Red Hat build of Keycloak 24.0

移行ガイド

Last Updated: 2025-04-22

法律上の通知

概要

このガイドは Red Hat build of Keycloak の移行ガイドです。

目次

第1章 RED HAT SINGLE SIGN-ON 7.6 から RED HAT BUILD OF KEYCLOAK への移行

このガイドの目的は、Red Hat Single Sign-On 7.6 を Red Hat build of Keycloak 24.0 に正常に移行するために必要な手順を説明することです。この手順では、次の要素の移行を説明します。

- Red Hat Single Sign-On 7.6 サーバー
- OpenShift の Operator のデプロイメント
- OpenShift のテンプレートのデプロイメント
- Red Hat Single Sign-On 7.6 によって保護されたアプリケーション
- カスタムプロバイダー
- カスタムテーマ

このガイドには、アップストリーム版の Keycloak を Red Hat build of Keycloak 24.0 に移行するためのガイドラインも記載されています。移行を開始する前に、Red Hat build of Keycloak の新しいインスタンスをインストールして、このリリースの変更点を理解することをお勧めします。Red Hat build of Keycloak スタートガイド を参照してください。

第2章 RED HAT SINGLE SIGN-ON 7.6 サーバーの移行

このセクションでは、ZIP ディストリビューションからデプロイしたスタンドアロンサーバーを移行する手順を説明します。Red Hat build of Keycloak 24.0 は、Quarkus を使用して構築されています。Quarkus は、Red Hat Single Sign-On 7.6 で使用されていた Red Hat JBoss Enterprise Application Platform (JBoss EAP) に代わるものです。

サーバーに対する主な変更点は次のとおりです。

- 設定エクスペリエンスが刷新および合理化され、設定オプションの柔軟性が大きく向上しました。
- サーバーの RPM ディストリビューションが利用できなくなりました。

2.1. 前提条件

- Red Hat Single Sign-On 7.6 の以前のインスタンスが、Red Hat build of Keycloak と同じデータベースインスタンスを使用しないように、シャットダウンされている。
- データベースをバックアップした。
- OpenJDK17 がインストールされている。
- リリースノート を確認した。

2.2. 移行プロセスの概要

次のセクションでは、以下の内容の移行手順を説明します。

1. Red Hat build of Keycloak をダウンロードします。
2. 設定を移行します。
3. データベースを移行します。
4. Red Hat build of Keycloak サーバーを起動します。

2.3. RED HAT BUILD OF KEYCLOAK のダウンロード

Red Hat build of Keycloak サーバーのダウンロード ZIP ファイルには、Red Hat build of Keycloak サーバーを実行するためのスクリプトとバイナリーが含まれています。

1. Red Hat build of Keycloak サーバーのディストリビューションファイルを Red Hat カスタマーポータル からダウンロードします。
2. unzip コマンドを使用して ZIP ファイルを解凍します。

2.4. 設定の移行

Red Hat build of Keycloak サーバーを設定する統一された方法として、設定オプションが新しく導入されました。Red Hat Single Sign-On 7.6 の設定メカニズム (standalone.xml、jboss-cli など) は使用できなくなりました。

各オプションは、次の設定ソースを通じて定義できます。

- CLI パラメーター
- 環境変数
- 設定ファイル
- Java KeyStore ファイル

同じ設定オプションが別々の設定ソースで指定されている場合は、上記のリストの最初のソースが使用されます。

すべての設定オプションは、さまざまな設定ソースすべてで使用できますが、主な違いはキーの形式です。たとえば、データベースのホスト名を設定する 4 つの方法を次に示します。

ソース	形式
CLI パラメーター	--db-url-host cliValue
環境変数	KC_DB_URL_HOST=envVarValue
設定ファイル	db-url-host=confFileValue
Java KeyStore ファイル	kc.db-url-host=keystoreValue

kc.sh --help コマンドおよび Red Hat build of Keycloak ドキュメントに、使用可能なすべての設定オプションの詳細なリストが記載されています。オプションはキャッシュ、データベースなどのカテゴリーごとにグループ化されています。また、データベースの設定 の章など、設定する領域ごとに個別の章があります。

関連情報

- Keycloak の設定

2.4.1. データベース設定の移行

Red Hat Single Sign-On 7.6 とは対照的に、Red Hat build of Keycloak には、サポート対象データベースのサポートが組み込まれているため、データベースドライバーを手動でインストールして設定する必要がありません。例外は Oracle と Microsoft SQL Server です。これらについては、ドライバーを手動でインストールする必要があります。

設定に関しては、Red Hat Single Sign-On 7.6 の既存インストールのデータソースサブシステムを考慮し、その設定を、Red Hat build of Keycloak のデータベース設定カテゴリーで利用可能なオプションにマッピングします。たとえば、以前の設定として、次のようなものがあるとします。

```
<datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-name="KeycloakDS"
enabled="true" use-java-context="true" statistics-enabled="true">
    <connection-url>jdbc:postgresql://mypostgres:5432/mydb?
currentSchema=myschema</connection-url>
    <driver>postgresql</driver>
    <pool>
        <min-pool-size>5</min-pool-size>
        <max-pool-size>50</max-pool-size>
    </pool>
```

```
    <security>
      <user-name>myuser</user-name>
      <password>myuser</password>
    </security>
</datasource>
```

Red Hat build of Keycloak では、CLI パラメーターを使用した同等の設定は、次のようなものになります。

```
kc.sh start
  --db postgres
  --db-url-host mypostgres
  --db-url-port 5432
  --db-url-database mydb
  --db-schema myschema
  --db-pool-min-size 5 --db-pool-max-size 50
  --db-username myser --db-password myuser
```

注記

データベース認証情報は、セキュアな KeyStore 設定ソースに保存することを検討してください。

関連情報

- データベースの設定。Oracle および Microsoft SQL Server JDBC ドライバーのインストール手順も記載されています。
- Java KeyStore ファイルを使用した機密オプションの設定。データベース認証情報をセキュアに保存する方法が記載されています。

2.4.2. HTTP および TLS 設定の移行

実稼働モード (**start** オプションで表される) を使用する場合は常に、HTTP が無効になり、TLS 設定がデフォルトで必要になります。

--http-enabled=true 設定オプションを使用すると、HTTP を有効にできます。ただし、Red Hat build of Keycloak サーバーが完全に分離されたネットワーク内にあり、内部または外部の攻撃者がネットワークトラフィックを監視できるリスクがない場合を除き、このオプションの使用は推奨されません。

Red Hat build of Keycloak インスタンスは、サーバーのルートを使用するため、異なるコンテキストルート (URL パス) を持ちます。一方、Red Hat Single Sign-On 7.6 はデフォルトで **/auth** を追加します。以前の動作を再現するには、**--http-relative-path=/auth** 設定オプションを使用できます。デフォルトのポートは同じままです。ただし、**--http-port** および **--https-port** オプションによって変更することもできます。

TLS を設定するには 2 つの方法があります。PEM 形式のファイルを使用するか、Java キーストアを使用します。たとえば、Java キーストアによる以前の設定として、次のようなものがあるとします。

```
<tls>
 <key-stores>
          <key-store name="applicationKS">
              <credential-reference
  clear-text="password"/>
```

```
    <implementation type="JKS"/>
                <file
    path="/path/to/application.keystore”/>
            </key-store>
      </key-stores>
      <key-managers>
        <key-manager name="applicationKM"
         key-store="applicationKS">
              <credential-reference
          clear-text="password"/>
        </key-manager>
      </key-managers>
      <server-ssl-contexts>
         <server-ssl-context name="applicationSSC"
      key-manager="applicationKM"/>
      </server-ssl-contexts>
</tls>
```

Red Hat build of Keycloak では、CLI パラメーターを使用した同等の設定は、次のようなものになります。

```
kc.sh start
  --https-key-store-file /path/to/application.keystore
  --https-key-store-password password
```

Red Hat build of Keycloak では、次のように PEM 形式の証明書を指定することで TLS を設定できます。

```
kc.sh start
  --https-certificate-file /path/to/certfile.pem
  --https-certificate-key-file /path/to/keyfile.pem
```

関連情報

- TLS の設定

2.4.3. クラスタリングとキャッシュ設定の移行

Red Hat Single Sign-On 7.6 は、サーバーをスタンドアロン、スタンドアロンクラスター、およびドメインクラスターとして実行するための個別の動作モードを備えていました。これらのモードは、起動スクリプトと設定ファイルが異なります。Red Hat build of Keycloak では、単一の起動スクリプト **kc.sh** を使用したシンプルな方法を利用できます。

サーバーをスタンドアロンまたはクラスター化されたスタンドアロンとして実行するには、**kc.sh** スクリプトを使用します。

Red Hat build of Keycloak	Red Hat Single Sign-On 7.6
./kc.sh start --cache=local	./standalone.sh
./kc.sh start [--cache=ispn]	./standalone.sh --server-config=standalone-ha.xml

--cache パラメーターのデフォルト値は、起動モードを認識します。

- **local** - start-dev コマンドの実行時
- **ispn** - start コマンドの実行時

Red Hat Single Sign-On 7.6 では、クラスタリングとキャッシュの設定を Infinispan サブシステムを通じて行っていました。一方、Red Hat build of Keycloak では、設定の大部分を、別の Infinispan 設定ファイルを通じて行います。たとえば、Infinispan の以前の設定として、次のようなものがあるとします。

```
<subsystem xmlns="urn:jboss:domain:infinispan:13.0">
<cache-container name="keycloak" marshaller="JBOSS" modules="org.keycloak.keycloak-model-infinispan">
        <local-cache name="realms">
            <heap-memory size="10000"/>
        </local-cache>
        <local-cache name="users">
            <heap-memory size="10000"/>
        </local-cache>
        <local-cache name="sessions"/>
        <local-cache name="authenticationSessions"/>
        <local-cache name="offlineSessions"/>
  ...
</cache-container>
</subsystem>
```

Red Hat build of Keycloak では、デフォルトの Infinispan 設定ファイルは **conf/cache-ispn.xml** ファイルにあります。独自の Infinispan 設定ファイルを用意し、次のように CLI パラメーターを使用して指定できます。

```
kc.sh start --cache-config-file my-cache-file.xml
```

