Red Hat build of Keycloak 24.0

認可サービスガイド

Last Updated: 2024-06-04

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 24.0 の認可サービスに関する情報が記載されています。

目次

多様性を受け入れるオープンソースの強化

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

第1章 認可サービスの概要

Red Hat build of Keycloak は詳細な認可ポリシーをサポートしており、次のようなさまざまなアクセス制御メカニズムを組み合わせることができます。

- **属性ベースのアクセス制御 (ABAC)**
- **ロールベースアクセス制御 (RBAC)**
- **ユーザーベースのアクセス制御 (UBAC)**
- **コンテキストベースのアクセス制御 (CBAC)**
- **ルールベースのアクセス制御**
  - JavaScript の使用
- **時間ベースのアクセス制御**
- **SPI (Service Provider Interface) によるカスタムアクセス制御メカニズム (ACM) のサポート**

Red Hat build of Keycloak は管理 UI と RESTful API のセットをベースとしており、必要な手段として保護されているリソースやスコープのパーミッションを作成し、それらのパーミッションを認可ポリシーに関連付け、アプリケーションやサービスで認可に関する決定を強制します。

リソースサーバー (アプリケーションまたはサービス提供で保護されているリソース) は、通常、保護されたリソースにアクセスを付与すべきかどうかを決定するために、一定種類の情報に依存します。RESTful ベースのリソースサーバーの場合、その情報は通常はセキュリティートークンから取得され、通常はすべてのリクエストに対してベアラートークンとして送信されます。セッションに依存してユーザーを認証する web アプリケーションでは、その情報は通常、ユーザーのセッションに保存され、各リクエストに対してそこから取得されます。

多くの場合、リソースサーバーは、ロールベースのアクセス制御 (RBAC) に基づいて認可の決定のみを行います。ここでは、保護されたリソースにアクセスしようとするユーザーに付与されたロールが、これらの同じリソースにマップされたロールと照合されます。ロールには非常に便利ですが、アプリケーションにはいくつかの制限があります。

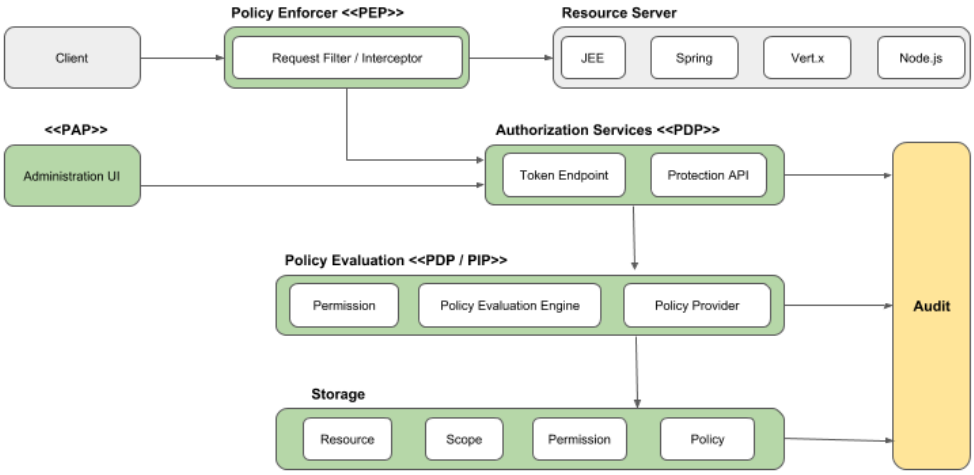
- リソースとロールが密接に結合され、ロール (アクセスコンテキストの追加、削除、または変更など) が複数のリソースに影響する可能性があります。
- セキュリティー要件を変更すると、アプリケーションコードへの深い変更を確認して、
- アプリケーションのサイズによっては、ロール管理が困難になり、
- 最も柔軟なアクセス制御メカニズムではありません。ロールは、コンテキスト情報がなく、コンテキスト情報がないユーザーを表すことはありません。ロールを付与している場合は、少なくともいくつかのアクセスがあります。

今日、ユーザーが異なる地域に分散し、異なるローカルポリシーを持ち、異なるデバイスを使用し、情報共有の要求が高い異種環境を考慮する必要があることを考えると、Red Hat build of Keycloak の認可サービスは、以下を提供することで、アプリケーションやサービスの認可機能を向上させることができます。

- 詳細な認可ポリシーと異なるアクセス制御メカニズムを使用したリソース保護
- 一元化されたリソース、パーミッション、およびポリシー管理

- 集中ポリシー定義点
- REST ベースの認可サービスのセットに基づく REST セキュリティー
- 認可ワークフローおよびユーザー管理のアクセス
- プロジェクト全体でのコードの複製を回避するためのインフラストラクチャー (および再デプロイ) は、セキュリティー要件の変更に迅速に対応します。

1.1. アーキテクチャー



設計の観点からは、認可サービスはこれらの機能を提供する明確に定義された認可パターンのセットに基づいています。

- **ポリシー管理ポイント (PAP)**
  リソースサーバー、リソース、スコープ、パーミッション、ポリシーを管理するために、Red Hat build of Keycloak の管理コンソールをベースにした一連の UI を提供します。この一部は、保護 API を使用してリモートで実行できます。
- **ポリシー決定ポイント (PDP)**
  分散可能なポリシーの意思決定ポイントを、認可要求が送信され、ポリシーを要求するパーミッションに応じて評価されます。詳細は、パーミッションの取得 を参照してください。
- **ポリシー実施ポイント (PEP)**
  リソースサーバー側で認可決定を強制するさまざまな環境の実装を提供します。Red Hat build of Keycloak は、いくつかのビルトイン ポリシーエンフォーサー を提供します。
- **ポリシー情報ポイント (PIP)**
  Red Hat build of Keycloak の Authentication Server をベースとしており、認可ポリシーの評価時にアイデンティティーおよびランタイム環境から属性を取得できます。

1.1.1. 認可プロセス

Red Hat build of Keycloak を使用してアプリケーションに対するきめ細かな認可を有効にする方法を理解するために必要な手順を、3 つの主なプロセスで定義します。

