



Red Hat build of Keycloak 22.0

サーバー管理ガイド

法律上の通知

Copyright © 2024 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

概要

このガイドには、Red Hat build of Keycloak 22.0 の設定に関する管理者向け情報が記載されていません。

目次

多様性を受け入れるオープンソースの強化	5
第1章 RED HAT BUILD OF KEYCLOAK の機能と概念	6
1.1. 機能	6
1.2. RED HAT BUILD OF KEYCLOAK の基本操作	7
1.3. コアとなる概念および用語	7
第2章 最初の管理者の作成	10
2.1. ローカルホストでのアカウントの作成	10
2.2. リモートでアカウントの作成	10
第3章 レルムの設定	11
3.1. 管理コンソールの使用	11
3.2. マスターレルム	12
3.3. レルムの作成	13
3.4. レルムでの SSL の設定	15
3.5. レルムへのメールの設定	16
3.6. テーマの設定	18
3.7. 国際化の有効化	19
3.8. ログインオプションの制御	21
3.9. レルムキーの設定	29
第4章 外部ストレージの使用	34
4.1. プロバイダーの追加	34
4.2. プロバイダーの失敗の処理	34
4.3. LDAP (LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL) および ACTIVE DIRECTORY	35
4.4. SSSD および FREEIPA IDENTITY MANAGEMENT の統合	41
4.5. カスタムプロバイダー	44
第5章 ユーザーの管理	45
5.1. ユーザーの作成	45
5.2. ユーザーの認証情報の定義	45
5.3. ユーザー属性の設定	47
5.4. ユーザーの自己登録の許可	48
5.5. ログイン時に必要なアクションの定義	54
5.6. アプリケーションが開始したアクション	56
5.7. ユーザーの検索	58
5.8. ユーザーの削除	58
5.9. ユーザーによるアカウントの削除の有効化	59
5.10. ユーザーの権限借用	63
5.11. RECAPTCHA の有効化	64
5.12. ユーザープロファイルの定義	66
5.13. RED HAT BUILD OF KEYCLOAK が収集する個人データ	91
第6章 ユーザーセッションの管理	93
6.1. セッションの管理	93
6.2. アクティブなセッションの取り消し	94
6.3. セッションおよびトークンのタイムアウト	95
6.4. オフラインアクセス	101
6.5. オフラインセッションの事前読み込み	102
6.6. 一時的なセッション	103
第7章 ロールとグループを使用した権限の割り当て	104

7.1. レルムロールの作成	104
7.2. クライアントロール	105
7.3. ロールの複合ロールへの変換	105
7.4. ロールマッピングの割り当て	106
7.5. デフォルトロールの使用	107
7.6. ロールマッピングのマッピング	108
7.7. グループ	109
第8章 認証の設定	113
8.1. パスワードポリシー	113
8.2. ワンタイムパスワード (OTP) ポリシー	115
8.3. 認証フロー	118
8.4. ユーザーセッションの制限	134
8.5. KERBEROS	136
8.6. X.509 クライアント証明書ユーザー認証	143
8.7. W3C WEB AUTHENTICATION (WEBAUTHN)	153
8.8. リカバリーコード (RECOVERYCODES)	163
8.9. 条件付きフローの条件	163
第9章 アイデンティティプロバイダーの統合	167
9.1. ブローカーの概要	167
9.2. デフォルトのアイデンティティプロバイダー	169
9.3. 一般的な設定	169
9.4. ソーシャルアイデンティティプロバイダー	173
9.5. OPENID CONNECT V1.0 アイデンティティプロバイダー	193
9.6. SAML V2.0 アイデンティティプロバイダー	198
9.7. クライアント提案されたアイデンティティプロバイダー	202
9.8. クレームとアサーションのマッピング	203
9.9. 利用可能なユーザーセッションデータ	204
9.10. FIRST LOGIN FLOW	205
9.11. 外部 IDP トークンの取得	209
9.12. アイデンティティブローカーのログアウト	209
第10章 SSO プロトコル	210
10.1. OPENID CONNECT	210
10.2. SAML	220
10.3. OPENID CONNECT と SAML の比較	222
10.4. DOCKER REGISTRY V2 認証	222
第11章 管理コンソールへのアクセス制御	224
11.1. マスターレルムアクセス制御	224
11.2. 専用レルム管理コンソール	225
第12章 OPENID CONNECT および SAML クライアントの管理	226
12.1. OPENID CONNECT クライアントの管理	226
12.2. SAML クライアントの作成	248
12.3. クライアントリンク	255
12.4. OIDC トークンおよび SAML アサーションマッピング	256
12.5. クライアントアダプター設定の生成	258
12.6. クライアントスコープ	259
12.7. クライアントポリシー	264
第13章 VAULT を使用したシークレットの取得	270
13.1. キーリゾルバー	270

第14章 イベントを追跡する監査の設定	272
14.1. ユーザーイベントの監査	272
14.2. 管理イベントの監査	278
第15章 セキュリティー脅威の軽減	280
15.1. ホスト	280
15.2. 管理エンドポイントおよび管理コンソール	280
15.3. 総当たり攻撃	280
15.4. 読み取り専用ユーザー属性	283
15.5. クリックジャッキング	284
15.6. SSL/HTTPS 要件	285
15.7. CSRF 攻撃	285
15.8. 特定のリダイレクト URI	286
15.9. FAPI コンプライアンス	286
15.10. 不正アクセスおよびトークンの更新	286
15.11. 侵害された認可コード	287
15.12. オープンリダイレクター	287
15.13. パスワードデータベースの漏洩	287
15.14. 制限の範囲	287
15.15. トークンオーディエンスの制限	287
15.16. 認証セッションの制限	288
15.17. SQL インジェクション攻撃	288
第16章 アカウントコンソール	289
16.1. アカウントコンソールへのアクセス	289
16.2. サインイン方法の設定	289
16.3. デバイスのアクティビティーの表示	293
16.4. アイデンティティープロバイダーアカウントの追加	293
16.5. 他のアプリケーションへのアクセス	294
16.6. グループメンバーシップの表示	294
第17章 管理 CLI	296
17.1. 管理 CLI のインストール	296
17.2. 管理 CLI の使用	296
17.3. 認証	297
17.4. 代替設定の使用	298
17.5. 基本操作およびリソース URI	298
17.6. レルム操作	300
17.7. ロール操作	306
17.8. クライアント操作	310
17.9. ユーザー操作	312
17.10. グループ操作	316
17.11. アイデンティティープロバイダー操作	318
17.12. ストレージプロバイダー操作	321
17.13. マッパーの追加	323
17.14. 認証操作	325

多様性を受け入れるオープンソースの強化

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

Red Hat build of Keycloak は、Web アプリおよび RESTful Web サービス用のシングルサインオンソリューションです。Red Hat build of Keycloak の目的は、アプリケーション開発者が組織にデプロイしたアプリケーションおよびサービスを容易に保護できるように、セキュリティーをシンプルにすることです。通常、開発者は自分で作成する必要のあるセキュリティー機能は、追加設定なしで提供され、組織の個々の要件に簡単に調整できます。Red Hat build of Keycloak は、ログイン、登録、管理、アカウント管理用にカスタマイズ可能なユーザーインターフェイスを提供します。Red Hat build of Keycloak を統合プラットフォームとして使用して、既存の LDAP および Active Directory サーバーにフックすることもできます。Facebook や Google などのサードパーティーアイデンティティプロバイダーに認証を委譲することもできます。

1.1. 機能

Red Hat build of Keycloak には次の機能があります。

- ブラウザーアプリケーションに対するシングルサインオンおよびシングルサインアウト。
- OpenID Connect のサポート。
- OAuth 2.0 サポート。
- SAML サポート。
- ID ブローカー: 外部 OpenID Connect または SAML ID プロバイダーでの認証。
- ソーシャルログイン: Google、GitHub、JWT、その他のソーシャルネットワークでのログインの有効化。
- ユーザーフェデレーション: LDAP および Active Directory サーバーからユーザーを同期します。
- Kerberos ブリッジ: Kerberos サーバーにログインしたユーザーの自動認証。
- ユーザー、ロール、ロールマッピング、クライアント、および設定の一元管理のための管理コンソール。
- ユーザーがアカウントを一元管理できるアカウントマネジメントコンソール。
- テーマサポート: アプリケーションおよびブランディングと統合するユーザー向けページのカスタマイズ。
- 2 要素認証: Google Authenticator または FreeOTP による TOTP/HOTP のサポート
- ログインフロー: オプションのユーザーの自己登録、パスワードのリカバリー、電子メールの確認、パスワードの更新など。
- セッション管理: 管理者およびユーザー自身がユーザーセッションを表示および管理する機能。
- トークンマッパー: トークンとステートメントへのユーザーの属性、ロールなどのマップ。
- レルム、アプリケーション、ユーザーごとの失効前ポリシー。
- CORS サポート - クライアントアダプターに組み込まれた CORS のサポート。
- JavaScript アプリケーション、JBoss EAP などのクライアントアダプター。

- OpenID Connect Relying Party ライブラリーまたは SAML 2.0 サービスプロバイダーライブラリーを持つすべてのプラットフォーム/言語のサポート。

1.2. RED HAT BUILD OF KEYCLOAK の基本操作

Red Hat build of Keycloak は、ネットワーク上で管理する個別サーバーです。アプリケーションは、このサーバーによって提供され、保護されるように設定されています。Red Hat build of Keycloak は、[OpenID Connect](#) や [SAML 2.0](#) などのオープンプロトコル標準を使用してアプリケーションを保護します。ブラウザアプリケーションは、ユーザーのブラウザをアプリケーションから Red Hat build of Keycloak 認証サーバーにリダイレクトし、認証情報の入力を求めます。ユーザーがアプリケーションから完全に分離され、アプリケーションにはユーザーの認証情報が確認されないため、このリダイレクトは重要です。アプリケーションには、暗号で署名された認証情報トークンまたはアサーションが与えられます。これらのトークンには、ユーザー名、アドレス、電子メール、他のプロフィールデータなどのアイデンティティ情報を使用できます。また、アプリケーションによる認可決定を実行できるように、パーミッションデータを保持することもできます。これらのトークンを使用して、REST ベースのサービスでセキュアな呼び出しを行うこともできます。

1.3. コアとなる概念および用語

Red Hat build of Keycloak を使用して Web アプリケーションと REST サービスを保護する前に、中核となるこれらの概念と用語について検討してください。

ユーザー

ユーザーは、システムにログインできるエンティティです。電子メール、ユーザー名、アドレス、電話番号、および日目などの属性を関連付けることができます。グループメンバーシップを割り当てることができ、特定のロールをそれらに割り当てることができます。

認証

ユーザーを特定し、検証するプロセスです。

認可

ユーザーに付与するプロセスです。

credentials

認証情報は、Red Hat build of Keycloak がユーザーのアイデンティティを確認するために使用するデータです。たとえば、パスワード、ワンタイムパスワード、デジタル証明書、またはフィンガープリントなどが挙げられます。

ロール

ロールはユーザーのタイプまたはカテゴリを識別します。**Admin**、**user**、**manager**、**employee** は、組織に存在する可能性のある通常のロールすべてです。アプリケーションは通常、多くの場合、個々のユーザーではなく、特定のロールにアクセスおよびパーミッションを割り当てます。これは、ユーザーの処理は複雑で、管理が困難となるためです。

ユーザーロールのマッピング

ユーザーロールのマッピングは、ロールとユーザー間のマッピングを定義します。ユーザーをゼロ以上のロールに関連付けることができます。このロールマッピング情報は、アプリケーションが管理するさまざまなリソースのアクセスパーミッションを決定できるように、トークンとアサーションにカプセル化できます。

複合ロール

複合ロールは、他のロールに関連付けることができるロールです。たとえば、**superuser** の複合ロールを **sales-admin** ロールおよび **order-entry-admin** ロールに関連付けることができます。ユーザーが **superuser** ロールにマップされている場合は、**sales-admin** ロールおよび **order-entry-admin** ロールも継承されます。

グループ

グループはユーザーのグループを管理します。グループに対して属性を定義できます。ロールをグループにマッピングすることもできます。グループのメンバーになるユーザーは、そのグループで定義される属性とロールマッピングを継承します。

レルム

レルムは、一連のユーザー、認証情報、ロール、およびグループを管理します。ユーザーはレルムに属し、レルムにログインします。レルムは相互に分離され、制御するユーザーのみを管理および認証できます。

クライアント

クライアントは、Red Hat build of Keycloak にユーザー認証を要求できるエンティティです。多くの場合、クライアントは Red Hat build of Keycloak を使用して自身を保護し、シングルサインオンソリューションを提供するアプリケーションとサービスです。クライアントは、Red Hat build of Keycloak により保護されるネットワーク上の他のサービスを安全に呼び出すために、アイデンティティ情報またはアクセストークンを要求するエンティティでもあります。

クライアントアダプター

クライアントアダプターは、アプリケーション環境にインストールするプラグインで、Red Hat build of Keycloak と通信し、アプリケーション環境を保護します。Red Hat build of Keycloak には、ダウンロードできるさまざまなプラットフォーム用のアダプターが多数含まれています。また、サードパーティーのアダプターも、対象外の環境で取得できます。

consent

Consent は、クライアントが認証プロセスに参加する前に、ユーザーにクライアントにパーミッションを付与する場合です。ユーザーが認証情報を提供すると、Red Hat build of Keycloak はログインを要求しているクライアントとユーザーに要求されているアイデンティティ情報を識別する画面をポップアップします。ユーザーは、要求を付与するかどうかを決定できます。

クライアントスコープ

クライアントが登録されたら、そのクライアントの プロトコルマッパーとロールスコープマッピングを定義する必要があります。多くの場合、クライアントスコープを保存し、共通の設定を共有することで新しいクライアントの作成を簡素化します。これは、一部の要求またはロールを **scope** パラメーターの値に基づいて条件付きで要求する場合にも便利です。Red Hat build of Keycloak は、このためのクライアントスコープの概念を提供します。

クライアントロール

クライアントは、それら固有のロールを定義できます。これは基本的に、クライアント専用のロール名前空間です。

ID トークン

ユーザーに関する情報を提供するトークン。OpenID Connect 仕様の一部。

アクセストークン

呼び出されるサービスへのアクセスを付与する HTTP リクエストの一部として提供できるトークン。これは、OpenID Connect および OAuth 2.0 仕様の一部です。

アサーション

ユーザーに関する情報。これは通常、認証されたユーザーに関連するアイデンティティメタデータを提供する SAML 認証応答に含まれる XML Blob に関連します。

サービスアカウント

各クライアントには、アクセストークンの取得を可能にする組み込み service account があります。

直接付与

REST 呼び出しでユーザーの代わりにアクセストークンを取得する方法。

プロトコルマッパー

各クライアントについて、OIDC トークンまたは SAML アサーションに保存される要求とアサーションを調整できます。プロトコルマッパーを作成および設定して、クライアントごとにこれを行います。

セッション

ユーザーがログインすると、セッションがログインセッションを管理します。セッションには、ユーザーがログインした時や、そのセッション中に単一署名に参加したアプリケーションなどの情報が含まれます。管理者およびユーザーの両方がセッション情報を表示できます。

ユーザーフェデレーションプロバイダー

Red Hat build of Keycloak はユーザーを保存および管理できます。多くの場合、ユーザーと認証情報を格納する LDAP または Active Directory サービスがすでにあります。Red Hat build of Keycloak を指定して、これらの外部ストアから認証情報を検証し、アイデンティティ情報を取り込むことができます。

アイデンティティプロバイダー

アイデンティティプロバイダー (IDP) はユーザーを認証できるサービスです。Red Hat build of Keycloak は IDP です。

ID プロバイダーフェデレーション

Red Hat build of Keycloak は、1つ以上の IDP に認証を委譲するように設定できます。Facebook または Google+ でのソーシャルログインは、アイデンティティプロバイダーのフェデレーションの例です。Red Hat build of Keycloak をフックして、他の OpenID Connect または SAML 2.0 IDP に認証を委譲することもできます。

アイデンティティプロバイダーマッパー

IDP フェデレーションを行う場合は、受信したトークンとアサーションを user および session 属性にマッピングできます。これは、外部の IDP から認証を要求するクライアントに ID 情報を伝播するのに役立ちます。

必須アクション

必須アクションは、ユーザーが認証プロセス中に実行する必要があるアクションです。ユーザーは、これらのアクションが完了するまで認証プロセスを完了できません。たとえば、管理者はユーザーが毎月パスワードをリセットできるようにスケジュールすることが可能です。これらの全ユーザーに対して、**update password** に必須アクションが設定されます。

認証フロー

認証フローは、システムの特定の側面と対話するときにユーザーが実行する必要があるフローです。ログインフローは必要な認証情報タイプを定義できます。登録フローは、ユーザーが入力しなければならないプロフィール情報や、ボットをフィルターするのに reCAPTCHA などのプロフィール情報を定義します。認証情報リセットフローは、パスワードをリセットする前にユーザーが行うべきアクションを定義します。

イベント

イベントは、管理者が表示やフックを表示できる監査ストリームです。

テーマ

Red Hat build of Keycloak によって提供されるすべての画面は、テーマに基づいています。テーマは、必要に応じて書き換え可能な HTML テンプレートとスタイルシートを定義します。

第2章 最初の管理者の作成

Red Hat build of Keycloak をインストールした後、それを管理するための完全な権限を持つ **super** 管理者として機能する管理者アカウントが必要です。このアカウントを使用すると、Red Hat build of Keycloak 管理コンソールにログインして、レルムとユーザーを作成し、Red Hat build of Keycloak が保護するアプリケーションを登録できます。

2.1. ローカルホストでのアカウントの作成

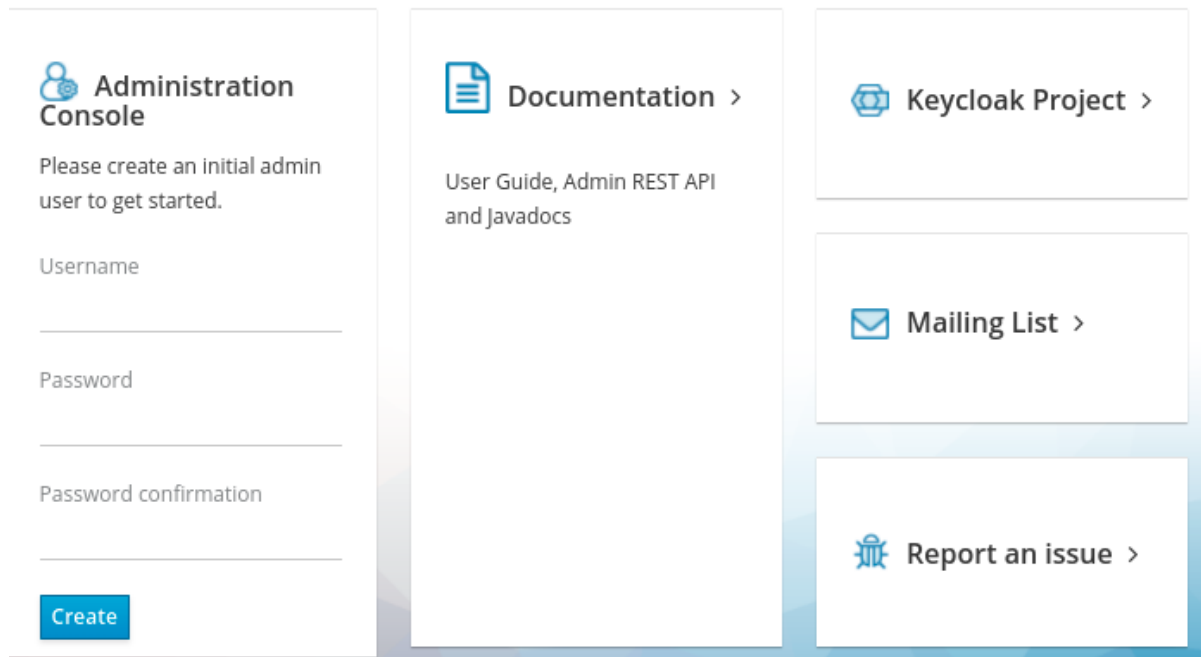
サーバーが **localhost** からアクセスできる場合は、以下の手順を実行します。

手順

1. Web ブラウザーで、<http://localhost:8080> URL に移動します。
2. 再呼び出しできるユーザー名とパスワードを指定します。

Welcome ページ

Welcome to **Keycloak**



Administration Console

Please create an initial admin user to get started.

Username

Password

Password confirmation

Create

Documentation >

User Guide, Admin REST API and Javadocs

Keycloak Project >

Mailing List >

Report an issue >

2.2. リモートでアカウントの作成

localhost アドレスからサーバーにアクセスできない場合、またはコマンドラインから Red Hat build of Keycloak を起動する場合は、**KEYCLOAK_ADMIN** および **KEYCLOAK_ADMIN_PASSWORD** 環境変数を使用して初期管理者アカウントを作成します。

以下に例を示します。

```
export KEYCLOAK_ADMIN=<username>
export KEYCLOAK_ADMIN_PASSWORD=<password>

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

第3章 レルムの設定

管理コンソールの管理者アカウントを取得したら、レルムを設定できます。レルムは、ユーザー、アプリケーション、ロール、グループなどのオブジェクトを管理するスペースです。ユーザーはレルムに属し、レルムにログインします。データベースにスペースがある限り、1つの Red Hat build of Keycloak デプロイメントで任意の数の定義、保存、管理できます。

3.1. 管理コンソールの使用

Red Hat build of Keycloak 管理コンソールでレルムを設定し、ほとんどの管理タスクを実行します。

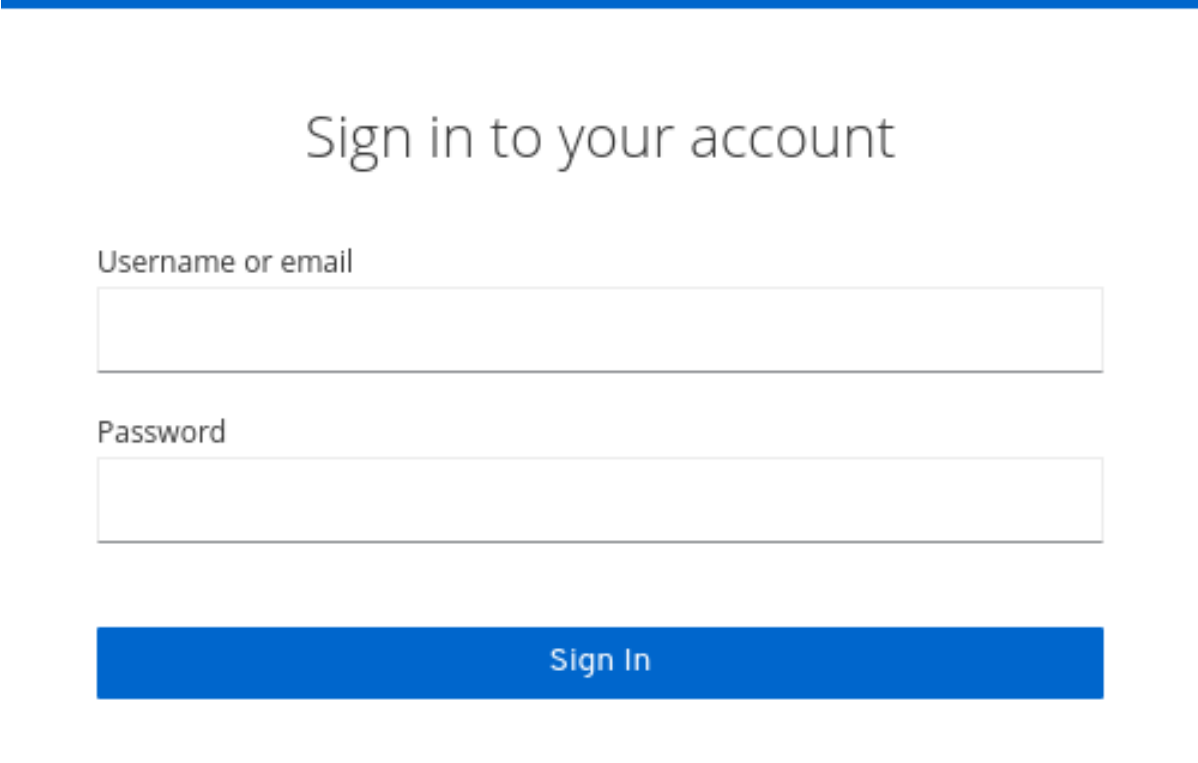
前提条件

- 管理者アカウントが必要です。 [最初の管理者の作成について](#) 参照してください。

手順

1. 管理コンソールの URL に移動します。
たとえば、ローカルホストの場合は、次の URL を使用します: <http://localhost:8080/admin/>

ログインページ



Sign in to your account

Username or email

Password

Sign In

2. Welcome Page で作成したユーザー名やパスワードを入力するか、bin ディレクトリーに **add-user-keycloak** スクリプトを入力します。このアクションは管理コンソールを表示します。

管理コンソール

3. 使用可能なメニューおよびその他のオプションに注意してください。

- **Master** というラベルのメニューをクリックし、管理するレルムを選択するか、新規レルムを作成します。
- 右上の一覧をクリックしてアカウントを表示するか、ログアウトします。
- 疑問符 ? アイコンにカーソルを合わせ、そのフィールドを記述するツールチップテキストを表示します。上記のイメージはアクションのツールチップを示しています。
- 疑問符をクリックしますか ? アイコンをクリックすると、そのフィールドを説明するツールチップテキストが表示されます。上記のイメージはアクションのツールチップを示しています。



注記

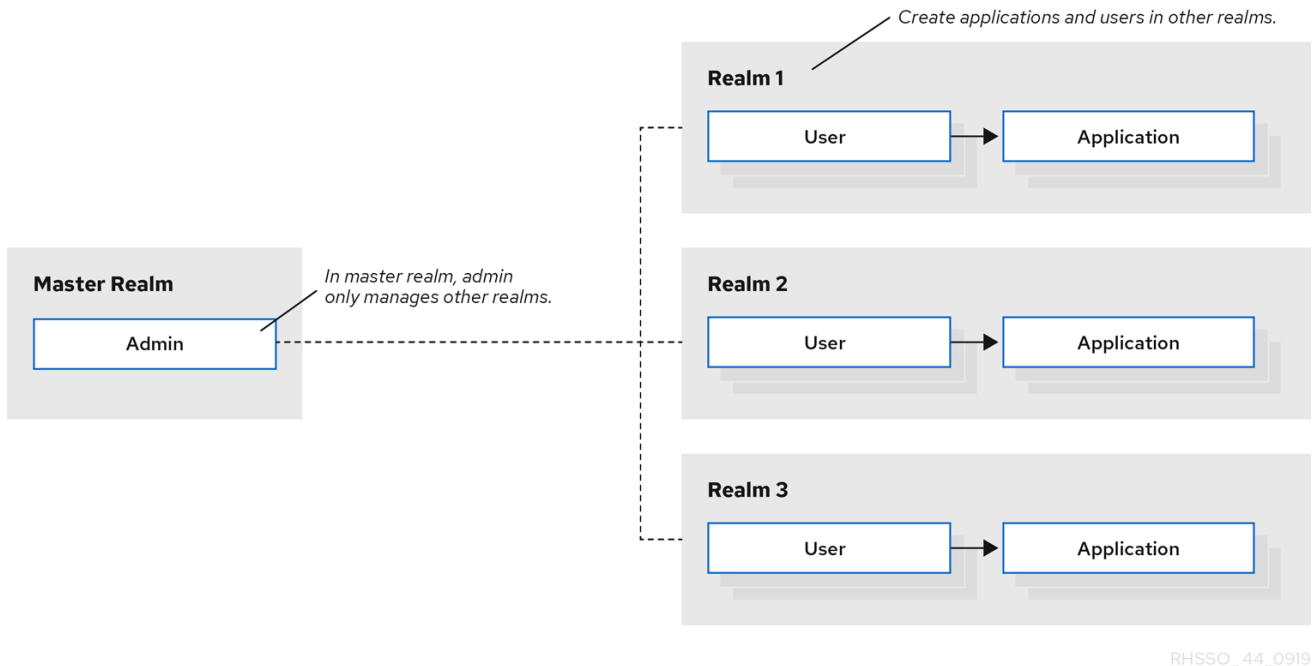
管理コンソールからのファイルのエクスポートは、サーバー間のバックアップやデータ転送には適していません。サーバー間のバックアップまたはデータ転送には、ブート時のエクスポートのみを使用できます。

3.2. マスターレルム

管理コンソールでは、2種類のレルムが存在します。

- **Master realm:** このレalmは、Red Hat build of Keycloak の初回起動時に作成されます。これには、初回ログイン時に作成した管理者アカウントが含まれます。マスターレalmは、システムでレalmの作成および管理にのみ使用してください。
- **他のレalm:** マスターレalmの管理者が、これらのレalmが作成されます。管理者は、これらのレalmで、組織と必要なアプリケーションのユーザーを管理します。アプリケーションはユーザーが所有します。

レalmおよびアプリケーション



RHSSO_44_0919

レalmは相互に分離され、制御するユーザーのみを管理および認証できます。このセキュリティーモデルに従うことで、誤って変更を回避し、ユーザーアカウントのクリメンティションに従って、現在のタスクの正常な完了に必要な特権と電源へのアクセスを許可します。

関連情報

- マスターレalmを無効にして、作成した新しいレalm内で管理者アカウントを定義する場合は、[専用レalm管理コンソール](#)を参照してください。各レalmには独自の専用管理コンソールがあり、ローカルアカウントでログインできます。

3.3. レalmの作成

レalmを作成して、ユーザーを作成し、アプリケーションを使用するパーミッションを付与できる管理スペースを提供します。初回ログイン時に、通常はマスターレalm (他のレalmを作成する最上位のレalm) です。

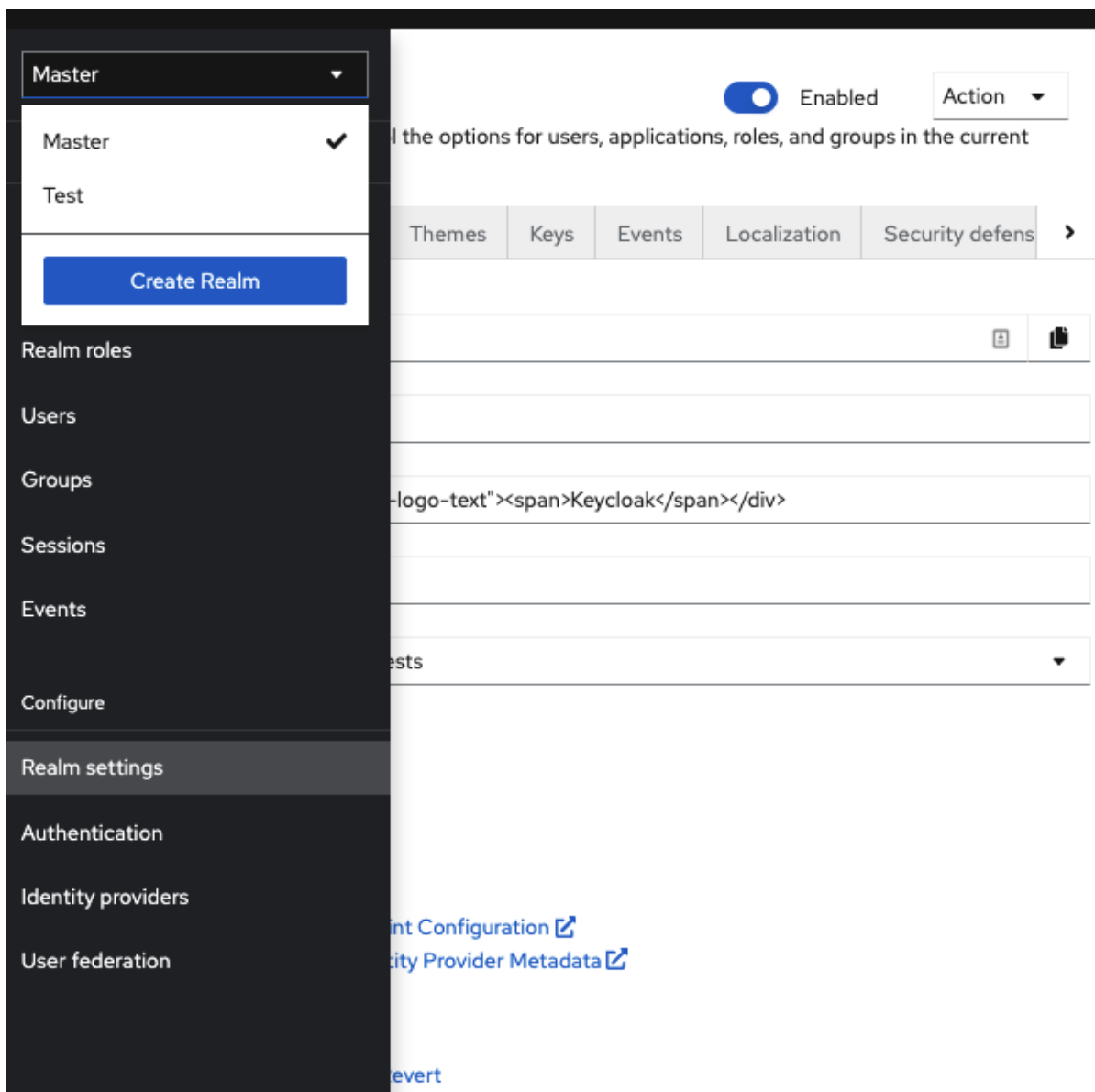
必要なレalmを決定する際には、ユーザーおよびアプリケーションに必要な分離の種類を考慮してください。たとえば、所属企業の従業員および顧客向けに別のレalmにレalmを作成することができます。従業員は従業員のレalmにログインし、内部の会社アプリケーションにしかアクセスできません。顧客が顧客のレalmにログインし、顧客がアクセスするアプリケーションと対話することしかできません。

手順

1. 左側のペインの上部を示します。

2. **Create Realm** をクリックします。

レルムメニューの追加



3. レルムの名前を入力します。
4. **Create** をクリックします。

レルムの作成

Create realm

A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

Resource file

Drag a file here or browse to upload Browse... Clear

1

Upload a JSON file

Realm name *

Enabled On

Create Cancel

これで、現在のレalmが作成したレalmに設定されます。メニュー内のレalm名をクリックすると、レalmを切り替えることができます。

3.4. レalmでの SSL の設定

各レalmには関連する SSL モードがあり、レalmと対話するための SSL/HTTPS 要件を定義します。レalmと相互作用するブラウザとアプリケーションは、SSL モードで定義された SSL/HTTPS 要件を有効にするか、サーバーと対話できません。

手順

1. メニューで **Realm Settings** をクリックします。
2. **General** タブをクリックします。

General タブ

Master Enabled Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

<
General
Login
Email
Themes
Keys
Events
Localization
Security defens
>

Realm ID *

Display name

HTML Display name

Frontend URL ?

Require SSL ? External requests ▾

User-managed access ? Off

User Profile Enabled ? Off

Endpoints ?
[OpenID Endpoint Configuration](#)

[SAML 2.0 Identity Provider Metadata](#)

Save
Revert

3. **Require SSL** を、以下の SSL モードのいずれかに設定します。

- **External requests:** ユーザーは、**localhost**、**127.0.0.1**、**10.xxx**、**192.168.xx**、**172.16.xx** などのプライベート IP アドレスを使用している限り、SSL を使用せずに Red Hat build of Keycloak と対話できます。非プライベート IP アドレスから SSL を使用せずに Red Hat build of Keycloak にアクセスしようとすると、エラーが発生します。
- **None:** Red Hat build of Keycloak は SSL を必要としません。この選択肢は、実験時にのみ適用され、このデプロイメントをサポートする予定はありません。
- **All requests:** Red Hat build of Keycloak では、すべての IP アドレスに SSL が必要です。

3.5. レルムへのメールの設定

Red Hat build of Keycloak は、ユーザーがパスワードを忘れた場合、または管理者がサーバーイベントに関する通知を受け取る必要がある場合に、ユーザーにメールアドレスを確認するためにメールを送信します。Red Hat build of Keycloak がメールを送信できるようにするには、Red Hat build of Keycloak に SMTP サーバー設定を提供します。

手順

1. メニューで **Realm Settings** をクリックします。

2. Email タブをクリックします。

Email tab

Master

Enabled

Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

<

General

Login

Email

Themes

Keys

Events

Localization

Security defens

>

Template

From *

Sender email address

From display name ⓘ

Display name for Sender email address

Reply to

Reply to email address

Reply to display name ⓘ

Display name for "reply to" email address

Envelope from ⓘ

Sender envelope email address

Connection & Authentication

Host *

SMTP host

Port

SMTP port (defaults to 25)

Encryption

Enable SSL
 Enable StartTLS

Authentication

Disabled

Save

Test connection

Revert

3. フィールドに入力し、必要に応じてスイッチを切り替えます。

テンプレート

From

from は、送信メールに **From** SMTP ヘッダーに使用されるアドレスを示します。

From Display Name

From display name では 使いやすい電子メールアドレスエイリアスを設定できます (オプション)。設定されていない場合、プレーン **From** のメールアドレスが電子メールクライアントに表示されません。

返信先

Reply to は、送信メールの **Reply-To** SMTP-Header に使用されるアドレス (任意) を示します。設定されていない場合、プレーン **From** メールアドレスが使用されます。

Reply To Display Name

Reply to display name により、ユーザーフレンドリーなメールアドレスエイリアス (オプション) を設定できます。設定しない場合には、プレーンな **Reply To** のメールアドレスが表示されます。

Envelope from

From は、送信メールの **Return-Path** Header に使用される **Bounce Address** アドレスを示します (任意)。

接続と認証

ホスト

ホスト は、電子メールの送信に使用される SMTP サーバーのホスト名を示します。

Port

Port は SMTP サーバーポートを示します。

暗号化

これらのチェックボックスのいずれかにチェックを入れると、特に SMTP サーバーが外部ネットワーク上にある場合に、ユーザー名とパスワードを回復するための電子メールの送信がサポートされます。多くの場合、**ポート** を SSL/TLS のデフォルトポートである 465 に変更する必要があります。

認証

SMTP サーバーで認証が必要な場合は、このスイッチを **ON** に設定します。プロンプトが表示されたら、**Username** および **Password** を指定します。**Password** フィールドの値は、外部 **ボールド** の値を参照できます。

3.6. テーマの設定

特定のレルムについて、テーマを使用して Red Hat build of Keycloak の UI の外観を変更できます。

手順

1. メニューで **Realm Setting** をクリックします。
2. **Themes** タブをクリックします。

themes タブ

Master Enabled Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

◀ gin Email Themes Keys Events Localization Security defenses Sessions ▶

Login theme ? keycloak ▾

Account theme ? keycloak ▾

Admin console theme ? keycloak ▾

Email theme ? Select a theme ▾

Save Revert

- 各 UI カテゴリーに対して必要なテーマを選択し、**Save** をクリックします。

ログインテーマ

ユーザー名エントリー、OTP エントリー、新しいユーザー登録、およびその他の同様の画面。

アカウントテーマ

各ユーザーに、ユーザーアカウントの管理 UI があります。

管理コンソールのテーマ

Red Hat build of Keycloak 管理コンソールのスキン。

メールテーマ

Red Hat build of Keycloak がメールを送信する必要がある場合は、このテーマで定義されたテンプレートを使用してメールを作成します。

関連情報

- [サーバー開発者ガイド](#) では、新しいテーマを作成する方法、または既存のテーマを変更する方法について説明しています。

3.7. 国際化の有効化

Red Hat build of Keycloak のすべての UI 画面は交際化されています。デフォルトの言語は英語ですが、使用するロケールや、デフォルトのロケールを選択できます。

手順

- メニューで **Realm Settings** をクリックします。
- Localization** タブをクリックします。

3. **Internationalization** を有効にします。
4. サポートする言語を選択します。

Localization タブ

Master Enabled Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

< General Login Email Themes Keys Events Localization >

Internationalization ?

Enabled

Supported locales

English × Français × Italiano × Select locales × ▾

Default locale

English ▾

ユーザーが次回ログインすると、そのユーザーはログインページの言語を選択して、ログイン画面、Account Console、および Admin Console に使用できます。

関連情報

- [サーバー開発者ガイド](#) では、追加の言語を提供する方法を説明しています。テーマによって提供されるすべての国際化されたテキストは、**ローカライゼーション** タブでレルム固有のテキストによって上書きできます。

3.7.1. ユーザーロケールの選択

ロケールセレクタープロバイダーは、利用可能な情報に関する最適なロケールを提案します。ただし、多くの場合は、ユーザーに不明です。このため、以前に認証されたユーザーのロケールは永続化されたクッキーに記憶されます。

ロケールを選択するには、以下の最初の項目を使用します。

- User selected: ドロップダウンロケールセレクターを使用してロケールを選択するとき
- User profile: 認証されたユーザーがあり、ユーザーにロケールセットが推奨されるとき
- Client selected: `ui_locales` パラメーターなどを使用してクライアントにより渡される
- Cookie: ブラウザーで選択した最後のロケール
- 許可される言語: `Accept-Language` ヘッダーからのロケール

- レルムのデフォルト
- 上記のいずれでなければ、英語に戻ります。

ユーザーが認証されたとき、アクションがトリガーされ、前述の永続化されたクッキーでロケールを更新します。ユーザーがログインページのロケールセクターでロケールをアクティブに切り替える場合は、この時点でユーザーロケールも更新されます。

ロケールを選択するロジックを変更する場合は、**LocaleSelectorProvider** を作成するオプションがあります。詳細は、[サーバー開発者ガイド](#) を参照してください。

3.8. ログインオプションの制御

Red Hat build of Keycloak には、いくつかのビルトインログインページ機能があります。

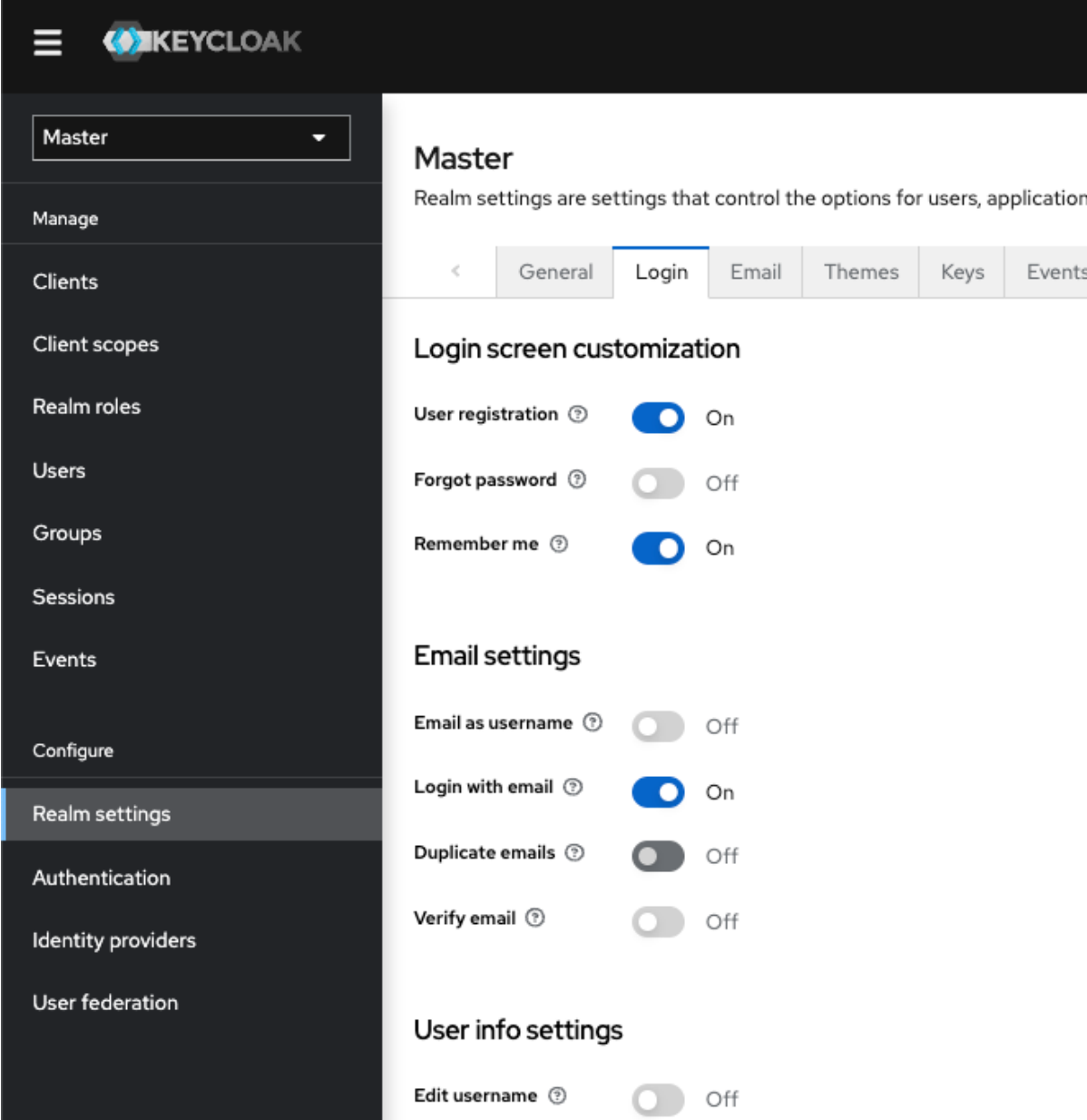
3.8.1. forgot password 有効化

Forgot password を有効にすると、パスワードを取得するか、OTP ジェネレーターを失う場合に、ログイン認証情報をリセットできます。

手順

1. メニューで **Realm Settings** をクリックします。
2. **Login** タブをクリックします。

ログインタブ



The screenshot shows the Keycloak Admin Console interface. On the left is a dark sidebar with a menu. The 'Master' realm is selected at the top. The 'Login' tab is active in the main content area. Under 'Login screen customization', the 'Forgot password' toggle is currently off. The 'Email as username' toggle is also off. The 'Login with email' toggle is on. The 'Duplicate emails' and 'Verify email' toggles are off. Under 'User info settings', the 'Edit username' toggle is off.

Master

Realm settings are settings that control the options for users, application

General Login Email Themes Keys Events

Login screen customization

User registration On

Forgot password Off

Remember me On

Email settings

Email as username Off

Login with email On

Duplicate emails Off

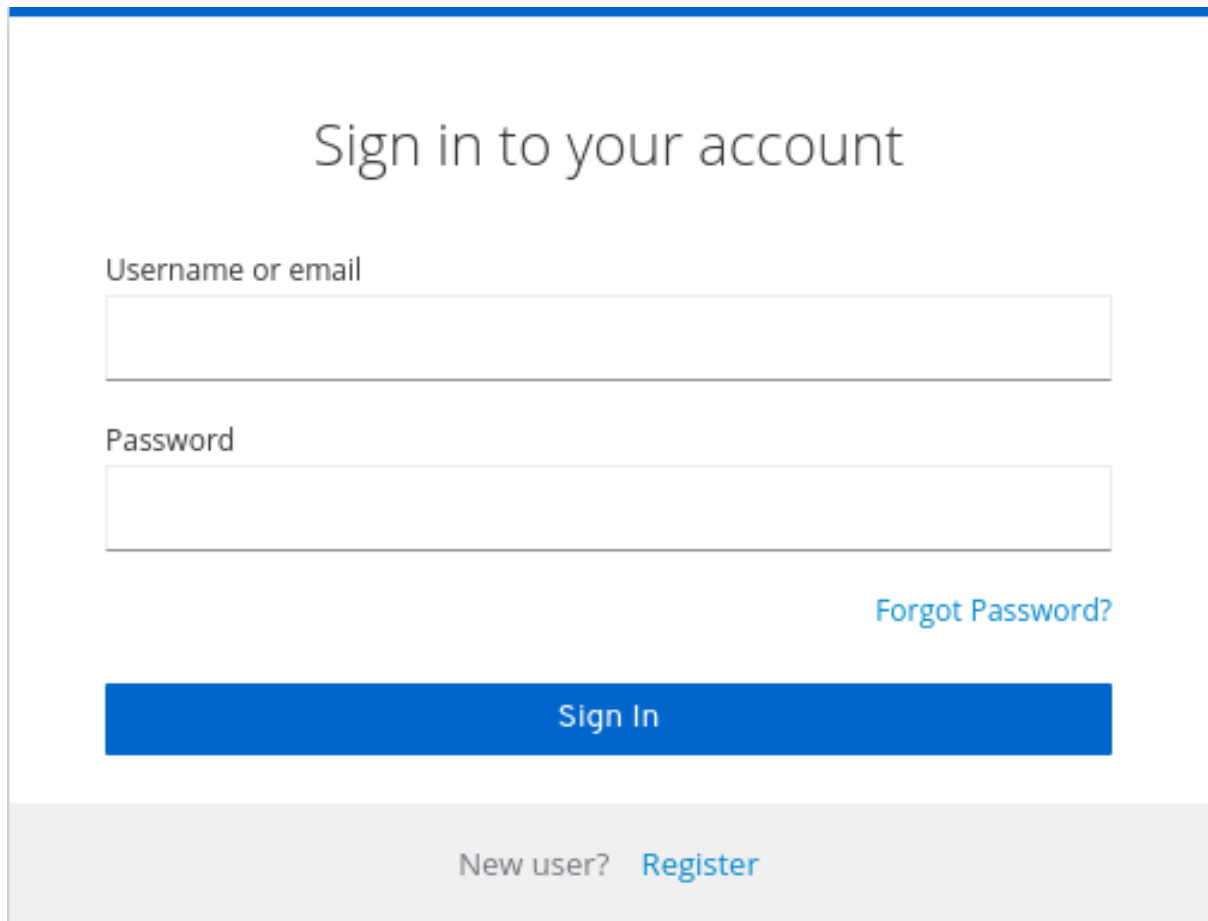
Verify email Off

User info settings

Edit username Off

3. **Forgot password** を **ON** に切り替えます。
Forgot Password? リンクがログインページに表示されます。

forgot password リンク



Sign in to your account

Username or email

Password

[Forgot Password?](#)

Sign In

New user? [Register](#)

4. Keycloak がリセット電子メールを送信できるようにするには、**Email** タブで **Host** と **From** を指定します。
5. このリンクをクリックして、ユーザー名またはメールアドレスを入力し、リンクのある電子メールを受信して認証情報をリセットできるユーザーを追加します。

Forgot password ページ

Forgot Your Password?

Username or email

[« Back to Login](#)

Submit

Enter your username or email address and we will send you instructions on how to create a new password.

メールで送信されるテキストは設定可能です。詳細は、[サーバー開発者ガイド](#)を参照してください。

ユーザーがメールリンクをクリックすると、Red Hat build of Keycloak によりパスワードを更新するように求められます。また、OTP ジェネレーターを設定している場合、Red Hat build of Keycloak により OTP ジェネレーターを再設定するように求められます。組織のセキュリティ要件によっては、ユーザーが電子メールで OTP ジェネレーターをリセットしたくないことがあります。

この動作を変更するには、以下の手順を実施します。

手順

1. メニューで **Authentication** をクリックします。
2. **Flows** タブをクリックします。
3. **Reset Credentials** フローを選択します。

認証情報フローをリセット

Authentication > Flow details







Reset credentials

Default

Built-in

Action ▼

  [Add step](#) [Add sub-flow](#)

Steps	Requirement
 <input type="text" value="Choose User"/>	Required
 <input type="text" value="Send Reset Email"/>	Required
 <input type="text" value="Reset Password"/>	Required ▼
 ▼ Reset - Conditional OTP Flow to determine if the OTP should be reset or not. Set to REQUIRED to force.	Conditional ▼
 <input type="text" value="Condition - user configured"/>	Required ▼
 <input type="text" value="Reset OTP"/>	Required ▼

OTP をリセットしない場合は、**Reset OTP** の要件を **Disabled** に設定します。

4. メニューで **Authentication** をクリックします。
5. **Required Actions** タブをクリックします。
6. **Update Password** が有効になっていることを確認します。

必要なアクション

Authentication

Authentication is the area where you can configure and manage different credential types. [Learn more](#)

Flows	Required actions	Policies
	Required actions	Enabled Set as default action
☰	Configure OTP	<input checked="" type="checkbox"/> On <input type="checkbox"/> Off
☰	Terms and Conditions	<input type="checkbox"/> Off <input type="checkbox"/> Disabled off
☰	Update Password	<input checked="" type="checkbox"/> On <input type="checkbox"/> Off
☰	Update Profile	<input checked="" type="checkbox"/> On <input type="checkbox"/> Off
☰	Verify Email	<input checked="" type="checkbox"/> On <input type="checkbox"/> Off
☰	Delete Account	<input type="checkbox"/> Off <input type="checkbox"/> Disabled off
☰	Update User Locale	<input checked="" type="checkbox"/> On <input type="checkbox"/> Off
☰	Webauthn Register Passwordless	<input checked="" type="checkbox"/> On <input checked="" type="checkbox"/> On
☰	Webauthn Register	<input checked="" type="checkbox"/> On <input checked="" type="checkbox"/> On
☰	Verify Profile	<input type="checkbox"/> Off <input type="checkbox"/> Disabled off

3.8.2. Remember Me の有効化

ブラウザを閉じたログインユーザーはセッションを破棄し、そのユーザーは再度ログインする必要があります。ユーザーがログイン時に **Remember Me** チェックボックスをクリックした場合に、そのユーザーのログインセッションを開いたままにするように Red Hat build of Keycloak を設定できます。このアクションは、ログインクッキーをセッションのみのクッキーから永続クッキーに変換します。

手順

1. メニューで **Realm Settings** をクリックします。
2. **Login** タブをクリックします。
3. **Remember Me** を **On** に切り替えます。

ログインタブ

Master

Enabled

Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

< General Login Email Themes Keys Events Localization Security defens >

Login screen customization

User registration  On

Forgot password  Off

Remember me  On

Email settings

Email as username  Off

Login with email  On

Duplicate emails  Off

Verify email  Off

User info settings

Edit username  Off

この設定を保存すると、レルムのログインページに **remember me** チェックボックスが表示されます。

Remember Me

Sign in to your account

Username or email

Password

Remember me

Sign In

New user? [Register](#)

3.8.3. ACR から認証レベル (LoA) へのマッピング

レルムのログイン設定では、どの **Authentication Context Class Reference (ACR)** 値をどの **Level of Authentication (LoA)** マップするかを定義できます。ACR には任意の値を指定できますが、LoA は数値でなければなりません。acr 要求は、OIDC 要求で送信される **claim** または **acr_values** パラメーターで要求でき、アクセストークンおよび ID トークンにも含めることができます。マッピングされた番号は、認証フロー条件で使用されます。

特定のクライアントがレルムとは異なる値を使用する必要がある場合は、マッピングをクライアントレベルで指定することもできます。ただし、レルムマッピングを行うのがベストプラクティスです。

ACR to LoA Mapping ⓘ

silver	1	-
gold	2	-
ACR	LOA	+

詳細は、[Step-up Authentication](#) と [公式の OIDC 仕様](#) を参照してください。

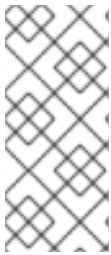
3.8.4. 電子メールワークフローの更新 (UpdateEmail)

このワークフローでは、ユーザーは UPDATE_EMAIL アクションを使用して独自のメールアドレスを変更する必要があります。

アクションは、単一のメール入力フォームに関連付けられます。レルムの電子メール検証が無効な場合は、この動作により、検証なしに電子メールを更新できます。レルムで電子メールの検証が有効になっている場合は、アカウントメールを変更せずにアクションが新しいメールアドレスにメール更新アク

シヨントークンを送信します。トリガーされるアクショントークンのみがメールの更新を完了します。

アプリケーションは UPDATE_EMAIL を AIA (Application Initiated Action) として利用することで、ユーザーをメール更新フォームに送信できます。



注記

UpdateEmail は **テクノロジープレビュー** であるため、完全にサポートされていません。デフォルトでは無効になっています。

有効にするには、**--features=preview** または **--features=update-email** を使用してサーバーを起動します。



注記

この機能を有効にして、以前のバージョンから移行する場合は、レルムで **電子メールの更新** に必要なアクションを有効にします。そうでない場合は、メールアドレスを更新できません。

3.9. レルムキーの設定

Keycloak で使用される認証プロトコルには、暗号化署名が必要であり、場合によっては暗号化も必要です。Red Hat build of Keycloak は、これを実現するために非対称のキーペア (秘密鍵と公開鍵) を使用します。

Red Hat build of Keycloak では、同時に持てるアクティブキーペアは1つですが、複数のパッシブキーを持つこともできます。アクティブキーペアは新しい署名の作成に使用され、パッシブキーペアは以前の署名の検証に使用できます。これにより、ダウンタイムやユーザーの中断なしにキーを定期的にローテーションできます。

レルムが作成されると、キーペアと自己署名付き証明書が自動的に生成されます。

手順

1. メニューで **Realm Settings** をクリックします。
2. **Keys** をクリックします。
3. **パッシブキー** を表示するには、フィルタードロップダウンからパッシブキーを選択します。
4. フィルタードロップダウンから **Disabled keys** を選択して、無効なキーを表示します。

キーペアのステータスは **Active** になりますが、レルムの現在アクティブなキーペアとしては選択されません。署名に使用されるアクティブなペアは、優先順位に基づきソートされたアクティブキーペアを提供できるキープロバイダーのうち、最初のキープロバイダーが選択されます。

3.9.1. 鍵のローテーション

鍵を定期的にローテーションすることが推奨されます。既存のアクティブなキーよりも優先度の高い新しいキーを作成することから始めます。代わりに、同じ優先度で新しいキーを作成し、以前のキーをパッシブにすることができます。

新しいキーが使用可能になると、すべての新しいトークンと cookie はその新しいキーで署名されます。ユーザーがアプリケーションに対して認証されると、SSO Cookie が新しい署名で更新されます。OpenID Connect トークンを更新すると、新しいキーで新たなトークンが署名されます。最終的には、

すべての Cookie とトークンは新しいキーを使用し、しばらくすると、古いキーを削除できます。

古いキーを削除する頻度は、セキュリティ間のトレードオフであり、すべてのクッキーとトークンが更新されるようにすることです。新しいキーの作成後に、3 か月から 6 か月までのすべてのキーを作成し、古いキーを 2 か月に削除することを検討してください。新しいキーが追加され、古いキーが削除されるまでの期間にユーザーが非アクティブである場合、そのユーザーは再認証する必要があります。

鍵をローテーションすると、オフライントークンにも適用されます。これらのアプリケーションが古いキーが削除される前にトークンを更新する必要があることを確認するには、アプリケーションを更新します。

3.9.2. 生成されたキーペアの追加

この手順を使用して、自己署名付き証明書を含むキーペアを生成できます。

手順

1. 管理コンソールで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. **Keys** タブをクリックします。
4. **Providers** タブをクリックします。
5. **Add provider** をクリックし、**rsa-generated** を選択します。
6. **Priority** フィールドに番号を入力します。この数字は、新しいキーペアがアクティブなキーペアになるかどうかを決定します。最も大きい番号のキーペアがアクティブになります。
7. **AES Key size** の値を選択します。
8. **Save** をクリックします。

プロバイダーの優先度を変更すると、キーが再生成されますが、キーサイズを変更する場合はプロバイダーを編集し、新しいキーが生成されます。

3.9.3. 証明書の抽出によるキーのローテーション

RSA で生成されたキーペアから証明書を抽出し、その証明書を新しいキーストアで使用することにより、キーをローテーションできます。

前提条件

- 生成されたキーペア

手順

1. 管理コンソールでレルムを選択します。
2. **Realm Settings** をクリックします。
3. **Keys** タブをクリックします。
Active キーのリストが表示されます。
4. RSA キーのある行で、**Public Keys** の下の **Certificate** をクリックします。

証明書はテキスト形式で表示されます。

5. 証明書をファイルに保存し、これらの行で囲みます。

```
----Begin Certificate----
<Output>
----End Certificate----
```

6. `keytool` コマンドを使用して、キーファイルを PEM 形式に変換します。
7. キーストアから現在の RSA 公開鍵証明書を削除します。

```
keytool -delete -keystore <keystore>.jks -storepass <password> -alias <key>
```

8. 新しい証明書をキーストアにインポートします。

```
keytool -importcert -file domain.crt -keystore <keystore>.jks -storepass <password> -alias
<key>
```

9. アプリケーションをリビルドします。

```
mvn clean install wildfly:deploy
```

3.9.4. 既存のキーペアと証明書の追加

別のユーザーが取得したキーペアと証明書を追加するには、**Providers** を選択し、ドロップダウンから **rsa** を選択します。新たなキーペアがアクティブなキーペアになるように、優先度を変更できます。

前提条件

- プライベートキーファイル。ファイルは PEM 形式である必要があります。

手順

1. 管理コンソールでレルムを選択します。
2. **Realm settings** をクリックします。
3. **Keys** タブをクリックします。
4. **Providers** タブをクリックします。
5. **Add provider** をクリックし、**rsa** を選択します。
6. **Priority** フィールドに番号を入力します。この数字は、新しいキーペアがアクティブなキーペアになるかどうかを決定します。
7. **Private RSA Key** の横にある **Browse...** をクリックして、秘密キーファイルをアップロードします。
8. 秘密キーの署名付き証明書がある場合は、**X509 Certificate** の横にある **Browse...** をクリックして証明書ファイルをアップロードします。証明書をアップロードしない場合、Red Hat build of Keycloak は自己署名付き証明書を自動的に生成します。

9. **Save** をクリックします。

3.9.5. Java キーストアからキーを読み込む

ホストの Java キーストアファイルに保存されているキーペアと証明書を追加するには、**Provider** を選択し、ドロップダウンから **java-keystore** を選択します。新たなキーペアがアクティブなキーペアになるように、優先度を変更できます。

関連する証明書チェーンをロードするには、キーペアのロードに使用したのと同じ **Key Alias** を使用して Java キーストアファイルにインポートする必要があります。

手順

1. 管理コンソールで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. **Keys** タブをクリックします。
4. **Providers** タブをクリックします。
5. **Add provider** をクリックし、**java-keystore** を選択します。
6. **Priority** フィールドに番号を入力します。この数字は、新しいキーペアがアクティブなキーペアになるかどうかを決定します。
7. **キーストア** の値を入力します。
8. **キーストアパスワード** の値を入力します。
9. **Key Alias** の値を入力します。
10. **Key Password** の値を入力します。
11. **Save** をクリックします。

3.9.6. 鍵のパッシブの作成

手順

1. 管理コンソールで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. **Keys** タブをクリックします。
4. **Providers** タブをクリックします。
5. パッシブに設定するキーのプロバイダーをクリックします。
6. **Active** を **Off** に切り替えます。
7. **Save** をクリックします。

3.9.7. キーの無効化

手順

1. 管理コンソールで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. **Keys** タブをクリックします。
4. **Providers** タブをクリックします。
5. パッシブに設定するキーのプロバイダーをクリックします。
6. **Enabled** を **Off** に切り替えます。
7. **Save** をクリックします。

3.9.8. 侵害された鍵

Red Hat build of Keycloak は署名鍵をローカルにのみ保存し、それをクライアントアプリケーション、ユーザー、または他のエンティティと共有することはありません。ただし、レルム署名鍵が侵害されたと思われる場合は、上記のとおり最初に新しいキーペアを生成し、侵害されたキーペアを即座に削除する必要があります。

または、プロバイダーを **Providers** テーブルから削除できます。

手順

1. メニューで **Clients** をクリックします。
2. **security-admin-console** をクリックします。
3. **Access settings** セクションまで下にスクロールします。
4. **Admin URL** フィールドに入力します。
5. **Advanced** タブをクリックします。
6. **Revocation** セクションで **Set to now** をクリックします。
7. **Push** をクリックします。

not-before ポリシーをプッシュすると、クライアントアプリケーションは、セキュリティー侵害を受けたキーで署名された既存のトークンを受け入れないようにします。クライアントアプリケーションは Red Hat build of Keycloak から新しいキーペアをダウンロードするように強制されるため、侵害されたキーで署名されたトークンは無効になります。



注記

REST および機密クライアントは、プッシュされた not-before ポリシーリクエストを Red Hat build of Keycloak がクライアントに送信できるように、**Admin URL** を設定する必要があります。

第4章 外部ストレージの使用

組織には、情報、パスワード、およびその他の認証情報が含まれるデータベースを設定できます。通常、既存のデータストレージを Red Hat build of Keycloak デプロイメントには移行できないため、Red Hat build of Keycloak は既存の外部ユーザーデータベースをフェデレーションできます。Red Hat build of Keycloak は LDAP と Active Directory をサポートしていますが、Red Hat build of Keycloak の User Storage SPI を使用してカスタムユーザーデータベースのエクステンションをコーディングすることもできます。

ユーザーがログインしようとする時、Red Hat build of Keycloak はそのユーザーのストレージを調べてそのユーザーを見つけます。ユーザーが見つからない場合、一致するまで Red Hat build of Keycloak はレルムの各ユーザーストレージプロバイダーに対して処理を繰り返します。外部データストレージからのデータは、Red Hat build of Keycloak ランタイムが消費する標準ユーザーモデルにマッピングされます。次に、このユーザーモデルは OIDC トークンクレームと SAML アサーション属性にマッピングします。

ほとんどの外部ユーザーデータベースには、Red Hat build of Keycloak のすべての機能をサポートするために必要なデータが存在しないため、ユーザーストレージプロバイダーはアイテムを Red Hat build of Keycloak ユーザーデータストレージにローカルに保存することを選択できます。プロバイダーは、ユーザーをローカルでインポートして、外部データストレージと定期的に同期できます。この方法は、プロバイダーの機能とプロバイダーの設定によって異なります。たとえば、外部ユーザーデータのストレージは OTP に対応していない可能性があります。プロバイダーによっては、OPTION を Red Hat build of Keycloak で処理および保存できます。

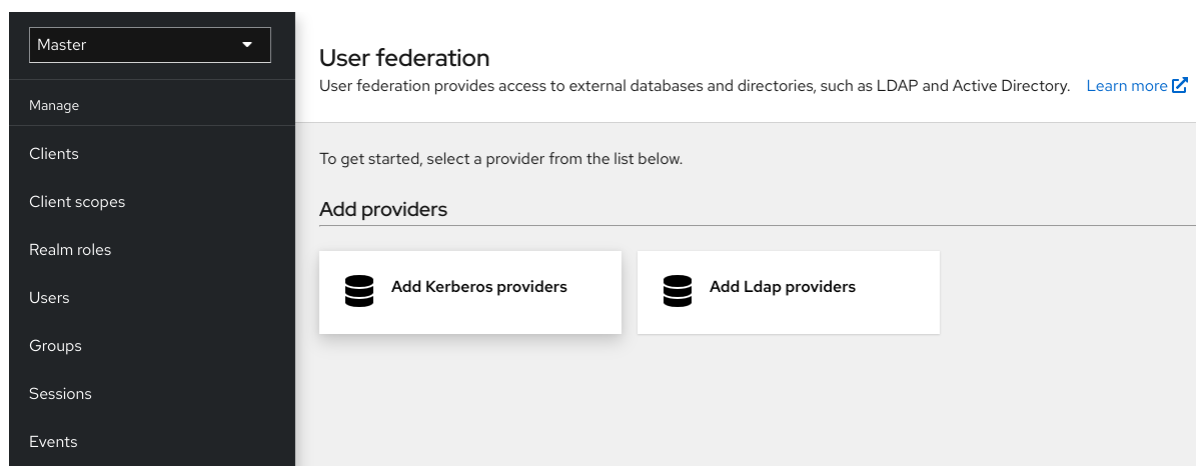
4.1. プロバイダーの追加

ストレージプロバイダーを追加するには、以下の手順を実行します。

手順

1. メニューの **User Federation** をクリックします。

ユーザーフェデレーション



2. リストされたカードからプロバイダータイプのカードを選択します。
Red Hat build of Keycloak により、そのプロバイダーの設定ページが表示されます。

4.2. プロバイダーの失敗の処理

User Storage Provider が失敗すると、ログインできず、管理コンソールでユーザーを表示できます。ストレージプロバイダーを使用してユーザーを検索する場合、Red Hat build of Keycloak は失敗を検出し

ないため、呼び出しをキャンセルします。ユーザーのルックアップ時に失敗する優先度が高いストレージプロバイダーの場合、ログインまたはユーザークエリーは例外を出して失敗し、次の設定されたプロバイダーにはフェイルオーバーしません。

Red Hat build of Keycloak は、LDAP またはカスタムユーザーストレージプロバイダーの前に、まずローカルの Red Hat build of Keycloak ユーザーデータベースを検索してユーザーを解決します。LDAP およびバックエンドへの接続に問題が発生した場合に備えて、ローカルの Red Hat build of Keycloak ユーザーデータベースに保存される管理者アカウントを作成することを検討してください。

各 LDAP およびカスタム User Storage Provider には、管理コンソールページで **enable** することができます。User Storage Provider を無効にすると、クエリーの実行時にプロバイダーがスキップされるので、優先度の低い別のプロバイダーでユーザーアカウントを表示し、ログインすることができます。プロバイダーが **import** ストラテジーを使用し、無効にされている場合、インポートユーザーは読み取り専用モードでも検索できます。

ストレージプロバイダーのルックアップが失敗した場合、ユーザーデータベース間に重複したユーザー名または重複したメールが存在することが多いため、Red Hat build of Keycloak はフェイルオーバーしません。ユーザー名とメールアドレスの重複により、管理者が別のデータストアからロードされることを想定した場合にユーザーが外部データストアからロードされるため、問題が発生する可能性があります。

4.3. LDAP (LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL) および ACTIVE DIRECTORY

Red Hat build of Keycloak には LDAP/AD プロバイダーが含まれています。1つの Red Hat build of Keycloak レルム内で複数の異なる LDAP サーバーをフェデレーションし、LDAP ユーザー属性を Red Hat build of Keycloak の共通ユーザーモデルにマッピングできます。

デフォルトでは、Red Hat build of Keycloak はユーザーアカウントのユーザー名、メール、名、姓をマッピングしますが、追加の [マッピング](#) を設定することもできます。Red Hat build of Keycloak の LDAP/AD プロバイダーは、LDAP/AD プロトコルとストレージ、編集、同期モードを使用したパスワード検証をサポートしています。

4.3.1. フェデレーションされた LDAP ストレージの設定

手順

1. メニューの **User Federation** をクリックします。

ユーザーフェデレーション

The screenshot shows the Keycloak administration console interface. On the left is a dark sidebar with a navigation menu. The 'User Federation' option is highlighted. The main content area has a white background and is titled 'User federation'. Below the title is a subtitle: 'User federation provides access to external databases and directories, such as LDAP and Active Directory.' followed by a 'Learn more' link. Below this is a light gray box with the text 'To get started, select a provider from the list below.' Underneath is a section titled 'Add providers' which contains two white buttons with database icons: 'Add Kerberos providers' and 'Add Ldap providers'.

2. Add LDAP providers をクリックします。

Red Hat build of Keycloak により、LDAP 設定ページが表示されます。

4.3.2. ストレージモード

Red Hat build of Keycloak は、LDAP から ローカルの Red Hat build of Keycloak ユーザーデータベースにユーザーをインポートします。このユーザーデータベースのコピーは、オンデマンドまたは定期的なバックグラウンドタスクを介して同期します。パスワード同期の例外が存在します。Red Hat build of Keycloak がパスワードをインポートすることはありません。パスワードの検証は LDAP サーバーで常に行われます。

同期の利点は、追加に必要なユーザーごとのデータがローカルに保存されるため、すべての Red Hat build of Keycloak 機能が効率的に動作することです。欠点は、Red Hat build of Keycloak が特定のユーザーに初めてクエリーを実行するたびに、Red Hat build of Keycloak が対応するデータベースの挿入を実行することです。

インポートを LDAP サーバーと同期できます。LDAP マッパーがデータベースではなく LDAP から特定の属性を常に読み取る場合、インポート同期は必要ありません。

Red Hat build of Keycloak のユーザーデータベースにユーザーをインポートしなくても、Red Hat build of Keycloak で LDAP を使用できます。LDAP サーバーは、Red Hat build of Keycloak ランタイムが使用する共通ユーザーモデルをバックアップします。Red Hat build of Keycloak の機能に必要なデータを LDAP がサポートしていない場合、その機能は動作しません。このアプローチの利点は、LDAP ユーザーのコピーを Red Hat build of Keycloak のユーザーデータベースにインポートして同期するためのリソースを使用しないことです。

LDAP 設定ページの **Import Users** スイッチは、このストレージモードを制御します。ユーザーをインポートするには、この切り替えを **ON** に切り替えます。



注記

Import Users を無効にすると、Red Hat build of Keycloak データベースにユーザープロフィール属性を保存できなくなります。また、LDAP にマッピングされたユーザープロフィールメタデータ以外のメタデータを保存することはできません。このメタデータには、LDAP マッパーの設定に基づくロールマッピング、グループマッピング、およびその他のメタデータを含めることができます。

LDAP 以外のマッピングユーザーデータを変更しようとする、ユーザーの更新ができません。たとえば、ユーザーの **enabled** フラグが LDAP 属性にマッピングされていない限り、LDAP マッピングされたユーザーを無効にすることはできません。

4.3.3. モードの編集

ユーザーと管理者はユーザーメタデータを変更できます。ユーザーは [Account Console](#) から、管理者は管理コンソールから変更できます。LDAP 設定ページの **Edit Mode** 設定により、ユーザーの LDAP 更新権限が定義されます。

READONLY

ユーザー名、メール、名、姓、他のマップされた属性を変更することはできません。Red Hat build of Keycloak では、ユーザーがこれらのフィールドを更新しようとするたびにエラーが表示されます。パスワードの更新はサポートされていません。

WRITABLE

ユーザー名、メール、名、姓、他のマップされた属性を変更し、LDAP ストアと自動的に同期することはできません。

UNSYNCED

Red Hat build of Keycloak はユーザー名、メール、名、姓、パスワードの変更を Red Hat build of Keycloak のローカルストレージに保存するため、管理者はこのデータを LDAP に同期する必要があります。このモードでは、Red Hat build of Keycloak デプロイメントは読み取り専用 LDAP サーバー上のユーザーメタデータを更新できます。このオプションは、LDAP からローカルの Red Hat build of Keycloak ユーザーデータベースにユーザーをインポートするときにも適用されます。



注記

Red Hat build of Keycloak は、LDAP プロバイダーを作成する際に最初の [LDAP マッパー](#) セットも作成します。Red Hat build of Keycloak は、**Vendor**、**Edit Mode**、**Import Users** スイッチの組み合わせに基き、これらのマッパーを設定します。たとえば、編集モードが UNSYNCED の場合、Red Hat build of Keycloak は LDAP サーバーからではなくデータベースから特定のユーザー属性を読み取るようにマッパーを設定します。ただし、後で編集モードを変更すると、設定が UNSYNCED モードで変更されたかどうかを検出できないため、マッパーの設定は変更できません。LDAP プロバイダーの作成時に **Edit Mode** を決定します。この注記は、**ユーザーのインポート** スイッチにも適用されません。

4.3.4. その他の設定オプション

コンソール表示名

管理コンソールで表示するプロバイダーの名前。

優先度

ユーザーを検索したり、ユーザーを追加したりする際のプロバイダーの優先順位です。

登録の同期

Red Hat build of Keycloak によって作成された新しいユーザーを LDAP に追加する場合は、このスイッチを **ON** に切り替えます。

Kerberos 認証を許可

LDAP からプロビジョニングされたユーザーデータを使用して、レルムで Kerberos/SPNEGO 認証を有効にします。詳細については、[Kerberos セクション](#) を参照してください。

その他のオプション

マウスポインターを管理コンソールのツールチップの上に置き、これらのオプションの詳細を表示します。

4.3.5. SSL 経由での LDAP への接続

LDAP ストアへの安全な接続 URL (たとえば、**ldaps://myhost.com:636**) を設定すると、Red Hat build of Keycloak は SSL を使用して LDAP サーバーと通信します。Red Hat build of Keycloak が LDAP への SSL 接続を信頼できるように、Red Hat build of Keycloak サーバー側でトラストストアを設定します。

Truststore SPI を使用して、Red Hat build of Keycloak のグローバルトラストストアを設定します。グローバルトラストストアの設定について、詳細は [トラストストアの設定](#) の章を参照してください。Truststore SPI を設定しないと、トラストストアは Java によって提供されるデフォルトのメカニズムにフォールバックします。これは、システムプロパティーが設定されていないと、**javax.net.ssl**. ONLY システムプロパティーまたは JDK からの cacerts ファイルによって提供されるファイルになります。

LDAP フェデレーションプロバイダー設定の **Use Truststore SPI** 設定プロパティーは、トラストストア SPI を制御します。デフォルトでは、Red Hat build of Keycloak はプロパティーを **Always** に設定します。これは、ほとんどのデプロイメントに適切な設定です。Red Hat build of Keycloak は、LDAP への接続 URL が **ldaps** のみで始まる場合、Truststore SPI を使用します。

4.3.6. LDAP ユーザーを Red Hat build of Keycloak に同期する

Import Users オプションを設定すると、LDAP プロバイダーが Red Hat build of Keycloak ローカルデータベースへの LDAP ユーザーのインポートを処理します。ユーザーが初めてログインするとき、またはユーザークエリーの一部として返されるとき (例: 管理コンソールの検索フィールドを使用する場合)、LDAP プロバイダーは LDAP ユーザーを Red Hat build of Keycloak データベースにインポートします。認証中に、LDAP パスワードが検証されます。

すべての LDAP ユーザーを Red Hat build of Keycloak データベースに同期する場合は、LDAP プロバイダー設定ページで **Sync Settings** を設定して有効にします。

2 種類の同期が存在します。

定期的な完全同期 (Periodic Full)

このタイプでは、すべての LDAP ユーザーが Red Hat build of Keycloak データベースに同期されます。LDAP ユーザーはすでに Red Hat build of Keycloak に存在しますが、LDAP とは異なり、Red Hat build of Keycloak データベースで直接更新されます。

定期的な変更したユーザー同期 (Periodic Changed)

同期する場合、Red Hat build of Keycloak は、最後の同期後に作成または更新されたユーザーのみを作成または更新します。

同期する最適な方法として、LDAP プロバイダーの初回作成時にすべての**ユーザーの同期**をクリックし、変更したユーザーの定期的な同期を設定するのが最適です。

4.3.7. LDAP マッパー

LDAP マッパーは LDAP プロバイダーによってトリガーされる **listeners** です。別の拡張は LDAP 統合を指定します。LDAP マッパーは、以下の場合にトリガーされます。

- ユーザーは LDAP を使用してログインします。
- ユーザーは最初に登録されます。
- 管理コンソールはユーザーにクエリーを実行します。

LDAP フェデレーションプロバイダーを作成すると、Red Hat build of Keycloak がこのプロバイダーの **mappers** セットを自動的に提供します。これは、ユーザーはマッパーの開発や、既存マッパーの更新/削除を行えます。

ユーザー属性マッパー

このマッパーは、どの LDAP 属性が Red Hat build of Keycloak ユーザーの属性にマッピングされるかを指定します。たとえば、**mail** LDAP 属性を Red Hat build of Keycloak データベースの **email** 属性に設定できます。このマッパーの実装では、1対1のマッピングが常に存在します。

fullName マッパー

このマッパーはユーザーのフルネームを指定します。Red Hat build of Keycloak は、名前を LDAP 属性 (通常は **cn**) に保存し、その名前を Red Hat build of Keycloak データベースの **firstName** および **lastName** 属性にマッピングします。ユーザーのフルネームを含む **cn** は LDAP デプロイメントで共通です。



注記

Red Hat build of Keycloak で新規ユーザーを登録し、**Sync Registrations** が LDAP プロバイダー用に ON になっている場合、fullName マッパーはユーザー名へのフォールバックを許可します。このフォールバックは、Microsoft Active Directory (MSAD) を使用する際に役に立ちます。MSAD の一般的なセットアップでは、**cn** LDAP 属性を fullName として設定し、同時に LDAP プロバイダー設定の **RDN LDAP Attribute** として **cn** LDAP 属性を使用します。この設定では、Red Hat build of Keycloak はユーザー名にフォールバックします。たとえば、"john123" という Red Hat build of Keycloak ユーザーを作成し、firstName と lastName を空のままにすると、フルネームマッパーは LDAP の **cn** 値として "john123" を保存します。firstName および lastName に "John Doe" を入力すると、fullName マッパーは LDAP **cn** を "John Doe" の値に更新し、ユーザー名へのフォールバックが必要ありません。

ハードコードされた属性マッパー

このマッパーは、LDAP にリンクされた各 Red Hat build of Keycloak ユーザーにハードコーディングされた属性値を追加します。また、このマッパーは **enabled** または **emailVerified** ユーザー プロパティの値を強制的に実行することもできます。

ロールマッパー

このマッパーは、LDAP から Red Hat build of Keycloak ロールマッピングにロールマッピングを設定します。単一のロールマッパーは LDAP ロール (通常は LDAP ツリーの特定ブランチからのグループ) を、指定されたクライアントのレルムロールまたはクライアントロールに対応するロールにマップできます。同じ LDAP プロバイダーに追加のロールマッパーを設定できます。たとえば、**ou=main,dc=example,dc=org** 下のグループからのロールマッピングをレルムロールマッピングにマッピングし、**ou=finance,dc=example,dc=org** 下のグループから、クライアント **finance** のクライアントロールマッピングへマップするように指定できます。

ハードコードされたロールマッパー

このマッパーは、LDAP プロバイダーから各 Red Hat build of Keycloak ユーザーに指定された Red Hat build of Keycloak ロールを付与します。

グループマッパー

このマッパーは、LDAP ツリーのブランチから、Red Hat build of Keycloak 内のグループに LDAP グループをマッピングします。また、LDAP からのユーザーグループのマッピングを Red Hat build of Keycloak のユーザーグループのマッピングに伝播します。

MSAD ユーザーアカウントマッパー

このマッパーは、Microsoft Active Directory (MSAD) に固有のもので、MSAD ユーザーアカウントの状態を、有効なアカウントや期限切れのパスワードなどの Red Hat build of Keycloak アカウントの状態に統合できます。このマッパーは **userAccountControl** および **pwdLastSet** LDAP 属性を使用します。これは MSAD に固有で、LDAP 標準ではありません。たとえば、**pwdLastSet** の値が **0** の場合 Red Hat build of Keycloak ユーザーはパスワードを更新する必要があります。結果として、UPDATE_PASSWORD の必要なアクションがユーザーに追加されます。**userAccountControl** の値が **514** (無効なアカウント) の場合、Red Hat build of Keycloak は無効になります。

証明書マッパー

このマッパーは X.509 証明書をマッピングします。Red Hat build of Keycloak はこれを、X.509 認証および **Full certificate in PEM format** と組み合わせてアイデンティティソースとして使用します。このマッパーは **User Attribute Mapper** と同様に機能しますが、Red Hat build of Keycloak は PEM または DER 形式の証明書を保存する LDAP 属性をフィルタリングできます。このマッパーを使用して、**Always Read Value From LDAP** を有効にします。

ユーザー名、名、姓、メールなどの基本的な Red Hat build of Keycloak ユーザー属性を、対応する LDAP 属性にマッピングするユーザー属性マッパー。これらを拡張し、独自の追加属性マッピングを提供できます。管理コンソールは、対応するマッパーの設定に役立つツールチップを提供します。

4.3.8. パスワードのハッシュ

Red Hat build of Keycloak は、パスワードを更新するとそのパスワードをプレーンテキスト形式で送信します。このアクションは、Red Hat build of Keycloak がデータベースに送信する前にパスワードをハッシュ化してソルト化する、Red Hat build of Keycloak のビルトインデータベースにおけるパスワード更新とは異なります。LDAP の場合、Red Hat build of Keycloak は LDAP サーバーに依存してパスワードのハッシュ化とソルト化を行います。

デフォルトでは、MSAD、RHDS、FreelIPA ハッシュ、ソルトパスワードなどの LDAP サーバー。RFC3062 で説明されているように **LDAPv3 Password Modify Extended Operation** を使用しない限り、OpenLDAP や ApacheDS などのその他の LDAP サーバーは、パスワードをプレーンテキストに保存します。LDAP 設定ページで LDAPv3 Password Modify Extended Operation を有効にします。詳細は、お使いの LDAP サーバーのドキュメントを参照してください。



警告

ldapsearch を使用して変更したディレクトリーエントリーを検査することで、ユーザーパスワードが正しくハッシュ化され、プレーンテキストとして保存されていることと、base64 が **userPassword** 属性値をデコードしていることを常に確認します。

4.3.9. トラブルシューティング

カテゴリ **org.keycloak.storage.ldap** では、ログレベルを TRACE に増やすと便利です。この設定により、多数のログインメッセージは **TRACE** レベルの server.log ファイルに送信されます。これには、すべてのクエリーのログが LDAP サーバーとクエリーの送信に使用されたパラメーターが含まれます。ユーザーフォーラムまたは JIRA で LDAP の質問を作成する場合は、TRACE ログを有効にしてサーバーログを添付することを検討してください。大きすぎる場合は、操作中にログに追加されたメッセージを含む、サーバーログのスニペットだけを含めるのが良いでしょう。これにより、問題が発生します。

- LDAP プロバイダーを作成すると、次で始まるメッセージがサーバーログの INFO レベルに表示されます。