注記

ドメインクラスターモードは、Red Hat build of Keycloak ではサポートされていません。

トランスポートスタック

Red Hat build of Keycloak のデフォルトおよびカスタム JGroups トランスポートスタックを移行する必要はありません。

唯一の改善点は、CLI オプションのキャッシュスタックを指定することで、キャッシュ設定ファイルで定義されたスタックをオーバーライドできることです。指定すると、そのキャッシュスタックが優先されます。カスタム JGroups トランスポートスタックを使用した上記の Infinispan 設定ファイル **my-cache-file.xml** の一部として、次のようなものがあるとします。

keycloak キャッシュコンテナーのトランスポートスタックが tcp に設定されていますが、次のように CLI オプションを使用してこのスタックをオーバーライドできます。

```
<jgroups>
 <stack name=”my-encrypt-udp” extends=”udp”>
  …
 </stack>
</jgroups>
```

```
<cache-container name=”keycloak”>
 <transport stack=”tcp”/>
 …
</cache-container>
```

```
kc.sh start
  --cache-config-file my-cache-file.xml
  --cache-stack my-encrypt-udp
```

上記のコマンドを実行すると、**my-encrypt-udp** トランスポートスタックが使用されます。

関連情報

- 分散キャッシュの設定

2.4.4. ホスト名とプロキシー設定の移行

Red Hat build of Keycloak では、ホスト名 SPI の設定が必須です。これは、ユーザーをリダイレクトするとき、またはユーザーのクライアントと通信するときに、サーバーによってフロントエンド URL とバックエンド URL をどのように作成するかを設定するために必要です。

たとえば、Red Hat Single Sign-On 7.6 インストールに次のような設定があるとします。

```
<spi name="hostname">
    <default-provider>default</default-provider>
    <provider name="default" enabled="true">
        <properties>
          <property name="frontendUrl" value="myFrontendUrl"/>
          <property name="forceBackendUrlToFrontendUrl" value="true"/>
        </properties>
    </provider>
</spi>
```

Red Hat build of Keycloak では、これは次のような設定オプションに変換できます。

```
kc.sh start
   --hostname-url myFrontendUrl
   --hostname-strict-backchannel true
```

hostname-url 設定オプションを使用すると、クラスターの前で実行されているイングレス層からクラスターをパブリックに公開する場所となるベース URL を設定できます。**hostname-admin-url** 設定オプションを設定して、管理リソースの URL を設定することもできます。

Red Hat build of Keycloak を使用すると、どのリバースプロキシーヘッダーを反映するかを設定できます。**Forwarded** ヘッダーまたは **X-Forwarded-*** ヘッダーのセットのいずれかを使用できます。以下に例を示します。

```
kc.sh start --proxy-headers xforwarded
```

注記

ホスト名とプロキシー設定は、リソース URL (リダイレクト URI、CSS および JavaScript リンク、OIDC の既知のエンドポイントなど) を決定するために使用され、受信要求を積極的にブロックするために使用されるわけではありません。ホスト名/プロキシーオプションは、Red Hat build of Keycloak サーバーがリッスンする実際のバインディングアドレスやポートを変更しません。これは HTTP/TLS オプションの役割です。Red Hat Single Sign-On 7.6 では、ホスト名の設定が強く推奨されていましたが、強制はされていませんでした。Red Hat build of Keycloak では、start コマンドを使用する場合にホスト名の設定が必須になりました。ただし、**--hostname-strict=false** オプションで明示的に無効にした場合を除きます。

関連情報

- リバースプロキシーの使用
- ホスト名の設定

2.4.5. トラストストア設定の移行

トラストストアは、HTTPS 要求や LDAP サーバーなどの外部 TLS 通信に使用します。トラストストアを使用するには、リモートサーバーまたは CA の証明書をトラストストアにインポートします。その後、システムのトラストストアを指定して Red Hat build of Keycloak サーバーを起動できます。

たとえば、以前の設定として、次のようなものがあるとします。

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

Red Hat build of Keycloak は、PEM ファイル、または拡張子が .p12 および .pfx の PKCS12 ファイル形式のトラストストアをサポートしています。PKCS12 ファイルの場合は、暗号化されている証明書は使用できません。つまり、パスワードは必要ありません。JKS トラストストアを変換する必要があります。Java **keytool** ではパスワードの最小長が 6 文字に制限されるため、暗号化されていない PKCS12 トラストストアを作成する代わりに **openssl** を使用できます。

1. 古い JKS トラストストアの内容をインポートし、keytool を使用して一時パスワードを持つ PKCS12 トラストストアを作成します。

```
keytool -importkeystore -srckeystore path/to/myTrustStore.jks \
    -destkeystore path/to/myTrustStore.p12 \
    -srcstoretype jks \
    -deststoretype pkcs12 \
    -srcstorepass password \
    -deststorepass temp-password
```

2. **openssl** を使用して、新しい PKCS12 トラストストアの内容をエクスポートします。

```
openssl pkcs12 -in path/to/myTrustStore.p12 \
  -out path/to/myTrustStore.pem \
  -nodes -passin pass:temp-password
```

3. **openssl** を使用して、暗号化されていない新しい PKCS12 トラストストアにコンテンツを再インポートします。

```
openssl pkcs12 -export -in path/to/myTrustStore.pem \
  -out path/to/myUnencryptedTrustStore.p12 \
  -nokeys -passout pass:
```

この例では、結果の CLI パラメーターは次のようになります。

```
kc.sh start
  --truststore-paths path/to/myUnencryptedTrustStore.p12
  --tls-hostname-verifier WILDCARD
```

関連情報

- 送信要求用の信頼済み証明書の設定

2.4.6. Vault 設定の移行

キーストア Vault は Vault SPI の実装であり、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。この Vault は、Red Hat Single Sign-On 7.6 の Elytron 認証情報ストアに代わるものです。

```
<spi name="vault">
    <provider name="elytron-cs-keystore" enabled="true">
      <properties>
          <property name="location" value="path/to/keystore.p12"/>
           <property name="secret" value="password"/>
    </properties>
  </provider>
</spi>
```

Red Hat build of Keycloak では、CLI パラメーターを使用した同等の設定は、次のようなものになります。

```
kc.sh start
  --vault keystore
  --vault-file /path/to/keystore.p12
  --vault-pass password
```

Vault に保存されたシークレットには、管理コンソール内の複数の場所からアクセスできます。既存の Elytron Vault から新しい Java KeyStore ベースの Vault への移行に関しては、レルム設定の変更は必要ありません。新しく作成された Java キーストアに同じシークレットが含まれている場合は、既存のレルム設定が機能するはずです。

デフォルトの **REALM_UNDERSCORE_KEY** キーリゾルバーを使用すると、以前と同じ方法で **${vault.realm-name_alias}** によって (たとえば、LDAP ユーザーフェデレーション設定内で) シークレットにアクセスできます。

関連情報

- vault の使用

2.4.7. JVM 設定の移行

Red Hat build of Keycloak での JVM 設定方法は、Red Hat Single Sign-On 7.6 の方法と似ています。引き続き特定の環境変数を設定する必要がありますが、**/bin** フォルダーに **standalone.conf** などの設定ファイルが含まれません。

Red Hat build of Keycloak は、さまざまなデフォルトの JVM 引数を提供します。これは、メモリー割り当てと CPU オーバーヘッドの面で優れたスループットと効率を実現するため、ほとんどのデプロイメントに適していることが判明しています。また、他のデフォルトの JVM 引数は、Red Hat build of Keycloak インスタンスをスムーズに実行するためのものであり、ユースケースに合わせて引数を変更する場合は注意が必要です。

JVM 引数または GC 設定を変更するには、Java オプションとして指定する特定の環境変数を設定します。これらの設定を完全にオーバーライドするには、**JAVA_OPTS** 環境変数を指定します。

特定の Java プロパティーの追加のみが必要な場合は、**JAVA_OPTS_APPEND** 環境変数を指定します。**JAVA_OPTS** 環境変数が指定されていない場合は、**./kc.sh** スクリプト内にあるデフォルトの Java プロパティーが使用されます。

たとえば、次のように特定の Java オプションを指定できます。

```
export JAVA_OPTS_APPEND=-XX:+HeapDumpOnOutOfMemoryError
kc.sh start
```

2.4.8. SPI プロバイダー設定の移行

SPI プロバイダーの設定は、新しい設定システムを通じて行います。以下は古い形式です。

```
<spi name="<spi-id>">
    <provider name="<provider-id>" enabled="true">
        <properties>
            <property name="<property>" value="<value>"/>
        </properties>
    </provider>
</spi>
```

以下は新しい形式です。

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

ソース	形式
CLI	./kc.sh start --spi-connections-http-client-default-connection-pool-size 10
Environment Variable	KC_SPI_CONNECTIONS_HTTP_CLIENT_DEFAULT_CONNECTION_POOL_SIZE=10
設定ファイル	spi-connections-http-client-default-connection-pool-size=10

ソース	形式
Java キーストアファイル	kc.spi-connections-http-client-default-connection-pool-size=10

関連情報

- すべてのプロバイダー設定

2.4.9. 設定のトラブルシューティング

トラブルシューティングには次のコマンドを使用します。

- **kc.sh show-config** - 特定のプロパティーのロード元である設定ソースとその値を表示します。プロパティーとその値が正しく伝播されているかどうかを確認できます。
- **kc.sh --verbose start** - エラーが発生した場合、エラースタックトレース全体を出力します。

2.5. データベースの移行

Red Hat build of Keycloak では、データベーススキーマを自動的に移行することも、手動で移行することもできます。デフォルトでは、新規インストールを初めて起動すると、データベースが自動的に移行されます。

2.5.1. リレーショナルデータベースの自動移行

自動移行を実行するには、目的のデータベースに接続されているサーバーを起動します。新しいバージョンのサーバーのデータベーススキーマが変更された場合は、データベースが移行されます。

2.5.2. 手動によるリレーショナルデータベース移行

データベーススキーマを手動でアップグレードするには、デフォルトの connections-jpa プロバイダーの **migration-strategy** プロパティー値を **manual** に設定します。

```
kc.sh start --spi-connections-jpa-legacy-migration-strategy manual
```

この設定でサーバーを起動すると、データベースを移行する必要があるかどうかがチェックされます。必要な変更は **bin/keycloak-database-update.sql** SQL ファイルに書き込まれます。このファイルを確認してデータベースに対して手動で実行できます。

エクスポートされた SQL ファイルのパスと名前を変更するには、デフォルトの **connections-jpa** プロバイダーの **migration-export** プロパティーを設定します。

```
kc.sh start
  --spi-connections-jpa-legacy-migration-export <path>/<file.sql>
```

このファイルをデータベースに適用する方法の詳細は、使用しているリレーショナルデータベースのドキュメントを参照してください。変更がファイルに書き込まれると、サーバーは終了します。