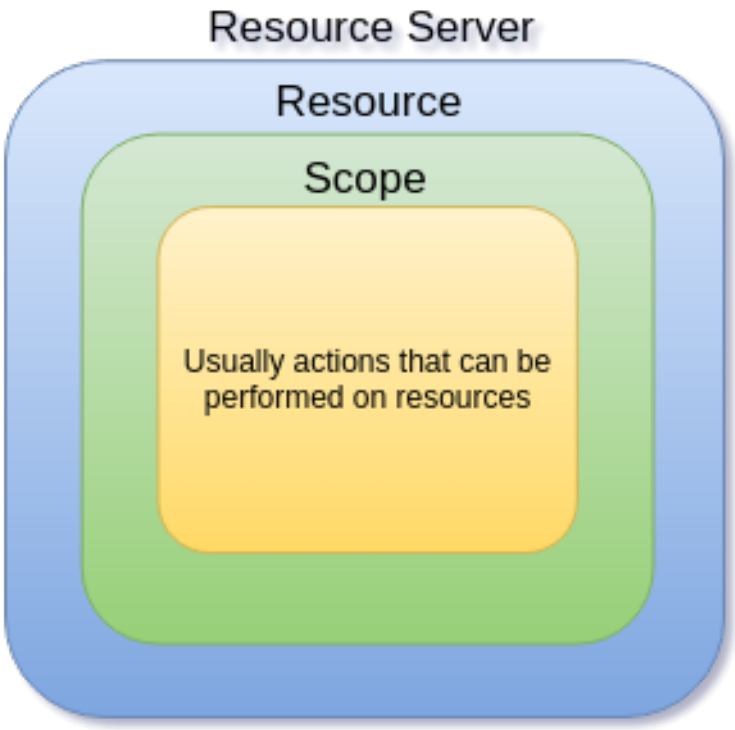
- **リソースの管理**
- **パーミッションおよびポリシーの管理**
- **ポリシーの強制**

1.1.1.1. リソース管理

リソースの管理 には、保護されているものを定義するために必要なすべての手順が含まれます。

まず、保護する Red Hat build of Keycloak を指定する必要があります。これは通常、Web アプリケーションまたは 1 つ以上のサービスのセットを表します。リソースサーバーの詳細は、用語 を参照してください。

リソースサーバーは、Red Hat build of Keycloak の管理コンソールを使用して管理されます。登録済みのクライアントアプリケーションをリソースサーバーとして有効にし、保護するリソースおよびスコープの管理を開始することができます。



リソースは、Web ページ、RESTFul リソース、ファイルシステム内のファイル、EJB などになります。これらは、リソースのグループを表すことも (Java の Class のように)、単一の特定のリソースを表すこともできます。

たとえば、すべての銀行口座を表す **Bank Account** リソースがあり、それを使用してすべての銀行口座に共通の認可ポリシーを定義する場合があります。ただし、所有者だけが一部の情報へのアクセスや操作の実行を許可されている **Alice Account** (顧客に属するリソースインスタンス) に特定のポリシーを定義したい場合があります。

リソースは、Red Hat build of Keycloak の管理コンソールまたは Protection API を使用して管理できます。後者の場合、リソースサーバーはリソースをリモートで管理できます。

スコープは、通常リソースで実行できるアクションを表しますが、これらに限定されません。スコープを使用してリソース内の 1 つまたは複数の属性を表すこともできます。

1.1.1.2. パーミッションおよびポリシーの管理

リソースサーバーと保護するすべてのリソースを定義した後に、パーミッションとポリシーを設定する必要があります。

このプロセスでは、リソースを管理するセキュリティーおよびアクセス要件を実際に定義するのに必要なすべてのステップが関係します。

ポリシーは、何か (リソースやスコープ) にアクセスしたり操作を実行したりする際に満たさなければならない条件を定義するものですが、何を保護しているかには縛られていません。これらは汎用的であり、パーミッションの構築やより複雑なポリシーの構築に使用できます。

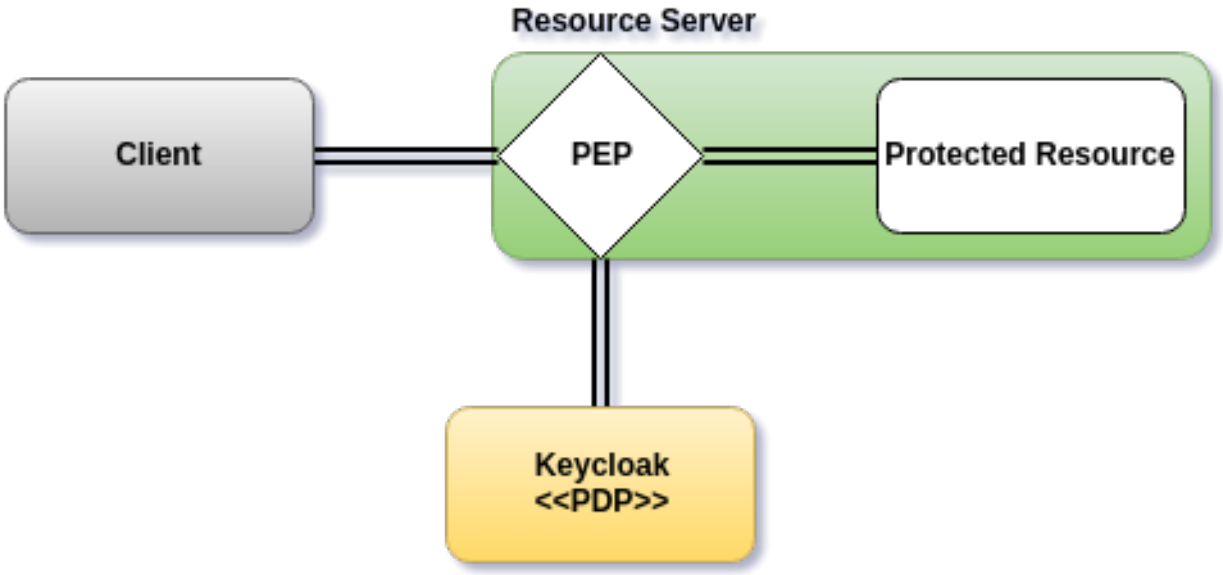
例えば、"User Premium "というロールを付与されたユーザーにのみ、あるリソース群へのアクセスを許可するには、RBAC(Role-based Access Control) を使用します。

Red Hat build of Keycloak は、最も一般的なアクセス制御メカニズムをカバーするいくつかのビルトインポリシータイプ (およびそれぞれのポリシープロバイダー) を提供します。JavaScript を使用して記述されたルールに基づいてポリシーを作成することもできます。

ポリシーを定義したら、パーミッションの定義を開始できます。パーミッションは、保護しているリソースと結合されます。ここでは、許可を許可または拒否するために満たさなければならないポリシー (リソースまたはスコープ) およびポリシーを指定します。

1.1.1.3. ポリシーの強制

ポリシーの強制 には、リソースサーバーに認可の決定を実際に強制するために必要な手順が含まれます。これは、認可サーバーと通信できるリソースサーバーで **ポリシー強制ポイント** または PEP を有効にし、サーバーが返した決定とパーミッションに基づいて認可データと、保護リソースへのアクセスを制御することによって使用できます。