```
Creating new LDAP Store for the LDAP storage provider: ...
```

LDAP プロバイダーの設定が表示されます。質問またはバグを報告する前に、このメッセージを LDAP 設定に含めることが推奨されます。最終的には、一部の設定変更 (含めない) をプレースホルダーの値に置き換えることもあると考えられます。1つ目は **bindDn=some-placeholder** です。 **connectionUrl** の場合は、自由に置き換えるようにしてください。ただし、一般的には、使用されたプロトコル (**ldap** vs **ldaps**) を含めると便利です。同様に、LDAP マッパーの設定の詳細を含めると役に立ちます。これは、DEBUG レベルで以下のようなメッセージと共に表示されます。

```
Mapper for provider: XXX, Mapper name: YYY, Provider: ZZZ ...
```

これらのメッセージは、有効になっている DEBUG ログとともに表示されることに注意してください。

- パフォーマンスまたは接続プールの問題を追跡するには、LDAP プロバイダーのプロパティ **Connection Pool Debug Level** の値を **all** の値に設定することを検討してください。これによ

り、LDAP 接続プールのロギングが含まれるサーバーログに多くの追加メッセージがサーバーログに追加されます。これは、接続プールまたはパフォーマンスに関連する問題を追跡するために使用できます。



注記

接続プールの設定を変更した後に、Keycloak サーバーを再起動して LDAP プロバイダー接続の再初期化を適用する必要がある場合があります。

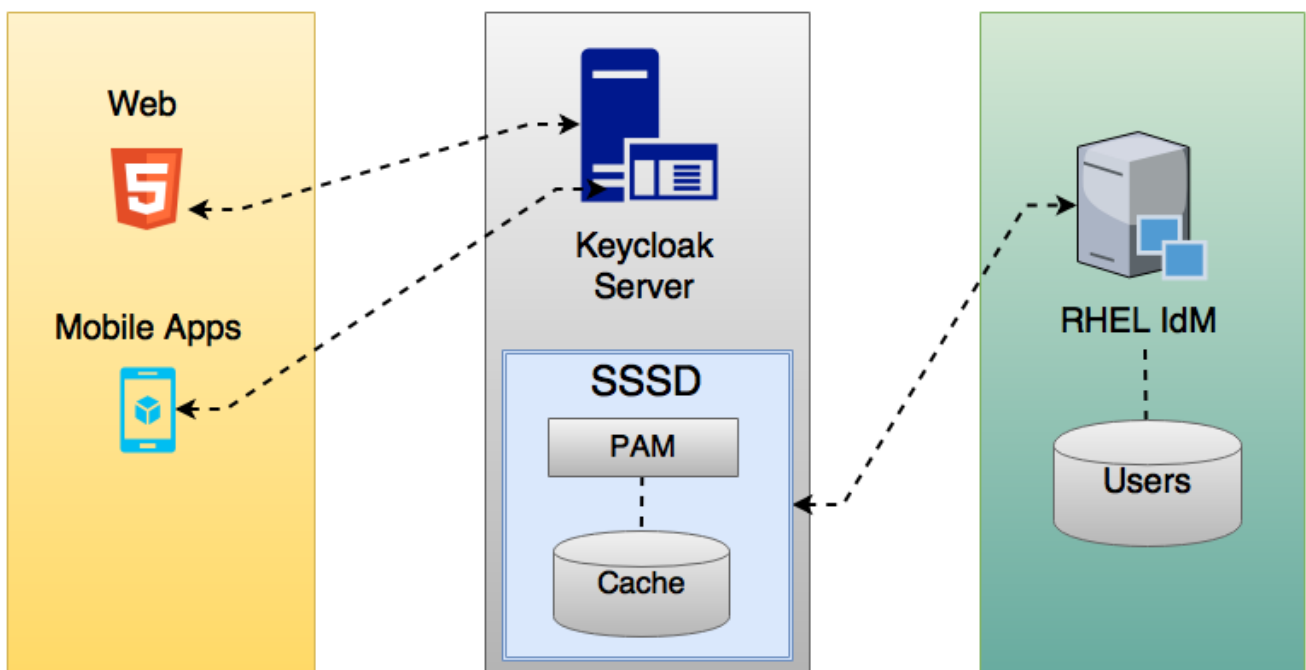
サーバーの再起動後に接続プールのメッセージがこれ以上表示されない場合は、接続プールが LDAP サーバーでは機能しないことを示すことができます。

- LDAP の問題を報告する場合、ターゲットデータとともに LDAP ツリーの一部を添付すると、環境で問題が発生します。たとえば、一部のユーザーログインに多くの時間がかかる場合、さまざまなグループエントリーの **member** の数を表示する LDAP エントリーをアタッチすることができます。この場合、それらのグループエントリーが Red Hat build of Keycloak などのグループ LDAP マッパー (またはロール LDAP マッパー) にマップされていれば、追加すると役立ちます。

4.4. SSSD および FREEIPA IDENTITY MANAGEMENT の統合

Red Hat build of Keycloak には、[System Security Services Daemon \(SSSD\)](#) プラグインが含まれています。SSSD は Fedora および Red Hat Enterprise Linux (RHEL) に含まれており、複数の ID プロバイダーおよび認証プロバイダーへのアクセスを提供します。SSSD は、フェイルオーバーやオフラインサポートなどの利点も提供します。詳細は、[Red Hat Enterprise Linux Identity Management のドキュメント](#) を参照してください。

SSSD は、認証とアクセス制御を提供する FreeIPA アイデンティティ管理 (IdM) サーバーとも統合します。この統合により、Red Hat build of Keycloak は特権アクセス管理 (PAM) サービスに対して認証し、SSSD からユーザーデータを取得できるようになります。Linux 環境における Red Hat Identity Management の使用に関する詳細は、[Red Hat Enterprise Linux Identity Management のドキュメント](#) を参照してください。



Red Hat build of Keycloak と SSSD は、読み取り専用の D-Bus インターフェイスを介して通信します。このため、ユーザーのプロビジョニングおよび更新方法は、FreeIPA/IdM 管理インターフェイスを使用

する方法です。デフォルトでは、インターフェイスはユーザー名、メール、名、および姓をインポートします。



注記

Red Hat build of Keycloak はグループとロールを自動的に登録しますが、同期はしません。Red Hat build of Keycloak 内で Red Hat build of Keycloak 管理者が加えた変更は、SSSD と同期されません。

4.4.1. freeipa/IdM サーバー

FreeIPA コンテナイメージは [Quay.io](https://quay.io) で入手できます。FreeIPA サーバーを設定するには、[FreeIPA のドキュメント](#) を参照してください。

手順

1. 以下のコマンドを使用して FreeIPA サーバーを実行します。

```
docker run --name freeipa-server-container -it \
-h server.freeipa.local -e PASSWORD=YOUR_PASSWORD \
-v /sys/fs/cgroup:/sys/fs/cgroup:ro \
-v /var/lib/ipa-data:/data:Z freeipa/freeipa-server
```

server.freeipa.local のパラメーター **-h** は FreeIPA/IdM サーバーのホスト名を表します。**YOUR_PASSWORD** を独自のパスワードに変更します。

2. コンテナが起動したら、**/etc/hosts** ファイルを以下のように変更します。

```
x.x.x.x server.freeipa.local
```

この変更を加えない場合は、DNS サーバーを設定する必要があります。

3. SSSD フェデレーションプロバイダーが Red Hat build of Keycloak で開始および実行されるように、以下のコマンドを使用して IPA ドメインに Linux サーバーを登録します。

```
ipa-client-install --mkhomedir -p admin -w password
```

4. クライアントで以下のコマンドを実行して、インストールが機能していることを確認します。

```
kinit admin
```

5. パスワードを入力します。

6. 以下のコマンドを使用して IPA サーバーにユーザーを追加します。

```
$ ipa user-add <username> --first=<first name> --last=<surname> --email=<email address> \
--phone=<telephoneNumber> --street=<street> --city=<city> --state=<state> --postalcode= \
<postal code> --password
```

7. kinit を使用してユーザーのパスワードを強制的に設定します。

```
kinit <username>
```

8. 以下のコマンドを入力して通常の IPA 操作を復元します。

```
kdestroy -A
kinit admin
```

4.4.2. SSSD および D-Bus

フェデレーションプロバイダーは、D-BUS を使用して SSSD からデータを取得します。PAM を使用してデータを認証します。

手順

1. sssd-dbus RPM をインストールします。

```
$ sudo yum install sssd-dbus
```

2. 以下のプロビジョニングスクリプトを実行します。

```
$ bin/federation-sssd-setup.sh
```

このスクリプトは、Red Hat build of Keycloak 用に SSSD と PAM を設定するためのガイドとしても使用できます。これを実行すると、`/etc/sss/sss.conf` に次の変更が加えられます。

```
[domain/your-hostname.local]
...
ldap_user_extra_attrs = mail:mail, sn:sn, givenname:givenname,
telephoneNumber:telephoneNumber
...
[sss]
services = nss, sudo, pam, ssh, ifp
...
[ifp]
allowed_uids = root, yourOSUsername
user_attributes = +mail, +telephoneNumber, +givenname, +sn
```

OS ユーザーがこのインターフェイスを介して IPA サーバーに問い合わせできるように、**ifp** サービスが SSSD に追加され、設定されます。

このスクリプトは、SSSD 経由でユーザーを認証するための新しい PAM サービス (`/etc/pam.d/keycloak`) も作成します。

```
auth required pam_sss.so
account required pam_sss.so
```

3. **dbus-send** を実行して、設定が正常に行われていることを確認します。

```
dbus-send --print-reply --system --dest=org.freedesktop.sssd.infopipe
/org/freedesktop/sss/infopipe org.freedesktop.sssd.infopipe.GetUserAttr string:<username>
array:string:mail,givenname,sn,telephoneNumber
```

```
dbus-send --print-reply --system --dest=org.freedesktop.sssd.infopipe
/org/freedesktop/sss/infopipe org.freedesktop.sssd.infopipe.GetUserGroups string:
<username>
```

セットアップが成功すると、各コマンドでユーザーの属性とグループがそれぞれ表示されます。タイムアウトまたはエラーが発生した場合、Red Hat build of Keycloak で実行されているフェデレーションプロバイダーはデータを取得できません。このエラーは、通常、サーバーが FreeIPA IdM サーバーに登録されていないか、SSSD サービスにアクセスするパーミッションがないために発生します。

SSSD サービスにアクセスする権限がない場合は、Red Hat build of Keycloak サーバーを実行しているユーザーが次のセクションの `/etc/sss/sss.conf` ファイルに存在することを確認してください。

```
[ifp]
allowed_uids = root, yourOSUsername
```

ホスト内に `ipaapi` システムユーザーが作成されます。このユーザーは、`ifp` サービスに必要です。システムにユーザーが作成されたことを確認します。

```
grep ipaapi /etc/passwd
ipaapi:x:992:988:IPA Framework User:/:/sbin/nologin
```

4.4.3. SSSD フェデレーションプロバイダーの有効化

Red Hat build of Keycloak は、[DBus-Java](#) プロジェクトを使用して D-Bus および [JNA](#) と低レベルで通信し、オペレーティングシステムのプラグ可能な認証モジュール (PAM) 経由で認証します。

現在、Red Hat build of Keycloak には **SSSD** プロバイダーの実行に必要なライブラリーがすべて含まれていますが、JDK バージョン 17 が必要です。そのため、ホスト設定が正しく、Red Hat build of Keycloak の実行に JDK 17 が使用されている場合に限り、**SSSD** プロバイダーが表示されます。

4.4.4. フェデレーションされた SSSD ストアの設定

インストール後に、フェデレーションされた SSSD ストアを設定します。

手順

1. メニューの **User Federation** をクリックします。
2. すべてが正常にセットアップされると、**Add Sssd providers** ボタンがページに表示されます。そのボタンをクリックします。
3. 新しいプロバイダーに名前を割り当てます。
4. **Save** をクリックします。

これで、Red Hat build of Keycloak に対して、FreeIPA/IdM ユーザーと認証情報を使用して認証できるようになりました。

4.5. カスタムプロバイダー

Red Hat build of Keycloak には、カスタムプロバイダーを開発するための User Storage Federation 用のサービスプロバイダーインターフェイス (SPI) があります。カスタマープロバイダーの開発に関するドキュメントは、[サーバー開発者ガイド](#) を参照してください。

第5章 ユーザーの管理

管理コンソールから、ユーザーの管理に使用できる幅広いアクションがあります。

5.1. ユーザーの作成

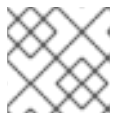
ユーザーが必要とするアプリケーションを持つレルムにユーザーを作成します。他のレルムの作成を目的としたマスターレルムでユーザーを作成しないでください。

前提条件

- マスターレルム以外のレルムを使用している。

手順

1. メニューの **Users** をクリックします。
2. **Add User** をクリックします。
3. 新規ユーザーの詳細を入力します。



注記

Username は、唯一の必須フィールドです。

4. **Save** をクリックします。詳細を保存すると、新規ユーザーの **Management** ページが表示されます。

5.2. ユーザーの認証情報の定義

Credentials タブでユーザーの認証情報を管理できます。

認証情報管理

The screenshot shows a user management interface. On the left is a dark sidebar with a menu including 'Master', 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users' (highlighted), 'Groups', 'Sessions', and 'Events'. The main content area shows the breadcrumb 'Users > User details' and the user name 'johndoe'. Below the name are tabs for 'Details', 'Attributes', 'Credentials' (active), 'Role mapping', 'Groups', 'Consents', and 'Identity provider links'. A large plus sign icon is centered, with the text 'No credentials' and a message: 'This user does not have any credentials. You can set password for this user.' A blue button labeled 'Set password' is at the bottom.

認証情報の優先順位を変更するには、行をドラッグアンドドロップします。新しい順序によって、そのユーザーの認証情報の優先順位が決まります。最上部にある認証情報が最も優先されます。優先順位は、ユーザーのログイン後に最初に表示される認証情報を決定します。

型

この列には、パスワードや OTP などの認証情報のタイプが表示されます。

User Label

これは、ログイン時に選択オプションとして示される際に認証情報を認識するための割り当て可能なラベルです。認証情報を記述するために任意の値に設定できます。

Data

これは、認証情報に関する機密ではない技術情報です。これは、デフォルトでは非表示になっています。Show data... をクリックすると、認証情報のデータを表示できます。

アクション

ユーザーのパスワードを変更するには **Reset password** をクリックし、認証情報を削除するには **Delete** をクリックします。

管理コンソールの特定のユーザーに他の種類の認証情報は設定できません。このタスクはユーザーの責任者です。

ユーザーが OTP デバイスを失った場合や、認証情報に不正アクセスが発生した場合は、ユーザーの認証情報を削除できます。**Credentials** タブでユーザーの認証情報のみを削除できます。

5.2.1. ユーザーのパスワードの設定

ユーザーにパスワードがない場合や、パスワードが削除された場合、パスワードの**設定セクション**が表示されます。

ユーザーのパスワードがすでにある場合は、**Reset Password** セクションでリセットできます。

手順

1. メニューの **Users** をクリックします。Users ページが表示されます。
2. ユーザーを選択します。
3. **Credentials** タブをクリックします。
4. **パスワードの設定** セクションに新しいパスワードを入力します。
5. **パスワードの設定** をクリックします。



注記

Temporary が **ON** の場合は、初回ログイン時にパスワードを変更する必要があります。ユーザーが指定したパスワードを維持できるようにするには、**Temporary** を **OFF** に設定します。ユーザーは、**Set Password** をクリックしてパスワードを変更する必要があります。

5.2.2. ユーザーにパスワードのリセットを要求する

ユーザーに、パスワードをリセットするように要求することもできます。

手順

1. メニューの **Users** をクリックします。Users ページが表示されます。
2. ユーザーを選択します。

3. **Credentials** タブをクリックします。
4. **認証情報のリセット** をクリックします。
5. リストから **パスワードの更新** を選択します。
6. **Send Email** をクリックします。送信されたメールには、ユーザーを **Update Password** ウィンドウに転送するリンクが含まれます。
7. オプションで、メールリンクの有効性を設定できます。これは、**Realm Settings** の **Tokens** タブでデフォルトの事前設定に設定されます。

5.2.3. OTP の作成

OTP がレルムの条件である場合、ユーザーは Red Hat build of Keycloak アカウントコンソールに移動し、新しい OTP ジェネレーターを再設定する必要があります。OTP が必要な場合は、ログイン時に新しい OTP ジェネレーターを再設定する必要があります。

この代わりに、ユーザーが OTP ジェネレーターをリセットするユーザーヘメールを送信することもできます。以下の手順では、ユーザーが OTP 認証情報をすでに持っているかどうかも適用されます。

前提条件

- 適切なレルムにログインしている。

手順

1. メインメニューで **Users** をクリックします。 **Users** ページが表示されます。
2. ユーザーを選択します。
3. **Credentials** タブをクリックします。
4. **認証情報のリセット** をクリックします。
5. **Reset Actions** を **Configure OTP** に設定します。
6. **Send Email** をクリックします。送信されたメールには、 **OTP 設定ページ** に転送するリンクが含まれます。

5.3. ユーザー属性の設定

ユーザー属性は、各ユーザーのカスタマイズされたエクスペリエンスを提供します。コンソールでユーザー属性を設定して、各ユーザーに個人的なアイデンティティを作成できます。

ユーザー

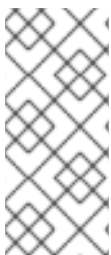
The screenshot shows the Keycloak user management interface. On the left is a dark sidebar menu with options: Master, Manage, Clients, Client scopes, Realm roles, Users (highlighted), Groups, Sessions, Events, Configure, and Realm settings. The main content area shows the breadcrumb 'Users > User details' and the user name 'johndoe'. Below this are tabs for 'Details', 'Attributes' (selected), 'Credentials', 'Role mapping', 'Groups', 'Consents', and 'Identity provider links'. The 'Attributes' tab contains a table with two columns: 'Key' and 'Value'. The first row has 'mobile' in the key field and '5555-5555-555' in the value field. The second row has 'Type a key' and 'Type a value'. Below the table is a '+ Add an attribute' button and 'Save' and 'Revert' buttons.

前提条件

- ユーザーが存在するレルムにある。

手順

1. メニューの **Users** をクリックします。
2. 管理するユーザーを選択します。
3. **Attributes** タブをクリックします。
4. キー フィールドに属性名を入力します。
5. 値 フィールド属性値を入力します。
6. **Save** をクリックします。



注記

一部の読み取り専用属性は管理者が更新されることは想定されていません。これには、LDAP プロバイダーによって自動的に入力される **LDAP_ID** のように設計によって読み取り専用の属性が含まれます。セキュリティ上の理由から、ユーザー管理者が他の属性を読み取り専用にする必要があります。詳細については、[セキュリティ上の脅威の軽減](#) の章を参照してください。

5.4. ユーザーの自己登録の許可

Red Hat build of Keycloak をサードパーティー認可サーバーとして使用して、自己登録ユーザーを含むアプリケーションユーザーを管理できます。自己登録を有効にすると、ログインページに登録リンクが表示され、ユーザーがアカウントを作成できるようにします。

登録リンク

Sign in to your account

Username or email

Password

Sign In

New user? [Register](#)

登録を完了するには、ユーザーは登録フォームにプロフィール情報を追加する必要があります。登録フォームは、ユーザーが完了する必要のあるフィールドを削除または追加することでカスタマイズできます。

ID ブローカリングと管理 API についての説明

自己登録が無効になっている場合でも、次のいずれかの方法で新しいユーザーを Red Hat build of Keycloak に追加できます。

- 管理者は、管理コンソール (または管理 REST API) を使用して新しいユーザーを追加できます。
- アイデンティティブローカリングが有効になっている場合、アイデンティティプロバイダーが認証した新しいユーザーが Red Hat build of Keycloak ストレージに自動的に追加/登録されることがあります。詳細は [アイデンティティブローカリングの章の最初のログインフローセクション](#) を参照してください。

また、特定のユーザーストレージが有効になっている場合、[サードパーティーのユーザーストレージ](#) (LDAP など) からのユーザーも Red Hat build of Keycloak で自動的に利用可能になります。

関連情報

- ユーザー登録のカスタマイズについて、詳細は [サーバー開発者ガイド](#) を参照してください。

5.4.1. ユーザー登録の有効化

ユーザーが自己登録できるようにします。

手順

1. メインメニューで **Realm Settings** をクリックします。
2. **Login** タブをクリックします。
3. **ユーザー登録** を **ON** に切り替えます。

この設定を有効にすると、管理コンソールのログインページに **登録** リンクが表示されます。

5.4.2. 新規ユーザーとしての登録

新しいユーザーとして、初回ログインするには、登録フォームを完了する必要があります。プロフィール情報と、登録するパスワードを追加します。

登録フォーム

Register

First name

Last name

Email

Username

Password

Confirm password

[« Back to Login](#)

Register

前提条件

- ユーザー登録が有効になっています。

手順

1. ログインページの **登録** リンクをクリックします。登録ページが表示されます。
2. ユーザープロフィール情報を入力します。

3. 新しいパスワードを入力します。

4. **Register** をクリックします。

5.4.3. 登録時に利用規約への同意を求める

ユーザーの登録時に、利用規約への同意を要求できます。

必要な契約条件を記載した登録フォーム

Register

First name



Last name

Email

Username

Password



Confirm password



Terms and Conditions

Terms and conditions to be defined

I agree to the terms and conditions

You must agree to our terms and conditions.

[« Back to Login](#)

Register

- ユーザー登録が有効になっている。
- 利用規約が必要なアクションが有効になっている。

手順

1. メニューで **Authentication** をクリックします。 **Flows** タブをクリックします。
2. **registration** フローをクリックします。
3. **Terms and Conditions** の行で **Required** を選択します。

登録時に利用規約への同意を必須にする

The screenshot shows the configuration page for the 'registration' flow. At the top, there are tabs for 'Default' and 'Built-in', and an 'Action' dropdown. Below are buttons for 'Add step' and 'Add sub-flow'. The main area is a table with two columns: 'Steps' and 'Requirement'.

Steps	Requirement
registration form registration form	Required
Registration User Creation	Required
Profile Validation	Required
Password Validation	Required
Recaptcha	Disabled
Terms and conditions	Required

5.5. ログイン時に必要なアクションの定義

ユーザーが初回ログイン時に実行する必要があるアクションを設定できます。これらのアクションは、ユーザーが認証情報を提供する後に必要になります。最初のログイン後、これらのアクションは不要になりました。必須アクションを、そのユーザーの **Details** タブに追加します。

管理者によってこのユーザーに明示的に追加されていない場合でも、ログイン時に一部の必須アクションが自動的にトリガーされます。たとえば、[Password policies](#) が、ユーザーパスワードを X 日ごとに変更する必要があるように設定されている場合は、**Update password** アクションがトリガーとなる可能性があります。または、**verify profile** アクションでは、一部のユーザー属性がユーザープロフィール設定の要件と一致しない限り、ユーザーに [User profile](#) 更新を要求することができます。

以下は、必須アクションタイプの例です。

パスワードの更新

ユーザーはパスワードを変更する必要があります。

OTP の設定

これを設定すると、Free OTP または Google Authenticator アプリケーションのいずれかを使用して、モバイルデバイスにワンタイムパスワードジェネレーターを設定する必要があります。

メールの確認

ユーザーは、メールアカウントを検証する必要があります。電子メールは、クリックする必要がある検証リンクを持つユーザーに送信されます。このワークフローが正常に完了したら、ユーザーはログインできるようになります。

プロフィールの更新

ユーザーは、名前、アドレス、電子メール、電話番号などのプロフィール情報を更新する必要があります。

5.5.1.1 人のユーザーに必要なアクションの設定

任意のユーザーに必要なアクションを設定できます。

手順

1. メニューの **Users** をクリックします。
2. リストからユーザーを選択します。
3. **Required User Actions** リストに移動します。

The screenshot shows the 'User details' page for a user named 'johndoe'. The left sidebar contains a navigation menu with 'Users' selected. The main content area has tabs for 'Details', 'Attributes', 'Credentials', 'Role mapping', 'Groups', 'Consents', and 'Identity provider links'. The 'Details' tab is active, showing fields for ID, Created at, Username, Email, Email verified (toggle), First name, Last name, Enabled (toggle), and Required user actions. The 'Required user actions' section shows a list of actions, with 'Update Password' selected and an 'X' icon to remove it. There are 'Save' and 'Revert' buttons at the bottom.

4. アカウントに追加するすべてのアクションを選択します。
5. アクション名の横にある X をクリックして削除します。
6. 追加するアクションを選択したら、**Save** をクリックします。

5.5.2. すべてのユーザーに必要なアクションの設定

すべての新規ユーザーの最初のログイン前に必要なアクションを指定できます。要件は、**Users** ページの **Add User** ボタンまたはログインページの **Register** リンクが作成したユーザーに適用されます。

手順

1. メニューで **Authentication** をクリックします。
2. **Required Actions** タブをクリックします。
3. 1つ以上の必要なアクションの **デフォルトのアクションとして設定** 列のチェックボックスをクリックします。新規ユーザーが初めてログインしたら、選択したアクションを実行する必要があります。

5.5.3. 必須アクションとしての利用規約の有効化

新規ユーザーが Red Hat build of Keycloak に初めてログインする前に、利用規約に同意する必須アクションを有効にできます。

手順

1. メニューで **Authentication** をクリックします。
2. **Required Actions** タブをクリックします。
3. **利用規約** のアクションを有効にします。
4. ベースログインテーマの **terms.ftl** ファイルを編集します。

関連情報

- テーマの拡張および作成の詳細は、[サーバー開発者ガイド](#) を参照してください。

5.6. アプリケーションが開始したアクション

アプリケーション開始アクション (AIA) を使用すると、クライアントアプリケーションは、ユーザーに Red Hat build of Keycloak でアクションを実行するように要求できます。通常、OIDC クライアントアプリケーションは、ユーザーにログインを要求する場合、[OIDC セクション](#) で説明されているように、そのユーザーをログイン URL にリダイレクトします。ログイン後、ユーザーはクライアントアプリケーションにリダイレクトされます。ユーザーは、[前のセクション](#) で説明したように管理者によって要求されたアクションを実行し、すぐにアプリケーションにリダイレクトされます。ただし、AIA を使用すると、クライアントアプリケーションはログイン時にユーザーにいくつかの必要なアクションを要求できます。これは、ユーザーがクライアント上ですでに認証されており、アクティブな SSO セッションがある場合でも実行できます。これは、要求されたアクションを含む値を持つ **kc_action** パラメーターを OIDC ログイン URL に追加することによってトリガーされます。たとえば、**kc_action=UPDATE_PASSWORD** パラメーターです。



注記

kc_action パラメーターは、OIDC 仕様でサポートされていない Red Hat build of Keycloak のプロプライエタリーメカニズムです。



注記

アプリケーションによって開始されるアクションは、OIDC クライアントに対してのみサポートされます。

したがって、AIA を使用する場合、フローの例は次のようになります。

- クライアントアプリケーションは、**kc_action=UPDATE_PASSWORD** などの追加パラメーターを使用して、ユーザーを OIDC ログイン URL にリダイレクトします。
- [認証フローのセクション](#) で説明されているように、常にトリガーされる **browser** フローがあります。ユーザーが認証されなかった場合、そのユーザーは通常のログイン時と同様に認証する必要があります。ユーザーがすでに認証されている場合、そのユーザーは、アクティブに再認証して認証情報を再度入力する必要なく、SSO Cookie によって自動的に再認証される可能性があります。この場合、そのユーザーは特定のアクション（この場合はパスワードの更新）を実行する画面に直接リダイレクトされます。ただし、場合によっては、ユーザーが SSO Cookie を持っている場合でも、アクティブな再認証が必要になります（詳細は [こちら](#) を参照してください）。
- 特定のアクション（この場合は **パスワードの更新**）の画面がユーザーに表示されるため、ユーザーは特定のアクションを実行する必要があります。
- その後、ユーザーはクライアントアプリケーションにリダイレクトされます。

AIA は、Red Hat build of Keycloak [アカウントコンソール](#) によってパスワードの更新を要求したり、OTP や WebAuthn などの他の認証情報をリセットしたりするために使用されることに注意してください。



警告

パラメーター **kc_action** が使用されたとしても、ユーザーが常にアクションを実行すると想定するだけでは不十分です。たとえば、ユーザーがブラウザの URL から **kc_action** パラメーターを手動で削除した可能性があります。したがって、クライアントが **kc_action=CONFIGURE_TOTP** を要求した後、ユーザーがアカウントの OTP を持っているという保証はありません。ユーザーが 2 要素認証を設定したことを確認する場合は、クライアントアプリケーションでそれが設定されていることを確認する必要があります。たとえば、トークン内の **acr** のようなクレームを確認します。

5.6.1. AIA 中の再認証

アクティブな SSO セッションによりユーザーがすでに認証されている場合、通常、そのユーザーはアクティブに再認証する必要はありません。ただし、そのユーザーが 5 分以上前にアクティブに認証されている場合、クライアントは AIA が要求されたときに再認証を要求できます。このガイドラインには次のような例外があります。

- **delete_account** アクションでは、常にユーザーが能動的に再認証する必要があります。
- より短い再認証を使用する場合は、指定されたより短い値を持つ **max_age** などのパラメータークエリーパラメーターを使用するか、最終的には **prompt=login** を使用できます。これにより、OIDC 仕様で説明されているように、ユーザーによる能動的な再認証が常に要求されま

す。デフォルトの5分よりも長い値に **max_age** を使用することはサポートされていないことに注意してください。現在、**max_age** は、値をデフォルトの5分より短くするためにのみ使用できます。

5.6.2. 利用可能なアクション

利用可能なすべてのアクションを表示するには、管理コンソールにログインし、右上隅にある **Realm info** → **Provider info** タブ→プロバイダーの **required-action** 検索をクリックします。ただし、**Required actions** タブでレルムに対して有効になっているアクションに基づいて、これがさらに制限される可能性があることに注意してください。

5.7. ユーザーの検索

ユーザーを検索すると、ユーザーのグループやロールなどのユーザーに関する詳細情報が表示されます。

前提条件

- ユーザーが存在するレルムにある。

手順

1. メインメニューで **Users** をクリックします。この **Users** ページが表示されます。
2. 検索ボックスに、検索するユーザーのフルネーム、姓名、またはメールアドレスを入力します。検索では、条件に一致するすべてのユーザーが返されます。ユーザーの照合に使用される基準は、検索ボックスで使用される構文により異なります。
 - a. **"somevalue"** → 文字列 **"somevalue"** の完全一致検索を実行します。
 - b. ***somevalue*** → **LIKE '%somevalue%'** DB クエリーと同じような接中辞検索を実行します。
 - c. **somevalue*** または **somevalue** → **LIKE 'somevalue%'** DB クエリーと同じような接頭辞検索を実行します。



注記

Users ページで実行される検索には、Red Hat build of Keycloak のデータベースと、ユーザーフェデレーションが行われた設定済みバックエンド (LDAP など) の両方の検索が含まれます。フェデレーションされたバックエンドで見つかったユーザーが Red Hat build of Keycloak のデータベースに存在しない場合、インポートされます。

関連情報

- ユーザーフェデレーションの詳細については、[ユーザーフェデレーション](#) を参照してください。

5.8. ユーザーの削除

アプリケーションへのアクセスがなくなったユーザーを削除できます。ユーザーが削除されると、ユーザープロフィールとデータも削除されます。

手順

1. メニューの **Users** をクリックします。 **Users** ページが表示されます。
2. **View all users** をクリックして、削除するユーザーを検索します。



注記

または、検索バーを使用してユーザーを検索することもできます。

3. 削除するユーザーの横にあるアクションメニューから **Delete** をクリックし、削除を確認します。

5.9. ユーザーによるアカウントの削除の有効化

管理コンソールでこの機能を有効にすると、エンドユーザーおよびアプリケーションは、アカウントコンソールでアカウントを削除できます。この機能を有効にすると、その機能を特定のユーザーに提供できます。

5.9.1. アカウントの削除機能の有効化

この機能により、**Required Actions** タブでこの機能を有効にします。

手順

1. メニューで **Authentication** をクリックします。
2. **Required Actions** タブをクリックします。
3. **Delete Account** 行で **Enabled** を選択します。

Required Actions タブでアカウントを削除します。

Authentication
Authentication is the area where you can configure and manage different credential types. [Learn more](#)

Flows Required actions Policies

Required actions	Enabled	Set as default action
☰ Configure OTP	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Terms and Conditions	<input type="checkbox"/> Off	<input checked="" type="checkbox"/> Disabled off
☰ Update Password	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Update Profile	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Verify Email	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Delete Account	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Update User Locale	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Webauthn Register Passwordless	<input type="checkbox"/> Off	<input checked="" type="checkbox"/> Disabled off
☰ Webauthn Register	<input type="checkbox"/> Off	<input checked="" type="checkbox"/> Disabled off
☰ Verify Profile	<input type="checkbox"/> Off	<input checked="" type="checkbox"/> Disabled off

5.9.2. ユーザーに delete-account ロールの指定

アカウントの削除を許可するロールを特定のユーザーに付与できます。

手順

1. メニューの **Users** をクリックします。
2. ユーザーを選択します。
3. **Role Mappings** タブをクリックします。
4. **Assign role** ボタンをクリックします。
5. **account delete-account** をクリックします。
6. **Assign** をクリックします。

delete-account ロール

Assign roles to johndoe account

✕

Filter by Origin ✕ Search by role name → 1-7 < >

account 7 ✕

<input type="checkbox"/>	Name	Description
<input type="checkbox"/>	account view-profile	`\${role_view-profile}`
<input type="checkbox"/>	account view-applications	`\${role_view-applications}`
<input type="checkbox"/>	account view-consent	`\${role_view-consent}`
<input type="checkbox"/>	account manage-account-links	`\${role_manage-account-links}`
<input checked="" type="checkbox"/>	account delete-account	`\${role_delete-account}`
<input type="checkbox"/>	account manage-account	`\${role_manage-account}`
<input type="checkbox"/>	account manage-consent	`\${role_manage-consent}`

1-7 < >

Assign Cancel

5.9.3. アカウントの削除

delete-account ロールを取得したら、独自のアカウントを削除できます。

1. アカウントコンソールにログインします。
2. **Personal Info** ページの下部に、**Delete Account** をクリックします。

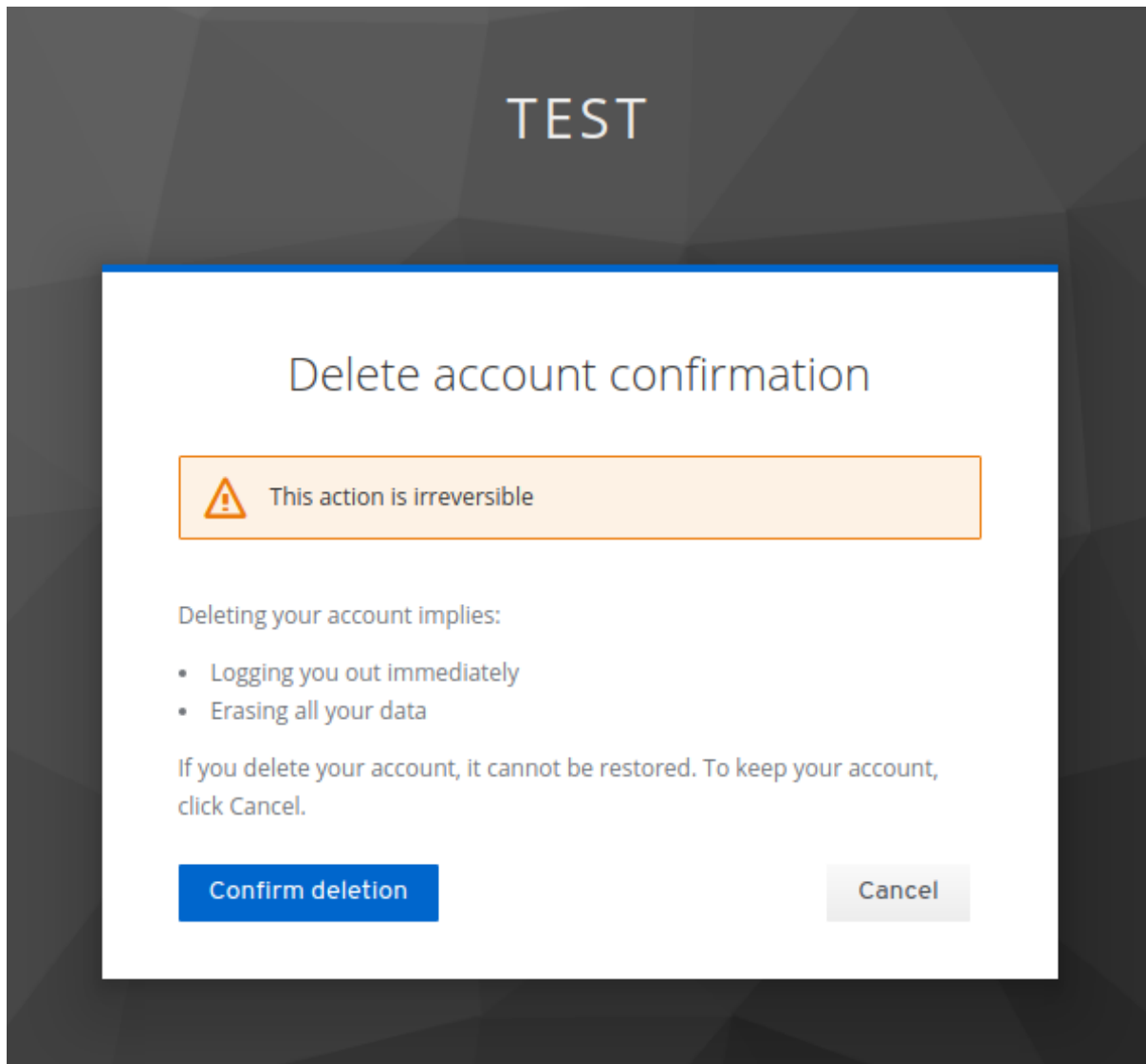
アカウントページの削除

The screenshot shows the 'Personal info' page in Keycloak. On the left is a dark sidebar with three menu items: 'Personal info' (selected), 'Account security', and 'Applications'. The main content area is titled 'Personal info' and contains the following elements:

- Header: 'Personal info' and 'Manage your basic information.'
- Text: 'All fields are required.'
- Form fields: 'Username' (filled with 'johndoe'), 'Email' (empty), 'First name' (filled with 'John'), and 'Last name' (filled with 'Doe').
- Buttons: 'Save' (blue) and 'Cancel' (light blue).
- Section: 'Delete Account' (indicated by a dropdown arrow).
- Text: 'This is irreversible. All your data will be permanently destroyed, and irretrievable.'
- Button: 'Delete' (red).

3. 認証情報を入力し、削除を確定します。

確認削除



注記

このアクションは元に戻せません。Red Hat build of Keycloak 内のすべてのデータが削除されます。

5.10. ユーザーの権限借用

適切なパーミッションを持つ管理者はユーザーの権限を借用できます。たとえば、アプリケーションでバグが発生した場合、管理者はユーザーの権限を借用して問題を調査または複製できます。

レルムの **impersonation** ロールを持つユーザーは、ユーザーの権限を借用できます。

手順

1. メニューの **Users** をクリックします。
2. 偽装するユーザーをクリックします。
3. **Action** リストから **Impersonate** を選択します。

The screenshot shows the Keycloak user management interface. On the left is a dark sidebar with navigation options: Master, Manage, Clients, Client scopes, Realm roles, Users (highlighted), Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area is titled 'Users > User details' and shows the profile for 'johndoe'. The profile includes fields for ID, Created at, Username, Email, Email verified (toggle), First name, Last name, Enabled (toggle), and Required user actions. At the bottom are 'Save' and 'Revert' buttons.

- 管理者とユーザーが同じレルムにある場合、管理者はログアウトされ、切り替えているユーザーとして自動的にログインします。
- 管理者およびユーザーが異なるレルムにある場合、管理者はログインし、そのユーザーのレルムでユーザーとしてログインします。

両方のインスタンスで、切り替えられたユーザーの **User Account Management** ページが表示されません。

関連情報

- 管理権限の割り当ての詳細については、[管理コンソールのアクセス制御](#) の章を参照してください。

5.11. RECAPTCHA の有効化

登録をボットから保護するために、Red Hat build of Keycloak には Google reCAPTCHA が統合されています。

reCAPTCHA を有効にしたら、ログインテーマの **register.ftl** を編集して、登録ページで reCAPTCHA ボタンの配置と停止を設定できます。

手順


1. 以下の URL をブラウザに入力します。

<https://developers.google.com/recaptcha/>

2. reCAPTCHA サイトキーおよびシークレットを取得するために API キーを作成します。本手順で後で使用できるように、reCAPTCHA サイトキーおよび秘密を書き留めておきます。

**注記**

localhost はデフォルトで動作します。ドメインを指定する必要はありません。


3. Red Hat build of Keycloak 管理コンソールに移動します。
4. メニューで **Authentication** をクリックします。
5. **Flows** タブをクリックします。
6. リストから **Registration** を選択します。
7. reCAPTCHA の要件を **Required** に設定します。これにより、reCAPTCHA が有効になります。
8. reCAPTCHA 行の **歯車アイコン**  をクリックします。
9. **Config** リンクをクリックします。

reCAPTCHA 設定ページ

Recaptcha config ×

Alias * 

recaptcha

Recaptcha Site Key 

AAA0aY-SRkc3sZyw4Aanqfa27Bn

Recaptcha Secret 

6LcFEAkTAAAAMOSer

use recaptcha.net 

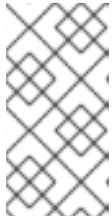
Off

Save

Cancel

- a. Google reCAPTCHA の Web サイトから生成された **Recaptcha Site Key** を入力します。
- b. Google reCAPTCHA の Web サイトから生成された **Recaptcha Secret** を入力します。

10. 登録ページを iframe として使用するよう Google を認可します。



注記

Red Hat build of Keycloak では、Web サイトの iframe にログインページのダイアログを含めることはできません。この制限は、クリック攻撃を防ぐことです。Red Hat build of Keycloak で設定されるデフォルトの HTTP 応答ヘッダーを変更する必要があります。

- a. メニューで **Realm Settings** をクリックします。
- b. **Security Defenses** タブをクリックします。
- c. **X-Frame-Options** ヘッダーのフィールドに <https://www.google.com> を入力します。
- d. **Content-Security-Policy** ヘッダーのフィールドに <https://www.google.com> を入力します。

関連情報

- テーマの拡張および作成の詳細は、[サーバー開発者ガイド](#) を参照してください。

5.12. ユーザープロファイルの定義

Red Hat build of Keycloak では、ユーザーは一連の属性に関連付けられます。これらの属性は、Red Hat build of Keycloak 内でユーザーをより適切に記述および識別するため、またユーザーに関する追加情報をアプリケーションに渡すために使用されます。

ユーザープロファイルは、ユーザー属性とレルム内で管理する方法を表す明確に定義されたスキーマを定義します。ユーザー情報に対する一貫したビューを提供することにより、管理者は属性の管理方法に関するさまざまな側面を制御できるだけでなく、追加の属性をサポートするように Red Hat build of Keycloak を容易に拡張できます。

他の機能の中で、管理者は以下を行うことができます。

- ユーザー属性のスキーマを定義
- コンテキスト情報に基づいて属性が必要かどうかを定義します (例: ユーザーまたは管理者に必要となる場合、または両方の場合、または要求されるスコープに応じて)。
- ユーザー属性を表示および編集するための特定のパーミッションを定義してください。これにより、一部の属性が表示できないか、3 番目の部分 (管理者を含む) によって変更されない、強力なプライバシー要件に準拠することができます。
- ユーザープロファイルのコンプライアンスを動的に実施して、ユーザー情報が常に、属性に関連付けられたメタデータとルールに準拠します。
- 組み込みバリデーターを利用するか、カスタムレジストリーを書き込むことで、属性ごとに検証ルールを定義します。
- 属性の定義やそれらを手動で変更しなくても、ユーザーがアカウントコンソール内の登録、更新プロファイル、ブローカー、個人情報などと対話するフォームを動的にレンダリングします。

ユーザープロファイルの機能は、ユーザープロファイル SPI によってサポートされます。デフォルトでは、これらの機能は無効になり、レルムは、レガシー動作と後方互換性を維持するデフォルト設定を使用するよう設定されます。



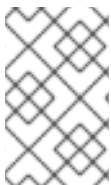
注記

レガシー動作とは、カスタム属性の管理方法に制限はなく、ユーザー名、メール、姓名などのユーザーのルート属性を管理する際に、Red Hat build of Keycloak によって使用されるデフォルトの制約が維持される動作です。登録、プロファイル更新、ブローカー、およびアカウントコンソールを介したアカウントフローなどのユーザーフローについては、ユーザーが上記の属性を変更できるように、上記の属性を使用するように制限されています。

すでに Red Hat build of Keycloak を使用している場合は、これまで使用してきた動作がレガシー動作です。

宣言的プロバイダーは、レガシー動作とは異なるため、管理コンソールと明確に定義された JSON スキーマを使用して、レルムにユーザープロファイル設定を定義する柔軟性が非常に高くなります。

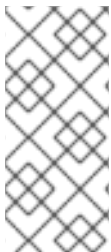
次のセクションでは、宣言型プロバイダーを使用して独自のユーザープロファイル設定を定義する方法を説明します。



注記

将来的には、Red Hat build of Keycloak ではレガシー動作がサポートされなくなります。理想的には、ユーザープロファイルが提供する新機能を確認し、それに応じてレルムを移行する必要があります。

5.12.1. ユーザープロファイルの有効化



注記

宣言型ユーザープロファイルは **テクノロジープレビュー** であるため、完全にサポートされていません。デフォルトでは無効になっています。

有効にするには、`--features=preview` または `--features=declarative-user-profile` を使用してサーバーを起動します。

declarative_user_profile 機能を有効にする他に、レルムのユーザープロファイルを有効にする必要があります。これには、左側のメニューの **Realm Settings** リンクをクリックし、**User Profile Enabled** スイッチを有効にします。

Myrealm

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

Enabled

Action ▾

<

General

Login

Email

Themes

Keys

Events

Localization

Security defenses

>

Realm ID *

Display name

HTML Display name

Frontend URL

Require SSL

User-managed access Off ?

User Profile Enabled On ?

Endpoints ? [OpenID Endpoint Configuration](#) [SAML 2.0 Identity Provider Metadata](#)

有効化して **Save** ボタンをクリックすると、ユーザー属性の設定を管理できる **User Profile** にアクセスできます。

レルムのユーザープロファイルを有効にすることで、Red Hat build of Keycloak はユーザープロファイル設定に基づいて属性を管理する方法に追加の制約を課すこととなります。つまり、以下は、この機能が有効な場合に予想される内容のリストです。

- 管理の観点からは、ユーザー詳細ページの **Attributes** タブに、ユーザープロファイル設定で定義された属性のみが表示されます。属性管理時には、属性ごとに定義された条件も考慮されません。
- アカウントコンソールにおける登録、更新プロファイル、ブローカー、個人情報などのユーザー向けフォームは、ユーザープロファイルの設定に基づいて動的にレンダリングされます。そのため、Red Hat build of Keycloak は、各種テンプレートに依存して、これらのフォームを動的にレンダリングします。

次のトピックでは、ユーザープロファイル設定を管理する方法と、レルムへの影響を説明します。

5.12.2. ユーザープロファイルの管理

ユーザープロファイル設定は、レルムごとに管理されます。これには、左側のメニューで **Realm Settings** リンクをクリックし、**User Profile** タブをクリックします。

ユーザープロファイルタブ

Realm settings > User profile

Master

 Enabled

Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

< Localization Security defenses Sessions Tokens Client policies **User profile** >

Attributes Attributes group JSON editor

▼ All groups ▾

Create attribute

Name	Display name	Attribute group
☰ username	`\${username}`	☰
☰ email	`\${email}`	☰
☰ firstName	`\${firstName}`	☰
☰ lastName	`\${lastName}`	☰
☰ test	test attribute	☰

Attributes サブタブでは、現在ユーザープロファイルに関連付けられている属性のリストがあります。デフォルトでは、設定はユーザー root 属性に基づいて作成され、各属性は検証およびパーミッションに関してデフォルトで設定されます。

Attribute Groups サブタブで、属性グループを管理できます。属性グループを使用すると、ユーザーに表示されるフォームをレンダリングする際に属性を関連付けることができます。



注記

現在、属性グループはレンダリング目的でのみ使用されますが、今後はリンク先の属性へのトップレベル設定の定義も有効にする必要があります。

JSON Editor サブタブで、明確に定義された JSON スキーマを使用して設定を表示および編集できます。他のタブで表示される JSON 設定に反映されたときに、変更を加えます。

次のセクションでは、**Attributes** サブタブで設定を管理する方法を学びます。

5.12.3. 属性の管理

Attributes サブタブで、ユーザープロファイルに関連する属性を作成、編集、および削除できます。

新しい属性を定義してユーザープロフィールに関連付けるには、属性リストの上部にある **Create attribute** ボタンをクリックします。

属性設定

[Realm settings](#) > [User profile](#) > [Create attribute](#)

Create attribute

Create a new attribute

General settings

Name [?]

Display name [?]

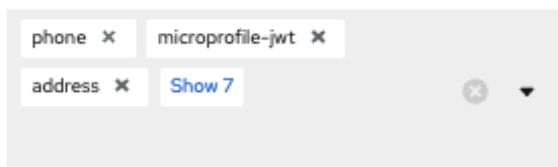
Attribute group [?]

Enabled when Always
 Scopes are requested

Required [?] On

Required for Both users and admins Only users Only admins

Required when Always
 Scopes are requested



Jump to section

[General settings](#)

[Permission](#)

[Validations](#)

[Annotations](#)

Permission

Who can edit? [?] User Admin

Who can view? [?] User Admin

who can view:     

Validations

[+ Add validator](#)

Validator name

Config

Create

Cancel

属性の設定時に、以下の設定を定義できます。

名前

属性の名前。

表示名

属性のユーザーフレンドリーな名前。主にユーザーに表示されるフォームをレンダリングする場合に使用されます。これは国際化をサポートするため、メッセージバンドルから値をロードできるようにします。

属性グループ

属性が属する属性グループ (ある場合)。

スコープを指定した場合に有効化

属性を動的に有効化するためにスコープのリストを定義します。設定されていない場合、属性は常に有効になり、ユーザーに表示されるフォームをレンダリングする際に、ユーザープロファイルの管理時に常にその制約が強制されます。そうしないと、リスト内のスコープのいずれかがクライアントによって要求される場合にのみ、同じ制約が適用されます。

必須

必要に応じて属性を設定します。有効でなければ、属性は任意です。それ以外の場合は、ユーザーおよび管理者が属性を提供する必要があります。また、必須の属性も、クライアントが必要とするスコープをベースにする必要があります。

パーミッション

このセクションでは、ユーザーおよび管理者の読み取りおよび書き込みパーミッションを定義できます。

検証

本セクションでは、属性値を管理する際に実行される検証を定義できます。Red Hat build of Keycloak では、選択可能なビルトインバリデーターのセットが提供されており、独自に追加することも可能です。

アノテーション



このセクションでは、アノテーションを属性に関連付けることができます。アノテーションは、レンダリングの目的で追加のメタデータをフロントエンドに渡すのに役立ちます。

5.12.3.1. パーミッションの管理

Permission セクションで、アクセスユーザーおよび管理者が属性の読み書きレベルを定義できます。

属性パーミッション

Permission

Who can edit? 	<input checked="" type="checkbox"/> User	<input checked="" type="checkbox"/> Admin
Who can view? 	<input checked="" type="checkbox"/> User	<input type="checkbox"/> Admin

そのためには、以下の設定を使用できます。

ユーザー表示可能か？

有効にすると、属性を表示できます。それ以外の場合は、ユーザーは属性にアクセスできません。

ユーザーを編集可能か？

有効にすると、ユーザーは属性を表示および編集できます。それ以外の場合は、ユーザーは属性への書き込み権限がありません。

管理者ビューはどれですか？

有効にすると、管理者は属性を表示できます。それ以外の場合は、管理者は属性にアクセスできません。

管理者を編集可能か？

有効にすると、管理者は属性を表示および編集できます。それ以外の場合は、管理者は属性への書き込み権限がありません。



注記

属性を作成すると、属性にパーミッションが設定されません。実質的には、ユーザーまたは管理者にも属性はアクセスできません。属性を作成したら、ターゲット対象者のみが属性を表示できるように、パーミッションを設定します。

パーミッションには、属性の管理方法とユーザーに表示される形式で属性の管理方法に直接影響があります。

たとえば、ユーザーが属性を表示可能とすると、管理コンソールを介してユーザーを管理する場合（ユーザー API から）、管理者は属性にアクセスできません。また、プロフィールの更新時には属性を変更できません。ユーザー属性が既存のアイデンティティストア（フェデレーション）から取得され、ソースアイデンティティストア以外の属性を更新せずに属性をユーザーに表示される場合に、興味のある設定。

同様に、ユーザーの読み取り専用アクセスを持つ管理者に対してのみ、属性を書き込み可能としてマークすることもできます。この場合、属性の管理は管理者のみが許可されます。

プライバシーの要件によっては、管理者にもアクセスできるが、ユーザーの読み取り/書き込みパーミッションで属性にアクセスすることが望ましい場合があります。

ユーザープロフィール設定に新しい属性を追加する場合は必ず、適切なパーミッションを設定してください。

5.12.3.2. 検証の管理

Validation セクションで、異なる形式の検証から選択し、属性値が特定のルールに準拠することを確認できます。

属性の検証

Validations

[+ Add validator](#)

Validator name	Config
local-date	Delete

Red Hat build of Keycloak では、すぐに使用できる各種バリデーターが提供されています。

名前	説明	設定
長さ	最小と最大長に基づいて文字列値の長さを確認します。	Min: 許可される最小の長さを定義する整数。 最大: 許容最大長を定義する整数。 trim-disabled: 検証前に値がトリミングされるかどうかを定義するブール値。
integer	値が整数であり、下限または上限の範囲内にあるかどうかを確認します。範囲が定義されていない場合、バリデーターは、値が有効な数字のみを確認します。	Min: 小さい範囲を定義する整数。 max: 上限を定義する整数。
double	値が二重で、下層または上位の範囲内であるかどうかを確認します。範囲が定義されていない場合、バリデーターは、値が有効な数字のみを確認します。	Min: 小さい範囲を定義する整数。 max: 上限を定義する整数。

名前	説明	設定
uri	値が有効な URI かどうかを確認します。	なし
pattern	値が特定の RegEx パターンと一致するかどうかを確認します。	パターン: 値の検証時に使用する RegEx パターン。 error-message: i18n バンドルのエラーメッセージのキー。設定されていない場合には、汎用メッセージが使用されます。
email	値が有効なメールアドレスの形式かどうかを確認します。	なし
local-date	レルムまたはユーザーロケールに基づいて、値が有効な形式かどうかを確認します。	なし
person-name-prohibited-characters	値がスクリプトインジェクションなどの攻撃にもう1つのバリアとして有効な人名であるかを確認します。検証は、デフォルト RegEx パターンをベースとしています。このパターンでは、文字が人名では一般的ではありません。	error-message: i18n バンドルのエラーメッセージのキー。設定されていない場合には、汎用メッセージが使用されます。
username-prohibited-characters	値がスクリプトインジェクションなどの攻撃のためにもう1つのバリアとして有効なユーザー名であるかを確認します。検証は、ユーザー名に共通の文字をブロックするデフォルトの RegEx パターンに基づいています。	error-message: i18n バンドルのエラーメッセージのキー。設定されていない場合には、汎用メッセージが使用されます。
options	値が許可された値の定義されたセットからのものであるかどうかを確認してください。選択フィールドと複数選択フィールドから入力された値を検証するのに便利です。	options: 許可された値を含む文字列の配列。

5.12.3.2.1. アノテーションの管理

フロントエンドに追加情報を渡すには、属性をレンダリングする方法を決定するアノテーションで属性を切り分けることができます。この機能は主に、Red Hat build of Keycloak テーマを拡張して、属性に関連付けられたアノテーションに基づきページを動的にレンダリングする場合に役立ちます。このメカニズムは、たとえば属性のフォーム入力フィールドを設定するために使用されます。

属性アノテーション

Validations

[+ Add validator](#)

Validator name	Config
local-date	Delete

Annotations

Annotations	Key	Value
	type	date -
	Type a key	Type a value -

[+ Add an attribute](#)

5.12.4. 属性グループの管理

属性グループ サブタブで、属性グループを作成、編集、および削除できます。属性グループを使用すると、関連属性のコンテナを定義でき、ユーザーに表示されるフォームで一緒にレンダリングできます。

属性グループリスト

Attributes			Attributes group	JSON editor
Create attributes group			1-2	< >
Name	Display name	Display description		
personallInfo	Personal Information		⋮	
addressInfo	Address Information		⋮	
			1-2	< >



注記


属性にバインドされる属性グループを削除できません。そのため、最初に属性を更新してバインディングを削除する必要があります。


新しいグループを作成するには、属性グループのリストの上部にある属性グループの作成 ボタンをクリックします。


属性グループ設定

Realm settings > User profile > Create attributes group

Create attributes group


Name * 

Display name 

Display description 

Annotations

Annotations	Key	Value
	<input type="text" value="Type a key"/>	<input type="text" value="Type a value"/>

 Add an attribute

グループを設定する際に、以下の設定を定義できます。

名前

グループの名前。

表示名

グループのユーザーフレンドリーな名前。主にユーザーに表示されるフォームをレンダリングする場合に使用されます。これは国際化をサポートするため、メッセージバンドルから値をロードできるようにします。

説明の表示

ユーザーに表示されるフォームをレンダリングする際にツールチップとして表示されるユーザーフレンドリーなテキストです。

アノテーション

このセクションでは、アノテーションを属性に関連付けることができます。アノテーションは、レンダリングの目的で追加のメタデータをフロントエンドに渡すのに役立ちます。

5.12.5. JSON 設定の使用

ユーザープロファイル設定は、明確に定義された JSON スキーマを使用して保存されます。JSON Editor サブタブで直接ユーザープロファイル設定の編集を選択できます。

JSON 設定

Attributes Attributes group JSON editor

JSON

```

1  {
2    "attributes": [
3      {
4        "name": "username",
5        "displayName": "${username}",
6        "validations": {
7          "length": {
8            "min": 3,
9            "max": 255
10         },
11         "username-prohibited-characters": {}
12       }
13     },
14     {
15       "name": "email",
16       "displayName": "${email}",
17       "validations": {
18         "email": {},
19         "length": {
20           "max": 255
21         }
22       }
23     },
24     {
25       "name": "firstName",

```

Save Revert

JSON スキーマは以下のように定義されます。

```

{
  "attributes": [
    {
      "name": "myattribute",
      "required": {
        "roles": [ "user", "admin" ],
        "scopes": [ "foo", "bar" ]
      },
      "permissions": {
        "view": [ "admin", "user" ],
        "edit": [ "admin", "user" ]
      },
      "validations": {
        "email": {},
        "length": {
          "max": 255
        }
      },
      "annotations": {
        "myannotation": "myannotation-value"
      }
    }
  ],
  "groups": [
    {
      "name": "personalInfo",
      "displayHeader": "Personal Information"
    }
  ]
}

```

```

    }
  ]
}

```

スキーマは、必要な数の属性をサポートします。

属性ごとに、**name**、オプションで **required**、**permission**、および **annotations** 設定を定義する必要があります。

5.12.5.1. 必須プロパティ

required 設定は、属性が必要であるかどうかを定義します。Red Hat build of Keycloak を使用すると、さまざまな条件に基づいて必要に応じて属性を設定できます。

required 設定が空のオブジェクトとして定義されている場合には、属性は常に必要になります。

```

{
  "attributes": [
    {
      "name": "myattribute",
      "required": {}
    }
  ]
}

```

一方、ユーザーか管理者、もしくは両方に必要な属性の作成を選択できます。また、ユーザーが Red Hat build of Keycloak で認証している際に、特定のスコープが要求される場合にのみ属性にマークを付けることができます。

ユーザーや管理者の必要に応じて属性にマークを付けるには、**roles** プロパティを以下のように設定します。

```

{
  "attributes": [
    {
      "name": "myattribute",
      "required": {
        "roles": ["user"]
      }
    }
  ]
}

```

roles プロパティは、**user** または **admin** によって属性が必要であるかどうかによって、値がユーザーまたは admin のいずれかである配列を想定します。

同様に、ユーザーの認証時に1つ以上のスコープのセットがクライアントによって要求される場合に、属性の作成を選択できます。そのため、以下のように **scopes** プロパティを使用できます。

```

{
  "attributes": [
    {
      "name": "myattribute",
      "required": {
        "scopes": ["foo"]
      }
    }
  ]
}

```

```

    }
  ]
}

```

scopes プロパティは、クライアントスコープを表す任意の文字列を持つことができます。

5.12.5.2. パーミッションプロパティ

属性レベルの **permissions** プロパティを使用すると、属性への読み取りと書き込みのパーミッションを定義できます。パーミッションは、ユーザー、管理者、またはその両方の属性に対してこれらの操作を実行するかどうかに基づいて設定されます。

```

{
  "attributes": [
    {
      "name": "myattribute",
      "permissions": {
        "view": ["admin"],
        "edit": ["user"]
      }
    }
  ]
}

```

view プロパティと **edit** プロパティはいずれも、**user** または **admin** が表示可能または管理者が管理者が編集できるかによって、値がユーザーまたは admin のいずれかであることを想定します。

edit パーミッションが付与されると、**view** パーミッションは暗黙的に付与されます。

5.12.5.3. annotations プロパティ

属性レベルの **annotation** プロパティを使用して、追加のメタデータを属性に関連付けることができます。アノテーションは、主に、ユーザープロファイル設定に基づいてユーザー属性のレンダリングに属性に関する追加情報を渡す場合に有用です。各アノテーションはキー/値のペアです。

```

{
  "attributes": [
    {
      "name": "myattribute",
      "annotations": {
        "foo": ["foo-value"],
        "bar": ["bar-value"]
      }
    }
  ]
}

```

5.12.6. 動的フォームの使用

ユーザープロファイルの主な機能の1つは、属性メタデータに基づいてユーザーに表示されるフォームを動的にレンダリングすることがあります。機能がレルムで有効にされている場合、登録や更新などのフォームは、ユーザープロファイル設定に基づいてページを動的にレンダリングする特定のテーマテンプレートを使用してレンダリングされます。

ただし、デフォルトのレンダリングメカニズムがニーズに対応する場合は、すべてのテンプレートをカスタマイズする必要はありません。テーマのカスタマイズをまだ必要な場合には、以下で参照する必要があるテンプレートを以下に示します。

テンプレート	説明
base/login/update-user-profile.ftl	更新プロファイルのページをレンダリングするテンプレート。
base/login/register-user-profile.ftl	登録ページをレンダリングするテンプレート。
base/login/idp-review-user-profile.ftl	ルールをレンダリングするテンプレートは、ブローカーを使用してユーザーをフェデレーションする際に、ユーザープロファイルをレビュー/更新するテンプレートです。
base/login/user-profile-commons.ftl	属性設定に基づいてフォームに入力フィールドをレンダリングするテンプレート。上記の3つのページテンプレートすべてから使用されます。ここで新しい入力タイプを実装できます。

デフォルトのレンダリングメカニズムでは、以下の機能を提供します。

- 属性に設定されたパーミッションに基づいて、フィールドを動的に表示します。
- 属性に設定された制約に基づいて、必須フィールドのマークを動的にレンダリングします。
- 属性に設定されたフィールド入力タイプ (テキスト、日付、数値、選択、複数選択) を動的にレンダリングします。
- 属性に設定されたパーミッションに応じて、読み取り専用フィールドを動的にレンダリングします。
- 属性に設定された順序フィールドに応じて動的に順序フィールドが続きます。
- 同じ属性グループに属する動的なグループフィールド。

5.12.6.1. 順序の属性

属性の順序は、属性リストページで属性行をドラッグアンドドロップすることで設定します。

ordering 属性

Attributes			Attributes group	JSON editor
Name	Display name	Attribute group		
☰ username	\${username}	☰		
☰ email	\${email}	☰		
☰ firstName	\${firstName}	☰		
☰ lastName	\${lastName}	☰		

このページに設定した順番は、フィールドが動的形式でレンダリングされると考慮されます。

5.12.6.2. 属性のグループ化

動的フォームがレンダリングされると、同じ属性グループに属する属性をグループ化しようとします。

動的更新プロファイルフォーム

* Required fields

Update Account Information

Email *

Personal Information

First name *

Last name *

Date of Birth

Address Information

Address *



Please specify this field.

Postal Code *



Please specify this field.



注記

属性が属性グループにリンクされている場合、属性の順番も同じグループ内の属性が一緒に閉じられるようにすることが重要になります。それ以外の場合は、グループ内の属性に連続的な順序がない場合、同じグループヘッダーが動的形式で複数回レンダリングされる可能性があります。

5.12.6.3. 属性用のフォーム入力レポートの設定

Red Hat build of Keycloak は、動的フォームや視覚化のその他の側面で属性に使用される入力タイプを設定するためのビルトインアノテーションを提供します。

利用可能なアノテーションは以下のとおりです。

名前	説明
inputType	フォーム入力フィールドのタイプ。利用可能なタイプは以下の表で説明されています。
inputHelperTextBefore	入力フィールドの前 (上) に表示されるヘルパーテキスト。ここでは、直接テキストまたは国際化パターン (<code>{i18n.key}</code>) を使用できます。テキストはページにレンダリングされるときに html エスケープされないため、ここで html タグを使用してテキストをフォーマットできますが、html 制御文字を正しくエスケープする必要もあります。
inputHelperTextAfter	入力フィールドの後にレンダリングされるヘルパーテキスト。ここでは、直接テキストまたは国際化パターン (<code>{i18n.key}</code>) を使用できます。テキストはページにレンダリングされるときに html エスケープされないため、ここで html タグを使用してテキストをフォーマットできますが、html 制御文字を正しくエスケープする必要もあります。
inputOptionsFromValidation	タイプを選択および複数選択するためのアノテーション。入力オプションの取得元となるカスタム属性検証のオプション名。詳しい説明を参照してください。
inputOptionLabelsI18nPrefix	タイプを選択および複数選択するためのアノテーション。UI でオプションをレンダリングする国際化キー接頭辞。詳しい説明を参照してください。
inputOptionLabels	タイプを選択および複数選択するためのアノテーション。オプションの UI ラベルを定義するためのオプションのマップ (直接または国際化を使用)。詳しい説明を参照してください。
inputTypePlaceholder	フィールドに適用される HTML 入力 placeholder 属性: 入力フィールドの期待値を説明する短いヒントを指定します (たとえば、サンプル値または期待される形式の簡単な説明)。ユーザーが値を入力する前に、入力フィールドに短いヒントが表示されます。

名前	説明
inputTypeSize	フィールドに適用される HTML 入力 size 属性: 単一行の入力フィールドの幅を文字数で指定します。HTML select タイプに基づくフィールドの場合は、オプションが表示された行数を指定します。使用されているテーマにある css によっては、機能しない場合があります。
inputTypeCols	フィールドに適用される HTML 入力 cols 属性: textarea タイプの幅を文字数で指定します。使用されているテーマにある css によっては、機能しない場合があります。
inputTypeRows	フィールドに適用される HTML 入力 rows 属性: textarea タイプの高さを文字数で指定します。選択フィールドの場合は、オプションが表示された行数を指定します。使用されているテーマにある css によっては、機能しない場合があります。
inputTypePattern	クライアント側の検証を提供するフィールドに適用される HTML 入力 pattern 属性: 入力フィールドの値がチェックされる正規表現を指定します。単一行入りに役立ちます。
inputTypeMaxLength	クライアント側の検証を提供するフィールドに適用される HTML 入力 maxlength 属性: 入力フィールドに入力できるテキストの最大長。テキストフィールドで役に立ちます。
inputTypeMinLength	クライアント側の検証を提供するフィールドに適用される HTML 入力 minlength 属性: 入力フィールドに入力できるテキストの最小長。テキストフィールドで役に立ちます。
inputTypeMax	クライアント側の検証を提供するフィールドに適用される HTML 入力 max 属性: 入力フィールドに入力できる最大値。数値フィールドに役立ちます。
inputTypeMin	クライアント側の検証を提供するフィールドに適用される HTML 入力 min 属性: 入力フィールドに入力できる最小値。数値フィールドに役立ちます。
inputTypeStep	フィールドに適用される HTML 入力 step 属性: 入力フィールドの有効な数値間隔を指定します。数値フィールドに役立ちます。



注記

フィールドタイプは、HTML フォームフィールドタグとそれに適用される属性を使用します。フィールドタイプは、HTML 仕様とブラウザサポートに基づいて動作します。

ビジュアルレンダリングは、使用するテーマに適用されている css スタイルにも依存します。

利用可能な **inputType** アノテーションの値:

名前	説明	使用される HTML タグ
text	単一行のテキスト入力。	入力 (input)
textarea	複数の行テキスト入力。	textarea
select	一般的な単一の選択入力。以下のオプションの設定方法の説明を参照してください。	select
select-radiobuttons	ラジオボタンをグループ化して入力を1つ選択します。以下のオプションの設定方法の説明を参照してください。	入力グループ
複数選択	一般的な複数選択入力。以下のオプションの設定方法の説明を参照してください。	select
multiselect-checkboxes	チェックボックスのグループで入力を複数選択します。以下のオプションの設定方法の説明を参照してください。	入力グループ
html5-email	HTML 5 仕様に基づくメールアドレスの単一行のテキスト入力。	入力 (input)
html5-tel	HTML 5 仕様に基づく電話番号の単一行のテキスト入力。	入力 (input)
html5-url	HTML 5 仕様に基づく URL の単一行のテキスト入力。	入力 (input)
html5-number	HTML 5 仕様に基づく数値 (step に応じて整数または浮動小数点) の1行入力。	入力 (input)
html5-range	HTML 5 仕様に基づいて入力した数字のスライダー。	入力 (input)

名前	説明	使用される HTML タグ
html5-datetime-local	HTML 5 仕様に基づく日付入力。	入力 (input)
html5-date	HTML 5 仕様に基づく日付入力。	入力 (input)
html5-month	HTML 5 仕様に基づいた月入力。	入力 (input)
html5-week	HTML 5 仕様に基づく週入力。	入力 (input)
html5-time	HTML 5 仕様に基づく時間入力。	入力 (input)

5.12.6.3.1. select フィールドおよび multiselect フィールドのオプションの定義

選択フィールドと複数選択フィールドのオプションは、属性に適用された検証から取得され、UI に表示される検証とフィールドオプションが常に一貫していることを確認します。デフォルトでは、オプションは組み込み **options** の検証から取得されます。

さまざまな方法を使用して、選択オプションと複数選択オプションに人間が判読できる適切なラベルを提供できます。最も単純なケースでは、属性値が UI ラベルと同じである場合です。この場合、追加の設定は必要ありません。

オプションの値は UI ラベルと同じです。

Validations

[+ Add validator](#)

Validator na...	Config	
options	{"options":["SW Engineer","SW architect"]}	Delete

Annotations

Annotations	Key	Value	
	<input type="text" value="inputType"/>	<input type="text" value="select"/>	-

[+ Add an attribute](#)

属性値が UI に適さない ID の種類である場合、**inputOptionLabels18nPrefix** アノテーションによって提供される単純な国際化サポートを使用できます。これは国際化キーの接頭辞を定義します。オプション値はこの接頭辞に追加されるドットになります。

i18n キー接頭辞を使用した UI ラベルの簡単な国際化

Validations

Validator name

[+ Add validator](#)

Validator n...	Config	
options	{"options":["SW Engineer","SW architect"]}	Delete

Annotations

Annotations	Key	Value	
	inputType	select	-
	inputOptionLabelsI18nPrefix	userprofile.jobtitle	-

[+ Add an attribute](#)

オプション値のローカライズされた UI ラベルテキストは、一般的なローカリゼーションメカニズムを使用して、**userprofile.jobtitle.sweng** キーおよび **userprofile.jobtitle.swarch** キーで提供する必要があります。

inputOptionLabels アノテーションを使用して、個別のオプションのラベルを指定することもできます。オプションのラベルのマップが含まれています - マップのキーはオプション値 (検証で定義) であり、マップの値は UI ラベルテキスト自体またはそのオプションの国際化パターン (**\$ {i18n.key}** など) です。



注記

User Profile **JSON Editor** を使用して map を **inputOptionLabels** アノテーションの値として入力する必要があります。

国際化なしで各オプションの直接入力されたラベルの例:

```
"attributes": [
  <...
  {
    "name": "jobTitle",
```

```

"validations": {
  "options": {
    "options": [
      "sweng",
      "swarch"
    ]
  }
},
"annotations": {
  "inputType": "select",
  "inputOptionLabels": {
    "sweng": "Software Engineer",
    "swarch": "Software Architect"
  }
}
}
...
]

```

個別オプションの国際化されたラベルの例:

