh"]
```

次に、Red Hat build of Keycloak をコンテナー内で実行する の説明に従って、FIPS を最適化された Docker イメージとしてビルドし、起動します。これらの手順では、イメージを起動する際に上記のように因数を使用する必要があります。

14.8. 非 FIPS 環境からの移行

以前に Red Hat build of Keycloak を非 FIPS 環境で使用していた場合は、そのデータを含めて FIPS 環境に移行できます。ただし、前のセクションで述べたように、次のような制限と考慮事項が存在します。

- キーストアに依存するすべての Red Hat build of Keycloak 機能が、サポートされているキーストアタイプのみを使用していることを確認してください。これは、strict モードと non-strict モードのどちらが使用されているかによりことなります。
- Kerberos 認証が機能しない可能性があります。認証フローで **Kerberos** オーセンティケーターを使用している場合、FIPS 環境に以降すると、そのオーセンティケーターは自動的に **DISABLED** に切り替わります。FIPS 環境に切り替える前に、レルムから **Kerberos** ユーザーストレージプロバイダーを削除し、LDAP プロバイダーの **Kerberos** 関連機能を無効にすることが推奨されます。

FIPS strict モードに切り替える前に、前述の要件に加えて、次の点を必ず再確認してください。

- キー (レルムキーやクライアントキーなど) に依存するすべての Red Hat build of Keycloak 機能が、2048 ビット以上の RSA Red Hat build of Keycloakを使用していることを確認します。
- **Signed JWT with Client Secret** に依存するクライアントが、長さが 14 文字以上のシークレット (理想的には生成されたシークレット) を使用していることを確認します。
- 前述したパスワードの長さの制限。ユーザーのパスワードがこれよりも短い場合は、前述したように、最大パディング長が 14 に設定された PBKDF2 プロバイダーを使用してサーバーを起動します。この方法を避ける場合は、たとえば全ユーザーに、新しい環境での初回認証時に

(**Forget password** リンクを使用するなどして) パスワードをリセットするように依頼できます。

14.9. 非 FIPS システム上の RED HAT BUILD OF KEYCLOAK FIPS モード

Red Hat build of Keycloak は、FIPS 対応の RHEL 8 システムおよび **ubi8** イメージでサポートされ、テストされています。RHEL 9 (および **ubi9** イメージ) でもサポートされています。RHEL 非互換プラットフォームまたは FIPS 非対応プラットフォームで実行している場合、FIPS 準拠が厳格に保証されることはなく、正式にサポートされません。

そのようなシステム上で Red Hat build of Keycloak を実行するように制限されている場合でも、**java.security** ファイルで設定されているセキュリティープロバイダーを更新することはできます。それを更新しても、FIPS 準拠には至りませんが、少なくともそれに近づきます。そのためには、前述したように、オーバーライドされたセキュリティープロバイダーのリストのみを含むカスタムセキュリティーファイルを提供します。推奨プロバイダーのリストは、OpenJDK 17 のドキュメント を参照してください。

起動時に Red Hat build of Keycloak サーバーログで、正しいセキュリティープロバイダーが使用されているか確認できます。前述の Keycloak 起動コマンドで説明したように、暗号化関連の Red Hat build of Keycloak パッケージに対して TRACE ロギングが有効になっている必要があります。

第15章 RED HAT BUILD OF KEYCLOAK のヘルスチェックを有効にする

Red Hat build of Keycloak には、ヘルスチェックのサポートが組み込まれています。この章では、Red Hat build of Keycloak のヘルスチェックを有効にして使用する方法について説明します。

15.1. RED HAT BUILD OF KEYCLOAK のヘルスチェックエンドポイント

Red Hat build of Keycloak は、次の 4 つのヘルスエンドポイントを公開します。

- **/health/live**
- **/health/ready**
- **/health/started**
- **/health**

各エンドポイントの意味については、Quarkus SmallRye Health のドキュメント を参照してください。

これらのエンドポイントは、次のような JSON オブジェクトにより、成功した場合は HTTP ステータス **200 OK**、失敗した場合は **503 Service Unavailable** で応答します。

追加のチェックごとの情報を含まないエンドポイントの成功応答:

```
{
    "status": "UP",
    "checks": []
}
```

データベース接続に関する情報を含むエンドポイントの成功応答:

```
{
    "status": "UP",
    "checks": [
        {
            "name": "Keycloak database connections health check",
            "status": "UP"
        }
    ]
}
```

15.2. ヘルスチェックを有効にする

ビルド時に **health-enabled** オプションを使用して、ヘルスチェックを有効にできます。

```
bin/kc.[sh|bat] build --health-enabled=true
```

デフォルトでは、ヘルスエンドポイントからチェックは返されません。

15.3. ヘルスチェックを使用する

ヘルスエンドポイントは、外部 HTTP 要求でモニタリングすることが推奨されます。セキュリティー対策として、Red Hat build of Keycloak のコンテナーイメージから **curl** とその他のパッケージを削除しているため、するセキュリティー対策のため、ローカルのコマンドベースのモニタリングは容易には機能しません。

コンテナーで Red Hat build of Keycloak を使用していない場合は、任意の手段でヘルスチェックエンドポイントにアクセスできます。

15.3.1. curl

シンプルな HTTP HEAD 要求を使用して、Red Hat build of Keycloak の状態が **live** か **ready** かを判断できます。**curl** は、この目的に適した HTTP クライアントです。

Red Hat build of Keycloak がコンテナーにデプロイされている場合は、前述のセキュリティー対策があるため、このコマンドをコンテナーの外部から実行する必要があります。以下に例を示します。

```
curl --head -fsS http://localhost:8080/health/ready
```

コマンドがステータス 0 を返した場合、呼び出したエンドポイントに応じて Red Hat build of Keycloak は **live** または **ready** になります。それ以外の場合は問題があります。

15.3.2. Kubernetes

Kubernetes が外部からヘルスエンドポイントをモニタリングできるように、HTTP Probe を定義します。liveness コマンドは使用しないでください。

15.3.3. HEALTHCHECK

Dockerfile イメージの **HEALTHCHECK** 命令は、コンテナーの実行中にコンテナー内で定期的に実行されるコマンドを定義します。Red Hat build of Keycloak コンテナーには、CLI HTTP クライアントがインストールされていません。コンテナーで Red Hat build of Keycloak を実行する で詳しく説明されているように、追加の RPM として **curl** をインストールすることを検討してください。これにより、コンテナーの安全性が低下する可能性がある点に注意してください。

15.4. 利用可能なチェック

下表は、使用可能なチェックを示しています。

チェック	説明	メトリクスの要否
Database	データベース接続プールのステータスを返します。	はい

一部のチェックでは、**Requires Metrics** (メトリクスの要否) の列で示されているとおり、メトリクスを有効にする必要があります。メトリクスを有効にするには、次のように **metrics-enabled** オプションを使用します。

```
bin/kc.[sh|bat] build --health-enabled=true --metrics-enabled=true
```

15.5. 関連するオプション

	値
health-enabled ▮ サーバーがヘルスチェックエンドポイントを公開する必要があるかどうか。 有効にすると、**/health**、**/health/ready**、および **/health/live** エンドポイントでヘルスチェックが利用可能になります。 CLI: **--health-enabled** Env: **KC_HEALTH_ENABLED**	**true**、**false** (デフォルト)

第16章 RED HAT BUILD OF KEYCLOAK のメトリクスを有効にする

Red Hat build of Keycloak には、メトリクスのサポートが組み込まれています。この章では、サーバーメトリクスを有効にし、設定する方法について説明します。

16.1. メトリクスを有効にする

ビルド時に **metrics-enabled** オプションを使用して、メトリクスを有効にできます。

```
bin/kc.[sh|bat] start --metrics-enabled=true
```

16.2. メトリクスのクエリー

Red Hat build of Keycloak は、次のエンドポイントでメトリクスを公開します。

- **/metrics**

エンドポイントからの応答は、**application/openmetrics-text** コンテンツタイプを使用し、Prometheus (OpenMetrics) テキスト形式に基づいています。以下は、応答例の抜粋です。