Red Hat build of Keycloak は、実行中のプラットフォームに応じてアプリケーションを保護するために使用できるビルトイン ポリシーエンフォーサー 実装を提供します。

1.1.2. 認可サービス

認可サービスは、以下の RESTFul エンドポイントで設定されます。

- **トークンエンドポイント**
- **リソース管理エンドポイント**
- **パーミッション管理エンドポイント**

これらのサービスごとに、認可プロセスに関係するさまざまなステップをカバーする特定の API を提供します。

1.1.2.1. トークンエンドポイント

OAuth2 クライアント (フロントエンドアプリケーションなど) は、トークンエンドポイントを使用してサーバーからアクセストークンを取得し、その同じトークンを使用してリソースサーバー (バックエンドサービスなど) で保護されたリソースにアクセスすることができます。同様に、Red Hat build of Keycloak の Authorization Services は OAuth2 の拡張機能を提供し、要求されたリソースまたはスコープに関連するすべてのポリシーの処理に基づいてアクセストークンを発行できます。つまり、リソースサーバーは、サーバーによって付与されるパーミッションに基づいて保護されているリソースへのアクセスを強制でき、アクセストークンによって保持されます。Red Hat build of Keycloak の Authorization Services では、パーミッションのあるアクセストークンは Requesting Party Token (RPT) と呼ばれます。

関連情報

- パーミッションの取得

1.1.2.2. 保護 API

保護 API は、リソースサーバーの一連の UMA 準拠 のエンドポイントで、それらに関連付けられたリソース、スコープ、パーミッション、およびポリシーの管理に役立ちます。この API へのアクセスはリソースサーバーのみです。また、**uma_protection** スコープも必要です。

保護 API が提供する操作は、以下の 2 つのグループに分類できます。

- **リソースの管理**
  - リソースの作成
  - リソースの削除
  - Id で検索
  - クエリー
- **パーミッションの管理**
  - パーミッションチケットの発行

注記

デフォルトでは、Remote Resource Management は有効になります。Red Hat build of Keycloak の管理コンソールを使用してこれを変更し、コンソール経由のリソース管理のみを許可できます。

UMA プロトコルを使用する場合、保護 API による Permission Tickets の発行は、認可プロセス全体における重要な部分です。後続のセクションで説明されているように、クライアントは要求するパーミッションを表し、サーバーへ送信され、要求されるリソースおよびスコープに関連付けられたパーミッションおよびポリシーの評価時に付与されるすべてのパーミッションで最終的なトークンを取得します。

関連情報

- 保護 API

1.2. 用語

先に進む前に、Red Hat build of Keycloak の Authorization Services で導入された用語と概念について理解することが重要です。

1.2.1. リソースサーバー

OAuth2 の用語ごとに、リソースサーバーは保護されたリソースをホストするサーバーであり、保護されているリソース要求を受け入れて応答できます。

リソースサーバーは通常、保護されているリソースへのアクセスを付与すべきかどうかを決定するために、一定種類の情報に依存します。RESTful ベースのリソースサーバーの場合、その情報は通常セキュリティートークンで実行されます。通常、サーバーへのすべてのリクエストとともにベアラートークンとして送信されます。セッションに依存する Web アプリケーションは、通常、ユーザーのセッションにその情報を保存し、リクエストごとにその情報を取得して取得します。

Red Hat build of Keycloak では、任意の **機密** クライアントアプリケーションがリソースサーバーとして機能できます。このクライアントのリソースとそれに対応するスコープは、認可ポリシーのセットで保護され、管理されます。

1.2.2. リソース

リソースは、アプリケーションと組織のアセットの一部です。これには、1つ以上のエンドポイント、HTML ページなどの従来の Web リソースなどを指定できます。認可ポリシーの用語では、リソースは保護される **オブジェクト** です。

すべてのリソースには、単一リソースまたはリソースセットを表す一意の ID があります。たとえば、すべての銀行口座の一連の認可ポリシーを表し、定義する**銀行口座リソース** を管理できます。ただし、**Alice's Banking Account**という名前の別のリソースを使用することもできます。このリソースは、1 人の顧客が所有する単一のリソースで、独自の認可ポリシーのセットを持つことができます。

1.2.3. スコープ

リソースのスコープは、リソースで実行できるアクセスのバインドエクステントです。認可ポリシーの用語では、スコープはリソースに論理的に適用できる多くの **動詞** の1つです。

通常、指定リソースで実行できることを示します。スコープの例には view、edit、delete などがあります。ただし、スコープはリソースによって提供される特定の情報に関連付けることもできます。この場合、プロジェクトリソースとコストスコープを設定できます。コストスコープを使用して、ユーザーがプロジェクトのコストにアクセスする特定のポリシーおよびパーミッションを定義するために使用されます。

1.2.4. パーミッション

この簡単なパーミッションと非常に一般的なパーミッションについて考えてみましょう。

パーミッション は、アクセスが許可されるかどうかを判別するために評価される必要のあるポリシーと保護されるオブジェクトを関連付けます。

- **X** はリソース **Z**で **Y** を実行できます。
    - ここでは、以下のようになります。
        - **X** は、1つ以上のユーザー、ロール、またはグループ、もしくはその組み合わせを表します。ここでは、要求およびコンテキストを使用することもできます。
        - **Y** は、書き込み、表示などの実行するアクションを表します。
        - **Z** は、保護されるリソース (例: /accounts) を表します。

Red Hat build of Keycloak は、シンプルなものから非常に複雑なルールベースの動的パーミッションまで、さまざまなパーミッションストラテジーを構築する豊富なプラットフォームを提供します。柔軟であり、以下に役立ちます。

- コードリファクタリングおよびパーミッション管理コストの削減
- より柔軟なセキュリティーモデルをサポートするため、セキュリティー要件の変更を簡単に調整できます。
- 実行時に変更を行います。アプリケーションは、保護されているリソースおよびスコープについてのみ関係し、それらの保護方法には関係しません。

1.2.5. ポリシー

ポリシーは、オブジェクトへのアクセスを付与するために満たさなければならない条件を定義します。パーミッションとは異なり、保護されているオブジェクトは指定しないでくださいが、指定のオブジェクトへのアクセスに対して満たさなければならない条件 (リソース、スコープ、またはその両方など) は指定しないでください。ポリシーは、リソースの保護に使用できるさまざまなアクセス制御メカニズム

(ACM) に関係があります。ポリシーを使用すると、属性ベースのアクセス制御 (ABAC)、ロールベースのアクセス制御 (RBAC)、コンテキストベースのアクセス制御またはこれらの組み合わせのストラテジーを実装できます。