```

"attributes": [
  ...
  {
    "name": "jobTitle",
    "validations": {
      "options": {
        "options": [
          "sweng",
          "swarch"
        ]
      }
    },
    "annotations": {
      "inputType": "select-radiobuttons",
      "inputOptionLabels": {
        "sweng": "${jobtitle.swengineer}",
        "swarch": "${jobtitle.swarchitect}"
      }
    }
  }
  ...
]

```

ローカライズされたテキストは、一般的なローカリゼーションメカニズムを使用して、**jobtitle.swengineer** キーと **jobtitle.swarchitect** キーで提供する必要があります。

inputOptionsFromValidation 属性アノテーションのおかげで、カスタムバリデーターを使用してオプションを提供できます。この検証には、オプションの配列を提供する **options** 設定が必要です。国際化は、組み込みの **options** 検証によって提供されるオプションの場合と同じように機能します。

カスタムバリデーターが提供するオプション

[Realm settings](#) > [User profile](#) > [Edit attribute](#)

jobTitle

General settings

Name * ⓘ

jobTitle

Display name ⓘ

job title

Attribute group ⓘ

None

Enabled when

- Always
- Scopes are requested

Required ⓘ

 Off

Jump to section

General settings

Permission

Validations

Annotations

Permission

Who can edit? ⓘ

 User Admin

Who can view? ⓘ

 User Admin

Validations

[+ Add validator](#)

Validat...	Config	
options	{"options":["SW engineer","SW architect"]}	Delete

Annotations

Annotations

Key

Value

Input type	select	-
	options	-
inputOptionsFromV...		

[+ Add an attribute](#)

Save

Cancel

5.12.7. ユーザープロファイルのコンプライアンスの強制

ユーザープロファイルが設定に準拠していることを確認するために、管理者は **VerifyProfile** の必須アクションを使用して、最終的に Red Hat build of Keycloak に対して認証する際にプロファイルの更新を強制できます。



注記

VerifyProfile アクションは **UpdateProfile** アクションに似ています。ただし、ユーザープロファイルが提供するすべての機能を活用して、ユーザープロファイルの設定により自動的にコンプライアンスが強制されます。

有効にすると、ユーザー認証時に **VerifyProfile** アクションは以下の手順を実行します。

- ユーザープロファイルが、レルムに設定されたユーザープロファイル設定に完全準拠しているかどうかを確認します。
- そうでない場合には、認証中に追加のステップを実行して、ユーザーが不足している属性や無効な属性を更新できるようにします。
- ユーザープロファイルが設定に準拠する場合は、追加のステップが実行されず、ユーザーは認証プロセスを続行します。

デフォルトでは、**VerifyProfile** アクションは無効になっています。これを有効にするには、左側のメニューの **Authentication** リンクをクリックしてから、**Required Actions** タブをクリックします。このタブで、**VerifyProfile** アクションの **Enabled** スイッチを選択します。

VerifyProfile の必須アクションの登録

Required actions	Enabled	Set as default action
☰ Configure OTP	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Terms and Conditions	<input type="checkbox"/> Off	<input type="checkbox"/> Disabled off
☰ Update Password	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Update Profile	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Verify Email	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Delete Account	<input type="checkbox"/> Off	<input type="checkbox"/> Disabled off
☰ Update User Locale	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Verify Profile	<input checked="" type="checkbox"/> On	<input type="checkbox"/> Off
☰ Webauthn Register Passwordless	<input type="checkbox"/> Off	<input type="checkbox"/> Disabled off
☰ Webauthn Register	<input type="checkbox"/> Off	<input type="checkbox"/> Disabled off

5.12.8. ユーザープロフィールへの移行

レムにユーザープロフィール機能を有効にする前に、いくつかの重要な考慮事項に注意してください。属性メタデータを管理する単一の場所を提供することで、ユーザーや管理方法に設定できる属性は非常に厳しいものです。

ユーザー管理では、管理者はユーザープロフィール設定で定義された属性のみを管理できます。ユーザープロフィール設定で定義されておらず、ユーザープロフィール設定で定義されていない他の属性にはアクセスできません。ユーザーまたは管理者へ公開する必要のあるすべてのユーザー属性で、ユーザープロフィール設定を更新することが推奨されます。

ユーザー情報をクエリーするために、ユーザー REST API にアクセスする場合も同じ推奨事項が適用されます。

LDAP_ID、**LDAP_ENTRY_DN**、**KERBEROS_PRINCIPAL** など、Red Hat build of Keycloak の内部ユーザー属性にアクセスできるようにするには、それをユーザープロフィール設定の属性として含める必要があります。表示可能な属性は管理者だけなので、管理コンソールを使用してユーザー属性を管理する場合や、ユーザー API でユーザーのクエリーを行う際には、この属性を表示可能としてマークすることが推奨されます。

そのため、レガシーテンプレート (ユーザー root 属性でハードコード化) へのカスタマイズがある場合、ユーザーに表示されるフォームをレンダリングする場合、これらのフォームを動的にレンダリングする新規テンプレートは使用されません。理想的には、テンプレートへのカスタマイズは回避し、これらの新規テンプレートが提供する動作のスタップをして、フォームを動的にレンダリングする必要があります。要件への対応が十分ではない場合は、それらをカスタマイズするか、フィードバックを提供してください。これにより、新しいテンプレートを拡張することが理にかなっているかどうかについて話し合うことができます。

5.13. RED HAT BUILD OF KEYCLOAK が収集する個人データ

デフォルトでは、Red Hat build of Keycloak は次のデータを収集します。

- ユーザーの電子メール、名字、姓など、基本的なユーザープロフィールデータ。
- ソーシャルログインの使用時にソーシャルアカウントに使用する基本的なユーザープロフィールデータとソーシャルアカウントへの参照。
- IP アドレス、オペレーティングシステム名、ブラウザ名など、監査およびセキュリティー上の目的で収集されるデバイス情報

Red Hat build of Keycloak で収集された情報は、高度なカスタマイズが可能です。カスタマイズを行う際には、以下のガイドラインが適用されます。

- 登録フォームやアカウントフォームには、誕生日、性別、国籍などのカスタムフィールドを含めることができます。管理者は、ソーシャルプロバイダーまたは LDAP などのユーザーストレージプロバイダーからデータを取得するように Red Hat build of Keycloak を設定できます。
- Red Hat build of Keycloak は、パスワード、OTP コード、WebAuthn 公開鍵などのユーザー認証情報を収集します。この情報は暗号化されてデータベースに保存されるため、Red Hat build of Keycloak 管理者には表示されません。それぞれの種類の認証情報には、パスワードのハッシュに使用されるアルゴリズムやパスワードのハッシュ化に使用されるハッシュ反復数など、管理者が見直す可能性がある、自信的なメタデータを含めることができます。
- 認可サービスと UMA サポートを有効にすると、Red Hat build of Keycloak は特定のユーザーが所有者である一部のオブジェクトに関する情報を保持できます。

第6章 ユーザーセッションの管理

ユーザーがレルムにログインすると、Red Hat build of Keycloak はユーザーごとにユーザーセッションを維持し、セッション内でユーザーが訪問した各クライアントを記憶します。レルム管理者は、各ユーザーセッションで複数のアクションを実行できます。

- レルムのログイン統計を表示します。
- アクティブユーザー、およびログイン先のユーザーを表示します。
- セッションからユーザーをログアウトします。
- トークンを取り消します。
- トークンのタイムアウトを設定します。
- セッションタイムアウトを設定します。

6.1. セッションの管理

Red Hat build of Keycloak でアクティブなクライアントとセッションの最上位ビューを表示するには、メニューから **Sessions** をクリックします。

Sessions

Sessions
Sessions are sessions of users in this realm and the clients that they access within the session.

Search session →

User	Started	Last access
admin	5/24/2022, 11:30:12 AM	5/24/2022, 11:30:12 AM

6.1.1. すべてのアクティブなセッションからサインアウトする

レルム内のすべてのユーザーをサインアウトできます。**Action** リストから、**Sign out all active sessions** を選択します。すべての SSO Cookie が無効になります。Red Hat build of Keycloak は、Red Hat build of Keycloak OIDC クライアントアダプターを使用してクライアントにログアウトイベントを通知します。アクティブなブラウザセッション内で認証を要求するクライアントは、再度ログインする必要があります。SAML などのクライアントタイプは、バックチャネルログアウト要求を受信しません。



注記

Sign out all active sessions をクリックしても、未処理のアクセストークンは取り消されません。未処理のトークンは自然に期限切れになる必要があります。Red Hat build of Keycloak OIDC クライアントアダプターを使用しているクライアントの場合、[revocation policy](#) をプッシュしてトークンを取り消すことができますが、これは他のアダプターでは機能しません。

6.1.2. クライアントセッションの表示

手順

1. メニューで **Clients** をクリックします。
2. **セッション** タブをクリックします。
3. クライアントをクリックすると、そのクライアントのセッションが表示されます。

クライアントセッション

The screenshot shows the Keycloak Admin Console interface. On the left is a navigation menu with 'Clients' selected. The main content area shows 'Client details' for 'security-admin-console-v2'. The 'Sessions' tab is active, displaying a table of sessions for the user 'admin'.

User	Started	Last access
admin	5/24/2022, 11:30:12 AM	5/24/2022, 11:36:24 AM

6.1.3. ユーザーセッションの表示

手順

1. メニューの **Users** をクリックします。
2. **セッション** タブをクリックします。
3. ユーザーをクリックすると、そのユーザーのセッションが表示されます。

ユーザーセッション

The screenshot shows the Keycloak Admin Console interface. On the left is a navigation menu with 'Users' selected. The main content area shows 'User details' for 'admin'. The 'Sessions' tab is active, displaying a table of sessions for the user 'admin'.

Started	Last access	Clients
5/24/2022, 11:30:12 AM	5/24/2022, 11:51:25 AM	security-admin-console-v2

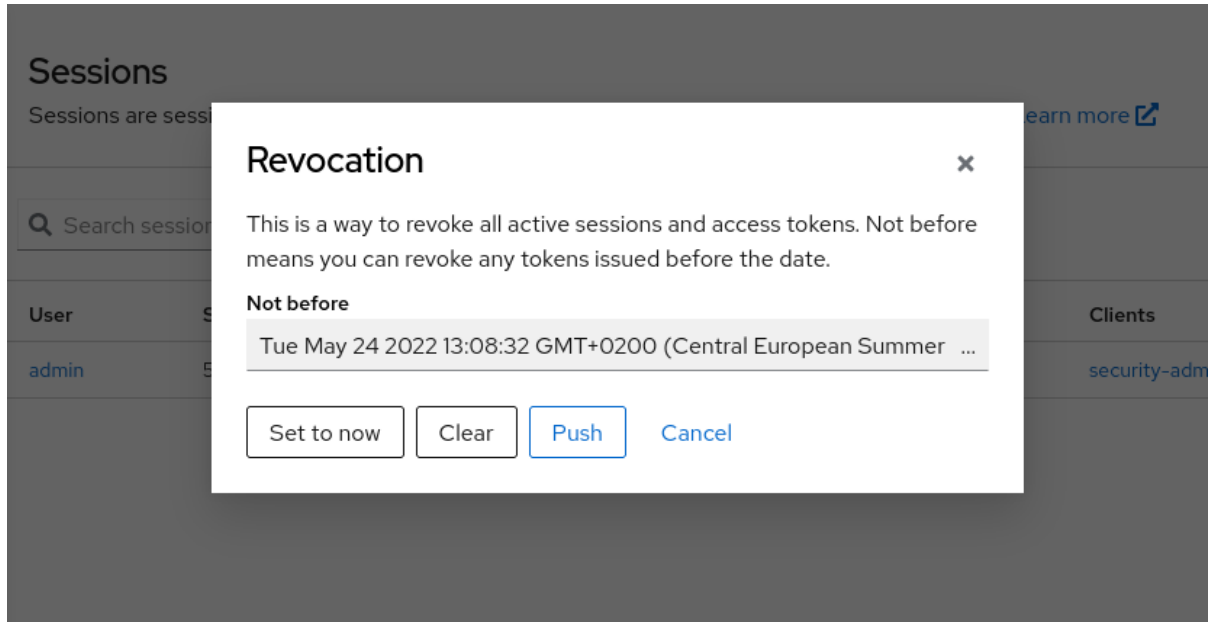
6.2. アクティブなセッションの取り消し

システムが侵害された場合は、アクティブなセッションとアクセストークンをすべて取り消すことができます。

手順

1. メニューの **Sessions** をクリックします。
2. **Actions** リストから **Revocation** を選択します。

取り消し



3. このコンソールを使用して、日時を指定します。その日時より前に発行されたセッションまたはトークンは無効になります。
 - **Set to now** をクリックして、ポリシーを現在の日時に設定します。
 - **Push** をクリックして、Red Hat build of Keycloak OIDC クライアントアダプターを使用してこの失効ポリシーを登録済みの OIDC クライアントにプッシュします。

6.3. セッションおよびトークンのタイムアウト

Red Hat build of Keycloak には、**Realm settings** メニューの **Sessions** タブと **Tokens** タブにセッション、Cookie、およびトークンのタイムアウトの制御が含まれています。

Sessions タブ

Master Enabled Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

◀ gin Email Themes Keys Events Localization Security defenses **Sessions** ▶

SSO Session Settings

SSO Session Idle ⓘ ▾

SSO Session Max [?]	<input type="text" value="10"/>	Hours <input type="button" value="▼"/>
SSO Session Idle Remember Me [?]	<input type="text" value="0"/>	Minutes <input type="button" value="▼"/>
SSO Session Max Remember Me [?]	<input type="text" value="0"/>	Minutes <input type="button" value="▼"/>

Client session settings

Client Session Idle [?]	<input type="text" value="0"/>	Minutes <input type="button" value="▼"/>
Client Session Max [?]	<input type="text" value="0"/>	Minutes <input type="button" value="▼"/>

Offline session settings

Offline Session Idle [?]	<input type="text" value="30"/>	Days <input type="button" value="▼"/>
Offline Session Max Limited [?]	<input type="checkbox"/> Disabled	

Login settings

Login timeout [?]	<input type="text" value="30"/>	Minutes <input type="button" value="▼"/>
Login action timeout [?]	<input type="text" value="5"/>	Minutes <input type="button" value="▼"/>

[Revert](#)

設定

説明

設定	説明
SSO Session Idle	この設定は OIDC クライアントのみを対象としています。ユーザーがこのタイムアウトよりも非アクティブである場合は、ユーザーセッションが無効になります。このタイムアウト値は、クライアントが認証を要求するか、更新トークン要求を送信するときリセットされます。Red Hat build of Keycloak は、セッションの無効化が有効になる前に、アイドルタイムアウトに期間を追加します。このセクションで後述する 注記 を参照してください。
SSO Session Max	ユーザーセッションが期限切れになるまでの最大時間。
SSO Session Idle Remember Me	この設定は標準の SSO セッション ID 設定に似ていますが、 Remember Me が有効になっているログインに固有の設定です。ユーザーは、ログイン時に Remember Me をクリックすると、セッションのアイドルタイムアウトが長く指定できます。この設定は任意の設定であり、その値がゼロより大きい場合、SSO Session Idle 設定と同じアイドルタイムアウトを使用します。
SSO Session Max Remember Me	この設定は標準の SSO セッション Max と似ていますが、 Remember Me ログインに固有です。ユーザーは、ログイン時に Remember Me をクリックするとセッションが長く指定できます。この設定は任意の設定であり、その値がゼロより大きい場合、SSO Session Max 設定と同じセッションライフスパンを使用します。
Client Session Idle	クライアントセッションのアイドルタイムアウト。このタイムアウトよりも長くユーザーの非アクティブ状態が続く場合、クライアントセッションは無効になり、更新トークンの要求に基づきアイドルタイムアウトが増加します。この設定が、固有の一般的な SSO ユーザーセッションに影響することはありません。SSO ユーザーセッションは 0 個以上のクライアントセッションの親であり、ユーザーがログインするクライアントアプリごとに1つのクライアントセッションが作成されることに注意してください。この値には、 SSO Session Idle よりも短いアイドルタイムアウトを指定する必要があります。ユーザーは、 Advanced Settings クライアントタブで、個々のクライアントに対してこれをオーバーライドできます。この設定は任意の設定であり、ゼロに設定すると SSO セッション ID 設定で同じアイドルタイムアウトを使用します。


設定	説明
Client Session Max	クライアントセッションの最長時間、および更新トークンの有効期限が切れて無効になるまでの時間。前のオプションと同様に、この設定はSSOユーザーセッションに影響しません。また、 SSO Session Max よりも短い値を指定する必要があります。ユーザーは、 Advanced Settings クライアントタブで、個々のクライアントに対してこれをオーバーライドできます。この設定はオプションの設定であり、ゼロに設定するとSSO Session Max の設定と同じアイドルタイムアウトが使用されます。
Offline Session Idle	この設定は オフラインアクセス 用です。Red Hat build of Keycloak がオフライントークンを取り消すまでに、セッションがアイドル状態に留まる時間。Red Hat build of Keycloak は、セッションの有効化が有効になる前に、アイドルタイムアウトに期間を追加します。このセクションで後述する 注記 を参照してください。
Offline Session Max Limited	この設定は オフラインアクセス 用です。このフラグが Enabled の場合、ユーザーアクティビティーにかかわらず、Offline Session Max がアクティブ状態を維持する最大時間を制御できます。フラグが Disabled の場合、オフラインセッションはライフスパンによって期限切れになることはなく、アイドル状態によってのみ期限切れになります。このオプションをアクティブにすると、 Offline Session Max (レルムレベルのグローバルオプション) と Client Offline Session Max (詳細 Advanced Settings タブにある特定のクライアントレベルオプション) を設定できます。
Offline Session Max	この設定は オフラインアクセス 用であり、Red Hat build of Keycloak が対応するオフライントークンを取り消すまでの最大時間です。このオプションは、ユーザーアクティビティーに関係なく、オフライントークンがアクティブな状態のままになる最大期間を制御します。
Login timeout	ログインにかかる合計時間。認証にかかる時間よりも長い場合は、ユーザーは認証プロセスを再度開始する必要があります。
Login action timeout	Maximum time ユーザーは、認証プロセス中に1つのページで費やすことができます。

tokens タブ

Master

 Enabled

Action ▼

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#) 

◀ Localization Security defenses Sessions **Tokens** Client policies User profile ▶

General

Default Signature
Algorithm 

RS256 ▼

Refresh tokens

Revoke Refresh Token Disabled



Access tokens

Access Token Lifespan

1

Minutes ▼



It is recommended for this value to be shorter than the SSO session idle timeout: 30 minutes

Access Token Lifespan

15

Minutes ▼

For Implicit Flow 

Client Login Timeout

1

Minutes ▼



Action tokens

User-Initiated Action
Lifespan 

5

Minutes ▼

Default Admin-
Initiated Action
Lifespan 

12

Hours ▼

Override Action Tokens

Email Verification

Minutes ▼

IdP account email
verification

Minutes ▼

Forgot password

Minutes ▼

Execute actions

Minutes ▼

Save

Revert

設定	説明
Default Signature Algorithm	レムにトークンを割り当てるために使用されるデフォルトのアルゴリズム。
Revoke Refresh Token	Enabled の場合、Red Hat build of Keycloak は更新トークンを取り消し、クライアントが必ず使用しなければならない別のトークンを発行します。このアクションは、更新トークンフローを実行する OIDC クライアントに適用されます。
Access Token Lifespan	Red Hat build of Keycloak が OIDC アクセストークンを作成するとき、この値でトークンの有効期間を制御します。
Access Token Lifespan For Implicit Flow	Implicit Flow では、Red Hat build of Keycloak は更新トークンを提供しません。Implicit Flow によって作成されるアクセストークンに別のタイムアウトが存在します。
Client login timeout	クライアントが OIDC で Authorization Code Flow を終了するまでの最大時間。
User-Initiated Action Lifespan	ユーザーのアクションパーミッションの有効期限が切れるまでの最大時間。ユーザーが通常、自己作成のアクションに迅速に対応するので、この値を短くしてください。
Default Admin-Initiated Action Lifespan	管理者によってユーザーに送信されるアクションパーミッションの最大時間。この値を長く維持して、管理者がオフラインユーザーに電子メールを送信できるようにします。管理者は、トークンを発行する前にデフォルトのタイムアウトを上書きできます。
Email Verification	電子メール検証の独立したタイムアウトを指定します。

設定	説明
IdP account email verification	IdP アカウントの電子メール検証の独立したタイムアウトを指定します。
Forgot password	パスワードを忘れた場合の独立したタイムアウトを指定します。
Execute actions	実行アクションの独立したタイムアウトを指定します。



注記

アイドルタイムアウトの場合は、セッションがアクティブである期間が2分のウィンドウになります。たとえば、タイムアウトが30分に設定されている場合、セッションの有効期限が切れるまでに32分になります。

このアクションは、クラスター間および複数のデータセンター環境で必要です。この場合、トークンは有効期限前に1つのクラスターノードで短期間に更新され、他のクラスターノードは更新されたノードから正常な更新についてのメッセージを受信していないため、セッションが期限切れと誤って考慮されます。

6.4. オフラインアクセス

オフラインアクセス ログイン時に、クライアントアプリケーションは更新トークンではなくオフライントークンを要求します。クライアントアプリケーションは、このオフライントークンを保存し、ユーザーがログアウトした場合に今後のログインに使用できます。このアクションは、ユーザーがオンラインにない場合でも、アプリケーションがユーザーの代わりにオフライン操作を実行する必要がある場合に便利です。たとえば、通常のデータバックアップです。

クライアントアプリケーションは、ストレージでオフライントークンを永続化し、これを使用して Red Hat build of Keycloak サーバーから新しいアクセストークンを取得します。

更新トークンとオフライントークンの相違点は、オフライントークンの期限が切れず、**SSO Session Idle timeout** および **SSO Session Max lifespan** の対象でないことです。オフライントークンは、ユーザーのログアウトまたはサーバーの再起動後に有効になります。オフライントークンは、少なくとも30日に1回の更新トークンアクション、または **Offline Session Idle** の値に使用する必要があります。

Offline Session Max Limited を有効にすると、トークンの更新アクションにオフライントークンを使用した場合でも、オフライントークンは60日後に期限切れになります。この値 **Offline Session Max** は、管理コンソールで変更できます。

オフラインアクセスを使用する場合、クライアントのアイドル状態と最大タイムアウトは **クライアントレベル** でオーバーライドできます。クライアントの **Advanced Settings** タブにある **Client Offline Session Idle** オプションと **Client Offline Session Max** オプションを使用すると、特定のアプリケーションのオフラインタイムアウトを短くすることができます。クライアントセッション値は更新トークンの有効期限も制御しますが、グローバルオフラインユーザー SSO セッションに影響を与えることはありません。**Client Offline Session Max** オプションは、レルムレベルで **Offline Session Max Limited** が **Enabled** の場合にのみクライアントで評価されます。

Revoke Refresh Token オプションを有効にすると、各オフライントークンを1回だけ使用できます。更新後、前の1つではなく、更新応答から新しいオフライントークンを保存する必要があります。

ユーザーは [User Account Console](#) で、Red Hat build of Keycloak が付与したオフライントークンの表示と取り消しを行えます。管理者は、**Consents** タブで管理コンソールの個々のユーザーのオフライントークンを取り消すことができます。管理者は、各クライアントの **Offline Access** タブで発行されたオフライントークンすべてを表示できます。管理者は、[失効ポリシー](#) を設定して、オフライントークンを取り消すことができます。

オフライントークンを発行するには、ユーザーは realm-level **offline_access** ロールのロールマッピングが必要です。クライアントには、そのロールをスコープで割り当てる必要もあります。クライアントは、**offline_access** クライアント スコープを **Optional client scope** としてロールに追加し、デフォルトでは実行する必要があります。

クライアントは、認可要求を Red Hat build of Keycloak に送信するときに **scope=offline_access** パラメーターを追加することで、オフライントークンを要求できます。Red Hat build of Keycloak OIDC クライアントアダプターは、アプリケーションのセキュアな URL (例: `http://localhost:8080/customer-portal/secured?scope=offline_access`) にアクセスするために使用すると、このパラメーターを自動的に追加します。認証要求の本文に **scope=offline_access** が含まれている場合、Direct Access Grant および Service Accounts はオフライントークンをサポートします。

オフラインセッションは、Infinispan キャッシュのほかに、データベースにも保存されます。Red Hat build of Keycloak サーバーが再起動するか、オフラインセッションが Infinispan キャッシュからエビクトする場合でも、データベースでは引き続き利用できます。オフラインセッションへのアクセスを試みると、データベースからセッションがロードされ、Infinispan キャッシュにもインポートされます。メモリ要件を削減するために、インポートされたオフラインセッションの有効期間を短縮する設定オプションを導入しました。このようなセッションは、指定された有効期間が経過すると Infinispan キャッシュからエビクトされますが、データベースでは引き続き利用可能です。これにより、特にオフラインセッションが多数あるデプロイメントでは、メモリ消費量が削減されます。現在、オフラインセッションの有効期間のオーバーライドはデフォルトで無効になっています。オフラインユーザーセッションの有効期間のオーバーライドを指定するには、次のパラメーターを使用して Red Hat build of Keycloak サーバーを起動します。

```
--spi-user-sessions-infinispan-offline-session-cache-entry-lifespan-override=<lifespan-in-seconds>
```

オフラインクライアントセッションの場合も同様です。

```
--spi-user-sessions-infinispan-offline-client-session-cache-entry-lifespan-override=<lifespan-in-seconds>
```

6.5. オフラインセッションの事前読み込み

オフラインセッションは、Infinispan キャッシュに加えてデータベースにも保存されます。そのため、サーバーの再起動後もオフラインセッションを使用できます。デフォルトでは、オフラインセッションは、サーバーの起動時にデータベースから Infinispan キャッシュにプリロードされません。これは、プリロードするオフラインセッションが多数ある場合、この方法には欠点があるためです。サーバーの起動時間が大幅に遅くなることがあります。したがって、オフラインセッションはデフォルトでデータベースから遅延フェッチされます。

ただし、Red Hat build of Keycloakは、サーバー起動時にデータベースから Infinispan キャッシュにオフラインセッションを事前にロードするように設定できます。これは、**userSessions** SPI の **preloadOfflineSessionsFromDatabase** プロパティを **true** に設定すると実現できます。

以下の例は、オフラインセッションを事前ロードする設定方法を示しています。

```
bin/kc.[sh|bat] start --spi-user-sessions-infinispan-preload-offline-sessions-from-database=true
```

6.6. 一時的なセッション

Red Hat build of Keycloak で一時セッションを実行できます。一時セッションを使用する場合、Red Hat build of Keycloak は認証の成功後にユーザーセッションを作成しません。Red Hat build of Keycloak は、ユーザーを正常に認証する現在の要求のスコープに対して一時セッションを作成します。Red Hat build of Keycloak は、認証後に一時セッションを使用して [プロトコルマッパー](#) を実行できます。

一時的なセッションでは、クライアントアプリケーションはトークンを更新したり、トークンイントロスペクションしたり、特定のセッションを検証したりすることはできません。これらのアクションは不要な場合があるため、ユーザーセッションを永続化するための追加のリソースの使用を避けることができます。このセッションは、パフォーマンス、メモリー、ネットワーク通信(クラスターおよびクロスデータセンター環境)のリソースを保存します。

第7章 ロールとグループを使用した権限の割り当て

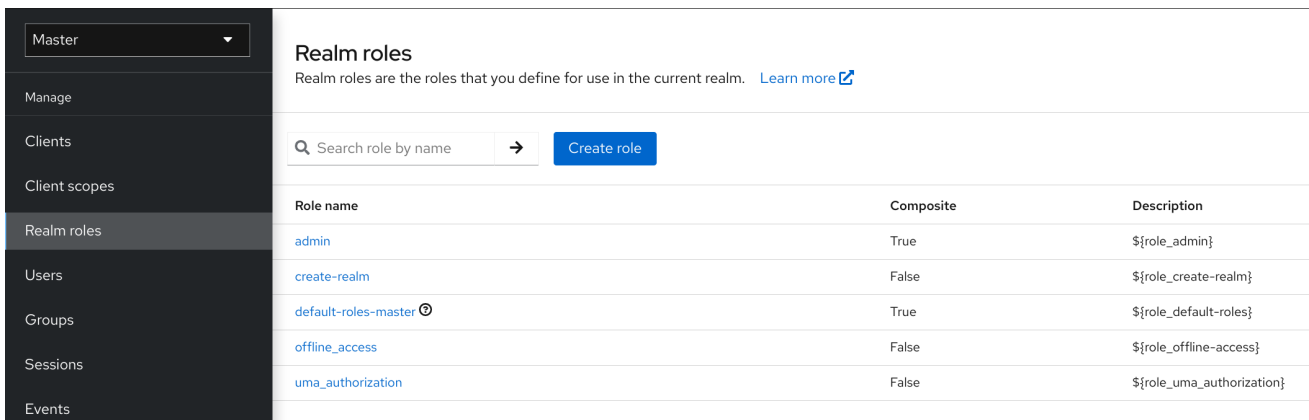
ロールとグループには同様の目的があります。この目的は、ユーザーにアプリケーションを使用するためのアクセスと権限を与えることです。グループは、ロールと属性を適用するユーザーの集まりです。ロールは、特定のアプリケーションのパーミッションおよびアクセス制御を定義します。

通常、ロールはユーザーの1種別に適用されます。たとえば、組織には **admin**、**user**、**manager**、**employee** のロールが含まれます。アプリケーションは、ロールにアクセスとパーミッションを割り当て、ユーザーが同じアクセスとパーミッションを持つように複数のユーザーを割り当てることができます。たとえば、管理コンソールには、管理コンソールの異なる部分にアクセスするための権限を付与するロールがあります。

また、ロールのグローバル名前空間があり、各クライアントにはロールを定義する独自の専用の名前空間もあります。

7.1. レルムロールの作成

レルムレベルのロールは、ロールを定義する namespace です。ロールのリストを表示するには、メニューで **Realm Roles** をクリックします。

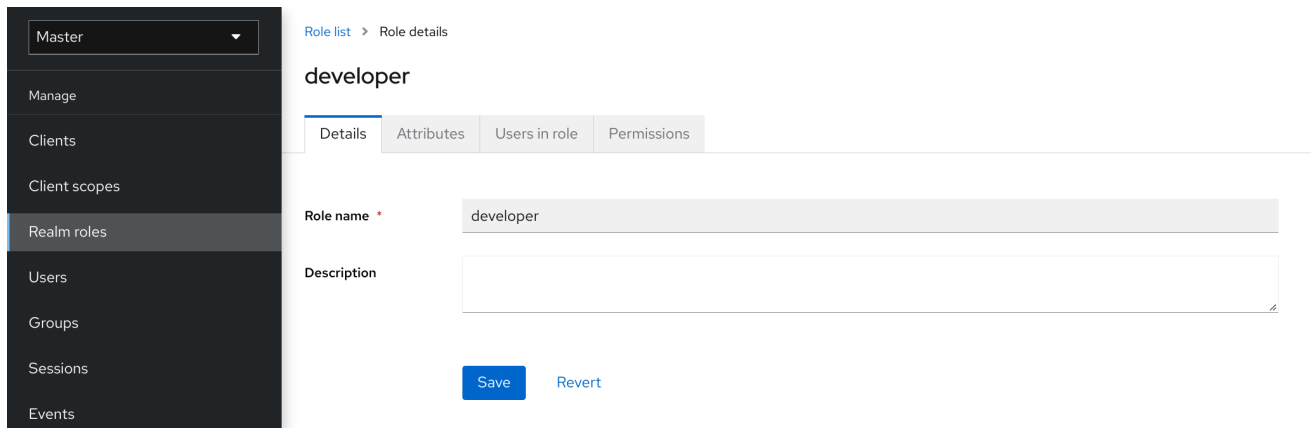


Role name	Composite	Description
admin	True	`\${role_admin}`
create-realm	False	`\${role_create-realm}`
default-roles-master	True	`\${role_default-roles}`
offline_access	False	`\${role_offline-access}`
uma_authorization	False	`\${role_uma_authorization}`

手順

1. **ロールの作成** をクリックします。
2. **Role Name** を入力します。
3. **Description** を入力します。
4. **Save** をクリックします。

ロールの追加



description フィールドは `${var-name}` 文字列で置換変数を指定することでローカライズできます。ローカライズされた値は themes プロパティファイル内のテーマに設定されています。詳細は、[サーバー開発者ガイド](#) を参照してください。

7.2. クライアントロール

クライアントロールはクライアント専用の namespace です。各クライアントは独自の名前空間を取得します。クライアントロールは、各クライアントの **Roles** タブで管理されます。この UI は、レルムレベルのロールの場合と同じように対話します。

7.3. ロールの複合ロールへの変換

レルムまたはクライアントレベルのロールは **複合ロール** になります。**複合ロール** は、1つ以上の追加ロールに関連付けられたロールです。複合ロールがユーザーにマップされると、ユーザーは複合ロールに関連付けられたロールを取得します。この継承は再帰的であるため、ユーザーは複合の複合も継承します。ただし、複合ロールが使用されないことを推奨します。

手順

1. メニューで **Realm Roles** をクリックします。
2. 変換するロールをクリックします。
3. **Action** リストから、**Add associated roles** を選択します。

複合ロール

Add roles to developer ×

Filter by roles ▾
Search role by name
→

1-6 ▾
<
>

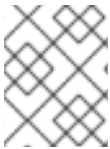
<input type="checkbox"/> Role name	Description
<input type="checkbox"/> admin	<code>role_admin</code>
<input type="checkbox"/> create-realm	<code>role_create-realm</code>
<input type="checkbox"/> default-roles-master	<code>role_default-roles</code>
<input checked="" type="checkbox"/> employee	
<input type="checkbox"/> offline_access	<code>role_offline-access</code>
<input type="checkbox"/> uma_authorization	<code>role_uma_authorization</code>

1-6 ▾
<
>

Add
Cancel

ロール選択 UI がページに表示され、レルムレベルとクライアントレベルのロールを、作成する複合ロールに関連付けることができます。

この例では、**従業員の**レルムレベルロールが **開発者の**複合ロールに関連付けられます。 **developer** ロールを持つユーザーは、 **employee** ロールも継承します。

**注記**

トークンと SAML アサーションの作成時に、複合には、クライアントに送信された認証応答の要求およびアサーションにも関連付けられたロールが追加されます。

7.4. ロールマッピングの割り当て

ユーザーの **Role Mappings** タブからユーザーにロールマッピングを割り当てることができます。

手順

1. メニューの **Users** をクリックします。
2. ロールマッピングを実行するユーザーをクリックします。
3. **Role mappings** タブをクリックします。
4. **Assign role** をクリックします。
5. ダイアログからユーザーに割り当てるロールを選択します。
6. **Assign** をクリックします。

ロールマッピング

Users > User details

johndoe

Details Attributes Credentials **Role mapping** Groups Consents Identity provider links

Search by name → Hide inherited roles **Assign role** Unassign

<input type="checkbox"/>	Name	Inherited	Description
<input type="checkbox"/>	default-roles-master	False	\${role_default-roles}
<input type="checkbox"/>	uma_authorization	True	\${role_uma_authorization}
<input type="checkbox"/>	offline_access	True	\${role_offline-access}
<input type="checkbox"/>	account view-profile	True	\${role_view-profile}
<input type="checkbox"/>	account manage-account-links	True	\${role_manage-account-links}
<input type="checkbox"/>	account manage-account	True	\${role_manage-account}

上記の例では、複合ロールの開発者を **ユーザー** に割り当てます。そのロールは [複合ロール](#) トピックで作成されました。

有効なロールマッピング

Users > User details

johndoe

Details Attributes Credentials **Role mapping** Groups Consents Identity provider links

Search by name → Hide inherited roles **Assign role** Unassign

<input type="checkbox"/>	Name	Inherited	Description
<input type="checkbox"/>	employee	True	-
<input type="checkbox"/>	developer	False	-
<input type="checkbox"/>	default-roles-master	False	\${role_default-roles}
<input type="checkbox"/>	uma_authorization	True	\${role_uma_authorization}
<input type="checkbox"/>	offline_access	True	\${role_offline-access}
<input type="checkbox"/>	account view-profile	True	\${role_view-profile}
<input type="checkbox"/>	account manage-account-links	True	\${role_manage-account-links}
<input type="checkbox"/>	account manage-account	True	\${role_manage-account}

developer ロールが割り当てられると、**developer** コンポジットに関連付けられた **employee** ロールが **Inherited "True"** で表示されます。**inherited** ロールは、ユーザーに明示的に割り当てられたロール、およびコンポジットから継承されたロールです。

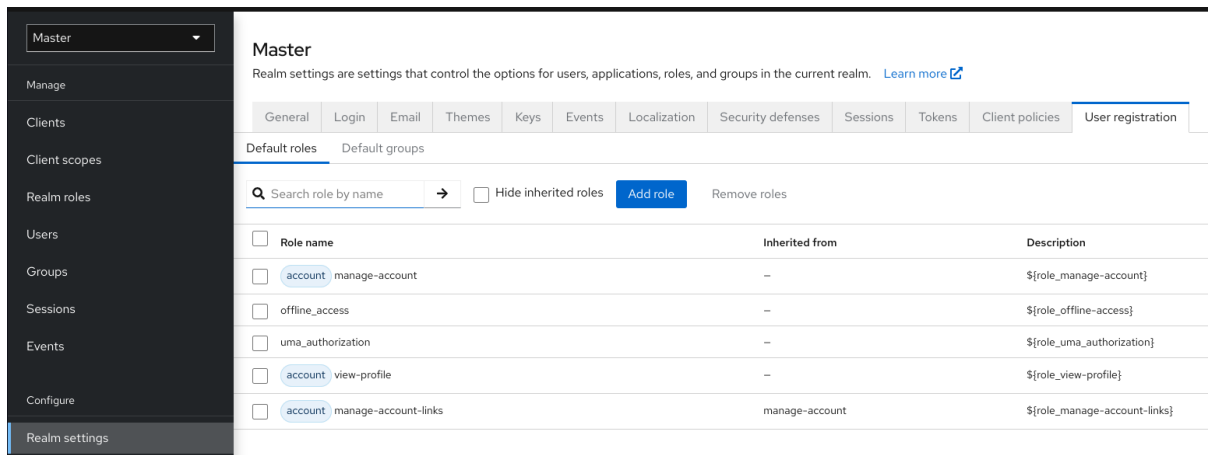
7.5. デフォルトロールの使用

[Identity Brokering](#) を介してユーザーが作成またはインポートされたときに、デフォルトのロールを使用して、ユーザーロールマッピングを自動的に割り当てます。

手順

1. メニューで **Realm Settings** をクリックします。
2. **User registration** タブをクリックします。

デフォルトロール



このスクリーンショットは、一部の **デフォルトロール** がすでに存在していることを示しています。

7.6. ロールマッピングのマッピング

OIDC アクセストークンまたは SAML アサーションの作成時に、ユーザーロールマッピングはトークンまたはアサーション内で要求されます。アプリケーションはこれらの要求を使用して、アプリケーションが制御するリソースにアクセスの決定を行います。Red Hat build of Keycloak は、アクセストークンとアプリケーションを再度使用して、リモートで保護された REST サービスを呼び出します。ただし、これらのトークンには関連するリスクがあります。攻撃者はこれらのトークンを取得し、パーミッションを使用してネットワークを危険にさらすことができます。この状況を防ぐには、**Role Scope Mappings** を使用します。

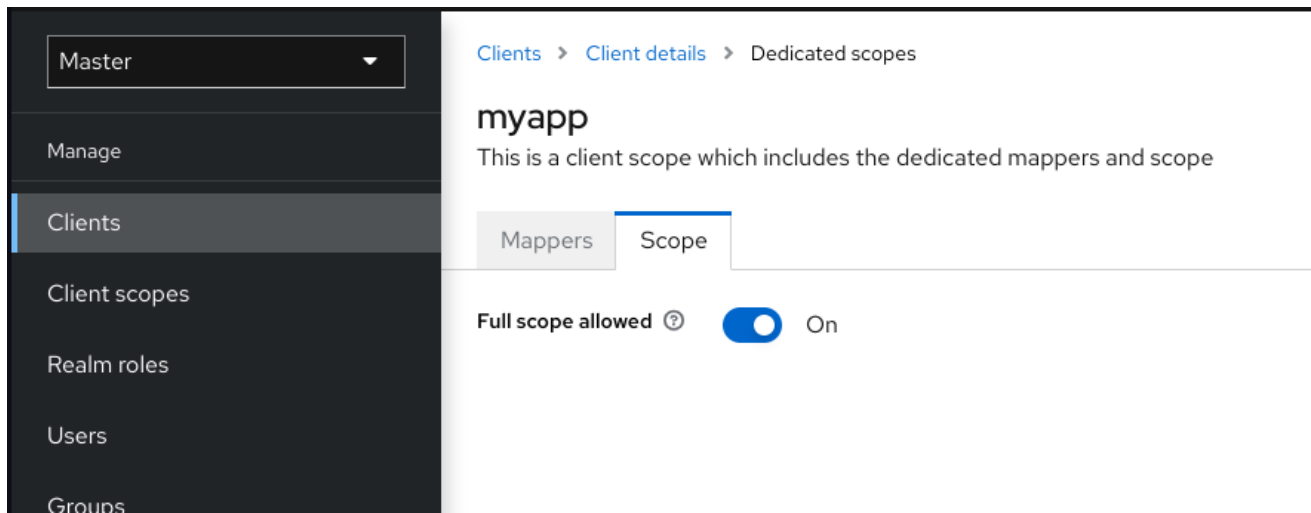
ロールスコープマッピングは、アクセストークン内に宣言されたロールを制限します。クライアントがユーザー認証を要求すると、受信するアクセストークンには、クライアントの範囲に対して明示的に指定されるロールマッピングのみが含まれます。その結果、すべてのユーザーのパーミッションにクライアントアクセスを付与するのではなく、個々のアクセストークンのパーミッションを制限ようになります。

デフォルトでは、各クライアントはユーザーのすべてのロールマッピングを取得します。クライアントのロールマッピングを表示できます。

手順

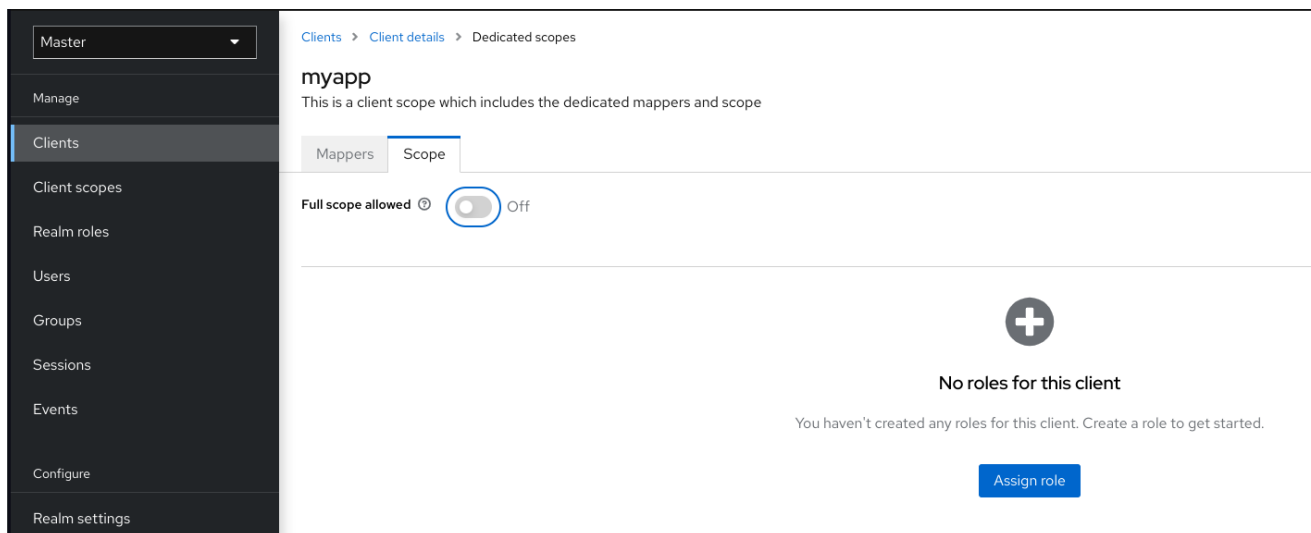
1. メニューで **Clients** をクリックします。
2. クライアントをクリックして詳細に移動します。
3. **Client Scopes** タブをクリックします。
4. **Dedicated scope and mappers for this client**の行のリンクをクリックします
5. **Scope** タブをクリックします。

フルサポート



デフォルトでは、スコープの有効なロールはすべてレルムで宣言されるロールです。このデフォルトの動作を変更するには、**Full Scope Allowed**を **OFF** に切り替え、クライアントごとに必要な特定のロールを宣言します。[client scopes](#) を使用して、一連のクライアントに対して同じロールスコープマッピングを定義することもできます。

部分的なスコープ

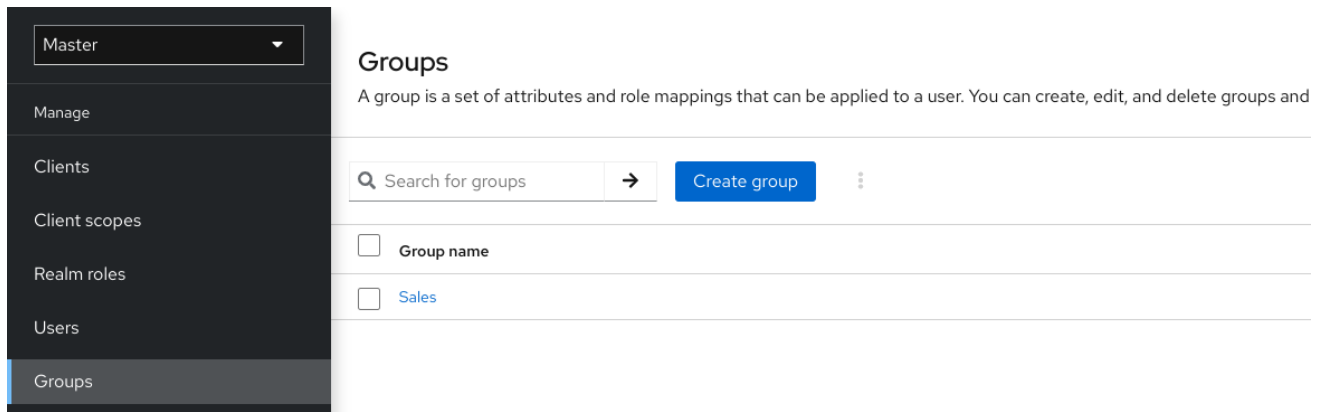


7.7. グループ

Red Hat build of Keycloak のグループは、各ユーザーの共通の属性セットとロールマッピングを管理します。ユーザーは任意の数のグループのメンバーとなり、各グループに割り当てられた属性およびロールマッピングを継承できます。

グループを管理するには、メニューで **Groups** をクリックします。

グループ



グループは階層です。グループには複数のサブグループを指定できますが、グループには親を1つだけ指定できます。サブグループは、親から属性とロールマッピングを継承します。ユーザーは、親から属性とロールマッピングも継承します。

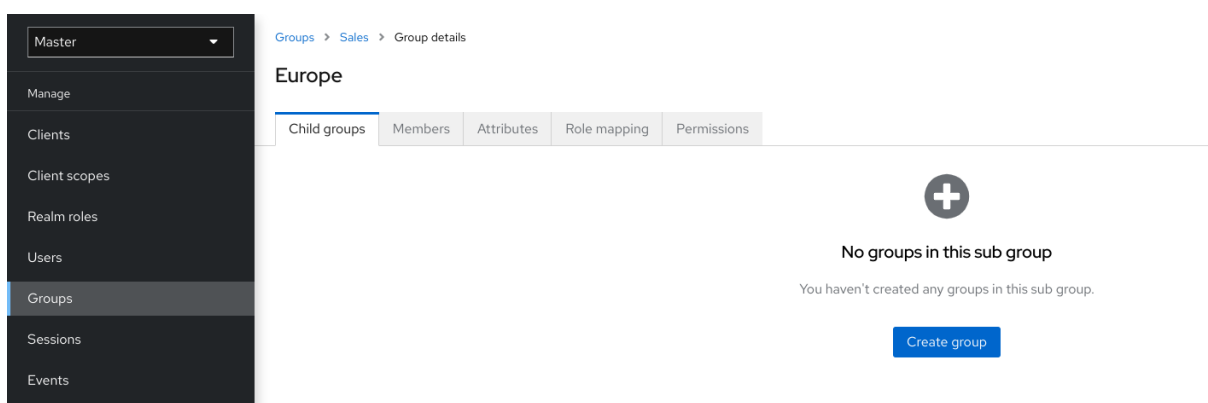
親グループと子グループがあり、子グループにのみ所属するユーザーがある場合は、子グループのユーザーは、親グループと子グループの両方の属性とロールマッピングを継承します。

以下の例には、トップレベル **セールス** グループと子の **北アメリカ** のサブグループが含まれます。

グループを追加するには、以下を実行します。

1. グループをクリックします。
2. **Create Group** をクリックします。
3. グループ名を入力します。
4. **Create** をクリックします。
5. グループ名をクリックします。
グループ管理ページが表示されます。

グループ



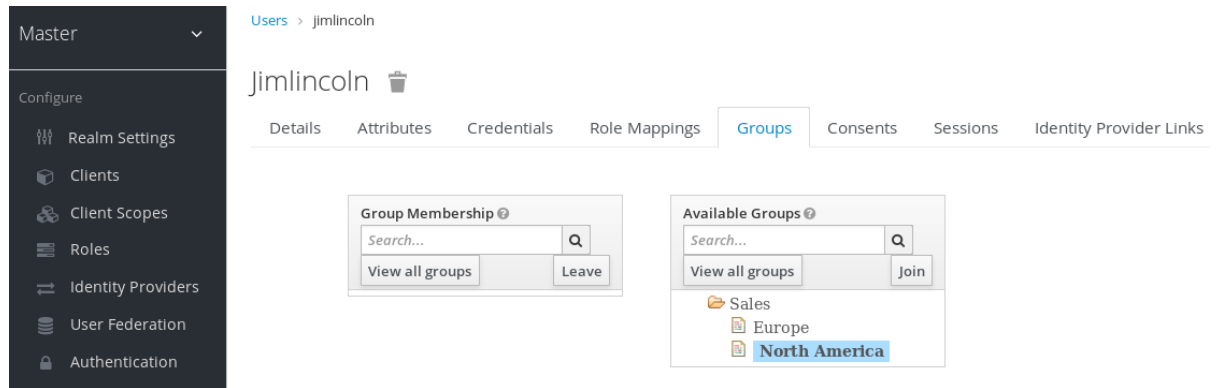
定義する属性およびロールマッピングは、グループのメンバーであるグループおよびユーザーによって継承されます。

ユーザーをグループに追加します。

1. メニューの **Users** をクリックします。
2. ロールマッピングを実行するユーザーをクリックします。ユーザーが表示されない場合は、**View all users** をクリックします。

3. Groups をクリックします。

ユーザーグループ



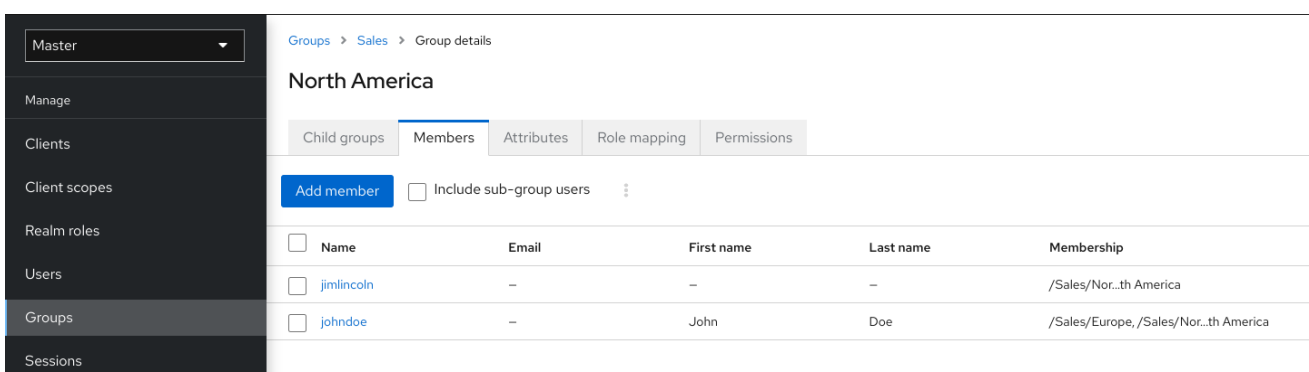
4. **Join Group** をクリックします。
5. ダイアログからグループを選択します。
6. **Available Groups** ツリーからグループを選択します。
7. **Join** をクリックします。

ユーザーからグループを削除するには、以下を実行します。

1. メニューの **Users** をクリックします。
2. グループから削除するユーザーをクリックします。
3. グループテーブルの行で **Leave** をクリックします。

この例では、ユーザー **jimlincoln** は **North America** グループにあります。グループの **Members** タブの下に **jimlincoln** が表示されます。

グループメンバーシップ



7.7.1. ロールと比べているグループ

グループとロールには、いくつかの類似点と違いがあります。Red Hat build of Keycloak では、グループはロールと属性を適用するユーザーのコレクションです。ロールは、ユーザーとアプリケーションのタイプを定義し、パーミッションをロールに割り当てます。

複合ロール は、同じ機能を提供するため、グループに似ています。その違いは概念的で、複合ロールは、パーミッションモデルをサービスやアプリケーションのセットに適用します。複合ロールを使用してアプリケーションおよびサービスを管理します。

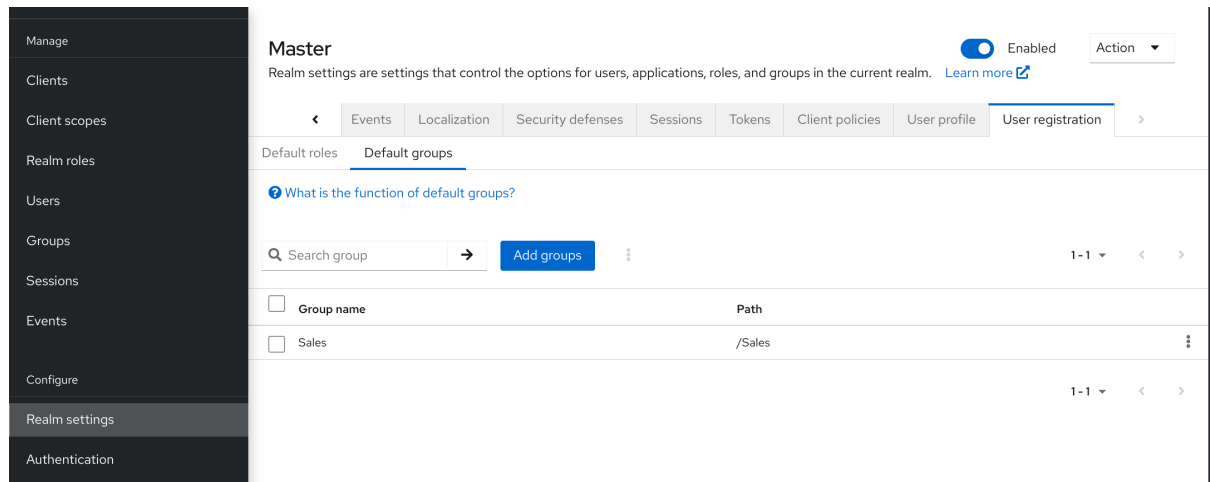
グループは、組織におけるユーザーとロールのコレクションに重点を置きます。グループを使用してユーザーを管理します。

7.7.2. デフォルトグループの使用

[Identity Brokering](#) を介して作成またはインポートされたユーザーにグループメンバーシップを自動的に割り当てるには、デフォルトのグループを使用します。

1. メニューで **Realm Settings** をクリックします。
2. **User registration** タブをクリックします。
3. **Default Groups** タブをクリックします。

デフォルトグループ



このスクリーンショットは、一部の **デフォルトグループ** がすでに存在していることを示しています。

第8章 認証の設定

本章では、複数の認証トピックについて説明します。以下のトピックを以下に示します。

- 厳密なパスワードおよびワンタイムパスワード (OTP) ポリシーを強制します。
- 異なる認証情報タイプの管理
- Kerberos でログインします。
- 組み込み認証情報タイプを無効にして有効化します。

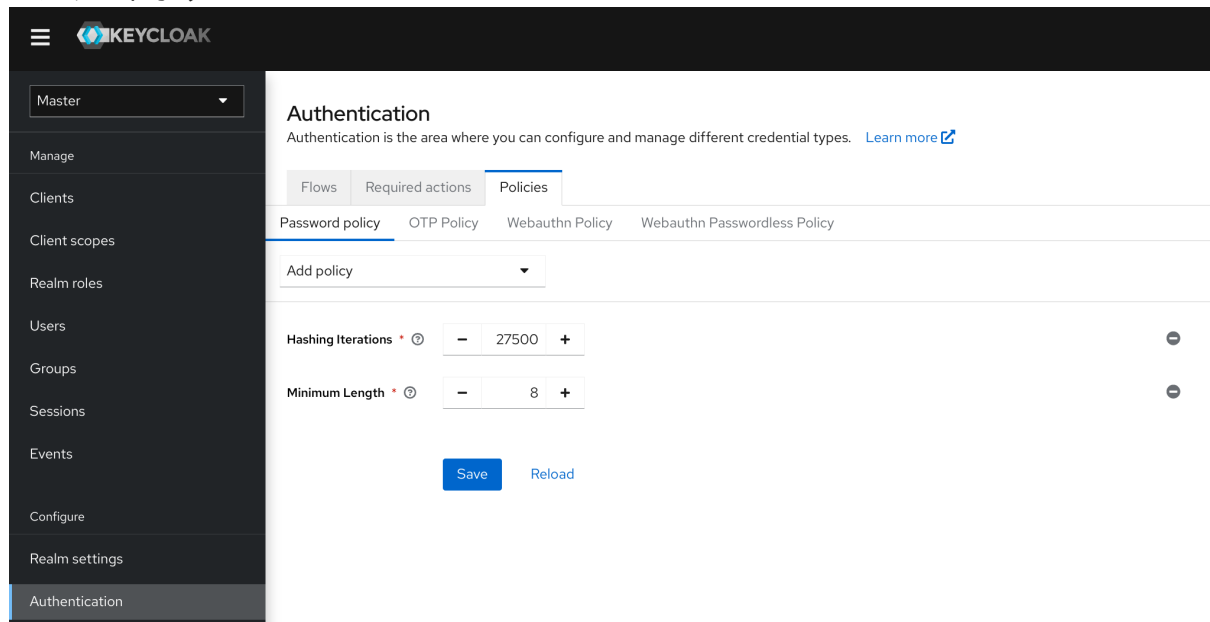
8.1. パスワードポリシー

Red Hat build of Keycloak は、レルム作成時にパスワードポリシーをレルムに関連付けません。長さ、セキュリティ、または複雑性に制限のない簡単なパスワードを設定できます。実稼働環境では、シンプルなパスワードは受け入れられません。Red Hat build of Keycloak には、管理コンソールから利用できるパスワードポリシーのセットが含まれています。

手順

1. メニューで **Authentication** をクリックします。
2. **Policies** タブをクリックします。
3. **Add policy** ドロップダウンボックスで、追加するポリシーを選択します。
4. 選択したポリシーに適用する値を入力します。
5. **Save** をクリックします。

パスワードポリシー



ポリシーを保存すると、Red Hat build of Keycloak が新規ユーザーにそのポリシーを適用します。



注記

新しいポリシーは既存のユーザーには無効になります。したがって、レلم作成の最初からパスワードポリシーを設定するか、既存のユーザーにパスワードの更新を追加するか、パスワードの期限切れを使用してユーザーが次の N 日以内にパスワードを更新するようにしてください。これにより、実際には新しいパスワードポリシーに合わせて調整します。

8.1.1. パスワードポリシータイプ

8.1.1.1. HashAlgorithm

パスワードは平文で保存されません。Red Hat build of Keycloak は、保存または検証の前に、標準のハッシュアルゴリズムを使用してパスワードをハッシュ化します。PBKDF2 は、利用可能な唯一の組み込みのデフォルトアルゴリズムです。独自のハッシュアルゴリズムを追加する方法は、[サーバー開発ガイド](#)を参照してください。



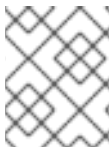
注記

ハッシュアルゴリズムを変更した場合、ストレージ内のパスワードハッシュは、ユーザーがログインするまで変更されません。

8.1.1.2. ハッシュの反復

Red Hat build of Keycloak が、保存または検証前にパスワードをハッシュ化する回数を指定します。デフォルト値は 27,500 です。

Red Hat build of Keycloak はパスワードをハッシュ化し、パスワードデータベースへのアクセスを持つ敵対者がリバースエンジニアリングを通じてパスワードを読み取ることができないようにします。



注記

ハッシュの反復値が高いと、CPU のべき乗を増やす必要があるため、パフォーマンスに影響する可能性があります。

8.1.1.3. 数字

パスワード文字列に必要な数字の数。

8.1.1.4. 小文字

パスワード文字列に必要な小文字の数。

8.1.1.5. 大文字

パスワード文字列に必要な大文字の数。

8.1.1.6. 特殊文字

パスワード文字列に必要な特殊文字の数。

8.1.1.7. ユーザー名なし

パスワードはユーザー名と同じにすることはできません。

8.1.1.8. メールなし

パスワードは、ユーザーのメールアドレスと同じにすることはできません。

8.1.1.9. 正規表現

パスワードは、定義済みの正規表現パターンを1つ以上一致させる必要があります。

8.1.1.10. パスワードが失効する

パスワードが有効な日数。有効期限が切れた日数が経過したら、パスワードを変更する必要があります。

8.1.1.11. 最近使用されていない

ユーザーがパスワードを使用できない。Red Hat build of Keycloak は、使用されたパスワードの履歴を保存します。保存される古いパスワードの数は、Red Hat build of Keycloak で設定できます。

8.1.1.12. パスワードのブラックリスト

パスワードをブラックリストファイルに含めることはできません。

- ブラックリストファイルは、Unix 行で終わる UTF-8 プレーンテキストファイルです。すべての行は、ブラックリストに指定されたパスワードを表します。
- Red Hat build of Keycloak は、大文字と小文字を区別せずにパスワードを比較します。ブラックリストのすべてのパスワードは小文字でなければなりません。
- ブラックリストファイルの値は、ブラックリストファイルの名前(たとえば、`100k_passwords.txt`)である必要があります。
- ブラックリストファイルは、デフォルトでは `${kc.home.dir}/data/password-blacklists/` に対して解決されます。以下を使用して、このパスをカスタマイズします。
 - `keycloak.password.blacklists.path` システムプロパティ。
 - `passwordBlacklist` ポリシー SPI 設定の `blacklistsPath` プロパティ。CLI を使用してブラックリストフォルダーを設定するには、`--spi-password-policy-password-blacklist-blacklists-path=/path/to/blacklistsFolder` を使用します。

誤検知に関する注意事項

現在の実装では、特定のパスワードがブラックリストに含まれているかどうかなど、誤検知の可能性がある封じ込めチェックを高速かつメモリー効率よく行うために BloomFilter を使用しています。

- デフォルトでは、**0.01%** の偽陽性確率が使用されます。
- CLI 設定で誤検知確率を変更するには、`--spi-password-policy-password-blacklist-false-positive-probability=0.00001` を使用します

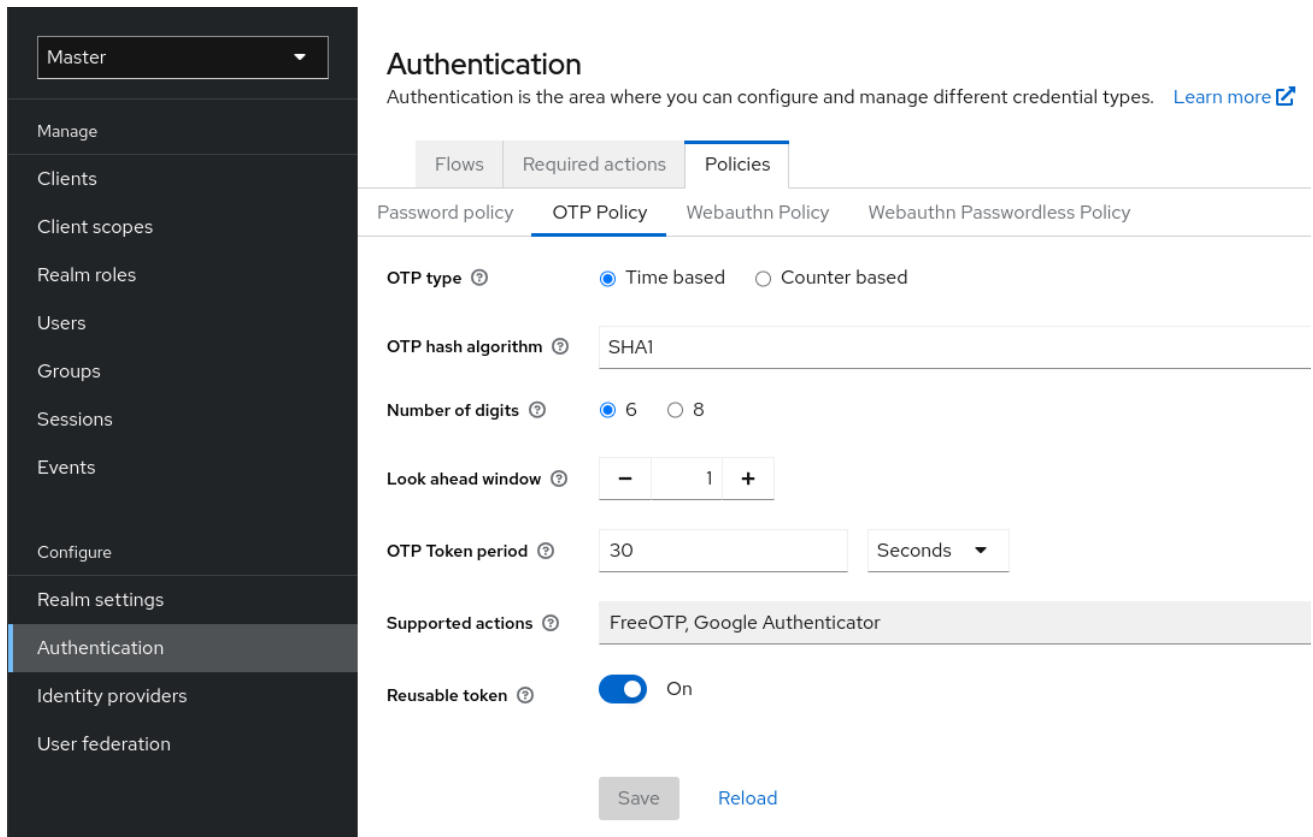
8.2. ワンタイムパスワード (OTP) ポリシー

Red Hat build of Keycloak には、FreeOTP または Google Authenticator のワンタイムパスワードジェネレーターを設定するためのいくつかのポリシーがあります。

手順

1. メニューで **Authentication** をクリックします。
2. **Policy** タブをクリックします。
3. **OTP Policy** タブをクリックします。

OTP ポリシー



The screenshot shows the Keycloak Administration Console interface. On the left is a dark sidebar menu with 'Authentication' selected. The main content area is titled 'Authentication' and has tabs for 'Flows', 'Required actions', and 'Policies'. Under the 'Policies' tab, there are sub-tabs for 'Password policy', 'OTP Policy', 'Webauthn Policy', and 'Webauthn Passwordless Policy'. The 'OTP Policy' sub-tab is active, showing configuration options: 'OTP type' (Time based selected), 'OTP hash algorithm' (SHA1), 'Number of digits' (6 selected), 'Look ahead window' (1), 'OTP Token period' (30 seconds), 'Supported actions' (FreeOTP, Google Authenticator), and 'Reusable token' (On). 'Save' and 'Reload' buttons are at the bottom.

Red Hat build of Keycloak は、**OTP Policy** タブで設定された情報に基づき、OTP 設定ページに QR コードを生成します。FreeOTP および Google Authenticator は、OTP の設定時に QR コードをスキャンします。

8.2.1. 時間ベースまたはカウンターベースのワンタイムパスワード

Red Hat build of Keycloak では、OTP ジェネレーター用に時間ベースとカウンターベースのアルゴリズムを使用できます。

タイムベースのワンタイムパスワード (TOTP) を使用すると、トークンジェネレーターは現在の時刻と共有秘密をハッシュ化します。サーバーは、ウィンドウ内のハッシュを送信した値と比較することにより、OTP を検証します。TOTP は短時間に有効です。

カウンターベースのワンタイムパスワード (HOTP) を使用すると、Red Hat build of Keycloak は現在の時刻ではなく共有カウンターを使用します。Red Hat build of Keycloak サーバーは、OTP ログインが成功するたびにカウンターをインクリメントします。ログインに成功した後、有効な OTP が変更されません。

一致可能な OTP は短時間で有効になり、OTP for HOTP は終了期間に有効であるため、TOTP は HOTP よりも安全です。OTP に入るのに時間制限が存在しないため、HOTP は TOTP よりも使いやすいことです。

HOTP では、サーバーがカウンターをインクリメントするたびにデータベースの更新が必要になります。この更新は、負荷が大きい間認証サーバーでのパフォーマンスドレイン (解放) です。TOTP は、効率性を向上させるために、使用するパスワードを記憶しないため、データベースの更新を行う必要はありません。欠点は、有効な期間で TOTP を再使用できることです。

8.2.2. TOTP 設定オプション

8.2.2.1. OTP ハッシュアルゴリズム

デフォルトのアルゴリズムは SHA1 です。これ以外のセキュアなオプションは SHA256 と SHA512 です。

8.2.2.2. 数字の数

OTP の長さ。短い OTP はユーザーフレンドリーで、種類が簡単で、覚えやすいものになります。OTP が長いほど、OTP が短くなるよりも安全です。

8.2.2.3. ウィンドウを見てください

サーバーがハッシュを照合しようとする間隔数。TOTP ジェネレーターまたは認証サーバーのクロックが同期しなくなった場合、Red Hat build of Keycloak にこのオプションが表示されます。デフォルト値は 1 です。たとえば、トークンの時間間隔が 30 秒の場合、デフォルト値の 1 は、90 秒のウィンドウで有効なトークンを受け入れることを意味します (時間間隔 30 秒 + 先読み 30 秒 + 後読み 30 秒)。この値を増やすたびに、有効なウィンドウが 60 秒ずつ増加します (30 秒先を見る + 30 秒後を見る)。

8.2.2.4. OTP トークン期間

サーバーがハッシュに一致する間隔 (秒単位)。間隔がパスするたびに、トークンジェネレーターは TOTP を生成します。

8.2.2.5. 再利用可能なコード

OTP トークンを認証プロセスで再利用できるかどうか、またはユーザーが次のトークンを待つ必要があるかどうかを決定します。デフォルトでは、ユーザーはこれらのトークンを再利用できないため、管理者はそれらのトークンが再利用できることを明示的に指定する必要があります。

8.2.3. HOTP 設定オプション

8.2.3.1. OTP ハッシュアルゴリズム

デフォルトのアルゴリズムは SHA1 です。これ以外のセキュアなオプションは SHA256 と SHA512 です。

8.2.3.2. 数字の数

OTP の長さ。短い OTP はユーザーフレンドリーで、種類が簡単で、覚えやすいものになります。長い OTP は、OTP を短くするより安全です。

8.2.3.3. ウィンドウを見てください

サーバーがハッシュとの一致を試みる前後の間隔の数。TOTP ジェネレーターまたは認証サーバーのクロックが同期しなくなった場合、Red Hat build of Keycloak にこのオプションが表示されます。デフォルト値は1です。このオプションは、ユーザーのカウンターがサーバーよりも先に進んだ場合に備えて Red Hat build of Keycloak に存在します。

8.2.3.4. 初期カウンター

初期カウンターの値。

8.3. 認証フロー

authentication flow は、ログイン、登録、その他の Red Hat build of Keycloak ワークフロー中の認証、画面、アクションのコンテナです。

8.3.1. 組み込みフロー

Red Hat build of Keycloak には、いくつかのビルトインフローがあります。これらのフローは変更できませんが、フローの要件をニーズに合わせて変更することができます。

手順

1. メニューで **Authentication** をクリックします。
2. リスト内の **Browser** 項目をクリックして詳細を表示します。

ブラウザーのフロー

The screenshot shows the 'Authentication > Flow details' page for the 'Browser' flow. The flow is marked as 'Default' and 'Built-in'. The steps and their requirements are as follows:

Steps	Requirement
Cookie	Alternative
Kerberos	Disabled
Identity Provider Redirector	Alternative
forms Username, password, otp and other auth forms.	Alternative
Username Password Form	Required
Browser - Conditional OTP Flow to determine if the OTP is required for the authentication	Conditional
Condition - user configured	Required
OTP Form	Required

At the bottom of the flow list, there are two buttons: '+ Add step' and '+ Add sub-flow'.

8.3.1.1. 認証タイプ

実行する認証またはアクションの名前。認証がインデントされると、これはサブフローにあります。これは、親の動作によって実行されるか、実行されていない可能性があります。

1. cookie
ユーザーが初めてログインに成功すると、Red Hat build of Keycloak はセッション Cookie を設定します。クッキーがすでに設定されている場合、この認証タイプは成功します。Cookie プロバイダーが成功を返し、このフローレベルのそれぞれの実行が **alternative** であるため、Red Hat build of Keycloak は他の実行を行いません。これにより、ログインに成功します。
2. Kerberos
このオーセンティケーターはデフォルトで無効になっており、ブラウザーフローではスキップされます。
3. アイデンティティプロバイダーのリダイレクター
このアクションは、**Actions > Config** リンクで設定されます。 **Identity Brokering** のために別の IdP にリダイレクトします。
4. フォーム
このサブフローは **代替** としてマークされているため、Cookie **認証タイプ** が渡されると実行されません。このサブフローには、実行する必要がある追加の認証タイプが含まれています。Red Hat build of Keycloak は、このサブフローの実行をロードして処理します。

最初の実行は、ユーザー名とパスワードのページをレンダリングする認証タイプである **Username Password Form** です。これは**必須**と識別されているため、ユーザーは有効なユーザー名とパスワードを入力する必要があります。

2 回目の実行は、**Browser - Conditional OTP**サブフローです。このサブフローは**conditional**で、ユーザー設定の実行結果 **Condition - User Configured** 実行に応じて実行されます。結果が true の場合、Red Hat build of Keycloak はこのサブフローの実行をロードして処理します。

次の実行は、**Condition - User Configured** 認証です。この認証は、Red Hat build of Keycloak がそのフロー内で、そのユーザーの他の実行を設定しているか確認します。**Browser - Conditional OTP**サブフローは、ユーザーが OTP 認証情報が設定された場合にのみ実行されます。

最後の実行は **OTP Form** です。Red Hat build of Keycloak はこの実行を **required** としてマークしますが、**conditional** サブフローでのセットアップのため、ユーザーが OTP 認証情報をセットアップしている場合にのみ実行されます。そうでない場合は、OTP フォームは表示されません。

8.3.1.2. 要件

アクション実行を制御するラジオボタンのセット。

8.3.1.2.1. 必須

フローで **必須** 要素がすべて順次実行される必要があります。フローは、必須要素が失敗すると終了します。

8.3.1.2.2. 代替方法

フローが正常に実行されると評価するには、単一の要素のみを正常に実行する必要があります。**Required** flow 要素にはフローに **successful** というマークを付けるだけで十分であるため、**Required** フロー要素が含まれるフロー内の **Alternative** flow 要素は実行されません。

8.3.1.2.3. Disabled

要素は、フローに `successful` というマークを付けることはできません。

8.3.1.2.4. 条件付き

この要件タイプはサブフローにのみ設定されます。

- **Conditional** サブフローには実行が含まれます。これらの実行は論理ステートメントに評価する必要があります。
- すべての実行が `true` として評価されると、**Conditional** サブフローは**必須**として動作します。
- すべての実行が `false` と評価されると、**Conditional** サブフローは **Disabled** として機能しません。
- 実行を設定しない場合、**Conditional** サブフローは **Disabled** として機能します。
- フローに実行が含まれ、かつ **Conditional** に設定されていない場合、Red Hat build of Keycloak は実行を評価せず、実行は機能的に **Disabled** とみなされます。

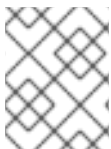
8.3.2. フローの作成

フローの設計時に、重要な機能およびセキュリティ上の考慮事項が適用されます。

フローを作成するには、以下を実行します。

手順

1. メニューで **Authentication** をクリックします。
2. **Create flow** をクリックします。



注記

既存のフローをコピーおよび変更できます。アクションリスト (行の最後にある3つの点) をクリックし、**Duplicate** をクリックして、新しいフローの名前を入力します。

新しいフローを作成する場合は、まず以下のオプションでトップレベルフローを作成する必要があります。

名前

フローの名前。

Description

フローに設定できる説明。

Top-Level Flow Type

フローのタイプ。タイプ **クライアント**は、**クライアント (アプリケーション)** の認証にのみ使用されます。それ以外の場合はすべて、**basic** を選択します。

トップレベルフローの作成

Authentication > Create flow

Create flow

You can create a top level flow within this from

Name * ⓘ

Description ⓘ

Flow type ⓘ Basic flow

[Create](#) [Cancel](#)

Red Hat build of Keycloak がフローを作成すると、Red Hat build of Keycloak に **Add step** ボタンと **Add sub-flow** ボタンが表示されます。

空の新規フロー

Authentication > Flow details

New flow Not in use

No steps

You can start defining this flow by adding a sub-flow or an execution

Add an execution
Execution can have a wide range of actions, from sending a reset email to validating an OTP [Add execution](#)

Add a sub-flow
Sub-Flows can be either generic or form. The form type is used to construct a sub-flow that generates a single flow for the user. Sub-flows are a special type of execution that evaluate as successful depending on how the executions they contain evaluate. [Add sub-flow](#)

3つの要因により、フローとサブフローの動作が決定されます。

- フローおよびサブフローの構造。
- フロー内での実行
- サブフローおよび実行内で設定される要件。

リセットメールの送信から OTP の検証まで、実行にはさまざまなアクションを設定できます。 **Add step** ボタンを使用して実行を追加します。

認証実行の追加

Add step to New flow



1 - 10 ▾



- Browser Redirect for Cookie free authentication
Perform a 302 redirect to get user agent's current URI on authenticate path with an `auth_session_id` query parameter. This is for client's that do not support cookies.
- Cookie
Validates the SSO cookie set by the auth server.
- Username Password Challenge
Proprietary challenge protocol for CLI clients that queries for username password
- Choose User
Choose a user to reset credentials for
- Password
Validates the password supplied as a 'password' form parameter in direct grant request
- WebAuthn Authenticator
Authenticator for WebAuthn. Usually used for WebAuthn two-factor authentication
- Kerberos
Initiates the SPNEGO protocol. Most often used with Kerberos.
- Reset Password
Sets the Update Password required action if execution is REQUIRED. Will also set it if execution is OPTIONAL and the password is currently configured for it.
- X509/Validate Username
Validates username and password from X509 client certificate received as a part of mutual SSL handshake.
- Password Form
Validates a password from login form.
- Docker Authenticator
Uses HTTP Basic authentication to validate docker users, returning a docker error token on auth failure

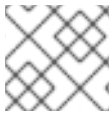
1 - 10 ▾

[Add](#)[Cancel](#)

自動実行とインタラクティブな実行の2種類の実行があります。自動実行は Cookie 実行に類似し、フローのアクションを自動的に実行します。インタラクティブな実行は、入力を取得するためにフローを停止します。正常に実行されると、ステータスを **success** に設定します。フローが完了するには、ステータスが **success** の実行が少なくとも1つ必要です。


Add sub-flow ボタンを使用して、サブフローを最上位のフローに追加できます。Add sub-flow ボタン

をクリックすると、**Create Execution Flow** ページが表示されます。このページは **Create Top Level Form** ページに似ています。違いは、**Flow Type** に **basic** (デフォルト) と **form** がある点です。**form** タイプは、組み込み **Registration** フローに似た、ユーザーのフォームを生成するサブフローを構築します。サブフローが成功するかどうかは、含まれるサブフローを含め、実行がどのように評価されるかによります。サブフローがどのように機能するかの詳細な説明については、[実行要件](#) セクションを参照してください。



注記

実行を追加したら、要件の値が正しいことを確認します。

フロー内のすべての要素には、要素の横に **Delete** オプションがあります。一部の実行には、実行を設定するための  メニュー項目 (歯車アイコン) があります。**Add step** および **Add sub-flow** リンクで、サブフローに実行とサブフローに追加することもできます。

実行の順序は重要であるため、名前をドラッグして実行とサブフローを上下に移動できます。



警告

認証フローを設定するときは、設定を適切にテストして、セットアップにセキュリティホールが存在しないことを確認してください。さまざまなコーナーケースをテストすることが推奨されます。たとえば、認証前にユーザーのアカウントからさまざまな認証情報を削除するときに、ユーザーの認証動作をテストすることを検討してください。

たとえば、OTP フォームや WebAuthn オーセンティケーターなどの第 2 要素オーセンティケーターがフローで **REQUIRED** として設定されていて、ユーザーが特定のタイプのクレデンシャルを持っていない場合、ユーザーは認証自体の間に特定のクレデンシャルをセットアップできます。この状況は、認証中にユーザーがこの認証情報で認証されないことを意味します。ブラウザ認証の場合は、**Password** や **WebAuthn Passwordless Authenticator** などの一部の 1 つの要素認証情報で認証フローを設定してください。

8.3.3. パスワードなしのブラウザログインフローの作成

フローの作成を説明するために、本セクションでは高度なブラウザログインフローの作成について説明します。このフローの目的は、ユーザーが、[WebAuthn](#) によるパスワードなしの方法でのログインと、パスワードと OTP を使用した二要素認証を選択できるようにすることです。

手順

1. メニューで **Authentication** をクリックします。
2. **Flows** タブをクリックします。
3. **Create flow** をクリックします。
4. 名前として **Browser Password-less** を入力します。
5. **Create** をクリックします。

6. **Add execution** をクリックします。
7. リストから **Cookie** を選択します。
8. **Add** をクリックします。
9. **Cookie** 認証タイプの **Alternative** を選択して、その要件を代替に設定します。
10. **Add step** をクリックします。
11. リストから **Kerberos** を選択します。
12. **Add** をクリックします。
13. **Add step** をクリックします。
14. リストから **Identity Provider Redirector** を選択します。
15. **Add** をクリックします。
16. **Identity Provider Redirector** 認証タイプの **Alternative** を選択して、その要件を代替に設定します。
17. **Add sub-flow** をクリックします。
18. 名前として **Forms** と入力します。
19. **Add** をクリックします。
20. **Forms** 認証タイプの **Alternative** を選択して、その要件を代替に設定します。

ブラウザーフローの共通部分

The screenshot shows the 'Authentication > Flow details' page for a flow named 'Browser Password-less' (status: Not in use). The main content is a table with two columns: 'Steps' and 'Requirement'. There are four rows of configuration items:

Steps	Requirement
Cookie	Alternative
Kerberos	Disabled
Identity Provider Redirector	Alternative
Forms	Alternative

At the bottom of the table, there are two links: '+ Add step' and '+ Add sub-flow'.

21. **Forms** 実行の + メニューをクリックします。
22. **Add step** を選択します。

23. リストから **Username Form** を選択します。

24. **Add** をクリックします。

この段階では、フォームにはユーザー名が必要ですが、パスワードは必要ありません。セキュリティリスクを回避するために、パスワード認証を有効にする必要があります。

1. **Forms** サブフローの + メニューをクリックします。
2. **Add sub-flow** をクリックします。
3. 名前として **Authentication** を入力します。
4. **Add** をクリックします。
5. **Authentication** 認証タイプで **Required** を選択し、その要件を必須に設定します。
6. **Authentication** サブフローの + メニューをクリックします。
7. **Add step** をクリックします。
8. リストから **WebAuthn Passwordless Authenticator** を選択します。
9. **Add** をクリックします。
10. **Webauthn Passwordless Authenticator** 認証タイプの **Alternative** を選択して、その要件を代替に設定します。
11. **Authentication** サブフローの + メニューをクリックします。
12. **Add sub-flow** をクリックします。
13. 名前として **Password with OTP** を入力します。
14. **Add** をクリックします。
15. **Password with OTP** で **Alternative** を選択し、その要件を代替に設定します。
16. **Password with OTP** サブフローの + メニューをクリックします。
17. **Add step** をクリックします。
18. リストから **Password Form** を選択します。
19. **Add** をクリックします。
20. **Password Form** 認証タイプで **Required** を選択し、その要件を必須に設定します。
21. **Password with OTP** サブフローの + メニューをクリックします。
22. **Add step** をクリックします。
23. リストから **OTP Form** を選択します。
24. **Add** をクリックします。
25. **OTP Form** 認証タイプの **Required** をクリックして、その要件を必須に設定します。

最後に、バインディングを変更します。

1. 画面上部の **Action** メニューをクリックします。
2. メニューから **Bind flow** を選択します。
3. **Browser Flow** ドロップダウンリストをクリックします。
4. **Save** をクリックします。

パスワードなしのブラウザログイン

Steps	Requirement
Cookie	Alternative
Kerberos	Disabled
Identity Provider Redirector	Alternative
Forms	Alternative
Username Form	Required
Authentication	Required
WebAuthn Passwordless Authenticator	Alternative
Password Form	Disabled
OTP Form	Required
Password with OTP	Disabled
Password Form	Disabled
OTP Form	Required

+ Add step + Add sub-flow

ユーザー名を入力すると、フローは以下のように機能します。

ユーザーの WebAuthn パスワードレス認証情報が記録されている場合は、これらの認証情報を使用して直接ログインできます。これはパスワードなしのログインです。**WebAuthn Passwordless** 実行および **Password with OTP** フローが **Alternative** に設定されているため、ユーザーは **Password with OTP** を選択することもできます。**Required** に設定されている場合は、ユーザーは WebAuthn、パスワード、および OTP を入力する必要があります。

ユーザーが **WebAuthn passwordless** 認証で **Try another way** のリンクを選択する場合、ユーザーは

Password と **Security Key**(WebAuthn パスワードレス) のいずれかを選択できます。パスワードを選択すると、ユーザーは続行して、割り当てられた OTP でログインする必要があります。ユーザーに WebAuthn 認証情報がない場合は、ユーザーはパスワードを入力し、続いて OTP を入力する必要があります。ユーザーに OTP 認証情報がない場合は、記録することが要求されます。

注記

WebAuthn Passwordless 実行は **Required** ではなく **Alternative** に設定されているため、このフローはユーザーに WebAuthn 認証情報を登録するように要求しません。ユーザーに WebAuthn 認証情報を設定するには、管理者がユーザーに必須アクションを追加する必要があります。これを行うには、以下を実行します。

1. レルムで **Webauthn Register Passwordless** で必須アクションを有効にします ([WebAuthn](#) のドキュメントを参照)。
2. ユーザーの **Credentials** 管理メニューの **Credential Reset** 部分を使用して、必須アクションを設定します。

このような高度なフローを作成すると、副次的な影響が生じる可能性があります。たとえば、ユーザーのパスワードをリセットできるようにする場合は、パスワードフォームからアクセスが可能になります。デフォルトの **Reset Credentials** フローで、ユーザーはユーザー名を入力する必要があります。ユーザーは **Browser Password-less** フローですでにユーザー名を入力しているため、Red Hat build of Keycloak ではこの操作は必要なく、ユーザーエクスペリエンスとしては最適ではありません。この問題を修正するには、以下を行います。

- **Reset Credentials** フローを複製します。たとえば、その名前を **Reset Credentials for password-less** に設定します。
- **Choose user** ステップの **Delete** (ゴミ箱アイコン) をクリックします。
- **Action** メニューで、**Bind flow** を選択し、ドロップダウンから **Reset credentials flow** を選択して、**Save** をクリックします。

8.3.4. ステップアップメカニズムを使用したブラウザーログインフローの作成


本セクションでは、ステップアップメカニズムを使用して高度なブラウザーログインフローを作成する方法を説明します。ステップ認証の目的は、ユーザーの特定の認証レベルに基づいてクライアントまたはリソースへのアクセスを許可することです。

手順

1. メニューで **Authentication** をクリックします。
2. **Flows** タブをクリックします。
3. **Create flow** をクリックします。
4. **Browser Incl Step up Mechanism** を名前として入力します。
5. **Save** をクリックします。
6. **Add execution** をクリックします。
7. リストから **Cookie** を選択します。

8. **Add** をクリックします。
9. **Cookie** 認証タイプの **Alternative** を選択して、その要件を代替に設定します。
10. **Add sub-flow** をクリックします。
11. **Auth Flow** を名前として入力します。
12. **Add** をクリックします。
13. **Auth Flow** 認証タイプの **Alternative** をクリックして、その要件を代替に設定します。

最初の認証レベルのフローを設定します。

1. **Auth Flow** の + メニューをクリックします。
2. **Add sub-flow** をクリックします。
3. **1st Condition Flow** を名前として入力します。
4. **Add** をクリックします。
5. **1st Condition Flow** 認証タイプの **Conditional** をクリックし、その要件を条件に設定します。
6. **1st Condition Flow** の + メニューをクリックします。
7. **Add condition** をクリックします。
8. リストから **Conditional - Level Of Authentication** を選択します。
9. **Add** をクリックします。
10. **Conditional - Level Of Authentication** 認証タイプの **Required** をクリックして、その要件を必須に設定します。
11.  (歯車アイコン) をクリックします。
12. **Level 1** をエイリアスとして入力します。
13. レベル認証 (LoA) に **1** と入力します。
14. **Max Age** を **36000** に設定します。この値は秒単位であり、10 時間に相当します。これは、レールムで設定されているデフォルトの **SSO Session Max** タイムアウトです。その結果、ユーザーがこのレベルで認証される場合、後続の SSO ログインはこのレベルを再利用でき、ユーザーセッションが終了するまでユーザーはこのレベルで認証する必要はありません (デフォルトでは 10 時間)。
15. **Save** をクリックします。

最初の認証レベルの条件を設定します。

Condition - Level of Authentication config ×

Alias * ?

Level 1

Level of Authentication (LoA) ?

1

Max Age ?

36000


Save

Cancel

16. 1st Condition Flow の + メニューをクリックします。
17. Add step をクリックします。
18. リストから Usermae Password Form を選択します。
19. Add をクリックします。

これで、2つ目の認証レベルのフローが設定されます。

1. Auth Flow の + メニューをクリックします。
2. Add sub-flow をクリックします。
3. 2nd Condition Flow をエイリアスとして入力します。
4. Add をクリックします。
5. 2st Condition Flow 認証タイプの Conditional をクリックし、その要件を条件に設定します。
6. 2nd Condition Flow の + メニューをクリックします。
7. Add condition をクリックします。
8. 項目 リストから Conditional - Level Of Authentication を選択します。
9. Add をクリックします。
10. Conditional - Level Of Authentication 認証タイプの Required をクリックして、その要件を必須に設定します。

11.  (歯車アイコン) をクリックします。
12. **Level 2** をエイリアスとして入力します。
13. 認証レベル (LoA) に **2** と入力します。
14. Max Age を **0** に設定します。その結果、ユーザーが認証する場合、このレベルは現在の認証に対してのみ有効ですが、後続の SSO 認証に対しては有効ではありません。したがって、このレベルが要求された場合、ユーザーは常にこのレベルで再度認証する必要があります。
15. **Save** をクリックします。

2つ目の認証レベルの条件を設定する

Condition - Level of Authentication config ×

Alias * 

Level of Authentication (LoA) 

Max Age 

16. 2nd Condition Flow の + メニューをクリックします。
17. **Add step** をクリックします。
18. リストから **OTP Form** を選択します。
19. **Add** をクリックします。
20. **OTP Form** 認証タイプの **Required** をクリックして、その要件を必須に設定します。

最後に、バインディングを変更します。

1. 画面上部の **Action** メニューをクリックします。
2. リストから **Bind flow** を選択します。

3. ドロップダウンから **Browser Flow** を選択します。

4. **Save** をクリックします。

ステップアップメカニズムによるブラウザログイン

Authentication > Flow details

Browser Incl Step up Mechanism Not in use

Steps	Requirement
Cookie	Alternative
Auth Flow	Alternative
1st Condition Flow	Conditional
Condition - Level of Authentication	Required
Username Password Form	Required
2nd Condition Flow	Conditional
Condition - Level of Authentication	Required
OTP Form	Disabled

特定の認証レベルを要求します。

ステップアップメカニズムを使用するには、認証リクエストに要求された認証レベル (LoA) を指定します。**claims** パラメーターは、この目的で使用されます。