```
# HELP base_gc_total Displays the total number of collections that have occurred. This attribute lists -1 if the collection count is undefined for this collector.
# TYPE base_gc_total counter
base_gc_total{name="G1 Young Generation",} 14.0
# HELP jvm_memory_usage_after_gc_percent The percentage of long-lived heap pool used after the last GC event, in the range [0..1]
# TYPE jvm_memory_usage_after_gc_percent gauge
jvm_memory_usage_after_gc_percent{area="heap",pool="long-lived",} 0.0
# HELP jvm_threads_peak_threads The peak live thread count since the Java virtual machine started or peak was reset
# TYPE jvm_threads_peak_threads gauge
jvm_threads_peak_threads 113.0
# HELP agroal_active_count Number of active connections. These connections are in use and not available to be acquired.
# TYPE agroal_active_count gauge
agroal_active_count{datasource="default",} 0.0
# HELP base_memory_maxHeap_bytes Displays the maximum amount of memory, in bytes, that can be used for memory management.
# TYPE base_memory_maxHeap_bytes gauge
base_memory_maxHeap_bytes 1.6781410304E10
# HELP process_start_time_seconds Start time of the process since unix epoch.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.675188449054E9
# HELP system_load_average_1m The sum of the number of runnable entities queued to available processors and the number of runnable entities running on the available processors averaged over a period of time
# TYPE system_load_average_1m gauge
system_load_average_1m 4.005859375

...
```

16.3. 利用可能なメトリクス

下表は、使用可能なメトリクスグループをまとめたものです。

メトリクス	説明
システム	CPU とメモリーの使用量に関連するシステムレベルのメトリクスセット。
JVM	GC およびヒープに関連する Java 仮想マシン (JVM) からのメトリクスセット。
Database	データベースを使用している場合は、データベース接続プールからのメトリクスセット。
Cache	Infinispan キャッシュからのメトリクスセット。詳細は、分散キャッシュの設定 を参照してください。

16.4. 関連するオプション

	値
metrics-enabled サーバーがメトリクスを公開する必要があるかどうか。 有効にすると、**/metrics** エンドポイントでメトリクスを利用できるようになります。 CLI: **--metrics-enabled** Env: **KC_METRICS_ENABLED**	**true**、**false** (デフォルト)

第17章 レルムのインポートとエクスポート

この章では、JSON ファイルを使用してレルムをインポートおよびエクスポートするさまざまな方法を説明します。

注記

単一ファイルにエクスポートおよびインポートすると大きなファイルが作成される可能性があるため、データベースに 500 ユーザーを超えるユーザーが含まれている場合は、1 つのファイルではなく、ディレクトリーにエクスポートしてください。ディレクトリープロバイダーは "ページ" (ユーザーのファイル) ごとに個別のトランザクションを使用するため、ディレクトリーを使用するとパフォーマンスが向上します。ファイルごとおよびトランザクションごとのデフォルトのユーザー数は 50 です。この値を大きくすると、実行時間が指数関数的に増加します。

17.1. データベース接続パラメーターのオプションを指定する

以下の **export** および **import** コマンドを使用する場合、Red Hat build of Keycloak は、レルム、クライアント、ユーザー、およびその他のエンティティーに関する情報が保存されているデータベースに接続する方法を認識している必要があります。Red Hat build of Keycloak の設定 で説明されているように、その情報はコマンドラインパラメーター、環境変数、または設定ファイルとして提供できます。使用可能なオプションを確認するには、各コマンドに対して **--help** コマンドラインオプションを使用します。

設定オプションの一部は、ビルド時の設定オプションです。デフォルトでは、Red Hat build of Keycloak は、ビルド時のパラメーターにおける変更を検出すると、**export** および **import** コマンドに対して自動的に再ビルドします。

Red Hat build of Keycloak の設定 で説明されているように、**build** コマンドを使用して Red Hat build of Keycloak の最適化バージョンをビルドした場合は、コマンドラインオプション **--optimized** を使用して、起動時間を短縮するために Red Hat build of Keycloak がビルドチェックをスキップするようにします。その際には、コマンドラインからビルド時オプションを削除し、実行時オプションのみを保持します。

17.2. レルムをディレクトリーにエクスポートする

レルムをエクスポートするには、**export** コマンドを使用します。このコマンドを呼び出すときに、Red Hat build of Keycloak サーバーインスタンスを開始しないでください。

```
bin/kc.[sh|bat] export --help
```

レルムをディレクトリーにエクスポートするには、**--dir <dir>** オプションを使用できます。

```
bin/kc.[sh|bat] export --dir <dir>
```

レルムをディレクトリーにエクスポートすると、サーバーはエクスポートされるレルムごとに個別のファイルを作成します。

17.2.1. ユーザーのエクスポート方法を設定する

--users <strategy> オプションを設定することで、ユーザーのエクスポート方法も設定できます。このオプションで使用できる値は次のとおりです。

different_files

--users-per-file で設定したファイルあたりの最大ユーザー数に応じて、ユーザーを別々の json ファイルにエクスポートします。これはデフォルト値です。

skip

ユーザーのエクスポートをスキップします。

realm_file

ユーザーをレルム設定と同じファイルにエクスポートします。"foo" という名前のレルムの場合、レルムデータとユーザーを含む "foo-realm.json" になります。

same_file

すべてのユーザーを 1 つの明示的なファイルにエクスポートします。したがって、1 つのレルムに対して 2 つの json ファイル (レルムデータを含むファイルとユーザーを含むファイルを 1 つずつ) を取得することになります。

different_files ストラテジーを使用してユーザーをエクスポートしている場合は、**--users-per-file** オプションを設定することで、ファイルごとに必要なユーザー数を設定できます。デフォルト値は **50** です。

```
bin/kc.[sh|bat] export --dir <dir> --users different_files --users-per-file 100
```

17.3. レルムをファイルにエクスポートする

レルムをファイルにエクスポートするには、**--file <file>** オプションを使用できます。

```
bin/kc.[sh|bat] export --file <file>
```

レルムをファイルにエクスポートする場合、サーバーは同じファイルを使用して、エクスポートされるすべてのレルムの設定を保存します。

17.4. 特定のレルムをエクスポートする

エクスポートする特定のレルムを指定しない場合、すべてのレルムがエクスポートされます。単一のレルムをエクスポートする場合、次のように **--realm** オプションを使用できます。

```
bin/kc.[sh|bat] export [--dir|--file] <path> --realm my-realm
```

17.5. ディレクトリーからレルムをインポートする

レルムをインポートするには、**import** コマンドを使用できます。このコマンドを呼び出すときに、Red Hat build of Keycloak サーバーインスタンスを開始しないでください。

```
bin/kc.[sh|bat] import --help
```

レルムをディレクトリーにエクスポートした後、次のように **--dir <dir>** オプションを使用してレルムをサーバーにインポートし直すことができます。

```
bin/kc.[sh|bat] import --dir <dir>
```

import コマンドを使用してレルムをインポートする場合、既存のレルムをスキップするかどうか、または新しい設定で既存のレルムをオーバーライドするかどうかを設定できます。そのためには、次のように **--override** オプションを設定します。

```
bin/kc.[sh|bat] import --dir <dir> --override false
```

デフォルトでは、**--override** オプションは **true** に設定されているため、レルムは常に新しい設定でオーバーライドされます。

17.6. ファイルからレルムをインポートする

以前に単一のファイルでエクスポートしたレルムをインポートするには、次のように **--file <file>** オプションを使用できます。

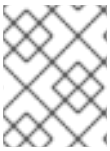
```
bin/kc.[sh|bat] import --file <file>
```

17.7. 起動時にレルムをインポートする

サーバーの起動時に、**--import-realm** オプションを使用してレルムをインポートすることもできます。

```
bin/kc.[sh|bat] start --import-realm
```

--import-realm オプションを設定すると、サーバーは **data/import** ディレクトリーからレルム設定ファイルをインポートしようとします。このディレクトリーからは **.json** 拡張子を使用する通常のファイルのみが読み取られ、サブディレクトリーは無視されます。



注記

Red Hat build of Keycloak コンテナーの場合、インポートディレクトリーは **/opt/keycloak/data/import** です。

すでにレルムがサーバーに存在する場合、インポート操作はスキップされます。この動作は、主にレルムの再作成を回避することを目的としており、サーバーを再起動すると状態が遷移する可能性があります。

レルムを再作成するには、サーバーを起動する前に **import** コマンドを明示的に実行する必要があります。

master レルムのインポートは非常に機密性の高い操作であるため、サポートされていません。

17.7.1. レルム設定ファイル内で環境変数を使用する

起動時にレルムをインポートする場合、プレースホルダーを使用して、任意のレルム設定で環境変数の値を解決できます。

プレースホルダーを使用したレルム設定