Red Hat build of Keycloak は、ポリシーの概念と、集約ポリシーの概念を提供することで、それらの定義方法を活用しています。ここでは、ポリシーのポリシーを構築し、評価の動作も制御することができます。指定のリソースへのアクセスするために満たさなければならないすべての条件で 1 つの大規模なポリシーを作成する代わりに、Red Hat build of Keycloak Authorization Services のポリシー実装は divide-and-conquer 手法に従います。つまり、個別のポリシーを作成して、異なるパーミッションで再利用し、個々のポリシーを組み合わせてより複雑なポリシーをビルドできます。

1.2.6. ポリシープロバイダー

ポリシープロバイダーは、特定のポリシータイプの実装です。Red Hat build of Keycloak は、対応するポリシープロバイダーが関係する組み込みポリシーを提供し、特定の要件に対応するために独自のポリシータイプを作成できます。

Red Hat build of Keycloak は、独自のポリシープロバイダー実装でプラグインするために使用できる SPI (サービスプロバイダーインターフェイス) を提供します。

1.2.7. パーミッションチケット

パーミッションチケットは、認可サーバーによってフォームが決定される不透明な構造を提供する、User-Managed Access (UMA) 仕様で定義されている特別なタイプのトークンです。この構造は、クライアント、アクセスコンテキスト、および認可データの要求に適用する必要のあるポリシー (要求側のトークン [RPT]) に対し、リソースやスコープを表します。

UMA では、パーソナツーの共有と人対組織共有にも対応するために、パーミッションチケットが重要です。認可ワークフローのパーミッションチケットを使用すると、リソースの所有者とリソースサーバーがこれらのリソースへのアクセスを管理する粒度の細かいポリシーに基づいて、リソースの所有者とリソースサーバーでリソースを完全に制御できます。

UMA ワークフローでは、リソースサーバーにパーミッションチケットが発行され、保護されているリソースにアクセスするクライアントに対するパーミッションチケットを返します。クライアントがチケットを受け取ると、チケットを認可サーバーに送信して、RPT(認可データを保持する最終トークン) のリクエストを行うことができます。

パーミッションチケットの詳細は、ユーザー管理アクセス および UMA 仕様を参照してください。

第2章 スタートガイド

特定のアプリケーションについては、次のリソースを参照して、Red Hat build of Keycloak の Authorization Services をすぐに開始できます。

- Securing a JakartaEE Application in Wildfly
- Securing a Spring Boot Application
- Securing Quarkus Applications
- Node.js アプリケーションの保護

第3章 リソースサーバーの管理

OAuth2 仕様によると、リソースサーバーは保護されたリソースをホストするサーバーであり、保護されているリソース要求を受け入れて応答できます。

Red Hat build of Keycloak では、リソースサーバーは、保護されているリソースのきめ細かな認可を可能にする豊富なプラットフォームで提供されるため、異なるアクセス制御メカニズムに基づき認可を決定できます。

粒度の細かいパーミッションをサポートするようにクライアントアプリケーションを設定できます。そのため、クライアントアプリケーションをリソースサーバーに概念的に有効にすることができます。

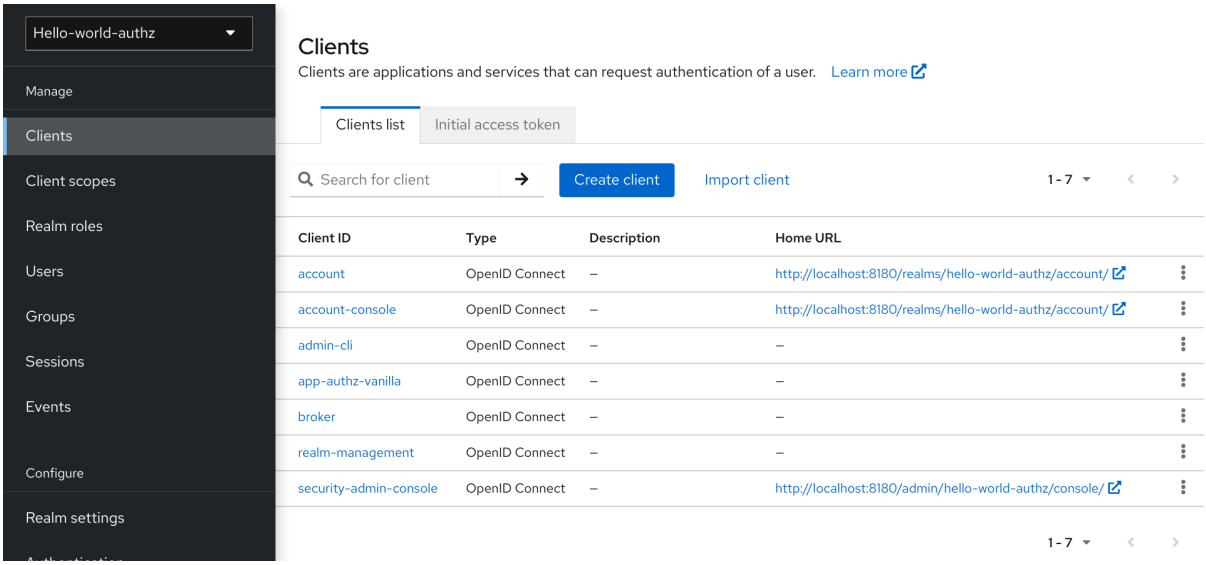
3.1. クライアントアプリケーションの作成

Red Hat build of Keycloak の Authorization Services を有効にする最初の手順は、リソースサーバーに変換するクライアントアプリケーションを作成することです。