```
https://{DOMAIN}/realms/{REALMNAME}/protocol/openid-connect/auth?client_id={CLIENT-ID}&redirect_uri={REDIRECT-URI}&scope=openid&response_type=code&response_mode=query&nonce=exg16fxdjcu&claims=%7B%22id_token%22%3A%7B%22acr%22%3A%7B%22essential%22%3Atrue%2C%22values%22%3A%5B%22gold%22%5D%7D%7D
```

claims パラメーターは JSON 表現で指定されます。

```
claims= {
  "id_token": {
    "acr": {
      "essential": true,
      "values": ["gold"]
    }
  }
}
```

Red Hat build of Keycloak の JavaScript アダプターは、この JSON を簡単に構築し、ログイン要求で送信できるようにサポートします。詳細は、[Javascript アダプターのドキュメント](#) を参照してください。

また、**claims** パラメーターの代わりに単純なパラメーター **acr_values** を使用して、特定のレベルを必須ではないものとして要求することもできます。これは OIDC 仕様で説明されています。

特定のクライアントのデフォルトレベルを設定することもできます。これは、**acr_values** パラメーターまたは **acr** 要求のある **claims** パラメーター要求が存在しない場合に使用されます。詳細については、[クライアント ACR 設定](#) を参照してください。



注記

acr_values を数値ではなくテキスト (**gold** など) として要求するには、ACR と LoA の間のマッピングを設定します。レルムレベル (推奨) またはクライアントレベルで設定できます。設定については、[ACR から LoA へのマッピング](#) を参照してください。

詳細は、[公式の OIDC 仕様](#) を参照してください。

フローロジック

以前に設定された認証フローのロジックは次のとおりです。

クライアントが高い認証レベル、つまり認証レベル 2 (LoA 2) を要求した場合、ユーザーは完全な 2 要素認証 (ユーザー名/パスワード + OTP) を実行する必要があります。ただし、ユーザーが Keycloak ですでにセッションを持っていて、それがユーザー名とパスワード (LoA 1) でログインしている場合、ユーザーは 2 番目の認証要素 (OTP) のみを要求されます。

条件の **Max Age** 時間オプションは、後続の認証レベルが有効である期間 (秒数) を決定します。この設定は、ユーザーが後続の認証中に認証要素を再度提示するように求められるかどうかを決定するのに役立ちます。特定のレベル X が **claims** または **acr_values** パラメーターによって要求され、ユーザーがすでにレベル X で認証されているが、有効期限が切れている場合 (たとえば、最大年齢が 300 に設定され、ユーザーが 310 秒前に認証された場合)、ユーザーは再認証を求められます。特定のレベルで再度認証します。ただし、レベルが期限切れでない場合、ユーザーは自動的にそのレベルで認証されていると見なされます。

Max Age を値 0 で使用すると、その特定のレベルはこの単一認証でのみ有効です。そのため、そのレベルを要求するすべての再認証では、そのレベルで再度認証する必要があります。これは、アプリケーションでより高いセキュリティを必要とし (支払いの送信など)、常に特定のレベルでの認証を必要とする操作に役立ちます。



警告

ログイン要求がクライアントからユーザーのブラウザ経由で Red Hat build of Keycloak に送信されるときに、**claim** や **acr_values** などのパラメーターが URL 内のユーザーによって変更される可能性があることに注意してください。この状況は、クライアントが PAR (プッシュされた認可要求)、要求オブジェクト、またはユーザーが URL のパラメーターを書き換えることを妨げるその他のメカニズムを使用する場合に軽減できます。そのため、認証後に、トークンの **acr** が予想されるレベルに対応するように ID トークンを確認することが推奨されます。

パラメーターによって明示的なレベルが要求されていない場合、Red Hat build of Keycloak では、前の例のユーザー名/パスワードなど、認証フローで最初に見つかった LoA 条件での認証が必要になります。ユーザーがそのレベルですでに認証され、そのレベルの有効期限が切れると、ユーザーは再認証する必要はありませんが、トークンの **acr** の値は 0 になります。この結果は、OIDC Core 1.0 仕様のセクション 2 で説明されているように、**long-lived browser cookie** のみに基づく認証と見なされます。



注記

管理者が複数のフローを指定し、異なる LoA レベルをそれぞれに設定し、フローを別のクライアントに割り当てると、競合状況が発生する可能性があります。ただし、ルールは常に同じです。ユーザーに特定のレベルがある場合は、クライアントに接続するためにそのレベルのみが必要になります。LoA が一貫していることを確認するのは管理者次第です。

シナリオ例

1. Max Age は、レベル 1 の状態で 300 秒として設定されています。
2. ログイン要求は、acr を要求せずに送信されます。レベル 1 が使用され、ユーザーはユーザー名とパスワードで認証する必要があります。トークンには **acr=1** があります。
3. 別のログイン要求は 100 秒後に送信されます。SSO によりユーザーが自動的に認証され、トークンは **acr=1** を返します。
4. さらに 201 秒 (ポイント 2 での認証から 301 秒) 後に別のログイン要求が送信されます。ユーザーは SSO により自動的に認証されますが、レベル 1 が期限切れと見なされるため、トークンは **acr=0** を返します。
5. 別のログイン要求が送信されますが、これで、**claims** パラメーターでレベル 1 の ACR が明示的に要求されます。ユーザーはユーザー名/パスワードで再認証するように求められ、その後 **acr=1** がトークンで返されます。

トークンの ACR クレーム

ACR 要求は、**acr** クライアントスコープで定義された **acr loa level** のプロトコルマッパーによってトークンに追加されます。このクライアントスコープはレルムのデフォルトのクライアントスコープであるため、レルム内に新しく作成されたすべてのクライアントに追加されます。

トークン内で **acr** クレームが必要ない場合、またはトークンを追加するためのカスタムロジックが必要な場合は、クライアントからクライアントスコープを削除できます。

ログイン要求が **acr** を **essential** クレームとして要求する **claims** パラメーターを使用して要求を開始すると、Red Hat build of Keycloak は常に指定されたレベルの 1 つを返すことに注意してください。指定されたレベルの 1 つを返すことができない場合 (たとえば、要求されたレベルが不明であるか、認証フローで設定された条件よりも大きい場合)、Red Hat build of Keycloak はエラーを出力します。

8.3.5. クライアントから要求された認証情報の登録またはリセット

通常、ユーザーがクライアントアプリケーションから Red Hat build of Keycloak にリダイレクトされると、**browser** フローがトリガーされます。このフローでは、レルム登録が有効になっていて、ユーザーがログイン画面で **Register** をクリックした場合に、ユーザーが **登録** できるようになります。また、レルムに対して **Forget password** が有効になっている場合、ユーザーはログイン画面で **Forget password** をクリックすることができ、これにより **Reset credentials** フローがトリガーとなり、ユーザーはメールアドレスの確認後に認証情報をリセットできます。

クライアントアプリケーションがユーザーを **Registration** 画面または **Reset credentials** フローに直接

リダイレクトすると便利な場合があります。結果のアクションは、ユーザーが通常のログイン画面で **Register** または **Forget password** をクリックしたときのアクションと一致します。登録画面または認証情報のリセット画面への自動リダイレクトは、次のように実行できます。

- クライアントがユーザーを登録に直接リダイレクトする場合、OIDC クライアントは OIDC ログイン URL パス (`/auth`) の最後のスニペットを `/registrations` に置き換える必要があります。したがって、完全な URL は https://keycloak.example.com/realms/your_realm/protocol/openid-connect/registrations のようになります。
- クライアントがユーザーを **Reset credentials** フローに直接リダイレクトすると、OIDC クライアントは OIDC ログイン URL パス (`/auth`) の最後のスニペットを `/forgot-credentials` に置き換える必要があります。




警告

上記の手順は、クライアントが登録または認証情報リセットフローを直接要求するためにサポートされている唯一の方法です。セキュリティ上の理由から、クライアントアプリケーションが OIDC/SAML フローを回避し、他の Red Hat build of Keycloak エンドポイント (たとえば、`/realms/realm_name/login-actions` または `/realms/realm_name/broker` 下のエンドポイントなど) に直接リダイレクトすることはサポートされておらず、推奨されていません。

8.4. ユーザーセッションの制限

ユーザーが持つことができるセッション数の制限を設定できます。セッションは、レルムごとまたはクライアントごとに制限できます。

フローにセッション制限を追加するには、次の手順を実行します。

1. フローの **Add step** をクリックします。
2. 項目 リストから **User session count limiter** を選択します。
3. **Add** をクリックします。
4. **ユーザーセッションカウントリミッター** 認証タイプの **必須** をクリックして、要件を必須に設定します。
5. **User Session Count Limiter** の  (歯車アイコン) をクリックします。
6. この設定のエイリアスを入力します。
7. このレルムでユーザーが持つことができるセッションの最大数を入力します。たとえば、値が 2 の場合、各ユーザーがこのレルムで保持できる最大数は 2 SSO セッションです。値が 0 の場合、このチェックは無効になります。
8. クライアントにユーザーが持つことができるセッションの最大数を入力します。たとえば、値が 2 の場合、このレルムでは各クライアントの SSO セッションが最大 2 つになります。したがって、ユーザーがクライアント **foo** に対して認証しようとしているが、そのユーザーがすでにクライアント **foo** に対して 2 つの SSO セッションで認証されている場合、設定された動作

に基づいて認証が拒否されるか、既存のセッションが強制終了されます。0の値を使用すると、このチェックは無効になります。セッション制限とクライアントセッション制限の両方が有効になっている場合は、クライアントセッション制限を常にセッション制限より低くすることが合理的です。クライアントごとの制限は、このユーザーのすべてのSSOセッションの制限を超えることはできません。

9. 制限に達するとユーザーがセッションを作成しようとする必要動作を選択します。利用可能な動作は次のとおりです。
 - **Deny new session** - 新しいセッションが要求されてセッション制限に達すると、新しいセッションを作成できなくなります。
 - **Terminate oldest session** - 新しいセッションが要求され、セッション制限に達すると、最も古いセッションが削除され、新しいセッションが作成されます。
10. 必要に応じて、制限に達すると表示されるカスタムエラーメッセージを追加します。

ユーザーセッション制限は、バインドされた **Browser Flow**、**Direct Grant Flow**、**Reset Credentials**、および **Post Broker Login Flow** に追加される必要があることに注意してください。オーセンティケータは、認証中にユーザーがすでに認識されている時点 (通常は認証フローの最後) に追加する必要があり、通常は必須です。ALTERNATIVE と REQUIRED を同じレベルで実行することはできないことに注意してください。

Direct grant flow、**Reset credentials** または **Post broker login flow** などのオーセンティケータのほとんどでは、認証フローの最後に REQUIRED として authenticator を追加することを推奨します。**Reset credentials** フローの例を次に示します。

reset credentials - userSessionLimits Not in use

Steps	Requirement	
Choose User	Required	
Send Reset Email	Required	
Reset Password	Required	
reset credentials - userSessionLimits Reset - Conditional OTP Flow to determine if the OTP should be reset or not. Set to REQUIRED to force.	Conditional	+
Condition - user configured	Required	
Reset OTP	Required	
User session count limiter	Required	

Browser フローの場合は、トップレベルフローにセッション制限オーセンティケータを追加しないことを検討してください。この推奨事項は、SSO Cookie に基づいてユーザーを自動的に再認証する **Cookie** 認証システムによるものです。これは最上位レベルにあり、ユーザーセッションがすでに存在するため、SSO 再認証中にセッション制限をチェックしないことを推奨します。そのため、代わり

に、次の **authenticate-user-with-session-limit** の例のように、別の ALTERNATIVE サブフローを **Cookie** と同じレベルに追加することを検討してください。次に、次の **real-authentication-subflow`example, as a nested subflow of `authenticate-user-with-session-limit** の REQUIRED サブフローを追加し、同じレベルで **User Session Limit** も追加できます。**real-authentication-subflow** 内では、デフォルトのブラウザーフローと同様の方法でリアルオーセンティケーターを追加できます。次のフロー例では、ユーザーがアイデンティティプロバイダーまたはパスワードと OTP を使用して認証できるようにします。

☰ 🗑️ Add step Add sub-flow

Steps	Requirement
☰ Cookie	Alternative
☰ ▼ authenticate-user-with-session-limit	Alternative + ▼ 🗑️
☰ ▼ real-authentication-subflow	Required + ▼ 🗑️
☰ Identity Provider Redirector	Alternative ⚙️ 🗑️
☰ ▼ forms-subflow	Alternative + ▼ 🗑️
☰ Username Password Form	Required 🗑️
☰ OTP Form	Required ▼ 🗑️
☰ User session count limiter	Required ▼ ⚙️ 🗑️

Post Broker login flow に関しては、アイデンティティプロバイダーによる認証後にトリガーする他の認証システムがない限り、認証フローの唯一の認証システムとして **User Session Limits** を追加できます。ただし、このフローがアイデンティティプロバイダーで **Post Broker Flow** として設定されていることを確認してください。この要件は、ID プロバイダーによる認証もセッション制限に参加するために必要です。



注記

現在、管理者は異なる設定間で整合性を維持します。したがって、すべてのフローで同じ **User Session Limits** の設定を使用するようにしてください。



注記

ユーザーセッション制限機能は CIBA では使用できません。

8.5. KERBEROS

Red Hat build of Keycloak は、Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) プロトコルを介した Kerberos チケットによるログインをサポートしています。ユーザーがセッションを認証した後、SPNEGO は Web ブラウザーを介して透過的に認証します。Web 以外の場合、またはログイン中にチケットが利用できない場合、Red Hat build of Keycloak は Kerberos ユーザー名とパスワードを使用したログインをサポートします。

Web 認証の一般的なユースケースは以下のとおりです。

1. ユーザーはデスクトップにログインしています。
2. ユーザーは、ブラウザを使用して Red Hat build of Keycloak が保護する Web アプリケーションにアクセスします。
3. アプリケーションは Red Hat build of Keycloak ログインにリダイレクトします。
4. Red Hat build of Keycloak は、ステータス 401 および HTTP ヘッダー **WWW-Authenticate: Negotiate** の HTML ログイン画面をレンダリングします。
5. ブラウザーにデスクトップログインからの Kerberos チケットがある場合、ブラウザはヘッダー **Authorization: Negotiate 'spnego-token'** で、デスクトップサインオン情報を Red Hat build of Keycloak に転送します。それ以外の場合は、標準のログイン画面が表示され、ユーザーはログイン認証情報を入力します。
6. Red Hat build of Keycloak は、ブラウザからトークンを検証し、ユーザーを認証します。
7. LDAPFederationProvider と Kerberos 認証サポートを使用している場合、Red Hat build of Keycloak は LDAP からユーザーデータをプロビジョニングします。KerberosFederationProvider を使用している場合、ユーザーは Red Hat build of Keycloak でプロフィールの更新とログインデータのプレフィルを行えます。
8. Red Hat build of Keycloak がアプリケーションに戻ります。Red Hat build of Keycloak とアプリケーションは、OpenID Connect または SAML メッセージを通じて通信します。Red Hat build of Keycloak は、Kerberos/SPNEGO ログインのブローカーとして機能します。そのため、Kerberos を介して認証する Red Hat build of Keycloak はアプリケーションから認識されません。



警告

Negotiate `www-authenticate` スキームにより、Kerberos へのフォールバックとして NTLM が可能になり、Windows の一部の Web ブラウザーでは NTLM がデフォルトでサポートされます。`www-authenticate` チャレンジがブラウザの許可リスト外のサーバーから送信された場合、ユーザーに NTLM ダイアログプロンプトが表示される場合があります。Keycloak はこのメカニズムをサポートしていないため、ユーザーはダイアログのキャンセルボタンをクリックして続行する必要があります。この状況は、イントラネット Web ブラウザーが厳密に設定されていない場合、または Keycloak がイントラネットとインターネットの両方でユーザーにサービスを提供している場合に発生する可能性があります。`custom authenticator` を使用すると、ネゴシエートチャレンジをホストのホワイトリストに制限できます。

Kerberos 認証を設定するには、以下の手順を実行します。

1. Kerberos サーバー (KDC) のセットアップと設定。
2. Red Hat build of Keycloak サーバーのセットアップと設定。
3. クライアントマシンのセットアップと設定。

8.5.1. Kerberos サーバーの設定

Kerberos サーバーを設定する手順は、オペレーティングシステム (OS) および Kerberos ベンダーによって異なります。Kerberos サーバーのセットアップおよび設定方法は、Windows Active Directory、MIT Kerberos、および OS のドキュメントを参照してください。

セットアップ時に、以下の手順を実行します。

1. Kerberos データベースにユーザープリンシパルを追加します。Kerberos と LDAP を統合することも可能です。つまり、ユーザーアカウントが LDAP サーバーからプロビジョニングされません。
2. HTTP サービスのサービスプリンシパルを追加します。たとえば、Red Hat build of Keycloak サーバーが **www.mydomain.org** で実行されている場合は、サービスプリンシパル **HTTP/www.mydomain.org@<kerberos realm>** を追加します。
MIT Kerberos では、kadmin セッションを実行します。MIT Kerberos を使用するマシンで、以下のコマンドを使用できます。

```
sudo kadmin.local
```

次に、以下のようなコマンドを使用して、HTTP プリンシパルを追加し、そのキーを keytab ファイルにエクスポートします。

```
addprinc -randkey HTTP/www.mydomain.org@MYDOMAIN.ORG
ktadd -k /tmp/http.keytab HTTP/www.mydomain.org@MYDOMAIN.ORG
```

Red Hat build of Keycloak が実行されているホスト上で、keytab ファイル **/tmp/http.keytab** にアクセスできることを確認します。

8.5.2. Red Hat build of Keycloak サーバーのセットアップと設定

Kerberos クライアントをマシンにインストールします。

手順

1. Kerberos クライアントをインストールします。マシンが Fedora、Ubuntu、または RHEL を実行している場合は、Kerberos クライアントおよびその他のユーティリティーが含まれる [freeipa-client](#) パッケージをインストールします。
2. Kerberos クライアントを設定します (Linux では、設定は [/etc/krb5.conf](#) ファイルにあります)。Kerberos レルムを設定に追加し、サーバー稼働している HTTP ドメインを設定します。

たとえば、MYDOMAIN.ORG レルムの場合は、以下のように **domain_realm** セクションを設定できます。

```
[domain_realm]
.mydomain.org = MYDOMAIN.ORG
mydomain.org = MYDOMAIN.ORG
```

3. HTTP プリンシパルを含む keytab ファイルをエクスポートし、そのファイルに Red Hat build of Keycloak サーバーを実行しているプロセスからアクセスできることを確認します。本番環境では、このプロセスだけがこのファイルを読み取れるようにします。

上記の MIT Kerberos の例では、キータブを `/tmp/http.keytab` ファイルにエクスポートしました。**Key Distribution Center (KDC)** と Red Hat build of Keycloak が同じホスト上で実行されている場合、ファイルはすでに使用可能です。

8.5.2.1. SPNEGO 処理の有効化

デフォルトでは、Red Hat build of Keycloak は SPNEGO プロトコルのサポートを無効にします。有効にするには、[ブラウザーフロー](#) に移動し、**Kerberos** を有効にします。

ブラウザーのフロー

The screenshot shows the 'Authentication > Flow details' page for the 'Browser' flow. The flow is currently 'Default' and 'Built-in'. The configuration table is as follows:

Steps	Requirement
Cookie	Alternative
Kerberos	Disabled
Identity Provider Redirector	Alternative
forms Username, password, otp and other auth forms.	Alternative
Username Password Form	Required
Browser - Conditional OTP Flow to determine if the OTP is required for the authentication	Conditional
Condition - user configured	Required
OTP Form	Required

At the bottom, there are buttons for '+ Add step' and '+ Add sub-flow'.

Kerberos 要件を **disabled** から **alternative**(Kerberos は任意) または **required**(ブラウザーは Kerberos を有効にする必要があります) に設定します。SPNEGO または Kerberos で動作するようにブラウザーを設定していない場合、Red Hat build of Keycloak は通常のログイン画面にフォールバックします。

8.5.2.2. Kerberos ユーザストレージフェデレーションプロバイダーを設定する

[User Storage Federation](#) を使用して、Red Hat build of Keycloak が Kerberos チケットを解釈する方法を設定する必要があります。Kerberos 認証のサポートには、2つの異なるフェデレーションプロバイダーがあります。

LDAP サーバーがサポートする Kerberos で認証するには、[LDAP フェデレーションプロバイダー](#) を設定します。

手順

1. LDAP プロバイダーの設定ページに移動します。

Ldap kerberos の統合

The screenshot shows the 'Synchronization settings' and 'Kerberos integration' sections of the Keycloak administration console. In the 'Synchronization settings' section, 'Import users' is turned on, while 'Batch size', 'Periodic full sync', and 'Periodic changed users sync' are turned off. In the 'Kerberos integration' section, 'Allow Kerberos authentication' and 'Use Kerberos for password authentication' are both turned off. A sidebar on the right lists navigation options, with 'Synchronization settings' highlighted.

2. **Allow Kerberos authentication** を **ON** に切り替えます。

Allow Kerberos authentication では、Red Hat build of Keycloak は Kerberos プリンシパルアクセスユーザー情報を使用するため、情報を Red Hat build of Keycloak 環境にインポートできます。

LDAP サーバーが Kerberos ソリューションをサポートしていない場合は、**Kerberos ユーザストレージフェデレーションプロバイダー**を使用します。

手順

1. メニューの **User Federation** をクリックします。
2. **Add provider** 選択ボックスから **Kerberos** を選択します。

Kerberos ユーザストレージプロバイダー

The screenshot shows the 'Add Kerberos provider' configuration page in the Keycloak administration console. The left sidebar shows the navigation menu with 'User federation' selected. The main content area has the following settings:

- Required Settings:**
 - Console display name: [Empty text input]
 - Kerberos realm: [Empty text input]
 - Server principal: [Empty text input]
 - Key tab: [Empty text input]
- Debug:** Off
- Allow password authentication:** Off
- Update first login:** Off
- Cache settings:**
 - Cache policy: DEFAULT

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

Kerberos プロバイダーは、シンプルなプリンシパル情報のために Kerberos チケットを解析し、情報をローカルの Red Hat build of Keycloak データベースにインポートします。ユーザー名、姓、電子メールなどのユーザープロファイル情報はプロビジョニングされません。

8.5.3. クライアントマシンの設定および設定

クライアントマシンには Kerberos クライアントが必要であり、[上記](#)のように、**krb5.conf** を設定する必要があります。クライアントマシンは、ブラウザの SPNEGO ログインサポートも有効にする必要があります。Firefox ブラウザーを使用している場合は、[configuring Firefox for Kerberos](#) を参照してください。

.mydomain.org URI は **network.negotiate-auth.trusted-uris** 設定オプションになければなりません。

Windows ドメインでは、クライアントは設定を調整する必要はありません。Internet Explorer および Edge は、すでに SPNEGO 認証に参加できます。

8.5.4. 認証情報の委譲

Kerberos は認証情報の委譲をサポートします。アプリケーションは Kerberos チケットを再利用して Kerberos でセキュリティーが保護された他のサービスと対話できるように、チケットへのアクセスが必要になる場合があります。Red Hat build of Keycloak サーバーが SPNEGO プロトコルを処理するため、OpenID Connect トークン要求または SAML アサーション属性内のアプリケーションに GSS 認証情報を反映させる必要があります。Red Hat build of Keycloak はこれを Red Hat build of Keycloak サーバーからアプリケーションに送信します。この要求をトークンまたはアサーションに挿入するには、各アプリケーションはビルトイン プロトコルマッパー **gss delegation credential** を有効にする必要があります。このマッパーは、アプリケーションのクライアントページの **Mappers** タブで利用できます。詳細は、[プロトコルマッパー](#) の章を参照してください。

アプリケーションは、Red Hat build of Keycloak から受信する要求を使用して他のサービスに対して GSS 呼び出しを行う前に、その要求をデシリアライズする必要があります。アクセストークンから GSSCredential オブジェクトに認証情報をデシリアライズする場合は、**GSSManager.createContext** メソッドに渡されるこのクレデンシャルで GSSContext を作成します。以下に例を示します。

```
// Obtain accessToken in your application.
KeycloakPrincipal keycloakPrincipal = (KeycloakPrincipal) servletReq.getUserPrincipal();
AccessToken accessToken = keycloakPrincipal.getKeycloakSecurityContext().getToken();

// Retrieve Kerberos credential from accessToken and deserialize it
String serializedGssCredential = (String) accessToken.getOtherClaims().
    get(org.keycloak.common.constants.KerberosConstants.GSS_DELEGATION_CREDENTIAL);

GSSCredential deserializedGssCredential = org.keycloak.common.util.KerberosSerializationUtils.
    deserializeCredential(serializedGssCredential);

// Create GSSContext to call other Kerberos-secured services
GSSContext context = gssManager.createContext(serviceName, krb5Oid,
    deserializedGssCredential, GSSContext.DEFAULT_LIFETIME);
```



注記

krb5.conf ファイルで **転送可能な** Kerberos チケットを設定し、委譲された認証情報のサポートをブラウザに追加します。



警告

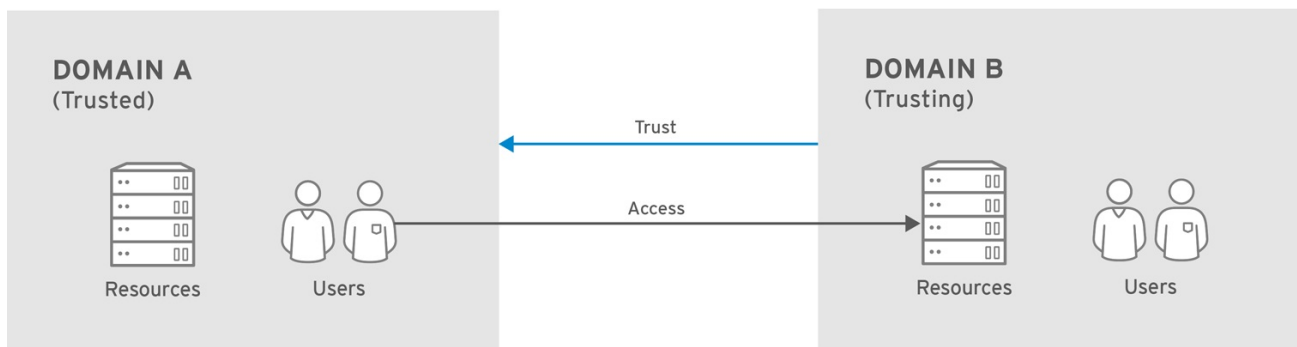
認証情報の委譲はセキュリティに影響するため、必要かつ HTTPS を使用する場合にのみ使用してください。詳細と例については、[この記事](#) を参照してください。

8.5.5. レルム間の信頼

Kerberos プロトコルでは、**レルム** は Kerberos プリンシパルのセットです。これらのプリンシパルの定義は Kerberos データベース (通常は LDAP サーバー) に存在します。

Kerberos プロトコルは、レルム間の信頼を許可します。たとえば、2つの Kerberos レルム A および B が存在する場合は、レルム間の信頼により、レルム A のユーザーがレルム B のリソースにアクセスできるようになります。レルム B がレルム A を信頼します。

Kerberos レルム間の信頼



RHEL_404973_0516

Red Hat build of Keycloak サーバーは、レルム間の信頼をサポートします。これを実装するには、以下を実行します。

- レルム間の信頼用に Kerberos サーバーを設定します。この手順の実装は、Kerberos サーバーの実装によって異なります。この手順は、Kerberos プリンシパル `krbtgt/B@A` をレルム A および B の Kerberos データベースに追加するために必要です。このプリンシパルは両方の Kerberos レルムで同じキーを持つ必要があります。プリンシパルは、両方のレルムで同じパスワード、キーバージョン番号、および暗号を持つ必要があります。詳細は、Kerberos サーバーのドキュメントを参照してください。



注記

レルム間の信頼は、デフォルトで一方向です。レルム A とレルム B 間の双方向の信頼のために、プリンシパル `krbtgt/A@B` を両方の Kerberos データベースに追加する必要があります。ただし、信頼はデフォルトで推移的です。レルム B がレルム A を信頼し、レルム C がレルム B を信頼する場合、レルム C は、プリンシパル `krbtgt/C@A` なしでレルム A を信頼します。クライアントが信頼パスを見つけることができるように、Kerberos クライアント側で追加の設定 (例: `capaths`) が必要になる場合があります。詳細は、Kerberos のドキュメントを参照してください。

- Red Hat build of Keycloak サーバーを設定する

- Kerberos がサポートされる LDAP ストレージプロバイダーを使用する場合は、レルム B のサーバープリンシパルを設定する必要があります (この例では **HTTP/mydomain.com@B**)。Red Hat build of Keycloak は SPNEGO フローを実行してからユーザーを見つける必要があるため、レルム A からのユーザーが Red Hat build of Keycloak で正常に認証されるためには、LDAP サーバーはレルム A からユーザーを見つける必要があります。

ユーザーの検索は、LDAP ストレージプロバイダーオプションの **Kerberos principal attribute** に基づきます。これがたとえば **userPrincipalName** などの値で設定されている場合、ユーザー **john@A** の SPNEGO 認証後、Red Hat build of Keycloak は **john@A** と同等の **userPrincipalName** 属性を持つ LDAP ユーザーを検索しようとします。**Kerberos principal attribute** が空のままの場合、Red Hat build of Keycloak は、レルムを省略した Kerberos プリンシパルの接頭辞に基づき LDAP ユーザーを検索します。たとえば、Kerberos プリンシパルユーザーの **john@A** は、LDAP 内 (通常は **uid=john,ou=People,dc=example,dc=com** などの LDAP DN 配下) でユーザー名 **john** として使用できる必要があります。レルム A および B からのユーザーを認証する場合は、LDAP がレルム A と B 両方からのユーザーを見つけられるようにします。

- Kerberos ユーザーストレージプロバイダー (通常は LDAP 統合のない Kerberos) を使用する場合は、サーバープリンシパルを **HTTP/mydomain.com@B** に設定し、Kerberos レルム A および B からのユーザーを認証する必要があります。

すべてのユーザーは、認証に使用される Kerberos プリンシパルを参照する **KERBEROS_PRINCIPAL** 属性を持ち、これを使用してさらにユーザーを検索するため、複数の Kerberos レルムからのユーザーの認証が許可されています。Kerberos レルム **A** と **B** の両方に **john** というユーザーが存在する場合、競合を回避するために Red Hat build of Keycloak ユーザーのユーザー名には Kerberos レルムが小文字で含まれる場合があります。たとえば、ユーザー名は **john@a** になります。設定された **Kerberos realm** とレルムが一致する場合に備えて、生成されたユーザー名からレルムの接尾辞が省略される場合があります。たとえば、Kerberos プロバイダーで設定された **Kerberos realm** が **A** である限り、Kerberos プリンシパル **john@A** のインスタンスユーザー名は **john** になります。

8.5.6. トラブルシューティング

問題がある場合は、追加のロギングを有効にして問題をデバッグしてください。

- Kerberos または LDAP フェデレーションプロバイダーについて、管理コンソールで **Debug** フラグを有効にします。
- カテゴリー **org.keycloak** に対して TRACE ロギングを有効にして、サーバーログで詳細情報を受け取る
- システムプロパティー **-Dsun.security.krb5.debug=true** および **-Dsun.security.spnego.debug=true** を追加します。

8.6. X.509 クライアント証明書ユーザー認証

相互 SSL 認証を使用するようにサーバーを設定した場合、Red Hat build of Keycloak は、X.509 クライアント証明書を使用したログインをサポートします。

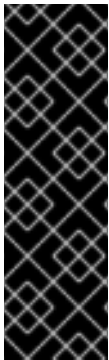
通常のワークフロー:

- クライアントは SSL/TLS チャンネルで認証リクエストを送信します。
- SSL/TLS ハンドシェイクの間、サーバーとクライアントは x.509/v3 証明書を交換します。
- コンテナー (JBoss EAP) は、証明書 PKIX パスと証明書の有効期限を検証します。

- X.509 クライアント証明書オーセンティケーターは、以下の方法を使用してクライアント証明書を検証します。
 - CRL または CRL Distribution Points を使用して、証明書失効ステータスを確認します。
 - OCSP (Online Certificate Status Protocol) を使用して、証明書失効ステータスを確認します。
 - 証明書の鍵が予想される鍵と一致するかどうかを確認します。
 - 証明書の拡張鍵が、想定された拡張鍵と一致するかどうかを検証します。
- これらのチェックのいずれかが失敗すると、x.509 認証は失敗します。それ以外の場合は、オーセンティケーターは証明書のアイデンティティを抽出し、これを既存ユーザーにマッピングします。

証明書が既存のユーザーにマッピングされると、認証フローに応じてさまざまな動作が行われます。

- Browser Flow では、サーバーはユーザーに対してアイデンティティを確認するか、ユーザー名とパスワードを使用してサインインするよう要求します。
- Direct Grant Flow では、サーバーはユーザーにサインインします。



重要

証明書 PKIX パスを検証するのは Web コンテナのロールであることに注意してください。Red Hat build of Keycloak 側の X.509 オーセンティケーターは、証明書の有効期限、証明書失効ステータス、およびキーの使用を確認する追加のサポートのみを提供します。リバースプロキシの背後にデプロイされた Red Hat build of Keycloak を使用している場合は、リバースプロキシが PKIX パスを検証するように設定されていることを確認します。リバースプロキシを使用せず、ユーザーが JBoss EAP に直接アクセスする場合は、以下のように PKIX パスが設定されている限り、JBoss EAP が PKIX パスが検証された状態を維持するので問題ありません。

8.6.1. 機能

サポートされる証明書アイデンティティソース:

- 正規表現を使用した SubjectDN の一致
- X500 サブジェクトの email 属性
- Subject Alternative Name Extension からの X500 サブジェクトの email (RFC822Name General Name)
- サブジェクトの別名エクステンションの X500 サブジェクトの他の名前。通常、この他の名前は User Principal Name (UPN) です。
- X500 サブジェクトのコモンネーム属性
- 正規表現を使用した IssuerDN の一致
- 証明書のシリアル番号
- Certificate Serial Number および IssuerDN
- SHA-256 証明書サムプリント

- PEM 形式の完全な証明書

8.6.1.1. 正規表現

Red Hat build of Keycloak は、正規表現をフィルターとして使用して、サブジェクト DN または発行者 DN から証明書のアイデンティティを抽出します。たとえば、以下の正規表現は email 属性とマッチします。

```
emailAddress=(.*?)(?:;|$)
```

正規表現のフィルターは、**Identity Source** が、**Match SubjectDN using regular expression** または **Match IssuerDN using regular expression** のいずれかに設定されている場合にのみ適用されます。

8.6.1.1.1. 既存のユーザーへの証明書 ID のマッピング

証明書アイデンティティマッピングは、抽出したユーザー ID を、既存のユーザーのユーザー名、メール、または値が証明書 ID とマッチするカスタム属性にマッピングできます。たとえば、**Identity source** を **Subject's email** に設定するか、**User mapping method** を **Username or email** に設定すると、X.509 クライアント証明書オーセンティケーターは、ユーザー名または電子メールで既存のユーザーを検索するときに、証明書の Subject DN の email 属性を検索条件として使用します。

重要

- レルム設定で **Login with email** を無効にすると、同じルールが証明書認証に適用されます。ユーザーは email 属性を使用してログインできません。
- **Certificate Serial Number and IssuerDN** を ID ソースとして使用するには、シリアル番号と IssuerDN に 2 つのカスタム属性が必要です。
- **SHA-256 Certificate thumbprint** は、SHA-256 証明書のサムプリントの小文字の 16 進数表記です。
- ID ソースとしての **Full certificate in PEM format** の使用は、LDAP などの外部フェデレーションソースにマッピングされたカスタム属性に限定されます。Red Hat build of Keycloak は、長さの制限によりデータベースに証明書を保存できません。そのため、LDAP の場合は、**Always Read Value From LDAP** を有効にする必要があります。

8.6.1.1.2. 拡張証明書の検証

- CRL を使用した失効ステータスの確認。
- CRL/分散ポイントを使用した失効ステータスの確認。
- OCSP/Responder URI を使用した失効ステータスの確認。
- 証明書 KeyUsage の検証。
- 証明書 ExtendedKeyUsage の検証。

8.6.2. ブラウザーフローへの X.509 クライアント証明書認証の追加

1. メニューで **Authentication** をクリックします。
2. **Browser** フローをクリックします。

3. **Action** リストから **Duplicate** を選択します。
4. コピーの名前を入力します。
5. **複製** をクリックします。
6. **Add step** をクリックします。
7. X509/Validate Username Form をクリックします。
8. **Add** をクリックします。

X509 実行

Add step to Copy of browser



1 - 10 ▾



- Browser Redirect for Cookie free authentication
Perform a 302 redirect to get user agent's current URI on authenticate path with an auth_session_id query parameter. This is for client's that do not support cookies.
- Cookie
Validates the SSO cookie set by the auth server.
- Username Password Challenge
Proprietary challenge protocol for CLI clients that queries for username password
- Choose User
Choose a user to reset credentials for
- Password
Validates the password supplied as a 'password' form parameter in direct grant request
- WebAuthn Authenticator
Authenticator for WebAuthn. Usually used for WebAuthn two-factor authentication
- Kerberos
Initiates the SPNEGO protocol. Most often used with Kerberos.
- Reset Password
Sets the Update Password required action if execution is REQUIRED. Will also set it if execution is OPTIONAL and the password is currently configured for it.
- X509/Validate Username
Validates username and password from X509 client certificate received as a part of mutual SSL handshake.
- Password Form
Validates a password from login form.
- Docker Authenticator
Uses HTTP Basic authentication to validate docker users, returning a docker error token on auth failure

1 - 10 ▾



Add

Cancel

9. X509/Validate Username Form をクリックし、Browser Forms 実行の上にドラッグします。
10. 要件を ALTERNATIVE に設定します。

X509 ブラウザーフロー

The screenshot shows the 'X.509 Browser' flow configuration in Keycloak. The left sidebar contains the navigation menu with 'Authentication' selected. The main area displays the flow details, including a list of steps and their requirements.

Steps	Requirement
Cookie	Alternative
Kerberos	Alternative
Identity Provider Redirector	Alternative
X509/Validate Username Form	Alternative
X.509 Browser forms Username, password, otp and other auth forms.	Alternative

11. Action メニューをクリックします。
12. Bind flow をクリックします。
13. ドロップダウンリストから Browser flow をクリックします。
14. Save をクリックします。

X509 ブラウザーフローバイন্ディング

Docker auth	Built-in	✓ Docker auth	Used by Docker clients to authenticate against the IDP
X.509 Browser		✓ Browser flow	browser based authentication
Client-flow		✓ Specific clients	

8.6.3. X.509 クライアント証明書認証の設定

X509 設定

X509/Validate Username Form config

Alias * ?

User Identity Source ?

Canonical DN representation enabled ?

 Off

Enable Serial Number hexadecimal representation ?

Off

A regular expression to extract user identity ?

(.*?)(?:\$)

User mapping method ?

Custom Attribute Mapper ▼

A name of user attribute ?

⊖

+ Add a name of user attribute

Check certificate validity ?

On

CRL Checking Enabled ?

Off

Enable CRL Distribution Point to check certificate revocation status ?

Off

CRL Path ?

⊖

+ Add crl path

OCSP Checking Enabled ?

Off

OCSP Fail-Open Behavior ?

Off

OCSP Responder Uri ?

User Identity Source

クライアント証明書からユーザーアイデンティティを抽出する方法を定義します。

Canonical DN representation enabled

正規の形式を使用して識別名を判断するかどうかを定義します。公式の [Java API ドキュメント](#) で形式が説明されています。このオプションは、2つのユーザーアイデンティティソース **Match SubjectDN using regular expression** および **Match IssuerDN using regular expression** にのみ影響します。新しい Red Hat build of Keycloak インスタンスを設定するときにこのオプションを有効にします。既存の Red Hat build of Keycloak インスタンスとの後方互換性を維持するには、このオプションを無効にします。

Enable Serial Number hexadecimal representation

シリアル番号を16進数で表します。符号ビットに1が設定されているシリアル番号は、00 オクテットを左に追加する必要があります。たとえば、10進値が161のシリアル番号、または16進表現のa1は、RFC5280に従って00a1としてエンコードされます。詳細は、[RFC5280, appendix-B](#)を参照してください。

A regular expression

証明書IDを抽出するフィルターとして使用する正規表現。表現には1つのグループを含める必要があります。

User Mapping Method

証明書IDを既存ユーザーに一致させる方法を定義します。**Username or email**は、ユーザー名またはメールアドレスで既存のユーザーを検索します。**Custom Attribute Mapper**は、証明書IDと一致するカスタム属性を持つ既存のユーザーを検索します。カスタム属性の名前は設定可能です。

A name of user attribute

値が証明書アイデンティティと照合されるカスタム属性。属性マッピングが複数の値に関連する場合は、複数のカスタム属性を使用します (例:Certificate Serial Number and IssuerDN)。

CRL Checking Enabled

Certificate Revocation List を使用して、証明書の失効ステータスを確認します。リストの場所は、**CRL file path**属性で定義されます。

Enable CRL Distribution Point to check certificate revocation status

CDP を使用して、証明書失効ステータスを確認します。ほとんどのPKI認証局には、証明書にCDPが含まれます。

CRL file path

CRL リストが含まれるファイルへのパス。**CRL Checking Enabled** オプションが有効になっている場合、値は有効なファイルへのパスである必要があります。

OCSP Checking Enabled

Online Certificate Status Protocol を使用して、証明書失効ステータスを確認します。

OCSP Fail-Open Behavior

デフォルトでは、認証を成功させるために、OCSP チェックは肯定応答を返す必要があります。ただし、このチェックが決定的でない場合があります。たとえば、OCSP サーバーに到達できない、過負荷になっている、またはクライアント証明書にOCSP レスポンダー URI が含まれていない可能性があります。この設定をオンにすると、OCSP レスポンダーが明示的な否定応答を受信し、証明書が確実に取り消された場合にのみ、認証が拒否されます。有効なOCSP 応答がない場合は、認証試行が許可されます。

OCSP Responder URI

証明書の OCSP レスポンダー URI の値を上書きします。

Validate Key Usage

証明書の KeyUsage 拡張ビットが設定されていることを検証します。たとえば、digitalSignature、KeyEncipherment は、KeyUsage 拡張のビット 0 と 2 が設定されているかどうかを検証します。Key Usage の検証を無効にするには、このパラメーターを空欄のままにします。詳細は、[RFC5280, Section-4.2.1.3](#) を参照してください。Red Hat build of Keycloak では、キーの使用が一致しないとエラーが発生します。

Validate Extended Key Usage

Extended Key Usage 拡張で定義された 1 つまたは複数の目的を検証します。詳細は、[RFC5280, Section-4.2.1.12](#) を参照してください。Extended Key Usage の検証を無効にするには、このパラメーターを空欄のままにします。Red Hat build of Keycloak では、発行元 CA が重要としてフラグを付け、キーの使用の拡張が一致しない場合、エラーが発生します。

証明書ポリシーの検証

証明書ポリシー拡張で定義されている 1 つ以上のポリシー OID を確認します。[RFC5280 のセクション-4.2.1.4](#) を参照してください。証明書要求の検証を無効にするには、パラメーターを空のままにします。複数のポリシーはコンマで区切る必要があります。

証明書ポリシーの検証モード

Validate Certificate Policy 設定で複数のポリシーが指定されている場合、要求されたすべてのポリシーが存在するかどうかを照合で確認するか、認証を成功させるには 1 つの照合で十分かを判断します。デフォルト値は **All** です。つまり、要求されたポリシーすべてがクライアント証明書に存在する必要があることを意味します。


Bypass identity confirmation

有効にすると、X.509 クライアント証明書認証は、証明書 ID を確認するようにユーザーに要求しません。Red Hat build of Keycloak は、認証が成功するとユーザーをサインインします。

Revalidate client certificate

設定された場合、クライアント証明書のトラストチェーンは、設定されたトラストストアにある証明書を使用して、アプリケーションレベルで常に検証されます。これは、基礎となる Web サーバーがクライアント証明書チェーンの検証を強制しない場合に便利です (非検証のロードバランサーやリバースプロキシの背後にある場合や、相互 SSL ネゴシエーションについて許可される CA の数が大きすぎる場合など (ほとんどのブラウザは最大の SSL ネゴシエーションパケットのサイズを 32767 バイトに制限します。これは、約 200 の広告済み CA に対応します)。デフォルトでは、このオプションは無効です。

8.6.4. X.509 クライアント証明書認証の Direct Grant Flow への追加

1. メニューで **Authentication** をクリックします。
2. アクションリストから **Duplicate** を選択して、組み込みの直接付与フローのコピーを作成します。
3. コピーの名前を入力します。
4. **複製** をクリックします。
5. 作成したフローをクリックします。
6. ユーザー名の検証のゴミ箱アイコン  をクリックし、**Delete** をクリックします。
7. パスワードのゴミ箱アイコン  をクリックし、**Delete** をクリックします。
8. **Add step** をクリックします。

9. X509/Validate Username をクリックします。

10. **Add** をクリックします。

X509 直接付与実行

Add step to Copy of direct grant

✕

11 - 20 ▾

◀ ▶

- Docker Authenticator
Uses HTTP Basic authentication to validate docker users, returning a docker error token on auth failure
- Username Password Form for identity provider reauthentication
Validates a password from login form. Username may be already known from identity provider authentication
- Allow access
Authenticator will always successfully authenticate. Useful for example in the conditional flows to be used after satisfying the previous conditions
- Verify existing account by Email
Email verification of existing Keycloak user, that wants to link his user account with identity provider
- Automatically set existing user
Automatically set existing user to authentication context without any verification
- X509/Validate Username Form
Validates username and password from X509 client certificate received as a part of mutual SSL handshake.
- Basic Auth Challenge
Challenge-response authentication using HTTP BASIC scheme.
- Deny access
Access will be always denied. Useful for example in the conditional flows to be used after satisfying the previous conditions
- Identity Provider Redirector
Redirects to default Identity Provider or Identity Provider specified with kc_idp_hint query parameter
- Username Validation
Validates the username supplied as a 'username' form parameter in direct grant request
- Reset OTP
Sets the Configure OTP required action.

11 - 20 ▾

◀ ▶

Add

Cancel

11. [x509 ブラウザーフロー](#) セクションで説明されている手順に従って、x509 認証設定をセットアップします。

12. **Bindings** タブをクリックします。

13. **Direct Grant Flow** ドロップダウンリストをクリックします。
14. 新規作成された x509 Direct Grant フローをクリックします。
15. **Save** をクリックします。

X509 直接付与フローバインディング

X509 Direct grant

✔ Direct grant flow

OpenID Connect Resource Owner Grant

8.7. W3C WEB AUTHENTICATION (WEBAUTHN)

Red Hat build of Keycloak は、[W3C Web Authentication \(WebAuthn\)](#) をサポートします。Red Hat build of Keycloak は、WebAuthn の [Relying Party \(RP\)](#) として機能します。



注記

WebAuthn 操作が成功するかどうかは、オーセンティケーター、ブラウザ、およびプラットフォームをサポートするユーザーの WebAuthn によります。オーセンティケーター、ブラウザ、およびプラットフォームが WebAuthn 仕様をサポートしていることを確認してください。

8.7.1. 設定


2FA の WebAuthn サポートの設定手順は、以下のようになります。

8.7.1.1. WebAuthn オーセンティケーター登録の有効化

1. メニューで **Authentication** をクリックします。
2. **Required Actions** タブをクリックします。
3. **Webauthn Register** スイッチを **ON** に切り替えます。

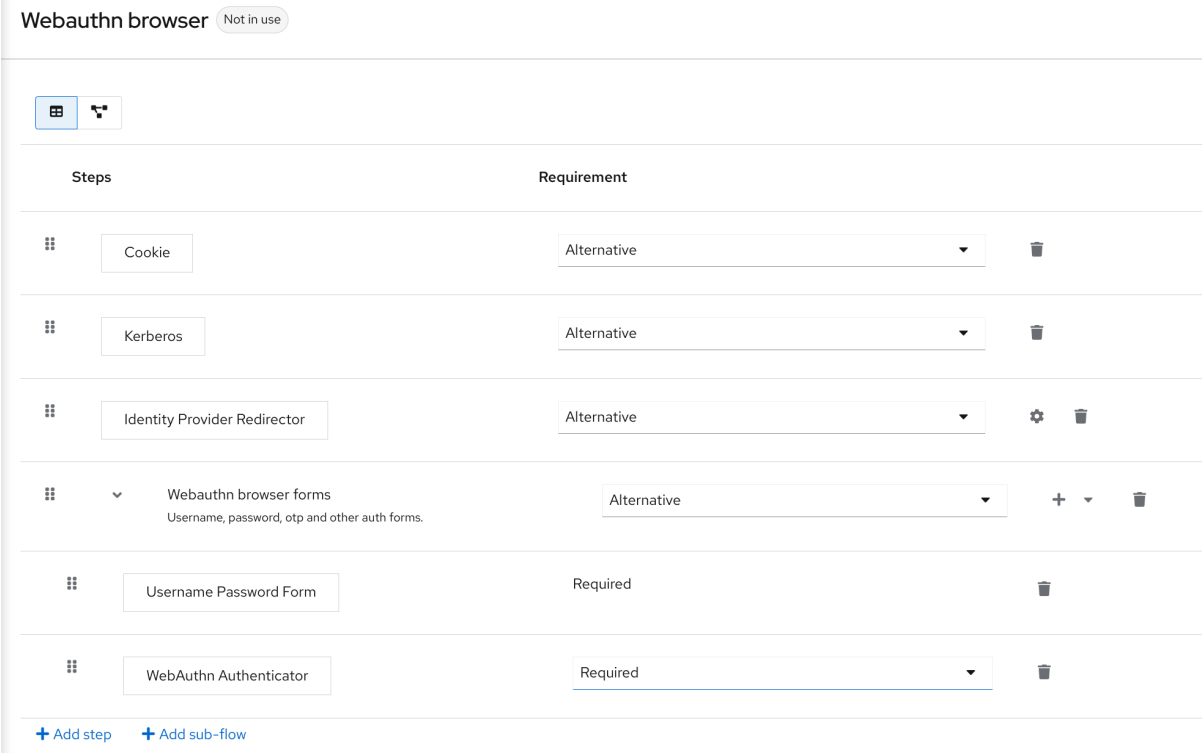
すべての新規ユーザーに WebAuthn 認証情報の登録を要求する場合は、**Default Action** スイッチを **ON** に切り替えます。

8.7.2. WebAuthn 認証のブラウザーフローへの追加

1. メニューで **Authentication** をクリックします。
2. **Browser** フローをクリックします。
3. アクションリストから **Duplicate** を選択して、組み込みの **Browser** フローのコピーを作成します。
4. コピーの名前として WebAuthn Browser と入力します。
5. **複製** をクリックします。
6. 名前をクリックすると詳細へ移動します
7. WebAuthn Browser - Conditional OTP のゴミ箱アイコン  をクリックし、**Delete** をクリックします。

全ユーザーの WebAuthn が必要な場合は、以下を実行します。

1. **WebAuthn ブラウザーフォーム** の + メニューをクリックします。
2. **Add step** をクリックします。
3. **WebAuthn Authenticator** をクリックします。
4. **Add** をクリックします。
5. **WebAuthn Authenticator** 認証タイプで **Required** を選択し、その要件を必須に設定します。

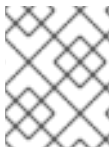


Webauthn browser Not in use

Steps	Requirement
Cookie	Alternative
Kerberos	Alternative
Identity Provider Redirector	Alternative
Webauthn browser forms Username, password, otp and other auth forms.	Alternative
Username Password Form	Required
WebAuthn Authenticator	Required

+ Add step + Add sub-flow

6. 画面上部の **Action** メニューをクリックします。
7. ドロップダウンリストから **Bind flow** を選択します。
8. ドロップダウンリストから **Browser** を選択します。
9. **Save** をクリックします。



注記

ユーザーに WebAuthn 認証情報がない場合、ユーザーは WebAuthn 認証情報を登録する必要があります。

WebAuthn 認証情報が登録されている場合に限り、ユーザーは WebAuthn でログインできます。そのため、**WebAuthn Authenticator** 実行を追加する代わりに、以下を実行できます。

手順

1. **WebAuthn Browser Forms** 行の + メニューをクリックします。
2. **Add sub-flow** をクリックします。

3. **name** フィールドに Conditional 2FA を入力します。
4. **Conditional 2FA** の **Conditional** を選択して、その要件を条件付きに設定します。
5. **Conditional 2FA** 行で、プラス記号 + をクリックし、**Add condition** を選択します。
6. **Add condition** をクリックします。
7. **Condition - User Configured** をクリックします。
8. **Add** をクリックします。
9. **Condition - User Configured** で **Required** を選択し、その要件を必須に設定します。
10. **WebAuthn Authenticator** を **Conditional 2FA** フローにドラッグアンドドロップします。
11. **WebAuthn Authenticator** の **Alternative** を選択して、その要件を代替に設定します。

Steps	Requirement
Cookie	Alternative
Kerberos	Alternative
Identity Provider Redirector	Alternative
Webauthn browser forms Username, password, otp and other auth forms.	Alternative
Username Password Form	Required
Conditional 2FA	Conditional
Condition - user configured	Required
WebAuthn Authenticator	Alternative

+ Add step + Add sub-flow

ユーザーは、2つ目の要素に WebAuthn または OTP のいずれかを使用することを選択できます。

手順

1. **Conditional 2FA** 行でプラス記号 + をクリックし、**Add step** を選択します。
2. リストから **OTP Form** を選択します。
3. **Add** をクリックします。
4. **OTP Form** の **Alternative** を選択して、その要件を代替に設定します。

Steps	Requirement
Cookie	Alternative
Kerberos	Alternative
Identity Provider Redirector	Alternative
Webauthn browser forms Username, password, otp and other auth forms.	Alternative
Username Password Form	Required
Conditional 2FA	Conditional
Condition - user configured	Required
WebAuthn Authenticator	Alternative
OTP Form	Alternative

+ Add step + Add sub-flow

8.7.3. WebAuthn オーセンティケーターを使用した認証

WebAuthn オーセンティケーターを登録した後に、ユーザーは以下の操作を実行します。

- ログインフォームを開きます。ユーザーは、ユーザー名とパスワードで認証する必要があります。
- ユーザーのブラウザーは、WebAuthn オーセンティケーターを使用して認証することをユーザーに要求します。

8.7.4. 管理者として WebAuthn の管理

8.7.4.1. 認証情報の管理

Red Hat build of Keycloak は、[ユーザー認証情報の管理](#) からの他の認証情報と同様に、WebAuthn 認証情報を管理します。

- Red Hat build of Keycloak は、**Reset Actions** の一覧から WebAuthn 認証情報を作成するために必要なアクションをユーザーに割り当て、**Webauthn Register** を選択します。
- 管理者は **Delete** をクリックして WebAuthn 認証情報を削除できます。
- 管理者は、**Show data...** を選択して、AAGUID などの認証情報のデータを表示することができます。
- 管理者は、**User Label** フィールドに値を設定し、データを保存することで、認証情報のラベルを設定できます。

8.7.4.2. ポリシーの管理

管理者は、WebAuthn 関連の操作をレلمごとに **WebAuthn Policy** として設定できます。

手順

1. メニューで **Authentication** をクリックします。
2. **Policy** タブをクリックします。
3. **WebAuthn Policy** タブをクリックします。
4. ポリシー内で項目を設定します (以下の説明を参照してください)。
5. **Save** をクリックします。

設定可能な項目とその説明は以下のとおりです。

設定	説明
エンティティ名の使用	WebAuthn Relying Party として読み取り可能なサーバー名。この項目は必須であり、WebAuthn オーセンティケーターの登録に適用されます。デフォルト設定は keycloak です。詳細は、 WebAuthn Specification を参照してください。
署名アルゴリズム	アルゴリズム。 公開鍵認証情報 に使用する署名アルゴリズムを WebAuthn オーセンティケーターに指示します。Red Hat build of Keycloak は、公開鍵認証情報を使用して Authentication Assertions に署名し、検証します。アルゴリズムが存在しない場合は、デフォルトの ES256 が適合されます。ES256 は、WebAuthn オーセンティケーターの登録に適用されるオプションの設定項目です。詳細は、 WebAuthn Specification を参照してください。
パート ID の使用	公開鍵認証情報 のスコープを決定する WebAuthn Relying Party の ID。ID は、オリジンの有効なドメインでなければなりません。この ID は、WebAuthn オーセンティケーターの登録に適用されるオプションの設定項目です。このエントリーが空白の場合、Red Hat build of Keycloak は Red Hat build of Keycloak のベース URL のホスト部分を適応させます。詳細は、 WebAuthn の仕様 を参照してください。
証明の伝達設定	ブラウザでの WebAuthn API 実装 (WebAuthn Client) は、Attestation ステートメントを生成するのに推奨される方法です。この設定は、WebAuthn オーセンティケーターの登録に適用されるオプションの設定項目です。オプションが存在しない場合、その動作は none の選択と同じになります。詳細は、 WebAuthn の仕様 を参照してください。

設定	説明
オーセンティケーター添付	WebAuthn Client に対する WebAuthn オーセンティケーターの許容割り当てパターン。このパターンは、WebAuthn オーセンティケーターの登録に適用されるオプションの設定項目です。詳細は、 WebAuthn Specification を参照してください。
識別キーが必要	WebAuthn オーセンティケーターが クライアント側の公開鍵認証情報ソース として公開鍵認証情報を生成することを要求するオプション。このオプションは、WebAuthn オーセンティケーターの登録に適用されます。空欄のままにすると、その動作は No の選択と同じになります。詳細は、 WebAuthn の仕様 を参照してください。
ユーザー検証要件	WebAuthn オーセンティケーターがユーザーの検証を確認することを要求するオプション。これは、WebAuthn オーセンティケーターの登録と、WebAuthn オーセンティケーターによるユーザーの認証に適用される任意の設定項目です。オプションが存在しない場合、その動作は preferred の選択と同じになります。詳細は、 WebAuthn Specification for registering a WebAuthn authenticator および WebAuthn Specification for authenticating the user by a WebAuthn authenticator を参照してください。
タイムアウト	WebAuthn オーセンティケーターを登録し、WebAuthn オーセンティケーターを使用してユーザーを認証する際のタイムアウト値 (秒単位)。ゼロに設定すると、その動作は WebAuthn オーセンティケーターの実装により異なります。デフォルト値は 0 です。詳細は、 WebAuthn Specification for registering a WebAuthn authenticator および WebAuthn Specification for authenticating the user by a WebAuthn authenticator を参照してください。
同じオーセンティケーター登録の回避	有効にすると、Red Hat build of Keycloak は、すでに登録されている WebAuthn オーセンティケーターを再登録できません。
許可される AAGUID	WebAuthn オーセンティケーターが登録する必要がある AAGUID のホワイトリスト。

8.7.5. 証明ステートメントの検証

WebAuthn オーセンティケーターを登録すると、Red Hat build of Keycloak は、WebAuthn オーセンティケーターが生成した証明ステートメントの信頼性を検証します。Red Hat build of Keycloak では、トラストアンカーの証明書を [トラストストア](#) にインポートする必要があります。

この検証を省略するには、このトラストストアを無効にするか、WebAuthn ポリシーの設定項目 Attestation Conveyance Preference を none に設定します。

8.7.6. ユーザーとして WebAuthn 認証情報の管理

8.7.6.1. WebAuthn オーセンティケーターの登録

WebAuthn オーセンティケーターの適切な登録方法は、ユーザーが Red Hat build of Keycloak にアカウントを登録しているかどうかにより異なります。

8.7.6.2. 新規ユーザー

レールムで **WebAuthn Register** の必須操作が **Default Action** の場合、新規ユーザーは初回ログインの後に WebAuthn セキュリティーキーを設定する必要があります。

手順

1. ログインフォームを開きます。
2. **Register** をクリックします。
3. フォームの項目に入力します。
4. **Register** をクリックします。

登録に成功すると、ブラウザーは、ユーザーに対して WebAuthn オーセンティケーターのラベルのテキストを入力するよう要求します。

8.7.6.3. 既存ユーザー

最初の例のように **WebAuthn Authenticator** が必要に応じて設定されている場合、既存のユーザーがログインを試みる際に、WebAuthn オーセンティケーターを自動的に登録する必要があります。

手順

1. ログインフォームを開きます。
2. フォームの項目に入力します。
3. **Save** をクリックします。
4. **Login** をクリックします。

登録に成功すると、ユーザーのブラウザーは、ユーザーに対して WebAuthn オーセンティケーターのラベルのテキストを入力するよう要求します。

8.7.7. パスワードなしの WebAuthn と 2 つのファクターの組み合わせ

Red Hat build of Keycloak は、2 要素認証に WebAuthn を使用しますが、第一要素認証として WebAuthn を使用できません。この場合、**passwordless** の WebAuthn 認証情報を持つユーザーは、パスワードなしで Red Hat build of Keycloak に対して認証できます。Red Hat build of Keycloak では、レールムおよび単一の認証フローのコンテキストで、パスワードレス認証および 2 要素認証のメカニズムとして WebAuthn を使用できます。

管理者は、通常、WebAuthn パスワードレス認証用にユーザーが登録するセキュリティキーが異なる

要件を満たすことを要求します。たとえば、ユーザーは、PIN を使用してセキュリティーキーに対して認証することが要求される場合があります。あるいは、セキュリティーキーを強力な認証局で認証する必要があります。

このため、Red Hat build of Keycloak では、管理者は個別の **WebAuthn Passwordless Policy** を設定できます。必須の **Webauthn Register Passwordless** アクションタイプと、別の **WebAuthn Passwordless Authenticator** オーセンティケータータイプがあります。

8.7.7.1. 設定

以下のように WebAuthn パスワードレスサポートを設定します。

1. すでに存在する場合は、WebAuthn パスワードレスサポートに新しい必須アクションを登録します。[WebAuthn Authenticator 登録を有効にする](#) で説明されている手順を使用します。**Webauthn Register Passwordless** アクションを登録します。
2. ポリシーを設定します。[ポリシーの管理](#) で説明されている手順と設定オプションを使用できます。管理コンソールの **WebAuthn Passwordless Policy** タブで、設定を実行します。通常、セキュリティーキーの要件は、2 要素ポリシーよりもより強力になります。たとえば、パスワードレスポリシーの設定時に、**User Verification Requirement** を **Required** に設定できます。
3. 認証フローを設定します。[WebAuthn 認証をブラウザーフローに追加する](#) で説明されている **WebAuthn ブラウザー** フローを使用します。以下のようにフローを設定します。
 - **WebAuthn Browser Forms** サブフローには、最初のオーセンティケーターとして **Username Form** が含まれます。デフォルトの **Username Password Form** オーセンティケーターを削除し、**Username Form** オーセンティケーターを追加します。このアクションでは、ユーザーに最初のステップとしてユーザー名を提供することを要求します。
 - 必須のサブフローがある場合があります (例: **Passwordless Or Two-factor**)。このサブフローは、ユーザーが Passwordless WebAuthn 認証情報または Two-factor 認証で認証できることを示しています。
 - フローには、第一の代替として **WebAuthn Passwordless Authenticator** が含まれます。
 - 2 つ目の代替は、**Password And Two-factor Webauthn** (例) という名前のサブフローです。このサブフローには、**Password Form** および **WebAuthn Authenticator** が含まれます。

フローの最終的な設定は以下のようになります。

PasswordLess フロー

Authentication > Flow details

Webauthn browser Not in use

Steps	Requirement	
Cookie	Alternative	
Kerberos	Alternative	
Identity Provider Redirector	Alternative	
Webauthn browser forms Username, password, otp and other auth forms.	Alternative	+
Username Password Form	Required	
Passwordless Or Two-factor	Required	+
WebAuthn Passwordless Authenticator	Alternative	
Password And Two-factor Webauthn	Alternative	+
Password Form	Required	
WebAuthn Authenticator	Required	

+ Add step + Add sub-flow

これをテストするために、Red Hat build of Keycloak ですでに認識されているユーザーに必須アクションとして **WebAuthn Register Passwordless** を追加できるようになりました。第一の認証時に、ユーザーはパスワードおよび第二要素の WebAuthn 認証情報を使用する必要があります。WebAuthn Passwordless 認証情報を使用する場合、ユーザーはパスワードおよび第二要素の WebAuthn 認証情報を提供する必要はありません。

8.7.8. LoginLess WebAuthn

Red Hat build of Keycloak は、2 要素認証に WebAuthn を使用しますが、第一要素認証として WebAuthn を使用できます。この場合、**passwordless** の WebAuthn 認証情報を持つユーザーは、ログインやパスワードなしで Red Hat build of Keycloak に対して認証できます。Red Hat build of Keycloak は、レルムのコンテキストで、ログインレス/パスワードレスおよび 2 要素認証メカニズムとして WebAuthn を使用できます。

管理者は、通常、WebAuthn ログインレス認証用にユーザーが登録するセキュリティーキーが異なる要件を満たすことを要求します。ログインレス認証では、ユーザーがセキュリティーキーに対して認証する必要があります (たとえば、PIN コードまたは指紋を使用して)、ログインレスクレデンシャルに関連付けられた暗号化キーがセキュリティーキーに物理的に保存されている必要があります。すべてのセキュリ

ティーキーがそのような要件を満たしているわけではありません。デバイスがユーザー検証および常駐キーをサポートしているかどうかをセキュリティーキーベンダーに確認してください。[サポートされているセキュリティーキー](#)を参照してください。

Red Hat build of Keycloak により、管理者はログインレス認証を可能にするように **WebAuthn Passwordless Policy** を設定できます。ログインレス認証は、**WebAuthn Passwordless Policy** と **WebAuthn Passwordless** クレデンシャルでのみ設定できることに注意してください。WebAuthn ログインレス認証と WebAuthn パスワードレス認証は同じレルムで設定できますが、同じポリシー **WebAuthn Passwordless Policy** ポリシーを共有します。

8.7.8.1. 設定

手順

以下のように WebAuthn ログインレスサポートを設定します。

- すでに存在する場合は、WebAuthn パスワードレスサポートに新しい必須アクションを登録します。[WebAuthn Authenticator 登録を有効にする](#) で説明されている手順を使用します。**Webauthn Register Passwordless** アクションを登録します。
- WebAuthn Passwordless Policy** を設定します。Admin Console の **Authentication** セクションの **Policies** → **WebAuthn Passwordless Policy** タブで設定を実行します。ログインレスシナリオのポリシーを設定するときは、**User Verification Requirement** を **required** に、**Require Resident Key** を **Yes** に設定する必要があります。専用のログインレスポリシーがないため、認証シナリオをユーザー verification=no/resident key=no および loginless のシナリオと混在させることはできません(ユーザー verification=yes/resident key=yes)。ストレージ容量は通常、セキュリティーキーによって非常に制限されています。つまり、多くの常駐キーをセキュリティーキーに保存できません。
- 認証フローを設定します。新しい認証フローを作成し、WebAuthn Passwordless 実行を追加して、実行の Requirement 設定を **Required** に設定します

フローの最終的な設定は以下のようになります。

LoginLess フロー

Steps	Requirement
WebAuthn Passwordless Authenticator	Required

+ Add step + Add sub-flow

これをテストするために、Red Hat build of Keycloak ですでに認識されているユーザーに必須アクションとして **WebAuthn Register Passwordless** を追加できるようになりました。必須アクションが設定されているユーザーは、(たとえば、ユーザー名/パスワードを使用して) 認証する必要があり、ログインレス認証に使用するセキュリティーキーを登録するように求められます。

8.7.8.2. ベンダー固有のマーク

8.7.8.2.1. 互換性チェックリスト

Red Hat build of Keycloak を使用したログインレス認証には、次の機能を満たすセキュリティーキーが必要です

- FIDO2 コンプライアンス: FIDO/U2F と混同しないでください
- ユーザー検証: セキュリティーキーがユーザーを認証する機能 (誰かがセキュリティーキーを見つけてログインなしおよびパスワードなしで認証できるようにする)
- 常駐キー: クライアントアプリケーションに関連付けられたログインキーと暗号化キーを保存するセキュリティーキーの機能

8.7.8.2.2. Windows Hello

Windows Hello ベースの認証情報を使用して Red Hat build of Keycloak に対して認証するには、**WebAuthn Passwordless Policy** の **Signature Algorithms** 設定に **RS256** 値を含めるように設定します。一部のブラウザでは、プライベートウィンドウ内のプラットフォームセキュリティーキー (Windows Hello など) へのアクセスが許可されていないことに注意してください。

8.7.8.2.3. サポートされているセキュリティーキー

次のセキュリティーキーは、Red Hat build of Keycloak を使用したログインレス認証のテストに成功しました。

- Windows Hello (Windows 10 21H1/21H2)
- Yubico Yubikey 5 NFC
- Feitian ePass FIDO-NFC

8.8. リカバリーコード (RECOVERYCODES)

認証フローに 2 要素認証システムとして回復認証コードフォームを追加することにより、2 要素認証の回復コードを設定できます。このオーセンティケーターの設定例については、[WebAuthn](#) を参照してください。



注記

RecoveryCodesI は **テクノロジープレビュー** であるため、完全にサポートされていません。デフォルトでは無効になっています。

有効にするには、**--features=preview** または **--features=recovery-codes** を使用してサーバーを起動します。

8.9. 条件付きフローの条件

実行要件 で述べたように、条件 実行は **条件付き** サブフローにのみ含めることができます。すべての条件実行が true と評価されると、**Conditional** サブフローは **Required** として機能します。**Conditional** サブフローの次の実行を処理できます。**Conditional** サブフローに含まれる一部の実行が false と評価されると、サブフロー全体が **Disabled** と見なされます。

8.9.1. 利用可能な条件

Condition - User Role

この実行では、ユーザーに **User role** フィールドで定義されているロールがあるかどうかを判別できます。ユーザーに必要なロールが割り当てられている場合、実行は `true` と判断され、その他の実行が評価されます。管理者は以下のフィールドを定義する必要があります。

Alias

認証フローに表示される実行の名前を記述します。

User role

このフローを実行するために必要なユーザーロール。アプリケーションロールを指定する場合、構文は `appname.approle(myapp.myrole など)` です。

Condition - User Configured

これは、フローの他の実行がユーザーに設定されているかどうかを確認します。実行要件のセクションには、OTP フォームの例が含まれています。

Condition - User Attribute

ユーザーが必須の属性を設定したかチェックします。このチェックでは、オプションでグループ属性も評価できます。出力が否定される可能性があります。この場合、ユーザーに属性を設定することはできません。[ユーザー属性](#) セクションでは、カスタム属性の追加方法を示します。以下のフィールドを指定できます。

Alias

認証フローに表示される実行の名前を記述します。

Attribute name

確認する属性の名前。

Expected attribute value

属性の想定される値。

Include group attributes

この条件を `On` にすると、参加したグループのいずれかに、設定された名前と値に一致する属性が1つあるかチェックします。このオプションはパフォーマンスに影響を与える可能性があります。

Negate output

出力を否定することができます。つまり、この属性は存在しません。

8.9.2. 条件付きフローでのアクセスの明示的な拒否/許可

条件付きフローのリソースへのアクセスを許可または拒否できます。2つのオーセンティケーターは **Deny Access** と **Allow Access** は、条件でリソースへのアクセスを制御します。

Allow Access

オーセンティケーターは常に認証に成功します。このオーセンティケーターは設定できません。

Deny Access

アクセスは常に拒否されます。ユーザーに表示されるエラーメッセージを定義できます。以下のフィールドを指定できます。

Alias

認証フローに表示される実行の名前を記述します。

エラーメッセージ

ユーザーに表示されるエラーメッセージ。エラーメッセージは、ローカリゼーションで使用するために、特定のメッセージまたはプロパティとして提供できます。(つまり `You do not have the role 'admin'.`、メッセージプロパティの `my-property-deny`) プロパティ `access-denied`

として定義されたデフォルトメッセージの場合は空白のままにします。

以下の例は、ロール **role1** を持たないすべてのユーザーのアクセスを拒否し、プロパティ **deny-role1** で定義されたエラーメッセージを表示する方法を示しています。この例には、**Condition - User Role** および **Deny Access** 実行が含まれます。

ブラウザーのフロー

☰	Conditions Form	Alternative	+ ▾	🗑️
☰	Username Form	Required		🗑️
☰	Access by Role	Conditional	+ ▾	🗑️
☰	Condition - user role	Required	⚙️	🗑️
☰	Deny access	Required	⚙️	🗑️
☰	Password Form	Required		🗑️

Condition - user role configuration

Condition - user role config



Alias * ?

Must not have role1

User role ?

master



role1



Negate output ?



On

Save


Cancel

Deny Access の設定は非常に簡単です。以下のように、任意のエイリアスと必要なメッセージを指定できます。

Deny access config



Alias * 

Error message 

最後に、ログインテーマのエラーメッセージのプロパティ **messages_en.properties**(英語の場合) を定義します。