2.6. RED HAT BUILD OF KEYCLOAK サーバーの起動

Red Hat Single Sign-On 7.6 ディストリビューションと Red Hat build of Keycloak ディストリビューションの起動の違いは、実行されるスクリプトにあります。これらのスクリプトは、サーバーディストリビューションの **/bin** フォルダーにあります。

2.6.1. 開発モードでのサーバーの起動

プロパティーの指定を気にせずに Red Hat build of Keycloak を試すには、以下の表の説明に従って開発モードでディストリビューションを起動できます。ただし、このモードは開発専用であり、実稼働環境では使用しないでください。

Red Hat build of Keycloak	Red Hat Single Sign-On 7.6
./kc.sh start-dev	./standalone.sh

警告

開発モードは実稼働環境では使用しないでください。

2.6.2. 実稼働モードでのサーバーの起動

Red Hat build of Keycloak には、実稼働用の専用の起動モード **./kc.sh start** があります。**start-dev** で実行する場合の違いは、デフォルトの設定値が異なることです。デフォルトでセキュリティー保護された厳密な設定セットアップが自動的に使用されます。実稼働モードでは、HTTP が無効になり、明示的な TLS とホスト名の設定が必要です。

関連情報

- 実稼働環境向けの Keycloak の設定
- Keycloak の起動の最適化

第3章 OPENSHIFT の OPERATOR デプロイメントの移行

Red Hat build of Keycloak Operator は、刷新されたサーバー設定に合わせて完全に再構築されました。Operator は、Red Hat build of Keycloak と完全に統合しますが、Red Hat Single Sign-On 7.6 Operator との下位互換性はありません。新しい Operator を使用するには、Red Hat build of Keycloak デプロイメントを作成する必要があります。詳細は、Operator ガイド を参照してください。

3.1. 前提条件

- Red Hat Single Sign-On 7.6 の以前のインスタンスが、Red Hat build of Keycloak と同じデータベースインスタンスを使用しないように、シャットダウンされている。
- サポート対象外の組み込みデータベース (Red Hat Single Sign-On 7.6 Operator によって管理されるもの) を使用していた場合、そのデータベースが、ユーザーによってプロビジョニングされた外部データベースに変換されている。
- データベースのバックアップが作成されている。
- リリースノート を確認した。

3.2. 移行プロセス

1. Red Hat build of Keycloak Operator を namespace にインストールします。
2. 新しい CR と関連するシークレットを作成します。Red Hat Single Sign-On 7.6 の設定を新しい Keycloak CR に手動で移行します。
3. カスタムプロバイダーを使用していた場合は、それらを移行し、カスタムプロバイダーを含めるカスタムの Red Hat build of Keycloak コンテナーイメージを作成します。
4. カスタムテーマを使用していた場合は、それらを移行し、カスタムプロバイダーを含めるカスタムの Red Hat build of Keycloak コンテナーイメージを作成します。

3.3. KEYCLOAK CR の移行

Keycloak CR は、すべてのサーバー設定オプションをサポートするようになりました。関連するすべてのオプションを、CR 仕様で、直接ファーストクラスシチズンフィールドとして使用できます。CR のすべてのオプションは、サーバーオプションと同じ命名規則に準拠しています。これにより、ベアメタルデプロイメントと Operator デプロイメント間のエクスペリエンスがシームレスになります。

さらに、SPI プロバイダー設定など、CR に不足しているオプションを **additionalOptions** フィールドで定義できます。また、サポートされる代替フィールドが CR のファーストクラスシチズンフィールドとして存在しない場合は、**podTemplate** (テクノロジープレビューのフィールド) を使用して、Kubernetes デプロイメントの RAW Pod テンプレートを変更することもできます。

以下は、Operator を通じて Red Hat build of Keycloak をデプロイするための Keycloak CR の例を示しています。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  instances: 1
```

```
db:
  vendor: postgres
  host: postgres-db
  usernameSecret:
    name: keycloak-db-secret
    key: username
  passwordSecret:
    name: keycloak-db-secret
    key: password
http:
  tlsSecret: example-tls-secret
hostname:
  hostname: test.keycloak.org
additionalOptions:
  - name: spi-connections-http-client-default-connection-pool-size
    value: 20
```

CLI 設定との類似点に注目してください。

```
./kc.sh start --db=postgres --db-url-host=postgres-db --db-username=user --db-password=pass --
https-certificate-file=mycertfile --https-certificate-key-file=myprivatekey --hostname=test.keycloak.org
--spi-connections-http-client-default-connection-pool-size=20
```

関連情報

- 基本的な Keycloak のデプロイメント
- 詳細設定

3.3.1. データベース設定の移行

Red Hat build of Keycloak は、以前に Red Hat Single Sign-On 7.6 で使用されていたのと同じデータベースインスタンスを使用できます。データベーススキーマは、Red Hat build of Keycloak が初めてデータベースに接続するときに自動的に移行されます。

警告

Red Hat Single Sign-On 7.6 Operator によって管理される組み込みデータベースの移行は、サポートされていません。

Red Hat Single Sign-On 7.6 Operator では、外部データベース接続はシークレットを使用して設定していました。次に例を示します。

```
apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-secret
  namespace: keycloak
  labels:
```

```
    app: sso
stringData:
  POSTGRES_DATABASE:  kc-db-name
  POSTGRES_EXTERNAL_ADDRESS: my-postgres-hostname
  POSTGRES_EXTERNAL_PORT: 5432
  POSTGRES_USERNAME: user
  POSTGRES_PASSWORD: pass
type: Opaque
```

Red Hat build of Keycloak では、データベースは、シークレットとして参照される認証情報を使用して Keycloak CR に直接設定されます。次に例を示します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  db:
    vendor: postgres
    host: my-postgres-hostname
    port: 5432
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  ...

apiVersion: v1
kind: Secret
metadata:
  name: keycloak-db-secret
stringData:
  username: "user"
  password: "pass"
type: Opaque
```

3.3.1.1. サポートされているデータベースベンダー

Red Hat Single Sign-On 7.6 Operator は PostgreSQL データベースのみをサポートしていました。一方、Red Hat build of Keycloak Operator はサーバーでサポートされているすべてのデータベースベンダーをサポートしています。

3.3.2. TLS 設定の移行

Red Hat Single Sign-On 7.6 Operator では、デフォルトで、OpenShift CA によって生成された TLS シークレットを使用するようにサーバーが設定されていました。Red Hat build of Keycloak Operator では、実稼働環境のベストプラクティスを実現するために、TLS については何も前提を設けておらず、ユーザーが独自の TLS 証明書とキーペアを提供する必要があります。次に例を示します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
```

```
  name: example-kc
spec:
  http:
    tlsSecret: example-tls-secret
  ...
```

tlsSecret で参照されるシークレットの形式には、標準の Kubernetes TLS シークレット (**kubernetes.io/tls**) タイプを使用する必要があります。

Red Hat Single Sign-On 7.6 Operator は、デフォルトで、ルートで再暗号化 TLS Termination ストラテジーを使用していました。Red Hat build of Keycloak Operator は、デフォルトでパススルーストラテジーを使用します。また、Red Hat Single Sign-On 7.6 Operator は、TLS Termination の設定をサポートしていました。Red Hat build of Keycloak Operator は、現在のリリースでは TLS Termination をサポートしていません。

デフォルトの Operator 管理のルートが必要な TLS 設定を満たさない場合は、ユーザーがカスタムルートを作成し、デフォルトのルートを次のようにして無効にする必要があります。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ingress:
    enabled: false
  ...
```

3.3.3. エクステンションのためのカスタムイメージの使用

Red Hat build of Keycloak Operator では、ベストプラクティスを反映し、イミュータブルコンテナーをサポートするために、Keycloak CR でのエクステンションの指定がサポートされていません。エクステンションをデプロイするには、最適化されたカスタムイメージをビルドする必要があります。Keycloak CR には、Red Hat build of Keycloak イメージを指定するための専用フィールドが含まれるようになりました。次に例を示します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  image: quay.io/my-company/my-keycloak:latest
  ...
```

注記

カスタムイメージを指定した場合、Operator はそれがすでに最適化されていると想定し、サーバーを起動するたびにコストのかかる最適化を実行しません。

関連情報

- Operator でのカスタム Keycloak イメージの使用
- 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

3.3.4. アップグレードストラテジーオプションの削除

Red Hat Single Sign-On 7.6 Operator は、サーバーのアップグレード実行時の再作成およびローリングストラテジーをサポートしていました。このアプローチは現実的ではありませんでした。Red Hat Single Sign-On 7.6 Operator がアップグレードとデータベース移行を実行する前にデプロイメントをスケールダウンするかどうかは、ユーザーが選択する必要がありました。ローリングストラテジーがいつ安全に使用できるかは、ユーザーには明らかではありませんでした。

したがって、このオプションは Red Hat build of Keycloak Operator では削除され、再作成ストラテジーは常に暗黙的に実行されるようになりました。これにより、新しいサーバーコンテナーイメージで Pod を作成する前にデプロイメント全体がスケールダウンされ、単一のサーバーバージョンのみがデータベースにアクセスできるようになります。

3.3.5. デフォルトで公開されるヘルスエンドポイント

Red Hat build of Keycloak は、OpenShift プローブによって使用される単純なヘルスエンドポイントをデフォルトで公開するようにサーバーを設定します。エンドポイントはデプロイメントに関するセキュリティー上の機密データを公開しませんが、エンドポイントには認証なしでアクセスできます。代替策として、**<your-server-context-root>/health** エンドポイントをカスタムルートでブロックできます。

以下に例を示します。

1. TLS edge termination 用に設定された Keycloak を作成します。
   tlsSecret フィールドは必ず省略してください。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  proxy:
   Headers:xforwarded
  hostname:
    hostname: example.com
  ...
```

2. ヘルスエンドポイントへのアクセスを禁止するブロッキングルートを作成します。

```
kind: Route
apiVersion: route.openshift.io/v1
metadata:
  name: example-kc-block-health
  annotations:
    haproxy.router.openshift.io/rewrite-target: /404
spec:
  host: example.com
  path: /health
  to:
    kind: Service
    name: example-kc-service
  port:
    targetPort: http
  tls:
    termination: edge
```

注記

パスベースのルートでは、エッジまたは再暗号化のいずれかに対して TLS Termination を設定する必要があります。デフォルトでは、Operator はパススルーを使用します。