手順

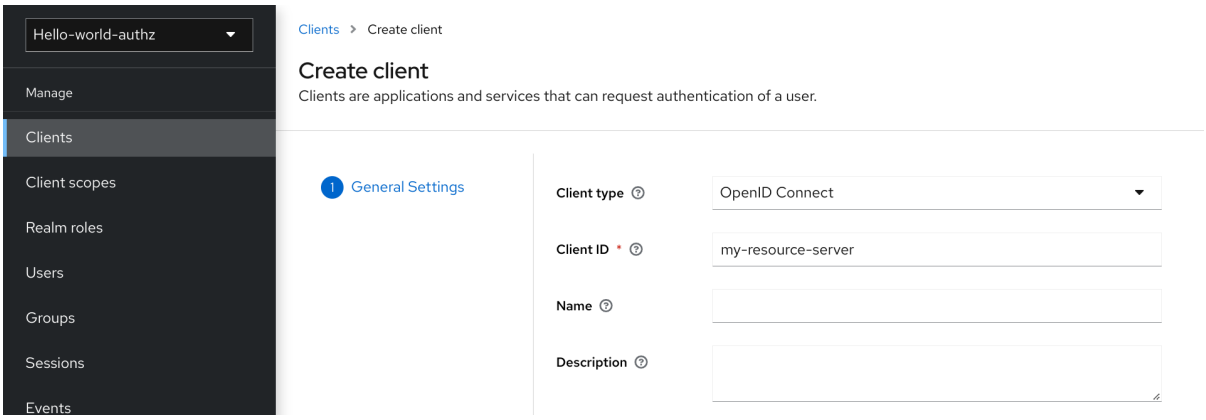
1. **Clients** をクリックします。

Clients



2. このページで、**Create** をクリックします。

クライアントの追加

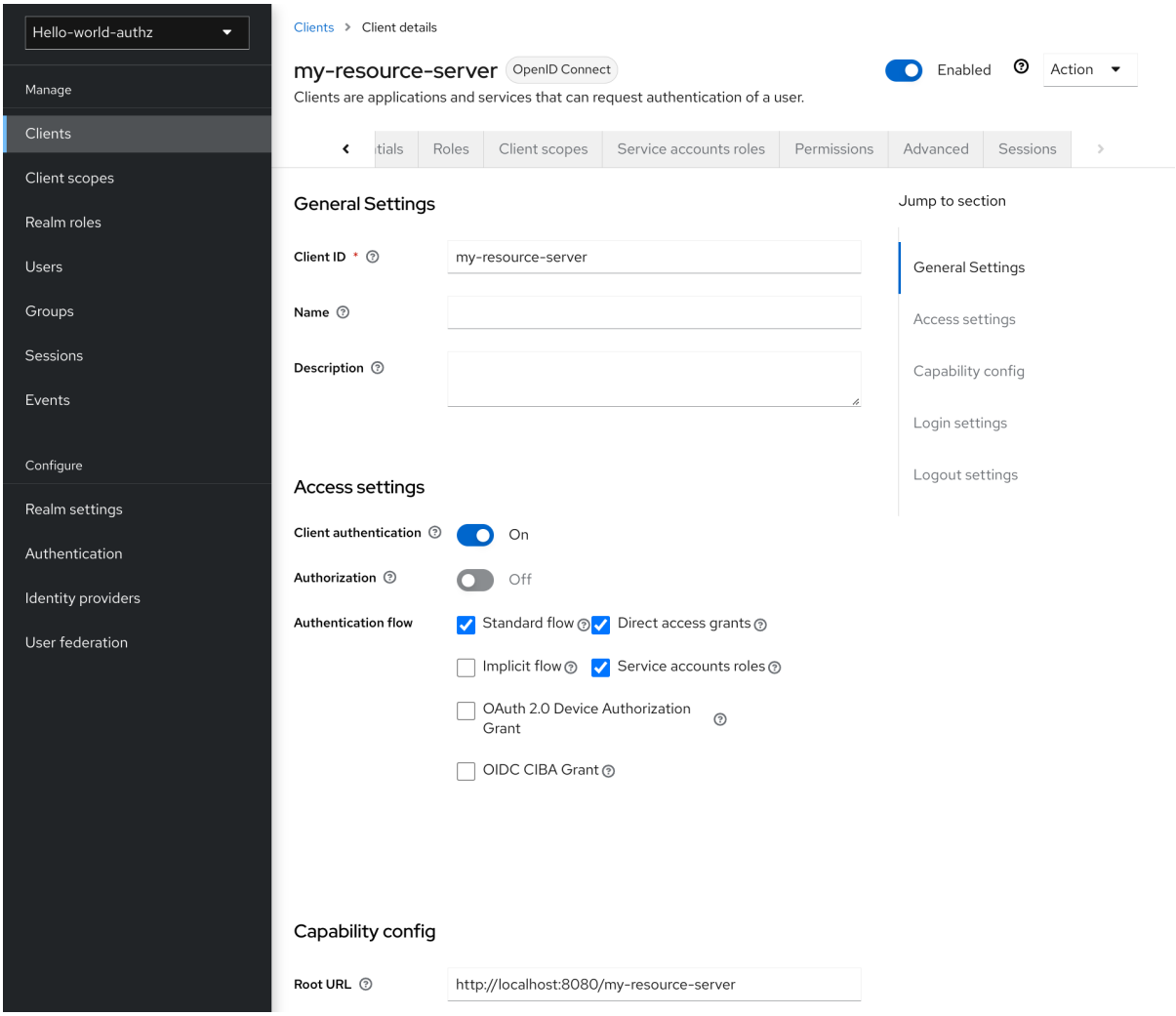


3. クライアントの **クライアント ID** を入力します。たとえば、**my-resource-server** です。
4. アプリケーションの **ルート URL** を入力します。以下に例を示します。

```
http://${host}:${port}/my-resource-server
```