```
{
    "realm": "${MY_REALM_NAME}",
    "enabled": true,
    ...
}
```

上記の例では、**MY_REALM_NAME** 環境変数に設定された値が **realm** プロパティーの設定に使用されます。

17.8. 管理コンソールを使用したインポートとエクスポート

管理コンソールを使用してレルムをインポートおよびエクスポートすることもできます。この機能は、前のセクションで説明した他の CLI オプションとは異なります。管理コンソールで使用できるのは、レルムを **部分的に** エクスポートする機能だけであるためです。この場合、現在のレルム設定と、クライアント、ロール、グループなどの一部のリソースをエクスポートできます。この方法では、そのレルムのユーザーをエクスポートすることは **できません**。

注記

管理コンソールのエクスポートを使用する場合、レルムと選択したリソースが、常に **realm-export.json** という名前のファイルにエクスポートされます。また、パスワードやクライアントシークレットなどの機密の高い値が、すべて * 記号でマスクされます。

管理コンソールを使用してレルムをエクスポートするには、次の手順を実行します。

1. レルムを選択します。
2. メニューで **Realm Settings** をクリックします。
3. レルム設定画面の右上隅にある **Action** メニューに移動し、**Partial export** を選択します。レルム設定とともにリソースのリストが表示されます。
4. エクスポートするリソースを選択します。
5. **Export** をクリックします。

注記

管理コンソールからのレルムのエクスポートは、サーバー間のバックアップやデータ転送には適していません。サーバー間のバックアップまたはデータ転送には、CLI エクスポートのみを使用できます。

警告

レルムに多数のグループ、ロール、およびクライアントが含まれている場合、この操作により、サーバーがしばらくの間、ユーザーの要求に応答しなくなる可能性があります。特に本番環境ではこの機能を使用してください。

同様の方法で、以前にエクスポートしたレルムをインポートすることもできます。以下の手順を実行します。

1. メニューで **Realm Settings** をクリックします。
2. レルム設定画面の右上隅にある **Action** メニューに移動し、**Partial import** を選択します。

インポートするファイルを選択できるプロンプトが表示されます。このファイルに基づいて、レルム設定とともにインポートできるリソースが表示されます。

3. **Import** をクリックします。

インポートするリソースがすでに存在する場合の Red Hat build of Keycloak の動作を制御することもできます。以下のオプションがあります。

Fail import

インポートを中断します。

Skip

プロセスを中断せずに重複リソースをスキップします。

Overwrite

既存のリソースをインポートするリソースに置き換えます。

注記

管理コンソールの部分的なインポートでは、CLI **export** コマンドによって作成されたファイルもインポートできます。つまり、CLI で作成した完全なエクスポートを、管理コンソールを使用してインポートできます。ファイルにユーザーが含まれている場合、そのユーザーも現在のレルムにインポートできます。

第18章 VAULT を使用する

Red Hat build of Keycloak は、すぐに使用できる Vault SPI 実装を 2 つ提供しています。それが、プレーンテキストファイルベースの vault と Java KeyStore ベースの Vault です。

ファイルベースの vault 実装は、Kubernetes/OpenShift シークレットで特に役立ちます。Kubernetes シークレットを Red Hat build of Keycloak コンテナーにマウントでき、データフィールドはフラットファイル構造のマウントされたフォルダーで使用可能になります。

Java KeyStore ベースの vault 実装は、ベアメタルインストールにシークレットを保存するのに役立ちます。パスワードを使用して暗号化された KeyStore vault を使用できます。

18.1. 利用可能な統合

vault に保存されたシークレットは、管理コンソールの次の場所で使用できます。

- SMTP メールサーバーのパスワードを取得します。
- LDAP ベースのユーザーフェデレーションを使用する場合に LDAP バインド認証情報を取得します。
- 外部アイデンティティープロバイダーを統合するときに OIDC アイデンティティープロバイダーのクライアントシークレットを取得します。

18.2. VAULT を有効にする

ファイルベースの vault を有効にするには、まず次のビルドオプションを使用して Red Hat build of Keycloak をビルドする必要があります。

```
bin/kc.[sh|bat] build --vault=file
```

同様に、Java KeyStore ベースの場合は、次のビルドオプションを指定する必要があります。

```
bin/kc.[sh|bat] build --vault=keystore
```

18.3. ファイルベースの VAULT を設定する

18.3.1. シークレット検索に使用するベースディレクトリーを設定する

Kubernetes/OpenShift シークレットは、基本的にマウントされたファイルです。これらのファイルをマウントするディレクトリーを設定するには、次のコマンドを入力します。

```
bin/kc.[sh|bat] start --vault-dir=/my/path
```

18.3.2. レルム固有のシークレットファイル

Kubernetes/OpenShift シークレットは、Red Hat build of Keycloak でレルムごとに使用されるため、ファイルの命名規則を適切に設定する必要があります。

```
${vault.<realmname>_<secretname>}
```

18.3.3. 名前にアンダースコアを使用する

シークレットを正しく処理するには、<realmname> または <secretname> 内のすべてのアンダースコアを それぞれ 2 つにし、1 つのアンダースコアで区切ります。

例

- レルム名: **sso_realm**
- 予定している名前: **ldap_credential**
- 結果のファイル名:

```
sso__realm_ldap__credential
```

sso と **realm** の間、および **ldap** と **credential** の間にアンダースコアが 2 つあることに注意してください。

18.4. JAVA KEYSTORE ベースの VAULT を設定する

Java KeyStore ベースの vault を使用するには、最初に KeyStore ファイルを作成する必要があります。これを行うには、次のコマンドを使用できます。

```
keytool -importpass -alias <realm-name>_<alias> -keystore keystore.p12 -storepass
keystorepassword
```

次に、vault に保存する値を入力します。**-alias** パラメーターの形式は、使用するキーリゾルバーによって異なることに注意してください。デフォルトのキーリゾルバーは **REALM_UNDERSCORE_KEY** です。

これにより、デフォルトでは、SecretKeyEntry 内の汎用 PBEKey (パスワードベースの暗号化) の形式で値が保存されます。

その後、次のランタイムオプションを使用して Red Hat build of Keycloak を起動できます。