3.3.6. Pod テンプレートを使用した高度なデプロイメントオプションの移行

Red Hat Single Sign-On 7.6 Operator は、ボリュームなどのデプロイメント設定用の複数の低レベルフィールドを公開していました。Red Hat build of Keycloak Operator は事前設定される部分が多く、これらのフィールドのほとんどは公開されません。ただし、**podTemplate** として指定されたデプロイメントフィールドを設定することは可能です。次に例を示します。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  unsupported:
    podTemplate:
      metadata:
        labels:
          foo: "bar"
      spec:
        containers:
          - volumeMounts:
              - name: test-volume
                mountPath: /mnt/test
        volumes:
          - name: test-volume
            secret:
              secretName: test-secret
  ...
```

注記

spec.unsupported.podTemplate フィールドは、すべての条件下で機能が十分にテストされていない低レベルの設定を公開するため、サポートが限定的になります。可能な限り、CR 仕様の最上位で、完全にサポートされているファーストクラスシチズンフィールドを使用してください。

たとえば、**spec.unsupported.podTemplate.spec.imagePullSecrets** の代わりに、**spec.imagePullSecrets** を直接使用します。

3.3.7. 外部インスタンスへの接続はサポート対象外

Red Hat Single Sign-On 7.6 Operator では、Red Hat Single Sign-On 7.6 の外部インスタンスへの接続がサポートされていました。Red Hat build of Keycloak Operator では、たとえば、クライアント CR を使用して既存のレルム内にクライアントを作成することは、サポートされなくなりました。

3.3.8. Horizontal Pod Autoscaler 対応デプロイメントの移行

Red Hat Single Sign-On 7.6 で Horizontal Pod Autoscaler (HPA) を使用するには、Keycloak CR の **disableReplicasSyncing: true** フィールドを設定し、サーバーの StatefulSet をスケーリングする必要がありました。これは不要になりました。Red Hat build of Keycloak Operator の Keycloak CR は、HPA

によって直接スケーリングできるためです。

3.4. KEYCLOAK REALM CR の移行

Realm CR は、同様の機能を提供し、同様のスキーマを持つ Realm Import CR に置き換えられました。Realm Import CR はレルムのブートストラップのみを提供するため、レルムの削除をサポートしなくなりました。以前の Realm CR と同様に、更新もサポートしていません。

一部のフィールドのみを提供していた以前の Realm CR と比較して、Realm Import CR には、完全なレルム表現が含まれるようになりました。

Red Hat Single Sign-On 7.6 Realm CR の例:

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

対応する Red Hat build of Keycloak Realm Import CR の例:

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: KeycloakRealmImport
metadata:
  name: example-keycloakrealm
spec:
  keycloakCRName: example-kc
  realm:
    id: "basic"
    realm: "basic"
    enabled: True
    displayName: "Basic Realm"
```

関連情報

- Realm Import

3.5. 削除された CR

Client CR と User CR は、Red Hat build of Keycloak Operator から削除されました。これらの CR の欠如は、新しい Realm Import CR によって部分的に軽減できます。Client CR のサポート追加は、Red Hat build of Keycloak の今後のリリースに関するロードマップに含まれていますが、User CR は現在計画されている機能ではありません。

第4章 OPENSHIFT のテンプレートデプロイメントの移行

OpenShift テンプレートは非推奨となり、Red Hat build of Keycloak コンテナーイメージから削除されました。OpenShift に Red Hat build of Keycloak をデプロイする場合は、代わりに Operator を使用することを推奨します。

注記

OpenShift 3.x はサポートされなくなりました。

通常は、外部管理データベースを参照する (Red Hat build of Keycloak Operator の) Keycloak CR を作成する必要があります。このテンプレートを使用する PostgreSQL データベースは、DeploymentConfig によって管理されます。まず、テンプレートによって作成された **application_name-postgresql** DeploymentConfig が保存されます。DeploymentConfig によって PostgreSQL データベースインスタンスが作成されると、そのインスタンスが Red Hat build of Keycloak Operator で使用できるようになります。

このガイドには、Operator またはクラウドプロバイダーによるこのインスタンスから自己管理型データベースへの移行手順は記載されてません。

Red Hat build of Keycloak Operator はデータベースを管理しないため、データベースを別途プロビジョニングして管理する必要があります。

4.1. 内部 H2 データベースを使用したデプロイメントの移行

関係するテンプレートは次のとおりです。

- sso76-ocp3-https
- sso76-ocp4-https
- sso76-ocp3-x509-https
- sso76-ocp4-x509-https

これらのテンプレートは devel データベースに依存しているため、実稼働環境での使用はサポートされていません。

4.2. 一時的な POSTGRESQL データベースを使用したデプロイメントの移行

関係するテンプレートは次のとおりです。

- sso76-ocp3-postgresql
- sso76-ocp4-postgresql

このテンプレートは、永続ストレージを使用せずに PostgreSQL データベースを作成します。これは開発目的でのみ推奨されます。

4.3. 永続的な POSTGRESQL データベースを使用したデプロイメントの移行

関係するテンプレートは次のとおりです。

- sso76-ocp3-postgresql-persistent
- sso76-ocp4-postgresql-persistent
- sso76-ocp3-x509-postgresql-persistent
- sso76-ocp4-x509-postgresql-persistent

4.3.1. 前提条件

- Red Hat Single Sign-On 7.6 の以前のインスタンスが、Red Hat build of Keycloak と同じデータベースインスタンスを使用しないように、シャットダウンされている。
- データベースのバックアップが作成されている。
- リリースノート を確認した。

4.4. 移行プロセス

1. Red Hat build of Keycloak Operator を namespace にインストールします。
2. 新しい CR と関連するシークレットを作成します。
   テンプレートベースの Red Hat Single Sign-On 7.6 設定を、新しい Red Hat build of Keycloak CR に手動で移行します。テンプレートパラメーターと Keycloak CR フィールド間の推奨マッピングについては、次の例を参照してください。

以下の例では、Red Hat build of Keycloak Operator CR と、Red Hat Single Sign-On 7.6 テンプレートによって以前に作成した DeploymentConfig を比較します。

Red Hat build of Keycloak の Operator CR

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: rhbk
spec:
  instances: 1
  db:
    vendor: postgres
    host: postgres-db
    usernameSecret:
      name: keycloak-db-secret
      key: username
    passwordSecret:
      name: keycloak-db-secret
      key: password
  http:
    tlsSecret: sso-x509-https-secret
```

Red Hat Single Sign-On 7.6 の DeploymentConfig

```
apiVersion: apps.openshift.io/v1
kind: DeploymentConfig
```

```
metadata:
  name: rhsso
spec:
  replicas: 1
  template:
    spec:
      volumes:
        - name: sso-x509-https-volume
          secret:
            secretName: sso-x509-https-secret
            defaultMode: 420
      containers:
          volumeMounts:
            - name: sso-x509-https-volume
              readOnly: true
          env:
            - name: DB_SERVICE_PREFIX_MAPPING
              value: postgres-db=DB
            - name: DB_USERNAME
              value: username
            - name: DB_PASSWORD
              value: password
```

次の表に、Keycloak CR のフィールドを JSON パス表記で示します。たとえば、**.spec** は **spec** フィールドを示します。**spec.unsupported** はテクノロジープレビューのフィールドです。その機能は、最終的に他の CR フィールドによって実現可能になる予定です。**太字** のパラメーターは、パススルーテンプレートと再暗号化テンプレートの両方でサポートされています。

4.4.1. 一般的なパラメーターの移行

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
APPLICATION_NAME	.metadata.name
IMAGE_STREAM_NAMESPACE	該当なし - イメージは Operator によって制御されます。または、ユーザーが主に spec.image を使用してカスタムイメージを指定します。
SSO_ADMIN_USERNAME	直接設定はありません。デフォルトは admin です。
SSO_ADMIN_PASSWORD	該当なし - 初期調整中に Operator によって作成されます。
MEMORY_LIMIT	**.spec.unsupported.podTemplate.spec.containers[0].resources.limits['memory']**
SSO_SERVICE_PASSWORD、SSO_SERVICE_USERNAME	使用されなくなりました。
SSO_TRUSTSTORE、SSO_TRUSTSTORE_PASSWORD、SSO_TRUSTSTORE_SECRET	**.spec.truststores** トラストストアはパスワード保護をしてはならないことに注意してください。

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
SSO_REALM	既存のデータベースを再利用する場合は不要です。代わりに RealmImport CR を使用することもできます。

4.4.2. データベースデプロイメントパラメーターの移行

POSTGRESQL_IMAGE_STREAM_TAG、**POSTGRESQL_MAX_CONNECTIONS**、**VOLUME_CAPACITY** および **POSTGRESQL_SHARED_BUFFERS** は、データベースデプロイメントを作成するために選択した代替パラメーターに移行する必要があります。

4.4.3. データベース接続パラメーターの移行

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
DB_VENDOR	**.spec.db.vendor** - PostgreSQL を引き続き使用する場合は、PostgreSQL に設定する必要があります。
DB_DATABASE	**.spec.db.database**
DB_MIN_POOL_SIZE	**.spec.db.poolMinSize**
DB_MAX_POOL_SIZE	**.spec.db.maxPoolSize**
DB_TX_ISOLATION	ドライバーでサポートされている場合、またはターゲットデータベースの一般設定として、**spec.db.url** によって設定される可能性があります。
DB_USERNAME	**.spec.db.usernameSecret**
DB_PASSWORD	**.spec.db.passwordSecret**
DB_JNDI	適用されなくなりました。

4.4.4. ネットワークパラメーターの移行

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0

Red Hat Single Sign-On 7.6	Red Hat build of Keycloak 24.0
HOSTNAME_HTTP	**.spec.hostname.hostname** - **.spec.http.httpEnabled=true** とともに指定します。Red Hat build of Keycloak Operator は、単一の Ingress/ルートのみを作成するため、http ルートを作成するには **.spec.http.tlsSecret** を未指定にする必要があります。
HOSTNAME_HTTPS	**.spec.hostname.hostname** - **.spec.http.tlsSecret** とともに指定します。
SSO_HOSTNAME	**.spec.hostname.hostname**
HTTPS_SECRET	**.spec.http.tlsSecret** - 下記の別の HTTPS パラメーターを参照してください。
HTTPS_KEYSTORE HTTPS_KEYSTORE_TYPE HTTPS_NAME HTTPS_PASSWORD	適用されなくなりました。**.spec.http.tlsSecret** によって参照されるシークレットは、**tls.crt** および **tls.key** エントリーを持つ **kubernetes.io/tls** タイプである必要があります。
X509_CA_BUNDLE	**.spec.truststores**

Red Hat build of Keycloak Operator は現在、TLS Termination を設定する方法をサポートしていないことに注意してください。デフォルトでは、パススルーストラテジーが使用されます。したがって、パススルーストラテジーと再暗号化ストラテジーのどちらが使用されるかは問題ではないため、プロキシーオプションは、まだファーストクラスシチズンオプションフィールドとして公開されていません。ただし、このオプションが必要な場合は、Red Hat build of Keycloak の証明書を信頼するために、デフォルトの Ingress Operator 証明書を置き換えてルートを手動で設定できます。

Red Hat build of Keycloak Operator のデフォルトの動作は、次のように指定することででオーバーライドできます。