```
deny-role1 = You do not have required role!
```

第9章 アイデンティティプロバイダーの統合

Identity Broker は、サービスプロバイダーをアイデンティティプロバイダーに接続する中間サービスです。アイデンティティプロバイダーは外部アイデンティティプロバイダーとの関係を作成し、プロバイダーのアイデンティティを使用してサービスプロバイダーが公開する内部サービスにアクセスします。

ユーザーの観点からすると、アイデンティティブローカーは、セキュリティドメインおよびレルムのアイデンティティを管理するユーザー中心の一元的な方法を提供します。アカウントをアイデンティティプロバイダーからの1つまたは複数のアイデンティティとリンクすることや、プロバイダーからの ID 情報に基づいてアカウントを作成することができます。

アイデンティティプロバイダーは特定のプロトコルに基づき、これを使用して認証を行い認証および認可情報をユーザーに送付します。以下をアイデンティティプロバイダーとすることができます。

- Facebook、Google、Twitter などのソーシャルプロバイダー。
- そのユーザーがお客様のサービスにアクセスする必要があるビジネスパートナー。
- 統合するクラウドベースのアイデンティティサービス。

通常、Red Hat build of Keycloak では、以下のプロトコルのアイデンティティプロバイダーがベースとなります。

- **SAML v2.0**
- **OpenID Connect v1.0**
- **OAuth v2.0**

9.1. ブローカーの概要

Red Hat build of Keycloak をアイデンティティブローカーとして使用する場合、Red Hat build of Keycloak は特定のレルムで認証する際に認証情報の提供をユーザーに強制しません。Red Hat build of Keycloak には、認証できるアイデンティティプロバイダーのリストが表示されます。

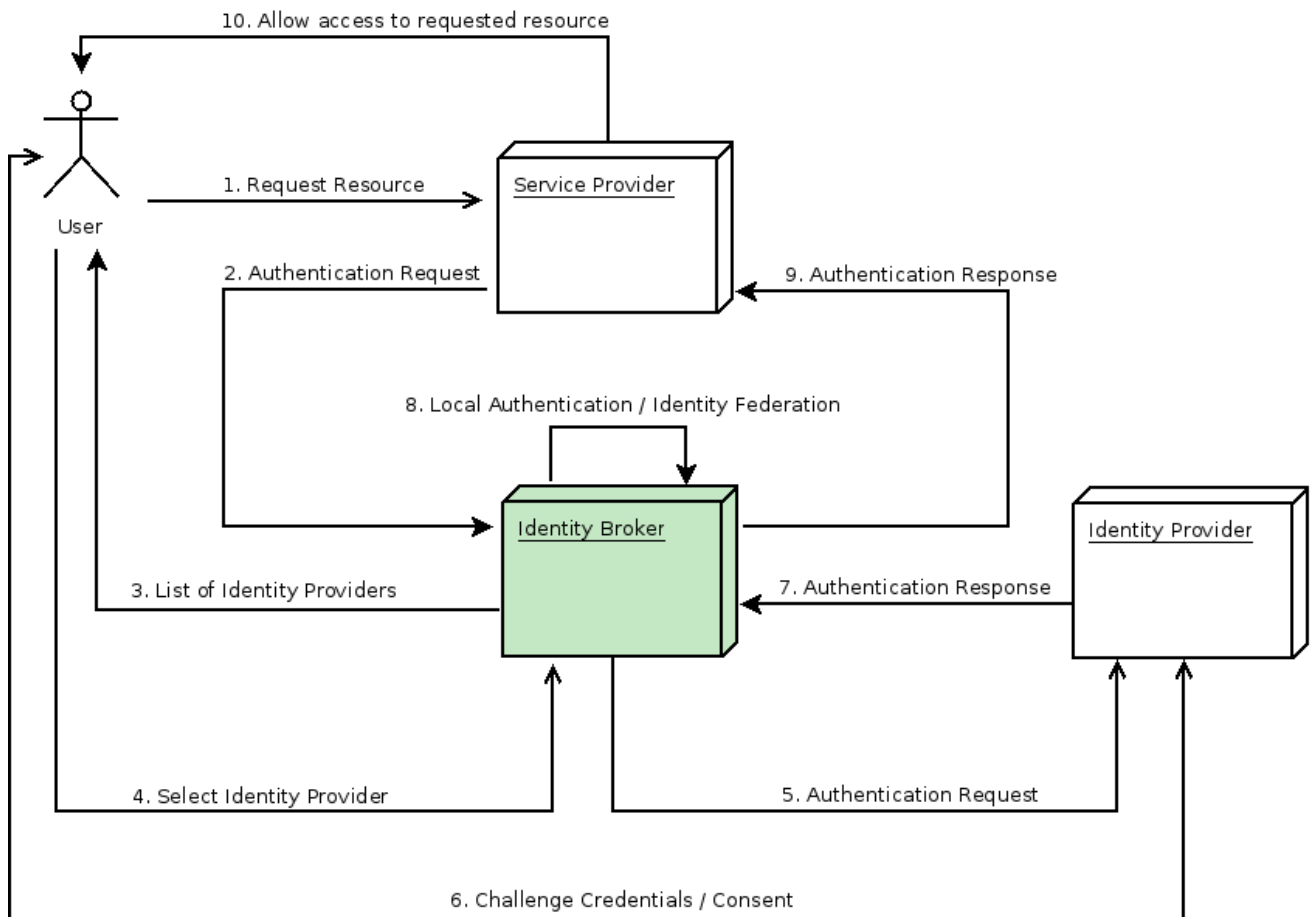
デフォルトのアイデンティティプロバイダーを設定すると、Red Hat build of Keycloak はユーザーをデフォルトのプロバイダーにリダイレクトします。



注記

プロトコルによって異なる認証フローが必要になる場合があります。Red Hat build of Keycloak でサポートされるすべてのアイデンティティプロバイダーは、次のフローを使用します。

アイデンティティブローカーフロー



1. 認証されていないユーザーは、クライアントアプリケーションの保護されているリソースを要求します。
2. クライアントアプリケーションは、認証のためにユーザーを Red Hat build of Keycloak にリダイレクトします。
3. Red Hat build of Keycloak では、レルム内に設定されたアイデンティティプロバイダーのリストを含むログインページが表示されます。
4. ユーザーは、ボタンまたはリンクをクリックしてアイデンティティプロバイダーの1つを選択します。
5. Red Hat build of Keycloak は、ターゲットのアイデンティティプロバイダーに認証要求を発行して認証を要求し、ユーザーをアイデンティティプロバイダーのログインページにリダイレクトします。管理者は、すでに管理コンソールのアイデンティティプロバイダーの接続プロパティおよびその他の設定オプションを設定しています。
6. ユーザーは、アイデンティティプロバイダーと認証を行うための認証情報または同意を提供します。
7. アイデンティティプロバイダーによる認証が成功すると、ユーザーは認証応答とともに Red Hat build of Keycloak にリダイレクトされます。通常、応答には、Red Hat build of Keycloak がアイデンティティプロバイダーの認証を信頼し、ユーザー情報を取得するために使用するセキュリティトークンが含まれます。
8. Red Hat build of Keycloak は、アイデンティティプロバイダーからの応答が有効かチェックします。応答が有効で、かつそのユーザーがまだ存在していない場合、Red Hat build of Keycloak はユーザーをインポートして作成します。トークンに必要な情報が含まれていない場合、Red Hat build of Keycloak はアイデンティティプロバイダーに追加のユーザー情報を求める場合があります。この動作は **ID フェデレーション** です。ユーザーがすでに存在する場合、Red Hat

build of Keycloak は、アイデンティティプロバイダーから返されたアイデンティティを既存のアカウントにリンクするようユーザーに要求します。この動作は、**アカウントのリンク**です。Red Hat build of Keycloak を使用すると、**Account linking** を設定し、**First Login Flow** で指定できます。このステップで、Red Hat build of Keycloak はユーザーを認証し、サービスプロバイダー内の要求されたリソースにアクセスするためのトークンを発行します。

9. ユーザーが認証されると、Red Hat build of Keycloak は、ローカル認証時に発行されたトークンを送信することで、ユーザーをサービスプロバイダーにリダイレクトします。
10. サービスプロバイダーは Red Hat build of Keycloak からトークンを受け取り、保護されたリソースへのアクセスを許可します。


このフローのバリエーションが可能です。たとえば、クライアントアプリケーションは、アイデンティティプロバイダーのリストを表示するのではなく特定のプロバイダーを要求するか、アイデンティティのフェデレーションを行う前にユーザーに追加情報の提供を強制するように Red Hat build of Keycloak を設定できます。

認証プロセスの最後に、Red Hat build of Keycloak がそのトークンをクライアントアプリケーションに発行します。クライアントアプリケーションは外部のアイデンティティプロバイダーから分離されるため、クライアントアプリケーションのプロトコルやユーザーのアイデンティティの検証方法を確認できません。プロバイダーが把握する必要があるのはRed Hat build of Keycloak だけです。

9.2. デフォルトのアイデンティティプロバイダー

Red Hat build of Keycloak は、ログインフォームを表示する代わりに、アイデンティティプロバイダーにリダイレクトできます。このリダイレクトを有効にするには、以下を実行します。

手順

1. メニューで **Authentication** をクリックします。
2. **Browser** フローをクリックします。
3. **Identity Provider Redirector** 行の歯車アイコン  をクリックします。
4. **Default Identity Provider** を、ユーザーをリダイレクトするアイデンティティプロバイダーに設定します。

Red Hat build of Keycloak で設定されたデフォルトのアイデンティティプロバイダーが見つからない場合は、ログインフォームが表示されます。

このオーセンティケーターは、**kc_idp_hint** クエリーパラメーターを処理します。詳細については、[クライアントが推奨する ID プロバイダー](#) セクションを参照してください。

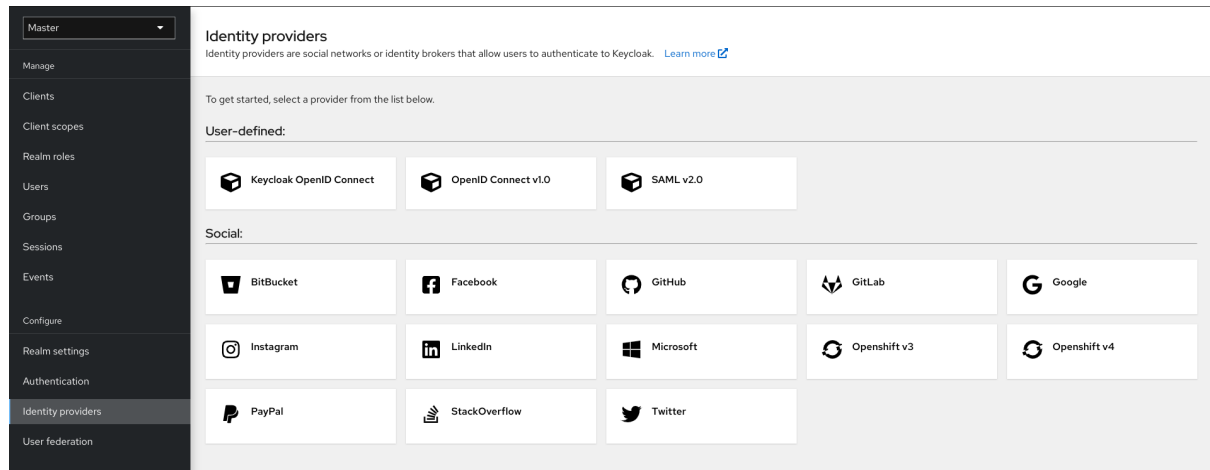
9.3. 一般的な設定

アイデンティティブローカー設定の基盤は、アイデンティティプロバイダー (IDP) です。Red Hat build of Keycloak は、各レルムにアイデンティティプロバイダーを作成し、デフォルトですべてのアプリケーションに対して有効にします。レルムからのユーザーは、アプリケーションへのサインイン時に登録されたいずれかのアイデンティティプロバイダーを使用できます。

手順

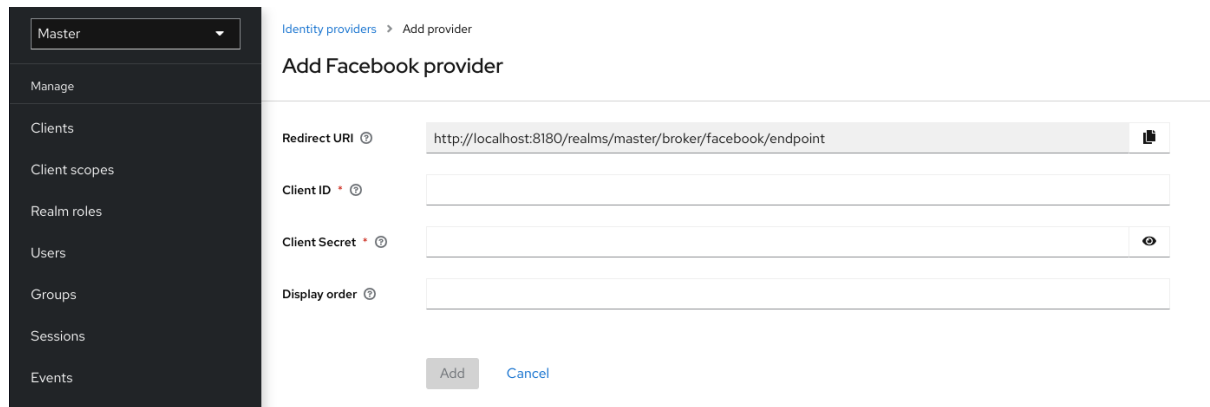
1. メニューで **Identity Providers** をクリックします。

ID プロバイダー



2. ID プロバイダーを選択します。Red Hat build of Keycloak には、選択したアイデンティティプロバイダーの設定ページが表示されます。

Facebook アイデンティティプロバイダーの追加



アイデンティティプロバイダーを設定すると、そのアイデンティティプロバイダーはオプションとして Red Hat build of Keycloak ログインページに表示されます。各アイデンティティプロバイダーについて、ログイン画面にカスタムアイコンを配置することができます。詳細は、[カスタムアイコン](#) を参照してください。

IDP ログインページ

Sign in to your account

Username or email

Password

Remember me

Sign In

Or sign in with



Facebook

New user? [Register](#)

ソーシャル

ソーシャルプロバイダーを使用すると、レルムでソーシャル認証を有効にできます。Red Hat build of Keycloak を使用すると、ユーザーはソーシャルネットワークアカウントを使用してアプリケーションにログインできます。サポートされるプロバイダーには、Twitter、Facebook、Google、LinkedIn、Instagram、Microsoft、PayPal、OpenShift v3、GitHub、GitLab、Bitbucket、および Stack Overflow が含まれます。

プロトコルベース

プロトコルベースのプロバイダーは、ユーザーの認証および認可を特定のプロトコルに依存します。これらのプロバイダーを使用して、特定のプロトコルに準拠する任意のアイデンティティプロバイダーに接続できます。Red Hat build of Keycloak は、SAML v2.0 および OpenID Connect v1.0 プロトコルをサポートします。これらのオープン標準に基づいて、任意のアイデンティティプロバイダーを設定し、ブローカーを設定できます。

それぞれの種類のアイデンティティプロバイダーにはその設定オプションがありますが、設定の一部はすべてに共通です。以下の設定オプションが利用できます。

表9.1 共通の設定

設定	説明
Alias	エイリアスは、アイデンティティプロバイダーの一意の識別子で、内部アイデンティティプロバイダーを参照します。Red Hat build of Keycloak は、エイリアスを使用して OpenID Connect プロトコルのリダイレクト URI を構築します。これらのプロトコルには、アイデンティティプロバイダーと通信するためのリダイレクト URI またはコールバック URL が必要です。すべてのアイデンティティプロバイダーにはエイリアスが必要です。エイリアスの例には facebook 、 google 、および idp.acme.com があります。
Enabled	プロバイダーのオン/オフを切り替えます。
Hide on Login Page	ON の場合、このプロバイダーは Red Hat build of Keycloak のログインページにログインオプションとして表示されません。クライアントがこのプロバイダーを要求するには、URL の 'kc_idp_hint' パラメーターを使用してログインを要求します。
Account Linking Only	ON の場合、Red Hat build of Keycloak は既存のアカウントをこのプロバイダーにリンクします。このプロバイダーはユーザーをログインできず、Red Hat build of Keycloak のログインページのオプションとして表示されません。
Store Tokens	ON の場合、Red Hat build of Keycloak はアイデンティティプロバイダーからのトークンを保存します。
Stored Tokens Readable	ON の場合、ユーザーは保存されたアイデンティティプロバイダートークンを取得できます。このアクションは、 broker クライアントレベルのロールの read token にも適用されます。
Trust Email	ON の場合、Red Hat build of Keycloak はアイデンティティプロバイダーからのメールアドレスを信頼します。レルムがメール検証を要求する場合は、このアイデンティティプロバイダーからログインするユーザーは、メールの検証プロセスを実行する必要はありません。
GUI Order	利用可能なアイデンティティプロバイダーの、ログインページでの並べ替え順序。
Verify essential claim	ON の場合、アイデンティティプロバイダーによって発行された ID トークンには特定の要求が必要です。ない場合、ユーザーはこのブローカーを介して認証できません。

設定	説明
Essential claim	Verify essential claim が ON の場合、フィルターする JWT トークン要求の名前 (照合時は大文字と小文字が区別されます)。
Essential claim value	Verify essential Claim が ON の場合、一致する JWT トークン要求の値 (正規表現形式がサポートされます)。
First Login Flow	ユーザーがこのアイデンティティプロバイダーを使用して初めて Red Hat build of Keycloak にログインする際に、Red Hat build of Keycloak がトリガーする認証フロー。
Post Login Flow	ユーザーが外部アイデンティティプロバイダーを使用したログインを終了した際に、Red Hat build of Keycloak がトリガーする認証フロー。
Sync Mode	マッパーを通じて、アイデンティティプロバイダーからのユーザー情報を更新するストラテジー。 legacy を選択すると、Red Hat build of Keycloak は現在の動作を使用しました。 Import の場合はユーザーデータを更新せず、 force の場合は、可能であればユーザーデータを更新します。詳細については、 アイデンティティプロバイダーマッパー を参照してください。

9.4. ソーシャルアイデンティティプロバイダー

ソーシャルアイデンティティプロバイダーは、認証を信頼できるソーシャルメディアアカウントに委譲できます。Red Hat build of Keycloak には、Google、Facebook、Twitter、GitHub、LinkedIn、Microsoft、Stack Overflow などのソーシャルネットワークのサポートが含まれます。

9.4.1. Bitbucket

Bitbucket でログインするには、以下の手順を実行します。



手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Bitbucket** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > [Add provider](#)

Add Bitbucket provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/bitbucket/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブで、[Bitbucket Cloud での OAuth](#) のプロセスを実行します。 **Add Consumer** をクリックする際に、
 - a. **Redirect URI** の値を **Callback URL** フィールドに貼り付けます。
 - b. **Account** セクションで **Email** および **Read** を選択し、アプリケーションが電子メールを読み取れるようにします。
5. コンシューマーの作成時に Bitbucket が表示する **Key** および **Secret** の値を書き留めておきます。
6. Red Hat build of Keycloak で、**Key** の値を **Client ID** フィールドにペーストします。
7. Red Hat build of Keycloak で、**Client Secret** の値を **Client Secret** フィールドにペーストします。
8. **Add** をクリックします。

9.4.2. Facebook

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Facebook** を選択します。

アイデンティティプロバイダーの追加

Identity providers > Add provider

Add Facebook provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/facebook/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	
Additional user's profile fields ⓘ	<input type="text"/>	

3. Redirect URI の値をクリップボードにコピーします。
4. 別のブラウザタブで、[Meta for Developers](#) を開きます。
 - a. My Apps をクリックします。
 - b. Create App を選択します。

ユースケースの追加

Create an app ✕ Cancel

Add use case
 App details

What do you want your app to do?
These are the most common use cases developers have used on Meta for Developers. Each use case unlocks secondary use cases with more functionality. Configure use cases once your app is created.

Allow people to log in with their Facebook account
Our most common use case. A secure, fast, and convenient way for users to log into your app, and for your app to ask users for permission to access their data. ▼

Get gaming login and request data from players
Give players a way to log into your game across multiple platforms and ask users for permission to access player data. Players can use custom player names and avatars. To create an Instant Games app, select Other below and select Instant Games. ▼

Looking for something else?
If you need something that isn't shown above, you can see more options by selecting Other.

Other
Explore other products and data permissions such as ads management, Instant Games and more. You'll be asked to select an app type and then you can add the permissions and products you need.

- c. Other を選択します。

アプリケーションタイプを選択します。

The screenshot shows the 'Create an app' dialog box with the 'Type' tab selected. The 'Select an app type' section lists several options, with 'Consumer' selected. The options are:

- Consumer** (Selected): Connect consumer products and permissions, like Facebook Login and Instagram Basic Display to your app.
- Business**: Create or manage business assets like Pages, Events, Groups, Ads, Messenger, WhatsApp, and Instagram Graph API using the available business permissions, features and products.
- Instant Games**: Create an HTML5 game hosted on Facebook.
- Gaming**: Connect an off-platform game to Facebook Login.
- Workplace**: Create enterprise tools for Workplace from Meta.
- Academic research**: Connect to Facebook data and tooling to perform research on Facebook.

A 'Next' button is visible at the bottom right of the dialog.

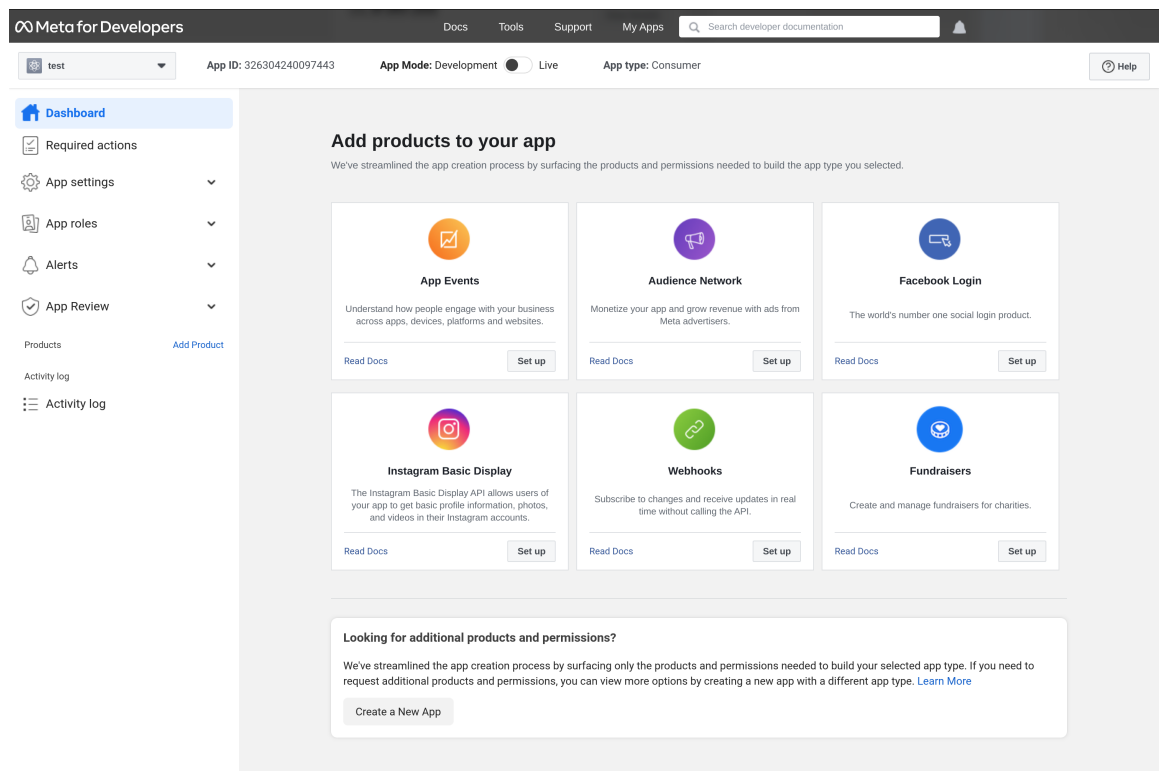
- d. **Consumer** を選択します。

アプリケーションの作成

The screenshot shows the 'Create an app' dialog box with the 'Details' tab selected. The 'Add an app name' section is active, with a text input field containing '0/30'. Below it, the 'App contact email' section has a text input field containing 'username@domain.com'. The 'Business Account - Optional' section has a dropdown menu with 'No Business Manager account selected'. At the bottom, there are 'Previous' and 'Create app' buttons.

- e. すべての必須フィールドに入力します。
- f. **Create App** をクリックします。次に、ダッシュボードに移動します。

製品の追加



- g. **Facebook Login** ボックスで **Set Up** をクリックします。
 - h. **Web** を選択します。
 - i. **Redirect URI** の値を **Site URL** フィールドに入力し、**Save** をクリックします。
 - j. ナビゲーションパネルで **App settings - Basic** を選択します。
 - k. **App Secret** フィールドで **Show** をクリックします。
 - l. **App ID** と **App Secret** をメモします。
5. Facebook アプリケーションの **App ID** および **App Secret** の値を、Red Hat build of Keycloak の **Client ID** フィールドおよび **Client Secret** フィールドに入力します。
 6. **Add** をクリックします。
 7. 必要なスコープを **Default Scopes** フィールドに入力します。デフォルトでは、Red Hat build of Keycloak は **email** スコープを使用します。Facebook スコープの詳細については、[グラフ API](#) を参照してください。

デフォルトでは、Red Hat build of Keycloak はプロフィール要求を **graph.facebook.com/me?fields=id,name,email,first_name,last_name** に送信します。応答には、id、name、email、first_name、および last_name フィールドのみが含まれます。Facebook プロファイルから追加のフィールドを取得するには、対応するスコープを追加し、**Additional user's profile fields** 設定オプションフィールドにフィールド名を追加します。

9.4.3. GitHub

GitHub でログインするには、以下の手順を実行します。

手順



1. メニューで **Identity Providers** をクリックします。

2. **Add provider** リストから **Github** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > [Add provider](#)

Add Github provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/github/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	
Base URL ⓘ	<input type="text"/>	
API URL ⓘ	<input type="text"/>	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブで [OAUTH アプリケーションを作成](#) します。
 - a. アプリケーションの作成時に、**Redirect URI** の値を **Authorization callback URL** フィールドに入力します。
 - b. OAUTH アプリケーションの管理ページの **Client ID** と **Client シークレット** をメモします。
5. Red Hat build of Keycloak で、**Client ID** の値を **Client ID** フィールドにペーストします。
6. Red Hat build of Keycloak で、**Client Secret** の値を **Client Secret** フィールドにペーストします。
7. **Add** をクリックします。

9.4.4. GitLab



手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **GitLab** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > Add provider

Add Gitlab provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/gitlab/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブで、[新しい GitLab アプリケーションを追加](#)します。
 - a. クリップボードの **Redirect URI** を **Redirect URI** として使用します。
 - b. アプリケーションを保存するときに、**Application ID** と **Secret** をメモします。
5. Red Hat build of Keycloak で、**Application ID** の値を **Client ID** フィールドにペーストします。
6. Red Hat build of Keycloak で、**Client Secret** の値を **Client Secret** フィールドにペーストします。
7. **Add** をクリックします。

9.4.5. Google

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Google** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > [Add provider](#)

Add Google provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/google/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	
Hosted Domain ⓘ	<input type="text"/>	
Use userlp param ⓘ	<input type="checkbox"/> Off	
Request refresh token ⓘ	<input type="checkbox"/> Off	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブで、[GoogleCloudPlatform コンソール](#) 開きます。
5. Google アプリの Google ダッシュボードの左側にあるナビゲーションメニューで、**APIs & Services** の上にマウスを置き、**OAuth consent screen** オプションをクリックします。同意画面を作成し、合意画面のユーザータイプが **External** であることを確認します。
6. Google ダッシュボードで以下を行います。
 - a. **Credentials** メニューをクリックします。
 - b. **CREATE CREDENTIALS - OAuth Client ID** をクリックします。
 - c. **Application type** リストから **Web application** を選択します。
 - d. クリップボード内の **Redirect URI** を **Authorized redirect URIs** として使用します。
 - e. **Create** をクリックします。
 - f. **Your Client ID** とクライアント **Your Client Secret** を書き留めます。
7. Red Hat build of Keycloak で、**Your Client ID** の値を **Client ID** フィールドにペーストします。
8. Red Hat build of Keycloak で、**Your Client Secret** の値を **Client Secret** フィールドにペーストします。
9. **Add** をクリックします。
10. 必要なスコープを **Default Scopes** フィールドに入力します。Red Hat build of Keycloak は、デフォルトで scopes: **openid profile email** のスコープを使用します。Google スコープのリストについては、[OAuth Playground](#) を参照してください。
11. アクセスを GSuite 組織のメンバーのみに制限するには、**Hosted Domain** フィールドに G Suite ドメインを入力します。

12. **Save** をクリックします。

9.4.6. Instagram



手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Instagram** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > Add provider

Add Instagram provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/instagram/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブで、[Meta for Developers](#) を開きます。
 - a. **My Apps** をクリックします。
 - b. **Create App** を選択します。

ユースケースの追加

The screenshot shows the 'Create an app' wizard in the Keycloak admin console. The current step is 'Add use case'. On the left, there are two tabs: 'Add use case' (selected) and 'App details'. The main area is titled 'What do you want your app to do?' and contains three radio button options:

- Allow people to log in with their Facebook account**: Our most common use case. A secure, fast, and convenient way for users to log into your app, and for your app to ask users for permission to access their data.
- Get gaming login and request data from players**: Give players a way to log into your game across multiple platforms and ask users for permission to access player data. Players can use custom player names and avatars. To create an Instant Games app, select Other below and select Instant Games.
- Other**: Explore other products and data permissions such as ads management, Instant Games and more. You'll be asked to select an app type and then you can add the permissions and products you need.

Below the options, there is a section 'Looking for something else?' with the text: 'If you need something that isn't shown above, you can see more options by selecting Other.' A 'Next' button is located at the bottom right.

c. **Other** を選択します。

アプリケーションタイプを選択します。

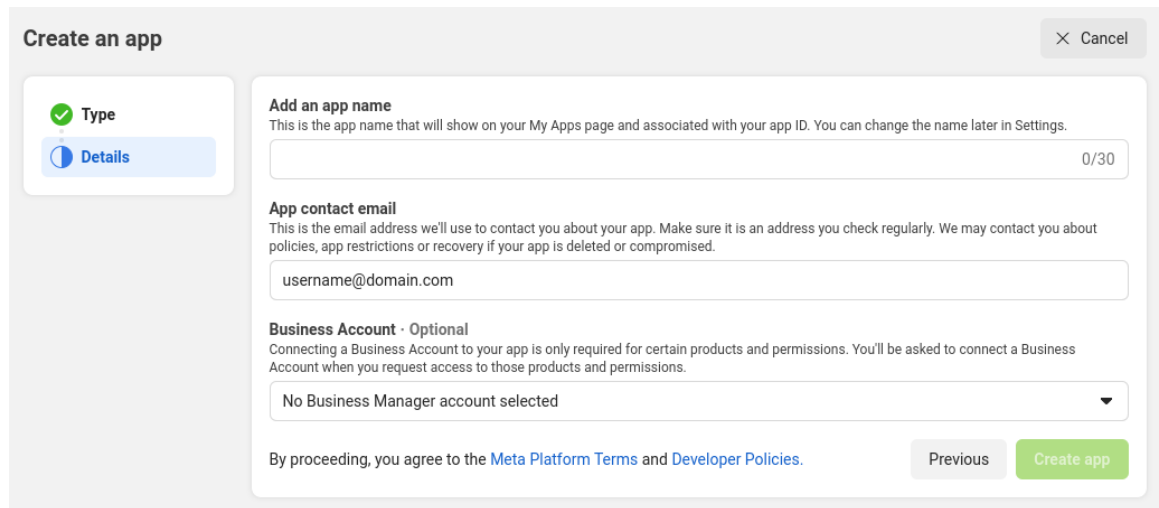
The screenshot shows the 'Create an app' wizard in the Keycloak admin console. The current step is 'Select an app type'. On the left, there are two tabs: 'Type' (selected) and 'Details'. The main area is titled 'Select an app type' and contains the text: 'The app type can't be changed after your app is created. [Learn more](#)'. Below this, there are six radio button options:

- Consumer**: Connect consumer products and permissions, like Facebook Login and Instagram Basic Display to your app. (Selected)
- Business**: Create or manage business assets like Pages, Events, Groups, Ads, Messenger, WhatsApp, and Instagram Graph API using the available business permissions, features and products.
- Instant Games**: Create an HTML5 game hosted on Facebook.
- Gaming**: Connect an off-platform game to Facebook Login.
- Workplace**: Create enterprise tools for Workplace from Meta.
- Academic research**: Connect to Facebook data and tooling to perform research on Facebook.

A 'Next' button is located at the bottom right.

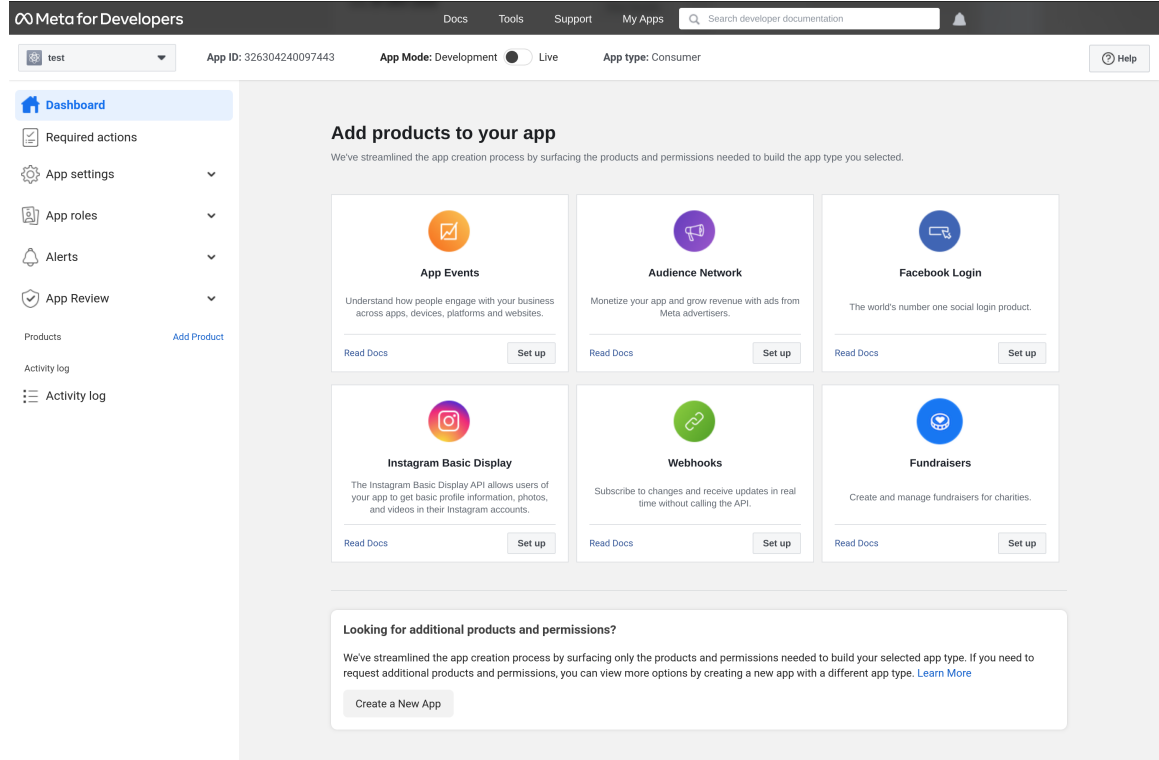
d. **Consumer** を選択します。

アプリケーションの作成



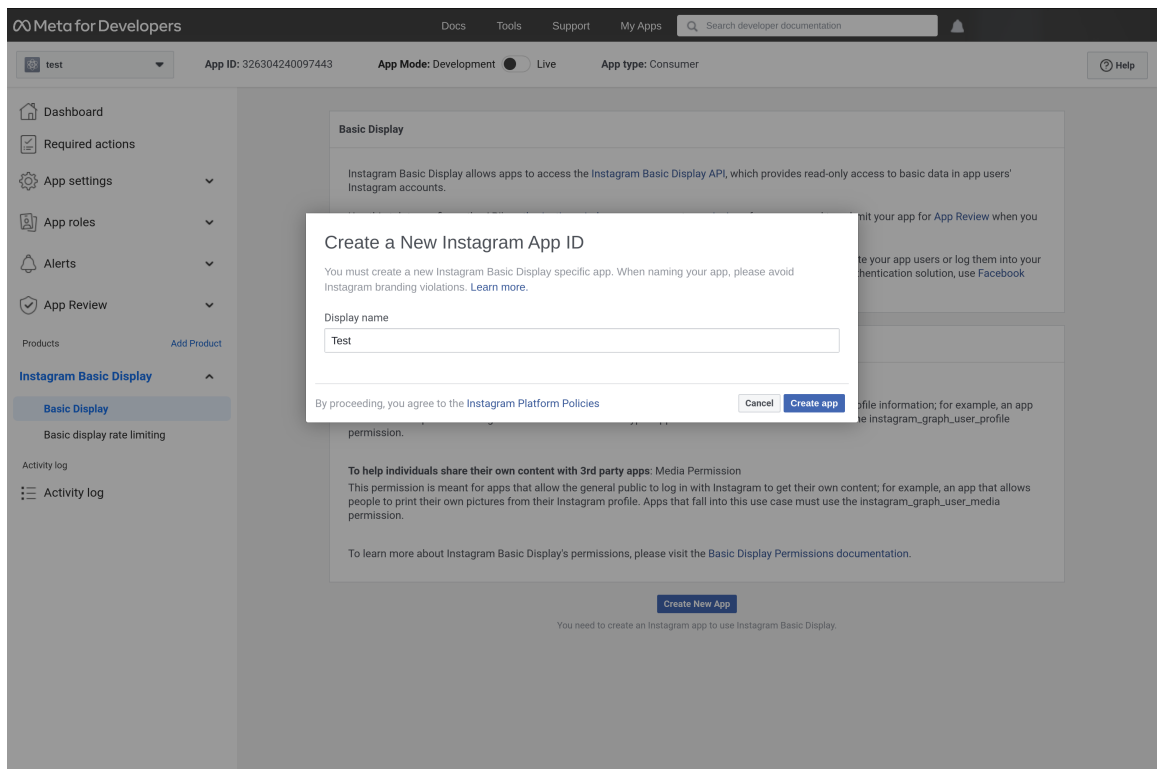
- e. すべての必須フィールドに入力します。
- f. **Create App** をクリックします。次に、ダッシュボードに移動します。
- g. ナビゲーションパネルで **App settings - Basic** を選択します。
- h. ページの下部にある **+ Add Platform** を選択します。
- i. **[Website]** をクリックします。
- j. サイトの URL を入力します。

製品の追加



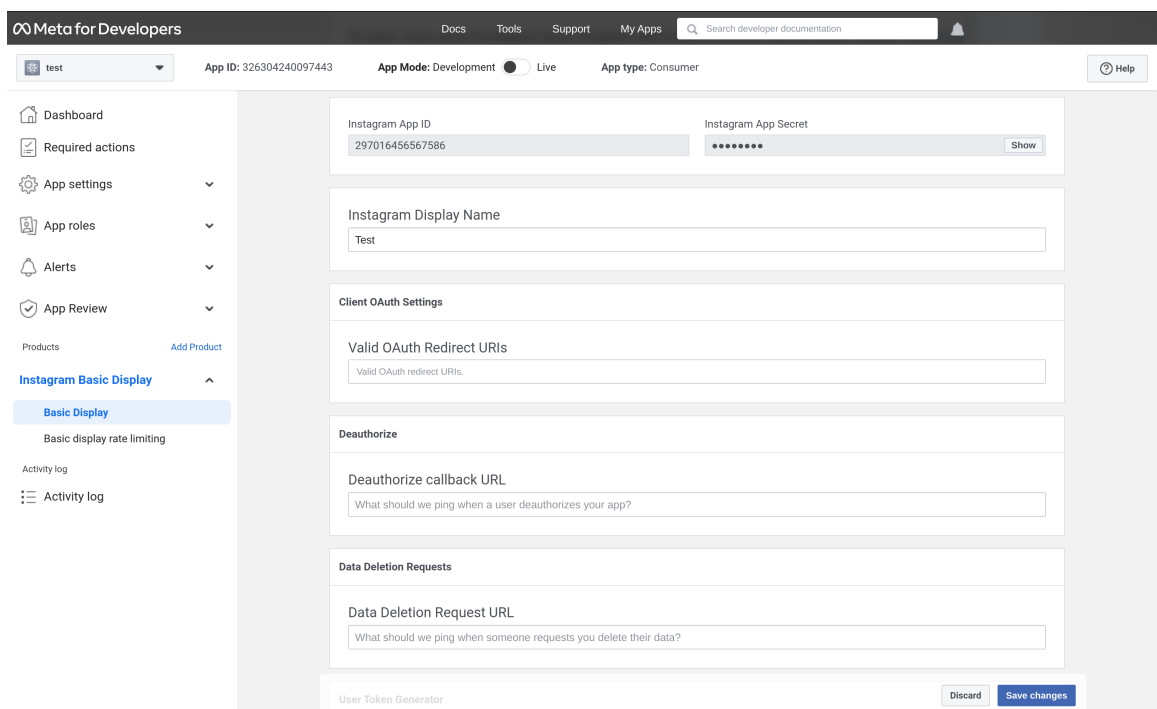
- k. メニューから **Dashboard** を選択します。
- l. **Instagram Basic Display** ボックスで **Set Up** をクリックします。
- m. **Create New App** をクリックします。

新規 Instagram アプリケーション ID の作成



- n. Display name フィールドに値を入力します。

アプリをセットアップします。



- o. Red Hat build of Keycloak の Redirect URL を Valid OAuth Redirect URIs フィールドにペーストします。
- p. Red Hat build of Keycloak の Redirect URL を Valid Deauthorize Callback URL フィールドにペーストします。
- q. Red Hat build of Keycloak の Redirect URL を Valid Data Deletion Request URL フィールドにペーストします。

- r. **Instagram App Secret** フィールドで **Show** をクリックします。
 - s. **Instagram App ID** と **Instagram App Secret** をメモします。
 - t. **App Review - Requests** をクリックします。
 - u. 画面の指示に従います。
5. Red Hat build of Keycloak で、**Instagram App ID** の値を **Client ID** フィールドにペーストします。
 6. Red Hat build of Keycloak で、**Instagram App Secret** の値を **Client Secret** フィールドにペーストします。
 7. **Add** をクリックします。

9.4.7. LinkedIn


手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **LinkedIn** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > [Add provider](#)

Add LinkedIn-openid-connect provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/linkedin-openid-connect/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="text"/>	
Display order ⓘ	<input type="text"/>	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブを使用して、LinkedIn 開発者ポータルで [アプリを作成](#) します。
 - a. アプリケーションの作成後に、**Auth** タブをクリックします。
 - b. **Redirect URI** の値を **Authorized redirect URLs for your app** フィールドに入力します。
 - c. **Your Client ID** とクライアント **Your Client Secret** を書き留めます。
 - d. **Products** タブをクリックし、**Sign In with LinkedIn using OpenID Connect** 製品の **Request access** をクリックします。
5. Red Hat build of Keycloak で、**Client ID** の値を **Client ID** フィールドにペーストします。

- Red Hat build of Keycloak で、**Client Secret** の値を **Client Secret** フィールドにペーストします。
- Add** をクリックします。

9.4.8. Microsoft



手順

- メニューで **Identity Providers** をクリックします。
- Add provider** リストから **Microsoft** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > [Add provider](#)

Add Microsoft provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/microsoft/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	

- Redirect URI** の値をクリップボードにコピーします。
- 別のブラウザタブの **App registrations** にある [Microsoft Azure](#) でアプリケーションを登録します。
 - Redirect URI セクションで、プラットフォームとして **Web** を選択し、**Redirect URI** の値をフィールドに貼り付けます。
 - App registrations** でアプリケーションを見つけ、**Certificates & secrets** セクションで新しいクライアントシークレットを追加します。
 - 作成されたシークレットの **値** をメモします。
 - Overview** セクションの **Application (client) ID** をメモします。
- Red Hat build of Keycloak で、**Application (client) ID** の値を **Client ID** フィールドにペーストします。
- Red Hat build of Keycloak で、**Client Secret** の値を **Client Secret** フィールドにペーストします。
- Add** をクリックします。

9.4.9. OpenShift 3

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Openshift v3** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > Add provider

Add Openshift-v3 provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/openshift-v3/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	
Base URL ⓘ	<input type="text"/>	

3. **Redirect URI** の値をクリップボードにコピーします。
4. **oc** コマンドラインツールを使用して、クライアントを登録します。

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: v1
metadata:
  name: kc-client ①
secret: "..." ②
redirectURIs:
  - "http://www.example.com/" ③
grantMethod: prompt ④
')
```

- ① OAuth クライアントの **名前**。<openshift_master>/oauth/authorize および <openshift_master>/oauth/token への要求の実行時に **client_id** 要求パラメーターとして渡されます。
- ② Red Hat build of Keycloak が **client_secret** リクエストパラメーターに使用する **secret**。
- ③ <openshift_master>/oauth/authorize および <openshift_master>/oauth/token への要求で指定される **redirect_uri** パラメーターは、**redirectURIs** のいずれかの URI と同じ (または接頭辞が付けられた) 必要があります。これは、Identity Provider 画面の **Redirect URI** フィールドから取得できます。
- ④ このクライアントがトークンを要求してもユーザーによってアクセスが許可されない場合のアクションを決定するために Red Hat build of Keycloak が使用する **grantMethod**。

1. Red Hat build of Keycloak で、**Client ID** の値を **Client ID** フィールドにペーストします。

2. Red Hat build of Keycloak で、**Client Secret** の値を **Client Secret** フィールドにペーストします。
3. **Add** をクリックします。

9.4.10. OpenShift 4

前提条件

1. Keycloak トラストストアに保存されている OpenShift 4 インスタンスの証明書。
2. トラストストアを使用するために設定された Keycloak サーバー。詳細は、[トラストストアの設定](#)の章を参照してください。

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Openshift v4** を選択します。
3. **Client ID** と **Client Secret** を入力し、**Base URL** フィールドに OpenShift 4 インスタンスの API URL を入力します。**Redirect URI** をクリップボードにコピーすることもできます。

アイデンティティプロバイダーの追加

[Identity providers](#) > Add provider

Add Openshift-v4 provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/openshift-v4/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="text"/>	
Display order ⓘ	<input type="text"/>	
Base URL ⓘ	<input type="text"/>	

4. OpenShift 4 コンソール (Home → API Explorer → OAuth Client → Instances) 経由で、または **oc** コマンドラインツールを使用して、クライアントを登録します。

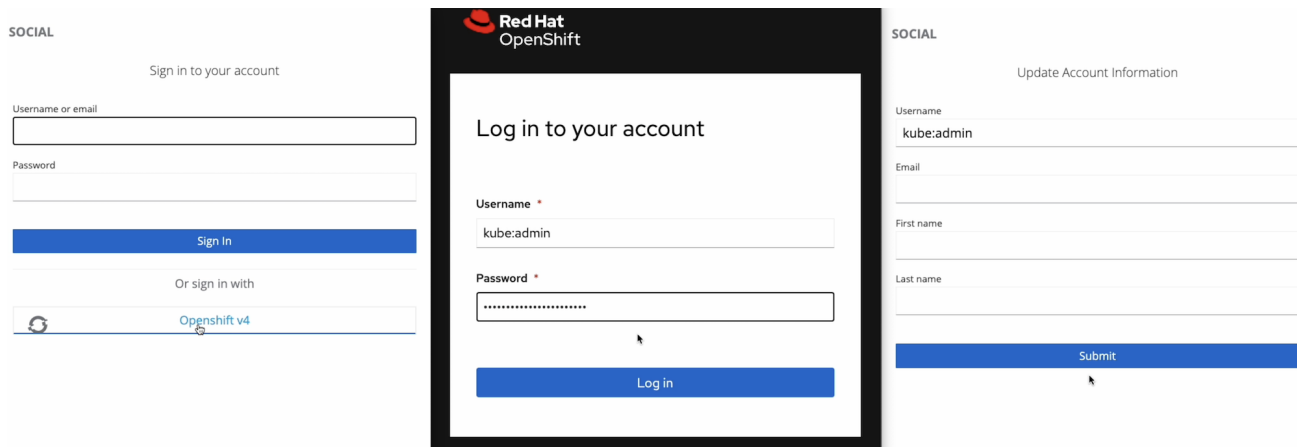
```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
  name: kc-client 1
  secret: "..." 2
  redirectURIs:
```

```
- "<here you can paste the Redirect URI that you copied in the previous step>" 3
grantMethod: prompt 4
')
```

- 1 OAuth クライアントの **名前**。<openshift_master>/oauth/authorize および <openshift_master>/oauth/token への要求の実行時に **client_id** 要求パラメーターとして渡されます。**name** パラメーターは、**OAuthClient** オブジェクトと Red Hat build of Keycloak 設定と同じである必要があります。
- 2 Red Hat build of Keycloak が **client_secret** 要求パラメーターとして使用する **secret**。
- 3 <openshift_master>/oauth/authorize および <openshift_master>/oauth/token への要求で指定される **redirect_uri** パラメーターは、**redirectURIs** のいずれかの URI と同じ (または接頭辞が付けられた) 必要があります。これを正しく設定する最も簡単な方法は、Red Hat build of Keycloak の OpenShift 4 アイデンティティプロバイダー設定ページ (**Redirect URI** フィールド) からコピーすることです。
- 4 このクライアントがトークンを要求してもユーザーによってアクセスが許可されない場合のアクションを決定するために Red Hat build of Keycloak が使用する **grantMethod**。

最終的に、Red Hat build of Keycloak インスタンスのログインページに、OpenShift 4 ID プロバイダーが表示されるはずですが、これをクリックすると、OpenShift 4 のログインページにリダイレクトされます。

結果



詳細は、[公式の OpenShift ドキュメント](#) を参照してください。

9.4.11. PayPal

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **PayPal** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > [Add provider](#)

Add Paypal provider

Redirect URI ⓘ	<input type="text" value="http://127.0.0.1:8080/realms/master/broker/paypal/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="password"/>	
Display order ⓘ	<input type="text"/>	
Target Sandbox ⓘ	<input type="checkbox"/> Off	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブで、[PayPal 開発者アプリケーションエリア](#) を開きます。
 - a. **Create App** をクリックして PayPal アプリケーションを作成します。
 - b. **Client ID** と **Client Secret** を書き留めます。 **Show** のリンクをクリックしてシークレットを表示します。
 - c. **Log in with PayPal** がオンになっていることを確認します。
 - d. Log in with PayPal で、 **Advanced Settings** をクリックします。
 - e. **Return URL** フィールドの値を、Red Hat build of Keycloak の **Redirect URI** の値に設定します。URL には **localhost** を含めることはできません。Red Hat build of Keycloak をローカルで使用する場合は、**Return URL** の **localhost** を **127.0.0.1** に置き換え、ブラウザで **localhost** の代わりに **127.0.0.1** を使用して Red Hat build of Keycloak にアクセスします。
 - f. **Full Name** フィールドと **Email** フィールドがチェックされていることを確認します。
 - g. **Save** をクリックしてから **Save Changes** をクリックします。
5. Red Hat build of Keycloak で、**Client ID** の値を **Client ID** フィールドにペーストします。
6. Red Hat build of Keycloak で、**Secret key 1** の値を **Client Secret** フィールドに貼り付けます。
7. **Add** をクリックします。

9.4.12. Stack overflow

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Stack Overflow** を選択します。

アイデンティティプロバイダーの追加

Identity providers > Add provider

Add Stackoverflow provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/stackoverflow/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="text"/>	
Display order ⓘ	<input type="text"/>	
Key ⓘ	<input type="text"/>	

- 別のブラウザタブで、[Stack Apps](#) でのアプリケーションの登録 にログインします。

アプリケーションの登録

StackExchange 1 help Search Q&A

stackapps Questions Tags Users Badges Unanswered Ask Question

Register Your V2.0 Application

Application Name

Description

OAuth Domain

Application Website

Application Icon (optional)
 Enable Client Side OAuth Flow

Why Register?

Because it's the neighborly thing to do. We like to know who is using our API, and how, so we can have the metrics we need to support your application and improve the API together.

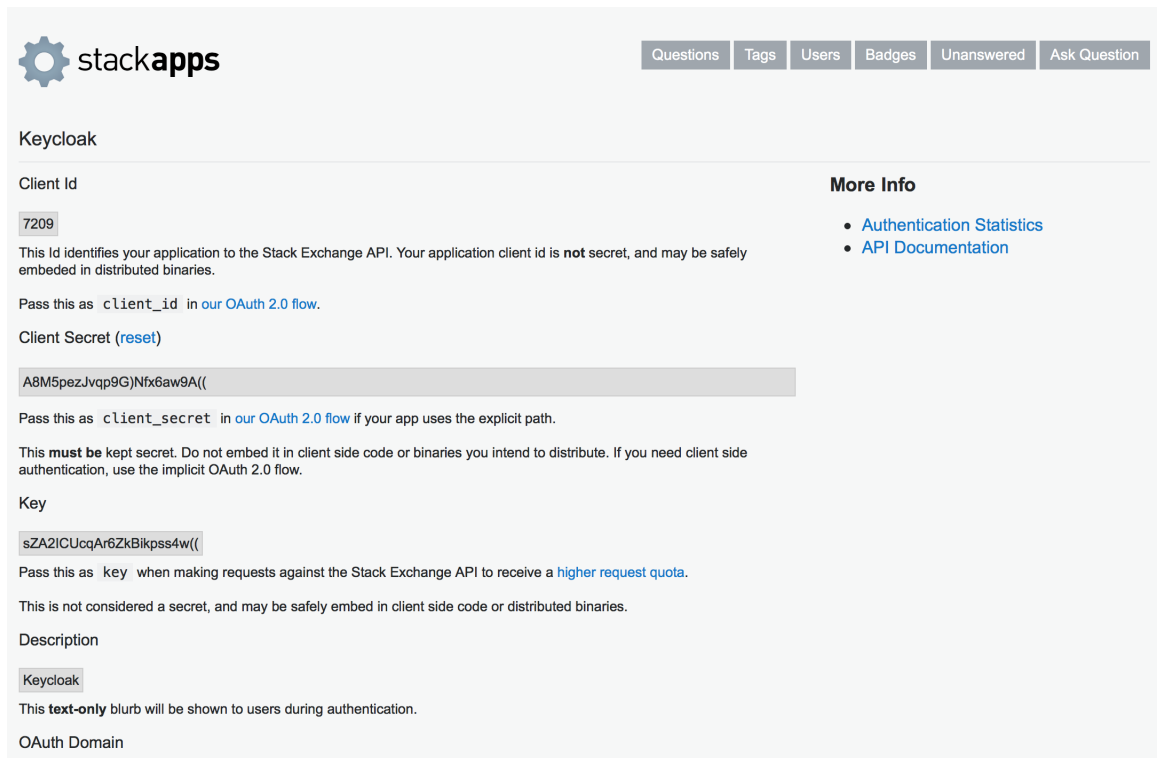
Once it's ready for public consumption, we'll help you promote your registered application here on Stack Apps.

Upon registering, you'll be provided an API key which grants your app a much larger per-day request quota than using the API anonymously.

You'll also receive parameters for authenticating users via OAuth 2.0.

- Application Name フィールドにアプリケーション名を入力します。
- OAuth domain フィールドに OAuth ドメインを入力します。
- Register Your Application をクリックします。

Settings



stackapps

Questions Tags Users Badges Unanswered Ask Question

Keycloak

Client Id

7209

This Id identifies your application to the Stack Exchange API. Your application client id is **not** secret, and may be safely embedded in distributed binaries.

Pass this as `client_id` in our [OAuth 2.0 flow](#).

Client Secret ([reset](#))

A8M5pezJvqp9G)Nfx6aw9A((

Pass this as `client_secret` in our [OAuth 2.0 flow](#) if your app uses the explicit path.

This **must be** kept secret. Do not embed it in client side code or binaries you intend to distribute. If you need client side authentication, use the implicit OAuth 2.0 flow.

Key

sZA2ICUcqAr6ZkBikps4w((

Pass this as `key` when making requests against the Stack Exchange API to receive a [higher request quota](#).

This is not considered a secret, and may be safely embed in client side code or distributed binaries.

Description

Keycloak

This **text-only** blurb will be shown to users during authentication.

OAuth Domain

More Info

- [Authentication Statistics](#)
- [API Documentation](#)

4. **Client Id**、**Client Secret**、および **Key** を書き留めます。
5. Red Hat build of Keycloak で、**Client ID** の値を **Client ID** フィールドにペーストします。
6. Red Hat build of Keycloak で、**Client Secret** の値を **Client Secret** フィールドにペーストします。
7. Red Hat build of Keycloak で、**Key** の値を **Key** フィールドに貼り付けます。
8. **Add** をクリックします。

9.4.13. Twitter

前提条件

1. Twitter 開発者アカウント。



手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **Twitter** を選択します。

アイデンティティプロバイダーの追加

[Identity providers](#) > Add provider

Add Twitter provider

Redirect URI ⓘ	<input type="text" value="http://localhost:8080/realms/master/broker/twitter/endpoint"/>	
Client ID * ⓘ	<input type="text"/>	
Client Secret * ⓘ	<input type="text"/>	
Display order ⓘ	<input type="text"/>	

3. **Redirect URI** の値をクリップボードにコピーします。
4. 別のブラウザタブで、[Twitter Application Management](#) でアプリケーションを作成します。
 - a. App name を入力し、**Next** をクリックします。
 - b. **API Key** および **API Key Secret** の値をメモし、**App settings** をクリックします。
 - c. **User authentication settings** セクションで、**Set up** ボタンをクリックします。
 - d. **Type of App** として **Web App** を選択します。
 - e. **Redirect URL** の値を **Callback URI / Redirect URL** フィールドに貼り付けます。
 - f. **Website URL** の値は、**localhost** 以外の有効な URL にすることができます。
 - g. **Save** をクリックしてから **Done** をクリックします。
5. Red Hat build of Keycloak で、**Consumer Key** の値を **Client ID** フィールドにペーストします。
6. Red Hat build of Keycloak で、**API Key Secret** の値を **Client Secret** フィールドにペーストします。
7. **Add** をクリックします。

9.5. OPENID CONNECT V1.0 アイデンティティプロバイダー

Red Hat build of Keycloak は、OpenID Connect プロトコルに基づいてアイデンティティプロバイダーのブローカーとして機能します。ユーザーを認証しアクセスを認可するには、これらのアイデンティティプロバイダー (IDP) は、仕様によって定義された [Authorization Code Flow](#) をサポートする必要があります。

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **OpenID Connect v1.0** を選択します。

アイデンティティプロバイダーの追加

Master

Identity providers > Add OpenID Connect provider

Add OpenID Connect provider

Redirect URI [ⓘ]

Alias * [ⓘ]

Display name [ⓘ]

Display order [ⓘ]

OpenID Connect settings

Use discovery endpoint [ⓘ] On

Discovery endpoint * [ⓘ]

> Show metadata

Client authentication [ⓘ]

Client ID * [ⓘ]

Client Secret * [ⓘ]

- 初期設定オプションを入力します。設定オプションの詳細については、[一般的な IDP 設定](#) を参照してください。

表9.2 OpenID の接続設定

設定	説明
Authorization URL	OIDC プロトコルが要求する認可 URL エンドポイント。
Token URL	OIDC プロトコルが要求するトークン URL エンドポイント。
Logout URL	OIDC プロトコルのログアウト URL エンドポイント。この値はオプションです。
Backchannel Logout	ユーザーをログアウトするための、IDP へのバックグラウンド (アウトオブバンド) REST リクエスト。一部の IDP は、ブラウザーCookieを使用してセッションを認識するため、ブラウザーリダイレクトによってしかログアウトを実行しません。
User Info URL	OIDC プロトコルが定義するエンドポイント。このエンドポイントは、ユーザープロフィール情報を参照します。

設定	説明
Client Authentication	Red Hat build of Keycloak が Authorization Code Flow で使用するクライアント認証メソッドを定義します。JWT が秘密鍵で署名されている場合、Red Hat build of Keycloak はレルムの秘密鍵を使用します。他の場合は、クライアントシークレットを定義します。詳細は、 クライアント認証の仕様 を参照してください。
Client ID	外部 IDP への OIDC クライアントとして動作するレルム。認可コードフローを使用して外部 IDP と対話する場合は、レルムに OIDC クライアント ID が必要です。
Client Secret	外部 ボールド からのクライアントシークレット。このシークレットは、Authorization Code Flow を使用している場合は必要になります。
Client Assertion Signature Algorithm	クライアント認証として JWT アサーションを作成する署名アルゴリズム。秘密鍵で署名した JWT または jwt としてのクライアントシークレットの場合、必要になります。アルゴリズムを指定しないと、以下のアルゴリズムが適合されます。 RS256 は、秘密鍵で署名した JWT の場合に適合されます。 HS256 は、jwt としてのクライアントシークレットの場合に適合されます。
Issuer	Red Hat build of Keycloak は、IDP からの応答で発行者の要求をこの値と照合して検証します。
Default Scopes	Red Hat build of Keycloak が認証要求とともに送信する OIDC スコープのリスト。デフォルト値は openid です。各スコープはスペースで区切ります。
Prompt	OIDC 仕様の prompt パラメーター。このパラメーターを使用して、再認証およびその他のオプションを強制的に実行できます。詳細は、仕様を参照してください。

設定	説明
Accepts prompt=none forward from client	<p>prompt=none クエリーパラメーターが含まれる転送された認証リクエストを IDP が受け入れるかどうかを指定します。レルムが prompt=none の認証リクエストを受信すると、レルムはユーザーが現在認証されているかを確認し、ユーザーがログインしていない場合は login_required エラーを返します。Red Hat build of Keycloak が認証リクエストのデフォルト IDP を決定する場合 (kc_idp_hint クエリーパラメーターを使用するか、レルムのデフォルト IDP を持つ場合)、prompt=none の認証要求をデフォルトの IDP に転送できます。デフォルトの IDP は、そこにユーザーの認証をチェックします。すべての IDP が prompt=none の要求をサポートしているわけではないため、Red Hat build of Keycloak は、認証要求をリダイレクトする前に、このスイッチを使用してデフォルトの IDP がパラメーターをサポートすることを示します。</p> <p>ユーザーが IDP で認証されていない場合、クライアントは login_required エラーを受け取ります。ユーザーが IDP で認証されている場合、Red Hat build of Keycloak がユーザーの対話を求める認証ページを表示する必要があると、クライアントは interaction_required エラーを受け取る場合があります。この認証には、必須アクション (パスワードの変更など)、合意画面、first broker login フローまたは post broker login フローで表示が設定された画面が含まれます。</p>
Validate Signatures	<p>Red Hat build of Keycloak が、この IDP によって署名された外部 ID トークンの署名を検証するかどうかを指定します。ON の場合、Red Hat build of Keycloak は外部の OIDC IDP の公開鍵を認識する必要があります。パフォーマンス上の目的で、Red Hat build of Keycloak は外部の OIDC アイデンティティプロバイダーの公開鍵をキャッシュします。</p>

設定	説明
Use JWKS URL	このスイッチは、 Validate Signatures が ON の場合に適用されます。Use JWKS URL が ON の場合、Red Hat build of Keycloak は JWKS URL から IDP の公開鍵をダウンロードします。アイデンティティプロバイダーが新しいキーペアを生成すると、新しいキーがダウンロードされます。OFF の場合、Red Hat build of Keycloak はデータベースからの公開鍵 (または証明書) を使用します。そのため、IDP キーペアが変更された場合は、新しいキーを Red Hat build of Keycloak データベースにもインポートします。
JWKS URL	IDP JWK キーの場所をポイントする URL。詳細は、 JWK の仕様 を参照してください。外部の Red Hat build of Keycloak を IDP として使用する場合、仲介された Red Hat build of Keycloak が http://broker-keycloak:8180 で実行されていて、そのレルムが test であれば、 http://broker-keycloak:8180/realms/test/protocol/openid-connect/certs などの URL を使用できます。
Validating Public Key	Red Hat build of Keycloak が外部 IDP 署名を検証するために使用する PEM 形式の公開鍵。このキーは、 Use JWKS URL が OFF の場合に適用されます。
Validating Public Key Id	この設定は、 Use JWKS URL が OFF の場合に適用されます。この設定は、公開鍵の ID を PEM 形式で指定します。キーからキー ID を計算する標準的な方法がないため、外部アイデンティティプロバイダーは Red Hat build of Keycloak が使用するアルゴリズムとは異なるアルゴリズムを使用できます。このフィールドの値が指定されていない場合、Red Hat build of Keycloak は、外部 IDP によって送信されるキー ID に関係なく、すべての要求に検証用の公開鍵を使用します。ON の場合、このフィールドの値がプロバイダーからの署名を検証するために Red Hat build of Keycloak によって使用されるキー ID となり、IDP によって指定されたキー ID と一致する必要があります。

OpenID Provider Metadata を参照する URL またはファイルを指定して、この設定データをすべてインポートできます。Red Hat build of Keycloak の外部 IDP に接続する場合は、`<root>/realms/{realm-name}/well-known/openid-configuration` から IDP 設定をインポートできます。このリンクは、IDP に関するメタデータを記述する JSON ドキュメントです。

プロバイダーで [Json Web Encryption \(JWE\)](#) ID トークンまたは UserInfo 応答を使用する場合、IDP は Red Hat build of Keycloak で使用する公開鍵を知っている必要があります。プロバイダーは、さまざま

な暗号化アルゴリズムに対して定義された [レルムキー](#) を使用してトークンを復号化します。Red Hat build of Keycloak は、IDP がキーを自動ダウンロードするために使用できる標準の [JWKS エンドポイント](#) を提供します。

9.6. SAML V2.0 アイデンティティプロバイダー

for the different encryption algorithms は、SAML v2.0 プロトコルに基づいてブローカーアイデンティティプロバイダーを使用できます。

手順

1. メニューで **Identity Providers** をクリックします。
2. **Add provider** リストから **SAML v2.0** を選択します。

アイデンティティプロバイダーの追加

3. 初期設定オプションを入力します。設定オプションの詳細については、[一般的な IDP 設定](#) を参照してください。

表9.3 SAML 設定

設定	説明
Service Provider Entity ID	リモートアイデンティティプロバイダーがこのサービスプロバイダーからのリクエストを識別するために使用する SAML エンティティ ID。デフォルトでは、この設定はレルムベース URL <code><root>/realms/{realm-name}</code> に設定されます。
Identity Provider Entity ID	受信した SAML アサーションの発行者を検証するために使用されるエンティティ ID。空の場合、発行者の検証は実行されません。

設定	説明
Single Sign-On Service URL	認証プロセスを開始する SAML エンドポイント。SAML IDP が IDP エンティティ記述子を公開する場合、このフィールドの値はそこで指定されます。
Single Logout Service URL	SAML ログアウトエンドポイント。SAML IDP が IDP エンティティ記述子を公開する場合、このフィールドの値はそこで指定されます。
Backchannel Logout	SAML IDP がバックチャネルのログアウトに対応している場合は、このスイッチを ON に切り替えます。
NameID Policy Format	名前識別子の形式に対応する URI 参照。Red Hat build of Keycloakは、デフォルトでこれを urn:oasis:names:tc:SAML:2.0:nameid-format:persistent に設定します。
Principal Type	外部ユーザー ID の特定および追跡に使用される SAML アサーションの一部を指定します。Subject NameID または SAML 属性のいずれかを指定できます (名前または分かりやすい名前のいずれか)。Subject NameID の値は、'urn:oasis:names:tc:SAML:2.0:nameid-format:transient' NameID Policy Format の値と共に設定することはできません。
Principal Attribute	Principal Type が空欄でない場合は、このフィールドで識別する属性の名前 (Attribute [Name]) または分かりやすい名前 (Attribute [Friendly Name]) を指定します。
Allow create	外部アイデンティティプロバイダーがプリンシパルを表す新しい識別子を作成するのを許可します。
HTTP-POST Binding Response	外部 IDP によって送信される SAML リクエストに回答する SAML バインディングを制御します。 OFF の場合、Red Hat build of Keycloak は Redirect Binding を使用します。
HTTP-POST Binding for AuthnRequest	外部 IDP からの認証を要求するときに SAML バインディングを制御します。 OFF の場合、Red Hat build of Keycloak は Redirect Binding を使用します。
Want AuthnRequests Signed	ON の場合、Red Hat build of Keycloak はレルムのキーペアを使用して、外部 SAML IDP に送信される要求に署名します。
Want Assertions Signed	このサービスプロバイダーが署名付きアサーションを期待しているかどうかを示します。

設定	説明
Want Assertions Encrypted	このサービスプロバイダーが暗号化されたアサーションを期待しているかどうかを示します。
Signature Algorithm	Want AuthnRequests Signed が ON の場合に、使用する署名アルゴリズム。 SHA1 ベースのアルゴリズムは非推奨となっており、将来のリリースでは削除される可能性があることに注意してください。*_ SHA1 の代わりに、より安全なアルゴリズムを使用することを推奨します。また、*_ SHA1 アルゴリズムでは、SAML ID プロバイダー (Red Hat build of Keycloak の別のインスタンスなど) が Java 17 以降で実行されている場合、署名の検証が機能しません。
Encryption Algorithm	SAML ドキュメント、アサーション、または ID の暗号化のために SAML IDP によって使用される暗号化アルゴリズム。SAML ドキュメント部分の復号化に対応する復号化キーは、この設定されたアルゴリズムに基づいて選択され、暗号化 (ENC) の使用にレルムキーで使用できる必要があります。アルゴリズムが設定されていない場合は、サポートされているアルゴリズムが許可され、SAML ドキュメント自体で指定されているアルゴリズムに基づいて復号キーが選択されます。
SAML Signature Key Name	POST バインディングを使用して送信された署名付き SAML ドキュメントには、 KeyName 要素に署名キーの識別が含まれており、デフォルトでは Red Hat build of Keycloak キー ID が含まれています。外部 SAML IDP には異なるキー名が必要です。このスイッチは、 KeyName に次のものが含まれるかどうかを制御します。* KEY_ID - キー ID* CERT_SUBJECT - レルムキーに対応する証明書からのサブジェクト。Microsoft Active Directory Federation Services には CERT_SUBJECT が必要です。* NONE - Red Hat build of Keycloak では、SAML メッセージからキー名のヒントが省略されます。
Force Authentication	ユーザーがすでにログインしている場合でも、ユーザーは外部の IDP に認証情報を入力する必要があります。
Validate Signature	ON の場合、レルムには SAML リクエストおよびデジタル署名する外部 IDP からの応答が必要です。
Validating X509 Certificate	Red Hat build of Keycloak が外部 IDP からの SAML リクエストとレスポンスの署名を検証するために使用するパブリック証明書。

設定	説明
Sign Service Provider Metadata	ON の場合、Red Hat build of Keycloak はレールのキーペアを使用して SAML Service Provider Metadata descriptor に署名します。
Pass subject	Red Hat build of Keycloak が、 login_hint クエリパラメーターを IDP に転送するかどうかを制御します。Red Hat build of Keycloak は、このフィールドの値を AuthnRequest の Subject の login_hint パラメーターに追加し、宛先プロバイダーがログインフォームを事前に入力できるようにします。
属性消費サービスインデックス	リモート IDP に要求する属性セットを識別します。Red Hat build of Keycloak は、ID プロバイダー設定でマップされた属性を自動生成された SP メタデータドキュメントに自動的に追加します。
サービス名を使用している属性	自動生成される SP メタデータドキュメントでアドバタイズされる属性セットの説明的な名前。

外部 IDP の SAML IDP エンティティ記述子をポイントする URL またはファイルを指定することで、すべての設定データをインポートできます。Red Hat build of Keycloak の外部 IDP に接続する場合は、URL `<root>/realms/{realm-name}/protocol/saml/descriptor` から IDP 設定をインポートできます。このリンクは、IDP に関するメタデータを記述する XML ドキュメントです。接続先の外部 SAML IDP のエンティティ記述子をポイントする URL または XML ファイルを指定することで、このすべての設定データをインポートすることもできます。

9.6.1. 特定の AuthnContexts の要求

アイデンティティプロバイダーは、クライアントがユーザーアイデンティティを検証する認証方法の制約を指定するのを容易にします。たとえば、MFA、Kerberos 認証、またはセキュリティ要件を要求します。これらの制約は、特定の AuthnContext 条件を使用します。クライアントは1つまたは複数の条件を要求し、アイデンティティプロバイダーがどの程度要求された AuthnContext と一致しなければならないか (完全に、または他の同等の条件を満たしながら) を指定できます。

Requested AuthnContext Constraints セクションの ClassRefs または DeclRefs を追加して、サービスプロバイダーが必要とする条件をリストにして設定できます。通常、ClassRefs または DeclRefs のいずれかを指定する必要があるため、どの値がサポートされるかをアイデンティティプロバイダーのドキュメントで確認してください。ClassRefs または DeclRefs が存在しない場合、アイデンティティプロバイダーは追加の制約を適用しません。

表9.4 要求される AuthnContext 制約

設定	説明
----	----

設定	説明
Comparison	アイデンティティプロバイダーがコンテキスト要件を評価するために使用する方法。設定可能な値は、 Exact 、 Minimum 、 Maximum 、または Better です。デフォルト値は Exact です。
AuthnContext ClassRefs	必要な基準を記述する AuthnContext ClassRefs。
AuthnContext DeclRefs	必要な基準を記述する AuthnContext DeclRefs。

9.6.2. SP 記述子

プロバイダーの SAML SP メタデータにアクセスする場合は、アイデンティティプロバイダー設定で **Endpoints** 項目を探します。これには、サービスプロバイダーの SAML エンティティ記述子を生成する **SAML 2.0 Service Provider Metadata** のリンクが含まれます。記述子をダウンロードするか、その URL をコピーして、それをリモートの ID プロバイダーにインポートすることができます。

このメタデータは、以下の URL で一般に公開されています。

```
http[s]://{host:port}/realms/{realm-name}/broker/{broker-alias}/endpoint/descriptor
```

記述子にアクセスする前に設定変更を保存するようにしてください。

9.6.3. SAML リクエストのサブジェクトの送信

デフォルトでは、SAML アイデンティティプロバイダーをポイントするソーシャルボタンは、ユーザーを以下のログイン URL にリダイレクトします。

```
http[s]://{host:port}/realms/${realm-name}/broker/{broker-alias}/login
```

この URL に **login_hint** という名前のクエリーパラメーターを追加すると、パラメーターの値が Subject 属性として SAML リクエストに追加されます。このクエリーパラメーターが空の場合、Red Hat build of Keycloak はリクエストにサブジェクトを追加しません。

Pass subject オプションを有効にして SAML リクエストのサブジェクトを送信します。

9.7. クライアント提案されたアイデンティティプロバイダー

OIDC アプリケーションは、使用するアイデンティティプロバイダーを示唆することで Red Hat build of Keycloak ログインページをバイパスできます。これを有効にするには、Authorization Code Flow 認可エンドポイントに **kc_idp_hint** クエリーパラメーターを設定します。

Red Hat build of Keycloak OIDC クライアントアダプターを使用すると、アプリケーション内の保護されたリソースにアクセスする際にこのクエリーパラメーターを指定できます。

以下に例を示します。

```
GET /myapplication.com?kc_idp_hint=facebook HTTP/1.1
Host: localhost:8080
```

この場合、レルムには **facebook** エイリアスを持つアイデンティティプロバイダーが必要です。このプロバイダーが存在しない場合は、ログインフォームが表示されます。

keycloak.js アダプターを使用している場合は、以下のように同じ動作を得ることもできます。

```
const keycloak = new Keycloak('keycloak.json');

keycloak.createLoginUrl({
  idpHint: 'facebook'
});
```

kc_idp_hint クエリーパラメーターを使用すると、**Identity Provider Redirector** オーセンティケーターにアイデンティティプロバイダーを設定した場合、クライアントはデフォルトのプロバイダーを上書きできます。**kc_idp_hint** クエリーパラメーターを空値に設定すると、クライアントは自動リダイレクトを無効にできます。

9.8. クレームとアサーションのマッピング

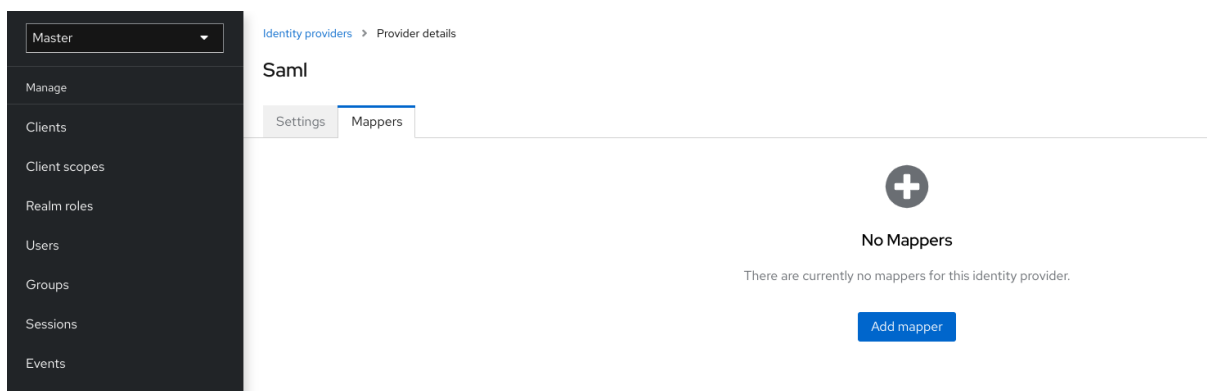
認証する外部 IDP が提供する SAML および OpenID Connect メタデータを、レルムにインポートできます。インポート後に、ユーザープロファイルのメタデータやその他の情報を抽出でき、アプリケーションが利用できるようにすることができます。

外部アイデンティティプロバイダー経由でレルムにログインする各ユーザーには、SAML または OIDC アサーションおよび要求からのメタデータに基づき、ローカルの Red Hat build of Keycloak データベースにエントリーがあります。

手順

1. メニューで **Identity Providers** をクリックします。
2. リストでアイデンティティプロバイダーを1つ選択します。
3. **Mappers** タブをクリックします。

アイデンティティプロバイダーマッパー



4. **Add mapper** をクリックします。

アイデンティティプロバイダーマッパー

5. **Sync Mode Override** の値を選択します。マッパーは、この設定に従ってユーザーが繰り返しログインする際に、ユーザー情報を更新します。
 - a. 以前の Red Hat build of Keycloak バージョンの動作を使用するには、**legacy** を選択します。
 - b. 特定のアイデンティティプロバイダーを使用して Red Hat build of Keycloak に初めてログインする際に、Red Hat build of Keycloak でユーザーが初めて作成されたところにデータをインポートするには、**import** を選択します。
 - c. 各ユーザーログイン時にユーザーデータを更新するには、**force** を選択します。
 - d. アイデンティティプロバイダーで設定された同期モードを使用するには、**inherit** を選択します。その他のオプションはすべて、この同期モードを上書きします。
6. **Mapper Type** リストからマッパーを選択します。Mapper Type にカーソルを合わせ、マッパーの説明とマッパーに入力する設定を確認します。
7. **Save** をクリックします。

JSON ベースの要求では、ドット表記を使用して、インデックスでアレイフィールドにアクセスするため、ネスト化と角括弧を使用します。たとえば、**contact.address[0].country** です。

ソーシャルプロバイダーが提供するユーザープロフィール JSON データの構造を調べるには、サーバーを起動する際の **DEBUG** レベルのロガー **org.keycloak.social.user_profile_dump** を有効にします。

9.9. 利用可能なユーザーセッションデータ

外部 IDP からユーザーがログインすると、Red Hat build of Keycloak は、アクセス可能なユーザーセッションノートデータを保存します。このデータは、適切なクライアントマッパーを使用してクライアントに渡されるトークンまたは SAML アサーションを介してログインを要求するクライアントに反映できません。

identity_provider

ログインの実行に使用されるブローカーの IDP エイリアス。

identity_provider_identity

現在認証されているユーザーの IDP ユーザー名。常にではありませんが、多くの場合は Red Hat

build of Keycloak ユーザー名と同じです。たとえば、Red Hat build of Keycloak は、john というユーザーを Facebook ユーザー **john123@gmail.com** とリンクできます。この場合、ユーザーセッションノートの値は **john123@gmail.com** です。

タイプ **User Session Note** の **プロトコルマッパー** を使用して、この情報をクライアントに伝達できます。

9.10. FIRST LOGIN FLOW

ユーザーがアイデンティティ仲介を使用してログインすると、Red Hat build of Keycloak はレルムのローカルデータベース内のユーザーの要素をインポートし、リンクします。Red Hat build of Keycloak が外部アイデンティティプロバイダーを介してユーザーを正常に認証すると、以下の2つの状況が発生する可能性があります。

- Red Hat build of Keycloak は、すでにユーザーアカウントをインポートし、認証されたアイデンティティプロバイダーアカウントにリンクしている。この場合、Red Hat build of Keycloak は既存のユーザーとして認証を行い、アプリケーションにリダイレクトします。
- Red Hat build of Keycloak に、このユーザーのアカウントが存在しない。通常、新しいアカウントを登録して Red Hat build of Keycloak データベースにインポートしますが、同じメールアドレスを持つ既存の Red Hat build of Keycloak アカウントが存在する場合があります。既存のローカルアカウントを外部アイデンティティプロバイダーに自動的にリンクすることは、セキュリティホールとなる可能性があります。外部アイデンティティプロバイダーから取得する情報を常に信頼することはできません。

このような状況に対処する場合、組織ごとに異なる要件があります。Red Hat build of Keycloak では、IDP 設定の **First Login Flow** オプションを使用して、外部 IDP から初めてログインするユーザーの **ワークフロー** を選択できます。デフォルトでは、**First Login Flow** オプションは **first broker login** フローをポイントしますが、独自のフローを使用することや、アイデンティティプロバイダーごとに異なるフローを使用することができます。

フローは管理コンソールの **Authentication** タブにあります。**First Broker Login** フローを選択すると、デフォルトで使用されるオーセンティケーターが表示されます。既存のフローを再設定できます。たとえば、一部のオーセンティケーターを無効にすることや、一部を **required** と識別することや、一部のオーセンティケーターを設定することができます。

9.10.1. デフォルトの first login flow のオーセンティケーター

プロファイルの確認

- このオーセンティケーターはプロファイル情報ページを表示するため、ユーザーは Red Hat build of Keycloak がアイデンティティプロバイダーから取得するプロファイルを確認できます。
- **Actions** メニューで **Update Profile On First Login** オプションを設定できます。
- **ON** の場合、ユーザーにはプロファイルページが表示され、ユーザーのアイデンティティのフェデレーションを行うために追加情報が要求されます。
- **missing** 場合、アイデンティティプロバイダーがメール、名、姓などの必須情報を提供しない場合、ユーザーにはプロファイルページが表示されます。
- **OFF** の場合、ユーザーが後で **Confirm Link Existing Account** オーセンティケーターによって表示されるページの **Review profile info** リンクをクリックしない限り、プロファイルページは表示されません。

Create User If Unique

このオーセンティケーターは、アイデンティティプロバイダーからのアカウントと同じメールまたはユーザー名を持つ既存の Red Hat build of Keycloak アカウントがすでにあるかどうかを確認します。ない場合、オーセンティケーターは新しいローカルの Red Hat build of Keycloak アカウントを作成し、それをアイデンティティプロバイダーとリンクし、フロー全体を終了します。それ以外の場合は、次の **Handle Existing Account** サブフローが処理されます。重複アカウントがないことを確認する場合は、このオーセンティケーターを **REQUIRED** とマークできます。この場合、既存の Red Hat build of Keycloak アカウントが存在すれば、ユーザーにはエラーページが表示され、ユーザーはアカウント管理を通じてアイデンティティプロバイダーアカウントをリンクする必要があります。

- このオーセンティケーターは、アイデンティティプロバイダーのアカウントと同じメールまたはユーザー名を持つ Red Hat build of Keycloak アカウントがすでに存在するか確認します。
- アカウントが存在しない場合、オーセンティケーターはローカルの Red Hat build of Keycloak アカウントを作成し、このアカウントをアイデンティティプロバイダーとリンクさせ、フローを終了します。
- アカウントが存在する場合、オーセンティケーターは次の **Handle Existing Account** サブフローを実装します。
- 重複アカウントがないようにするには、このオーセンティケーターを **REQUIRED** とマークします。Red Hat build of Keycloak アカウントが存在する場合、ユーザーにはエラーページが表示され、ユーザーはアカウント管理を通じてアイデンティティプロバイダーアカウントをリンクする必要があります。

既存のアカウントのリンクの確認

- 情報ページで、ユーザーには同じメールアドレスを持つ Red Hat build of Keycloak アカウントが表示されます。ユーザーはプロフィールを再度確認し、別の電子メールまたはユーザー名を使用できます。フローが再起動され、**Review Profile** オーセンティケーターに戻ります。
- あるいは、ユーザーはアイデンティティプロバイダーアカウントを既存の Keycloak アカウントにリンクすることを確認できます。
- ユーザーにこの確認ページを表示させず、直接メール検証または再認証によりアイデンティティプロバイダーのアカウントをリンクさせるには、このオーセンティケーターを無効にします。

既存のアカウントをメールで検証

- このオーセンティケーターは、デフォルトで **ALTERNATIVE** です。Red Hat build of Keycloak は、レルムに SMTP が設定されている場合にこのオーセンティケーターを使用します。
- オーセンティケーターはユーザーにメールを送信し、アイデンティティプロバイダーを Red Hat build of Keycloak アカウントにリンクすることを確認します。
- メールによるリンクを確認せず、ユーザーをそのパスワードで再認証する方が望ましい場合には、このオーセンティケーターを無効にします。

再認証による既存のアカウントの確認

- メールオーセンティケーターが利用できない場合には、このオーセンティケーターを使用し

ます。たとえば、レルムに SMTP を設定していない場合。このオーセンティケーターは、Red Hat build of Keycloak アカウントをアイデンティティプロバイダーにリンクすることを認証するログイン画面をユーザーに表示します。

- ユーザーは、すでに Red Hat build of Keycloak アカウントにリンクされている別のアイデンティティプロバイダーで再認証することもできます。
- ユーザーに OTP の使用を強制することができます。それ以外の場合は、任意で、ユーザーアカウントに OTP を設定した場合に使用されます。

9.10.2. 既存の first login flow の自動リンク



警告

AutoLink オーセンティケーターは、ユーザーが任意のユーザー名またはメールアドレスを使用して自己登録できる一般の環境では危険です。慎重にユーザー登録を管理し、ユーザー名とメールアドレスを割り当てない限り、このオーセンティケーターは使用しないでください。

プロンプトなしでユーザーを自動的にリンクする first login flow を設定するには、以下の2つのオーセンティケーターで新しいフローを作成します。

Create User If Unique

この認証システムにより、Red Hat build of Keycloak は一意のユーザーを処理できるようになります。オーセンティケーターの要件を **Alternative** に設定します。

Automatically Set Existing User

このオーセンティケーターは、検証なしで既存のユーザーを認証コンテキストに設定します。オーセンティケーター要件を **Alternative** に設定します。



注記

利用できる中ではこの設定が最も単純ですが、他のオーセンティケーターを使用することができます。たとえば、* エンドユーザーがプロフィール情報を確認する場合は、レビュープロフィールオーセンティケーターをフローの開始に追加できます。* このフローに認証メカニズムを追加し、ユーザーがクレデンシャルを検証するように強制することができます。認証メカニズムを追加するには、複雑なフローが必要です。たとえば、Alternative サブフローで Automatically Set Existing User および Password Form を Required として設定できます。

9.10.3. 自動ユーザー作成の無効化

デフォルトの初回ログインフローでは、外部アイデンティティに一致する Red Hat build of Keycloak アカウントを検索し、リンクを提案します。一致する Red Hat build of Keycloak アカウントが存在しない場合、フローによって自動的に作成されます。

このデフォルトの動作は、一部のセットアップでは適切ではない可能性があります。1つの例は、すべてのユーザーが事前に作成されている、読み取り専用の LDAP ユーザーストアを使用する場合です。この場合は、自動ユーザー作成をオフにする必要があります。

ユーザーの作成を無効にするには、以下を実行します。

手順

1. メニューで **Authentication** をクリックします。
2. リストから **First Broker Login** を選択します。
3. **Create User if Unique** を **DISABLED** に設定します。
4. **Confirm Link Existing Account** を **DISABLED** に設定します。

これを設定すると、Red Hat build of Keycloak 自体では、どの内部アカウントが外部アイデンティティに対応するか判断できません。そのため、**Verify Existing Account By Re-authentication** オーセンティケーターにより、ユーザー名およびパスワードが提供されます。



注記

ID プロバイダーによるユーザー作成の有効化または無効化は、レルムの [User Registration switch](#) とは完全に独立しています。ID プロバイダーによるユーザーの作成を有効にし、同時にレルムのログイン設定でユーザーの自己登録を無効にすることも、その逆を行うこともできます。

9.10.4. 既存ユーザーの最初のログインフローの検出

以下の条件で、最初のログインフローを設定するには、

- このレルムにすでに登録されているユーザーのみがログインできる。
- ユーザーはプロンプトなしで自動的にリンクされる。

以下の2つのオーセンティケーターで新しいフローを作成します。

Detect Existing Broker User

このオーセンティケーターは、一意のユーザーが処理されるようにします。オーセンティケーターの要件を **REQUIRED** に設定します。

Automatically Set Existing User

検証なしで既存のユーザーを認証コンテキストに自動的に設定します。オーセンティケーターの要件を **REQUIRED** に設定します。

アイデンティティプロバイダー設定の **First Login Flow** をそのフローに設定する必要があります。ユーザープロフィール (姓、名など) をアイデンティティプロバイダー属性で更新する場合は、**Sync Mode** を **force** に設定することもできます。