```
bin/kc.[sh|bat] start  --vault-file=/path/to/keystore.p12 --vault-pass=<value> --vault-type=<value>
```

--vault-type パラメーターはオプションであり、デフォルトは **PKCS12** であることに注意してください。

vault に保存されているシークレットは、プレースホルダー **${vault.realm-name_alias}** を介してレルム内でアクセスできます (**REALM_UNDERSCORE_KEY** キーリゾルバーを使用していると仮定)。

18.5. 例: 管理コンソールで LDAP バインド認証情報シークレットを使用する

セットアップ例

- 名前が **Secrettest** のレルム
- バインド認証情報の名前は **ldapBc**
- 結果のファイル名: **Secrettest_ldapBc**

管理コンソールでの使用法

その後、LDAP ユーザーフェデレーションを設定する際に **Bind Credential** の値として **${vault.ldapBc}** を使用することで、管理コンソールからこのシークレットを使用できます。

18.6. 関連するオプション

	値
vault ▮ vault プロバイダーを有効にします。 CLI: **--vault** Env: **KC_VAULT**	**file**、**keystore**
vault-dir これを設定すると、指定されたディレクトリー内のファイルの内容を読み取ることでシークレットを取得できます。 CLI: **--vault-dir** Env: **KC_VAULT_DIR**	
vault-file キーストアファイルへのパス。 CLI: **--vault-file** Env: **KC_VAULT_FILE**	
vault-pass vault キーストアのパスワード。 CLI: **--vault-pass** Env: **KC_VAULT_PASS**	
vault-type キーストアファイルの型を指定します。 CLI: **--vault-type** Env: **KC_VAULT_TYPE**	**PKCS12** (デフォルト)

第19章 すべての設定

19.1. CACHE

	値
cache ▮ 高可用性のためのキャッシュメカニズムを定義します。 実稼働モードのデフォルトでは、複数のサーバーノード間のクラスターは **ispn** キャッシュを使用して作成されます。開発モードのデフォルトでは、**local** キャッシュはクラスタリングを無効にし、開発とテスト目的で使用されます。 CLI: **--cache** Env: **KC_CACHE**	**ispn** (デフォルト)、**local**
cache-config-file ▮ キャッシュ設定のロード元となるファイルを定義します。 設定ファイルは **conf/** ディレクトリーに相対です。 CLI: **--cache-config-file** Env: **KC_CACHE_CONFIG_FILE**	
cache-embedded-mtls-enabled Keycloak サーバー間のネットワーク通信を暗号化します。 CLI: **--cache-embedded-mtls-enabled** Env: **KC_CACHE_EMBEDDED_MTLS_ENABLED**	**true**、**false** (デフォルト)
cache-embedded-mtls-key-store-file キーストアファイルのパス。 キーストアには、TLS プロトコルが使用する証明書が含まれている必要があります。デフォルトでは、conf/ ディレクトリーの下にある **cache-mtls-keystore.p12** が検索されます。 CLI: **--cache-embedded-mtls-key-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_FILE**	
cache-embedded-mtls-key-store-password キーストアにアクセスするためのパスワード。 CLI: **--cache-embedded-mtls-key-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_KEY_STORE_PASSWORD**	

	値
cache-embedded-mtls-trust-store-file トラストストアファイルのパス。 トラストストアには、信頼済み証明書または証明書に署名した認証局が含まれている必要があります。デフォルトでは、conf/ ディレクトリーの下にある **cache-mtls-truststore.p12** が検索されます。 CLI: **--cache-embedded-mtls-trust-store-file** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_FILE**	
cache-embedded-mtls-trust-store-password トラストストアにアクセスするためのパスワード。 CLI: **--cache-embedded-mtls-trust-store-password** Env: **KC_CACHE_EMBEDDED_MTLS_TRUST_STORE_PASSWORD**	
cache-remote-host リモートストア設定のリモートサーバーのホスト名。 これは、XML ファイルで指定された設定の **remote-server** タグの **host** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-username** と **cache-remote-password** も必要となり、XML ファイル内に関連する設定を含めることができません。 CLI: **--cache-remote-host** Env: **KC_CACHE_REMOTE_HOST**	
cache-remote-password リモートストアのリモートサーバーへの認証に使用するパスワード。 これは、XML ファイルで指定された設定の **digest** タグの **password** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-host** と **cache-remote-username** も必要となり、XML ファイル内に関連する設定を含めることができません。 CLI: **--cache-remote-password** Env: **KC_CACHE_REMOTE_PASSWORD**	
cache-remote-port リモートストア設定用のリモートサーバーのポート。 これは、XML ファイルで指定された設定の **remote-server** タグの **port** 属性を置き換えます (**cache-config-file** オプションを参照)。 CLI: **--cache-remote-port** Env: **KC_CACHE_REMOTE_PORT**	**11222** (デフォルト)

	値
cache-remote-username リモートストアのリモートサーバーへの認証に使用するユーザー名。 これは、XML ファイルで指定された設定の **digest** タグの **username** 属性を置き換えます (**cache-config-file** オプションを参照)。このオプションを指定する場合、**cache-remote-host** と **cache-remote-password** も必要となり、XML ファイル内に関連する設定を含めることができません。 CLI: **--cache-remote-username** Env: **KC_CACHE_REMOTE_USERNAME**	
cache-stack ▮ クラスター通信とノード検出に使用するデフォルトのスタックを定義します。 このオプションは、**cache** が **ispn** に設定されている場合にのみ有効です。デフォルト: udp. CLI: **--cache-stack** Env: **KC_CACHE_STACK**	**tcp**, **udp**, **kubernetes**, **ec2**, **azure**, **google**

19.2. DATABASE

	値
db ▮ データベースベンダー CLI: **--db** Env: **KC_DB**	**dev-file** (デフォルト)、**dev-mem**、**mariadb**、**mssql**、**mysql**、**oracle**、**postgres**
db-driver ▮ JDBC ドライバーの完全修飾名。 設定されていない場合、選択したデータベースに応じてデフォルトのドライバーが設定されます。 CLI: **--db-driver** Env: **KC_DB_DRIVER**	
db-password データベースユーザーアカウントのパスワード CLI: **--db-password** Env: **KC_DB_PASSWORD**	

	値
db-pool-initial-size 接続プールの初期サイズ。 CLI: **--db-pool-initial-size** Env: **KC_DB_POOL_INITIAL_SIZE**	
db-pool-max-size 接続プールの最大サイズ。 CLI: **--db-pool-max-size** Env: **KC_DB_POOL_MAX_SIZE**	**100** (デフォルト)
db-pool-min-size 接続プールの最小サイズ。 CLI: **--db-pool-min-size** Env: **KC_DB_POOL_MIN_SIZE**	
db-schema 使用するデータベーススキーマ CLI: **--db-schema** Env: **KC_DB_SCHEMA**	
db-url データベースの完全な JDBC URL 指定しない場合、選択したデータベースベンダーに基づきデフォルト URL が設定されます。たとえば、**postgres** を使用している場合、デフォルトの JDBC URL は **jdbc:postgresql://localhost/keycloak** になります。 CLI: **--db-url** Env: **KC_DB_URL**	
db-url-database 選択したベンダーのデフォルト JDBC URL のデータベース名を設定します。 **db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-database** Env: **KC_DB_URL_DATABASE**	

	値
db-url-host 選択したベンダーのデフォルト JDBC URL のホスト名を設定します。 **db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-host** Env: **KC_DB_URL_HOST**	
db-url-port 選択したベンダーのデフォルト JDBC URL のポートを設定します。 **db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-port** Env: **KC_DB_URL_PORT**	
db-url-properties 選択したベンダーのデフォルト JDBC URL のプロパティーを設定します。 データベースベンダーが想定する形式に従ってプロパティーを設定し、そのプロパティー値の先頭に正しい文字を追加してください。**db-url** オプションが設定されている場合、このオプションは無視されます。 CLI: **--db-url-properties** Env: **KC_DB_URL_PROPERTIES**	
db-username データベースユーザーのユーザー名 CLI: **--db-username** Env: **KC_DB_USERNAME**	

19.3. TRANSACTION

	値
transaction-xa-enabled ▮ false に設定すると、データベースが XA トランザクションをサポートしない場合に備えて、Keycloak は非 XA データソースを使用します。 CLI: **--transaction-xa-enabled** Env: **KC_TRANSACTION_XA_ENABLED**	**true** (デフォルト)、**false**

19.4. 機能

	値
features ▮ 1 つ以上の機能セットを有効にします。 CLI: **--features** Env: **KC_FEATURES**	**account-api[:v1]**, **account2[:v1]**, **account3[:v1]**, **admin-api[:v1]**, **admin-fine-grained-authz[:v1]**, **admin2[:v1]**, **authorization[:v1]**, **ciba[:v1]**, **client-policies[:v1]**, **client-secret-rotation[:v1]**, **client-types[:v1]**, **declarative-ui[:v1]**, **device-flow[:v1]**, **docker[:v1]**, **dpop[:v1]**, **dynamic-scopes[:v1]**, **fips[:v1]**, **hostname[:v1]**, **impersonation[:v1]**, **js-adapter[:v1]**, **kerberos[:v1]**, **linkedin-oauth[:v1]**, **login2[:v1]**, **multi-site[:v1]**, **offline-session-preloading[:v1]**, **oid4vc-vci[:v1]**, **par[:v1]**, **preview**, **recovery-codes[:v1]**, **scripts[:v1]**, **step-up-authentication[:v1]**, **token-exchange[:v1]**, **transient-users[:v1]**, **update-email[:v1]**, **web-authn[:v1]**

	値