5. **Save** をクリックします。クライアントが作成され、クライアントの Settings ページが開きます。以下のようなページが表示されます。

クライアントの設定



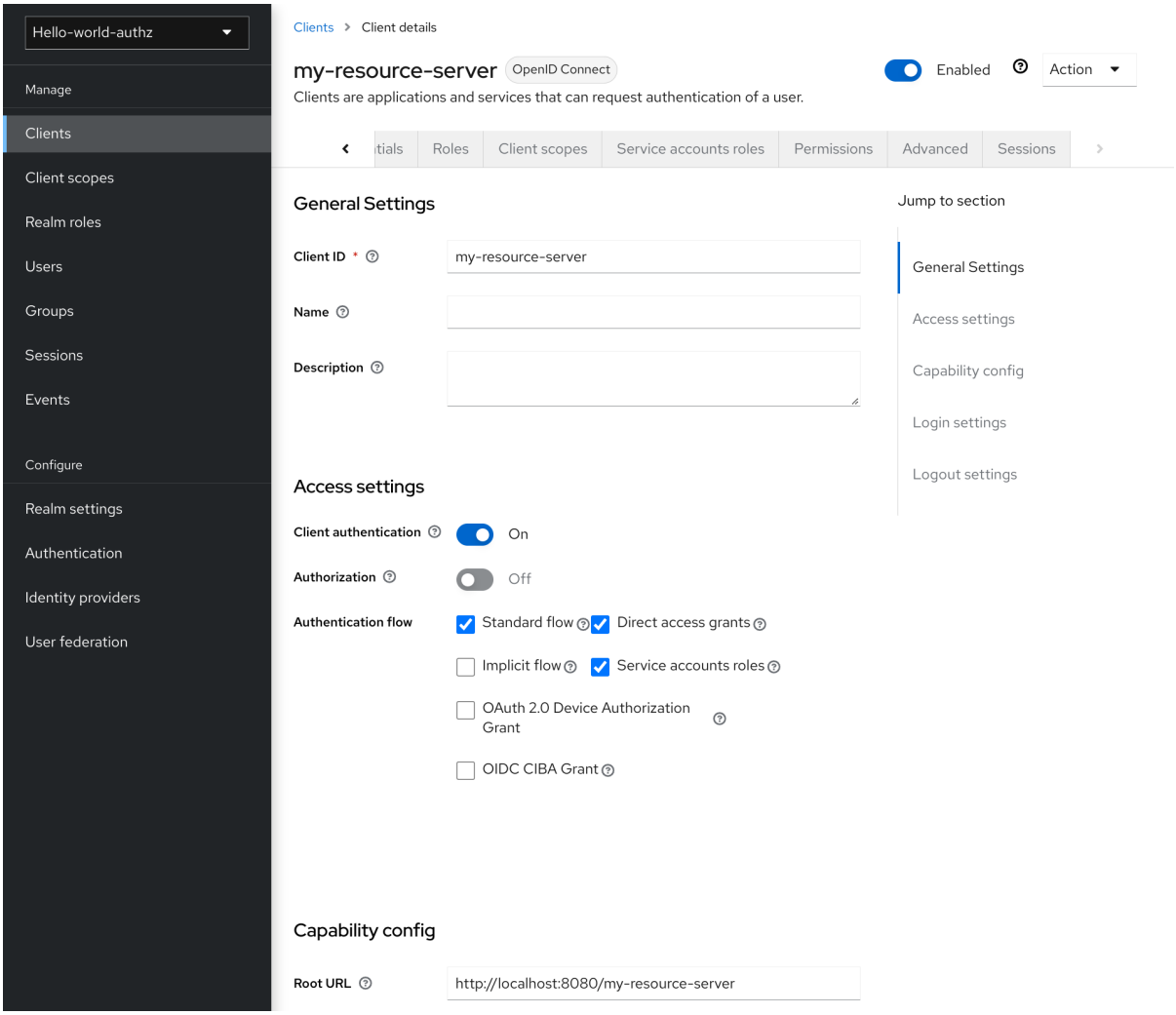
3.2. 認可サービスの有効化

OIDC クライアントをリソースサーバーに切り替え、きめ細かい承認を有効にできます。

手順

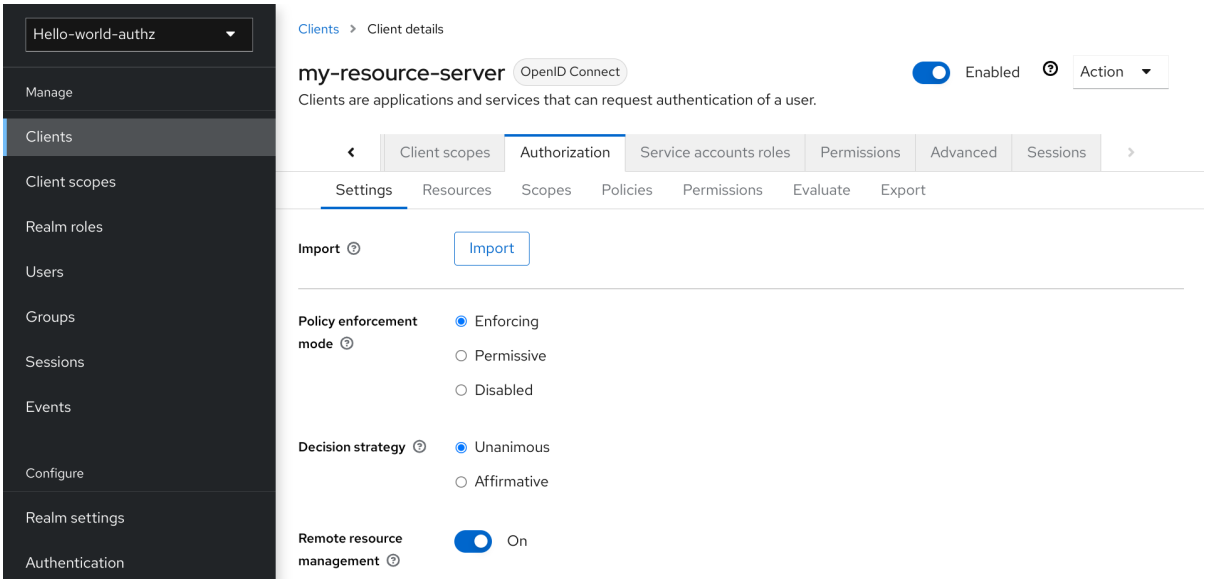
1. **Authorization Enabled** を `On に切り替えます。
2. **Save** をクリックします。

認可サービスの有効化



このクライアントの新しい Authorization タブが表示されます。**Authorization** タブをクリックして、以下のようなページが表示されます。

リソースサーバーの設定



Authorization タブには、アプリケーションのリソースを実際に保護するために従うさまざまな手順をカバーする追加のサブタブが含まれます。各タブについては、本書では特定のトピックで別個に扱われます。ただし、以下はそれぞれの詳細について簡単に説明しています。

- **設定**

リソースサーバーの一般的な設定。このページの詳細は、リソースサーバー設定 セクションを参照してください。

- **リソース**
  このページから、アプリケーションの リソース を管理できます。

- **認可スコープ**
  このページでは、スコープ を管理できます。

- **ポリシー**
  このページでは、認可ポリシー を管理し、パーミッションを付与するために満たさなければならない条件を定義できます。

- **パーミッション**
  このページから、作成したポリシーにリンクすることで、保護されているリソースおよびスコープの パーミッション を管理できます。

- **評価**
  このページでは、認可要求をシミュレート し、定義したパーミッションおよび認可ポリシーの評価結果を表示できます。

- **エクスポート設定**
  このページから、認可設定を JSON ファイルに エクスポート できます。

3.2.1. リソースサーバーの設定

リソースサーバーの設定ページでは、ポリシー実施モードの設定、リモートリソース管理の許可、認可設定のエクスポートを行うことができます。

- **ポリシー強制モード**
  サーバーに送信される認可要求の処理時にポリシーを適用する方法を指定します。

  - Enforcing
    (デフォルトモード) 指定されたリソースに関連付けられたポリシーがない場合でも、要求はデフォルトで拒否されます。

  - Permissive
    指定のリソースに関連付けられたポリシーがない場合でも、要求は許可されます。

  - Disabled
    すべてのポリシーの評価を無効にし、すべてのリソースへのアクセスを許可します。

- Decision Strategy
  この設定により、ポリシー評価エンジンが、評価されたすべてのパーミッションの結果に基づいてリソースまたはスコープが付与されるかを決定する方法が変更されます。**Affirmative** とは、リソースとそのスコープへのアクセスを付与するために、少なくとも 1 つのパーミッションが正として評価される必要があることを意味します。**Unanimous** とは、最終的な決定が正になるために、すべてのパーミッションが正として評価される必要があることを意味します。たとえば、同じリソースまたはスコープの 2 つのパーミッションが競合している場合 (その 1 つがアクセスを付与し、もう 1 つがアクセスを拒否している場合)、選択ストラテジーが **Affirmative** の場合は、リソースまたはスコープの権限が付与されます。それ以外の場合は、すべてのパーミッションから単一の拒否も、リソースまたはスコープへのアクセスを拒否します。