注記

このフローは、アイデンティティを他のアイデンティティプロバイダー (GitHub、Facebook など) に移譲するが、ログインできるユーザーを管理する場合に使用できません。

この設定では、Red Hat build of Keycloak は外部アイデンティティに対応する内部アカウントを判断できません。**Verify Existing Account By Re-authentication** オーセンティケーターにより、プロバイダーにユーザー名とパスワードの入力が求められます。

9.11. 外部 IDP トークンの取得

Red Hat build of Keycloak を使用すると、IDP の設定ページで **Store Token** 設定オプションを使用して、外部 IDP との認証プロセスからのトークンと応答を保存できます。

アプリケーションコードは、これらのトークンと応答を取得し、追加のユーザー情報をインポートしたり、外部 IDP を安全に要求したりすることができます。たとえば、アプリケーションは Google トークンを使用して他の Google サービスおよび REST API を使用できます。特定のアイデンティティプロバイダーのトークンを取得するには、以下のようにリクエストを送信します。

```
GET /realms/{realm}/broker/{provider_alias}/token HTTP/1.1
Host: localhost:8080
Authorization: Bearer <KEYCLOAK ACCESS TOKEN>
```

アプリケーションは Red Hat build of Keycloak で認証し、アクセストークンを受け取る必要があります。このアクセストークンには **broker** クライアントレベルのロール **read-token** が設定されている必要があるため、ユーザーにこのロールのロールマッピングが必要であり、クライアントアプリケーションにはそのスコープ内でそのロールが必要です。この場合は Red Hat build of Keycloak で保護されたサービスにアクセスするため、ユーザー認証時に Red Hat build of Keycloak が発行するアクセストークンを送信します。ブローカー設定ページで、**Stored Tokens Readable** スイッチを **ON** に設定して、このロールを新たにインポートされたユーザーに割り当てることができます。

これらの外部トークンは、プロバイダーを介して再度ログインするか、クライアントが開始したアカウントリンク API を使用して再確立できます。

9.12. アイデンティティブローカーのログアウト

ログアウトする場合、Red Hat build of Keycloak は、最初にログインするのに使用した外部アイデンティティプロバイダーにリクエストを送信し、このアイデンティティプロバイダーからユーザーをログアウトします。この動作を省略し、外部アイデンティティプロバイダーからのログアウトを回避することができます。詳細は、[アダプターのログアウトドキュメント](#) を参照してください。

第10章 SSO プロトコル

このセクションでは、認証プロトコル、Red Hat build of Keycloak 認証サーバー、および Red Hat build of Keycloak 認証サーバーで保護されたアプリケーションがどのようにこれらのプロトコルと対話するかを説明します。

10.1. OPENID CONNECT

OpenID Connect (OIDC) は、OAuth 2.0 の拡張機能である認証プロトコルです。

OAuth 2.0 は認可プロトコルを構築するためのフレームワークで、不完全です。一方、OIDC は、JSON Web Token (JWT) 標準を使用する完全な認証および認可プロトコルです。JWT 標準は、アイデンティティトークンの JSON 形式を定義し、コンパクトで Web フレンドリーにデータをデジタル署名および暗号化する方法を定義しています。

通常、OIDC は2つのユースケースを実装します。最初のケースは、Red Hat build of Keycloak サーバーがユーザーを認証することを要求するアプリケーションです。ログインに成功すると、アプリケーションは ID トークンと アクセストークン を受け取ります。ID トークンには、ユーザー名、電子メール、プロフィール情報などのユーザー情報が含まれています。レルムは、アクセス情報 (ユーザーロールマッピングなど) が含まれる アクセストークン にデジタル署名します。アプリケーションは、この情報を使用してユーザーがアプリケーションでアクセスできるリソースを判断します。

2つ目のユースケースは、リモートサービスにアクセスするクライアントです。

- クライアントは Red Hat build of Keycloak から アクセストークン を要求し、ユーザーの代わりにリモートサービスを呼び出します。
- Red Hat build of Keycloak はユーザーを認証し、要求元のクライアントへのアクセスを許可するための同意をユーザーに求めます。
- クライアントは、レルムによってデジタル署名される アクセストークン を受け取ります。
- クライアントは、アクセストークン を使用して、リモートサービスで REST 要求を行います。
- リモート REST サービスは アクセストークン を抽出します。
- リモート REST サービスは、トークンの署名を検証します。
- リモート REST サービスは、トークン内のアクセス情報に基づいて、リクエストを処理するか拒否するかを決定します。

10.1.1. OIDC 認証フロー

OIDC には、複数のメソッド (またはフロー) があり、クライアントまたはアプリケーションがユーザーを認証し、アイデンティティ および アクセストークン を受け取るために使用できます。このメソッドは、アクセスを要求するアプリケーションまたはクライアントのタイプによって異なります。

10.1.1.1. 認可コードフロー

認可コードフローはブラウザーベースのプロトコルで、ブラウザーベースのアプリケーションの認証および認可に適しています。ブラウザーのリダイレクトを使用してアイデンティティおよびアクセストークンを取得します。

1. ユーザーは、ブラウザーを使用してアプリケーションに接続します。アプリケーションは、ユーザーがアプリケーションにログインしていないことを検出します。

2. アプリケーションは、認証のためにブラウザーを Red Hat build of Keycloak にリダイレクトします。
3. アプリケーションは、コールバック URL をブラウザーリダイレクトのクエリーパラメーターとして渡します。Red Hat build of Keycloak は、認証に成功するとパラメーターを使用します。
4. Red Hat build of Keycloak がユーザーを認証し、有効期間が短い1回限りの一時コードを作成します。
5. Red Hat build of Keycloak はコールバック URL を使用してアプリケーションにリダイレクトし、一時コードをコールバック URL のクエリーパラメーターとして追加します。
6. アプリケーションは一時コードを抽出し、Red Hat build of Keycloak にバックグラウンド REST 呼び出しを行い、**アイデンティティ**、**アクセス**、**更新** トークンのコードを交換します。再生攻撃を防止するため、一時コードは複数回使用できません。



注記

システムは、トークンが有効な間、そのトークンの盗難に対して脆弱です。セキュリティおよびスケーラビリティの理由から、アクセストークンは通常すぐに期限切れになるように設定されるため、後続のトークンリクエストは失敗します。トークンの有効期限が切れると、アプリケーションはログインプロトコルによって送信される追加の**更新** トークンを使用して新しいアクセストークンを取得できます。

機密性が要求される クライアントは、トークンの一時コードを交換する際にクライアントシークレットを提供します。**パブリック** クライアントは、クライアントシークレットの提供を要求されません。**パブリック** クライアントは、HTTPS が厳格に適用され、クライアントに登録されているリダイレクト URI が厳格に制御される場合に保護されます。HTML5/JavaScript クライアントは、クライアントシークレットを HTML5/JavaScript クライアントへ安全に送信する方法がないため、**パブリック** クライアントである必要があります。詳細は、[クライアントの管理](#) の章を参照してください。

Red Hat build of Keycloak は、[Proof Key for Code Exchange](#) 仕様もサポートします。

10.1.1.2. Implicit Flow

インプリシットフローはブラウザーベースのプロトコルです。これは Authorization Code Flow と似ていますが、要求が少なく、更新トークンはありません。



注記

トークンがリダイレクト URI で送信されると、**アクセス** トークンがブラウザーの履歴に漏洩する可能性があります (以下を参照)。

また、このフローはクライアントに更新トークンを提供しません。したがって、アクセストークンの有効期限を長くするか、期限切れになったらユーザーを再認証する必要があります。

このフローの使用を推奨していません。このフローは OIDC および OAuth 2.0 仕様にあるためサポートされます。

このプロトコルは以下のように動作します。

1. ユーザーは、ブラウザーを使用してアプリケーションに接続します。アプリケーションは、ユーザーがアプリケーションにログインしていないことを検出します。

2. アプリケーションは、認証のためにブラウザーを Red Hat build of Keycloak にリダイレクトします。
3. アプリケーションは、コールバック URL をブラウザーリダイレクトのクエリーパラメーターとして渡します。認証に成功すると、Red Hat build of Keycloak はクエリーパラメーターを使用します。
4. Red Hat build of Keycloak はユーザーを認証し、**アイデンティティー** および **アクセス トークン** を作成します。Red Hat build of Keycloak はコールバック URL を使用してアプリケーションにリダイレクトし、さらに **アイデンティティー** および **アクセス トークン** をコールバック URL のクエリーパラメーターとして追加します。
5. アプリケーションはコールバック URL から **identity** および **access** トークンを抽出します。

10.1.1.3. リソースオーナーパスワードクレデンシャルの付与 (直接アクセスグラント)

Direct Access Grants は、ユーザーの代わりにトークンを取得するために REST クライアントによって使用されます。これは、以下の項目を含む HTTP POST リクエストです。

- ユーザーの認証情報。認証情報はフォームパラメーターで送信されます。
- クライアントの ID。
- クライアントシークレット (機密性が要求されるクライアントの場合)。

HTTP 応答には、**アイデンティティー**、**アクセス**、および **更新** トークンが含まれます。

10.1.1.4. クライアント認証情報の付与

Client Credentials Grant は、外部ユーザーの代わりに機能するトークンを取得するのではなく、クライアントに関連付けられたサービスアカウントのメタデータおよびパーミッションに基づいてトークンを作成します。**Client Credentials Grants** は REST クライアントによって使用されます。

詳細については、[サービスアカウント](#) の章を参照してください。

10.1.2. リフレッシュトークンの付与

デフォルトでは、Red Hat build of Keycloak は、ほとんどのフローからのトークン応答で更新トークンを返します。いくつかの例外としては、上で説明した暗黙的なフローまたはクライアント認証情報の付与があります。

リフレッシュトークンは、SSO ブラウザーセッションのユーザーセッションに関連付けられており、ユーザーセッションの有効期間中有効です。ただし、そのクライアントは指定された間隔ごとに少なくとも1回はリフレッシュトークン要求を送信する必要があります。そうでない場合、セッションは "アイドル" とみなされ、期限切れになる可能性があります。詳細は、[タイムアウトのセクション](#) を参照してください。

Red Hat build of Keycloak は [offline tokens](#) をサポートしています。これは通常、対応するブラウザー SSO セッションがすでに期限切れであっても、クライアントがリフレッシュトークンを使用する必要がある場合に使用できます。

10.1.2.1. リフレッシュトークンのローテーション

リフレッシュトークンは、一度使用すると無効とみなされるように指定できます。つまり、すでに使用されている古いリフレッシュトークンは、Red Hat build of Keycloak では有効とは見なされなくなるため、クライアントは常に最後のリフレッシュレスポンスからリフレッシュトークンを保存する必要があります。

ります。これは、[タイムアウトセクション](#) で指定されているように、**Revoke Refresh token** オプションを使用して設定できます。

Red Hat build of Keycloak は、リフレッシュトークンのローテーションが存在しない状況もサポートします。この場合、ログイン時にリフレッシュトークンが返されますが、リフレッシュトークン要求からの後続の応答では新しいリフレッシュトークンは返されません。この方法は、たとえば [FAPI 2 ドラフト仕様](#) で推奨されています。Red Hat build of Keycloak では、[クライアントポリシー](#) を使用してリフレッシュトークンのローテーションをスキップできます。一部のクライアントプロファイルに executor の **suppress-refresh-token-rotation** を追加し、クライアントポリシーを設定して、どのクライアントに対してプロファイルがトリガーされるかを指定できます。つまり、それらのクライアントでは、更新トークンのローテーションがスキップされます。

10.1.2.2. デバイス認可の付与

これは、入力機能が制限されているか、適切なブラウザがないインターネット接続デバイスで実行されているクライアントによって使用されます。以下は、プロトコルの概要です。

1. アプリケーションは、Red Hat build of Keycloak にデバイスコードとユーザーコードを要求します。Red Hat build of Keycloak は、デバイスコードとユーザーコードを作成します。Red Hat build of Keycloak は、デバイスコードおよびユーザーコードなどの応答をアプリケーションに返します。
2. アプリケーションはユーザーにユーザーコードと検証 URI を提供します。ユーザーは検証 URI にアクセスし、別のブラウザを使用して認証されます。プロキシ内の Keycloak 外部の Keycloak 検証 URI (/realms/realm_name/device) にリダイレクトされる短い verify_uri を定義できます。
3. アプリケーションは Red Hat build of Keycloak に繰り返しポーリングを行い、ユーザーがユーザー認可を完了したか確認します。ユーザー認証が完了すると、アプリケーションは **アイデンティティ**、**アクセス**、**更新** のトークンのデバイスコードを交換します。

10.1.2.3. クライアントが開始したバックチャネル認証の付与

この機能は、OAuth 2.0 の認可コード付与のようなユーザーのブラウザを介したリダイレクトを行わずに、OpenID プロバイダーと直接通信して認証フローを開始したいクライアントによって使用されます。以下は、プロトコルの概要です。

1. クライアントは、クライアントによる認証要求を識別する auth_req_id を Red Hat build of Keycloak に要求します。Red Hat build of Keycloak は auth_req_id を作成します。
2. このクライアントは、auth_req_id を受信してからユーザーが認証されるまで、auth_req_id と引き換えに Red Hat build of Keycloak からアクセストークン、更新トークン、および ID トークンを取得するため、Red Hat build of Keycloak を繰り返しポーリングする必要があります。

管理者は、Client Initiated Backchannel Authentication (CIBA) 関連の操作をレルムごとに **CIBA ポリシー** として設定できます。

また、アプリケーションとサービスの保護ガイドの [バックチャネル認証エンドポイント](#) および [クライアントが介したバックチャネル認証付与](#) など、Red Hat build of Keycloak ドキュメントの他の箇所も参照してください。

10.1.2.3.1. CIBA ポリシー

管理者は、**Admin Console** で以下の操作を実行します。

- **Authentication** → **CIBA Policy** タブを開きます。

- 項目を設定し、**Save** をクリックします。

設定可能な項目とその説明を以下に示します。

設定	説明
Backchannel Token Delivery Mode	CD(Consumption Device) が認証結果および関連するトークンを取得する方法を指定します。poll、ping、および push の3つのモードがあります。Red Hat build of Keycloak は "poll" のみサポートします。デフォルトの設定は poll です。この設定は必須です。詳細は、 CIBA 仕様 を参照してください。
Expires In	認証リクエストが受信されてからの auth_req_id の有効期限 (秒単位)。デフォルト設定は 120 です。この設定は必須です。詳細は、 CIBA 仕様 を参照してください。
Interval	CD (Consumption Device) がトークンエンドポイントへのポーリングリクエストを待機する必要がある間隔 (秒単位)。デフォルト設定は 5 です。この設定はオプションです。詳細は、 CIBA 仕様 を参照してください。
Authentication Requested User Hint	認証が要求されているエンドユーザーを識別する方法。デフォルト設定は login_hint です。login_hint、login_hint_token、および id_token_hint の3つのモードがあります。Red Hat build of Keycloak は "login_hint" のみサポートします。この設定は必須です。詳細は、 CIBA 仕様 を参照してください。

10.1.2.3.2. プロバイダー設定

CIBA 付与は以下2つのプロバイダーを使用します。

1. Authentication Channel Provider: Red Hat build of Keycloak と、AD (認証デバイス) 経由でユーザーを実際に認証するエンティティ間の通信を提供します。
2. User Resolver Provider: クライアントが提供する情報から Red Hat build of Keycloak の **UserModel** を取得し、ユーザーを特定します。

Red Hat build of Keycloak には、両方のデフォルトプロバイダーがあります。ただし、管理者は以下のように Authentication Channel Provider を設定する必要があります。

```
kc.[sh|bat] start --spi-ciba-auth-channel-ciba-http-auth-channel-http-authentication-channel-uri=https://backend.internal.example.com
```

設定可能な項目とその説明を以下に示します。

設定	説明
http-authentication-channel-uri	AD(認証デバイス)でユーザーを実際に認証するエンティティの URI を指定します。

10.1.2.3.3. Authentication Channel Provider

CIBA 標準ドキュメントでは、AD によるユーザーの認証方法は指定されていません。したがって、製品の判断で実装される可能性があります。Red Hat build of Keycloak は、この認証を外部認証エンティティに委譲します。Red Hat build of Keycloak は、認証エンティティと通信するために Authentication Channel Provider を提供しています。

Red Hat build of Keycloak の実装では、認証エンティティが Red Hat build of Keycloak 管理者の管理下にあることを前提としているため、Red Hat build of Keycloak は認証エンティティを信頼します。Red Hat build of Keycloak の管理者が制御できない認証エンティティを使用することは推奨されません。

Authentication Channel Provider は SPI プロバイダーとして提供され、Red Hat build of Keycloak のユーザーは環境を満たすために独自のプロバイダーを実装できます。Red Hat build of Keycloak は、HTTP を使用して認証エンティティと通信する HTTP Authentication Channel Provider と呼ばれるデフォルトプロバイダーを提供します。

Red Hat build of Keycloak ユーザーのユーザーが HTTP Authentication Channel Provider を使用する場合は、以下の2つの部分で設定される Red Hat build of Keycloak と認証エンティティの契約を把握している必要があります。

認証委譲リクエスト/レスポンス

Red Hat build of Keycloak は、認証リクエストを認証エンティティに送信します。

認証結果通知/ACK

認証エンティティは、認証結果を Red Hat build of Keycloak に通知します。

認証委譲リクエスト/レスポンスは、以下のメッセージングで設定されます。

認証委譲リクエスト

リクエストは、AD によるユーザー認証を要求するために、Red Hat build of Keycloak から認証エンティティに送信されます。

POST [delegation_reception]

- ヘッダー

名前	値	説明
Content-Type	application/json	メッセージのボディは json 形式です。

名前	値	説明
承認	Bearer [token]	[token] は、認証エンティティーが認証結果を Red Hat build of Keycloak に通知する場合に使用されます。

- パラメーター

型	名前	説明
パス	delegation_reception	委譲リクエストを受信するために認証エンティティーによって提供されるエンドポイント

- Body

名前	説明
login_hint	だれが AD で認証されるかを認証エンティティーに指示します。 デフォルトでは、ユーザーのユーザー名です。 このフィールドは必須で、CIBA 標準ドキュメントで定義されています。
scope	これは、認証されたユーザーから認証エンティティーが合意を取得するスコープを示します。 このフィールドは必須で、CIBA 標準ドキュメントで定義されています。
is_consent_required	これは、認証エンティティーがスコープについて認証済みユーザーから合意を取得する必要があるかどうかを示します。 このフィールドは必須です。
binding_message	この値は、CD と AD 両方の UI に表示され、ユーザーが AD による認証が CD によってトリガーされることを認識できるようにします。 このフィールドはオプションで、CIBA 標準ドキュメントで定義されています。
acr_values	これは、CD からの要求元の Authentication Context Class Reference を指示します。 このフィールドはオプションで、CIBA 標準ドキュメントで定義されています。

認証委譲レスポンス

認証エンティティーが Red Hat build of Keycloak から認証リクエストを受け取ったことを通知するために、認証エンティティーから Red Hat build of Keycloak に応答が返されます。

- 応答

HTTP ステータスコード	説明
201	Red Hat build of Keycloak に認証移譲リクエストの受信を通知します。

認証結果通知/ACK は以下のメッセージングで設定されます。

認証結果通知

認証エンティティは、認証リクエストの結果を Red Hat build of Keycloak に送信します。

POST /realms/[realm]/protocol/openid-connect/ext/ciba/auth/callback

- ヘッダー

名前	値	説明
Content-Type	application/json	メッセージのボディは json 形式です。
承認	Bearer [token]	[token] は、認証エンティティが認証移譲リクエストで Red Hat build of Keycloak から受け取ったトークンである必要があります。

- パラメーター

型	名前	説明
パス	realm	レルム名

- Body

名前	説明
status	これは、AD によるユーザー認証の結果を示します。以下のステータスのいずれかである必要があります。 SUCCEED: AD による認証が正常に完了しました。 UNAUTHORIZED: AD による認証が完了していません。 CANCELLED: AD による認証はユーザーによりキャンセルされています。

認証結果 ACK

Red Hat build of Keycloak から認証エンティティにレスポンスが返され、Red Hat build of Keycloak が認証エンティティから AD によるユーザー認証の結果を受け取ったことが通知されません。

- 応答

HTTP ステータスコード	説明
200	認証の結果の通知を受信したことを認証エンティティに通知します。

10.1.2.3.4. User Resolver Provider

同じユーザーであっても、その表現は CD、Red Hat build of Keycloak、認証エンティティごとに異なる場合があります。

CD、Red Hat build of Keycloak、認証エンティティが同じユーザーを認識するために、この User Resolver Provider は独自のユーザー表現を 3 者の間で変換します。

User Resolver Provider は SPI プロバイダーとして提供され、Red Hat build of Keycloak のユーザーは環境を満たすために独自のプロバイダーを実装できます。Red Hat build of Keycloak は、以下の特性を持つ Default User Resolver Provider と呼ばれるデフォルトプロバイダーを提供します。

- **login_hint** パラメーターのみをサポートし、デフォルトとして使用されます。
- Red Hat build of Keycloak の UserModel の **username** は、CD、Red Hat build of Keycloak、認証エンティティ上のユーザーを表すために使用されます。

10.1.3. OIDC ログアウト

OIDC には、ログアウトメカニズムに関連する 4 つの仕様があります。

1. [セッション管理](#)
2. [RP-Initiated Logout](#)
3. [フロントチャンネルログアウト](#)
4. [バックチャンネルログアウト](#)

繰り返しになりますが、これらはすべて OIDC 仕様に記載されていますが、ここでは概要のみを説明します。

10.1.3.1. セッション管理

これはブラウザーベースのログアウトです。アプリケーションは、Red Hat build of Keycloak から定期的にセッションステータス情報を取得します。Red Hat build of Keycloak でセッションが終了すると、アプリケーションはそれを認識して自身のログアウトをトリガーします。

10.1.3.2. RP-Initiated Logout

これもブラウザーベースのログアウトで、Red Hat build of Keycloak でユーザーを特定のエンドポイントにリダイレクトすることでログアウトが開始されます。通常、Red Hat build of Keycloak を使用して

ユーザー認証を行っていたアプリケーションのページで、ユーザーが **Log Out** リンクをクリックすると、このリダイレクトが発生します。

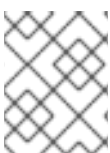
ユーザーがログアウトエンドポイントにリダイレクトされると、Red Hat build of Keycloak はクライアントにログアウト要求を送信してローカルユーザーセッションを無効にし、ログアウトプロセスが完了するとユーザーをいずれかの URL にリダイレクトする可能性があります。**id_token_hint** パラメーターが使用されない場合に、ユーザーはオプションでログアウトを確認するように要求される場合があります。ログアウト後、指定された **post_logout_redirect_uri** がパラメーターとして提供されている限り、ユーザーは自動的にリダイレクトされます。**post_logout_redirect_uri** が含まれている場合には、**client_id** または **id_token_hint** パラメーターのいずれかを含める必要があります。また、**post_logout_redirect_uri** パラメーターは、クライアント設定で指定された **Valid Post Logout Redirect URI** のいずれかと一致する必要があります。

クライアントの設定に応じて、ログアウト要求はフロントチャンネルまたはバックチャンネルを介してクライアントに送信できます。前のセクションで説明したセッション管理に依存するフロントエンドブラウザクライアントの場合、Red Hat build of Keycloak はログアウト要求をクライアントに送信する必要はありません。これらのクライアントは、ブラウザの SSO セッションがログアウトしていることを自動的に検出します。

10.1.3.3. フロントチャンネルログアウト

フロントチャンネルを介してログアウト要求を受信するようにクライアントを設定するには、[Front-Channel Logout](#) クライアント設定を確認します。この方法を使用するときは、次のことを考慮してください。

- Red Hat build of Keycloak によってクライアントに送信されるログアウト要求は、ブラウザと、ログアウトページ用にレンダリングされるビルトイン **iframe** に依存します。
- **iframe** に基づいているため、フロントチャンネルのログアウトはコンテンツセキュリティポリシー (CSP) の影響を受け、ログアウト要求がブロックされる可能性があります。
- ログアウトページを表示する前、またはログアウト要求が実際にクライアントに送信される前にユーザーがブラウザを閉じた場合、クライアントでのセッションが無効にならない可能性があります。



注記

Back-Channel Logout の使用を検討してください。これは、ユーザーがログアウトしてクライアントでセッションを終了するための、信頼性が高く安全な方法を提供します。

クライアントでフロントチャンネルログアウトが有効になっていない場合、Red Hat build of Keycloak はまず [バックチャンネルログアウト URL](#) を使用して、バックチャンネル経由でログアウト要求を送信しようとします。定義されていない場合、サーバーは [管理 URL](#) を使用するようにフォールバックします。

10.1.3.4. バックチャンネルログアウト

これは、Red Hat build of Keycloak とクライアント間の直接バックチャンネル通信を使用する非ブラウザベースのログアウトです。Red Hat build of Keycloak は、ログアウトトークンが含まれる HTTP POST リクエストを、Keycloak にログインしたすべてのクライアントに送信します。これらのリクエストは、Red Hat build of Keycloak で登録されたバックチャンネルログアウト URL に送信され、クライアント側でのログアウトをトリガーすると想定されています。

10.1.4. Red Hat build of Keycloak サーバーの OIDC URI エンドポイント

以下は、Red Hat build of Keycloak が発行する OIDC エンドポイントのリストです。Red Hat build of

Keycloak 以外のクライアントアダプターが OIDC を使用して認証サーバーと通信する場合に、これらのエンドポイントを使用できます。これらはすべて相対 URL です。URL のルートは、HTTP (S) プロトコル、ホスト名、およびオプションでパスで設定されます。たとえば、

```
https://localhost:8080
```

`/realms/{realm-name}/protocol/openid-connect/auth`

Authorization Code Flow で一時的なコードを取得する場合、または Implicit Flow、Direct Grants、または Client Grants を使用してトークンを取得する際に使用されます。

`/realms/{realm-name}/protocol/openid-connect/token`

一時コードをトークンに変換するために認可コードフローによって使用されます。

`/realms/{realm-name}/protocol/openid-connect/logout`

ログアウトの実行に使用されます。

`/realms/{realm-name}/protocol/openid-connect/userinfo`

OIDC 仕様で説明されている User Info サービスに使用されます。

`/realms/{realm-name}/protocol/openid-connect/revoke`

[RFC7009](#) で説明されているように OAuth 2.0 Token Revocation に使用されます。

`/realms/{realm-name}/protocol/openid-connect/certs`

JSON Web Token (jwks_uri) の検証に使用される公開鍵が含まれる JSON Web Key Set (JWKS) に使用されます。

`/realms/{realm-name}/protocol/openid-connect/auth/device`

デバイスコードおよびユーザーコードを取得するために Device Authorization Grant に使用されません。

`/realms/{realm-name}/protocol/openid-connect/ext/ciba/auth`

これは、クライアントによって行われた認証要求を識別する auth_req_id を取得するためのクライアント主導のバックチャネル認証許可の URL エンドポイントです。

`/realms/{realm-name}/protocol/openid-connect/logout/backchannel-logout`

これは、OIDC 仕様で説明されているバックチャネルログアウトを実行するための URL エンドポイントです。

これらすべてで、`{realm-name}` をレルムの名前に置き換えます。

10.2. SAML

[SAML 2.0](#) は OIDC と類似の仕様ですが、より成熟したものです。これは、SOAP および Web サービスメッセージングの仕様から派生するため、通常は OIDC よりも詳細になります。SAML 2.0 は、認証サーバーとアプリケーション間の XML ドキュメントを変換する認証プロトコルです。XML 署名と暗号化は、要求と応答の検証に使用されます。

通常、SAML は 2 つのユースケースを実装します。

最初のケースは、Red Hat build of Keycloak サーバーがユーザーを認証するように要求するアプリケーションです。ログインに成功すると、アプリケーションは XML ドキュメントを受け取ります。本書には、ユーザー属性を指定する SAML アサーションが含まれています。レルムは、アクセス情報 (ユーザーロールマッピングなど) が含まれるドキュメントにデジタル署名します。アプリケーションは、この情報を使用してユーザーがアプリケーションでアクセスできるリソースを判断します。

2つ目のユースケースは、リモートサービスにアクセスするクライアントです。クライアントは Red Hat build of Keycloak から SAML アサーションを要求し、ユーザーの代わりにリモートサービスを呼び出します。

10.2.1. SAML バインディング

Red Hat build of Keycloak は、3つのバインディングタイプをサポートします。

10.2.1.1. リダイレクトバインディング

リダイレクト バインディングは一連のブラウザーリダイレクト URI を使用して情報を交換します。

1. ユーザーは、ブラウザーを使用してアプリケーションに接続します。アプリケーションは、ユーザーが認証されていないことを検出します。
2. アプリケーションは XML 認証リクエストドキュメントを生成し、これを URI のクエリーパラメーターとしてエンコードします。この URI は、Red Hat build of Keycloak サーバーにリダイレクトするために使用されます。設定によっては、アプリケーションは XML ドキュメントにデジタル署名し、署名を Red Hat build of Keycloak へのリダイレクト URI のクエリーパラメーターとして追加することもできます。この署名は、リクエストを送信するクライアントを検証するために使用されます。
3. ブラウザーは Red Hat build of Keycloak にリダイレクトします。
4. サーバーは XML 認証リクエストドキュメントを抽出し、必要に応じてデジタル署名を検証します。
5. ユーザーは認証情報を入力します。
6. 認証後、サーバーは XML 認証応答ドキュメントを生成します。ドキュメントには、名前、アドレス、電子メール、およびユーザーが持つロールマッピングなどのユーザーに関するメタデータを保持する SAML アサーションが含まれます。ドキュメントは通常、XML 署名を使用してデジタル署名され、暗号化もされる場合があります。
7. XML 認証応答ドキュメントは、リダイレクト URI のクエリーパラメーターとしてエンコードされます。URI により、ブラウザーがアプリケーションに返されます。デジタル署名も、クエリーパラメーターとして含まれます。
8. アプリケーションはリダイレクト URI を受け取り、XML ドキュメントを抽出します。
9. アプリケーションはレルムの署名を検証し、有効な認証応答を受信していることを確認します。SAML アサーション内の情報は、アクセスの決定やユーザーデータの表示に使用されません。

10.2.1.2. POST バインディング

POST バインディングは **リダイレクト バインディング**と似ていますが、**POST** バインディングは GET リクエストの代わりに POST リクエストを使用して XML ドキュメントを交換します。**POST** バインディングは JavaScript を使用して、ドキュメント交換時にブラウザーが Red Hat build of Keycloak サーバーまたはアプリケーションに POSTS リクエストを送信するようにします。HTTP は、埋め込み JavaScript などの HTML フォームが含まれる HTML ドキュメントで応答します。ページを読み込むと、JavaScript はフォームを自動的に呼び出します。

POST バインディングは2つの制限があるために推奨されます。

- **セキュリティ:** Redirect バインディングでは、SAML 応答は URL の一部です。ログで応答をキャプチャーする可能性があるため、安全性は低くなります。
- **サイズ:** HTTP ペイロードでドキュメントを送信すると、制限された URL よりも、大量のデータに、より多くのスコープが提供されます。

10.2.1.3. ECP

Enhanced Client or Proxy (ECP) は、SAML v.2.0 プロファイルを表します。これにより、Web ブラウザーのコンテキスト外にある SAML 属性を交換できます。多くの場合、REST または SOAP ベースのクライアントによって使用されます。

10.2.2. Red Hat build of Keycloak サーバーの SAML URI エンドポイント

Red Hat build of Keycloak には、すべての SAML 要求に対して単一のエンドポイントがあります。

`http(s)://authserver.host/realm/{realm-name}/protocol/saml`

すべてのバインディングはこのエンドポイントを使用します。

10.3. OPENID CONNECT と SAML の比較

以下は、プロトコルの選択時に考慮する必要のある複数の要素のリストです。

ほとんどの目的において、Red Hat build of Keycloak では OIDC の使用が推奨されます。

OIDC

- OIDC は、特に Web と連携するように設計されています。
- OIDC は、SAML よりもクライアント側に簡単に実装できるため、HTML5/JavaScript アプリケーションにも適しています。
- OIDC トークンは JSON 形式で、Javascript を簡単に消費できます。
- OIDC には、セキュリティ実装を容易にするための機能があります。たとえば、ユーザーのログイン状態を決定するために仕様が使用する `iframe` のヒントを参照してください。

SAML

- SAML は、Web 上で機能するレイヤーとして設計されています。
- SAML は OIDC よりも詳細に指定できます。
- 成熟していると考えられているので、ユーザーは OIDC ではなく SAML を選択します。
- ユーザーは、セキュアな OIDC で SAML を選択します。

10.4. DOCKER REGISTRY V2 認証



注記

Docker 認証はデフォルトで無効になっています。Docker 認証を有効にするには、[機能の有効化と無効化](#)を参照してください。

Docker レジストリー V2 認証 は OIDC と同様に、Docker レジストリーに対してユーザーを認証します。このプロトコルの Red Hat build of Keycloak 実装により、Docker クライアントは Red Hat build of Keycloak 認証サーバーを使用してレジストリーに対して認証を行うことができます。このプロトコルは標準のトークンと署名メカニズムを使用しますが、実際の OIDC 実装とは異なります。要求と応答に非常に特殊な JSON 形式を使用し、リポジトリ名とパーミッションを OAuth スコープメカニズムにマッピングすることで、非常に特殊な JSON 形式を使用します。

10.4.1. Docker 認証フロー

認証フローについては、[Docker API のドキュメント](#) で説明されています。以下は、Red Hat build of Keycloak 認証サーバーの観点から見た概要です。

- **docker login** を実行します。
- Docker クライアントは Docker レジストリーからリソースを要求します。リソースが保護されていて、要求に認証トークンがない場合には、Docker レジストリーサーバーは、必要なパーミッションに関する情報と認可サーバーの場所を示す 401 HTTP メッセージを返します。
- Docker クライアントは、Docker レジストリーから 401 HTTP メッセージに基づいて認証要求を作成します。クライアントは、Red Hat build of Keycloak 認証サーバーへの [HTTP Basic 認証](#) 要求の一部として、(**docker login** コマンドからの) ローカルにキャッシュされた認証情報を使用します。
- Red Hat build of Keycloak 認証サーバーは、ユーザーの認証を試行し、OAuth スタイルの Bearer トークンが含まれる JSON ボディを返します。
- Docker クライアントは JSON 応答から Bearer トークンを受け取り、これを Authorization ヘッダーで使用し、保護されているリソースを要求します。
- Docker レジストリーは、トークンを使用して保護されたリソースに対する新しい要求を、Red Hat build of Keycloak サーバーから受信します。レジストリーはトークンを検証し、要求されたリソースへのアクセスを付与します (該当する場合)。



注記

Red Hat build of Keycloak では、Docker プロトコルで認証に成功した後に、ブラウザーの SSO セッションは作成されません。ブラウザーの SSO セッションでは、トークンを更新したり、Red Hat build of Keycloak サーバーからトークンまたはセッションのステータスを取得したりできないため、Docker プロトコルは使用されません。したがって、ブラウザー SSO セッションは必要ありません。詳細については、[一時的なセッションセッション](#) を参照してください。

10.4.2. Red Hat build of Keycloak の Docker Registry v2 認証サーバー URI エンドポイント

Red Hat build of Keycloak には、すべての Docker auth v2 リクエストに対して1つのエンドポイントがあります。

`http(s)://authserver.host/realms/{realm-name}/protocol/docker-v2`

第11章 管理コンソールへのアクセス制御

Red Hat build of Keycloak で作成された各レルムには、そのレルムを管理できる専用の管理コンソールがあります。**master** レルムは、管理者がシステムで複数のレルムを管理できるようにする特別なレルムです。本章では、このシナリオをすべて実施します。

11.1. マスターレルムアクセス制御

Red Hat build of Keycloak の **master** レルムは特別なレルムで、他のレルムとは異なる方法で処理されます。Red Hat build of Keycloak **master** レルム内のユーザーには、Red Hat build of Keycloak サーバーにデプロイされた 0 個以上のレルムを管理する権限を付与できます。レルムが作成されると、Red Hat build of Keycloak は、その新しいレルムにアクセスするための詳細な権限を付与するさまざまなロールを自動的に作成します。管理コンソールおよび管理 REST エンドポイントへのアクセスは、これらのロールを **master** レルムのユーザーにマッピングすることで制御できます。特定のレルムのみを管理できるユーザーは、複数のスーパーユーザーを作成することができます。

11.1.1. グローバルロール

master レルムには、2つのレルムレベルのロールがあります。以下のとおりです。

- admin
- create-realm

admin ロールを持つユーザーはスーパーユーザーであり、サーバー上のレルムを管理するためにフルアクセスできます。**create-realm** ロールを持つユーザーは、新しいレルムを作成できます。このユーザーは、作成する新しいレルムに完全アクセスできます。

11.1.2. レルム固有のロール

master レルム内の管理者ユーザーには、システムの他のレルムに対して管理権限を付与できます。Red Hat build of Keycloak の各レルムは、**master** レルムのクライアントによって表されます。クライアントの名前は **<realm name>-realm** です。これらのクライアントには、クライアントレベルのロールが定義されており、個別のレルムを管理するアクセスレベルを定義します。

利用可能なロールは以下のとおりです。

- view-realm
- view-users
- view-clients
- view-events
- manage-realm
- manage-users
- create-client
- manage-clients
- manage-events

- view-identity-providers
- manage-identity-providers
- impersonation

ユーザーに必要なロールを割り当てると、管理コンソールのその特定の部分のみを使用できます。



重要

manage-users ロールを持つユーザー管理者は、admin ロールを所有したユーザーにのみ割り当てることができます。したがって、admin に **manage-users** ロールがあり、**manage-realm** ロールがない場合、このロールを割り当てることができます。

11.2. 専用レルム管理コンソール

各レルムには、URL `/auth/admin/{realm-name}/console` に移動してアクセス可能な専用の管理コンソールがあります。特定のユーザーロールマッピングを割り当てることによって、そのレルム内のユーザーにレルム管理パーミッションを付与できます。

各レルムには、**realm-management** と呼ばれる組み込みクライアントがあります。このクライアントは、レルムの左側のメニュー項目 **Clients** に移動すると表示できます。このクライアントは、レルムの管理に付与できるパーミッションを指定するクライアントレベルのロールを定義します。

- view-realm
- view-users
- view-clients
- view-events
- manage-realm
- manage-users
- create-client
- manage-clients
- manage-events
- view-identity-providers
- manage-identity-providers
- impersonation

ユーザーに必要なロールを割り当てると、管理コンソールのその特定の部分のみを使用できます。

第12章 OPENID CONNECT および SAML クライアントの管理

クライアントは、ユーザーの認証を要求できるエンティティです。クライアントは2つの形式で提供されます。クライアントの最初のタイプは、single-sign-on に参加するアプリケーションです。これらのクライアントが Red Hat build of Keycloak に期待するのは、セキュリティーの提供のみです。他のタイプのクライアントは、認証されたユーザーの代わりに他のサービスを呼び出すことができるように、アクセストークンを要求するものです。このセクションでは、クライアントの設定に関するさまざまな側面と、その方法について説明します。

12.1. OPENID CONNECT クライアントの管理

[OpenID Connect](#) は、アプリケーションのセキュリティーを保護するのに推奨されるプロトコルです。Web でも簡単に使用できるように、ゼロから設計されているので、HTML5/JavaScript アプリケーションで最適に動作します。

12.1.1. OpenID Connect クライアントの作成

OpenID 接続プロトコルを使用するアプリケーションを保護するには、クライアントを作成します。

手順

1. メニューで **Clients** をクリックします。
2. **Create client** をクリックします

Create client

The screenshot shows the Keycloak administration interface for creating a new client. The left sidebar contains a navigation menu with options like Master, Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, and Authentication. The 'Clients' menu item is highlighted. The main content area is titled 'Create client' and includes a breadcrumb 'Clients > Create client'. Below the title, it says 'Clients are applications and services that can request authentication of a user.' The 'General Settings' tab is active, showing four input fields: 'Client type' (set to 'OpenID Connect'), 'Client ID' (empty), 'Name' (empty), and 'Description' (empty). At the bottom of the form are three buttons: 'Next' (blue), 'Back' (grey), and 'Cancel' (blue).

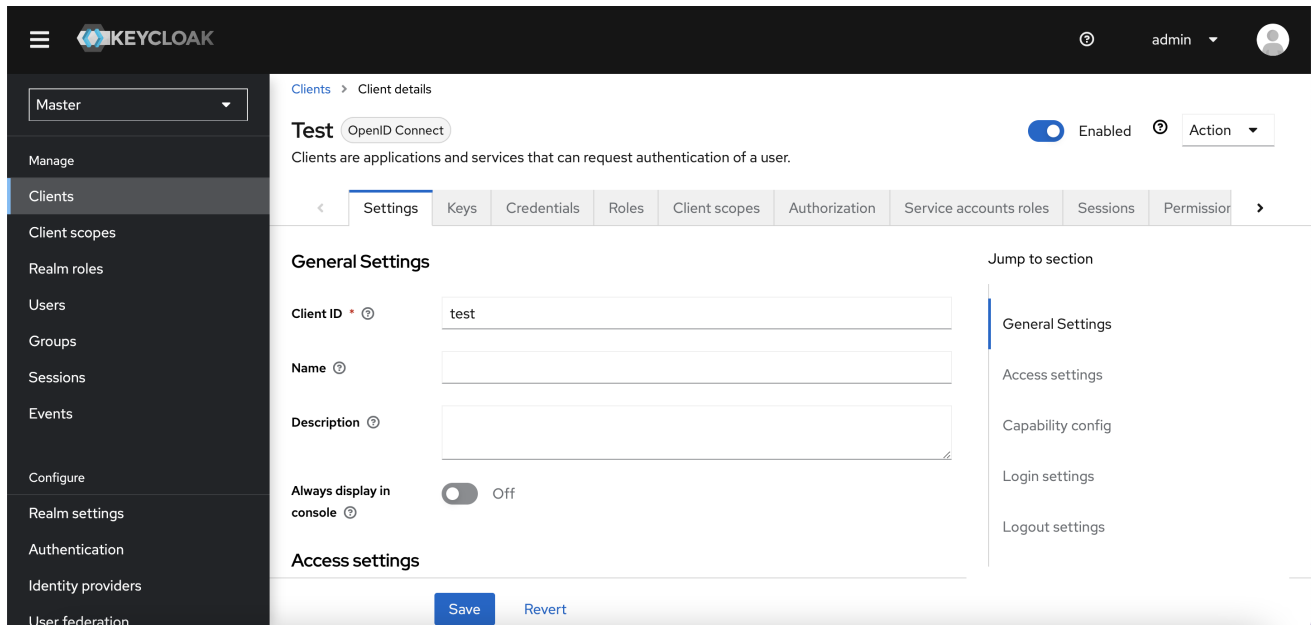
3. **Client type** を **OpenID Connect** に設定したままにします。
4. **クライアント ID** を入力します。
この ID は、OIDC リクエストおよび Red Hat build of Keycloak データベースでクライアントを識別するために使用される英数字の文字列です。
5. クライアントの **Name** を指定します。
この名前をローカライズする予定がある場合は、置換文字列値を設定します。たとえば、`${myapp}` などの文字列値です。詳細は、[サーバー開発者ガイド](#) を参照してください。
6. **Save** をクリックします。

この操作により、クライアントが作成され、[Basic configuration](#) を実行できる **Settings** タブが表示されます。

12.1.2. Basic configuration

Settings タブには、このクライアントを設定するための多くのオプションが含まれています。

Settings タブ



12.1.2.1. 一般設定

Client ID

OIDC リクエストおよび Red Hat build of Keycloak データベースでクライアントを識別するために使用される英数字の ID 文字列です。

名前

Red Hat build of Keycloak UI 画面に表示されるクライアントの名前。名前をローカライズするには、代替文字列値を設定します。たとえば、`${myapp}` などの文字列値です。詳細は、[サーバー開発者ガイド](#) を参照してください。

Description

クライアントの説明。この設定はローカライズすることもできます。

Always Display in Console

このユーザーがアクティブなセッションを持っていない場合でも、常にこのクライアントをアカウントコンソールにリストします。

12.1.2.2. Access Settings

Root URL

Red Hat build of Keycloak が設定済みの相対 URL を使用する場合、そのバック等はこの値が追加されます。

Home URL

認証サーバーがクライアントにリダイレクトまたはリンクバックする必要がある場合のデフォルト URL を提供します。

Valid Redirect URIs

必須フィールド。URL パターンを入力し、+ をクリックして既存の URL を追加し、-、Save の順にクリックします。有効なリダイレクト URI を比較するために、正確な (大文字と小文字を区別する) 文字列一致が使用されます。

URL パターンの最後にワイルドカードを使用できます。たとえば、`http://host.com/path/*` です。セキュリティ上の問題を回避するために、渡されたリダイレクト URI に `userinfo` 部分が含まれている場合、またはその `path` が親ディレクトリー (`/../`) へのアクセスを管理する場合、ワイルドカードの比較は実行されず、標準の安全な正確な文字列一致が実行されます。

完全なワイルドカード * 有効なリダイレクト URI を設定して、任意の `http` または `https` リダイレクト URI を許可することもできます。実稼働環境では使用しないでください。

通常、排他的リダイレクト URI パターンの方が安全です。詳細は [unspecific Redirect URIs](#) を参照してください。

Web Origins

URL パターンを入力し、+ をクリックして既存の URL を追加し、- をクリックして削除します。Save をクリックします。

このオプションは、[CORS\(Cross-Origin Resource Sharing\)](#) を処理します。ブラウザの JavaScript が、JavaScript コードの元のドメインとは異なるドメインを持つサーバーに対して AJAX HTTP リクエストを試行する場合、リクエストは CORS を使用する必要があります。サーバーは、CORS リクエストを処理する必要があります。処理しないと、ブラウザは表示されず、リクエストの処理を許可しません。このプロトコルは、XSS、CSRF、およびその他の JavaScript ベースの攻撃から保護します。

ここに記載のドメイン URL は、クライアントアプリケーションに送信されたアクセストークン内に埋め込まれています。クライアントアプリケーションはこの情報を使用して、CORS リクエストの呼び出しを許可するかどうかを決定します。Red Hat build of Keycloak クライアントアダプターのみがこの機能をサポートしています。詳細は、[アプリケーションおよびサービスガイド](#) を参照してください。

Admin URL

クライアントのコールバックエンドポイント。サーバーは、この URL を使用して失効ポリシーのプッシュ、バックチャネルログアウトの実行などのコールバックを行います。Red Hat build of Keycloak サブレットアダプターの場合、この URL をサブレットアプリケーションのルート URL にできます。詳細は、[アプリケーションおよびサービスガイド](#) を参照してください。

12.1.2.3. 機能設定

クライアント認証

OIDC クライアントのタイプ。

- **ON**
ブラウザログインを実行し、アクセストークン要求の実行時にクライアントシークレットを必要とするサーバー側のクライアントの場合。この設定はサーバー側のアプリケーションに使用する必要があります。
- **OFF**
ブラウザログインを実行するクライアント側のクライアントの場合。クライアント側のクライアントでシークレットを安全に保つことができないため、正しいリダイレクト URI を設定してアクセスを制限することが重要です。

Authorization

このクライアントに対する詳細な認可サポートを有効または無効にします。

Standard Flow

有効にすると、このクライアントは [OIDC Authorization Code Flow](#) を使用できます。

Direct Access Grants

有効にすると、このクライアントは [OIDC Direct Access Grants](#) を使用できます。

Implicit Flow

有効にすると、このクライアントは [OIDC Implicit Flow](#) を使用できます。

Service account roles

有効にすると、このクライアントは Red Hat build of Keycloak に対して認証を行い、このクライアント専用のアクセストークンを取得できます。OAuth2 仕様の観点から、これにより、このクライアントに対する **Client Credentials Grant** のサポートが有効になります。

Auth 2.0 Device Authorization Grant

有効にすると、このクライアントは [OIDC Device Authorization Grant](#) を使用できます。

OIDC CIBA Grant

有効にすると、このクライアントは [OIDC Client Initiated Backchannel Authentication Grant](#) を使用できます。

12.1.2.4. ログイン設定

Login theme

ログイン、OTP、許可登録、およびパスワードを忘れたページに使用するテーマ。

Consent required

有効にすると、ユーザーはクライアントアクセスに同意する必要があります。ブラウザログインを実行するクライアント側のクライアントの場合。クライアント側のクライアントでシークレットを安全に保つことができないため、正しいリダイレクト URI を設定してアクセスを制限することが重要です。

Display client on screen

このスイッチは、**Consent Required** が **Off** の場合に適用されます。

- **Off**
同意画面には、設定されたクライアントスコープに対応する同意のみが含まれます。
- **On**
同意画面には、このクライアント自体に関する項目も1つあります。

Client consent screen text

Consent required と **Display client on screen** が有効になっている場合に適用されます。このクライアントの権限に関する同意画面に表示されるテキストが含まれます。

12.1.2.5. ログアウト設定

Front channel logout

Front Channel Logout が有効になっている場合、アプリケーションは、[OpenID Connect フロントチャンネルログアウト](#) 仕様に従って、フロントチャンネルを介してユーザーをログアウトできる必要があります。有効になっている場合は、**フロントチャンネルログアウト URL** も指定する必要があります。

Front-channel logout URL

フロントチャネルを介してクライアントにログアウト要求を送信するために Red Hat build of Keycloak が使用する URL。

Backchannel logout URL

ログアウト要求がこのレムに (`end_session_endpoint` 経由で) 送信されたときにクライアントが自分自身をログアウトさせる URL。省略した場合、ログアウト要求はクライアントに送信されません。

Backchannel logout session required

Backchannel Logout URL が使用されている場合に、ログアウトトークンにセッション ID クレームを含めるかどうかを指定します。

Backchannel logout revoke offline sessions

バックチャネルログアウト URL が使用される場合に、`revoke_offline_access` イベントをログアウトトークンに含めるかどうかを指定します。Red Hat build of Keycloak は、このイベントでログアウトトークンを受信すると、オフラインセッションを取り消します。

12.1.3. 詳細設定

Settings タブのフィールドに入力したら、他のタブを使用して高度な設定を実行できます。

12.1.3.1. Advanced タブ

Advanced タブをクリックすると、追加のフィールドが表示されます。特定のフィールドの詳細は、そのフィールドの疑問符アイコンをクリックしてください。ただし、特定のフィールドは、このセクションで詳しく説明します。

12.1.3.2. Fine grain OpenID Connect configuration

Logo URL

クライアントアプリケーションのロゴを参照する URL。

Policy URL

プロファイルデータがどのように使用されるかについて読むために、証明書利用者クライアントがエンドユーザーに提供する URL。

Terms of Service URL

依拠当事者の利用規約について読むために、依拠当事者クライアントがエンドユーザーに提供する URL。

Signed and Encrypted ID Token Support

Red Hat build of Keycloak は、[Json Web Encryption \(JWE\)](#) 仕様に従って ID トークンを暗号化できます。管理者は、各クライアントに対して ID トークンが暗号化されているかどうかを判断します。

ID トークンの暗号化に使用されるキーは、コンテンツ暗号化キー (CEK) です。Red Hat build of Keycloak およびクライアントは、使用する CEK とその配信方法をネゴシエートする必要があります。CEK の決定に使用されるメソッドはキー管理モードです。Red Hat build of Keycloak がサポートするキー管理モードは、キー暗号化です。

キーの暗号化の場合:

1. クライアントは非対称暗号キーペアを生成します。

2. 公開鍵は CEK の暗号化に使用されます。
3. Red Hat build of Keycloak は ID トークンごとに CEK を生成します
4. Red Hat build of Keycloak は、この生成された CEK を使用して ID トークンを暗号化します
5. Red Hat build of Keycloak は、クライアントの公開鍵を使用して CEK を暗号化します。
6. クライアントは、秘密鍵を使用して暗号化された CEK を復号します。
7. クライアントは復号化された CEK を使用して ID トークンを復号化します。

クライアント以外のパーティーは ID トークンを復号化できます。

クライアントは、CEK を暗号化する公開鍵を Red Hat build of Keycloak に渡す必要があります。Red Hat build of Keycloak は、クライアントが提供する URL からの公開鍵のダウンロードをサポートします。クライアントは、[Json Web Keys\(JWK\)](#) 仕様に従って公開鍵を提供する必要があります。

手順は以下のとおりです。

1. クライアントの **Keys** タブを開きます。
2. **JWKS URL** を ON に切り替えます。
3. **JWKS URL** テキストボックスにクライアントの公開鍵 URL を入力します。

キー暗号化のアルゴリズムは、[Json Web Algorithm \(JWA\)](#) 仕様で定義されています。Red Hat build of Keycloak は以下をサポートします。

- RSAES-PKCS1-v1_5(RSA1_5)
- デフォルトパラメーターを使用した RSAES OAEP(RSA-OAEP)
- SHA-256 と MFG1(RSA-OAEP-256) を使用した RSAES OAEP 256

アルゴリズムを選択する手順は次のとおりです。

1. クライアントの **Advanced** タブを開きます。
2. **Fine Grain OpenID Connect Configuration** を開きます。
3. **ID Token Encryption Content Encryption Algorithm** プルダウンメニューからアルゴリズムを選択します。

12.1.3.3. Open ID Connect 互換モード

このセクションは下位互換性のために存在します。各フィールドの詳細については、疑問符アイコンをクリックしてください。

OAuth 2.0 Mutual TLS Certificate Bound Access Tokens Enabled

相互 TLS は、アクセストークンと更新トークンをクライアント証明書と共にバインドします。クライアント証明書は、TLS ハンドシェイク時に交換されます。このバインディングは、攻撃者が盗まれたトークンを使用するのを防ぎます。

このタイプのトークンは holder-of-key トークンです。Bearer トークンとは異なり、holder-of-key トークンの受信側は、トークンの送信側が正当であるかどうかを検証できます。

この設定が有効な場合に、ワークフローは以下のようになります。

1. トークンリクエストが、認可コードフローまたはハイブリッドフローのトークンエンドポイントに送信されます。
2. Red Hat build of Keycloak はクライアント証明書を要求します。
3. Red Hat build of Keycloak はクライアント証明書を受け取ります。
4. Red Hat build of Keycloak はクライアント証明書を正常に検証します。

検証が失敗した場合、Red Hat build of Keycloak はトークンを拒否します。

次の場合、Red Hat build of Keycloak は、アクセストークンまたは更新トークンを送信するクライアントを検証します。

- トークンの更新リクエストは、holder-of-key 更新トークンでトークンエンドポイントに送信されます。
- UserInfo リクエストは、holder-of-key アクセストークンで UserInfo エンドポイントに送信されます。
- ログアウトリクエストは、holder-of-key 更新トークンで Logout エンドポイントに送信されません。

詳細は、OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens の [Mutual TLS Client Certificate Bound Access Tokens](#) を参照してください。



注記

現在、Red Hat build of Keycloak クライアントアダプターは、キー所有者のトークン検証をサポートしていません。Red Hat build of Keycloak アダプターは、アクセストークンと更新トークンをベアラートークンとして扱います。

OIDC の詳細設定

OpenID Connect の詳細設定を使用すると、クライアントレベルで [セッションタイムアウトとトークンタイムアウト](#) のオーバーライドを設定できます。

Advanced Settings

This section is used to configure advanced settings of this client related to OpenID Connect protocol

Access Token Lifespan ?	Inherits from realm settings ▼	1	Minutes ▼
Client Session Idle ?	Inherits from realm settings ▼		Minutes ▼
Client Session Max ?	Inherits from realm settings ▼		Minutes ▼
Client Offline Session Idle ?	Inherits from realm settings ▼	30	Days ▼
Client Offline Session Max ?	Inherits from realm settings ▼	60	Days ▼

設定	説明
Access Token Lifespan	この値は、同じ名前のレルムオプションをオーバーライドします。
Client Session Idle	この値は、同じ名前のレルムオプションをオーバーライドします。この値は、グローバルの SSO Session Idle よりも小さくする必要があります。
Client Session Max	この値は、同じ名前のレルムオプションをオーバーライドします。この値は、グローバルの SSO Session Max よりも小さくする必要があります。
Client Offline Session Idle	この設定を使用すると、クライアントのオフラインセッションのアイドルタイムアウトを短く設定できます。タイムアウトは、Red Hat build of Keycloak がオフライントークンを取り消すまで、セッションがアイドル状態に留まる時間です。設定されていない場合、レルムの Offline Session Idle が使用されます。
Client Offline Session Max	この設定を使用すると、クライアントのオフラインセッションの最大ライフスパンを短く設定できます。ライフスパンは、Red Hat build of Keycloak が対応するオフライントークンを取り消すまでの最大時間です。このオプションを使用するには、レルム内でグローバルに Offline Session Max Limited が有効になっている必要があります。デフォルトは Offline Session Max です。

Proof Key for Code Exchange Code Challenge Method

攻撃者が正当なクライアントの認可コードを盗んだ場合には、PKCE (Proof Key for Code Exchange) により、コードに適用されるトークンを、攻撃者が受け取れないようにします。

管理者は、以下のオプションのいずれかを選択できます。

(blank)

Red Hat build of Keycloak は、クライアントが Offline Session Idle の認証エンドポイントに適切な PKCE パラメーターを送信しなければ、PKCE を適用しません。

S256

Red Hat build of Keycloak は、コードチャレンジメソッドが S256 であるクライアント PKCE に適用されます。

plain

Red Hat build of Keycloak は、コードチャレンジメソッドが plain のクライアント PKCE に適用されます。

詳細は、[OAuth パブリッククライアントによるコード Exchange の RFC 7636 Proof キー](#) を参照してください。

ACR から認証レベル (LoA) へのマッピング

クライアントの詳細設定では、どの **Authentication Context Class Reference (ACR)** 値をどの **Level of Authentication (LoA)** マップするかを定義できます。このマッピングは、[ACR から LoA へのマッピング](#) で説明されているように、レルムでも指定できます。ベストプラクティスは、このマッピングをレルムレベルで設定することです。これにより、複数のクライアント間で同じ設定を共有できます。

Default ACR Values を使用すると、**acr_values** パラメーターと、**acr** クレームがアタッチされた **claims** パラメーターがない状態で、ログイン要求がこのクライアントから Red Hat build of Keycloak に送信される場合のデフォルト値を指定できます。[公式の OIDC 動的クライアント登録仕様](#) を参照してください。



警告

デフォルトの ACR 値がデフォルトのレベルとして使用されますが、特定のレベルでのログインを強制するために確実に使用できるわけではないことに注意してください。たとえば、**Default ACR Values** をレベル 2 に設定するとします。次に、デフォルトで、ユーザーはレベル 2 で認証する必要があります。ただし、ユーザーが **acr_values=1** などのログイン要求にパラメーターを明示的にアタッチすると、レベル 1 が使用されます。その結果、クライアントが本当にレベル 2 を必要とする場合、クライアントは ID トークン内の **acr** クレームの存在を確認し、要求されたレベル 2 が含まれていることを再確認することが推奨されます。

ACR to LoA Mapping	Key	Value
?	<input type="text" value="Type a key"/>	<input type="text" value="Type a value"/> −
	+ Add an attribute	
Default ACR Values ?	<input type="text"/>	
	+ Add −	
	<input type="button" value="Save"/>	<input type="button" value="Revert"/>

詳細は、[Step-up Authentication](#) と [公式の OIDC 仕様](#) を参照してください。

12.1.4. 機密なクライアント認証情報

クライアントの [Client authentication](#) が **ON** に設定されている場合は、クライアントの認証情報を [Credentials](#) タブで設定する必要があります。

認証情報タブ

The screenshot displays the 'Client details' page for a client named 'myapp'. The 'Credentials' tab is active, showing the 'Client Authenticator' dropdown menu set to 'Client Id and Secret'. Below this is a 'Save' button. The 'Client secret' field is masked with dots and has a 'Regenerate' button. The 'Registration access token' field is also masked and has a 'Regenerate' button. The page is titled 'Clients > Client details' and shows the client is 'Enabled'.

Client Authenticator ドロップダウンリストは、クライアントに使用する認証情報のタイプを指定します。

クライアント ID およびシークレット

この選択はデフォルトの設定です。シークレットは自動的に生成されます。必要に応じて、**Regenerate** をクリックしてシークレットを再作成します。

署名付き JWT

The screenshot shows the 'Credentials' tab for the client 'myapp'. The 'Client Authenticator' is set to 'Signed Jwt' and the 'Signature algorithm' is set to 'Any algorithm'. There is a 'Save' button below these settings. Below that, there is a 'Registration access token' field with a 'Regenerate' button.

署名済み JWT は Signed Json Web Token です。

この認証情報タイプを選択すると、**Keys** タブでクライアントの秘密鍵と証明書も生成する必要があります。秘密鍵は JWT に署名するために使用されます。一方、証明書は、署名の検証にサーバーによって使用されます。

keys タブ

The screenshot shows the 'Keys' tab for the client 'myapp'. Under the 'JWKS URL configs' section, there is a text box with the following text: "If 'Use JWKS URL switch' is on, you need to fill a valid JWKS URL. After saving, admin can download keys from the JWKS URL or keys will be downloaded automatically by Keycloak server when see the stuff signed by the unknown KID". Below this text box is a toggle switch for 'Use JWKS URL' which is currently turned off. At the bottom of the page, there are three buttons: 'Save', 'Generate new keys', and 'Import'.

Generate new keys ボタンをクリックして、このプロセスを開始します。

キーの生成

Generate keys? ×

If you generate new keys, you can download the keystore with the private key automatically and save it on your client's side. Keycloak server will save just the certificate and public key, but not the private key.

Archive format ?

JKS ▼

Key alias * ?

myapp

Key password * ?



Store password * ?



Generate

Cancel

1. 使用するアーカイブ形式を選択します。
2. **鍵パスワード** を入力します。
3. **ストアパスワード** を入力します。
4. **Generate** をクリックします。

これらの鍵を生成する場合、Red Hat build of Keycloak は証明書を保存し、ユーザーはクライアントが使用する秘密鍵と証明書をダウンロードする必要があります。

外部ツールを使用して鍵を生成し、**Import Certificate** をクリックしてクライアントの証明書をインポートすることもできます。

証明書のインポート

Generate keys? ×

If you generate new keys, you can download the keystore with the private key automatically and save it on your client's side. Keycloak server will save just the certificate and public key, but not the private key.

Archive format ?

JKS ▼

Key alias * ?

Store password * ?

👁

Import file

Drag a file here or browse to upload

Browse...

Clear

Import

Cancel

1. 証明書のアーカイブ形式を選択します。
2. ストアパスワードを入力します。
3. **Import File** をクリックして証明書ファイルを選択します。
4. **Import** をクリックします。

JWKS URL を使用 をクリックすると、証明書をインポートする必要がなくなります。この場合、公開鍵が **JWK** 形式で公開される URL を指定できます。このオプションを使用すると、鍵が変更されると Red Hat build of Keycloak はキーを再インポートします。

Red Hat build of Keycloak アダプターで保護されたクライアントを使用している場合は、<https://myhost.com/myapp> がクライアントアプリケーションのルート URL であると仮定して、この形式で JWKS URL を設定できます。

```
https://myhost.com/myapp/k_jwks
```

詳細は、[サーバー開発者ガイド](#) を参照してください。

クライアントシークレットでの署名済み JWT

このオプションを選択する場合は、秘密鍵の代わりにクライアントシークレットで署名された JWT を使用できます。

このクライアントシークレットは、クライアントによって JWT に署名するために使用されます。

X509 証明書

Red Hat build of Keycloak は、TLS ハンドシェイク中にクライアントが適切な X509 証明書を使用しているか検証します。

X509 証明書

The screenshot shows the 'Credentials' configuration page for a client named 'myapp'. The 'Client Authenticator' is set to 'X509 Certificate'. The 'Allow regex pattern comparison' toggle is turned off. The 'Subject DN' field contains 'cn=localhost, ou=keycloak'. A 'Save' button is visible below the field. Below this, there is a 'Registration access token' field with a 'Regenerate' button.

バリデーターは、設定された正規表現検証式を使用して、証明書のサブジェクト DN フィールドも確認します。いくつかのユースケースでは、すべての証明書を受け入れるだけで十分です。この場合は、`(.*?)(?:$)` 式を使用できます。

Red Hat build of Keycloak では、次の 2 つの方法でリクエストからクライアント ID を取得できます。

- クエリー内の `client_id` パラメーター ([OAuth 2.0 仕様](#) のセクション 2.2 で説明されています)。
- `client_id` をフォームパラメーターとして指定します。

12.1.5. クライアントのシークレットローテーション



重要

クライアントシークレットローテーションのサポートは開発中であることに注意してください。この機能は実験的に使用してください。

[Confidential Client authentication](#) を使用するクライアントの場合、Red Hat build of Keycloak は、[クライアントポリシー](#) を通じてクライアントシークレットをローテーションする機能をサポートします。

クライアントシークレットローテーションポリシーは、シークレットの漏えいなどの問題を軽減するため、セキュリティが強化されます。有効にすると、Red Hat build of Keycloak ではクライアントごとに最大 2 つの同時アクティブシークレットがサポートされます。ポリシーは、以下の設定に従ってローテーションを管理します。

- **シークレットの有効期限:** [秒]: シークレットがローテーションされると、これは新しいシークレットの有効期限です。シークレット作成日に追加された量 (**秒単位**)。ポリシー実行時に計算されます。
- **ローテーションされたシークレットの有効期限:** [秒]: シークレットがローテーションされた場合、この値は古いシークレットの残りの有効期限です。この値は、常にシークレットの有効期限よりも小さくする必要があります。値が 0 の場合、クライアントのローテーション時に古いシークレットはすぐに削除されます。シークレットローテーションの日付に追加された量 (**秒単位**)。ポリシー実行時に計算されます。

- **更新中のローテーションの残りの有効期限:** [秒]: 動的クライアントへの更新でクライアントシークレットローテーションを実行する必要がある期間。ポリシー実行時に計算されます。

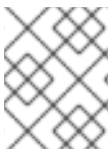
クライアントシークレットのローテーションが発生すると、新規のメインシークレットが生成され、古いクライアントシークレットが新しい有効期限を持つセカンダリーシークレットになります。

12.1.5.1. クライアントシークレットローテーションのルール

ローテーションは自動的に行われず、バックグラウンドプロセスを介して行われません。ローテーションを実行するには、Red Hat build of Keycloak 管理コンソールにおけるクライアントの認証情報タブで **Regenerate Secret** 機能を使用するか、Admin REST API を使用して、クライアント上で更新アクションを実行する必要があります。クライアント更新アクションを呼び出すと、シークレットのローテーションはルールに基づいて行われます。

- シークレットの有効期限の値が現在の日付未満の場合。
- 動的クライアント登録クライアント更新要求中に、**更新中のローテーションの残りの有効期限**の値が現在の日付と **シークレットの有効期限** の間の期間と一致する場合、クライアントシークレットは自動的にローテーションされます。

さらに、Admin REST API を介して、いつでもクライアントシークレットローテーションを強制することができます。



注記

新規クライアントの作成時に、クライアントシークレットのローテーションポリシーがアクティブになると、動作が自動的に適用されます。



警告

シークレットローテーション動作を既存のクライアントに適用するには、ポリシーを定義した後でそのクライアントを更新して、動作が適用されるようにします。

12.1.6. OIDC クライアントシークレットローテーションポリシーの作成

以下は、シークレットローテーションポリシーを定義する例です。

手順

1. メニューで **Realm Settings** をクリックします。
2. **Client Policies** タブをクリックします。
3. **Profiles** ページで、**Create client profile** をクリックします。

プロファイルの作成

4. **Name** に任意の名前を入力します。
5. **Description** のプロファイルの目的を特定するのに役立つ説明を入力します。
6. **Save** をクリックします。
このアクションによりプロファイルが作成され、エグゼキューターを設定できます。
7. **Add executor** をクリックして、このプロファイルにエグゼキューターを設定します。

プロファイルエグゼキューターを作成する

8. **Executor Type** に **secret-rotation** を選択します。
9. **Secret Expiration** の各シークレットの最大継続時間を秒単位で入力します。
10. **Rotated Secret Expiration** について、ローテーションされた各シークレットの最大継続時間を秒単位で入力します。



警告

Rotated Secret Expiration の値は、常に **Secret Expiration** よりも小さくする必要があるので注意してください。

11. 更新アクションがクライアントの **RemainExpirationTime** を更新するまでの時間を秒単位で入力します。

12. **Add** をクリックします。
上記の例では、以下のようになります。
 - 各シークレットは1週間で有効です。
 - ローテーションされたシークレットは2日後に有効期限が切れます。
 - 動的クライアントを更新するウィンドウは、シークレットの有効期限が切れる前に1日後に開始します。
13. **Client Policies** タブに戻ります。
14. **Policies** をクリックします。
15. **Create client policy** をクリックします。

クライアントシークレットローテーションポリシーの作成

Master

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Realm settings > Client policies > Create policy

Create policy Enabled

Name * Weekly client secret rotation policy

Description Enables secret rotation behavior for confidential clients

Save Cancel

16. **Name** に任意の名前を入力します。
17. **Description** のポリシーの目的を特定するのに役立つ説明を入力します。
18. **Save** をクリックします。
このアクションによりポリシーが作成され、ポリシーをプロファイルに関連付けることができます。また、ポリシー実行の条件を設定することもできます。
19. 条件で、**Add condition** をクリックします。

クライアントシークレットローテーションポリシー条件の作成

Master

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Realm settings > Client policies > Policy details > Add condition

Add condition

Condition type ② client-access-type

Negative Logic ② Off

Client Access Type ② confiden... x

Add Cancel

20. すべての機密クライアントに動作を適用するには、**Condition Type** フィールドで **client-access-type** を選択します。



注記

クライアントの特定のグループに適用するには、**Condition Type** フィールドに **client-roles** タイプを選択する方法もあります。これにより、特定のロールを作成し、カスタムのローテーション設定を各ロールに割り当てることができます。

21. **Client Access Type** フィールドに **confidential** を追加します。
22. **Add** をクリックします。
23. ポリシー設定に戻り、**Client Profiles** で、**Add client profile** をクリックし、リストから **Weekly Client Secret Rotation Profile** を選択して、**Add** をクリックします。

クライアントシークレットローテーションポリシー

The screenshot shows the configuration page for a 'Weekly client secret rotation policy'. The page is titled 'Weekly client secret rotation policy' and is currently 'Enabled'. The 'Name' field contains 'Weekly client secret rotation policy' and the 'Description' field contains 'Enables secret rotation behavior for confidential clients.' There are 'Save' and 'Revert' buttons. Below the description, there is a 'Conditions' section with one condition listed: 'client-access-type'. There is also a 'Client profiles' section with one profile listed: 'Weekly client secret rotation profile'.



注記

シークレットのローテーション動作を既存のクライアントに適用するには、以下の手順に従います。

管理コンソールの使用

1. メニューで **Clients** をクリックします。
2. クライアントをクリックします。
3. **Credentials** タブをクリックします。
4. クライアントシークレットの **Re-generate** をクリックします。

クライアント REST サービスを使用すると、以下のいずれかの方法で実行できます。

- クライアントでの更新操作経由
- 再生成クライアントシークレットエンドポイント経由

12.1.7. サービスアカウントの使用

各 OIDC クライアントには、ビルトインの **サービスアカウント** があります。この **サービスアカウント** を使用してアクセストークンを取得します。

手順

1. メニューで **Clients** をクリックします。
2. クライアントを選択します。
3. **Settings** タブをクリックします。
4. **Client authentication** を **On** に切り替えます。
5. **Service accounts roles** を選択します。
6. **Save** をクリックします。
7. **クライアント認証情報** を設定します。
8. **Scope** タブをクリックします。
9. ロールがあることを確認するか、**Full Scope Allowed** を **ON** に切り替えます。
10. **Service Account Roles** タブをクリックします。
11. クライアントのこのサービスアカウントで利用可能なロールを設定します。

アクセストークンからのロールは、以下の交差部分になります。

- クライアントのロールスコープと、リンクされたクライアントスコープから継承されたロールスコープのマッピング
- サービスアカウントロール

呼び出す REST URL は、**/realms/{realm-name}/protocol/openid-connect/token** です。この URL は POST 要求として呼び出され、要求でクライアントクレデンシャルを投稿する必要があります。

デフォルトでは、クライアント認証情報は **Authorization: Basic** ヘッダーでクライアントの `clientId` および `clientSecret` によって表されますが、署名付きの JWT アサーションやその他のクライアント認証用のカスタムメカニズムを使用してクライアントを認証することもできます。

OAuth2 仕様に従って、**grant_type** パラメーターを `client_credentials` に設定する必要があります。

たとえば、サービスアカウントを取得する POST 呼び出しは以下のようになります。

```
POST /realms/demo/protocol/openid-connect/token
Authorization: Basic cHJvZHVjdC1zYS1jbGllbnQ6cGFzc3dvcmQ=
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

応答は、OAuth 2.0 仕様からのこの **アクセストークン応答** と似ています。

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache
```

```
{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
  "token_type":"bearer",
  "expires_in":60
}
```

デフォルトでは、アクセストークンのみが返されます。認証が成功した場合、デフォルトでは更新トークンは返されず、Red Hat build of Keycloak 側にユーザーセッションは作成されません。更新トークンがないため、アクセストークンの期限が切れると再認証が必要になります。ただし、セッションはデフォルトでは作成されないため、この状況で Red Hat build of Keycloak サーバーにオーバーヘッドが追加されることはありません。

このような状況では、ログアウトは必要ありません。ただし、発行されたアクセストークンは、[OpenID Connect Endpoints](#) セクションで説明するように、OAuth2 Revocation Endpoint にリクエストを送信して取り消すことができます。

関連情報

詳細については、[クライアント認証情報の付与](#) を参照してください。

12.1.8. オーディエンスのサポート

通常、Red Hat build of Keycloak のデプロイ環境は、認証に Red Hat build of Keycloak を使用する **機密** または **公開** クライアントアプリケーションのセットで設定されます。

また、クライアントアプリケーションからの要求に対応し、これらのアプリケーションにリソースを提供する **サービス (OAuth 2 仕様 のリソースサーバー)** も利用できます。これらのサービスでは、要求の認証に **アクセストークン** (Bearer トークン) を送信する必要があります。このトークンは、Red Hat build of Keycloak へのログイン時にフロントエンドアプリケーションによって取得されます。

サービス間の信頼性が低い環境では、以下のシナリオが発生する可能性があります。

1. フロントエンドクライアントアプリケーションには、Red Hat build of Keycloak に対する認証が必要です。
2. Red Hat build of Keycloak はユーザーを認証します。
3. Red Hat build of Keycloak はアプリケーションにトークンを発行します。
4. アプリケーションはトークンを使用して信頼できないサービスを呼び出す。
5. 信頼できないサービスはアプリケーションへの応答を返す。ただし、アプリケーショントークンを保持します。
6. その後、信頼されていないサービスは、アプリケーショントークンを使用して信頼されるサービスを呼び出す。これにより、信頼できないサービスがトークンを使用してクライアントアプリケーションの代わりに他のサービスにアクセスするため、セキュリティが損なわれます。

このシナリオは、サービス間の信頼度が高い環境では発生する可能性が低いですが、信頼度が低い環境では発生する可能性があります。一部の環境では、信頼されていないサービスが信頼できるサービスからデータを取得して、元のクライアントアプリケーションにデータを返す必要があるため、このワークフローは正しい場合があります。

対象範囲に制限がないと、サービス間で高いレベルの信頼が存在する場合に役立ちます。そうでない場合は、対象範囲を制限する必要があります。対象範囲を制限しつつ、信頼できないサービスが信頼でき

るサービスからデータを取得できるようにしますこの場合は、信頼できないサービスと信頼できるサービスが対象としてトークンに追加されていることを確認します。

アクセストークンの誤用を防ぐには、トークンの対象範囲を制限し、トークンの対象を確認するようにサービスを設定します。フローは以下のように変わります。

1. フロントエンドアプリケーションは、Red Hat build of Keycloak に対して認証を行います。
2. Red Hat build of Keycloak はユーザーを認証します。
3. Red Hat build of Keycloak はアプリケーションにトークンを発行します。アプリケーションは、信頼されていないサービスを呼び出す必要があることを認識しているため、Red Hat build of Keycloak に送信される認証リクエストに `scope=<untrusted service>` を配置します (`scope` パラメーターの詳細については、[クライアントスコープ](#) セクションを参照してください)。アプリケーションに発行されたトークンには、対象範囲内の信頼できないサービスへの参照 (`"audience": ["<untrusted service>"]`) が含まれます。これで、クライアントがこのアクセストークンを使用して信頼できないサービスを呼び出すことを宣言します。
4. 信頼されないサービスは、トークンを使用して信頼できるサービスを呼び出します。信頼できるサービスがトークンのオーディエンスをチェックし、そのオーディエンスが信頼できないサービスのみを対象としていることを検出したため、呼び出しは成功しません。これは想定内の動作であり、セキュリティが破損していません。

クライアントが後で信頼されるサービスを呼び出す場合は、`scope=<trusted service>` で SSO ログインを再発行して別のトークンを取得する必要があります。返されるトークンには、信頼できるサービスが対象として含まれます。

```
"audience": [ "<trusted service>" ]
```

この値を使用して `<trusted service>` を起動します。

12.1.8.1. 設定

対象チェックを設定する場合は、以下を行います。

- アダプター設定に `verify-token-audience` フラグを追加して、サービスが送信されたアクセストークンのオーディエンスを確認するように設定されていることを確認します。詳細は [アダプターの設定](#) を参照してください。
- Red Hat build of Keycloak によって発行されたアクセストークンに、必要なすべてのオーディエンスが含まれていることを確認します。オーディエンスは、[次のセクション](#) で説明するように、クライアントロールを使用するか、ハードコーディングして追加できます。[ハードコーディングされたオーディエンス](#) を参照してください。

12.1.8.2. 対象の自動追加

Audience Resolve プロトコルマッパーは、デフォルトのクライアントスコープ `roles` で定義されます。マッパーは、現在のトークンで使用可能なクライアントロールが少なくとも1つあるクライアントをチェックします。その後、各クライアントのクライアント ID がオーディエンスとして追加されます。これは、サービスクライアントがクライアントロールに依存する場合に役立ちます。通常、サービスクライアントはフローが有効になっていないクライアントの可能性があり、それ自体に直接発行されたトークンを持たない場合があります。これは OAuth 2 [リソースサーバー](#) を表します。

たとえば、サービスクライアントと機密クライアントの場合、機密クライアントに対して発行されたアクセストークンを使用して、サービスクライアントの REST サービスを呼び出すことができます。以下が `true` の場合、サービスクライアントは機密クライアント向けに発行されたアクセストークンのオー

ディエンズとして自動的に追加されます。

- サービスクライアントには、それ自体に定義されたクライアントロールがあります。
- ターゲットユーザーには、少なくとも1つのクライアントロールが割り当てられている。
- 機密クライアントには割り当てられたロールのロールスコープマッピングがある。



注記

対象が自動的に追加されないようにする場合は、機密クライアントに直接ロールスコープマッピングを設定しないでください。代わりに、専用のクライアントスコープのクライアントロールにロールスコープマッピングが含まれる専用のクライアントスコープを作成できます。

クライアントスコープが任意のクライアントスコープとし機密クライアントに追加されると、`scope=<trusted service>` パラメーターで明示的に要求されている場合は、クライアントロールと対象がトークンに追加されます。



注記

フロントエンドクライアント自体はアクセストークンオーディエンズに自動的に追加されないため、アクセストークンには、トークンが対象として発行されるクライアントが含まれないので、アクセストークンと ID トークンを簡単に区別できます。

クライアント自体がオーディエンズとして必要な場合は、[ハードコーディングされたオーディエンズ](#) オプションを参照してください。ただし、フロントエンドと REST サービスと同じクライアントを使用することは推奨されていません。

12.1.8.3. ハードコードされたオーディエンズ

サービスがレルムロールに依存する場合や、トークンのロールに依存しない場合は、ハードコーディングされた対象範囲を使用すると便利です。ハードコーディングされた対象範囲は、指定されたサービスクライアントのクライアント ID を対象としてトークンに追加するプロトコルマッパーです。クライアント ID 以外の対象を使用する場合は、URL などのカスタム値を使用できます。

プロトコルマッパーを直接フロントエンドクライアントに追加できます。プロトコルマッパーが直接追加される場合、対象も常に追加されます。

プロトコルマッパーをより詳細に制御するには、`good-service` などと呼ばれる、専用のクライアントスコープでプロトコルマッパーを作成できます。

オーディエンズプロトコルマッパー

- `good-service` のクライアントの [Client details タブ](#) から、アダプター設定を生成し、`verify-token-audience` が `true` に設定されていることを確認できます。この設定を使用する場合、このアクションによりアダプターは対象ユーザーを検証するように強制されます。
- 機密クライアントがトークン内の対象として `good-service` を要求できます。機密クライアントで以下を行います。
 1. [Client Scopes](#) タブをクリックします。
 2. `good-service` をオプション (またはデフォルト) クライアント範囲として割り当てます。詳細は、[クライアントスコープのリンクセクション](#) を参照してください。
- 必要に応じて、[クライアントスコープを評価](#) し、サンプルアクセストークンを生成することができます。任意のクライアントスコープとして割り当てられた場合は、`good-service` が `scope` パラメーターに含まれている場合に、生成されるアクセストークンの対象者に `good-service` が追加されます。
- 機密クライアントアプリケーションで、`scope` パラメーターが使用されていることを確認します。`good-service` にアクセスするためのトークンを発行する場合は、`good-service` の値を含める必要があります。

参照:

- アプリケーションがサブリットアダプターを使用する場合は [パラメーター転送セクション](#)。
- アプリケーションが javascript アダプターを使用する場合は [JavaScript アダプターセクション](#)。



注記

Audience および **Audience Resolve** プロトコルマッパーの両方はデフォルトで、対象をアクセストークンに追加します。ID トークンには通常、OpenID Connect 仕様の要件である単一の対象のみ (トークンが発行されたクライアント ID) が含まれます。ただし、アクセストークンには、対象マッパーが追加されない限り、トークンが発行されたクライアント ID があるとは限りません。

12.2. SAML クライアントの作成

Red Hat build of Keycloak は、登録済みアプリケーションの [SAML 2.0](#) をサポートします。POST およびリダイレクトバインディングがサポートされます。クライアント署名の検証を要求することもできます。サーバー署名や暗号化応答も可能です。

手順

1. メニューで **Clients** をクリックします。
2. **Create client** をクリックして、**Create client** ページに移動します。
3. **Client type** を **SAML** に設定します。

Create client

The screenshot shows the 'Create client' page in Keycloak. The left sidebar has 'Clients' selected. The main content area has the title 'Create client' and a subtitle 'Clients are applications and services that can request authentication of a user.' Below this, there is a 'General Settings' tab. The form fields are: 'Client type' (SAML), 'Client ID' (mysamlapp), 'Name' (saml), and 'Description' (empty). At the bottom, there are 'Save', 'Back', and 'Cancel' buttons.