features-disabled ▮ 1 つ以上の機能のセットを無効にします。 CLI: **--features-disabled** Env: **KC_FEATURES_DISABLED**	**account-api**, **account2**, **account3**, **admin-api**, **admin-fine-grained-authz**, **admin2**, **authorization**, **ciba**, **client-policies**, **client-secret-rotation**, **client-types**, **declarative-ui**, **device-flow**, **docker**, **dpop**, **dynamic-scopes**, **fips**, **impersonation**, **js-adapter**, **kerberos**, **linkedin-oauth**, **login2**, **multi-site**, **offline-session-preloading**, **oid4vc-vci**, **par**, **preview**, **recovery-codes**, **scripts**, **step-up-authentication**, **token-exchange**, **transient-users**, **update-email**, **web-authn**

19.5. HOSTNAME

	値
hostname Keycloak サーバーのホスト名 CLI: **--hostname** Env: **KC_HOSTNAME**	
hostname-admin 管理コンソールにアクセスするためのホスト名。 **hostname** オプションに設定された値以外のホスト名を使用して管理コンソールを公開する場合は、このオプションを使用します。 CLI: **--hostname-admin** Env: **KC_HOSTNAME_ADMIN**	

	値
hostname-admin-url 管理コンソールにアクセスするためのベース URL (スキーム、ホスト、ポート、パスなどを含む) を設定します。 CLI: **--hostname-admin-url** Env: **KC_HOSTNAME_ADMIN_URL**	
hostname-debug /realms/master/hostname-debug でアクセスできるホスト名のデバッグページを切り替えます。 CLI: **--hostname-debug** Env: **KC_HOSTNAME_DEBUG**	**true**、**false** (デフォルト)
hostname-path プロキシーが Keycloak のために異なるコンテキストパスを使用する場合は、これを設定する必要があります。 CLI: **--hostname-path** Env: **KC_HOSTNAME_PATH**	
hostname-port ホスト名を公開する際に、プロキシーが使用するポート。 プロキシーが、デフォルトの HTTP および HTTPS ポート以外のポートを使用する場合は、このオプションを設定します。 CLI: **--hostname-port** Env: **KC_HOSTNAME_PORT**	**-1** (デフォルト)
hostname-strict 要求ヘッダーからのホスト名の動的解決を無効にします。 プロキシーが Host ヘッダーを検証しない限り、実稼働環境では常に true に設定する必要があります。 CLI: **--hostname-strict** Env: **KC_HOSTNAME_STRICT**	**true** (デフォルト)、**false**

	値
hostname-strict-backchannel デフォルトでは、バックチャネル URL は要求ヘッダーから動的に解決され、内部および外部アプリケーションが許可されます。 すべてのアプリケーションがパブリック URL を使用する場合、このオプションを有効にする必要があります。 CLI: **--hostname-strict-backchannel** Env: **KC_HOSTNAME_STRICT_BACKCHANNEL**	**true**、**false** (デフォルト)
hostname-url フロントエンド URL のベース URL (スキーム、ホスト、ポート、パスなどを含む) を設定します。 CLI: **--hostname-url** Env: **KC_HOSTNAME_URL**	

19.6. HTTP(S)

	値
http-enabled HTTP リスナーを有効にします。 CLI: **--http-enabled** Env: **KC_HTTP_ENABLED**	**true**、**false** (デフォルト)
http-host 使用された HTTP ホスト。 CLI: **--http-host** Env: **KC_HTTP_HOST**	**0.0.0.0** (デフォルト)
http-max-queued-requests キューに入れる HTTP 要求の最大数。 これは過負荷状況で負荷を軽減するために使用します。要求数が超過すると、"503 Server not Available" という応答が返されます。 CLI: **--http-max-queued-requests** Env: **KC_HTTP_MAX_QUEUED_REQUESTS**	

	値
http-pool-max-threads 最大スレッド数。 これを指定しなかった場合、使用可能なプロセッサーの数を 8 倍した値と 200 のうちの大きい方に自動的にサイズ設定されます。たとえば、プロセッサーが 4 つの場合、最大スレッド数は 200 になります。プロセッサーが 48 個の場合、最大スレッド数は 384 になります。 CLI: **--http-pool-max-threads** Env: **KC_HTTP_POOL_MAX_THREADS**	
http-port 使用された HTTP ポート。 CLI: **--http-port** Env: **KC_HTTP_PORT**	**8080** (デフォルト)
http-relative-path ▮ リソースの提供に使用する、/ に相対するパスを設定します。 パスは / で始まる必要があります。 CLI: **--http-relative-path** Env: **KC_HTTP_RELATIVE_PATH**	/ (デフォルト)
https-certificate-file PEM 形式のサーバー証明書または証明書チェーンへのファイルパス。 CLI: **--https-certificate-file** Env: **KC_HTTPS_CERTIFICATE_FILE**	
https-certificate-key-file PEM 形式の秘密鍵へのファイルパス。 CLI: **--https-certificate-key-file** Env: **KC_HTTPS_CERTIFICATE_KEY_FILE**	
https-cipher-suites 使用する暗号スイート。 指定されていない場合、適切なデフォルトが選択されます。 CLI: **--https-cipher-suites** Env: **KC_HTTPS_CIPHER_SUITES**	

	値
https-client-auth ▮ クライアント認証を必要とする、または要求するようにサーバーを設定します。 CLI: **--https-client-auth** Env: **KC_HTTPS_CLIENT_AUTH**	**none** (デフォルト)、**request**、**required**
https-key-store-file 個別のファイルを指定する代わりに、証明書情報を保持するキーストア。 CLI: **--https-key-store-file** Env: **KC_HTTPS_KEY_STORE_FILE**	
https-key-store-password キーストアファイルのパスワード CLI: **--https-key-store-password** Env: **KC_HTTPS_KEY_STORE_PASSWORD**	**password** (デフォルト)
https-key-store-type キーストアファイルの型。 指定されていない場合、ファイル名に基き自動的に型が検出されます。**fips-mode** が **strict** に設定され、値が設定されていない場合、デフォルトの **BCFKS** になります。 CLI: **--https-key-store-type** Env: **KC_HTTPS_KEY_STORE_TYPE**	
https-port 使用される HTTPS ポート。 CLI: **--https-port** Env: **KC_HTTPS_PORT**	**8443** (デフォルト)
https-protocols 明示的に有効にするプロトコルのリスト。 CLI: **--https-protocols** Env: **KC_HTTPS_PROTOCOLS**	**[TLSv1.3, TLSv1.2]** (デフォルト)

	値
https-trust-store-file 信頼する証明書の証明書情報を保持するトラストストア。 CLI: **--https-trust-store-file** Env: **KC_HTTPS_TRUST_STORE_FILE** **非推奨。**代わりにシステムトラストストアを使用してください。詳細はドキュメントを参照してください。	
https-trust-store-password トラストストアファイルのパスワード。 CLI: **--https-trust-store-password** Env: **KC_HTTPS_TRUST_STORE_PASSWORD** **非推奨。**代わりにシステムトラストストアを使用してください。詳細はドキュメントを参照してください。	
https-trust-store-type トラストストアファイルの型。 指定されていない場合、ファイル名に基き自動的に型が検出されます。**fips-mode** が **strict** に設定され、値が設定されていない場合、デフォルトの **BCFKS** になります。 CLI: **--https-trust-store-type** Env: **KC_HTTPS_TRUST_STORE_TYPE** **非推奨。**代わりにシステムトラストストアを使用してください。詳細はドキュメントを参照してください。	

19.7. HEALTH

	値
health-enabled ▮ サーバーがヘルスチェックエンドポイントを公開する必要があるかどうか。 有効にすると、**/health**、**/health/ready**、および **/health/live** エンドポイントでヘルスチェックが利用可能になります。 CLI: **--health-enabled** Env: **KC_HEALTH_ENABLED**	**true**、**false** (デフォルト)

19.8. CONFIG

	値
config-keystore KeyStore 設定ソースへのパスを指定します。 CLI: **--config-keystore** Env: **KC_CONFIG_KEYSTORE**	
config-keystore-password KeyStore 設定ソースへのパスワードを指定します。 CLI: **--config-keystore-password** Env: **KC_CONFIG_KEYSTORE_PASSWORD**	
config-keystore-type KeyStore 設定ソースのタイプを指定します。 CLI: **--config-keystore-type** Env: **KC_CONFIG_KEYSTORE_TYPE**	**PKCS12** (デフォルト)

19.9. メトリクス

	値
metrics-enabled サーバーがメトリクスを公開する必要があるかどうか。 有効にすると、**/metrics** エンドポイントでメトリクスを利用できるようになります。 CLI: **--metrics-enabled** Env: **KC_METRICS_ENABLED**	**true**、**false** (デフォルト)

19.10. PROXY

	値
proxy サーバーがリバースプロキシーの背後にある場合のプロキシーアドレス転送モード。 CLI: **--proxy** Env: **KC_PROXY** **非推奨。proxy-headers** を使用してください。	**none** (デフォルト)、**edge**、**reencrypt**、**passthrough**

	値
proxy-headers サーバーが受け入れる必要があるプロキシーヘッダー。 設定を誤ると、サーバーがセキュリティー上の脆弱性にさらされる可能性があります。非推奨のプロキシーオプションよりも優先されます。 CLI: **--proxy-headers** Env: **KC_PROXY_HEADERS**	**forwarded、xforwarded**

19.11. VAULT

	値
vault vault プロバイダーを有効にします。 CLI: **--vault** Env: **KC_VAULT**	**file、keystore**
vault-dir これを設定すると、指定されたディレクトリー内のファイルの内容を読み取ることでシークレットを取得できます。 CLI: **--vault-dir** Env: **KC_VAULT_DIR**	
vault-file キーストアファイルへのパス。 CLI: **--vault-file** Env: **KC_VAULT_FILE**	
vault-pass vault キーストアのパスワード。 CLI: **--vault-pass** Env: **KC_VAULT_PASS**	
vault-type キーストアファイルの型を指定します。 CLI: **--vault-type** Env: **KC_VAULT_TYPE**	**PKCS12** (デフォルト)

19.12. LOGGING

	値
log コンマ区切りリストで1つ以上のログハンドラーを有効にします。 CLI: **--log** Env: **KC_LOG**	**console**、**file**
log-console-color コンソールへのログイン時に、色を有効または無効にします。 CLI: **--log-console-color** Env: **KC_LOG_CONSOLE_COLOR**	**true**、**false** (デフォルト)
log-console-format 非構造化コンソールログエントリーの形式。 形式にスペースが含まれている場合は、"<format>" を使用して値をエスケープします。 CLI: **--log-console-format** Env: **KC_LOG_CONSOLE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (デフォルト)
log-console-output ログ出力を、JSON またはデフォルトの (プレーン) 非構造化ロギングに設定します。 CLI: **--log-console-output** Env: **KC_LOG_CONSOLE_OUTPUT**	**default** (デフォルト)、**json**
log-file ログファイルのパスとファイル名を設定します。 CLI: **--log-file** Env: **KC_LOG_FILE**	**data/log/keycloak.log** (デフォルト)