- **リモートリソースの管理**

リソースサーバーでリソースをリモートで管理できるかどうかを指定します。false の場合、リソースは管理コンソールからのみ管理できます。

3.3. デフォルト設定

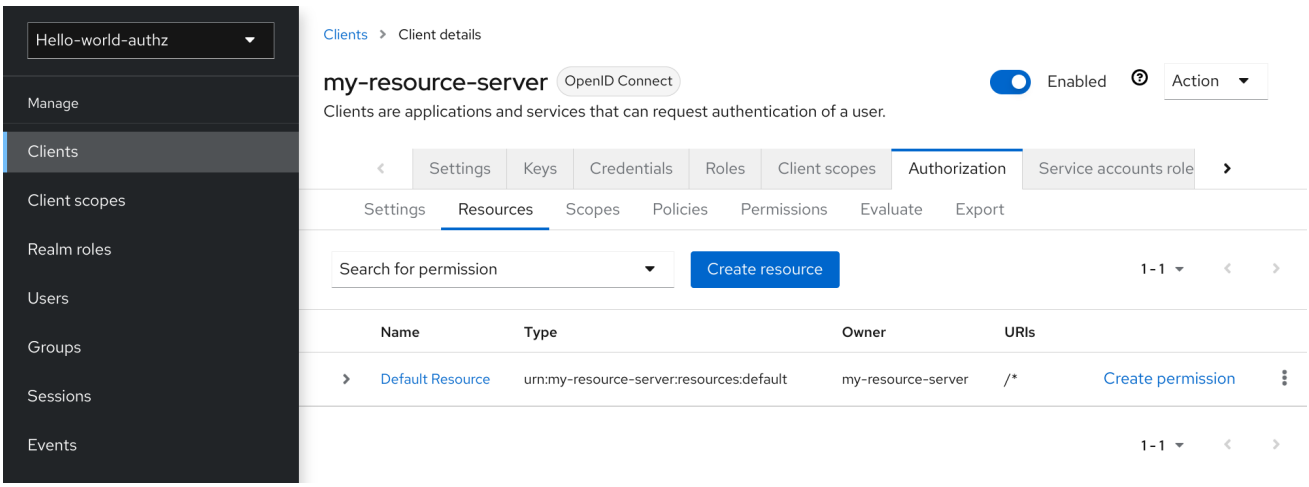
リソースサーバーを作成すると、Red Hat build of Keycloak は新たに作成されたリソースサーバーのデフォルト設定を作成します。

デフォルト設定は以下で設定されています。

- アプリケーションのすべてのリソースを表すデフォルトの保護リソース。
- このポリシーで保護されたリソースへのアクセスを常に許可するポリシー。
- デフォルトポリシーに基づいてすべてのリソースへのアクセスを管理するパーミッション。

デフォルトの保護リソースは **デフォルトのリソース** と呼ばれ、**Resources** タブに移動すると表示できます。

デフォルトのリソース

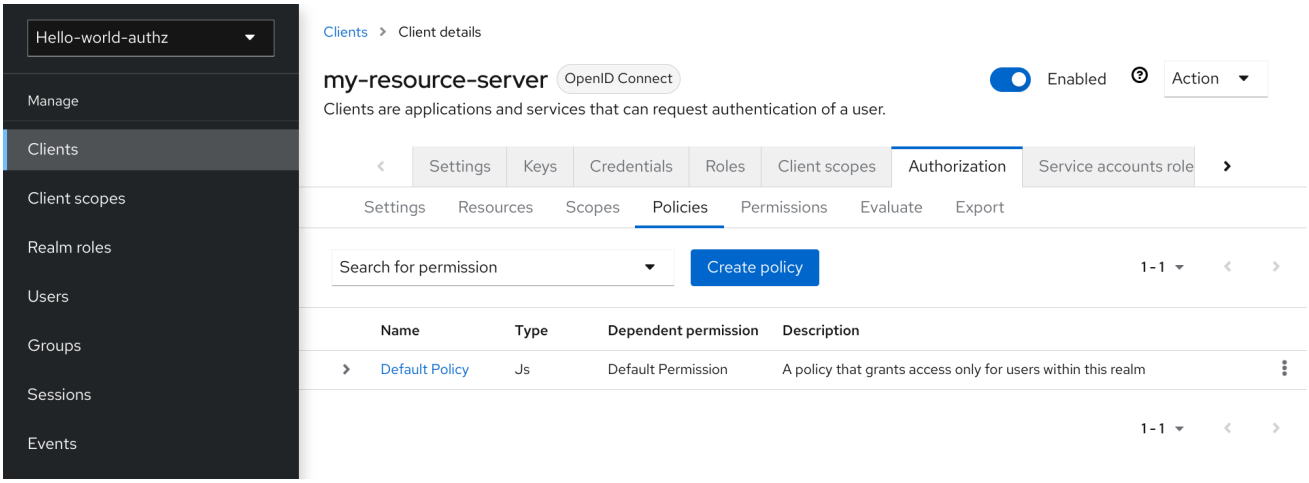


このリソースは **Type**、具体的には **urn:my-resource-server:resources:default** および **URI** /* を定義します。ここで、**URI** フィールドは、このリソースがアプリケーション内のすべてのパスを表すことを Red Hat build of Keycloak に示すワイルドカードパターンを定義します。つまり、アプリケーションのポリシー強制 を有効にすると、アクセスを付与する前にリソースに関連するすべてのパーミッションが評価されます。

前述の **Type** は、デフォルトのリソースまたは同じタイプを使用して作成するその他のリソースに適用する必要のある タイプ化されたリソースパーミッション の作成に使用できる値を定義します。