```
additionalOptions:
  name: proxy
  value: reencrypt
```

4.4.5. JGroups パラメーターの移行

JGROUPS_ENCRYPT_SECRET、JGROUPS_ENCRYPT_KEYSTORE、JGROUPS_ENCRYPT_NAME、JGROUPS_ENCRYPT_PASSWORD、および **JGROUPS_CLUSTER_PASSWORD** には、Keycloak CR のファーストクラス表現がありません。キャッシュ通信の保護は、キャッシュ設定ファイルを使用して引き続き可能です。

関連情報

- 分散キャッシュの設定

第5章 RED HAT SINGLE SIGN-ON 7.6 によってセキュリティー保護されたアプリケーションの移行

Red Hat build of Keycloak では、アプリケーションによる一部の Red Hat Single Sign-On 7.6 クライアントアダプターの使用方法に重要な変更が導入されています。

Red Hat build of Keycloak では、一部のクライアントアダプターのリリースが中止されることに加えて、クライアントアプリケーションによる OpenID Connect および SAML プロトコルの使用方法に影響を与える修正と改善も導入されています。

この章では、これらの変更に対処し、アプリケーションを移行して Red Hat build of Keycloak と統合する手順を説明します。

5.1. OPENID CONNECT クライアントの移行

Red Hat build of Keycloak のこのリリース以降、以下の Java クライアント OpenID Connect アダプターはリリースされなくなりました。

- Red Hat JBoss Enterprise Application Platform 6.x
- Red Hat JBoss Enterprise Application Platform 7.x
- Spring Boot
- Red Hat Fuse

これらのアダプターが最初にリリースされたときと比較して、OpenID Connect は Java エコシステム全体で広く利用できるようになりました。また、アプリケーションサーバーやフレームワークなどのテクノロジースタックから利用可能な機能を使用することにより、相互運用性とサポートが大幅に向上します。

これらのアダプターはサポートが終了しており、Red Hat Single Sign-On 7.6 からのみ利用可能です。OAuth2 および OpenID Connect プロトコルの最新の更新を使用してアプリケーションを最新の状態に保つための代替手段を探すことを強く推奨します。

5.1.1. OpenID Connect プロトコルとクライアント設定の主な変更点

5.1.1.1. Access Type クライアントオプションは利用不可

OpenID Connect クライアントの作成または更新時に **Access Type** が使用できなくなります。ただし、他の方法を使用してこの機能を実現することもできます。

- **Bearer Only** 機能を実現するには、認証フローを持たないクライアントを作成します。クライアントの詳細の **Capability config** セクションで、フローが選択されていないことを確認してください。クライアントは Keycloak からトークンを取得できません。これは、**Bearer Only** アクセスタイプを使用した場合と同じです。
- **Public** 機能を実現するには、このクライアントに対してクライアント認証が無効になっていて、少なくとも 1 つのフローが有効になっていることを確認してください。
- **Confidential** 機能を実現するには、クライアントに対して **Client Authentication** が有効になっていて、少なくとも 1 つのフローが有効になっていることを確認してください。

ブール値フラグの **bearerOnly** と **publicClient** は、クライアント JSON オブジェクトに引き続き存在します。これらは、管理 REST API によってクライアントを作成または更新するとき、または部分イン

ポートまたはレルムインポートによってこのクライアントをインポートするときに使用できます。ただし、これらのオプションは管理コンソール v2 では直接使用できません。

5.1.1.2. 有効なリダイレクト URI の検証スキームの変更

アプリケーションクライアントが http(s) 以外のカスタムスキームを使用している場合、有効なリダイレクトパターンでそのスキームを明示的に許可することが検証に必要になりました。カスタムスキームを許可するパターンの例には、custom:/test、custom:/test/*、custom: などがあります。セキュリティー上の理由から、* などの一般的なパターンは、これらのスキームに対して無効になりました。

5.1.1.3. OpenID Connect ログアウトエンドポイントでの client_id パラメーターのサポート

client_id パラメーターをサポートします。これは、OIDC RP-Initiated Logout 1.0 仕様に基づいています。この機能は、**id_token_hint** パラメーターが使用できない場合に、ログアウト後の URI リダイレクト検証にどのクライアントを使用する必要があるかを検出するのに役立ちます。**id_token_hint** パラメーターを使用せずに **client_id** パラメーターのみを使用した場合でも、ログアウト確認画面をユーザーに表示する必要があります。そのため、ユーザーにログアウト確認画面を表示する必要がない場合は、クライアントに **id_token_hint** パラメーターを使用することを推奨します。

5.1.2. Valid Post Logout Redirect URI

Valid Post Logout Redirect URIs 設定オプションが OIDC クライアントに追加されました。これは、OIDC 仕様に準拠しています。ログインおよびログアウト後のリダイレクトには、別のリダイレクト URI セットを使用できます。**Valid Post Logout Redirect URIs** に使用される値 **+** は、**Valid Redirect URIs** オプションで指定されたのと同じリダイレクト URI のセットがログアウトに使用されることを意味します。この変更は、下位互換性のため、以前のバージョンから移行するときのデフォルトの動作にも一致します。

5.1.2.1. UserInfo エンドポイントの変更

5.1.2.1.1. エラー応答の変更

UserInfo エンドポイントが、RFC 6750 (The OAuth 2.0 Authorization Framework: Bearer Token Usage) に完全に準拠したエラー応答を返すようになりました。エラーコードと説明 (利用可能な場合) は、JSON オブジェクトフィールドではなく **WWW-Authenticate** チャレンジ属性として提供されます。

応答は、エラーの状態に応じて次のようになります。

- アクセストークンが提供されていない場合:

```
401 Unauthorized
WWW-Authenticate: Bearer realm="myrealm"
```

- アクセストークンを提供するために複数の方法が同時に使用された場合 (たとえば、認可ヘッダー + POST access_token パラメーター)、または POST パラメーターが重複する場合:

```
400 Bad Request
WWW-Authenticate: Bearer realm="myrealm", error="invalid_request", error_description="..."
```

- アクセストークンに **openid** スコープがない場合:

```
403 Forbidden
WWW-Authenticate: Bearer realm="myrealm", error="insufficient_scope",
error_description="Missing openid scope"
```

- UserInfo 応答の署名/暗号化の暗号鍵を解決できない場合:

```
500 Internal Server Error
```

- トークン検証エラーの場合、**invalid_token** エラーコードと併せて **401 Unauthorized** が返されます。このエラーには、ユーザーおよびクライアント関連のチェックが含まれており、残りのすべてのエラーケースがキャプチャーされます。

```
401 Unauthorized
WWW-Authenticate: Bearer realm="myrealm", error="invalid_token", error_description="..."
```

5.1.2.1.2. UserInfo エンドポイントに対するその他の変更

アクセストークンに **openid** スコープが必要になりました。これは、OAuth 2.0 ではなく OpenID Connect に固有の機能である UserInfo によって規定されています。**openid** スコープがトークンにない場合、要求が **403 Forbidden** として拒否されます。前のセクションを参照してください。

UserInfo がユーザーのステータスをチェックするようになり、ユーザーが無効な場合は、**invalid_token** 応答を返すようになりました。

5.1.2.1.3. Service Account Client のデフォルト Client ID マッパーの変更

Service Account Client のデフォルトの **Client ID** マッパーが変更されました。**Token Claim Name** フィールドの値が **clientId** から **client_id** に変更されました。**client_id** クレームは OAuth2 仕様に準拠しています。

- JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens
- OAuth 2.0 Token Introspection
- OAuth 2.0 Token Exchange

clientId userSession ノートは引き続き存在します。

5.1.2.1.4. OAuth 2.0/OpenID Connect Authentication Response に iss パラメーターを追加

RFC 9207 OAuth 2.0 Authorization Server Issuer Identification 仕様で、セキュアな認証応答を実現するために、OAuth 2.0/OpenID Connect Authentication Response に **iss** パラメーターが追加されました。

過去のリリースにはこのパラメーターはありませんでしたが、仕様の要求に従って、Red Hat build of Keycloak にはデフォルトでこのパラメーターが追加されました。ただし、一部の OpenID Connect/OAuth2 アダプター、特に Red Hat build of Keycloak の古いアダプターでは、この新しいパラメーターで問題が発生する可能性があります。たとえば、クライアントアプリケーションへの認証に成功すると、パラメーターは常にブラウザー URL に表示されます。

このような場合、認証応答への **iss** パラメーターの追加を無効にすると役立つことがあります。これは、管理コンソールから、**OpenID Connect Compatibility Modes** を含むセクション内のクライアントの詳細で、特定のクライアントに対して行うことができます。**Exclude Issuer From Authentication Response** を有効にすると、認証応答への **iss** パラメーターの追加を回避できます。

5.2. RED HAT JBOSS ENTERPRISE APPLICATION PLATFORM アプリケーションの移行

5.2.1. Red Hat JBoss Enterprise Application Platform 8.x

アプリケーションに、Red Hat build of Keycloak やその他の OpenID プロバイダーと統合するための追加の依存関係が不要になりました。

代わりに、JBoss EAP のネイティブ OpenID Connect クライアントからの OpenID Connect サポートを利用できます。詳細は、JBoss EAP での OpenID Connect を参照してください。

JBoss EAP のネイティブアダプターは、Red Hat build of Keycloak のアダプター JSON 設定によく似た設定スキーマに依存します。たとえば、**keycloak.json** 設定ファイルを使用したデプロイメントは、JBoss EAP の次の設定にマッピングできます。