log-file-format ファイルログエントリーに固有の形式を設定します。 CLI: **--log-file-format** Env: **KC_LOG_FILE_FORMAT**	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c] (%t) %s%e%n** (デフォルト)

	値
log-file-output ログ出力を、JSON またはデフォルトの (プレーン) 非構造化ロギングに設定します。 CLI: **--log-file-output** Env: **KC_LOG_FILE_OUTPUT**	**default** (デフォルト)、**json**
log-level ルートカテゴリーのログレベル、または個々のカテゴリーとそのレベルのコンマ区切りリスト。 ルートカテゴリーの場合、カテゴリーを指定する必要はありません。 CLI: **--log-level** Env: **KC_LOG_LEVEL**	**[info]** (デフォルト)

19.13. TRUSTSTORE

	値
tls-hostname-verifier 送信 HTTPS および SMTP 要求の TLS ホスト名検証ポリシー。 CLI: **--tls-hostname-verifier** Env: **KC_TLS_HOSTNAME_VERIFIER**	**ANY**、**WILDCARD** (デフォルト)、**STRICT**
truststore-paths システムトラストストアとして使用される pkcs12 (p12 または pfx ファイル拡張子)、PEM ファイル、またはそれらのファイルを含むディレクトリーのリスト。 CLI: **--truststore-paths** Env: **KC_TRUSTSTORE_PATHS**	

19.14. セキュリティー

	値
fips-mode ▮ FIPS モードを設定します。 **non-strict** が設定されている場合、FIPS は有効になりますが、非承認モードになります。FIPS に完全に準拠するには、承認モードで実行するように **strict** を設定します。このオプションは、**FIPS** 機能が無効な場合 (デフォルト)、デフォルトで **disabled** になります。このオプションは、**fips** 機能が有効な場合、デフォルトで **non-strict** になります。 CLI: **--fips-mode** Env: **KC_FIPS_MODE**	**non-strict**、**strict**

19.15. EXPORT

	値
dir エクスポートされたデータを含むファイルが作成されるディレクトリーへのパスを設定します。 CLI: **--dir** Env: **KC_DIR**	
realm エクスポートするレルムの名前を設定します。 設定されていない場合、すべてのレルムがエクスポートされます。 CLI: **--realm** Env: **KC_REALM**	
users ユーザーをエクスポートする方法を設定します。 CLI: **--users** Env: **KC_USERS**	**skip**、**realm_file**、**same_file**、**different_files** (デフォルト)

	値
users-per-file ファイルごとのユーザー数を設定します。 これは、**users** が **different_files** に設定されている場合にのみ使用されます。この数値を増やすと、エクスポート時間が急激に増加します。 CLI: **--users-per-file** Env: **KC_USERS_PER_FILE**	**50** (デフォルト)

19.16. インポート

	値
file 読み取られるファイルへのパスを設定します。 CLI: **--file** Env: **KC_FILE**	
override 既存のデータを上書きするかどうかを設定します。 false に設定すると、データは無視されます。 CLI: **--override** Env: **KC_OVERRIDE**	**true** (デフォルト)、**false**

第20章 すべてのプロバイダー設定

20.1. AUTHENTICATION-SESSIONS

20.1.1. infinispan

	値
spi-authentication-sessions-infinispan-auth-sessions-limit RootAuthenticationSession ごとの同時認証セッションの最大数。 CLI: **--spi-authentication-sessions-infinispan-auth-sessions-limit** Env: **KC_SPI_AUTHENTICATION_SESSIONS_INFINISPAN_AUTH_SESSIONS_LIMIT**	**300** (デフォルト) または任意の **int**

20.2. CIBA-AUTH-CHANNEL

20.2.1. ciba-http-auth-channel

	値
spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri 認証チャネルの HTTP(S) URI。 CLI: **--spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri** Env: **KC_SPI_CIBA_AUTH_CHANNEL_CIBA_HTTP_AUTH_CHANNEL_HTTP_AUTHENTICATION_CHANNEL_URI**	任意の **string**

20.3. CONNECTIONS-HTTP-CLIENT

20.3.1. default

	値
spi-connections-http-client-default-client-key-password キーのパスワード。 CLI: **--spi-connections-http-client-default-client-key-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEY_PASSWORD**	**-1** (デフォルト) または任意の **string**

	値
spi-connections-http-client-default-client-keystore TLS 接続をセットアップするためにキーマテリアルが読み取られるキーストアのファイルパス。 CLI: **--spi-connections-http-client-default-client-keystore** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE**	任意の **string**
spi-connections-http-client-default-client-keystore-password キーストアのパスワード。 CLI: **--spi-connections-http-client-default-client-keystore-password** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CLIENT_KEYSTORE_PASSWORD**	任意の **string**
spi-connections-http-client-default-connection-pool-size 合計接続数の最大値を割り当てます。 CLI: **--spi-connections-http-client-default-connection-pool-size** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_POOL_SIZE**	任意の **int**
spi-connections-http-client-default-connection-ttl-millis 永続的な接続の最大存続時間をミリ秒単位で設定します。 CLI: **--spi-connections-http-client-default-connection-ttl-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_TTL_MILLIS**	**-1** (デフォルト) または任意の **long**
spi-connections-http-client-default-disable-cookies 状態 (Cookie) の管理を無効にします。 CLI: **--spi-connections-http-client-default-disable-cookies** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_COOKIES**	**true** (デフォルト)、**false**

	値
spi-connections-http-client-default-disable-trust-manager 信頼管理とホスト名の検証を無効にします。 注記: これはセキュリティーホールです。通信しているホストのアイデンティティーを検証できない、または検証したくない場合にのみこのオプションを設定してください。 CLI: **--spi-connections-http-client-default-disable-trust-manager** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_DISABLE_TRUST_MANAGER**	**true**、**false** (デフォルト)
spi-connections-http-client-default-establish-connection-timeout-millis 最初のソケット接続を試みた場合のタイムアウトの長さは? CLI: **--spi-connections-http-client-default-establish-connection-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_ESTABLISH_CONNECTION_TIMEOUT_MILLIS**	**-1** (デフォルト) または任意の **long**
spi-connections-http-client-default-max-connection-idle-time-millis アイドル状態の接続をプールから削除する時間をミリ秒単位で設定します。 CLI: **--spi-connections-http-client-default-max-connection-idle-time-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_CONNECTION_IDLE_TIME_MILLIS**	**900000** (デフォルト) または任意の **long**
spi-connections-http-client-default-max-pooled-per-route ルートごとの最大接続値を割り当てます。 CLI: **--spi-connections-http-client-default-max-pooled-per-route** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_MAX_POOLED_PER_ROUTE**	**64** (デフォルト) または任意の **int**
spi-connections-http-client-default-proxy-mappings 正規表現ベースのホスト名パターンと proxy-uri の組み合わせを hostnamePattern;proxyUri の形式で示します。 CLI: **--spi-connections-http-client-default-proxy-mappings** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_PROXY_MAPPINGS**	任意の **string**

	値
spi-connections-http-client-default-reuse-connections 接続を再利用する必要があるかどうか。 CLI: **--spi-connections-http-client-default-reuse-connections** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_REUSE_CONNECTIONS**	**true** (デフォルト)、**false**
spi-connections-http-client-default-socket-timeout-millis ソケットの非アクティブタイムアウト。 CLI: **--spi-connections-http-client-default-socket-timeout-millis** Env: **KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_SOCKET_TIMEOUT_MILLIS**	**5000** (デフォルト) または任意の **long**

20.4. CONNECTIONS-INFINISPAN

20.4.1. quarkus

	値
spi-connections-infinispan-quarkus-site-name マルチサイトデプロイメントのサイト名 CLI: **--spi-connections-infinispan-quarkus-site-name** Env: **KC_SPI_CONNECTIONS_INFINISPAN_QUARKUS_SITE_NAME**	任意の **string**

20.5. CONNECTIONS-JPA

20.5.1. quarkus

	値
spi-connections-jpa-quarkus-initialize-empty 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 を手動に設定します。これにより、データベースを初期化するための SQL コマンドを含むファイルが作成されます。 CLI: **--spi-connections-jpa-quarkus-initialize-empty** Env: **KC_SPI_CONNECTIONS_JPA_QUARKUS_INITIALIZE_EMPTY**	**true** (デフォルト)、**false**
spi-connections-jpa-quarkus-migration-export 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。 CLI: **--spi-connections-jpa-quarkus-migration-export** Env: **KC_SPI_CONNECTIONS_JPA_QUARKUS_MIGRATION_EXPORT**	任意の **string**