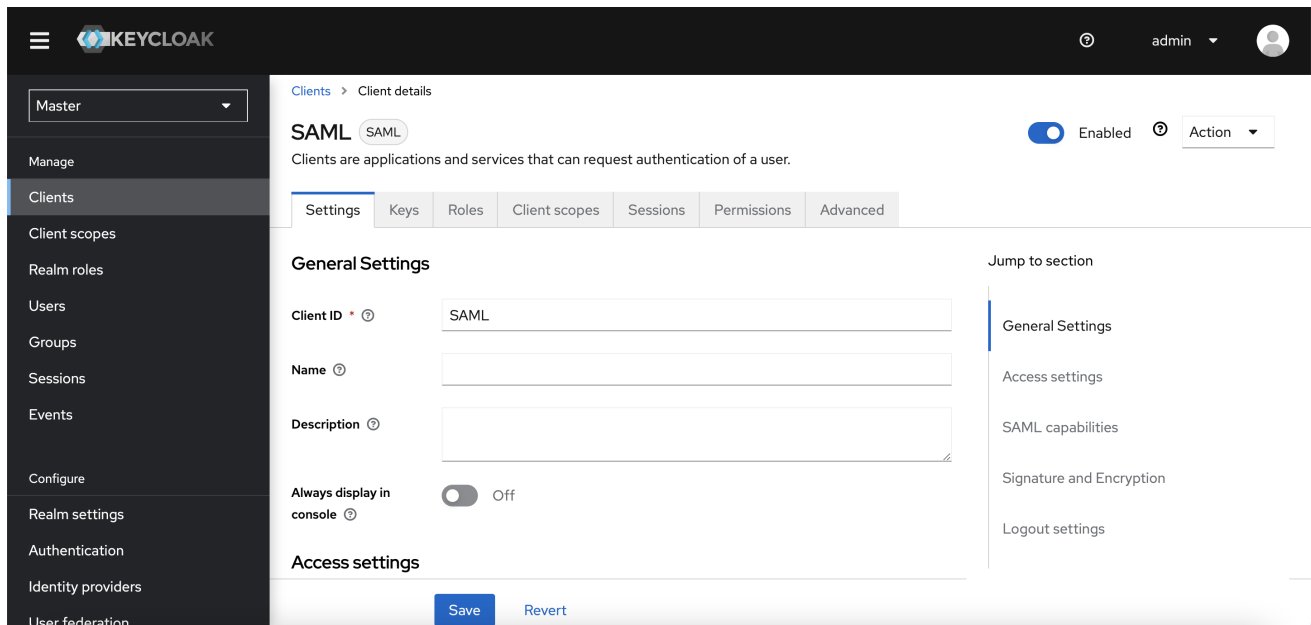
4. クライアントの **クライアント ID** を入力します。これは通常 URL であり、アプリケーションによって送信される SAML リクエストの **issuer** 値になります。
5. **Save** をクリックします。このアクションにより、クライアントが作成され、**Settings** タブが表示されます。

次のセクションでは、このタブの各設定について説明します。

12.2.1. Settings タブ

Settings タブには、このクライアントを設定するための多くのオプションが含まれています。

クライアントの設定



12.2.1.1. 一般設定

Client ID

OIDC リクエストおよび Red Hat build of Keycloak データベースでクライアントを識別するために使用される英数字の ID 文字列です。この値は、AuthNRequests で送信される発行者の値と一致する必要があります。Red Hat build of Keycloak は、Authn SAML 要求から発行者をプルし、この値によってクライアントと照合します。

名前

Red Hat build of Keycloak UI 画面に表示されるクライアントの名前。名前をローカライズするには、代替文字列値を設定します。たとえば、`{myapp}` などの文字列値です。詳細は、[サーバー開発者ガイド](#) を参照してください。

Description

クライアントの説明。この設定はローカライズすることもできます。

Always Display in Console

このユーザーがアクティブなセッションを持っていない場合でも、常にこのクライアントをアカウントコンソールにリストします。

12.2.1.2. Access Settings

Root URL

Red Hat build of Keycloak が設定された相対 URL を使用する場合、この値は URL の先頭に付加されます。

Home URL

Red Hat build of Keycloak をクライアントにリンクする必要がある場合は、この URL が使用されません。

Valid Redirect URIs

URL パターンに入力し、+ 記号をクリックして追加します。- 記号をクリックして削除します。Save をクリックして変更を保存します。ワイルドカードの値は URL の最後のみ使用できません。(例: `http://host.com/*$$`)。このフィールドは、正確な SAML エンドポイントが登録されておらず、Red Hat build of Keycloak がリクエストから Assertion Consumer URL をプルする場合に使用されます。

IDP-Initiated SSO URL name

IDP Initiated SSO を実行するときクライアントを参照する URL フラグメント名。これを空のままにすると、IDP Initiated SSO が無効になります。ブラウザから参照する URL は次のようになります: `server-root/realms/{realm}/protocol/saml/clients/{client-url-name}`

IDP Initiated SSO Relay State

IDP Initiated SSO を行いたい場合に、SAML リクエストで送信するリレー状態。

Master SAML Processing URL

この URL はすべての SAML 要求に使用され、応答は SP に転送されます。これは、Assertion Consumer Service URL および Single Logout Service URL として使用されます。

ログイン要求に Assertion Consumer Service URL が含まれる場合には、これらのログイン要求が優先されます。この URL は、登録された Valid Redirect URI パターンで妥当性を検証する必要があります。

12.2.1.3. SAML 機能

Name ID Format

サブジェクトの名前 ID 形式。この形式は、要求に名前 ID ポリシーが指定されていない場合や、Force Name ID Format 属性が ON に設定されている場合に使用されます。

Force Name ID Format

要求に名前 ID ポリシーがある場合は無視し、管理コンソールで設定された値を Name ID Format で使用されます。

Force POST Binding

デフォルトでは、Red Hat build of Keycloak は、元の要求の最初の SAML バインディングを使用して応答します。Force POST Binding を有効にすると、Red Hat build of Keycloak は元の要求がリダイレクトバインディングを使用した場合でも SAML POST バインディングを使用して応答します。

Force artifact binding

有効にすると、SAML ARTIFACT バインディングシステムを通じて応答メッセージがクライアントに返されます。

Include AuthnStatement

SAML ログイン応答は、使用される認証方法 (パスワードなど) と、ログインのタイムスタンプおよびセッションの有効期限を指定できます。Include AuthnStatement は、AuthnStatement 要素がログイン応答に含まれるように、デフォルトで有効になっています。これを OFF に設定すると、クライアントが最大セッションの長さを判別できなくなるので、期限切れにならないクライアントセッションが作成できます。

Include OneTimeUse Condition

有効にすると、ログイン応答に OneTimeUse 条件が含まれます。

Optimize REDIRECT signing key lookup

ON に設定すると、SAML プロトコルメッセージには Red Hat build of Keycloak ネイティブの拡張が含まれます。この拡張には、署名キー ID のヒントが含まれています。SP は、鍵を使用した署名の検証を試みる代わりに、署名検証の拡張を使用します。

このオプションは、署名がクエリーパラメーターで転送され、この情報は署名情報では見つからない REDIRECT バインディングに適用されます。これは、キー ID が常にドキュメント署名に含まれる POST バインディングメッセージとは対照的です。

このオプションは、Red Hat build of Keycloak サーバーとアダプターが IDP および SP を提供する場合に使用されます。このオプションは、Sign Documents がオンの場合にのみ関連します。

12.2.1.4. 署名と暗号化

サインインドキュメント

ON に設定すると、Red Hat build of Keycloak はレルム秘密鍵を使用してドキュメントに署名します。

アサーションへの署名

アサーションは署名され、SAML XML Auth 応答に組み込まれます。

Signature Algorithm

SAML ドキュメントの署名に使用されるアルゴリズム。**SHA1** ベースのアルゴリズムは非推奨となっており、将来のリリースでは削除される可能性があることに注意してください。*_**SHA1** の代わりに、より安全なアルゴリズムを使用することを推奨します。また、*_**SHA1** アルゴリズムでは、SAML クライアントが Java 17 以降で実行されている場合、署名の検証が機能しません。

SAML Signature Key Name

POST バインディングを使用して送信される署名付き SAML ドキュメントには、**KeyName** 要素に署名キーの ID が含まれています。このアクションは、**SAML 署名キー名** のオプションを使用して制御できます。このオプションは、**Keyname** の内容を制御します。

- **KEY_ID**: **KeyName** にはキー ID が含まれます。このオプションはデフォルトのオプションです。
- **CERT_SUBJECT**: **KeyName** には、レルムキーに対応する証明書のサブジェクトが含まれます。このオプションは、Microsoft Active Directory Federation Services で必要です。
- **NONE**: **KeyName** ヒントは、SAML メッセージから完全に省略されます。

正規化メソッド

XML 署名の正規化メソッド。

12.2.1.5. ログイン設定

Login theme

ログイン、OTP、許可登録、およびパスワードを忘れたページに使用するテーマ。

Consent required

有効にすると、ユーザーはクライアントアクセスに同意する必要があります。

ブラウザーログインを実行するクライアント側のクライアントの場合。クライアント側のクライアントでシークレットを安全に保つことができないため、正しいリダイレクト URI を設定してアクセスを制限することが重要です。

Display client on screen

このスイッチは、**Consent Required** が **Off** の場合に適用されます。

- **Off**
同意画面には、設定されたクライアントスコープに対応する同意のみが含まれます。
- **On**
同意画面には、このクライアント自体に関する項目も1つあります。

Client consent screen text

Consent required と **Display client on screen** が有効になっている場合に適用されます。このクライアントの権限に関する同意画面に表示されるテキストが含まれます。

12.2.1.6. ログアウト設定

Front channel logout

Front Channel Logout が有効になっている場合、アプリケーションのログアウトにはブラウザのリダイレクトが必要です。たとえば、アプリケーションでは Cookie をリセットする必要がありますが、リダイレクトでのみ実行可能です。**Front Channel Logout** が無効な場合、Keycloak はバックグラウンドの SAML 要求を呼び出して、アプリケーションからログアウトします。

12.2.2. keys タブ

Encrypt Assertions

SAML ドキュメントのアサーションをレルムの秘密鍵で暗号化します。AES アルゴリズムは、128 ビットのキーサイズを使用します。

Client Signature Required

Client Signature Required が有効な場合は、クライアントからのドキュメントは署名されている必要があります。Red Hat build of Keycloak は、**Keys** タブで設定されたクライアント公開鍵または証明書を使用してこの署名を検証します。

Allow ECP Flow

true の場合、このアプリケーションは認証に SAML ECP プロファイルを使用できます。

12.2.3. Advanced タブ

このタブには、特定の状況に対応する多くのフィールドがあります。一部のフィールドは、他のトピックで説明します。他のフィールドの詳細は、疑問符アイコンをクリックしてください。

12.2.3.1. Fine Grain SAML Endpoint Configuration

Logo URL

クライアントアプリケーションのロゴを参照する URL。

Policy URL

プロファイルデータがどのように使用されるかについて読むために、証明書利用者クライアントがエンドユーザーに提供する URL。

Terms of Service URL

依拠当事者の利用規約について読むために、依拠当事者クライアントがエンドユーザーに提供する URL。

Assertion Consumer Service POST Binding URL

Assertion Consumer Service の POST バインディング URL。

Assertion Consumer Service Redirect Binding URL

Assertion Consumer Service のリダイレクトバインディング URL。

Logout Service POST Binding URL

Logout Service の POST バインディング URL。

Logout Service Redirect Binding URL

Logout Service のリダイレクトバインディング URL。

Logout Service Artifact Binding URL

Logout Service の **Artifact** リダイレクトバインディング URL。 **Force Artifact Binding** オプションと共に設定すると、**Artifact** バインディングがログインフローとログアウトフローの両方で強制的に実行されます。このプロパティが設定されていない限り、**Artifact** バインディングはログアウトには使用されません。

Logout Service SOAP Binding URL

Logout Service のリダイレクトバインディング URL。バックチャネルログアウト が使用されている場合にのみ適用されます。

Artifact Binding URL

HTTP アーティファクトメッセージの送信先となる URL。

Artifact Resolution Service

ArtifactResolve メッセージの送信先となるクライアント SOAP エンドポイントの URL。

12.2.4. IDP でのログイン

IDP Initiated Login は、Red Hat build of Keycloak サーバー上にエンドポイントを設定して、特定のアプリケーション/クライアントにログインできるようにする機能です。クライアントの **Settings** タブで、**IDP Initiated SSO URL Name** を指定する必要があります。これは、空白のない単純な文字列です。この後、**root/realms/{realm}/protocol/saml/clients/{url-name}** の URL でクライアントを参照できます。

IDP によって開始されるログインの実装では、**REDIRECT** バインディングよりも **POST** が優先されます (詳細については、[saml バインディング](#) を確認してください)。そのため、最終バインディングおよび SP URL は以下の方法で選択されます。

1. 特定の **Assertion Consumer Service POST Binding URL** が定義される場合 (クライアント設定の **Fine Grain SAML Endpoint Configuration** 設定)。POST バインディングはその URL で使用されます。
2. 一般的な **Master SAML Processing URL** が指定されている場合には、この一般的な URL ではなく **POST** バインディングが使用されます。
3. 最後に、(Fine Grain SAML Endpoint Configuration 内で) **Assertion Consumer Service Redirect Binding URL** が設定されている場合は、この URL で **REDIRECT** バインディングが使用されます。

クライアントに特別なリレー状態が必要な場合は、**IDP Initiated SSO Relay State** フィールドの **設定** タブで設定することもできます。あるいは、ブラウザーは **RelayState** クエリーパラメーター、つまり **root/realms/{realm}/protocol/saml/clients/{url-name}?RelayState=thestate** でリレー状態を指定できます。

アイデンティティブローカー を使用する場合は、外部 IDP からクライアントの IDP 開始ログインを設定できます。前述のように、ブローカー IDP で IDP 開始ログインに実際のクライアントが設定されている。外部 IDP は、ブローカーを示す特別な URL を参照するアプリケーションの IDP Initiated ログインにクライアントを設定し、仲介 IDP で選択したクライアントの IDP Initiated ログインエンドポイントを表す必要があります。これは、外部 IDP のクライアント設定を意味します。

- **IDP Initiated SSO URL Name** は、IDP Initiated Login initial point として公開される名前に設定されます。
- **Fine Grain SAML Endpoint Configuration** セクションの **Assertion Consumer Service POST Binding URL** は、**broker-root/realms/{broker-realm}/broker/{idp-name}/endpoint/clients/{client-id}** の URL に設定する必要があります。ここでは、以下のようになります。
 - **broker-root** はベースブローカー URL です。
 - **broker-realm** は、外部 IDP が宣言されるブローカーのレルムの名前です。
 - **IDP-name** はブローカーの外部 IDP の名前です。

- **client-id** は、ブローカーで定義された SAML クライアントの **IDP Initiated SSO URL Name** 属性の値です。このクライアントは、外部 IDP から IDP Initiated ログインで利用できます。

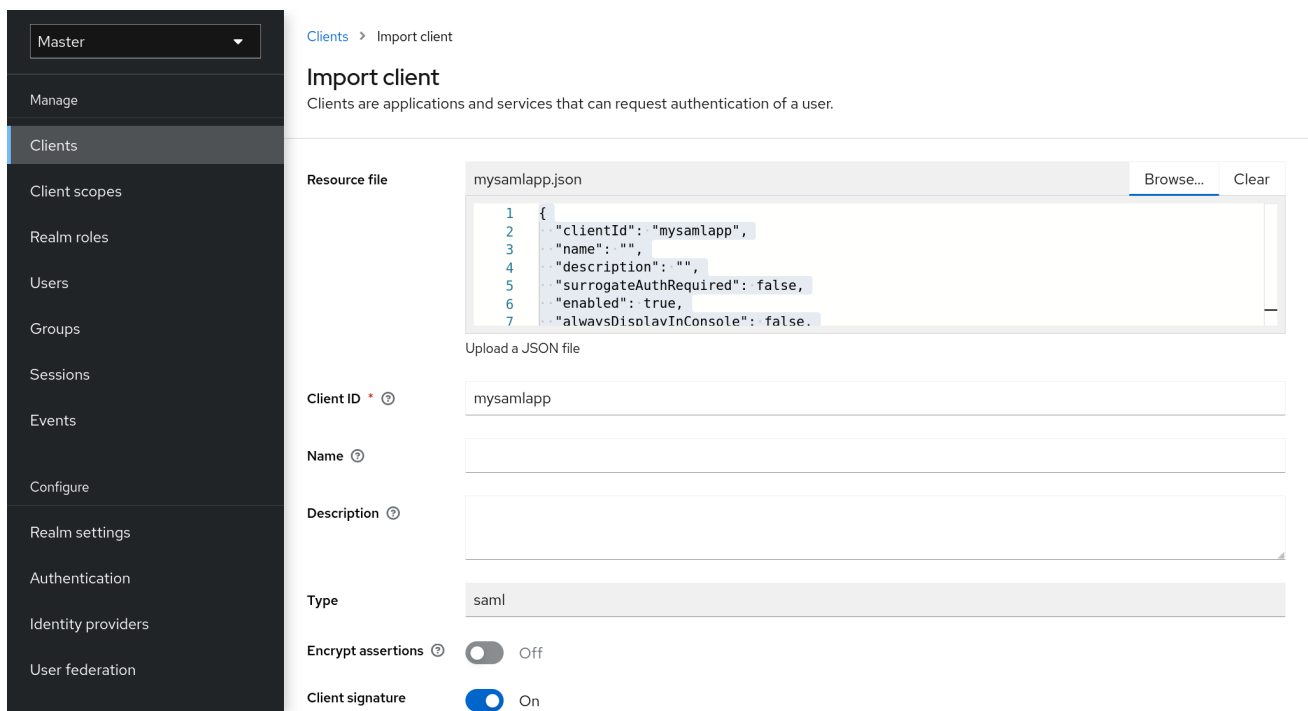
ブローカー IDP から外部 IDP のクライアント設定に基本的なクライアント設定をインポートできることに注意してください。ブローカー IDP でアイデンティティプロバイダーの設定から利用可能な [SP 記述子](#) を使用し、**clients/client-id** をエンドポイント URL に追加するだけです。

12.2.5. エンティティ記述子を使用したクライアントの作成

SAML 2.0 クライアントを手動で登録するのではなく、標準の SAML Entity Descriptor XML ファイルを使用してクライアントをインポートできます。

クライアントページには、**Import client** オプションが含まれています。

クライアントの追加



The screenshot shows the 'Import client' interface. On the left is a dark sidebar with a 'Clients' menu item highlighted. The main area has a breadcrumb 'Clients > Import client' and a title 'Import client' with a subtitle 'Clients are applications and services that can request authentication of a user.' Below this is a 'Resource file' section with a text area containing a JSON configuration for a client named 'mysamlapp'. The JSON includes fields for 'clientId', 'name', 'description', 'surrogateAuthRequired', 'enabled', and 'alwaysDisalavInConsole'. Below the JSON editor is an 'Upload a JSON file' button. Further down are input fields for 'Client ID' (filled with 'mysamlapp'), 'Name', and 'Description'. A 'Type' dropdown is set to 'saml'. At the bottom, there are two toggle switches: 'Encrypt assertions' (set to Off) and 'Client signature' (set to On).

手順

1. **Browse** をクリックします。
2. XML エンティティ記述子情報が含まれるファイルを読み込みます。
3. 情報を確認し、すべてが正しく設定されていることを確認します。

mod-auth-mellon などの SAML クライアントアダプターの中には、IDP の XML エンティティ記述子が必要です。この記述子は、以下の URL に移動して確認できます。

```
root/realms/{realm}/protocol/saml/descriptor
```

realm は、クライアントのレルムに置き換えます。

12.3. クライアントリンク

あるクライアントから別のクライアントにリンクするために、Red Hat build of Keycloak はリダイレクトエンドポイント `/realms/realm_name/clients/{client-id}/redirect` を提供します。

クライアントが **HTTP GET** リクエスト経由でこのエンドポイントにアクセスする場合、Red Hat build of Keycloak はレスポンスの **Location** ヘッダーを通じて **HTTP 307** (Temporary Redirect) の形式で提供されたクライアントおよびレルム用に設定済みのベース URL を返します。この結果、クライアントは、レルム名とそれらにリンクするクライアント ID を認識するだけで済みます。この間接参照により、クライアントベース URL のハードコーディングが回避されます。

たとえば、レルム **master** と client-id **account** がある場合:

```
http://host:port/realms/master/clients/account/redirect
```

この URL は、<http://host:port/realms/master/account> に一時的にリダイレクトします。

12.4. OIDC トークンおよび SAML アサーションマッピング

ID トークン、アクセストークン、または SAML アサーションを受信するアプリケーションは、異なるロールおよびユーザーメタデータが必要になる場合があります。

Red Hat build of Keycloak では、以下を行えます。

- ロール、要求、およびカスタム属性をハードコーディングする。
- ユーザーメタデータをトークンまたはアサーションにプルする。
- ロールの名前を変更する。

これらのアクションは、管理コンソールの **Mappers** タブで実行します。

Mappers タブ

The screenshot shows the Keycloak management console interface. On the left is a dark sidebar with navigation options: Master, Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, and Events. The 'Clients' menu item is selected. The main content area shows the breadcrumb 'Clients > Client details > Dedicated scopes' and the client name 'myapp' with the description 'This is a client scope which includes the dedicated mappers and scope'. Below this, there are two tabs: 'Mappers' (active) and 'Scope'. The 'Mappers' tab displays a large plus sign icon and the text 'No mappers'. Below this, a message states: 'If you want to add mappers, please click the button below to add some predefined mappers or to configure a new mapper.' At the bottom of the tab, there are two buttons: 'Add predefined mapper' and 'Configure a new mapper'.

新しいクライアントにはビルトインマッパーがありませんが、クライアントスコープから一部のマッパーを継承できます。詳細については、[クライアントスコープ](#) セクションを参照してください。

プロトコルマッパーは項目 (メールアドレスなど) を ID およびアクセストークンの特定の要求にマッピングします。マッパーの機能は、その名前を見ただけでわかるようにしておく必要があります。 **Add Builtin** をクリックして、事前設定されたマッパーを追加します。

各マッパーには共通の設定のセットがあります。マッパーのタイプに応じて、追加の設定を利用できます。マッパーの横にある **Edit** をクリックして設定画面にアクセスし、これらの設定を調整します。

マッパーの設定

Master

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Configure

Realm settings

Authentication

Identity providers

User federation

Clients > Client details > Dedicated scopes > Mapper details

Add mapper

If you want more fine-grain control, you can create protocol mapper on this client

Mapper type: User Realm Role

Name * ⓘ

Realm Role prefix ⓘ

Multivalued ⓘ On

Token Claim Name ⓘ

Claim JSON Type ⓘ String

Add to ID token ⓘ On

Add to access token ⓘ On

Add to userinfo ⓘ On

Save Cancel

各オプションの詳細は、ツールチップの上にマウスをかざして表示できます。

要求を配置する場所を制御するには、ほとんどの OIDC マッパーでも使用できます。**Add to ID token** と **Add to access token** のスイッチを調整して、**id** および **access** トークンから要求を追加するか、除外することができます。

以下のようにマッパータイプを追加できます。

手順

1. Mappers タブに移動します。
2. **Configure a new mapper** をクリックします。

マッパーの追加

Master

Manage

Clients

Client scopes

Realm roles

Users

Groups

Sessions

Events

Configure

Realm settings

Authentication

Clients > Client details > Dedicated scopes > Mapper details

Add mapper

If you want more fine-grain control, you can create protocol mapper on this client

Mapper type: Group Membership

Name * ⓘ

Token Claim Name ⓘ

Full group path ⓘ Off

Add to ID token ⓘ Off

Add to access token ⓘ Off

Add to userinfo ⓘ Off

Save Cancel

3. リストボックスから **Mapper Type** を選択します。

12.4.1. 優先順位

マッパー実装には **優先順位** があります。**優先順位** はマッパーの設定プロパティーではありません。これはマッパーの具体的な実装のプロパティーです。

マッパーは、マッパーリストの順番にソートされます。トークンまたはアサーションの変更は、最も低いものから順に適用されます。そのため、他の実装に依存する実装は必要な順序で処理されます。

たとえば、トークンに含まれるロールを計算するには、以下を実行します。

1. それらのロールに基づいて対象を解決します。
2. トークンにすでに利用可能なロールおよび対象を使用する JavaScript スクリプトを処理します。

12.4.2. OIDC ユーザーセッションノートマッパー

ユーザーセッションの詳細はマッパーを使用して定義され、クライアントで機能を使用できません。**Add builtin** をクリックして、セッションの詳細を追加します。

切り替え後のユーザーセッションは以下の詳細を提供します。

- **IMPERSONATOR_ID**: 偽装ユーザーの ID
- **IMPERSONATOR_USERNAME**: 偽装ユーザーのユーザー名

サービスアカウントセッションは以下の詳細を提供します。

- **clientID**: サービスアカウントのクライアント ID
- **client_id**: サービスアカウントのクライアント ID。
- **clientAddress**: サービスアカウントの認証されたデバイスのリモートホスト IP
- **clientHost**: サービスアカウントの認証デバイスのリモートホスト名

12.4.3. スクリプトマッパー

Script Mapper を使用して、ユーザー定義の JavaScript コードを実行して要求をトークンにマッピングします。サーバーにスクリプトをデプロイする方法は、[JavaScript プロバイダー](#) を参照してください。

スクリプトのデプロイ時に、利用可能なマッパーのリストからデプロイされたスクリプトを選択できるはずですが、

12.5. クライアントアダプター設定の生成

Red Hat build of Keycloak は、アプリケーションのデプロイメント環境にクライアントアダプターをインストールするために使用できる設定ファイルを生成します。OIDC と SAML の両方で、多くのアダプタータイプがサポートされます。

1. **Action** メニューをクリックし、**Download adapter config** オプションを選択します。

Download adaptor configs ✕

i description
keycloak.json file used by the Keycloak OIDC client adapter to configure clients. This must be saved to a keycloak.json file and put in your WEB-INF directory of your WAR file. You may also want to tweak this file after you download it.

Format option ⓘ

Keycloak OIDC JSON

Details ⓘ

```
{
  "realm": "master",
  "auth-server-url": "http://localhost:8180/",
  "ssl-required": "external",
  "resource": "myapp",
  "credentials": {
    "secret": "36gLgP28Ak4Czp9o3JetORP0qCZQ3jwX"
  },
  "confidential-port": 0
}
```

Download
Cancel

2. 設定用に生成された **Format Option** を選択します。

OIDC および SAML 用のすべての Red Hat build of Keycloak クライアントアダプターがサポートされています。SAML の mod-auth-mellon Apache HTTPD アダプターと、標準の SAML エンティティー記述子ファイルがサポートされます。

12.6. クライアントスコープ

Keycloak を使用して、**client scope** と呼ばれるエンティティーで共有クライアント設定を定義します。**クライアントスコープ** は、複数のクライアントの **プロトコルマッパー** と **ロールスコープマッピング** を設定します。

クライアントスコープは OAuth 2 **scope** パラメーターもサポートします。クライアントアプリケーションは、アプリケーションの要件に応じて、このパラメーターを使用してアクセストークンの要求またはロールを要求します。

クライアントスコープを作成するには、以下の手順に従います。

1. メニューの **Client Scopes** をクリックします。

クライアントスコープのリスト

The screenshot shows the 'Client scopes' management interface. On the left is a navigation menu with options like 'Manage', 'Clients', 'Client scopes', 'Realm roles', 'Users', 'Groups', 'Sessions', 'Events', and 'Configure'. The main area is titled 'Client scopes' and includes a search bar and a 'Create client scope' button. Below is a table with the following data:

Name	Assigned type	Protocol	Display order	Description
acr	Default	OpenID Connect	-	OpenID Connect scope for add acr (authentication context class reference) to the token
address	Optional	OpenID Connect	-	OpenID Connect built-in scope: address
email	Default	OpenID Connect	-	OpenID Connect built-in scope: email
microprofile-jwt	Optional	OpenID Connect	-	Microprofile - JWT built-in scope

2. **Create** をクリックします。
3. クライアントスコープに名前を付けます。
4. **Save** をクリックします。

クライアントスコープには、通常のクライアントと同様のタブがあります。[プロトコルマッパー](#)と[ロールスコープマッピング](#)を定義できます。これらのマッピングは他のクライアントで継承でき、このクライアントスコープから継承するように設定されます。

12.6.1. プロトコル

クライアントスコープを作成する場合は、**Protocol** を選択します。同じスコープにリンクされるクライアントには、同じプロトコルが必要です。

各レルムには、メニューに事前に定義された組み込みクライアントスコープのセットがあります。

- SAML プロトコル: **role_list** このスコープには、SAML アサーションのロールリストのプロトコルマッパーが1つ含まれます。
- OpenID Connect プロトコル: いくつかのクライアントスコープが利用できます。
 - **roles**
このスコープは OpenID Connect 仕様では定義されず、アクセストークンの **スコープ** 要求に自動的に追加されません。このスコープにはマッパーがあり、ユーザーのロールをアクセストークンに追加して、クライアントロールが1つ以上あるクライアントの対象を追加するために使用されます。これらのマッパーの詳細については、[オーディエンス](#) セクションを参照してください。
 - **web-origins**
このスコープは OpenID Connect 仕様には定義されず、アクセストークンを要求する **スコープ** には追加されません。これは、許可される Web オリジンをアクセストークンの **allowed-origins** 要求に追加するために使用されます。
 - **microprofile-jwt**
このスコープは、[MicroProfile/JWT 認証仕様](#) で定義された要求を処理します。このクライアントスコープは、**upn** 要求のユーザープロパティマッパーと、**groups** 要求のレルムロールマッパーを定義します。これらのマッパーは、MicroProfile/JWT 固有の要求を作成するために、これらのマッパーを変更できます。
 - **offline_access**

このスコープは、クライアントがオフライントークンを取得する必要がある場合に使用されます。オフライントークンの詳細については、[オフラインアクセス](#) セクションと [OpenID Connect 仕様](#) を参照してください。

- profile
- email
- address
- phone

クライアントスコーププロファイル、**profile**、**email**、**address**、および **phone** は、[OpenID Connect 仕様](#) に定義されています。これらのスコープにはロールスコープマッピングが定義されていませんが、プロトコルマッパーが定義されます。これらのマッパーは OpenID Connect 仕様で定義された要求に対応します。

たとえば、**phone** クライアントのスコープ、**Mappers** タブの順に開くと、プロトコルマッパーが表示されます。これは、スコープの **phone** の仕様で定義される要求に対応します。

クライアントスコープマッパー

The screenshot shows the 'Client scopes > Client scope details' page for the 'phone' scope. The 'Mappers' tab is selected, showing a search bar and an 'Add mapper' button. Below is a table of mappers:

Name	Category	Type	Priority
phone number	Token mapper	User Attribute	0
phone number verified	Token mapper	User Attribute	0

phone クライアントスコープがクライアントにリンクされると、そのクライアントは **phone** クライアントスコープで定義されたすべてのプロトコルマッパーを自動的に継承します。このクライアントに発行されたアクセストークンには、ユーザーに関する電話番号 (電話番号が定義されているか) が含まれます。

組み込みクライアントスコープには、仕様で定義されているプロトコルマッパーが含まれます。クライアントスコープを編集し、プロトコルマッパーまたはロールスコープマッピングを作成、更新、または削除できます。

12.6.2. 関連設定

クライアントスコープには、同意画面に関連するオプションが含まれます。これらのオプションは、リンクされたクライアントで **Consent Required** が有効になっている場合に、役立ちます。

画面の表示

Display On Consent Screen が有効になっていて、このスコープが同意が必要なクライアントに追加されると、**Consent Screen Text** で指定したテキストが同意画面に表示されます。このテキストは、ユーザーが認証され、そのユーザーが Red Hat build of Keycloak からクライアントにリダイレクトされる前に表示されます。**Display On Consent Screen** が無効な場合には、このクライアントスコープは同意画面に表示されません。

Consent Screen Text

このクライアントスコープが、必要なデフォルト値がクライアントの範囲名である一部のクライア

ントに追加されると、consent 画面にテキストが表示されます。このテキストの値は、`${var-name}` 文字列で代入変数を指定してカスタマイズできます。カスタマイズされる値は、テーマのプロパティファイル内で設定されます。詳細は、[サーバー開発者ガイド](#) を参照してください。

12.6.3. クライアントとリンククライアントスコープ

クライアントスコープとクライアント間のリンクは、クライアントの **Client Scopes** タブで設定されます。クライアントスコープとクライアント間でリンクする方法が 2 つあります。

デフォルトのクライアントスコープ

この設定は、OpenID Connect および SAML クライアントに適用されます。デフォルトのクライアントスコープは、このクライアントの OpenID Connect トークンまたは SAML アサーションを発行するときに適用されます。クライアントはプロトコルマッパーとロールスコープのマッピングをクライアントスコープで継承します。OpenID Connect Protocol の場合、OpenID Connect の認可要求の `scope` パラメーターで使用される値にかかわらず、マッパーおよびロールの範囲マッピングは常に適用されます。

オプションのクライアントスコープ

この設定は、OpenID Connect クライアントにのみ適用されます。オプションのクライアントスコープはこのクライアントのトークンを発行するときに適用されますが、これらは OpenID Connect 認可要求で `scope` パラメーターによって要求される場合にのみ適用されます。

12.6.3.1. 例

この例では、クライアントに `profile` と `email` がデフォルトのクライアントスコープとしてリンクされていることと、`phone` と `address` がオプションのクライアントスコープとしてリンクされていることを前提とします。クライアントは、要求を OpenID Connect 認可エンドポイントに送信する際に、`scope` パラメーターの値を使用します。

```
scope=openid phone
```

`scope` パラメーターには文字列が含まれ、スコープの値はスペースで区切られます。`openid` の値は、すべての OpenID Connect 要求に使用されるメタ値です。トークンには、クライアントスコープの `profile` および `email` (デフォルトのスコープ)、および `phone` (`scope` パラメーターで要求される任意のクライアントスコープ) からのマッパーおよびロールスコープのマッピングが含まれます。

12.6.4. クライアントスコープの評価

Mappers タブにはプロトコルマッパーが含まれ、**Scope** タブにはこのクライアントに対して宣言されたロールスコープマッピングが含まれます。クライアントスコープから継承されるマッパーおよびスコープのマッピングは含まれません。有効なプロトコルマッパー (クライアント自体に定義されたプロトコルマッパー、リンクされたクライアントスコープから継承されるプロトコルマッパー) と、クライアントのトークンの生成時に使用される効果的なロールスコープマッピングを確認できます。

手順

1. クライアントの **クライアントスコープ** タブをクリックします。
2. サブタブ **Evaluate** を開きます。
3. 適用する任意のクライアントスコープを選択します。

これにより、`scope` パラメーターの値も表示されます。このパラメーターは、アプリケーションから Red Hat build of Keycloak OpenID Connect 認証エンドポイントに送信する必要があります。

クライアントスコープの評価

The screenshot shows the 'Client details' page for a client named 'account'. The 'Client scopes' tab is selected, and the 'Evaluate' sub-tab is active. The page displays a JSON token and a list of effective mappings.

```

{
  "exp": 1654088072,
  "iat": 1654088012,
  "jti": "b41a3190-72e8-4aff-82b3-4eee2d283930",
  "iss": "http://localhost:8180/realms/master",
  "sub": "007a55d0-3366-402c-9ea7-350872387577",
  "typ": "Bearer",
  "azp": "account",
  "session_state": "91cf48ef-7730-40e0-h9ca-390dc2ba411a"
}

```

Effective protocol mappers

Effective role scope mappings

Generated access token

Generated ID token



注記

アプリケーションから **scope** パラメーターのカスタム値を送信するには、サーバーレットアダプターの場合は [パラメーター転送セクション](#)、javascript アダプターの場合は [javascript アダプターセクション](#) を参照してください。

すべての例は、**scope** パラメーターに指定された値を使用して、特定のユーザー用に生成され、特定のクライアント用に発行されます。この例には、使用されるすべての要求およびロールマッピングが含まれます。

12.6.5. クライアントスコープのパーミッション

ユーザーにトークンを発行する場合、クライアントスコープは、ユーザーがトークンを使用できる場合にのみ適用されます。

クライアントスコープにロールスコープマッピングが定義されていない場合は、各ユーザーはこのクライアントスコープを使用できます。ただし、クライアントスコープにロールスコープマッピングが定義されている場合に、ユーザーは少なくとも1つのロールに所属する必要があります。ユーザーロールとクライアントスコープのロール間に交差部分が必要です。この交差点の評価には、複合ロールが考慮されます。

ユーザーがクライアントスコープを使用できない場合には、トークンの生成時にプロトコルマッパーまたはロールスコープのマッピングは使用されません。クライアントスコープはトークンの **scope** 値には表示されません。

12.6.6. レalmのデフォルトクライアントスコープ

Realm Default Client Scopes を使用して、新たに作成されたクライアントに自動的にリンクされるクライアントスコープのセットを定義します。

手順

1. クライアントの **クライアントスコープ** タブをクリックします。
2. **デフォルトのクライアントスコープ** をクリックします。

ここから、**Default Client Scopes** として追加するクライアントスコープを選択し、**Optional Client Scopes** を新規作成したクライアントに設定します。

デフォルトのクライアントスコープ

クライアントが作成されると、必要に応じて、デフォルトのクライアントスコープのリンクを解除できます。これは、[デフォルトロール](#) の削除に似ています。

12.6.7. 記述されたスコープ

クライアントスコープ

クライアントスコープは、レルムレベルで設定され、クライアントにリンクできる Red Hat build of Keycloak 内のエンティティです。**scope** パラメーターの対応する値で Keycloak 認可エンドポイントに要求を送信すると、クライアントスコープはその名前によって参照されます。詳細については、[クライアントスコープのリンク](#) セクションを参照してください。

ロールスコープのマッピング

これは、クライアントまたはクライアントスコープの **Scope** タブで利用できます。**ロールスコープマッピング** を使用して、アクセストークンで使用できるロールを制限します。詳細については、[ロールスコープマッピング](#) セクションを参照してください。

12.7. クライアントポリシー

クライアントアプリケーションを簡単に保護するには、以下の点を統一して認識しておく便利です。

- クライアントの実行できる設定でのポリシーの設定
- クライアント設定の検証
- FAPI (enterprise-grade API) などの必要なセキュリティー標準やプロファイルへの適合

これらのポイントを統一された方法で実現するために、**クライアントポリシー** の概念が導入されました。

12.7.1. ユースケース

クライアントポリシーでは、以下の点を実現しています。

クライアントの実行できる設定でのポリシーの設定

クライアントの設定は、クライアントの作成/更新中だけでなく、特定のクライアントに関連する Red Hat build of Keycloak サーバーへの OpenID Connect 要求時にも、クライアントポリシーで適用できます。Red Hat build of Keycloak は、[アプリケーションおよびサービスのセキュリティ保護ガイド](#) で説明されているクライアント登録ポリシーを使用する場合も、同様の方法をサポートしています。ただし、クライアント登録ポリシーは、OIDC 動的クライアント登録のみに対応します。クライアントポリシーは、クライアント登録ポリシーが実行できるものだけでなく、他のクライアント登録および設定方法もカバーします。現在の計画では、クライアント登録はクライアントポリシーに置き換えられます。

クライアント設定の検証

Red Hat build of Keycloak は、認可エンドポイントやトークンエンドポイントなどの一部のエンドポイントで、クライアントが Proof Key for Code Exchange、Request Object Signing Algorithm、Holder-of-Key Token などの設定に従うかどうかの検証をサポートします。これらは、各設定項目 (管理コンソール、スイッチ、プルダウンメニューなど) で指定できます。クライアントアプリケーションをセキュアにするには、管理者が適切な方法で多くの設定を指定する必要があるため、管理者がクライアントアプリケーションのセキュリティを保護することは困難になります。クライアントポリシーは、上記のクライアント設定に対してこれらの検証を行うことができ、高度なセキュリティ要件を満たすように、一部のクライアント設定スイッチの自動設定に使用できます。今後、個々のクライアント設定が、直接必要な検証を実行するクライアントポリシーに置き換えられる可能性があります。

FAPI などの必要なセキュリティ標準やプロファイルへの適合

Global クライアントプロファイル は、デフォルトで Red Hat build of Keycloak に事前設定されたクライアントプロファイルです。[FAPI](#) などの標準のセキュリティプロファイルに準拠するように事前設定されているため、管理者は容易にクライアントアプリケーションを特定のセキュリティプロファイルに準拠させることができます。現時点では、Red Hat build of Keycloak には、FAPI 1 仕様をサポートするためのグローバルプロファイルがあります。管理者は、FAPI に準拠するクライアントを指定するようにクライアントポリシーを設定する必要があります。管理者は、クライアントプロファイルとクライアントポリシーを設定できるため、容易に Red Hat build of Keycloak クライアントを SPA、ネイティブアプリケーション、Open Banking などの他のセキュリティプロファイルに準拠させることができます。

12.7.2. プロトコル

クライアントポリシーの概念は、特定のプロトコルからは独立しています。ただし、Red Hat build of Keycloak は現在、[OpenID Connect \(OIDC\) プロトコル](#) のみをサポートしています。

12.7.3. アーキテクチャー

クライアントポリシーは、Condition、Executor、Profile、および Policy の 4 つのビルディングブロックで設定されます。

12.7.3.1. 条件

条件は、ポリシーが採用されるクライアントと、そのクライアントが採用されるタイミングを決定します。一部の条件は、クライアント要求 (OIDC 認可要求、トークンエンドポイント要求など) で他の条件がクライアント要求時にチェックされる場合に、クライアントの作成/更新時にチェックされます。条

件は、指定基準の1つが満たされているかどうかを確認します。たとえば、一部の条件では、クライアントのアクセスタイプが機密であるかどうかをチェックします。

この条件は単独で使用できません。後述の [ポリシー](#) で使用できます。

他の設定可能なプロバイダーと同じ条件を設定できます。設定可能なものは、各条件の性質によって異なります。

以下の条件が提供されます。

クライアントの作成/更新方法

- 動的クライアント登録 (初期アクセストークンまたは登録アクセストークンで認証されていない、または認証されていない)
- Admin REST API(管理コンソールなど)

たとえば、クライアントの作成時に、このクライアントが初期アクセストークンのない OIDC 動的クライアント登録 (匿名動的クライアント登録) で作成される場合に、条件が true に評価されるように設定できます。たとえば、この条件を使用して、OIDC 動的クライアント登録などで登録されたすべてのクライアントが FAPI に準拠するように使用できます。

クライアントの作成者 (特定のロールまたはグループに存在するかどうかで確認)

OpenID Connect 動的クライアント登録では、クライアントの作成者は、認証済みのエンドユーザーで、アクセストークンを使用して登録エンドポイントに実際にアクセスする既存のクライアントのサービスアカウントではなく、新しいクライアントを生成するためのアクセストークンを取得できます。Admin REST API による登録では、クライアントの作成者は Red Hat build of Keycloak の管理者のようなエンドユーザーです。

クライアントアクセスタイプ (機密、パブリック、Bearer のみ)

たとえば、クライアントが認可要求を送信すると、このクライアントが機密である場合はポリシーが採用されます。パブリッククライアントがクライアント認証を無効にしている場合、機密クライアントはクライアント認証を有効にしています。Bearer-only は非推奨のクライアントタイプです。

クライアントスコープ

クライアントに固有のクライアントスコープがある場合 (デフォルトまたは現在のリクエストで使用される任意のスコープ) は、true と評価します。たとえば、スコープが **fapi-example-scope** の OIDC 認可要求が FAPI に準拠する必要があることを確認できます。

クライアントロール

指定の名前のクライアントロールが割り当てられたクライアントに適用されます。通常、要求されたクライアントに対して指定された名前のクライアントロールを作成し、それを "マーカーロール" として使用して、要求されたクライアントに指定されたクライアントポリシーが適用されるようにすることができます。

クライアントドメイン名、ホスト名、または IP アドレス

クライアントの特定のドメイン名に適用されます。または、管理者が特定のホストまたは IP アドレスからクライアントを登録/更新する場合。

任意のクライアント

この条件は常に true と評価されます。たとえば、特定のレルム内のすべてのクライアントが FAPI に準拠することを確認するために使用できます。

12.7.3.2. エグゼキューター (Executor)

エグゼキューターは、ポリシーが採用されるクライアントで実行されるアクションを指定します。エグゼキューターは、1つまたは複数の指定されたアクションを実行します。たとえば、一部のエクゼ

キューターは、認可要求のパラメーター `redirect_uri` の値が Authorization Endpoint の事前登録リダイレクト URI と完全に一致するかどうかを確認し、一致しない場合はこの要求を拒否します。

エグゼキューターは、単独では使用できません。後述の [プロファイル](#) で使用できます。

エグゼキューターは、他の設定可能なプロバイダーと同じように設定できます。設定可能なものは、各エグゼキューターの性質によって異なります。

エグゼキューターはさまざまなイベントで機能します。エグゼキューター実装は、特定のタイプのイベントを無視できます (たとえば、OIDC 要求 オブジェクトをチェックするためのエグゼキューターは、OIDC 許可要求に対してのみ機能します)。イベントは以下のとおりです。

- クライアントの作成 (動的クライアント登録による作成を含む)
- クライアントの更新
- 認可要求の送信
- トークン要求の送信
- トークン更新要求の送信
- トークン失効要求の送信
- トークンイントロスペクション要求の送信
- userinfo 要求の送信
- リフレッシュトークンを使用してログアウト要求を送信します (リフレッシュトークンを使用したログアウトは、どの仕様でもサポートされていないプロプライエタリー Red Hat build of Keycloak 機能であることに注意してください)。むしろ、[公式の OIDC ログアウト](#) に頼ることを推奨します。

各イベントでは、エグゼキューターは複数のフェーズで機能します。たとえば、クライアントの作成/更新時に、エグゼキューターは特定のクライアント設定を自動設定してクライアント設定を変更できます。その後、エグゼキューターはこの設定を検証フェーズで検証します。

このエグゼキューターの目的の1つは、FAPI などのクライアント適合性プロファイルのセキュリティー要件を実現することです。これを行うには、以下のエグゼキューターが必要です。

- クライアントにセキュアな [クライアント認証方式](#) が使用されるようにします。
- [Holder-of-key トークン](#) が使用されるようにします。
- [Proof Key for Code Exchange \(PKCE\)](#) が使用されるようにします。
- [署名付き JWT クライアント認証 \(private-key-jwt\)](#) のセキュアな署名アルゴリズムが使用されるようにします。
- HTTPS リダイレクト URI を強制的に使用し、設定したリダイレクト URL にワイルドカードが含まれないようにします。
- 高いセキュリティーレベルを満たす [OIDC 要求 オブジェクト](#) を有効にします。
- FAPI1 仕様で説明されているように、[デタッチされた署名](#) として使用される ID トークンを含む [OIDC ハイブリッドフローの応答タイプ](#) を有効にします。つまり、認証応答から返される ID トークンにはユーザーのプロファイルデータが含まれないようにします。

- CSRF を防止するために、よりセキュアな **状態** および **nonce** パラメーターの処理を有効にします。
- クライアント登録時によりセキュアな署名アルゴリズムを有効にします。
- **binding_message** パラメーターを CIBA 要求に強制的に使用します。
- [クライアントシークレットローテーション](#) を適用します。
- クライアント登録アクセストークンを適用します。
- UK OpenBanking のようなアクセストークンを取得するための認可コードフローを開始する前にインテントが発行されるユースケースでは、クライアントがインテントの発行先であるかどうかのチェックを強制します。
- 暗黙的およびハイブリッドフローの禁止を強制します。
- PAR 要求に、認可要求に含まれる必要なパラメーターが含まれているかどうかのチェックを強制します。
- [リフレッシュトークンのローテーション](#) をスキップし、リフレッシュトークンの応答からリフレッシュトークンが返されないようにします。

12.7.3.3. プロファイル

プロファイルは複数のエグゼキューターで設定されており、FAPI などのセキュリティープロファイルを導入できます。プロファイルは、Admin REST API (Admin Console) によりエグゼキューターとともに設定できます。**グローバルプロファイル** が3つ存在します。これらは、デフォルトでは FAPI Baseline、FAPI Advanced および FAPI CIBA 仕様に準拠する事前設定されたエグゼキューターを使用して、Red Hat build of Keycloak で設定されます。詳細は、[アプリケーションおよびサービスの保護ガイド](#) の FAPI セクションを参照してください。

12.7.3.4. ポリシー

ポリシーは複数の条件およびプロファイルで設定されます。このポリシーは、対象ポリシーの全条件を満たすクライアントに採用できます。このポリシーは複数のプロファイルを参照し、これらのプロファイルの全エグゼキューターは、このポリシーが採用されるクライアントに対してそのタスクを実行します。

12.7.4. 設定

ポリシー、プロファイル、条件、エグゼキューターは Admin REST API で設定できます。これは管理コンソールでもあります。これを行うために **Realm → Realm Settings → Client Policies** タブがあり、管理者はレルムごとにクライアントポリシーを持つことができます。

グローバルクライアントプロファイル は、各レルムで自動的に利用できます。ただし、デフォルトではクライアントポリシーは設定されません。つまり、レルムのクライアントを FAPI に準拠させる場合などに、管理者はクライアントポリシーを作成する必要があります。グローバルプロファイルは更新できませんが、管理者はそれらをテンプレートとして簡単に使用し、グローバルプロファイル設定で多少変更を加える場合は、独自のプロファイルを作成できます。管理コンソールでは JSON Editor があり、一部のグローバルプロファイルに基づいて新規プロファイルを簡単に作成できます。

12.7.5. 後方互換性

クライアントポリシーは、[アプリケーションおよびサービスの保護ガイド](#) で説明されているクライアン

ト登録ポリシーを置き換えることができます。ただし、クライアント登録ポリシーは引き続き共存します。これは、クライアントの作成/更新要求時などに、クライアントポリシーとクライアント登録ポリシーの両方が適用されることを意味します。

現在の計画では、クライアント登録ポリシー機能が削除され、既存のクライアント登録ポリシーは自動的に新しいクライアントポリシーに移行されます。

12.7.6. クライアントシークレットローテーション例

[クライアントシークレットローテーション](#) の設定例を参照してください。

第13章 VAULT を使用したシークレットの取得

Keycloak は現在、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 という、すぐに使用できる Vault SPI の実装を 2 つ提供しています。

直接入力せずに Vault からシークレットを取得するには、以下の特別に作成された文字列を適切なフィールドに入力します。

```
`${vault.key}
```

key は vault によって認識されたシークレットの名前です。

レルム間でシークレットのリークを防ぐために、Red Hat build of Keycloak はレルム名と vault 式から取得した **key** を組み合わせます。このメソッドは、キーが vault のエントリーに直接マップされるのではなく、**key** をレルム名と組み合わせるために使用されるアルゴリズムに従って最終的なエントリー名を作成します。ファイルベースの Vault の場合、そのような組み合わせは特定のファイル名に反映されます。Java KeyStore ベースの Vault の場合、それは特定のエイリアス名になります。

以下のフィールドの vault からシークレットを取得できます。

SMTP パスワード

レルム [SMTP 設定](#)

LDAP バインド認証情報

LDAP ベースのユーザーフェデレーションの [LDAP 設定](#)

OIDC アイデンティティプロバイダーシークレット

アイデンティティプロバイダー [OpenID Connect Config](#) 内の [クライアントシークレット](#)

13.1. キーリゾルバー

すべての組み込みプロバイダーはキーリゾルバーの設定をサポートします。キーリゾルバーは、レルム名とキーを組み合わせる (**`\${vault.key}** 式から取得)、vault からシークレットの取得に使用される最終エントリー名を組み合わせるためのアルゴリズムまたはストラテジーを実装します。Red Hat build of Keycloak は、**keyResolvers** プロパティを使用して、プロバイダーが使用するリゾルバーを設定します。この値は、リゾルバー名のコンマ区切りリストです。**files-plaintext** プロバイダーの設定例を以下に示します。

```
kc.[sh|bat] start --spi-vault-file-key-resolvers=REALM_UNDERSCORE_KEY,KEY_ONLY
```

リゾルバーは、設定で宣言するのと同じ順序で実行されます。Red Hat build of Keycloak は、リゾルバーが生成する最後のエントリー名を使用し、レルムと vault キーを組み合わせる vault のシークレットを検索します。Red Hat build of Keycloak がシークレットを見つけると、そのシークレットを返します。そうでない場合、Red Hat build of Keycloak は次のリゾルバーを使用します。この検索は、Red Hat build of Keycloak が空ではないシークレットを見つけるか、リゾルバーがなくなるまで続行されます。Red Hat build of Keycloak がシークレットを見つけられない場合、Red Hat build of Keycloak は空のシークレットを返します。

前の例では、Red Hat build of Keycloak は最初に **REALM_UNDERSCORE_KEY** リゾルバーを使用します。Red Hat build of Keycloak がそのリゾルバーを使用しているエントリーを Vault 内で見つけた場合、Red Hat build of Keycloak はそのエントリーを返します。そうでない場合、Red Hat build of Keycloak は **KEY_ONLY** リゾルバーを使用して再度検索します。Red Hat build of Keycloak が **KEY_ONLY** リゾルバーを使用してエントリーを見つけた場合、Red Hat build of Keycloak はそのエントリーを返します。Red Hat build of Keycloak がすべてのリゾルバーを使用すると、Red Hat build of Keycloak は空のシークレットを返します。

現在利用可能なリゾルバーのリストは以下のようになります。

Name	説明
KEY_ONLY	Red Hat build of Keycloak はレルム名を無視し、Vault 式のキーを使用します。
REALM_UNDERSCORE_KEY	Red Hat build of Keycloak は、アンダースコア文字を使用してレルムとキーを組み合わせます。Red Hat build of Keycloak は、レルムまたはキーでのアンダースコアの出現を別のアンダースコア文字でエスケープします。たとえば、レルムが master_realm と呼ばれ、キーが smtp_key の場合は、組み合わせたキーは master__realm_smtp__key になります。
REALM_FILESEPARATOR_KEY	Red Hat build of Keycloak は、プラットフォームファイル区切り文字を使用してレルムとキーを組み合わせます。

ビルトインプロバイダーのリゾルバーを設定していない場合、Red Hat build of Keycloak は **REALM_UNDERSCORE_KEY** を選択します。

第14章 イベントを追跡する監査の設定

Red Hat build of Keycloak には、一連の監査機能が含まれています。すべてのログインおよび管理者のアクションを記録し、管理コンソールでそれらのアクションを確認できます。Red Hat build of Keycloak には、イベントをリッスンしてアクションをトリガーできる Listener SPI も含まれています。ビルトインリスナーの例には、ログファイルとイベント発生時のメールの送信が含まれます。

14.1. ユーザーイベントの監査

ユーザーに影響するすべてのイベントを記録および表示できます。Red Hat build of Keycloak は、ユーザーがログインに成功した場合、ユーザーが間違っただパスワードを入力した場合、ユーザーがアカウントを更新した場合などのアクションに対して、ログインイベントをトリガーします。デフォルトでは、Red Hat build of Keycloak は、管理コンソールにイベントを保存または表示しません。管理コンソールとサーバーのログファイルにエラーイベントのみがログに記録されます。

手順




この手順を使用して、ユーザーイベントの監査を開始します。

1. メニューで **Realm Settings** をクリックします。
2. **Events** タブをクリックします。
3. **User events settings** タブをクリックします。
4. **Save events** を **ON** に切り替えます。

ユーザーイベントの設定

[Event listeners](#)[User events settings](#)[Admin events settings](#)

User events configuration

Save events  OnExpiration Minutes [Save](#)[Revert](#)Clear user events [Clear user events](#)[Add saved types](#)

Event saved type	Description
Login	Login

5. Expiration フィールドにイベントを保存する時間を指定します。
6. Add saved types をクリックして、保存できる他のイベントを表示します。

タイプを追加

Add types

✕

✕ →

1-2 ▾ < >

	Event saved type	Description
<input checked="" type="checkbox"/>	Refresh token	Refresh token
<input checked="" type="checkbox"/>	Refresh token error	Refresh token error

1-2 ▾ < >

Add
Cancel

7. **Add** をクリックします。

保存したすべてのイベントを削除するには、**Clear user events** をクリックします。

手順

イベントを閲覧できるようになりました。

1. メニューの **Events** タブをクリックします。

ユーザーイベント

User events

Admin events

Search user event ▾

Refresh

1-4 ▾ < >

	Time	User	Event type	IP address	Client
>	May 2, 2022 4:00 PM	a4d4e4a8-3440-46f0-b29f-4bcl2ec45e44	✔ CODE_TO_TOKEN	127.0.0.1	security-admin-console
>	May 2, 2022 4:00 PM	a4d4e4a8-3440-46f0-b29f-4bcl2ec45e44	✔ LOGIN	127.0.0.1	security-admin-console
>	May 2, 2022 2:55 PM	a4d4e4a8-3440-46f0-b29f-4bcl2ec45e44	✔ CODE_TO_TOKEN	127.0.0.1	security-admin-console
>	May 2, 2022 2:55 PM	a4d4e4a8-3440-46f0-b29f-4bcl2ec45e44	✔ LOGIN	127.0.0.1	security-admin-console

2. イベントをフィルタリングするには、**Search user event** をクリックします。

ユーザーイベントの検索

User events
Admin events

Refresh

User ID

Event type

Client

Date(from)

Date(to)

LOGIN ×
✕ ▼

LOGIN
✓

LOGIN_ERROR

REGISTER

REGISTER_ERROR

LOGOUT

Search events

14.1.1. イベントタイプ

ログインイベント:

イベント	説明
ログイン	ユーザーがログインする。
登録	ユーザーが登録する。
ログアウト	ユーザーがログアウトする。
トークンへのコード	アプリケーションまたはクライアントはトークンのコードを交換する。
トークンの更新	アプリケーションまたはクライアントはトークンを更新する。

アカウントイベント:

イベント	説明
ソーシャルリンク	ユーザーアカウントはソーシャルメディアプロバイダーにリンクされます。
ソーシャルリンクの削除	ソーシャルメディアアカウントからユーザーアカウントへのリンク。
メールの更新	アカウントのメールアドレスを変更します。
プロフィールの更新	アカウントのプロファイルを変更します。
パスワードリセットの送信	Red Hat build of Keycloak は、パスワードリセットメールを送信します。
パスワードの更新	アカウントのパスワードを変更します。
TOTP の更新	アカウントの時間ベースのワンタイムパスワード (TOTP) 設定を変更します。
TOTP の削除	Red Hat build of Keycloak は、アカウントから TOTP を削除します。
確認メールの送信	Red Hat build of Keycloak は、メールを検証するためのメールを送信します。
メールの確認	Red Hat build of Keycloak は、アカウントのメールアドレスを検証します。

各イベントには、対応するエラーイベントがあります。

14.1.2. イベントリスナー

イベントリスナーはイベントをリッスンし、そのイベントに基づいてアクションを実行します。Red Hat build of Keycloak には、Logging Event Listener と Email Event Listener の 2 つのビルトインリスナーが含まれています。

14.1.2.1. Logging Event Listener (ロギングイベントリスナー)

Logging Event Listener を有効にすると、このリスナーはエラーイベントの発生時にログファイルに書き込みます。

Logging Event Listener からのログメッセージの例:

```
11:36:09,965 WARN [org.keycloak.events] (default task-51) type=LOGIN_ERROR, realmId=master,
clientId=myapp,
userId=19aeb848-96fc-44f6-b0a3-59a17570d374, ipAddress=127.0.0.1,
error=invalid_user_credentials, auth_method=openid-connect, auth_type=code,
redirect_uri=http://localhost:8180/myapp,
code_id=b669da14-cdbb-41d0-b055-0810a0334607, username=admin
```

Logging Event Listener を使用して、ハッカーのボット攻撃から保護できます。

1. **LOGIN_ERROR** イベントのログファイルを解析します。
2. 失敗したログインイベントの IP アドレスを抽出します。
3. IP アドレスを侵入防止ソフトウェアフレームワークツールに送信します。

Logging Event Listener は、イベントを **org.keycloak.events** ログカテゴリに記録します。Red Hat build of Keycloak には、デフォルトでサーバーログにデバッグログイベントが含まれていません。

デバッグログイベントをサーバーログに含めるには、以下を実行します。

1. **org.keycloak.events** カテゴリのログレベルを変更します。
2. Logging Event Listener によって使用されるログレベルを変更します。

Logging Event Listener によって使用されるログレベルを変更するには、以下を追加します。

```
bin/kc.[sh|bat] start --spi-events-listener-jboss-logging-success-level=info --spi-events-listener-jboss-logging-error-level=error
```

ログレベルの有効な値は **debug**、**info**、**warn**、**error**、および **fatal** です。

14.1.2.2. Email Event Listener (メールイベントリスナー)

Email Event Listener は、イベント発生時にユーザーのアカウントにメールを送信し、以下のイベントをサポートします。

- ログインエラー
- パスワードの更新
- 時間ベースのワンタイムパスワード (TOTP) を更新します。
- 時間ベースのワンタイムパスワード (TOTP) を削除します。

手順

メールリスナーを有効にするには、以下を実行します。

1. メニューで **Realm Settings** をクリックします。
2. **Events** タブをクリックします。
3. **Event listeners** フィールドをクリックします。
4. **Email** を選択します。

Event listeners

Event listeners ?

Save

Revert

--spi-events-listener-email-exclude-events 引数を使用してイベントを除外できます。以下に例を示します。

```
kc.[sh|bat] --spi-events-listener-email-exclude-events=UPDATE_TOTP,REMOVE_TOTP
```

--spi-events-store-jpa-max-detail-length 引数を使用して、データベース内の各イベント詳細の最大長を設定できます。この設定は、詳細 (redirect_uri など) が長い場合に便利です。以下に例を示します。

```
kc.[sh|bat] --spi-events-store-jpa-max-detail-length=1000
```

また、**--spi-events-store-jpa-max-field-length** 引数を使用して、すべてのイベントの詳細の最大長を設定できます。この設定は、基礎となるストレージ制限を遵守したい場合に便利です。以下に例を示します。

```
kc.[sh|bat] --spi-events-store-jpa-max-field-length=2500
```

14.2. 管理イベントの監査

管理コンソールの管理者が実行するすべてのアクションを記録できます。管理コンソールは、Red Hat build of Keycloak REST インターフェイスを呼び出すことで管理アクションを実行し、Red Hat build of Keycloak はこれらの REST 呼び出しを監査します。作成されるイベントは管理コンソールで確認できます。

手順

この手順を使用して、管理アクションの監査を開始します。

1. メニューで **Realm Settings** をクリックします。
2. **Events** タブをクリックします。
3. **Admin events settings** タブをクリックします。
4. **Save events** を **ON** に切り替えます。
Red Hat build of Keycloak は、**Include representation** スイッチを表示します。
5. **Include Representation** を **ON** に切り替えます。
Include Representation スイッチには、admin REST API から送信された JSON ドキュメントが含まれるため、管理者アクションを表示できます。

管理イベントの設定

Master Enabled Action ▾

Realm settings are settings that control the options for users, applications, roles, and groups in the current realm. [Learn more](#)

<
General
Login
Email
Themes
Keys
Events
Localization
Security defens
>

Event listeners
User events settings
Admin events settings

Admin events configuration

Save events ? On

Include representation ? Off

Save Revert

Clear admin events ? Clear admin events

6. **Save** をクリックします。
7. 保存されたアクションのデータベースを削除するには、**Clear admin events** をクリックします。

手順

管理イベントを表示できるようになりました。

1. メニューの **Events** をクリックします。
2. **Admin events** タブをクリックします。

管理イベント

User events
Admin events

Refresh

Time	Resource path	Resource type	Operation type	User
April 29, 2022 9:06 PM	events/config	REALM	UPDATE	a4d4e4a8-3440-46f0-b29f-4bc12ec45e44
April 29, 2022 8:48 PM	events/config	REALM	UPDATE	a4d4e4a8-3440-46f0-b29f-4bc12ec45e44
April 29, 2022 7:57 PM	events/config	REALM	UPDATE	a4d4e4a8-3440-46f0-b29f-4bc12ec45e44

Include Representation スイッチがオンの場合、データベースに大量の情報が格納される可能性があります。--spi-events-store-jpa-max-field-length 引数を使用して、表現の最大長を設定できます。この設定は、基礎となるストレージ制限を遵守したい場合に便利です。以下に例を示します。

```
kc.[sh|bat] --spi-events-store-jpa-max-field-length=2500
```

第15章 セキュリティー脅威の軽減

セキュリティー脆弱性は任意の認証サーバーに存在します。詳細は、Internet Engineering Task Force (IETF) [OAuth 2.0 Threat Model](#) および [OAuth 2.0 Security Best Current Practice](#) を参照してください。

15.1. ホスト

Red Hat build of Keycloak は、トークン発行者フィールドやパスワードリセットメールの URL など、いくつかの方法でパブリックホスト名を使用します。

デフォルトでは、ホスト名はリクエストヘッダーから導出します。ホスト名が有効であることを確認する検証は存在しません。Red Hat build of Keycloak でロードバランサーやプロキシを使用しない場合は、無効なホストヘッダーを回避するために、使用可能なホスト名を設定してください。

ホスト名の Service Provider Interface (SPI) は、要求のホスト名を設定する方法を提供します。この組み込みプロバイダーを使用してフロントエンド要求の固定 URL を設定し、リクエスト URI に基づいてバックエンドリクエストを許可できます。組み込みプロバイダーに必要な機能がない場合は、プロバイダーをカスタマイズ開発できます。

15.2. 管理エンドポイントおよび管理コンソール

Red Hat build of Keycloak は、管理 REST API と Web コンソールを、管理者以外のユーザーが使用するのと同じポートに公開します。外部アクセスが必要なければ、管理エンドポイントを外部に公開できません。

15.3. 総当たり攻撃

総当たり攻撃は、複数回ログインを試みることで、ユーザーのパスワードを推測しようとします。Red Hat build of Keycloak には、ブルートフォース検出機能があり、ログインの失敗数が指定のしきい値を超えた場合にユーザーアカウントを一時的に無効にできます。



注記

Red Hat build of Keycloak は、デフォルトでブルートフォース検出を無効にします。この機能を有効にして、総当たり攻撃から保護します。

手順

この保護を有効にするには、以下を実行します。

1. メニューで **Realm Settings** をクリックします。
2. **Security Defenses** タブをクリックします。
3. **Brute Force Detection** タブをクリックします。

総当たり攻撃の検出

The screenshot shows the 'Master' configuration page for 'Brute force detection'. The left sidebar contains navigation options: Master, Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area is titled 'Master' and includes a sub-header 'Brute force detection'. The settings are as follows:

- Enabled: On
- Max login failures: 30
- Permanent lockout: Off
- Wait increment: 1 Minutes
- Max wait: 15 Minutes
- Failure reset time: 12 Hours
- Quick login check milliseconds: 1000
- Minimum quick login wait: 1 Minutes

Buttons for 'Save' and 'Revert' are located at the bottom of the configuration area.

Red Hat build of Keycloak は、攻撃の検出時に永続的なロックアウトおよび一時ロックアウトアクションをデプロイできます。永続的なロックアウトは、管理者が有効にするまでユーザーアカウントを無効にします。一時的なロックアウトは、特定の期間、ユーザーアカウントを無効にします。攻撃が続くと、アカウントが無効になる期間が長くなります。



注記

ユーザーが一時的にロックされ、ログインを試みると、Red Hat build of Keycloak にデフォルトの **Invalid username or password** エラーメッセージが表示されます。このメッセージは、アカウントが無効になっていることに攻撃者が気付かないようにするために、無効なユーザー名または無効なパスワードに対して表示されるメッセージと同じエラーメッセージです。

一般的なパラメーター

Name	説明	デフォルト
Max Login Failures	ログイン失敗の最大数。	30 回失敗
Quick Login Check Milliseconds	ログイン試行の最小時間。	1000 ミリ秒
Minimum Quick Login Wait	ログイン試行が Quick Login Check Milliseconds よりも短い場合にユーザーが無効になる最小時間。	1分

永続的なロックアウトフロー

1. 正常なログイン時
 - a. **count** のリセット
2. ログインの失敗
 - a. **count** のインクリメント
 - b. **count** が **Max Login Failures** を超える数の場合
 - i. ユーザーを完全に無効にする
 - c. それ以外の場合、この障害から最後の障害までの時間が **Quick Login Check Milliseconds** 未満の場合
 - i. **最小クイックログイン待機**のユーザーを一時的に無効にする

Red Hat build of Keycloak がユーザーを無効にすると、管理者がユーザーを有効にするまでユーザーはログインできません。アカウントを有効にすると、**カウント** がリセットされます。

一時的なロックアウトパラメーター

Name	説明	デフォルト
Wait Increment	ユーザーのログイン試行が Max Login Failures を超えると、ユーザーが一時的に無効になる時間に追加された時間。	1分
Max Wait	ユーザーが一時的に無効になっている最大時間。	15分
Failure Reset Time	失敗数がリセットされる時間。最後にログインに失敗したタイミングからタイマーが実行されます。	12時間

一時的なロックアウトアルゴリズム

1. 正常なログイン時
 - a. **count** のリセット
2. ログインの失敗
 - a. この障害から最後の障害までの時間が **Failure Reset Time** よりも長い場合
 - i. **count** のリセット
 - b. **count** のインクリメント
 - c. **Wait Increment * (count / Max Login Failures)** を使用して **wait** を計算します。除算は、整数に丸められる整数除算です。
 - d. **wait** 時間が 0 で、この失敗から最後の失敗までの時間が **Quick Login Check Milli Seconds** よりも小さい場合は、代わりに **wait** を **Minimum Quick Login Wait** に設定してください。

- i. **wait** 秒および **Max Wait** 秒の短い方のユーザーを一時的に無効にします。

一時的に無効にされたアカウントがログインに失敗した場合、**count** は増加しません。

Red Hat build of Keycloak のブルートフォース検出の欠点として、サーバーが DoS 攻撃に対して脆弱になることが挙げられます。攻撃者は、DoS 攻撃を実行する際に知っているアカウントのパスワードを推測してログインを試み、最終的にそのアカウントが Red Hat build of Keycloak によって無効にされる可能性があります。

侵入防止ソフトウェア (IPS) の使用を検討してください。Red Hat build of Keycloak は、ログインの失敗とクライアント IP アドレスの失敗をすべてログに記録します。IPS を、Red Hat build of Keycloak サーバーのログファイルを指すように設定できます。IPS は、ファイアウォールを変更して IP アドレスからの接続をブロックできます。

15.3.1. パスワードポリシー

複雑なパスワードポリシーで、ユーザーが強制的に複雑なパスワードを選択するようにします。詳細については、[パスワードポリシー](#) の章を参照してください。Red Hat build of Keycloak でワンタイムパスワードを使用するように設定することで、パスワードの推測を防ぎます。

15.4. 読み取り専用ユーザー属性

Red Hat build of Keycloak に保存されている一般的なユーザーは、ユーザープロファイルに関連するさまざまな属性を持っています。このような属性には、email、firstname、lastname が含まれます。ただし、ユーザーには通常のプロファイルデータではなくメタデータの属性もあります。通常、ユーザーのメタデータ属性は読み取り専用でなければならず、一般的なユーザーは Red Hat build of Keycloak のユーザーインターフェイスまたは Account REST API からこれらの属性を更新できないはずですが、Admin REST API を使用してユーザーを作成または更新する場合に、管理者は一部の属性を読み取り専用にする必要があります。

メタデータ属性は通常、これらのグループの属性です。

- ユーザーストレージプロバイダーに関連するさまざまなリンクまたはメタデータ。たとえば、LDAP 統合の場合に、**LDAP_ID** 属性には LDAP サーバーのユーザーの ID が含まれます。
- ユーザーストレージによってプロビジョニングされるメタデータ。たとえば、LDAP からプロビジョニングされる **createdTimestamp** は、常にユーザーまたは管理者によって読み取り専用である必要があります。
- さまざまなオーセンティケーターに関連するメタデータ。たとえば、**KERBEROS_PRINCIPAL** 属性には、特定ユーザーの kerberos プリンシパル名を含めることができます。同様に、属性 **usercertificate** には、X.509 証明書からのデータを使用したユーザーのバインディングに関連するメタデータを含めることができます。これは、通常 X.509 証明書認証が有効な場合に使用されます。
- applications/clients によるユーザーの識別に関連するメタデータ。たとえば、**saml.persistent.name.id.my_app** には SAML NameID を含めることができます。これは、クライアントアプリケーション **my_app** がユーザーの識別子として使用されます。
- 認可ポリシーに関連するメタデータ。属性ベースのアクセス制御 (ABAC) に使用されます。これらの属性の値は、認可の決定に使用できます。したがって、ユーザーはこれらの属性を更新できないことが重要です。

長期的な観点から見ると、Red Hat build of Keycloak には適切なユーザープロファイル SPI があり、すべてのユーザー属性を詳細に設定できます。現在、この機能はまだ完全には利用できません。そのため Red Hat build of Keycloak には、ユーザーおよび、サーバーレベルで設定された管理者にとって読み

取り専用のユーザー属性の内部リストがあります。

これは、読み取り専用属性のリストで、Red Hat build of Keycloak のデフォルトプロバイダーや機能により内部で使用されます。そのため、これらは必ず読み取り専用になっています。

- ユーザーの場合:
KERBEROS_PRINCIPAL、**LDAP_ID**、**LDAP_ENTRY_DN**、**CREATED_TIMESTAMP**、**createTimestamp**、**modifyTimestamp**、**userCertificate**、**saml.persistent.name.id.for.***、**ENABLED**、**EMAIL_VERIFIED**
- 管理者の場合:
KERBEROS_PRINCIPAL、**LDAP_ID**、**LDAP_ENTRY_DN**、**CREATED_TIMESTAMP**、**createTimestamp**、**modifyTimestamp**

システム管理者は、このリストに属性を追加することもできます。現在、設定はサーバーレベルで利用できます。

この設定は、**spi-user-profile-declarative-user-profile-read-only-attributes** オプションと **spi-user-profile-declarative-user-profile-admin-read-only-attributes** オプションを使用して追加できます。以下に例を示します。

```
kc.[sh|bat] start --spi-user-profile-declarative-user-profile-read-only-attributes=foo,bar*
```

この例では、ユーザーおよび管理者は **foo** 属性を更新できません。ユーザーは、**bar** で始まる属性を編集できません。たとえば、**bar** や **barrier** などです。設定では大文字と小文字が区別されないため、この例では **FOO** や **BarRier** などの属性も拒否されます。ワイルドカード文字 ***** は属性名の末尾でのみサポートされるので、管理者は指定された文字で始まるすべての属性を効果的に拒否できます。属性の途中の ***** は通常の文字と見なされます。

15.5. クリックジャッキング

クリックジャッキングは、ユーザーが希望するユーザーとは異なるユーザーインターフェイス要素をクリックする手法です。悪意のあるサイトでは、対象のサイトにある重要なボタンのすぐ下にダミーのボタンが配置された、透明な iFrame に対象のサイトを読み込みます。ユーザーが表示されているボタンをクリックすると、隠されたページのボタンをクリックされます。攻撃者は、この方法を使用して、ユーザーの認証認証情報を盗み、そのリソースにアクセスする可能性があります。

デフォルトでは、Red Hat build of Keycloak によるすべての応答に、その発生を防止できる特定の HTTP ヘッダーが設定されています。具体的には、[X-Frame-Options](#) および [Content-Security-Policy](#) を設定します。制御できる詳細なブラウザーアクセスが多数あるため、これらのヘッダーの両方の定義を確認する必要があります。

手順

管理コンソールでは、X-Frame-Options ヘッダーおよび Content-Security-Policy ヘッダーの値を指定できます。

1. **Realm Settings** メニュー項目をクリックします。
2. **Security Defenses** タブをクリックします。

セキュリティー保護

The screenshot shows the Keycloak Master console interface. On the left is a dark sidebar with navigation options: Master, Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main area is titled 'Master' and contains a sub-header 'Realm settings are settings that control the options for users, applications, roles, and groups in the current realm.' Below this are tabs for General, Login, Email, Themes, Keys, Events, Localization, Security defenses, and Sessi. The 'Security defenses' tab is active, showing sub-tabs for Headers and Brute force detection. The 'Headers' sub-tab is selected, displaying a list of security headers with their values in input fields: X-Frame-Options (SAMEORIGIN), Content-Security-Policy (frame-src 'self'; frame-ancestors 'self'; object-src 'none;'), Content-Security-Policy-Report-Only, X-Content-Type-Options (nosniff), X-Robots-Tag (none), X-XSS-Protection (1; mode=block), and HTTP Strict Transport Security (HSTS) (max-age=31536000; includeSubDomains). At the bottom right are 'Save' and 'Revert' buttons.