```
{
  "realm": "quickstart",
  "auth-server-url": "http://localhost:8180",
  "ssl-required": "external",
  "resource": "jakarta-servlet-authz-client",
  "credentials": {
    "secret": "secret"
  }
}
```

JBoss EAP のネイティブアダプターを使用して Jakarta ベースのアプリケーションを Red Hat build of Keycloak と統合する例については、Red Hat build of Keycloak クイックスタートリポジトリーにある次の例を参照してください。

- JAX-RS Resource Server
- Servlet Application

JBoss EAP のネイティブ OpenID Connect クライアントに移行することを強く推奨します。これは JBoss EAP 8 以降にデプロイされた Jakarta アプリケーションに最適です。

5.2.2. Red Hat JBoss Enterprise Application Platform 7.x

Red Hat JBoss Enterprise Application Platform 7.x は、フルサポートの終了に近づいているため、Red Hat build of Keycloak ではサポートされません。Red Hat JBoss Enterprise Application Platform 7.x にデプロイされた既存のアプリケーションについては、メンテナンスサポート対象のアダプターが Red Hat Single Sign-On 7.6 を通じて利用可能です。

Red Hat Single Sign-On 7.6 アダプターは、Red Hat build of Keycloak 24.0 サーバーと組み合わせて使用することがサポートされています。

5.2.3. Red Hat JBoss Enterprise Application Platform 6.x

Red Hat JBoss Enterprise Application Platform JBoss EAP 6.x はメンテナンスサポートの終了に達したため、今後は Red Hat Single Sign-On 7.6 でも Red Hat build of Keycloak でもサポートされません。

5.3. SPRING BOOT アプリケーションの移行

Spring Framework エコシステムは急速に発展しており、エコシステムが提供する OpenID Connect サポートを活用することで、より優れたエクスペリエンスが得られます。

アプリケーションに、Red Hat build of Keycloak やその他の OpenID プロバイダーと統合するための追加の依存関係が不要になりました。アプリケーションは、Spring Security の包括的な OAuth2/OpenID Connect サポートに依存します。詳細は、OAuth2/OpenID Connect support from Spring Security を参照してください。

このサポートは、機能の面では、標準ベースの OpenID Connect クライアント実装を提供します。標準プロトコルをまだ使用していない場合は、たとえば **Logout** 機能を再検討することをお勧めします。Red Hat build of Keycloak は、OpenID Connect エコシステムの標準ベースのログアウトプロトコルを完全にサポートしています。

Spring Security アプリケーションを Red Hat build of Keycloak と統合する方法の例については、クイックスタートリポジトリー を参照してください。

Spring Boot 用の Red Hat build of Keycloak クライアントアダプターから移行できない場合は、現在メンテナンスのみのサポートとなっている Red Hat Single Sign-On 7.6 からアダプターを利用できます。

Red Hat Single Sign-On 7.6 アダプターは、Red Hat build of Keycloak 24.0 サーバーと組み合わせて使用することがサポートされています。

5.4. RED HAT FUSE アプリケーションの移行

Red Hat Fuse はフルサポートの終了に達したため、Red Hat build of Keycloak 24.0 ではサポートされません。Red Hat Fuse アダプターは、Red Hat Single Sign-On 7.6 によるメンテナンスサポートを通じて引き続き利用できます。

Red Hat Single Sign-On 7.6 アダプターは、Red Hat build of Keycloak 24.0 サーバーと組み合わせて使用することがサポートされています。

5.5. AUTHORIZATION SERVICES POLICY ENFORCER を使用したアプリケーションの移行

Red Hat build of Keycloak Authorization Services との統合をサポートするために、ポリシーエンフォーサーは Java クライアントアダプターとは別に提供されています。

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-policy-enforcer</artifactId>
    <version>${Red Hat build of Keycloak .version}</version>
</dependency>
```

ポリシーエンフォーサーが Java クライアントアダプターから切り離されたため、Red Hat build of Keycloak を、OAuth2 または OpenID Connect のビルトインサポートを提供する Java テクノロジーに統合できるようになりました。Red Hat build of Keycloak Policy Enforcer は、次のタイプのアプリケーションに対するビルトインサポートを提供します。

- 粒度の細かい認可を使用したサーブレットアプリケーション
- Red Hat build of Keycloak Authorization Services を使用してセキュリティー保護された Spring Boot REST サービス

Red Hat build of Keycloak Policy Enforcer をさまざまなタイプのアプリケーションと統合するには、次の例を参照してください。

- Servlet Application Using Fine-grained Authorization
- Spring Boot REST Service Protected Using Keycloak Authorization Services

使用している Red Hat Single Sign-On 7.6 Java アダプターから移行できない場合は、現在メンテナンスサポートの対象となっている Red Hat Single Sign-On 7.6 からアダプターを利用できます。

Red Hat Single Sign-On 7.6 アダプターは、Red Hat build of Keycloak 24.0 サーバーと組み合わせて使用することがサポートされています。

関連情報

- ポリシーエンフォーサー

5.6. RED HAT BUILD OF KEYCLOAK JS アダプターを使用した SINGLE PAGE APPLICATIONS (SPA) の移行

Red Hat Single Sign-On 7.6 アダプターでセキュリティー保護されたアプリケーションを移行するには、より新しいバージョンのアダプターを提供する Red Hat build of Keycloak 24.0 にアップグレードします。使用方法に応じて、以下に説明する若干の変更が必要になります。

5.6.1. レガシー Promise API の削除

このリリースでは、Red Hat build of Keycloak JS アダプターからレガシー Promise API メソッドが削除されました。そのため、アダプターから返された Promise で **.success()** および **.error()** を呼び出すことができなくなりました。

5.6.2. 新しい Operator を使用してインスタンス化する必要性

以前のリリースでは、Red Hat build of Keycloak JS アダプターを新しい Operator を使用せずに構築すると、非推奨の警告がログに記録されていました。このリリース以降、これを行うと代わりに例外が出力されます。この変更は、JavaScript クラス の想定される動作との整合性を確保し、将来的にアダプターのさらなるリファクタリングを可能にするためのものです。

Red Hat Single Sign-On 7.6 アダプターでセキュリティー保護されたアプリケーションを移行するには、より新しいバージョンのアダプターを提供する Red Hat build of Keycloak 24.0 にアップグレードします。

5.7. SAML アプリケーションの移行

5.7.1. Red Hat JBoss Enterprise Application Platform アプリケーションの移行

5.7.1.1. Red Hat JBoss Enterprise Application Platform 8.x

Red Hat build of Keycloak 24.0 には、Jakarta EE のサポートを含む、Red Hat JBoss Enterprise Application Platform 8.x のクライアントアダプターが含まれています。

5.7.1.2. Red Hat JBoss Enterprise Application Platform 7.x

Red Hat JBoss Enterprise Application Platform 7.x は、フルサポートの終了に近づいているため、Red Hat build of Keycloak ではサポートされません。Red Hat JBoss Enterprise Application Platform 7.x にデプロイされた既存のアプリケーションについては、メンテナンスサポート対象のアダプターが Red Hat Single Sign-On 7.6 を通じて利用可能です。

Red Hat Single Sign-On 7.6 アダプターは、Red Hat build of Keycloak 24.0 サーバーと組み合わせて使用することがサポートされています。

5.7.1.3. Red Hat JBoss Enterprise Application Platform 6.x

Red Hat JBoss Enterprise Application Platform JBoss EAP 6.x はメンテナンスサポートの終了に達したため、今後は Red Hat Single Sign-On 7.6 でも Red Hat build of Keycloak でもサポートされません。

5.7.2. SAML プロトコルとクライアント設定の主な変更点

5.7.2.1. SAML SP メタデータの変更

このリリースより前は、SAML SP メタデータに、署名と暗号化の両方に使用する同じキーが含まれていました。このバージョンの Keycloak 以降、SP メタデータには、暗号化に使用する暗号化専用のレルムキーのみが含まれます。また、各暗号化キー記述子に対して、併用すべきアルゴリズムが指定されています。次の表に、サポートされている XML-Enc アルゴリズムと、Red Hat build of Keycloak レルムキーへのマッピングを示します。

XML-Enc アルゴリズム	レルムキーアルゴリズム
rsa-oaep-mgf1p	RSA-OAEP
rsa-1_5	RSA1_5

関連情報

- Keycloak Upgrading Guide

5.7.2.2. SAML の RSA_SHA1 および DSA_SHA1 アルゴリズムの非推奨化

SAML アダプター、クライアント、およびアイデンティティープロバイダーで **Signature algorithms** として設定できるアルゴリズム **RSA_SHA1** および **DSA_SHA1** が非推奨になりました。**SHA256** または **SHA512** をベースとした、より安全な代替手段を使用することを推奨します。また、これらのアルゴリズムを使用した署名付き SAML ドキュメントまたはアサーションの署名の検証は、Java 17 以降では機能しません。このアルゴリズムを使用しても、SAML ドキュメントを使用する相手側で Java 17 以降が実行されている場合、署名の検証は機能しません。

考えられる回避策は、次のようなアルゴリズムを削除することです。

- リストから **http://www.w3.org/2000/09/xmldsig#rsa-sha1** または **http://www.w3.org/2000/09/xmldsig#dsa-sha1** を削除する
- "禁止アルゴリズム" を、ファイル **$JAVA_HOME/conf/security/java.security** のプロパティー **jdk.xml.dsig.secureValidationPolicy** で設定する

第6章 カスタムプロバイダーの移行

Red Hat Single Sign-On 7.6 と同様に、カスタムプロバイダーをデプロイメントディレクトリーにコピーすると、そのプロバイダーが Red Hat build of Keycloak にデプロイされます。Red Hat build of Keycloak では、プロバイダーを、削除された **standalone/deployments** の代わりに **providers** ディレクトリーにコピーします。追加の依存関係も **providers** ディレクトリーにコピーする必要があります。

Red Hat build of Keycloak はカスタムプロバイダーに個別のクラスパスを使用しないため、追加の依存関係を含める場合には、より注意する必要があります。さらに、**EAR** および **WAR** パッケージ形式、および **jboss-deployment-structure.xml** ファイルはサポートされなくなりました。

Red Hat Single Sign-On 7.6 はカスタムプロバイダーを自動的に検出し、Keycloak の実行中にカスタムプロバイダーをホットデプロイする機能もサポートしていましたが、この動作はサポートされなくなりました。また、**providers** ディレクトリー内のプロバイダーまたは依存関係を変更した後は、ビルドを実行するか、自動ビルド機能を使用してサーバーを再起動する必要があります。

プロバイダーが使用する API によっては、プロバイダーに変更を加える必要がある場合もあります。詳細は、以下のセクションを参照してください。

6.1. JAVA EE から JAKARTA EE への移行

Keycloak のコードベースが Java EE (Enterprise Edition) から Jakarta EE に移行したことに伴い、さまざまな点が変更されました。Jakarta EE 10 をサポートするために、たとえば以下の Jakarta EE 仕様がすべてアップグレードされました。

- Jakarta Persistence 3.1
- Jakarta RESTful Web Services 3.1
- Jakarta Mail API 2.1
- Jakarta Servlet 6.0
- Jakarta Activation 2.1

Jakarta EE 10 は、クラウドネイティブの Java アプリケーションを構築するシンプルかつ先進的な軽量のアプローチを提供します。この取り組みにおける主な変更点は、**javax.*** から **jakarta.*** への名前空間の変更です。この変更は、**javax.security**、**javax.net**、**javax.crypto** など、JDK で直接提供される **javax.*** パッケージには適用されません。