spi-connections-jpa-quarkus-migration-strategy 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かどうかを確認します。 CLI: **--spi-connections-jpa-quarkus-migration-strategy** Env: **KC_SPI_CONNECTIONS_JPA_QUARKUS_MIGRATION_STRATEGY**	**update** (デフォルト)、**manual**、**validate**

20.6. COOKIE

20.6.1. default

	値
spi-cookie-default-same-site-legacy SameSite パラメーターなしでレガシー Cookie を追加します。 CLI: **--spi-cookie-default-same-site-legacy** Env: **KC_SPI_COOKIE_DEFAULT_SAME_SITE_LEGACY**	**true** (デフォルト)、**false**

20.7. DBLOCK

20.7.1. jpa

	値
spi-dblock-jpa-lock-wait-timeout データベースロックの解放を待機する際の最大待機時間。 CLI: **--spi-dblock-jpa-lock-wait-timeout** Env: **KC_SPI_DBLOCK_JPA_LOCK_WAIT_TIMEOUT**	任意の **int**

20.8. EVENTS-LISTENER

20.8.1. email

	値
spi-events-listener-email-exclude-events ユーザーのアカウントにメールで送信するべきではないイベントのコンマ区切りリスト。 CLI: **--spi-events-listener-email-exclude-events** Env: **KC_SPI_EVENTS_LISTENER_EMAIL_EXCLUDE_EVENTS**	**authreqid_to_token**, **authreqid_to_token_error**, **client_delete**, **client_delete_error**, **client_info**, **client_info_error**, **client_initiated_account_linking**, **client_initiated_account_linking_error**, **client_login**, **client_login_error**, **client_register**, **client_register_error**, **client_update**, **client_update_error**, **code_to_token**, **code_to_token_error**, **custom_required_action**, **custom_required_action_error**, **delete_account**, **delete_account_error**, **execute_action_token**, **execute_action_token_error**, **execute_actions**, **execute_actions_error**, **federated_identity_link**, **federated_identity_link_error**,

	値
	grant_consent, **grant_consent_error**, **identity_provider_first_login**, **identity_provider_first_login_error**, **identity_provider_link_account**, **identity_provider_link_account_error**, **identity_provider_login**, **identity_provider_login_error**, **identity_provider_post_login**, **identity_provider_post_login_error**, **identity_provider_response**, **identity_provider_response_error**, **identity_provider_retrieve_token**, **identity_provider_retrieve_token_error**, **impersonate**, **impersonate_error**, **introspect_token**, **introspect_token_error**, **invalid_signature**, **invalid_signature_error**, **login**, **login_error**, **logout**, **logout_error**, **oauth2_device_auth**, **oauth2_device_auth_error**, **oauth2_device_code_to_token**, **oauth2_device_code_to_token_error**, **oauth2_device_verify_user_code**, **oauth2_device_verify_user_code_error**, **oauth2_extension_grant**, **oauth2_extension_grant_error**, **permission_token**, **permission_token_error**, **pushed_authorization_request**, **pushed_authorization_request_error**,

	値
	refresh_token, refresh_token_error, register, register_error, register_node, register_node_error, remove_federated_identity, remove_federated_identity_error, remove_totp, remove_totp_error, reset_password, reset_password_error, restart_authentication, restart_authentication_error, revoke_grant, revoke_grant_error, send_identity_provider_link, send_identity_provider_link_error, send_reset_password, send_reset_password_error, send_verify_email, send_verify_email_error, token_exchange, token_exchange_error, unregister_node, unregister_node_error, update_consent, update_consent_error, update_email, update_email_error, update_password, update_password_error, update_profile, update_profile_error, update_totp, update_totp_error, user_disabled_by_permanent_lockout, user_disabled_by_permanent_lockout_error, user_disabled_by_temporary_lockout, user_disabled_by_temporary_lockout_error,

	値
	user_info_request, **user_info_request_error**, **validate_access_token**, **validate_access_token_error**, **verify_email**, **verify_email_error**, **verify_profile**, **verify_profile_error**
spi-events-listener-email-include-events ユーザーのアカウントにメールで送信する必要があるイベントのコンマ区切りリスト。 CLI: **--spi-events-listener-email-include-events** Env: **KC_SPI_EVENTS_LISTENER_EMAIL_INCLUDE_EVENTS**	**authreqid_to_token**, **authreqid_to_token_error**, **client_delete**, **client_delete_error**, **client_info**, **client_info_error**, **client_initiated_account_linking**, **client_initiated_account_linking_error**, **client_login**, **client_login_error**, **client_register**, **client_register_error**, **client_update**, **client_update_error**, **code_to_token**, **code_to_token_error**, **custom_required_action**, **custom_required_action_error**, **delete_account**, **delete_account_error**, **execute_action_token**, **execute_action_token_error**, **execute_actions**, **execute_actions_error**, **federated_identity_link**, **federated_identity_link_error**, **grant_consent**, **grant_consent_error**, **identity_provider_first_login**, **identity_provider_first_login_error**, **identity_provider_lin**

	値
	k_account, **identity_provider_link_account_error**, **identity_provider_login**, **identity_provider_login_error**, **identity_provider_post_login**, **identity_provider_post_login_error**, **identity_provider_response**, **identity_provider_response_error**, **identity_provider_retrieve_token**, **identity_provider_retrieve_token_error**, **impersonate**, **impersonate_error**, **introspect_token**, **introspect_token_error**, **invalid_signature**, **invalid_signature_error**, **login**, **login_error**, **logout**, **logout_error**, **oauth2_device_auth**, **oauth2_device_auth_error**, **oauth2_device_code_to_token**, **oauth2_device_code_to_token_error**, **oauth2_device_verify_user_code**, **oauth2_device_verify_user_code_error**, **oauth2_extension_grant**, **oauth2_extension_grant_error**, **permission_token**, **permission_token_error**, **pushed_authorization_request**, **pushed_authorization_request_error**, **refresh_token**, **refresh_token_error**, **register**, **register_error**, **register_node**, **register_node_error**, **remove_federated_i**

	値
	dentity, remove_federated_identity_error, remove_totp, remove_totp_error, reset_password, reset_password_error, restart_authentication, restart_authentication_error, revoke_grant, revoke_grant_error, send_identity_provider_link, send_identity_provider_link_error, send_reset_password, send_reset_password_error, send_verify_email, send_verify_email_error, token_exchange, token_exchange_error, unregister_node, unregister_node_error, update_consent, update_consent_error, update_email, update_email_error, update_password, update_password_error, update_profile, update_profile_error, update_totp, update_totp_error, user_disabled_by_permanent_lockout, user_disabled_by_permanent_lockout_error, user_disabled_by_temporary_lockout, user_disabled_by_temporary_lockout_error, user_info_request, user_info_request_error, validate_access_token, validate_access_token_error,

	値
	verify_email, **verify_email_error**, **verify_profile**, **verify_profile_error**