デフォルトで、Red Hat build of Keycloak は iframe に対して **same-origin** ポリシーのみを設定します。

15.6. SSL/HTTPS 要件

OAuth 2.0/OpenID Connect はセキュリティーにアクセストークンを使用します。攻撃者はネットワークをスキャンしてアクセストークンを探し、トークンが許可されている悪意のある操作を実行するためにそれらを使用する可能性があります。この攻撃は中間者攻撃と呼ばれます。Red Hat build of Keycloak 認証サーバーと Red Hat build of Keycloak が保護するクライアント間の通信では、中間者攻撃を防ぐために SSL/HTTPS を使用します。

Red Hat build of Keycloak には、[SSL/HTTPS 用のモードが3つ](#) あります。SSL は設定が複雑なため、Red Hat build of Keycloak は localhost、192.168x.x などのプライベート IP アドレスを介した HTTPS 以外の通信を許可します。実稼働環境では、SSL が有効で、全操作に対して SSL が必須であることを確認します。

アダプター/クライアント側で、SSL トラストマネージャーを無効にできます。トラストマネージャーは、Red Hat build of Keycloak が通信するクライアントのアイデンティティーが有効であることを確認します。また、サーバーの証明書に対して DNS ドメイン名を確認します。実稼働環境では、各クライアントアダプターがトラストストアを使用して DNS の中間者攻撃を防いでいることを確認します。

15.7. CSRF 攻撃

クロスサイトリクエストフォージェリー (CSRF) 攻撃は、Web サイトがすでに認証されているユーザーからの HTTP 要求を使用します。cookie ベースの認証を使用するサイトはすべて CSRF 攻撃に対して脆弱です。これらの攻撃は、投稿されたフォームまたはクエリーパラメーターに対して状態クッキーを照合することで軽減されます。

OAuth 2.0 ログイン仕様では、状態 cookie を使用し、伝送された状態 state パラメーターと照合する必要があります。Red Hat build of Keycloak は仕様のこの部分を完全に実装しているため、すべてのログインが保護されます。

Red Hat build of Keycloak 管理コンソールは、バックエンドの Red Hat build of Keycloak 管理 REST API への REST 呼び出しを行う JavaScript/HTML5 アプリケーションです。これらの呼び出しにはすべての Bearer トークン認証が必要であり、JavaScript Hadoop 呼び出しで設定されているため、CSRF は不可能です。管理 REST API を、CORS のオリジンを検証するよう設定できます。

Red Hat build of Keycloak のユーザーアカウント管理セクションは、CSRF に対して脆弱になる可能性があります。CSRF 攻撃を防ぐために、Red Hat build of Keycloak は状態クッキーを設定し、このクッキーの値を、アクションリンク内の非表示フォームフィールドまたはクエリーパラメーターに組み込みます。Red Hat build of Keycloak は、クエリー/フォームパラメーターを状態 cookie に対してチェックし、ユーザーが呼び出しを行ったことを確認します。

15.8. 特定のリダイレクト URI

登録済みのリダイレクト URL をできるだけ具体的なものにします。[Authorization Code Flows](#) にあまいリダイレクト URI を登録すると、悪意のあるクライアントがより広範なアクセス権のある別のクライアントになりすますことができます。たとえば、2つのクライアントが同じドメインに存在すると、なりすましが発生する可能性があります。

15.9. FAPI コンプライアンス

クライアントがセキュアで FAPI に準拠していることを Red Hat build of Keycloak サーバーが検証するために、FAPI サポート用のクライアントポリシーを設定できます。詳細は、[アプリケーションおよびサービスの保護](#) の FAPI セクションを参照してください。特に、これにより、クライアントに必要な SSL、使用される安全なリダイレクト URI、その他の同様のベストプラクティスなど、上記のセキュリティのベストプラクティスが保証されます。

15.10. 不正アクセスおよびトークンの更新

Red Hat build of Keycloak には、悪意のあるアクターがアクセストークンを盗んだりトークンを更新したりするのを防ぐためのアクションが複数含まれています。特に重要なのは、Red Hat build of Keycloak とそのクライアントおよびアプリケーションの間で SSL/HTTPS 通信を強制するアクションです。Red Hat build of Keycloak では、デフォルトで SSL が有効になっていません。

アクセストークンの漏洩からの損失を減らすためのもう1つのアクションとして、トークンの有効期限を短くすることが挙げられます。[タイムアウトページ](#) 内でトークンの有効期間を指定できます。アクセストークンの有効期限を短くして、クライアントとサーバーは短期間にアクセストークンを強制的に更新します。管理者がリークを検出すると、すべてのユーザーサービスセッションをログアウトして、これらの更新トークンを無効にするか、取り消しポリシーを設定できます。

更新トークンは常にクライアントには非公開のまま、送信されることはありません。

これらのトークンをキー所有者トークンとして発行することにより、リークされたアクセストークンによる損害を軽減し、トークンを更新できます。詳細については、[OAuth 2.0 相互 TLS クライアント証明書バインドアクセストークン](#) を参照してください。

アクセストークンまたは更新トークンが漏洩された場合は、管理コンソールにアクセスし、失効前にポリシーをすべてのアプリケーションにプッシュします。not-before ポリシーをプッシュすると、その時間前に発行されたトークンが無効になります。新しい not-before ポリシーをプッシュすることで、アプリケーションは Red Hat build of Keycloak から新しい公開鍵をダウンロードして、漏洩したレلم署名鍵による損害を軽減しなければならなくなります。詳細は、[キーの章](#) を参照してください。

特定のアプリケーション、クライアント、またはユーザーが危険にさらされる場合は、無効にできません。

15.11. 侵害された認可コード

[OIDC 認証コードフロー](#) では、Red Hat build of Keycloak は承認コードに強力な暗号化を使用してランダムな値を生成します。認可コードは、アクセストークンを取得するために一度だけ使用されます。

管理コンソールの [タイムアウトページ](#) で、認可コードの有効期間を指定できます。時間の長さが 10 秒未満であることを確認します。これは、クライアントがコードからトークンを要求するのに十分な時間です。

また、[Proof Key for Code Exchange \(PKCE\)](#) をクライアントに適用することで、認可コードの漏洩を防ぐこともできます。

15.12. オープンリダイレクター

オープンリダイレクターは、検証を行わずにユーザーエージェントをパラメーター値で指定された場所に自動的にリダイレクトするパラメーターを使用するエンドポイントです。攻撃者は、エンドユーザーの認可エンドポイントとリダイレクト URI パラメーターを使用し、認可サーバーのユーザーの信頼を使用してフィッシング攻撃を起動して、認可サーバーをオープンディレクトリーとして使用できます。

Red Hat build of Keycloak では、登録されたすべてのアプリケーションとクライアントで少なくとも 1 つのリダイレクト URI パターンを登録する必要があります。クライアントが Red Hat build of Keycloak にリダイレクトを実行するようリクエストすると、Red Hat build of Keycloak はリダイレクト URI を有効な登録済み URI パターンのリストと照合してチェックします。クライアントとアプリケーションは、オープンリダイレクタ攻撃を軽減するために、可能な限り特定の URI パターンを登録する必要があります。

アプリケーションで http (s) 以外のカスタムスキームが必要な場合は、それを検証パターンの明示的な部分にする必要があります (例: **Custom:/app/***)。セキュリティ上の理由から、* のような一般的なパターンは http (s) 以外のスキームをカバーしません。

15.13. パスワードデータベースの漏洩

Red Hat build of Keycloak は、PBKDF2 ハッシュアルゴリズムを使用して、パスワードを raw テキストとして保存する代わりに、ハッシュ化されたテキストとして保存します。Red Hat build of Keycloak は、セキュリティコミュニティが推奨する反復回数である 27,500 回のハッシュ反復を実行します。このハッシュ反復回数は、PBKDF2 ハッシュ処理が大量の CPU リソースを使用するため、パフォーマンスに悪影響を及ぼす可能性があります。

15.14. 制限の範囲

デフォルトでは、新規クライアントアプリケーションには、無制限の [ロールスコープマッピング](#) があります。そのクライアントのすべてのアクセストークンには、ユーザーが所有するすべてのパーミッションが含まれます。攻撃者がクライアントに不正アクセスして、クライアントのアクセストークンを取得すると、ユーザーがアクセスできる各システムが危険にさらされます。

各クライアントの [Scope メニュー](#) を使用して、アクセストークンのロールを制限します。または、クライアントスコープレベルでロールスコープマッピングを設定し、[クライアントスコープメニュー](#) を使用して、クライアントスコープをクライアントに割り当てることもできます。

15.15. トークンオーディエンスの制限

サービス間で信頼レベルが低い環境では、トークン対象者を制限します。詳細は、[OAuth2 Threat Model](#) および [Audience Support](#) セクションを参照してください。

15.16. 認証セッションの制限

Web ブラウザーで初めてログインページを開くと、Red Hat build of Keycloak は、認証セッションと呼ばれるオブジェクトを作成し、そこに要求に関する有用な情報を保存します。同じブラウザの別のタブから新しいログインページが開かれるたびに、Red Hat build of Keycloak は、認証セッション内に保存される認証サブセッションと呼ばれる新しいレコードを作成します。認証要求は、管理 CLI などの任意のタイプのクライアントから取得できます。この場合、認証サブセッションが1つ含まれる、新しい認証セッションが作成されます。認証セッションは、ブラウザーフローを使用する以外の方法でも作成できます。以下のテキストは、ソースフローに関係なく適用されます。



注記

このセクションでは、認証セッションに Data Grid プロバイダーを使用するデプロイメントについて説明します。

認証セッションは、**rootAuthenticationSessionEntity** として内部に保存されます。各 **RootAuthenticationSessionEntity** は、**AuthenticationSessionEntity** オブジェクトのコレクションとして **RootAuthenticationSessionEntity** 内に保存された複数の認証サブセッションを含めることができます。Red Hat build of Keycloak は、認証セッションを専用の Data Grid キャッシュに保存します。**RootAuthenticationSessionEntity** ごとの **AuthenticationSessionEntity** の数は、各キャッシュエントリーのサイズに影響します。認証セッションキャッシュの合計メモリーフットプリントは、保存された **RootAuthenticationSessionEntity** の数と各 **RootAuthenticationSessionEntity** 内の **AuthenticationSessionEntity** の数によって決まります。

維持される **RootAuthenticationSessionEntity** オブジェクトの数は、ブラウザーから未完了のログインフローの数に対応します。**RootAuthenticationSessionEntity** 数のコントロールを保持するには、高度なファイアウォール制御を使用して Ingress ネットワークトラフィックを制限することを推奨します。

AuthenticationSessionEntity が多数含まれる、アクティブな **RootAuthenticationSessionEntity** が多いデプロイメントでは、メモリー使用量が高くなる場合があります。ロードバランサーがセッションのスティックネスをサポートしていないか、それ用に設定されていない場合には、クラスターのネットワーク上の負荷が増大する可能性があります。この負荷の理由は、適切な認証セッションを所有していないノードに到達する各要求は、所有者ノードの認証セッションレコードを取得および更新する必要があります。これには、取得と保存の両方に個別のネットワーク送信が含まれます。

RootAuthenticationSessionEntity ごとの **AuthenticationSessionEntity** の最大数は、プロパティー **authSessionsLimit** を指定して、**authenticationSessionsSPI** で設定できます。デフォルト値は、**RootAuthenticationSessionEntity** ごとの **300AuthenticationSessionEntity** に設定されています。この制限に達すると、新しい認証セッション要求後に、最も古い認証サブセッションが削除されます。

次の例は、**RootAuthenticationSessionEntity** ごとのアクティブな **AuthenticationSessionEntity** の数を 100 に制限する方法を示しています。

```
bin/kc.[sh|bat] start --spi-authentication-sessions-infinispan-auth-sessions-limit=100
```

15.17. SQL インジェクション攻撃

現在、Red Hat build of Keycloak に既知の SQL インジェクションの脆弱性はありません。

第16章 アカウントコンソール

Red Hat build of Keycloak ユーザーは、アカウントコンソールを通じてアカウントを管理できます。ユーザーはプロフィールの設定、2要素認証の追加、アイデンティティプロバイダーのアカウントの追加、デバイスのアクティビティの設定が可能です。

関連情報

- アカウントコンソールは、外観および言語の設定により設定できます。例として、**Personal info** リンクをクリックして詳細を入力して保存することにより、**Personal info** ページに属性を追加します。詳細は、[サーバー開発者ガイド](#)を参照してください。

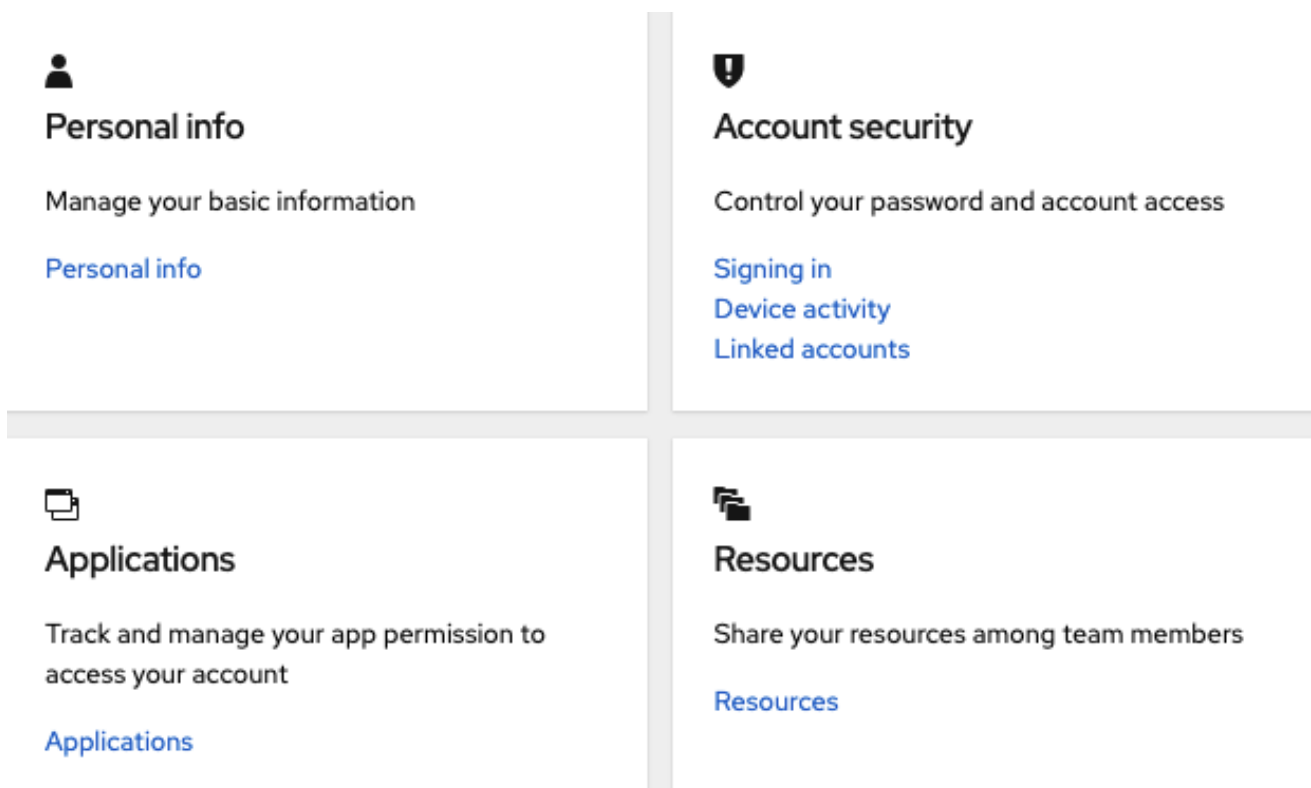
16.1. アカウントコンソールへのアクセス

すべてのユーザーがアカウントコンソールにアクセスできます。

手順

1. アカウントが存在する Red Hat build of Keycloak サーバーのレルム名と IP アドレスを書き留めておきます。
2. Web ブラウザーに、`server-root/realms/{realm-name}/account` の形式で URL を入力します。
3. ログイン名とパスワードを入力します。

アカウントコンソール



16.2. サインイン方法の設定

Basic 認証 (ログイン名とパスワード) または 2 要素認証を使用して、このコンソールにログインします。2 要素認証では、以下のいずれかの手順を使用します。

16.2.1. OTP を使用した二要素認証

前提条件

- OTP がレルムの有効な認証メカニズムである。

手順

1. メニューで **Account security** をクリックします。
2. **Signing in** をクリックします。
3. **Set up authenticator application** をクリックします。

サインイン

Signing in

Configure ways to sign in.

Basic authentication

Password

Sign in by entering your password.

My password

Created May 3, 2022,
11:56 AM

Update

Two-factor authentication

Authenticator application



Enter a verification code from authenticator application.

Authenticator application is not set up.

4. 画面の指示に従い、モバイルデバイスで **FreeOTP** または **Google Authenticator** のいずれかを OTP ジェネレーターとして使用します。
5. スクリーンショットの QR コードを、モバイルデバイスの OTP ジェネレーターにスキャンします。
6. ログアウトして再度ログインします。
7. モバイルデバイスで提供されている OTP を入力して、プロンプトに応答します。

16.2.2. WebAuthn を使用した二要素認証

前提条件

- WebAuthn は、レルムの有効な 2 要素認証メカニズムです。詳細は、[WebAuthn](#) セクションを参照してください。

手順

1. メニューで **Account Security** をクリックします。
2. **Signing In** をクリックします。
3. **Set up Security Key** をクリックします。

サインイン

Basic Authentication

Password

Log in by entering your password.

My Password	Created: August 19, 2021, 11:26 AM	Update
-------------	---------------------------------------	------------------------

Two-Factor Authentication

Authenticator Application

Enter a verification code from authenticator application.

[Set up Authenticator Application](#)

Authenticator Application is not set up.

Security Key

Use your security key to log in.

[Set up Security Key](#)

Security Key is not set up.

4. WebAuthn セキュリティーキーを準備します。このキーの準備方法は、使用する WebAuthn セキュリティーキーのタイプによって異なります。たとえば、USB ベースの Yubikey では、キーをラップトップの USB ポートに挿入する必要があります。
5. **登録** をクリックしてセキュリティーキーを登録します。
6. ログアウトして再度ログインします。
7. 認証フローが正しく設定されていると仮定すると、2 番目の要素としてセキュリティーキーを使用して認証するように求めるメッセージが表示されます。

16.2.3. WebAuthn を使用したパスワードレス認証

前提条件

- WebAuthn は、レルムの有効なパスワードレス認証メカニズムです。詳細については、[パスワードレス WebAuthn](#) セクションに従ってください。

手順

1. メニューで **Account Security** をクリックします。
2. **Signing In** をクリックします。
3. **Passwordless** セクションで **Set up Security Key** をクリックします。

サインイン

Basic Authentication

Password

Log in by entering your password.

My Password	Created: August 19, 2021, 11:26 AM	Update
-------------	---	------------------------

Two-Factor Authentication

Authenticator Application [Set up Authenticator Application](#)

Enter a verification code from authenticator application.

Authenticator Application is not set up.

Passwordless

Security Key [Set up Security Key](#)

Use your security key for passwordless log in.

Security Key is not set up.

4. WebAuthn セキュリティーキーを準備します。このキーの準備方法は、使用する WebAuthn セキュリティーキーのタイプによって異なります。たとえば、USB ベースの Yubikey では、キーをラップトップの USB ポートに挿入する必要があります。
5. **登録** をクリックしてセキュリティーキーを登録します。
6. ログアウトして再度ログインします。
7. 認証フローが正しく設定されていると仮定すると、2 番目の要素としてセキュリティーキーを使用して認証するように求めるメッセージが表示されます。ログインでパスワードを指定する必要がなくなりました。

16.3. デバイスのアクティビティーの表示

アカウントにログインしているデバイスを表示できます。

手順

1. メニューで **Account security** をクリックします。
2. **デバイスアクティビティー** をクリックします。
3. 疑わしいと思われる場合は、デバイスからログアウトします。


Devices

Device activity

Sign out of any unfamiliar devices.

Signed in devices

 Refresh

 Mac OS X 10.15.7 / Chrome/101.0.4951

Current session

IP address	Last accessed	Clients	Started	Expires
127.0.0.1	June 1, 2022, 11:36 AM	security-admin-console-v2, Account, Account Console	June 1, 2022, 10:54 AM	June 1, 2022, 8:54 PM

16.4. アイデンティティプロバイダーアカウントの追加

アカウントを [アイデンティティブローカー](#) にリンクできます。このオプションは、ソーシャルプロバイダーアカウントをリンクする時によく使用されます。

手順

1. 管理コンソールにログインします。
2. メニューで **Identity providers** をクリックします。
3. プロバイダーを選択し、フィールドに入力します。
4. アカウントコンソールに戻ります。
5. メニューで **Account security** をクリックします。
6. **Linked accounts** をクリックします。

追加した ID プロバイダーがこのページに表示されます。

リンクされたアカウント

Linked accounts

Manage logins through third-party accounts.

Linked login providers

No linked providers

Unlinked login providers



GitHub

Social login

[Link account](#)

16.5. 他のアプリケーションへのアクセス

アプリケーションメニュー項目には、アクセスできるアプリケーションがユーザーに表示されます。この場合には、アカウントコンソールのみを使用できます。

アプリケーション

Applications

Manage your application permissions.

	Name	Application type	Status
>	Security-admin-console-v2	Internal	In use
>	Account	Internal	In use
>	Account Console	Internal	In use

16.6. グループメンバーシップの表示

Groupsメニューをクリックすると、関連付けられているグループを表示できます。Direct membership チェックボックスを選択すると、直接関連付けられているグループのみが表示されます。

前提条件

- Groupsメニューを表示するには、view-groups アカウントロールが必要です。

グループのメンバーシップを表示する

Groups		
<input type="checkbox"/> direct membership		
Name	path	direct membership
gov	/parent/gov	<input checked="" type="checkbox"/>
test	/test	<input checked="" type="checkbox"/>
parent	/parent	<input type="checkbox"/>

第17章 管理 CLI

Red Hat build of Keycloak では、管理 CLI コマンドラインツールを使用して、コマンドラインインターフェイス (CLI) から管理タスクを実行できます。

17.1. 管理 CLI のインストール

Red Hat build of Keycloak は、**bin** ディレクトリーの実行スクリプトを使用して、管理 CLI サーバーのディストリビューションをパッケージ化します。

Linux スクリプトは **kcadm.sh** と呼ばれ、Windows のスクリプトは **kcadm.bat** と呼ばれます。Red Hat build of Keycloak サーバーディレクトリーを **PATH** に追加し、ファイルシステム上の任意の場所からクライアントを使用できます。

以下に例を示します。

- Linux:

```
$ export PATH=$PATH:$KEYCLOAK_HOME/bin
$ kcadm.sh
```

- Windows:

```
c:\> set PATH=%PATH%;%KEYCLOAK_HOME%\bin
c:\> kcadm
```



注記

Red Hat build of Keycloak Server ディストリビューションを展開したパスに、**KEYCLOAK_HOME** 環境変数を設定する必要があります。

繰り返し実行しないように、本書の残りの部分では、CLI の相違点が **kcadm** コマンド名ではなく、Windows の例のみが説明されています。

17.2. 管理 CLI の使用

管理 CLI は、管理 REST エンドポイントに HTTP 要求を行います。Admin REST エンドポイントにアクセスするには認証が必要です。



注記

特定のエンドポイントの JSON 属性に関する詳細は、Admin REST API のドキュメントを参照してください。

1. ログインして認証済みセッションを開始します。これで、作成、読み取り、更新、および削除 (CRUD) 操作を実行できるようになりました。以下に例を示します。

- Linux:

```
$ kcadm.sh config credentials --server http://localhost:8080 --realm demo --user admin --client admin
$ kcadm.sh create realms -s realm=demorealms -s enabled=true -o
```



```
$ CID=$(kcadm.sh create clients -r demorealm -s clientId=my_client -s 'redirectUri=
["http://localhost:8980/myapp/*"]' -i)
$ kcadm.sh get clients/$CID/installation/providers/keycloak-oidc-keycloak-json
```

- Windows:

```
c:\> kcadm config credentials --server http://localhost:8080 --realm demo --user admin --
client admin
c:\> kcadm create realms -s realm=demorealm -s enabled=true -o
c:\> kcadm create clients -r demorealm -s clientId=my_client -s "redirectUri=
["http://localhost:8980/myapp/*\"]" -i > clientid.txt
c:\> set /p CID=<clientid.txt
c:\> kcadm get clients/%CID%/installation/providers/keycloak-oidc-keycloak-json
```

2. 実稼働環境では、トークンの公開を避けるために、**https:** を使用して Red Hat build of Keycloak にアクセスします。信頼できる認証局が Java のデフォルトの証明書トラストストアに含まれており、サーバーの証明書を発行していない場合は **truststore.jks** ファイルを準備して使用するよう管理 CLI に指示します。

以下に例を示します。

- Linux:

```
$ kcadm.sh config truststore --trustpass $PASSWORD ~/.keycloak/truststore.jks
```

- Windows:

```
c:\> kcadm config truststore --trustpass %PASSWORD%
%HOMEPATH%\keycloak\truststore.jks
```

17.3. 認証

管理 CLI でログインする場合に、以下を指定します。

- サーバーエンドポイント URL
- レルム
- ユーザー名

別のオプションとして、clientId のみを指定して、使用する固有のサービスアカウントを作成します。

ユーザー名を使用してログインする場合は、指定したユーザーのパスワードを使用します。clientId を使用してログインする場合、ユーザーパスワードではなくクライアントシークレットのみが必要になります。クライアントシークレットの代わりに **Signed JWT** を使用することもできます。

セッションに使用されるアカウントに、Admin REST API 操作を呼び出すための適切なパーミッションがあることを確認してください。たとえば、**realm-management** クライアントの **realm-admin** ロールは、ユーザーのレルムを管理できます。

認証には2つの主なメカニズムを使用できます。1つのメカニズムは **kcadm config credentials** を使用して認証セッションを開始します。

```
$ kcadm.sh config credentials --server http://localhost:8080 --realm master --user admin --password
admin
```

この仕組みでは、取得したアクセストークンと関連する更新トークンを保存することで、**kcadm** コマンド呼び出し間の認証セッションを維持します。プライベート設定ファイルで他のシークレットを保持することができます。詳細については、[次の章](#)を参照してください。

2つ目のメカニズムは、呼び出し中に各コマンド呼び出しを認証します。このメカニズムは、サーバーの負荷と、ラウンドトリップに費やされた時間を増やします。このアプローチの利点は、呼び出し間でトークンを保存する必要がないので、ディスクには何も保存されない点です。Red Hat build of Keycloak は、**--no-config** 引数が指定されている場合にこのモードを使用します。

たとえば、操作の実行時に、認証に必要な情報をすべて指定します。

```
$ kcadm.sh get realms --no-config --server http://localhost:8080 --realm master --user admin --password admin
```

管理 CLI の使用に関する詳細情報は、**kcadm.sh help** コマンドを実行します。

認証セッションの開始に関する詳細は、**kcadm.sh config credentials --help** コマンドを実行します。

17.4. 代替設定の使用

デフォルトでは、管理 CLI は **kcadm.config** という名前の設定ファイルを維持します。Red Hat build of Keycloak は、このファイルをユーザーのホームディレクトリーに配置します。Linux ベースのシステムでは、完全パス名は **\$HOME/.keycloak/kcadm.config** になります。Windows では、完全パス名は **%HOMEPATH%\keycloak\kcadm.config** になります。

--config オプションを使用して、異なるファイルまたは場所を指定して、複数の認証セッションを並行して管理することができます。



注記

単一のスレッドから単一の設定ファイルに関連付けられる操作を実行します。

設定ファイルがシステム上の他のユーザーに表示されないようにします。これには、非公開にする必要のあるアクセストークンおよびシークレットが含まれます。Red Hat build of Keycloak は、適切なアクセス制限を設定して **~/.keycloak** ディレクトリーとその内容を自動的に作成します。このディレクトリーがすでに存在する場合、Red Hat build of Keycloak はディレクトリーの権限を更新しません。

設定ファイル内にシークレットを保存することを回避することは可能ですが、これは不便であり、トークン要求の数が増えます。すべてのコマンドで **--no-config** オプションを使用して、**kcadm** の呼び出しごとに **config credentials** コマンドが必要とする認証情報を指定します。

17.5. 基本操作およびリソース URI

管理 CLI は、特定のタスクを単純化する追加のコマンドを使用して、管理 REST API エンドポイントに対して CRUD 操作を汎用的に実行できます。

主な使用方法パターンを以下に示します。

```
$ kcadm.sh create ENDPOINT [ARGUMENTS]
$ kcadm.sh get ENDPOINT [ARGUMENTS]
$ kcadm.sh update ENDPOINT [ARGUMENTS]
$ kcadm.sh delete ENDPOINT [ARGUMENTS]
```

create、**get**、**update**、および **delete** コマンドは HTTP 動詞である **POST**、**GET**、**PUT**、**DELETE** に

それぞれマップします。ENDPOINT はターゲットリソース URI であり、絶対 URI (**http:** または **https:** で始まる) または相対 URI にすることができます。Red Hat build of Keycloak は、次の形式で絶対 URL を設定するためにこれを使用します。

```
SERVER_URI/admin/realms/REALM/ENDPOINT
```

たとえば、サーバー <http://localhost:8080> に対して認証し、レルムが **master** である場合は、**users** を ENDPOINT として使用すると <http://localhost:8080/admin/realms/master/users> のリソース URL となります。

ENDPOINT を **clients** に設定する場合、有効なリソース URI は <http://localhost:8080/admin/realms/master/clients> になります。

Red Hat build of Keycloak には、レルムのコンテナである **realms** エンドポイントがあります。以下に対して解決します。

```
SERVER_URI/admin/realms
```

Red Hat build of Keycloak には、**serverinfo** エンドポイントがあります。このエンドポイントはレルムとは独立しています。

realm-admin 権限を持つユーザーとして認証する場合は、複数のレルムでコマンドを実行する必要がある場合があります。その場合は、**-r** オプションを指定して、コマンドを明示的に実行するレルムを CLI に指示します。**kcadm.sh config credentials** の **--realm** オプションで指定された **REALM** を使用する代わりに、コマンドは **TARGET_REALM** を使用します。

```
SERVER_URI/admin/realms/TARGET_REALM/ENDPOINT
```

以下に例を示します。

```
$ kcadm.sh config credentials --server http://localhost:8080 --realm master --user admin --password admin
$ kcadm.sh create users -s username=testuser -s enabled=true -r demorealm
```

この例では、**master** レルムで **admin** ユーザーとして認証されたセッションを開始します。次に、リソース URL <http://localhost:8080/admin/realms/demorealm/users> に対して POST 呼び出しを実行します。

create および **update** コマンドは、JSON ボディをサーバーに送信します。**-f FILENAME** を使用して、ファイルから既製のドキュメントを読み取ることができます。**-f** オプションを使用できる場合、Red Hat build of Keycloak は標準入力からメッセージボディを読み取ります。**ユーザーの作成** の例にあるように、個別の属性とその値を指定できます。Red Hat build of Keycloak は、属性を JSON ボディにしてサーバーに送信します。

Red Hat build of Keycloak では、**update** コマンドを使用してリソースを更新するための方法がいくつかあります。まず、リソースの現在の状態を決定してファイルに保存し、そのファイルを編集して更新のためにサーバーに送信できます。

以下に例を示します。

```
$ kcadm.sh get realms/demorealm > demorealm.json
$ vi demorealm.json
$ kcadm.sh update realms/demorealm -f demorealm.json
```

このメソッドは、送信された JSON ドキュメントの属性でサーバーのリソースを更新します。

別のオプションとして、**-s, --set** オプションを使用してオンザフライで更新を実行し、新しい値を設定することもできます。

以下に例を示します。

```
$ kcadm.sh update realms/demorealm -s enabled=false
```

このメソッドは、**enabled** 属性を **false** に設定します。

デフォルトでは、**update** コマンドは **get** を実行し、新しい属性値を既存の値とマージします。エンドポイントが **PUT** コマンドをサポートするかもしれませんが、**GET** コマンドではない場合もあります。**-n** オプションを使用して **no-merge** 更新を実行できます。これは、**GET** コマンドを最初に実行せずに **PUT** コマンドを実行します。

17.6. レルム操作

新しいレルムの作成

realms エンドポイントで **create** コマンドを使用して、新しい有効なレルムを作成します。属性を **realm** に設定し、**enabled** に設定します。

```
$ kcadm.sh create realms -s realm=demorealm -s enabled=true
```

Red Hat build of Keycloak では、デフォルトでレルムが無効になっています。これを有効にすると、認証にレルムをすぐに使用できます。

新規オブジェクトの説明には、JSON 形式を使用することもできます。

```
$ kcadm.sh create realms -f demorealm.json
```

レルム属性を使用して JSON ドキュメントをファイルから直接送信するか、標準入力にドキュメントをパイプして送信できます。

以下に例を示します。

- Linux:

```
$ kcadm.sh create realms -f - << EOF
{ "realm": "demorealm", "enabled": true }
EOF
```

- Windows:

```
c:\> echo { "realm": "demorealm", "enabled": true } | kcadm create realms -f -
```

既存のレルムのリスト表示

このコマンドは、すべてのレルムのリストを返します。

```
$ kcadm.sh get realms
```



注記

Red Hat build of Keycloak は、サーバー上のレルムのリストをフィルタリングして、ユーザーのみが表示できるレルムを返します。

ほとんどのユーザーは、レルム名やレルムの有効化ステータスなどの属性のサブセットに関心があるので、すべてのレルム属性のリストは詳細にすることができます。--fields オプションを使用して、返す属性を指定できます。

```
$ kcadm.sh get realms --fields realm,enabled
```

結果をコンマ区切りの値として表示することができます。

```
$ kcadm.sh get realms --fields realm --format csv --noquotes
```

特定のレルムの取得

レルム名をコレクション URI に追加し、個別のレルムを取得します。

```
$ kcadm.sh get realms/master
```

レルムの更新

- レルムの属性すべてを変更しない場合に、-s オプションを使用して属性に新しい値を設定します。
以下に例を示します。

```
$ kcadm.sh update realms/demorealm -s enabled=false
```

- 書き込み可能な属性をすべて新しい値に設定する場合は、以下のようになります。
 - get コマンドを実行します。
 - JSON ファイルの現在の値を編集します。
 - 再度送信します。
以下に例を示します。

```
$ kcadm.sh get realms/demorealm > demorealm.json
$ vi demorealm.json
$ kcadm.sh update realms/demorealm -f demorealm.json
```

レルムの削除

以下のコマンドを実行してレルムを削除します。

```
$ kcadm.sh delete realms/demorealm
```

レルムのすべてのログインページオプションをオンにする

特定のレイバビリティを制御する属性を true に設定します。

以下に例を示します。

```
$ kcadm.sh update realms/demorealm -s registrationAllowed=true -s
registrationEmailAsUsername=true -s rememberMe=true -s verifyEmail=true -s
resetPasswordAllowed=true -s editUsernameAllowed=true
```

レルムキーのリスト表示

ターゲットレルムの **keys** エンドポイントで **get** 操作を使用します。

```
$ kcadm.sh get keys -r demorealm
```

新しいレルムキーの生成

1. 新しい RSA 生成鍵のペアを追加する前に、ターゲットレルムの ID を取得します。以下に例を示します。

```
$ kcadm.sh get realms/demorealm --fields id --format csv --noquotes
```

2. **kcadm.sh get keys -r demorealm** によって明らかになったように、既存のプロバイダーよりも優先度の高い新規キープロバイダーを追加します。以下に例を示します。

- Linux:

```
$ kcadm.sh create components -r demorealm -s name=rsa-generated -s providerId=rsa-generated -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s 'config.priority=["101"]' -s 'config.enabled=["true"]' -s 'config.active=["true"]' -s 'config.keySize=["2048"]'
```

- Windows:

```
c:\> kcadm create components -r demorealm -s name=rsa-generated -s providerId=rsa-generated -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s "config.priority=["101"]" -s "config.enabled=["true"]" -s "config.active=["true"]" -s "config.keySize=["2048"]"
```

3. **parentId** 属性をターゲットレルムの ID の値に設定します。
kcadm.sh get keys -r demorealm で示されるように、新しく追加されたキーがアクティブなキーになります。

Java キーストアファイルから新しいレルムキーの追加

1. 新しいキープロバイダーを追加して、JKS ファイルとして新しいキーペアを事前に追加します。たとえば、以下のようになります。

- Linux:

```
$ kcadm.sh create components -r demorealm -s name=java-keystore -s providerId=java-keystore -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s 'config.priority=["101"]' -s 'config.enabled=["true"]' -s 'config.active=["true"]' -s 'config.keystore=["/opt/keycloak/keystore.jks"]' -s 'config.keystorePassword=["secret"]' -s 'config.keyPassword=["secret"]' -s 'config.keyAlias=["localhost"]'
```

- Windows:

```
c:\> kcadm create components -r demorealm -s name=java-keystore -s providerId=java-keystore -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s "config.priority=["101"]" -s "config.enabled=["true"]" -s "config.active=["true"]" -s "config.keystore=["/opt/keycloak/keystore.jks"]" -s "config.keystorePassword=["secret"]" -s "config.keyPassword=["secret"]" -s "config.keyAlias=["localhost"]"
```

2. 特定のキーストアに一致するように、**keystore**、**keystorePassword**、**keyPassword**、および **alias** 属性値を変更するようにしてください。
3. **parentId** 属性をターゲットレルムの ID の値に設定します。

鍵のパッシブの作成または鍵の無効化

1. パッシブを作成する鍵を特定します。

```
$ kcadm.sh get keys -r demorealm
```

2. キーの **providerId** 属性を使用して、**components/PROVIDER_ID** などのエンドポイント URI を作成します。
3. **update** を実行します。
以下に例を示します。

- Linux:

```
$ kcadm.sh update components/PROVIDER_ID -r demorealm -s 'config.active=["false"]'
```

- Windows:

```
c:\> kcadm update components/PROVIDER_ID -r demorealm -s "config.active=["false"]"
```

他のキー属性を更新できます。

- 新しい **enabled** 値を設定してキーを無効にします (例: **config.enabled=["false"]**)。
- 新規の **priority** の値を設定し、キーの優先度を変更します (例: **config.priority=["110"]**)。

古いキーの削除

1. 削除するキーがアクティブでなくなり、無効になっていることを確認します。このアクションは、アプリケーションおよびユーザーが保持する既存のトークンが失敗しないようにするためです。
2. 削除するキーを特定します。

```
$ kcadm.sh get keys -r demorealm
```

3. キーの **providerId** を使用して削除を実行します。

```
$ kcadm.sh delete components/PROVIDER_ID -r demorealm
```

レルムのイベントロギングの設定

events/config エンドポイントで **update** コマンドを使用します。

eventsListeners 属性には、イベントを受信するすべてのイベントリスナーを指定する EventListenerProviderFactory ID のリストが含まれます。組み込みイベントストレージを制御する属性を利用できるため、管理 REST API を使用して過去のイベントをクエリーできます。Red Hat build of Keycloak では、サービス呼び出しのロギング (**eventsEnabled**) と、管理コンソールまたは Admin REST API (**adminEventsEnabled**) によってトリガーされる監査イベントのロギングを個別に制御できます。**eventsExpiration** イベントを期限切れにし、データベースがいっぱいにならないようにすることができます。Red Hat build of Keycloak は、**eventsExpiration** を秒単位で表される存続期間に設定します。

すべてのイベントを受信し、JBoss-logging でイベントをログに記録する組み込みイベントリスナーを設定できます。Red Hat build of Keycloak は **org.keycloak.events** ロガーを使用して、エラーイベントを **WARN** としてログに記録し、その他のイベントを **DEBUG** としてログに記録します。

以下に例を示します。

- Linux:

```
$ kcadm.sh update events/config -r demorealm -s 'eventsListeners=["jboss-logging"]'
```

- Windows:

```
c:\> kcadm update events/config -r demorealm -s "eventsListeners=["jboss-logging\""]"
```

以下に例を示します。

監査イベントを除く、利用可能なすべての ERROR イベントのストレージを 2 日間オンにして、Admin REST を使用してイベントを取得できるようします。

- Linux:

```
$ kcadm.sh update events/config -r demorealm -s eventsEnabled=true -s
'enabledEventTypes=
["LOGIN_ERROR","REGISTER_ERROR","LOGOUT_ERROR","CODE_TO_TOKEN_ERRO
R","CLIENT_LOGIN_ERROR","FEDERATED_IDENTITY_LINK_ERROR","REMOVE_FEDE
RATED_IDENTITY_ERROR","UPDATE_EMAIL_ERROR","UPDATE_PROFILE_ERROR","U
PDATE_PASSWORD_ERROR","UPDATE_TOTP_ERROR","VERIFY_EMAIL_ERROR","RE
MOVE_TOTP_ERROR","SEND_VERIFY_EMAIL_ERROR","SEND_RESET_PASSWORD_E
RROR","SEND_IDENTITY_PROVIDER_LINK_ERROR","RESET_PASSWORD_ERROR","ID
ENTITY_PROVIDER_FIRST_LOGIN_ERROR","IDENTITY_PROVIDER_POST_LOGIN_ER
ROR","CUSTOM_REQUIRED_ACTION_ERROR","EXECUTE_ACTIONS_ERROR","CLIEN
T_REGISTER_ERROR","CLIENT_UPDATE_ERROR","CLIENT_DELETE_ERROR"] -s
eventsExpiration=172800
```

- Windows:

```
c:\> kcadm update events/config -r demorealm -s eventsEnabled=true -s
"enabledEventTypes=
[\"LOGIN_ERROR\", \"REGISTER_ERROR\", \"LOGOUT_ERROR\", \"CODE_TO_TOKEN_ER
ROR\", \"CLIENT_LOGIN_ERROR\", \"FEDERATED_IDENTITY_LINK_ERROR\", \"REMOVE_
FEDERATED_IDENTITY_ERROR\", \"UPDATE_EMAIL_ERROR\", \"UPDATE_PROFILE_ER
ROR\", \"UPDATE_PASSWORD_ERROR\", \"UPDATE_TOTP_ERROR\", \"VERIFY_EMAIL_E
RROR\", \"REMOVE_TOTP_ERROR\", \"SEND_VERIFY_EMAIL_ERROR\", \"SEND_RESET_
PASSWORD_ERROR\", \"SEND_IDENTITY_PROVIDER_LINK_ERROR\", \"RESET_PASSW
ORD_ERROR\", \"IDENTITY_PROVIDER_FIRST_LOGIN_ERROR\", \"IDENTITY_PROVIDE
```



```
R_POST_LOGIN_ERROR\","CUSTOM_REQUIRED_ACTION_ERROR\","EXECUTE_ACTIONS_ERROR\","CLIENT_REGISTER_ERROR\","CLIENT_UPDATE_ERROR\","CLIENT_DELETE_ERROR\"]" -s eventsExpiration=172800
```

保存されたイベントタイプは、**利用可能なすべてのイベントタイプ** にリセットできます。値を空のリストに設定すると、すべてを列挙することと同じです。

```
$ kcadm.sh update events/config -r demorealm -s enabledEventTypes=[]
```

監査イベントのストレージを有効にできます。

```
$ kcadm.sh update events/config -r demorealm -s adminEventsEnabled=true -s adminEventsDetailsEnabled=true
```

過去 100 件のイベントを取得できます。イベントは新しいものから古いものの順に並べられています。

```
$ kcadm.sh get events --offset 0 --limit 100
```

保存されたすべてのイベントを削除できます。

```
$ kcadm delete events
```

キャッシュのフラッシュ

1. 以下のエンドポイントのいずれかで **create** コマンドを使用して、キャッシュを消去します。

- **clear-realm-cache**
- **clear-user-cache**
- **clear-keys-cache**

2. **realm** をターゲットレルムと同じ値に設定します。
以下に例を示します。

```
$ kcadm.sh create clear-realm-cache -r demorealm -s realm=demorealm
$ kcadm.sh create clear-user-cache -r demorealm -s realm=demorealm
$ kcadm.sh create clear-keys-cache -r demorealm -s realm=demorealm
```

エクスポートされた .json ファイルからのレルムのインポート

1. **partialImport** エンドポイントで **create** コマンドを使用します。
2. **ifResourceExists** を **FAIL**、**SKIP**、または **OVERWRITE** に設定します。
3. **-f** を使用して、エクスポートしたレルムの **.json** ファイルを送信します。
以下に例を示します。

```
$ kcadm.sh create partialImport -r demorealm2 -s ifResourceExists=FAIL -o -f demorealm.json
```

レルムが存在しない場合は、最初に作成します。

以下に例を示します。

```
$ kcadm.sh create realms -s realm=demorealm2 -s enabled=true
```

17.7. ロール操作

レルムロールの作成

roles エンドポイントを使用してレルムロールを作成します。

```
$ kcadm.sh create roles -r demorealm -s name=user -s 'description=Regular user with a limited set of permissions'
```

クライアントロールの作成

1. クライアントを特定します。
2. **get** コマンドを使用して、利用可能なクライアントをリスト表示します。

```
$ kcadm.sh get clients -r demorealm --fields id,clientId
```

3. **clientId** 属性を使用して、**clients/ID/roles** などのエンドポイント URI を構築して新規ロールを作成します。
以下に例を示します。

```
$ kcadm.sh create clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles -r demorealm -s name=editor -s 'description=Editor can edit, and publish any article'
```

レルムロールのリスト表示

roles エンドポイントで **get** コマンドを使用して、既存のレルムロールをリスト表示します。

```
$ kcadm.sh get roles -r demorealm
```

get-roles コマンドも使用できます。

```
$ kcadm.sh get-roles -r demorealm
```

クライアントロールのリスト表示

Red Hat build of Keycloak には、レルムロールおよびクライアントロールの一覧表示を単純化する専用の **get-roles** コマンドがあります。このコマンドは **get** コマンドの拡張であり、**get** コマンドと同じように動作しますが、ロールをリストするための追加のセマンティクスがあります。

get-roles コマンドに **clientId** (**--cclientid**) オプションまたは **id** (**--cid**) オプションを指定して使用し、クライアントを識別してクライアントロールをリスト表示します。

以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --cclientid realm-management
```

特定のレルムロールの取得

get コマンドおよびロール名を使用して、特定のレルムロール (**roles/ROLE_NAME**) のエンドポイント URI を作成します。**user** は、既存ロールの名前に置き換えます。

以下に例を示します。

```
$ kcadm.sh get roles/user -r demorealm
```

get-roles コマンドを使用して、ロール名 (**--rolename** オプション) または ID (**--roleid** オプション) を指定できます。

以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --rolename user
```

特定のクライアントロールの取得

get-roles コマンドを使用して `clientId` 属性 (**--cclientid** オプション) または ID 属性 (**--cid** オプション) を渡してクライアントを特定し、ロール名 (**--rolename** オプション) またはロール ID 属性 (**--roleid**) を渡して、特定のクライアントロールを特定します。

以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --cclientid realm-management --rolename manage-clients
```

レルムロールの更新

特定のレルムロールを取得するために使用したエンドポイント URI で **update** コマンドを使用します。

以下に例を示します。

```
$ kcadm.sh update roles/user -r demorealm -s 'description=Role representing a regular user'
```

クライアントロールの更新

update コマンドを、特定のクライアントロールの取得に使用したエンドポイント URI で使用します。

以下に例を示します。

```
$ kcadm.sh update clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles/editor -r demorealm -s 'description=User that can edit, and publish articles'
```

レルムロールの削除

特定のレルムロールを取得するために使用したエンドポイント URI で **delete** コマンドを使用します。

以下に例を示します。

```
$ kcadm.sh delete roles/user -r demorealm
```

クライアントロールの削除

特定のクライアントロールを取得するために使用したエンドポイント URI で **delete** コマンドを使用します。

以下に例を示します。

```
$ kcadm.sh delete clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles/editor -r demorealm
```

複合ロールに割り当てられている、利用可能、有効なレルムロールのリスト表示

複合ロールに割り当てられ、利用可能で有効なレルムロールをリスト表示するには、**get-roles** コマンドを使用します。

1. 複合ロールに **割り当てられた** レルムロールをリスト表示するには、名前 (**--rname** オプション) または ID (**--rid** オプション) でターゲットの複合ロールを指定します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --rname testrole
```

2. **有効な** レルムロールをリスト表示するには、**--effective** オプションを使用します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --rname testrole --effective
```

3. **--available** オプションを使用して、複合ロールに追加できるレルムロールをリスト表示します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --rname testrole --available
```

複合ロールに割り当てられている、利用可能、有効なクライアントロールのリスト表示

複合ロールに割り当てられ、利用可能で有効なクライアントロールをリスト表示するには、**get-roles** コマンドを使用します。

1. 複合ロールに **割り当てられた** クライアントロールをリスト表示するには、名前 (**--rname** オプション) または ID (**--rid** オプション) でターゲット複合ロールを指定し、`clientId` 属性 (**--cclientid** オプション) または ID (**--cid** オプション) でクライアントを指定します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management
```

2. **有効な** レルムロールをリスト表示するには、**--effective** オプションを使用します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management --effective
```

3. **--available** オプションを使用して、ターゲット複合ロールに追加できるレルムロールをリスト表示します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management --available
```

レルムロールの複合ロールへの追加

Red Hat build of Keycloak には、レルムロールとクライアントロールを追加するための **add-roles** コマンドがあります。

この例では、**user** ロールを複合ロール **testrole** に追加します。

```
$ kcadm.sh add-roles --rname testrole --rolename user -r demorealm
```

複合ロールからのレルムロールの削除

Red Hat build of Keycloak には、レルムロールおよびクライアントロールを削除するための **remove-roles** コマンドがあります。

以下の例では、ターゲット複合ロール **testrole** から **user** ロールを削除します。

```
$ kcadm.sh remove-roles --rname testrole --rolename user -r demorealm
```

クライアントロールのレルムロールへの追加

Red Hat build of Keycloak には、レルムロールとクライアントロールを追加するための **add-roles** コマンドがあります。

以下の例では、クライアント **realm-management** に定義したロールを追加します。 **create-client** および **view-users** に定義されたロールを **testrole** 複合ロールに追加します。

```
$ kcadm.sh add-roles -r demorealm --rname testrole --cclientid realm-management --rolename create-client --rolename view-users
```

クライアントロールへのクライアントロールの追加

1. **get-roles** コマンドを使用して、複合クライアントロールの ID を確認します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --cclientid test-client --rolename operations
```

2. **test-client** という名前の `clientId` 属性、 **support** という名前のクライアントロール、および **operations** という名前のクライアントロールでクライアントが存在する場合には、ID が "fc400897-ef6a-4e8c-872b-1581b7fa8a71" び複合ロールになります。
3. 以下の例を使用して、別のロールを複合ロールに追加します。

```
$ kcadm.sh add-roles -r demorealm --cclientid test-client --rid fc400897-ef6a-4e8c-872b-1581b7fa8a71 --rolename support
```

4. **get-roles --all** コマンドを使用して、複合ロールのロールをリスト表示します。以下に例を示します。

```
$ kcadm.sh get-roles --rid fc400897-ef6a-4e8c-872b-1581b7fa8a71 --all
```

複合ロールからのクライアントロールの削除

remove-roles コマンドを使用して、複合ロールからクライアントロールを削除します。

以下の例を使用して、クライアントの **realm-management** で定義された2つのロールを削除します (**create-client** ロールと **view-users** ロール) を **testrole** 複合ロールから削除します。

```
$ kcadm.sh remove-roles -r demorealm --rname testrole --cclientid realm-management --rolename create-client --rolename view-users
```

クライアントロールのグループへの追加

add-roles コマンドを使用して、レルムロールおよびクライアントロールを追加します。

以下の例では、クライアントの **realm-management**、 **create-client** および **view-users** で定義されたロールを **Group** グループに追加します (**--gname** オプション)。または、ID でグループを指定できます (**--gid** オプション)。

詳細については、[グループ操作](#) を参照してください。

```
$ kcadm.sh add-roles -r demorealm --gname Group --cclientid realm-management --rolename  
create-client --rolename view-users
```

グループからのクライアントロールの削除

remove-roles コマンドを使用して、グループからクライアントロールを削除します。

以下の例では、クライアントの **レルム管理**、**create-client** および **view-users** に定義された 2 つのロールを **Group** グループから削除します。

詳細については、[グループ操作](#) を参照してください。

```
$ kcadm.sh remove-roles -r demorealm --gname Group --cclientid realm-management --rolename  
create-client --rolename view-users
```

17.8. クライアント操作

クライアントの作成

1. クライアントエンドポイントで **create** コマンドを実行して、新しい **clients** エンドポイントを作成します。
以下に例を示します。

```
$ kcadm.sh create clients -r demorealm -s clientId=myapp -s enabled=true
```

2. 認証するアダプターのシークレットを設定する場合は、シークレットを指定します。
以下に例を示します。

```
$ kcadm.sh create clients -r demorealm -s clientId=myapp -s enabled=true -s  
clientAuthenticatorType=client-secret -s secret=d0b8122f-8dfb-46b7-b68a-f5cc4e25d000
```

クライアントのリスト表示

clients エンドポイントで **get** コマンドを使用してクライアントをリスト表示します。

この例では、出力をフィルタリングして、**id** および **clientId** 属性のみを一覧表示します。

```
$ kcadm.sh get clients -r demorealm --fields id,clientId
```

特定のクライアントの取得

クライアントの ID を使用して、**クライアント/ID** などの特定のクライアントをターゲットとするエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm
```

特定クライアントの現在のシークレットの取得

クライアントの ID を使用して、**client/ID/client-secret** などのエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh get clients/$CID/client-secret -r demorealm
```

特定のクライアントの新規シークレットを生成します。

クライアントの ID を使用して、**client/ID/client-secret** などのエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh create clients/$CID/client-secret -r demorealm
```

特定クライアントの現在のシークレットの更新

クライアント ID を使用して **clients/ID** などのエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh update clients/$CID -s "secret=newSecret" -r demorealm
```

特定クライアントのアダプター設定ファイル (keycloak.json) の取得

クライアントの ID を使用して、**clients/ID/installation/providers/keycloak-oidc-keycloak-json** などの特定のクライアントをターゲットとするエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3/installation/providers/keycloak-oidc-keycloak-json -r demorealm
```

特定クライアントの WildFly サブシステムアダプター設定の取得

クライアントの ID を使用して、**clients/ID/installation/providers/keycloak-oidc-jboss-subsystem** などの特定のクライアントをターゲットとするエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3/installation/providers/keycloak-oidc-jboss-subsystem -r demorealm
```

特定クライアントの Docker-v2 の設定例

クライアントの ID を使用して、**clients/ID/installation/providers/docker-v2-compose-yaml** などの特定のクライアントをターゲットとするエンドポイント URI を作成します。

応答の形式は **.zip** です。

以下に例を示します。

```
$ kcadm.sh get http://localhost:8080/admin/realms/demorealm/clients/8f271c35-44e3-446f-8953-b0893810ebe7/installation/providers/docker-v2-compose-yaml -r demorealm > keycloak-docker-compose-yaml.zip
```

クライアントの更新

update コマンドを、特定のクライアントを取得するために使用したのと同じエンドポイント URI で使います。

以下に例を示します。

- Linux:

```
$ kcadm.sh update clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm -s
enabled=false -s publicClient=true -s 'redirectUri=["http://localhost:8080/myapp/*"]' -s
baseUrl=http://localhost:8080/myapp -s adminUrl=http://localhost:8080/myapp
```

- Windows:

```
c:\> kcadm update clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm -s
enabled=false -s publicClient=true -s "redirectUri=["http://localhost:8080/myapp/*"]" -s
baseUrl=http://localhost:8080/myapp -s adminUrl=http://localhost:8080/myapp
```

クライアントの削除

特定のクライアントを取得するために使用したエンドポイント URI で **delete** コマンドを使用します。

以下に例を示します。

```
$ kcadm.sh delete clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm
```

クライアントのサービスアカウントのロールの追加または削除

クライアントのサービスアカウントは、ユーザー名 **service-account-CLIENT_ID** を持つユーザーアカウントです。このアカウントで、通常のアカウントと同じユーザー操作を実行できます。

17.9. ユーザー操作

ユーザーの作成

users エンドポイントで **create** コマンドを実行して新規ユーザーを作成します。

以下に例を示します。

```
$ kcadm.sh create users -r demorealm -s username=testuser -s enabled=true
```

ユーザーのリスト表示

users エンドポイントを使用してユーザーをリスト表示します。ターゲットユーザーは、次回ログイン時にパスワードを変更する必要があります。

以下に例を示します。

```
$ kcadm.sh get users -r demorealm --offset 0 --limit 1000
```

username、**firstName**、**lastName**、または **email** でユーザーを絞り込むことができます。

以下に例を示します。

```
$ kcadm.sh get users -r demorealm -q email=google.com
$ kcadm.sh get users -r demorealm -q username=testuser
```



注記

フィルタリングは完全一致を使用しません。この例では、***testuser*** パターンに対して **username** 属性の値と一致します。

複数の属性でフィルタリングするには、複数の **-q** オプションを指定します。Red Hat build of Keycloak は、すべての属性の条件に一致するユーザーを返します。

特定ユーザーの取得

ユーザーの ID を使用して、**users/USER_ID** などのエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh get users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm
```

ユーザーの更新

update コマンドを、特定のクライアントを取得するために使用したのと同じエンドポイント URI で使用します。

以下に例を示します。

- Linux:

```
$ kcadm.sh update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm -s
'requiredActions=
["VERIFY_EMAIL","UPDATE_PROFILE","CONFIGURE_TOTP","UPDATE_PASSWORD"]'
```

- Windows:

```
c:\> kcadm update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm -s
'requiredActions=
["VERIFY_EMAIL","UPDATE_PROFILE","CONFIGURE_TOTP","UPDATE_PASSWORD
"]'
```

ユーザーの削除

delete コマンドを、特定のクライアントを取得するために使用したのと同じエンドポイント URI で使用します。

以下に例を示します。

```
$ kcadm.sh delete users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm
```

ユーザーのパスワードのリセット

専用の **set-password** コマンドを使用して、ユーザーのパスワードをリセットします。

以下に例を示します。

```
$ kcadm.sh set-password -r demorealm --username testuser --new-password NEWPASSWORD --
temporary
```

このコマンドは、ユーザーに一時パスワードを設定します。ターゲットユーザーは、次回ログイン時にパスワードを変更する必要があります。

id 属性では、**--userid** を使用してユーザーを指定できます。

users/USER_ID/reset-password など、特定のユーザーから構築されたエンドポイントで **update** コマンドを使用すると同じ結果が得られます。

以下に例を示します。

```
$ kcadm.sh update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2/reset-password -r demorealm -s
type=password -s value=NEWPASSWORD -s temporary=true -n
```

-n パラメーターを使用すると、Red Hat build of Keycloak は **PUT** コマンドの前に **GET** コマンドを実行しなくても、**PUT** コマンドを実行できます。**reset-password** エンドポイントは **GET** に対応していないため、この設定が必要です。

ユーザーに割り当てられている利用可能で有効なレルムロールのリスト表示

ユーザーに割り当てられ、利用可能で有効なレルムロールをリスト表示するには、**get-roles** コマンドを使用できます。

1. **割り当てられた** ユーザーのレルムロールをリスト表示するには、ユーザー名または ID でターゲットユーザーを指定します。
以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --username testuser
```

2. **有効な** レルムロールをリスト表示するには、**--effective** オプションを使用します。
以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --username testuser --effective
```

3. **--available** オプションを使用して、ユーザーに追加できるレルムロールをリスト表示します。
以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --username testuser --available
```

ユーザーに割り当てられている利用可能で有効なクライアントロールのリスト表示

ユーザーに割り当てられ、利用可能で有効なクライアントロールをリスト表示するには、**get-roles** コマンドを使用します。

1. ユーザー名 (**--username** オプション) または ID (**--uid** オプション) でターゲットユーザーを指定し、`clientId` 属性 (**--cclientid** オプション) または ID (**--cid** オプション) でクライアントを指定して、そのユーザーに **割り当てられた** クライアントロールをリスト表示します。
以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --username testuser --cclientid realm-management
```

2. **有効な** レルムロールをリスト表示するには、**--effective** オプションを使用します。
以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --username testuser --cclientid realm-management --effective
```

3. **--available** オプションを使用して、ユーザーに追加できるレルムロールをリスト表示します。
以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --username testuser --cclientid realm-management --available
```

レルムロールのユーザーへの追加

add-roles コマンドを使用して、レルムロールをユーザーに追加します。

以下の例を使用して、**user** ロールを **testuser** ユーザーに追加します。

```
$ kcadm.sh add-roles --username testuser --rolename user -r demorealm
```

ユーザーからのレルムロールの削除

ユーザーからレルムロールを削除するには、**remove-roles** コマンドを使用します。

以下の例を使用して、ユーザー **testuser** から **user** ロールを削除します。

```
$ kcadm.sh remove-roles --username testuser --rolename user -r demorealm
```

クライアントロールのユーザーへの追加

add-roles コマンドを使用して、クライアントロールをユーザーに追加します。

以下の例を使用して、クライアントの **レルム管理** で定義された2つのロールを追加します (**create-client** ロールと **view-users** ロールをユーザー **testuser** に追加します)。

```
$ kcadm.sh add-roles -r demorealm --username testuser --cclientid realm-management --rolename create-client --rolename view-users
```

ユーザーからのクライアントロールの削除

ユーザーからクライアントロールを削除するには、**remove-roles** コマンドを使用します。

以下の例を使用して、レルム管理クライアントで定義された2つのロールを削除します。

```
$ kcadm.sh remove-roles -r demorealm --username testuser --cclientid realm-management --rolename create-client --rolename view-users
```

ユーザーのセッションのリスト表示

1. ユーザーの ID を特定します。
2. ID を使用して、**users/ID/sessions** などのエンドポイント URI を作成します。
3. **get** コマンドを使用して、ユーザーのセッションのリストを取得します。
以下に例を示します。

```
$ kcadm.sh get users/6da5ab89-3397-4205-afaa-e201ff638f9e/sessions -r demorealm
```

特定のセッションからユーザーをログアウト

1. 上記のようにセッションの ID を決定します。
2. セッションの ID を使用して、**sessions/ID** などのエンドポイント URI を作成します。
3. **delete** コマンドを使用してセッションを無効にします。
以下に例を示します。

```
$ kcadm.sh delete sessions/d0eaa7cc-8c5d-489d-811a-69d3c4ec84d1 -r demorealm
```

全セッションからユーザーをログアウト

users/ID/logout などのエンドポイント URI を作成するには、ユーザーの ID を使用します。

create コマンドを使用して、そのエンドポイント URI で **POST** を実行します。

以下に例を示します。

```
$ kcadm.sh create users/6da5ab89-3397-4205-afaa-e201ff638f9e/logout -r demorealm -s realm=demorealm -s user=6da5ab89-3397-4205-afaa-e201ff638f9e
```

17.10. グループ操作

グループの作成

groups エンドポイントで **create** コマンドを使用して新規グループを作成します。

以下に例を示します。

```
$ kcadm.sh create groups -r demorealm -s name=Group
```

グループのリスト表示

groups エンドポイントで **get** コマンドを使用してグループのリストを表示します。

以下に例を示します。

```
$ kcadm.sh get groups -r demorealm
```

特定グループの取得

グループの ID を使用して、**groups/GROUP_ID** などのエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh get groups/51204821-0580-46db-8f2d-27106c6b5ded -r demorealm
```

グループの更新

update コマンドを、特定のグループの取得に使用したのと同じエンドポイント URI で使用します。

以下に例を示します。

```
$ kcadm.sh update groups/51204821-0580-46db-8f2d-27106c6b5ded -s 'attributes.email=["group@example.com"]' -r demorealm
```

グループの削除

delete コマンドを、特定のグループを取得するために使用したのと同じエンドポイント URI で使用します。

以下に例を示します。

```
$ kcadm.sh delete groups/51204821-0580-46db-8f2d-27106c6b5ded -r demorealm
```

サブグループの作成

グループをリスト表示して、親グループの ID を検索します。その ID を使用して **groups/GROUP_ID/children** などのエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh create groups/51204821-0580-46db-8f2d-27106c6b5ded/children -r demorealm -s name=SubGroup
```

別のグループでのグループの移動

1. 既存の親グループの ID と既存の子グループの ID を検索します。
2. 親グループの ID を使用して **groups/PARENT_GROUP_ID/children** などのエンドポイント URI を作成します。
3. このエンドポイントで **create** コマンドを実行し、子グループの ID を JSON の本文として渡します。

以下に例を示します。

```
$ kcadm.sh create groups/51204821-0580-46db-8f2d-27106c6b5ded/children -r demorealm -s id=08d410c6-d585-4059-bb07-54dcb92c5094 -s name=SubGroup
```

特定ユーザーのグループを取得する

ユーザーの ID を使用してグループ内のユーザーのメンバーシップを判別し、**users/USER_ID/groups** などのエンドポイント URI を作成します。

以下に例を示します。

```
$ kcadm.sh get users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups -r demorealm
```

グループへのユーザーの追加

users/USER_ID/groups/GROUP_ID などのユーザーの ID とグループの ID で設定されるエンドポイント URI で **update** コマンドを使用し、ユーザーをグループに追加します。

以下に例を示します。

```
$ kcadm.sh update users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups/ce01117a-7426-4670-a29a-5c118056fe20 -r demorealm -s realm=demorealm -s userId=b544f379-5fc4-49e5-8a8d-5cfb71f46f53 -s groupId=ce01117a-7426-4670-a29a-5c118056fe20 -n
```

グループからのユーザーの削除

users/USER_ID/groups/GROUP_ID などのグループにユーザーを追加するのに使用するエンドポイント URI で **delete** コマンドを使用し、グループからユーザーを削除します。

以下に例を示します。

```
$ kcadm.sh delete users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups/ce01117a-7426-4670-a29a-5c118056fe20 -r demorealm
```

グループに割り当てられている利用可能で有効なレルムロールのリスト表示

専用の **get-roles** コマンドを使用して、グループに割り当てられた、利用可能かつ有効なレルムロールをリスト表示します。

1. グループに **割り当てられた** レルムロールをリスト表示するには、名前 (**--gname** オプション)、パス (**--gpath** オプション)、ID (**--gid** オプション) でターゲットグループを指定します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --gname Group
```

2. **有効な** レルムロールをリスト表示するには、**--effective** オプションを使用します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --gname Group --effective
```

- 3. **--available** オプションを使用して、グループに追加できるレルムロールをリスト表示します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --gname Group --available
```

グループに割り当てられた、利用可能、有効なクライアントロールのリスト表示

グループに割り当てられ、利用可能で有効なクライアントロールをリスト表示するには、**get-roles** コマンドを使用します。

1. ターゲットグループを名前 (**--gname** オプション) または ID (**--gid** オプション) で指定します。
2. `clientId` 属性 (**--cclientid** オプション) または ID (**--id** オプション) でクライアントを指定して、そのユーザーに **割り当てられた** クライアントロールをリスト表示します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management
```

3. **有効な** レルムロールをリスト表示するには、**--effective** オプションを使用します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management --effective
```

4. **--available** オプションを使用して、グループに依然として追加可能なレルムロールをリスト表示します。以下に例を示します。

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management --available
```

17.11. アイデンティティプロバイダー操作

利用可能なアイデンティティプロバイダーのリスト表示

serverinfo エンドポイントを使用して利用可能な ID プロバイダーをリスト表示します。

以下に例を示します。

```
$ kcadm.sh get serverinfo -r demorealm --fields 'identityProviders(*)'
```



注記

Red Hat build of Keycloak は、**serverinfo** エンドポイントを **Realms** エンドポイントと同じように処理します。そのエンドポイントは特定のレルムの外側に存在するため、Red Hat build of Keycloak はターゲットレルムとの関連でそのエンドポイントを解決しません。

設定済みのアイデンティティプロバイダーのリスト表示

identity-provider/instances エンドポイントを使用します。

以下に例を示します。

```
$ kcadm.sh get identity-provider/instances -r demorealm --fields alias,providerId,enabled
```

特定の設定されたアイデンティティプロバイダーの取得

アイデンティティプロバイダーの **alias** 属性を使用して、**identity-provider/instances/ALIAS** などのエンドポイント URI を作成して特定のアイデンティティプロバイダーを取得します。

以下に例を示します。

```
$ kcadm.sh get identity-provider/instances/facebook -r demorealm
```

特定の設定されたアイデンティティプロバイダーの削除

delete コマンドを、特定の設定済みのアイデンティティプロバイダーを取得するために使用した同じエンドポイント URI で使用して、特定の設定済みのアイデンティティプロバイダーを削除します。

以下に例を示します。

```
$ kcadm.sh delete identity-provider/instances/facebook -r demorealm
```

Keycloak OpenID Connect アイデンティティプロバイダーの設定

1. 新規 ID プロバイダーインスタンスの作成時に、**keycloak-oidc** を **providerId** として使用します。
2. **config** 属性を指定します (**authorizationUrl**、**tokenUrl**、**clientId**、および **clientSecret**)。以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=keycloak-oidc -s
providerId=keycloak-oidc -s enabled=true -s 'config.useJwksUrl="true"' -s
config.authorizationUrl=http://localhost:8180/realms/demorealm/protocol/openid-connect/auth
-s config.tokenUrl=http://localhost:8180/realms/demorealm/protocol/openid-connect/token -s
config.clientId=demo-oidc-provider -s config.clientSecret=secret
```

OpenID Connect ID プロバイダーの設定

providerId 属性値を **oidc** に設定した場合を除き、Keycloak OpenID Connect プロバイダーを設定する方法で汎用 OpenID Connect プロバイダーを設定します。

SAML 2 アイデンティティプロバイダーの設定

1. **saml** を **providerId** として使用します。
2. **config** 属性 (**singleSignOnServiceUrl**、**nameIDPolicyFormat**、**signatureAlgorithm**) を指定します。

以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=saml -s providerId=saml -s
enabled=true -s 'config.useJwksUrl="true"' -s
config.singleSignOnServiceUrl=http://localhost:8180/realms/saml-broker-realm/protocol/saml -s
config.nameIDPolicyFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent -s
config.signatureAlgorithm=RSA_SHA256
```

Facebook アイデンティティプロバイダーの設定

1. **facebook** を **providerId** として使用します。

2. **config** 属性 **clientId** および **clientSecret** を指定します。これらの属性は、アプリケーションの Facebook Developers アプリケーション設定ページにあります。詳細は、[Facebook identity broker](#) のページを参照してください。
以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=facebook -s
providerId=facebook -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=FACEBOOK_CLIENT_ID -s
config.clientSecret=FACEBOOK_CLIENT_SECRET
```

Google ID プロバイダーの設定

1. **providerId** として **google** を使用します。
2. **config** 属性 **clientId** および **clientSecret** を指定します。これらの属性は、アプリケーションの Google Developers アプリケーション設定ページで確認できます。詳細については、[Google アイデンティティブローカー](#) ページを参照してください。
以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=google -s
providerId=google -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=GOOGLE_CLIENT_ID -s config.clientSecret=GOOGLE_CLIENT_SECRET
```

Twitter アイデンティティブローダーの設定

1. **twitter** を **providerId** として使用します。
2. **config** 属性 **clientId** および **clientSecret** を指定します。これらの属性は、アプリケーションの Twitter Application Management アプリケーション設定ページにあります。詳細については、[Twitter アイデンティティブローカー](#) ページを参照してください。
以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=google -s
providerId=google -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=TWITTER_API_KEY -s config.clientSecret=TWITTER_API_SECRET
```

GitHub ID プロバイダーの設定

1. **github** を **providerId** として使用します。
2. **config** 属性 **clientId** および **clientSecret** を指定します。これらの属性は、アプリケーションの GitHub Developer Application Settings ページにあります。詳細は、[GitHub identity broker](#) ページを参照してください。
以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=github -s
providerId=github -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=GITHUB_CLIENT_ID -s config.clientSecret=GITHUB_CLIENT_SECRET
```

LinkedIn アイデンティティブローダーの設定

1. **linkedin** を **providerId** として使用します。

2. **config** 属性 **clientId** および **clientSecret** を指定します。これらの属性は、アプリケーションの LinkedIn Developer Console アプリケーションページにあります。詳細については、[LinkedIn アイデンティティブローカー](#) ページを参照してください。
以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=linkedin -s
providerId=linkedin -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=LINKEDIN_CLIENT_ID -s config.clientSecret=LINKEDIN_CLIENT_SECRET
```

Microsoft Live アイデンティティプロバイダーの設定

1. **microsoft** を **providerId** として使用します。
2. **config** 属性 **clientId** および **clientSecret** を指定します。これらの属性は、アプリケーションの Microsoft Application Registration Portal ページにあります。詳細については、[Microsoft アイデンティティブローカー](#) ページを参照してください。
以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=microsoft -s
providerId=microsoft -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=MICROSOFT_APP_ID -s config.clientSecret=MICROSOFT_PASSWORD
```

Stack Overflow アイデンティティプロバイダーの設定

1. **stackoverflow** コマンドを **providerId** として使用します。
2. **config** 属性 (**clientId**、**clientSecret**、および **key**) を指定します。これらの属性は、アプリケーションの Stack Apps OAuth ページにあります。詳細については、[Stack Overflow アイデンティティブローカー](#) ページを参照してください。
以下に例を示します。

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=stackoverflow -s
providerId=stackoverflow -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=STACKAPPS_CLIENT_ID -s
config.clientSecret=STACKAPPS_CLIENT_SECRET -s config.key=STACKAPPS_KEY
```

17.12. ストレージプロバイダー操作

Kerberos ストレージプロバイダーの設定

1. **components** エンドポイントに対して **create** コマンドを使用します。
2. レルム ID を **parentId** 属性の値として指定します。
3. **kerberos** を **providerId** 属性値として指定し、**org.keycloak.storage.UserStorageProvider** を **providerType** 属性値として指定します。
4. 以下に例を示します。

```
$ kcadm.sh create components -r demorealm -s parentId=demorealmId -s id=demokerberos
-s name=demokerberos -s providerId=kerberos -s
providerType=org.keycloak.storage.UserStorageProvider -s 'config.priority=["0"]' -s
'config.debug=["false"]' -s 'config.allowPasswordAuthentication=["true"]' -s 'config.editMode=
["UNSYNCED"]' -s 'config.updateProfileFirstLogin=["true"]' -s
```

```
'config.allowKerberosAuthentication=["true"]' -s 'config.kerberosRealm=["KEYCLOAK.ORG"]'
-s 'config.keyTab=["http.keytab"]' -s 'config.serverPrincipal=
["HTTP/localhost@KEYCLOAK.ORG"]' -s 'config.cachePolicy=["DEFAULT"]'
```

LDAP ユーザストレージプロバイダーの設定

1. **components** エンドポイントに対して **create** コマンドを使用します。
2. **ldap** を **providerId** 属性の値に、**org.keycloak.storage.UserStorageProvider** を **providerType** 属性値として指定します。
3. レルム ID を **parentId** 属性の値として指定します。
4. 以下の例を使用して、Kerberos が統合する LDAP プロバイダーを作成します。

```
$ kcadm.sh create components -r demorealm -s name=kerberos-ldap-provider -s
providerId=ldap -s providerType=org.keycloak.storage.UserStorageProvider -s
parentId=3d9c572b-8f33-483f-98a6-8bb421667867 -s 'config.priority=["1"]' -s
'config.fullSyncPeriod=["-1"]' -s 'config.changedSyncPeriod=["-1"]' -s 'config.cachePolicy=
["DEFAULT"]' -s config.evictionDay=[] -s config.evictionHour=[] -s config.evictionMinute=[] -s
config.maxLifespan=[] -s 'config.batchSizeForSync=["1000"]' -s 'config.editMode=
["WRITABLE"]' -s 'config.syncRegistrations=["false"]' -s 'config.vendor=["other"]' -s
'config.usernameLDAPAttribute=["uid"]' -s 'config.rdnLDAPAttribute=["uid"]' -s
'config.uuidLDAPAttribute=["entryUUID"]' -s 'config.userObjectClasses=["inetOrgPerson,
organizationalPerson"]' -s 'config.connectionUrl=["ldap://localhost:10389"]' -s
'config.usersDn=["ou=People,dc=keycloak,dc=org"]' -s 'config.authType=["simple"]' -s
'config.bindDn=["uid=admin,ou=system"]' -s 'config.bindCredential=["secret"]' -s
'config.searchScope=["1"]' -s 'config.useTruststoreSpi=["always"]' -s
'config.connectionPooling=["true"]' -s 'config.pagination=["true"]' -s
'config.allowKerberosAuthentication=["true"]' -s 'config.serverPrincipal=
["HTTP/localhost@KEYCLOAK.ORG"]' -s 'config.keyTab=["http.keytab"]' -s
'config.kerberosRealm=["KEYCLOAK.ORG"]' -s 'config.debug=["true"]' -s
'config.useKerberosForPasswordAuthentication=["true"]'
```

ユーザストレージプロバイダーインスタンスの削除

1. ストレージプロバイダーインスタンスの **id** 属性を使用して、**components/ID** などのエンドポイント URI を作成します。
2. このエンドポイントに対して **delete** コマンドを実行します。
以下に例を示します。

```
$ kcadm.sh delete components/3d9c572b-8f33-483f-98a6-8bb421667867 -r demorealm
```

特定のユーザストレージプロバイダーに対するすべてのユーザーの同期のトリガー

1. ストレージプロバイダーの **id** 属性を使用して、**user-storage/ID_OF_USER_STORAGE_INSTANCE/sync** などのエンドポイント URI を作成します。
2. **action=triggerFullSync** クエリーパラメーターを追加します。
3. **create** コマンドを実行します。
以下に例を示します。

```
$ kcadm.sh create user-storage/b7c63d02-b62a-4fc1-977c-947d6a09e1ea/sync?
action=triggerFullSync
```

特定のユーザーストレージプロバイダーに対する変更済みのユーザーの同期のトリガー

1. ストレージプロバイダーの **id** 属性を使用して、**user-storage/ID_OF_USER_STORAGE_INSTANCE/sync** などのエンドポイント URI を作成します。
2. **action=triggerChangedUsersSync** クエリーパラメーターを追加します。
3. **create** コマンドを実行します。
以下に例を示します。

```
$ kcadm.sh create user-storage/b7c63d02-b62a-4fc1-977c-947d6a09e1ea/sync?
action=triggerChangedUsersSync
```

LDAP ユーザーのストレージ接続性のテスト

1. **testLDAPConnection** エンドポイントで **get** コマンドを実行します。
2. クエリーパラメーター **bindCredential**、**bindDn**、**connectionUrl**、および **useTruststoreSpi** を提供します。
3. **action** クエリーパラメーターを **testConnection** に設定します。
以下に例を示します。

```
$ kcadm.sh create testLDAPConnection -s action=testConnection -s bindCredential=secret -s
bindDn=uid=admin,ou=system -s connectionUrl=ldap://localhost:10389 -s
useTruststoreSpi=always
```

LDAP ユーザーのストレージ認証のテスト

1. **testLDAPConnection** エンドポイントで **get** コマンドを実行します。
2. クエリーパラメーター **bindCredential**、**bindDn**、**connectionUrl**、および **useTruststoreSpi** を指定します。
3. **action** クエリーパラメーターを **testAuthentication** に設定します。
以下に例を示します。

```
$ kcadm.sh create testLDAPConnection -s action=testAuthentication -s
bindCredential=secret -s bindDn=uid=admin,ou=system -s
connectionUrl=ldap://localhost:10389 -s useTruststoreSpi=always
```

17.13. マッパーの追加

ハードコーディングされたロールの LDAP マッパーの追加

1. **components** エンドポイントで **create** コマンドを実行します。
2. **providerType** 属性を **org.keycloak.storage.ldap.mappers.LDAPStorageMapper** に設定します。

3. **parentId** 属性を LDAP プロバイダーインスタンスの ID に設定します。
4. **providerId** 属性を **hardcoded-ldap-role-mapper** に設定します。 **role** 設定パラメーターの値を指定するようにしてください。
以下に例を示します。

```
$ kcadm.sh create components -r demorealm -s name=hardcoded-ldap-role-mapper -s
providerId=hardcoded-ldap-role-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config.role=["realm-
management.create-client"]'
```

MS Active Directory マッパーの追加

1. **components** エンドポイントで **create** コマンドを実行します。
2. **providerType** 属性を **org.keycloak.storage.ldap.mappers.LDAPStorageMapper** に設定します。
3. **parentId** 属性を LDAP プロバイダーインスタンスの ID に設定します。
4. **providerId** 属性を **msad-user-account-control-mapper** に設定します。
以下に例を示します。

```
$ kcadm.sh create components -r demorealm -s name=msad-user-account-control-mapper -
s providerId=msad-user-account-control-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea
```

ユーザー属性 LDAP マッパーの追加

1. **components** エンドポイントで **create** コマンドを実行します。
2. **providerType** 属性を **org.keycloak.storage.ldap.mappers.LDAPStorageMapper** に設定します。
3. **parentId** 属性を LDAP プロバイダーインスタンスの ID に設定します。
4. **providerId** 属性を **user-attribute-ldap-mapper** に設定します。
以下に例を示します。

```
$ kcadm.sh create components -r demorealm -s name=user-attribute-ldap-mapper -s
providerId=user-attribute-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."user.model.attribute"=
["email"]' -s 'config."ldap.attribute"=["mail"]' -s 'config."read.only"=["false"]' -s
'config."always.read.value.from.ldap"=["false"]' -s 'config."is.mandatory.in.ldap"=["false"]'
```

グループの LDAP マッパーの追加

1. **components** エンドポイントで **create** コマンドを実行します。
2. **providerType** 属性を **org.keycloak.storage.ldap.mappers.LDAPStorageMapper** に設定します。

3. **parentId** 属性を LDAP プロバイダーインスタンスの ID に設定します。
4. **providerId** 属性を **group-ldap-mapper** に設定します。
以下に例を示します。

```
$ kcadm.sh create components -r demorealm -s name=group-ldap-mapper -s
providerId=group-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."groups.dn"=[]' -s
'config."group.name.ldap.attribute"=["cn"]' -s 'config."group.object.classes"=
["groupOfNames"]' -s 'config."preserve.group.inheritance"=["true"]' -s
'config."membership.ldap.attribute"=["member"]' -s 'config."membership.attribute.type"=
["DN"]' -s 'config."groups.ldap.filter"=[]' -s 'config.mode=["LDAP_ONLY"]' -s
'config."user.roles.retrieve.strategy"=["LOAD_GROUPS_BY_MEMBER_ATTRIBUTE"]' -s
'config."mapped.group.attributes"=["admins-group"]' -s
'config."drop.non.existing.groups.during.sync"=["false"]' -s 'config.roles=["admins"]' -s
'config.groups=["admins-group"]' -s 'config.group=[]' -s 'config.preserve=["true"]' -s
'config.membership=["member"]'
```

フルネームの LDAP マッパーの追加

1. **components** エンドポイントで **create** コマンドを実行します。
2. **providerType** 属性を **org.keycloak.storage.ldap.mappers.LDAPStorageMapper** に設定しま
す。
3. **parentId** 属性を LDAP プロバイダーインスタンスの ID に設定します。
4. **providerId** 属性を **full-name-ldap-mapper** に設定します。
以下に例を示します。

```
$ kcadm.sh create components -r demorealm -s name=full-name-ldap-mapper -s
providerId=full-name-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."ldap.full.name.attribute"=
["cn"]' -s 'config."read.only"=["false"]' -s 'config."write.only"=["true"]'
```

17.14. 認証操作

パスワードポリシーの設定

1. レルムの **passwordPolicy** 属性を、特定のポリシープロバイダー ID と任意の設定が含まれる列
挙式に設定します。
2. 以下の例を使用して、パスワードポリシーをデフォルト値に設定します。デフォルト値には以
下が含まれます。
 - 27,500 ハッシュの反復
 - 少なくとも1つの特殊文字
 - 少なくとも1つの大文字の文字
 - 最低でも1桁の文字

- ユーザーの **username** と等しくない
- 8 文字以上であること

```
$ kcadm.sh update realms/demorealm -s 'passwordPolicy="hashIterations and specialChars and upperCase and digits and notUsername and length"
```

3. デフォルト値と異なる値を使用する場合は、括弧で設定を渡します。
4. 以下の例を使用して、パスワードポリシーを以下のように設定します。

- 25,000 ハッシュの反復
- 少なくとも 2 つの特殊文字
- 少なくとも 2 つの大文字
- 少なくとも 2 つの小文字
- 少なくとも 2 桁
- 最低 9 文字の長さであること
- ユーザーの **username** と等しくない
- 少なくとも 4 つの変更の場合、繰り返すことはありません。

```
$ kcadm.sh update realms/demorealm -s 'passwordPolicy="hashIterations(25000) and specialChars(2) and upperCase(2) and lowerCase(2) and digits(2) and length(9) and notUsername and passwordHistory(4)"
```

現在のパスワードポリシーの取得

passwordPolicy 属性以外のすべての出力をフィルターすると、現在のレルム設定を取得できます。

たとえば、**demorealm** の **passwordPolicy** を表示します。

```
$ kcadm.sh get realms/demorealm --fields passwordPolicy
```

認証フローのリスト表示

authentication/flows エンドポイントで **get** コマンドを実行します。

以下に例を示します。

```
$ kcadm.sh get authentication/flows -r demorealm
```

特定の認証フローの取得

authentication/flows/FLOW_ID エンドポイントで **get** コマンドを実行します。

以下に例を示します。

```
$ kcadm.sh get authentication/flows/febfd772-e1a1-42fb-b8ae-00c0566fafb8 -r demorealm
```

フローの実行のリスト表示

authentication/flows/FLOW_ALIAS/executions エンドポイントで **get** コマンドを実行します。

以下に例を示します。

```
$ kcadm.sh get authentication/flows/Copy%20of%20browser/executions -r demorealm
```

実行への設定の追加

1. フローの実行を取得します。
2. フローの ID をメモします。
3. **authentication/executions/{executionId}/config** エンドポイントで **create** コマンドを実行します。

以下に例を示します。

```
$ kcadm.sh create "authentication/executions/a3147129-c402-4760-86d9-3f2345e401c7/config" -r demorealm -b '{"config":{"x509-cert-auth.mapping-source-selection":"Match SubjectDN using regular expression","x509-cert-auth.regular-expression":"(.*?)(?:$)","x509-cert-auth.mapper-selection":"Custom Attribute Mapper","x509-cert-auth.mapper-selection.user-attribute-name":"usercertificate","x509-cert-auth.crl-checking-enabled":"","x509-cert-auth.crdp-checking-enabled":false,"x509-cert-auth.crl-relative-path":"crl.pem","x509-cert-auth.ocsp-checking-enabled":"","x509-cert-auth.ocsp-responder-uri":"","x509-cert-auth.keyusage":"","x509-cert-auth.extendedkeyusage":"","x509-cert-auth.confirmation-page-disallowed":"","alias":"my_otp_config"}'}
```

実行設定の取得

1. フローの実行を取得します。
2. 設定 ID が含まれる **authenticationConfig** 属性をメモします。
3. **authentication/config/ID** エンドポイントで **get** コマンドを実行します。

以下に例を示します。

```
$ kcadm get "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r demorealm
```

実行設定の更新

1. フローの実行を取得します。
2. フローの **authenticationConfig** 属性を取得します。
3. 属性の設定 ID をメモします。
4. **authentication/config/ID** エンドポイントで **update** コマンドを実行します。

以下に例を示します。

```
$ kcadm update "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r demorealm -b '{"id":"dd91611a-d25c-421a-87e2-227c18421833","alias":"my_otp_config","config":{"x509-cert-auth.extendedkeyusage":"","x509-cert-auth.mapper-selection.user-attribute-name":"usercertificate","x509-cert-auth.ocsp-responder-uri":"","x509-cert-auth.regular-expression":"(.*?)(?:$)","x509-cert-auth.crl-checking-enabled":"true","x509-cert-auth.confirmation-page-disallowed":"","x509-cert-auth.keyusage":"","x509-cert-auth.mapper-selection":"Custom Attribute
```

```
Mapper", "x509-cert-auth.crl-relative-path": "crl.pem", "x509-cert-auth.crl-dp-checking-enabled": "false", "x509-cert-auth.mapping-source-selection": "Match SubjectDN using regular expression", "x509-cert-auth.ocsp-checking-enabled": ""}}'
```

実行設定の削除

1. フローの実行を取得します。
2. フロー **authenticationConfig** 属性を取得します。
3. 属性の設定 ID をメモします。
4. **authentication/config/ID** エンドポイントで **delete** コマンドを実行します。

以下に例を示します。

```
$ kcadm delete "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r demorealm
```