さらに、セッション/ステートレス Bean などの Jakarta EE API がサポートされなくなりました。

6.2. サードパーティーの依存関係の削除

Red Hat build of Keycloak では、以下を含むいくつかの依存関係が削除されました。

- **openshift-rest-client**
- **okio-jvm**
- **okhttp**
- **commons-lang**
- **commons-compress**

- **jboss-dmr**
- **kotlin-stdlib**

また、Red Hat build of Keycloak は EAP をベースとしていないため、EAP の依存関係のほとんどが削除されました。そのため、上記のライブラリーを、Red Hat build of Keycloak にデプロイした独自のプロバイダーの依存関係として使用する場合、それらの JAR ファイルを、Keycloak ディストリビューションの **providers** ディレクトリーに明示的にコピーする必要があります。

6.3. JAX-RS リソースに対するコンテキストと依存性注入の無効化

より優れたランタイムを提供し、基礎となるスタックを可能な限り活用するために、**javax.ws.rs.core.Context** アノテーションを使用したコンテキストデータの注入ポイントがすべて削除されました。期待されるパフォーマンスの改善点として、要求のライフサイクル中にプロキシーインスタンスを複数回作成する必要がなくなり、実行時のリフレクションコードの量が大幅に削減されることが挙げられます。

現在の要求および応答オブジェクトにアクセスする必要がある場合は、**KeycloakSession** から直接それらのインスタンスを取得できるようになりました。

```
@Context
org.jboss.resteasy.spi.HttpRequest request;
@Context
org.jboss.resteasy.spi.HttpResponse response;
```

これは以下に置き換えられました。

```
KeycloakSession session = // obtain the session, which is usually available when creating a custom provider from a factory
KeycloakContext context = session.getContext();

HttpRequest request = context.getHttpRequest();
HttpResponse response = context.getHttpResponse();
```

追加のコンテキストデータは、**KeycloakContext** インスタンスを通じてランタイムから取得できます。

```
KeycloakSession session = // obtain the session
KeycloakContext context = session.getContext();
MyContextualObject myContextualObject = context.getContextObject(MyContextualObject.class);
```

6.4. データプロバイダーおよびモデルの非推奨メソッド

以前に非推奨となったいくつかのメソッドが、Red Hat build of Keycloak で削除されました。

- **RealmModel#searchForGroupByNameStream(String, Integer, Integer)**
- **UserProvider#getUsersStream(RealmModel, boolean)**
- **UserSessionPersisterProvider#loadUserSessions(int, int, boolean, int, String)**
- Streamification 処理用に追加されたインターフェイス (**RoleMapperModel.Streams** など)
- **KeycloakModelUtils#getClientScopeMappings**

- **KeycloakSession** の非推奨メソッド
- **UserQueryProvider#getUsersStream** メソッド

また、次のような他の変更も加えられました。

- **UserSessionProvider** の一部のメソッドが **UserLoginFailureProvider** に移動しました。
- フェデレーションストレージプロバイダークラスの **Streams** インターフェイスが非推奨になりました。
- Streamification - インターフェイスにストリームベースのメソッドのみが含まれるようになりました。
  たとえば、**GroupProvider** インターフェイスの場合:

  ```
  @Deprecated
  List<GroupModel> getGroups(RealmModel realm);
  ```

  これは以下に置き換えられました。

  ```
  Stream<GroupModel> getGroupsStream(RealmModel realm);
  ```

- 一貫したパラメーター順序付け - **RealmModel** が常に最初のパラメーターとなるように、メソッドに対して厳密なパラメーター順序付けが行われるようになりました。
  たとえば、**UserLookupProvider** インターフェイスの場合:

  ```
  @Deprecated
  UserModel getUserById(String id, RealmModel realm);
  ```

  これは以下に置き換えられました。

  ```
  UserModel getUserById(RealmModel realm, String id)
  ```

6.4.1. 変更されたインターフェイスのリスト

(**o.k.** は **org.keycloak.** パッケージを表します)

- **server-spi** モジュール
  - **o.k.credential.CredentialInputUpdater**
  - **o.k.credential.UserCredentialStore**
  - **o.k.models.ClientProvider**
  - **o.k.models.ClientSessionContext**
  - **o.k.models.GroupModel**
  - **o.k.models.GroupProvider**
  - **o.k.models.KeyManager**
  - **o.k.models.KeycloakSessionFactory**

- **o.k.models.ProtocolMapperContainerModel**
- **o.k.models.RealmModel**
- **o.k.models.RealmProvider**
- **o.k.models.RoleContainerModel**
- **o.k.models.RoleMapperModel**
- **o.k.models.RoleModel**
- **o.k.models.RoleProvider**
- **o.k.models.ScopeContainerModel**
- **o.k.models.UserCredentialManager**
- **o.k.models.UserModel**
- **o.k.models.UserProvider**
- **o.k.models.UserSessionProvider**
- **o.k.models.utils.RoleUtils**
- **o.k.sessions.AuthenticationSessionProvider**
- **o.k.storage.client.ClientLookupProvider**
- **o.k.storage.group.GroupLookupProvider**
- **o.k.storage.user.UserLookupProvider**
- **o.k.storage.user.UserQueryProvider**

- **server-spi-private** モジュール
  - **o.k.events.EventQuery**
  - **o.k.events.admin.AdminEventQuery**
  - **o.k.keys.KeyProvider**

6.4.2. ストレージ層でのリファクタリング

Red Hat build of Keycloak では、API の使用を簡素化するために大規模なリファクタリングが行われました。これは既存のコードに影響を与えます。以下の変更の中には、既存のコードの更新が必要なものもあります。以下のセクションでさらに詳しく説明します。

6.4.2.1. モジュール構造の変更

KeycloakSession のストレージ機能に関するいくつかのパブリック API が統合されました。その一部は移動、非推奨化、または削除されました。3 つの新しいモジュールが導入され、**server-spi**、**server-spi-private**、および **services** モジュールのデータ指向コードがそこに移動しました。

org.keycloak:keycloak-model-legacy

User Storage API など、レガシーストアのパブリック API がすべて含まれます。

org.keycloak:keycloak-model-legacy-private

ストレージの ***Manager** クラスなど、ユーザーストレージ管理に関連するプライベート実装が含まれます。

org.keycloak:keycloak-model-legacy-services

レガシーストアで直接動作するすべての REST エンドポイントが含まれます。

たとえば、新しいモジュールに移動したクラスをカスタムユーザーストレージプロバイダーの実装で使用している場合は、依存関係を更新して上記の新しいモジュールを含める必要があります。

6.4.2.2. KeycloakSession の変更

KeycloakSession が簡素化されました。**KeycloakSession** のいくつかのメソッドが削除されました。

KeycloakSession セッションには、特定のオブジェクトタイプのプロバイダーを取得するためのメソッドが含まれていました。たとえば、**UserProvider** には、**users()**、**userLocalStorage()**、**userCache()**、**userStorageManager()**、および **userFederatedStorage()** があります。このような状況は、各メソッドの正確な意味を理解する必要がある開発者にとってわかりにくい可能性があります。

これらの理由により、**users()** メソッドのみが **KeycloakSession** に保持されました。上記の他のすべての呼び出しは、このメソッドで置き換える必要があります。残りのメソッドは削除されました。同じ非推奨化のパターンが、**client()** や **groups()** などの他のオブジェクト領域のメソッドにも適用されます。末尾が ***StorageManager()** および ***LocalStorage()** のメソッドはすべて削除されました。次のセクションでは、これらの呼び出しを新しい API に移行する方法、またはレガシー API を使用する方法を説明します。

6.4.3. 既存のプロバイダーの移行

削除されたメソッドを呼び出さない場合、既存のプロバイダーを移行する必要はありません。これはほとんどのプロバイダーに当てはまります。

プロバイダーが削除されたメソッドを使用しているが、ローカルストレージと非ローカルストレージに依存していない場合は、削除された **userLocalStorage()** から **users()** メソッドに呼び出しを変更するのが最適です。ここでは、セマンティクスの変更に注意してください。ローカル設定でキャッシュが有効になっている場合、新しいメソッドにキャッシュが含まれるためです。

移行前: 削除された API へのアクセスがコンパイルされない

```
session.userLocalStorage();
```

移行後: 呼び出し元がレガシーストレージ API に依存していない場合、新しい API にアクセスする

```
session.users();
```

カスタムプロバイダーが特定のプロバイダーのモードを区別する必要がある稀なケースでは、非推奨オブジェクトへのアクセスを、**LegacyStoreManagers** データストアプロバイダーを使用して提供します。これに該当するのは、プロバイダーがローカルストレージに直接アクセスする場合や、プロバイダーがキャッシュをスキップする場合などです。この方法は、レガシーモジュールがデプロイメントの一部である場合にのみ使用できます。

移行前: 削除された API にアクセスする

```
session.userLocalStorage();
```

移行後: LegacyStoreManagers API を介して新機能にアクセスする

```
((LegacyDatastoreProvider) session.getProvider(DatastoreProvider.class)).userLocalStorage();
```

一部のユーザーストレージ関連の API は、便宜上 **org.keycloak.storage.UserStorageUtil** にラップされています。

6.4.4. RealmModel への変更

メソッド **getUserStorageProviders**、**getUserStorageProvidersStream**、**getClientStorageProviders**、**getClientStorageProvidersStream**、**getRoleStorageProviders**、および **getRoleStorageProvidersStream** が削除されました。これらのメソッドに依存するコードは、次のようにインスタンスをキャストする必要があります。

移行前: API が変更されたため、コードがコンパイルされない

```
realm.getClientStorageProvidersStream()...;
```

移行後: インスタンスをレガシーインターフェイスにキャストする

```
((LegacyRealmModel) realm).getClientStorageProvidersStream()...;
```

同様に、以前 **RealmModel** インターフェイスを実装していて、これらのメソッドを提供するコードは、新しいインターフェイス **LegacyRealmModel** を実装する必要があります。このインターフェイスは **RealmModel** のサブインターフェイスであり、古いメソッドを含んでいます。

移行前: 古いインターフェイスをコードが実装する

```
public class MyClass extends RealmModel {
    /* might not compile due to @Override annotations for methods no longer present
       in the interface RealmModel. / / ... */
}
```

移行後: 新しいインターフェイスをコードが実装する

```
public class MyClass extends LegacyRealmModel {
    /* ... */
}
```