20.8.2. jboss-logging

	値
spi-events-listener-jboss-logging-error-level エラーメッセージのログレベル。 CLI: **--spi-events-listener-jboss-logging-error-level** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_ERROR_LEVEL**	**debug**、**error**、**fatal**、**info**、**trace**、**warn** (デフォルト)
spi-events-listener-jboss-logging-quotes 値に使用する引用符は、" または ' のように 1 文字にする必要があります。 引用符が必要ない場合は、none を使用します。 CLI: **--spi-events-listener-jboss-logging-quotes** 環境: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_QUOTES**	**''** (デフォルト) または任意の **文字列**
spi-events-listener-jboss-logging-sanitize true の場合、改行を避けるためにログメッセージがサニタイズされます。 誤ったメッセージがサニタイズされていない場合。 CLI: **--spi-events-listener-jboss-logging-sanitize** 環境: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_SANITIZE**	**true** (デフォルト)、**false**
spi-events-listener-jboss-logging-success-level 成功メッセージのログレベル。 CLI: **--spi-events-listener-jboss-logging-success-level** Env: **KC_SPI_EVENTS_LISTENER_JBOSS_LOGGING_SUCCESS_LEVEL**	**debug** (デフォルト)、**error**、**fatal**、**info**、**trace**、**warn**

20.9. EXPORT

20.9.1. dir

	値
spi-export-dir-dir エクスポート先のディレクトリー CLI: **--spi-export-dir-dir** Env: **KC_SPI_EXPORT_DIR_DIR**	任意の **string**
spi-export-dir-realm-name エクスポートするレルム CLI: **--spi-export-dir-realm-name** Env: **KC_SPI_EXPORT_DIR_REALM_NAME**	任意の **string**
spi-export-dir-users-export-strategy ユーザーのエクスポートストラテジー CLI: **--spi-export-dir-users-export-strategy** Env: **KC_SPI_EXPORT_DIR_USERS_EXPORT_STRATEGY**	**DIFFERENT_FILES** (デフォルト)、または任意の **string**
spi-export-dir-users-per-file エクスポートされたファイルごとのユーザー CLI: **--spi-export-dir-users-per-file** Env: **KC_SPI_EXPORT_DIR_USERS_PER_FILE**	**50** (デフォルト) または任意の **int**

20.9.2. single-file

	値
spi-export-single-file-file エクスポート先のファイル CLI: **--spi-export-single-file-file** Env: **KC_SPI_EXPORT_SINGLE_FILE_FILE**	任意の **string**
spi-export-single-file-realm-name エクスポートするレルム CLI: **--spi-export-single-file-realm-name** Env: **KC_SPI_EXPORT_SINGLE_FILE_REALM_NAME**	任意の **string**

20.10. IMPORT

20.10.1. dir

	値
spi-import-dir-dir インポート元のディレクトリー CLI: **--spi-import-dir-dir** Env: **KC_SPI_IMPORT_DIR_DIR**	任意の **string**
spi-import-dir-realm-name エクスポートするレルム CLI: **--spi-import-dir-realm-name** Env: **KC_SPI_IMPORT_DIR_REALM_NAME**	任意の **string**
spi-import-dir-strategy インポートストラテジー: IGNORE_EXISTING、OVERWRITE_EXISTING CLI: **--spi-import-dir-strategy** Env: **KC_SPI_IMPORT_DIR_STRATEGY**	任意の **string**

20.10.2. single-file

	値
spi-import-single-file-file インポート元のファイル CLI: **--spi-import-single-file-file** Env: **KC_SPI_IMPORT_SINGLE_FILE_FILE**	任意の **string**
spi-import-single-file-realm-name エクスポートするレルム CLI: **--spi-import-single-file-realm-name** Env: **KC_SPI_IMPORT_SINGLE_FILE_REALM_NAME**	任意の **string**
spi-import-single-file-strategy インポートストラテジー: IGNORE_EXISTING、OVERWRITE_EXISTING CLI: **--spi-import-single-file-strategy** Env: **KC_SPI_IMPORT_SINGLE_FILE_STRATEGY**	任意の **string**

20.11. PUBLIC-KEY-STORAGE

20.11.1. infinispan

	値
spi-public-key-storage-infinispan-max-cache-time すべての鍵方式によって鍵を取得したときに鍵をキャッシュする最大間隔 (秒単位)。 エントリーのすべての鍵を取得しても、(たとえば、鍵を ID によって取得する場合とは異なり) 鍵が欠落しているかどうかを検出する方法はありません。そのような場合、このオプションで、定期的に強制的に更新を行います。デフォルトは 24 時間です。 CLI: **--spi-public-key-storage-infinispan-max-cache-time** Env: **KC_SPI_PUBLIC_KEY_STORAGE_INFINISPAN_MAX_CACHE_TIME**	**86400** (デフォルト) または任意の **int**
spi-public-key-storage-infinispan-min-time-between-requests 新しい公開鍵を取得する 2 つの要求間の最小間隔 (秒単位)。 1 つの鍵が要求されても見つからなかった場合、サーバーは常に新しい公開鍵のダウンロードを試みます。ただし、前回の更新が 10 秒未満前に行われた場合は、ダウンロードが回避されます (デフォルト)。この動作は、外部の鍵エンドポイントに対する DoS 攻撃を回避するために使用されます。 CLI: **--spi-public-key-storage-infinispan-min-time-between-requests** Env: **KC_SPI_PUBLIC_KEY_STORAGE_INFINISPAN_MIN_TIME_BETWEEN_REQUESTS**	**10** (デフォルト) または任意の **int**

20.12. RESOURCE-ENCODING

20.12.1. gzip

	値
spi-resource-encoding-gzip-excluded-content-types エンコードから除外するコンテンツ型のスペース区切りリスト。 CLI: **--spi-resource-encoding-gzip-excluded-content-types** Env: **KC_SPI_RESOURCE_ENCODING_GZIP_EXCLUDED_CONTENT_TYPES**	**image/png image/jpeg** (デフォルト) または任意の **string**

20.13. STICKY-SESSION-ENCODER

20.13.1. infinispan

	値
spi-sticky-session-encoder-infinispan-should-attach-route 特定のセッションを所有するノードを反映するために、ルートを cookie にアタッチする必要があるかどうか。 CLI: **--spi-sticky-session-encoder-infinispan-should-attach-route** Env: **KC_SPI_STICKY_SESSION_ENCODER_INFINISPAN_SHOULD_ATTACH_ROUTE**	**true** (デフォルト)、**false**

20.14. TRUSTSTORE

20.14.1. file

	値
spi-truststore-file-file 非推奨: TLS 接続を検証するための証明書の読み取り先であるトラストストアのファイルパス。 CLI: **--spi-truststore-file-file** Env: **KC_SPI_TRUSTSTORE_FILE_FILE**	任意の **string**
spi-truststore-file-hostname-verification-policy 非推奨: ホスト名検証ポリシー。 CLI: **--spi-truststore-file-hostname-verification-policy** Env: **KC_SPI_TRUSTSTORE_FILE_HOSTNAME_VERIFICATION_POLICY**	**any**、**wildcard** (デフォルト)、**strict**
spi-truststore-file-password 非推奨: トラストストアのパスワード。 CLI: **--spi-truststore-file-password** Env: **KC_SPI_TRUSTSTORE_FILE_PASSWORD**	任意の **string**
spi-truststore-file-type 非推奨: トラストストアのタイプ。 指定しない場合、トラストストアのファイル拡張子またはデフォルトのプラットフォームタイプに基づきタイプが検出されます。 CLI: **--spi-truststore-file-type** Env: **KC_SPI_TRUSTSTORE_FILE_TYPE**	任意の **string**

20.15. USER-PROFILE

20.15.1. declarative-user-profile

	値
spi-user-profile-declarative-user-profile-admin-read-only-attributes 管理者が変更できないように、読み取り専用として扱う必要があるフィールドを識別するための正規表現の配列。 CLI: **--spi-user-profile-declarative-user-profile-admin-read-only-attributes** Env: **KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_ADMIN_READ_ONLY_ATTRIBUTES**	任意の **MultivaluedString**
spi-user-profile-declarative-user-profile-max-email-local-part-length ユーザープロファイルのメールローカル部分の最大長を設定します。 CLI: **--spi-user-profile-declarative-user-profile-max-email-local-part-length** Env: **KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_MAX_EMAIL_LOCAL_PART_LENGTH**	任意の **String**
spi-user-profile-declarative-user-profile-read-only-attributes ユーザーが変更できないように、読み取り専用として扱う必要があるフィールドを識別するための正規表現の配列。 CLI: **--spi-user-profile-declarative-user-profile-read-only-attributes** Env: **KC_SPI_USER_PROFILE_DECLARATIVE_USER_PROFILE_READ_ONLY_ATTRIBUTES**	任意の **MultivaluedString**

20.16. WELL-KNOWN

20.16.1. openid-configuration

	値
spi-well-known-openid-configuration-include-client-scopes サポートされるスコープのリストを計算するために、クライアントスコープを使用する必要があかどうか。 CLI: **--spi-well-known-openid-configuration-include-client-scopes** Env: **KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_INCLUDE_CLIENT_SCOPES**	**true** (デフォルト)、**false**
spi-well-known-openid-configuration-openid-configuration-override メタデータのロード元となるファイルパス。 絶対ファイルパスを使用できます。また、ファイルがサーバークラスパスにある場合は、**classpath:** を接頭辞として使用してクラスパスからファイルをロードすることもできます。 CLI: **--spi-well-known-openid-configuration-openid-configuration-override** Env: **KC_SPI_WELL_KNOWN_OPENID_CONFIGURATION_OPENID_CONFIGURATION_OVERRIDE**	任意の **string**