6.4.5. UserCache インターフェイスがレガシーモジュールに移動

オブジェクトのキャッシュステータスがサービスに対して透過的になるため、インターフェイス **UserCache** がモジュール **keycloak-model-legacy** に移動しました。

レガシー実装に依存するコードは、**UserCache** に直接アクセスする必要があります。

移行前: コードがコンパイルされない [source,java,subs="+quotes"]

```
session**.userCache()**.evict(realm, user);
```

移行後: API を直接使用する

```
UserStorageUitl.userCache(session);
```

レルムの無効化をトリガーするには、**UserCache** API を使用する代わりに、イベントをトリガーすることを検討してください。

移行前: コードがキャッシュ API を使用する [source,java,subs="+quotes"]

```
UserCache cache = session.getProvider(UserCache.class);
if (cache != null) cache.evict(realm)();
```

移行後: 無効化 API を使用する

```
session.invalidate(InvalidationHandler.ObjectType.REALM, realm.getId());
```

6.4.6. ユーザーの認証情報管理

ユーザーの認証情報は、以前は **session.userCredentialManager().method(realm, user, ...)** を使用して管理されていました。新しい方法では、**user.credentialManager().method(...)** を使用します。この形式は、認証情報の機能をユーザーの API に近づけるものであり、レルムとストレージに関するユーザー認証情報の場所の知識を前提としていません。

古い API は削除されました。

移行前: 削除された API にアクセスする

```
session.userCredentialManager().createCredential(realm, user, credentialModel)
```

移行後: 新しい API にアクセスする

```
user.credentialManager().createStoredCredential(credentialModel)
```

カスタムの **UserStorageProvider** の場合、**UserModel** を返すときに、新しいメソッド **credentialManager()** を実装する必要があります。このメソッドは、**LegacyUserCredentialManager** のインスタンスを返す必要があります。

移行前: UserModel に新しいメソッド credentialManager() が必要なため、コードがコンパイルされない

```
public class MyUserStorageProvider implements UserLookupProvider, ... {
   /* ... */
   protected UserModel createAdapter(RealmModel realm, String username) {
      return new AbstractUserAdapter(session, realm, model) {
         @Override
         public String getUsername() {
            return username;
         }
```

```
        };
    }
}
```

移行後: レガシーストア用の API UserModel.credentialManager() を実装する

```
public class MyUserStorageProvider implements UserLookupProvider, ... {
    /* ... */
    protected UserModel createAdapter(RealmModel realm, String username) {
        return new AbstractUserAdapter(session, realm, model) {
            @Override
            public String getUsername() {
                return username;
            }

            @Override
            public SubjectCredentialManager credentialManager() {
                return new LegacyUserCredentialManager(session, realm, this);
            }
        };
    }
}
```

第7章 カスタムテーマの移行

7.1. 新しい管理コンソール

新しい管理コンソール (keycloak.v2) は React を使用して構築されています。古い管理コンソール (keycloak) は、少し前にサポートが終了した AngularJS 1.x で構築されていました。したがって、古いコンソールやそれを拡張するテーマから移行する方法はありません。基本テーマの管理コンソールも同じ理由でサポートされていません。

7.2. 新しいアカウントコンソール

新しいアカウントコンソール (keycloak.v2) は React を使用して構築されており、より優れたユーザーエクスペリエンスを提供します。古いアカウントコンソール (keycloak) は、基本的なサーバー側テンプレートを使用して構築されていました。したがって、古いコンソールやそれを拡張するテーマから移行する方法はありません。

7.3. ログインテーマの移行

テーマは、ログインページとアカウントコンソールの外観を設定するために使用されます。

カスタムテーマを作成または更新するとき、特にテンプレートをオーバーライドするときは、ビルトインテンプレートを参考にすると便利な場合があります。これらのテンプレートは **${KC_HOME}/lib/lib/main/org.keycloak.keycloak-themes-${KC_VERSION}.jar** にあり、標準の ZIP アーカイブツールを使用して開くことができます。

start-dev を使用してサーバーを開発モードで実行すると、テーマがキャッシュされないため、変更時にサーバーを再起動することなく簡単にテーマを操作できます。

カスタムテーマをインストールするには、テーマファイルを JAR ファイルとしてパッケージ化して **${KC_HOME}/providers** ディレクトリーに配置するか、ファイルを **${KC_HOME}/themes** ディレクトリーに直接コピーするかを選択できます。どちらの場合も、サーバーが求めるファイルおよびディレクトリー構造の詳細は、サーバー開発者ガイド を参照してください。

第8章 アップストリームの KEYCLOAK から RED HAT BUILD OF KEYCLOAK 24.0 への移行

バージョン 22 以降、Red Hat build of Keycloak とアップストリームの Keycloak の間にわずかな違いが存在します。相違点は次のとおりです。

- アップストリームの Keycloak の場合、配布アーティファクトは keycloak.org にあります。Red Hat build of Keycloak の場合、配布アーティファクトは Red Hat カスタマーポータル にあります。
- Oracle および MSSQL データベースドライバーは、アップストリームの Keycloak にバンドルされていますが、Red Hat build of Keycloak にはバンドルされていません。これらのドライバーをインストールする方法の詳細な手順は、データベースの設定 を参照してください。
- GELF ログハンドラーは、Red Hat build of Keycloak では使用できません。

移行プロセスは、移行する Keycloak のバージョンと Keycloak インストールのタイプによって異なります。詳細は、以下のセクションを参照してください。

8.1. KEYCLOAK バージョンの一致

移行プロセスは、移行する Keycloak のバージョンによって異なります。

- Keycloak プロジェクトのバージョンが Red Hat build of Keycloak のバージョンと一致する場合は、Red Hat カスタマーポータル の Red Hat build of Keycloak アーティファクトを使用して Keycloak を移行します。
- Keycloak プロジェクトのバージョンが古いバージョンの場合は、Keycloak アップグレードガイド を使用して Keycloak をアップグレードし、Red Hat build of Keycloak バージョンと一致するようにします。その後、Red Hat カスタマーポータル のアーティファクトを使用して Keycloak を移行します。
- Keycloak プロジェクトのバージョンが Red Hat build of Keycloak のバージョンより大きい場合、Red Hat build of Keycloak に移行することはできません。代わりに、Red Hat build of Keycloak の新しいデプロイメントを作成するか、Red Hat build of Keycloak の今後のリリースを待ちます。

8.2. KEYCLOAK インストールのタイプに基づく移行

一致するバージョンの Keycloak を用意したら、インストールのタイプに基づいて Keycloak を移行します。

- Keycloak を ZIP ディストリビューションからインストールした場合は、Red Hat カスタマーポータル のアーティファクトを使用して Keycloak を移行します。
- Keycloak Operator をデプロイした場合は、それをアンインストールし、Operator ガイド を使用して Red Hat build of Keycloak Operator をインストールします。CR は、アップストリームの Keycloak と Red Hat build of Keycloak との間で互換性があります。
- カスタムサーバーコンテナーイメージを作成した場合は、Red Hat build of Keycloak イメージを使用してそれを再ビルドします。コンテナー内での Keycloak の実行 を参照してください。

第9章 その他の主な変更点

9.1. クラスパスでデフォルトで利用可能な JAVASCRIPT エンジン

以前のバージョンでは、Java 17 で Keycloak を JavaScript プロバイダー (スクリプト認証システム、JavaScript 認可ポリシー、または OIDC および SAML クライアント用のスクリプトプロトコルマッパー) とともに使用する場合、JavaScript エンジンをディストリビューションにコピーする必要がありました。これは不要になりました。Red Hat build of Keycloak サーバーでは、Nashorn JavaScript エンジンをデフォルトで利用できるためです。スクリプトプロバイダーをデプロイする場合、Nashorn のスクリプトエンジンとその依存関係を Red Hat build of Keycloak ディストリビューションにコピーしないことを推奨します。

9.2. 名前が変更された KEYCLOAK 管理クライアントアーティファクト

Jakarta EE へのアップグレードを経て、Keycloak 管理クライアントのアーティファクトが、長期的な保守性を考慮して、よりわかりやすい名前に変更されました。ただし、2 つの別個の Keycloak 管理クライアントが引き続き存在しています。1 つは Jakarta EE のサポートを備え、もう 1 つは Java EE のサポートを備えています。

org.keycloak:keycloak-admin-client-jakarta アーティファクトはリリースされなくなりました。Jakarta EE のサポートを備えたデフォルトの Keycloak 管理クライアントは、**org.keycloak:keycloak-admin-client** (バージョン 24.0.0 以降) です。

Java EE のサポートを備えた新しいアーティファクトは、**org.keycloak:keycloak-admin-client-jee** です。

9.2.1. Jakarta EE のサポート

Java EE のサポートを備えた新しいアーティファクトは、**org.keycloak:keycloak-admin-client-jee** です。Jakarta EE のサポート

移行前:

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client-jakarta</artifactId>
    <version>18.0.0.redhat-00001</version>
</dependency>
```

移行後:

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
    <version>22.0.0.redhat-00001</version>
</dependency>
```

9.2.2. Java EE のサポート

移行前:

```
<dependency>
```

```
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client</artifactId>
    <version>18.0.0.redhat-00001</version>
</dependency>
```

移行後:

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-admin-client-jee</artifactId>
    <version>22.0.0.redhat-00001</version>
</dependency>
```

9.3. クライアントの ADVANCED SETTINGS コンボからの NEVER EXPIRES オプションの削除

Never expires オプションが **Advanced Settings** クライアントタブのすべてのコンボから削除されました。各種の有効期間やアイドルタイムアウトは、無期限ではなく、一般的なユーザーセッションやレルム値によって制限されるため、このオプションは誤解を招いていました。したがって、このオプションは残りの 2 つのオプションである **Inherits from the realm settings** (一般的なレルムタイムアウトがクライアントで使用される) と **Expires in** (値がクライアントでオーバーライドされる) のために削除されました。内部的には、**Never expires** は **-1** で表されていました。この値は管理コンソールに警告とともに表示されるようになり、管理者が直接設定することはできません。

9.4. メールアドレスの新しいルールおよび制限の検証

Red Hat build of Keycloak に、メールアドレス作成に関する新しいルールが追加され、メールアドレス作成時に ASCII 文字を使用できるようになりました。また、メールアドレスのローカル部分 (@ の前) に対して 64 文字の制限が新しく設けられました。そのため、下位互換性を考慮してメールアドレスのローカル部分の最大長を設定するために、新しいパラメーター **--spi-user-profile-declarative-user-profile-max-email-local-part-length** が追加されました。デフォルト値は 64 です。

```
kc.sh start --spi-user-profile-declarative-user-profile-max-email-local-part-length=100
```