デフォルトのポリシーは **レルムポリシーからのみ** 参照され、**Policies** タブに移動すると表示できます。

デフォルトポリシー

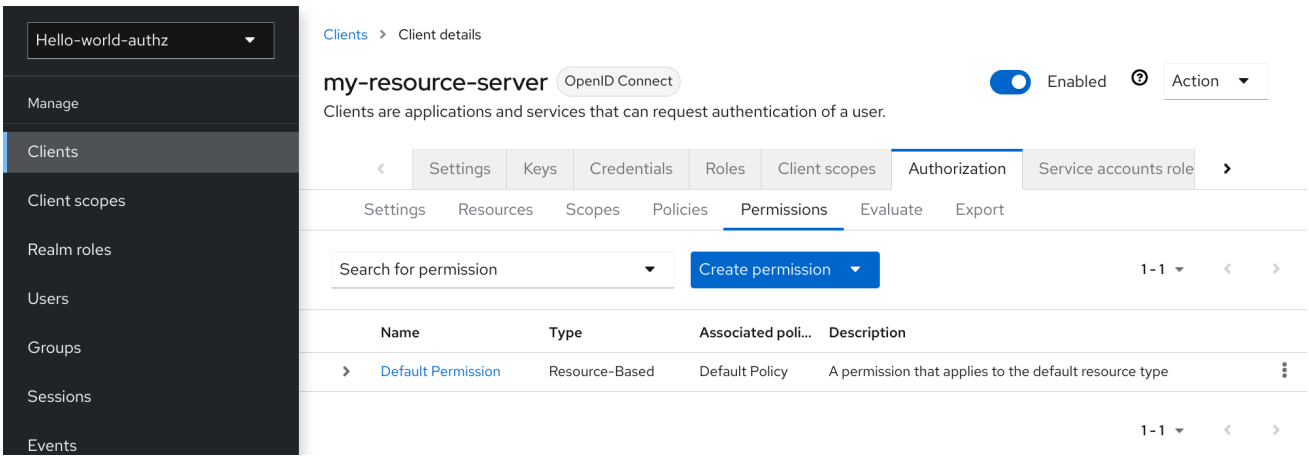


このポリシーは JavaScript ベースのポリシー で、このポリシーによって保護されるリソースへのアクセスを常に付与する条件を定義します。このポリシーをクリックすると、以下のようにルールが定義されていることを確認できます。

```
// by default, grants any permission associated with this policy
$evaluation.grant();
```

最後に、デフォルトのパーミッションは **デフォルトのパーミッション** と呼ばれ、**Permissions** タブに移動すると表示できます。

デフォルトのパーミッション



このパーミッションは リソースベースのパーミッション で、特定のタイプを持つすべてのリソースに適用される 1 つ以上のポリシーのセットを定義します。

3.3.1. デフォルト設定の変更

デフォルトの設定を変更するには、デフォルトのリソース、ポリシー、パーミッションの定義を削除し、独自の定義を作成します。

デフォルトのリソースは、/* パターンを使用してアプリケーションのリソースまたはパスにマップする **URI** で作成されます。独自のリソース、パーミッションおよびポリシーを作成する前に、デフォルト設定が独自の設定と競合しないようにしてください。

注記

デフォルト設定は、アプリケーションのすべてのパスにマップするリソースを定義します。独自のリソースにパーミッションを書き込む場合は、**デフォルトリソース** を削除するか、その **URIS** フィールドをアプリケーションのより具体的なパスに変更してください。それ以外の場合は、デフォルトのリソースに関連付けられたポリシー (デフォルトではアクセスを許可します) に基づき、Red Hat build of Keycloak が保護されているリソースへのアクセスを許可します。

3.4. 認可設定のエクスポートおよびインポート

リソースサーバー (またはクライアント) の設定をエクスポートおよびダウンロードすることができます。また、リソースサーバーの既存の設定ファイルをインポートすることもできます。設定ファイルのインポートおよびエクスポートは、リソースサーバーの初期設定を作成するか、既存の設定を更新する場合に便利です。設定ファイルには以下の定義が含まれます。

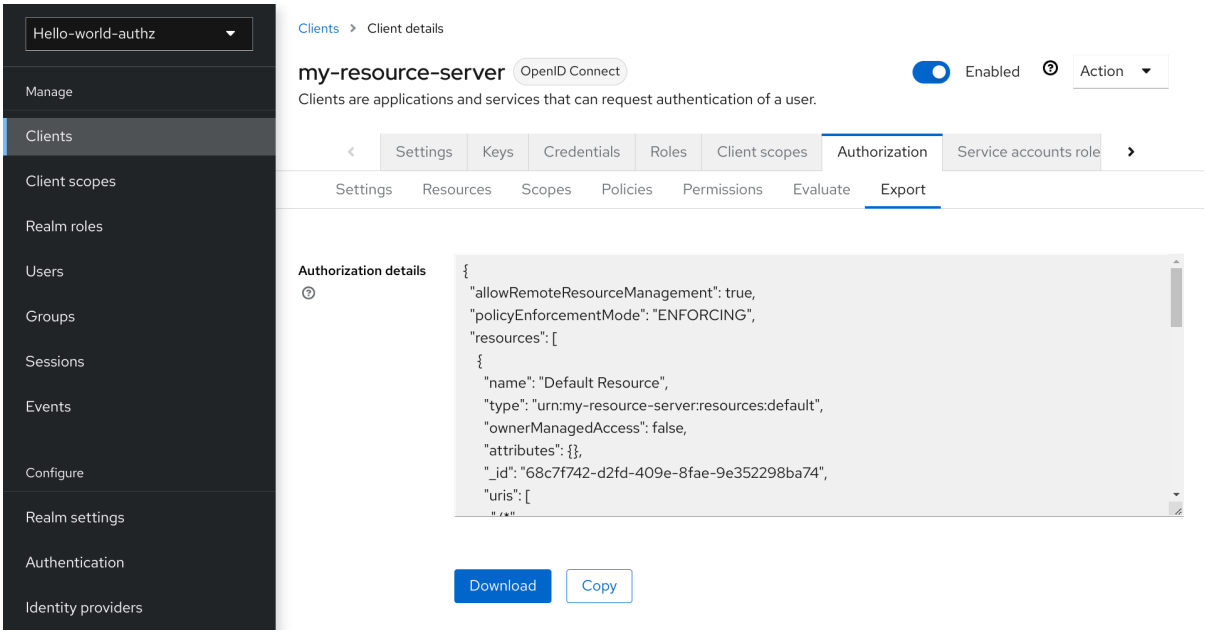
- 保護リソースおよびスコープ
- ポリシー
- 権限

3.4.1. 設定ファイルのエクスポート

手順

1. メニューで **Clients** をクリックします。
2. リソースサーバーとして作成したクライアントをクリックします。
3. **Export** タブをクリックします。

エクスポート設定



設定ファイルは JSON 形式でエクスポートされ、テキストエリアに表示され、コピーアンドペーストできます。**Download** をクリックして設定ファイルをダウンロードし、保存することもできます。

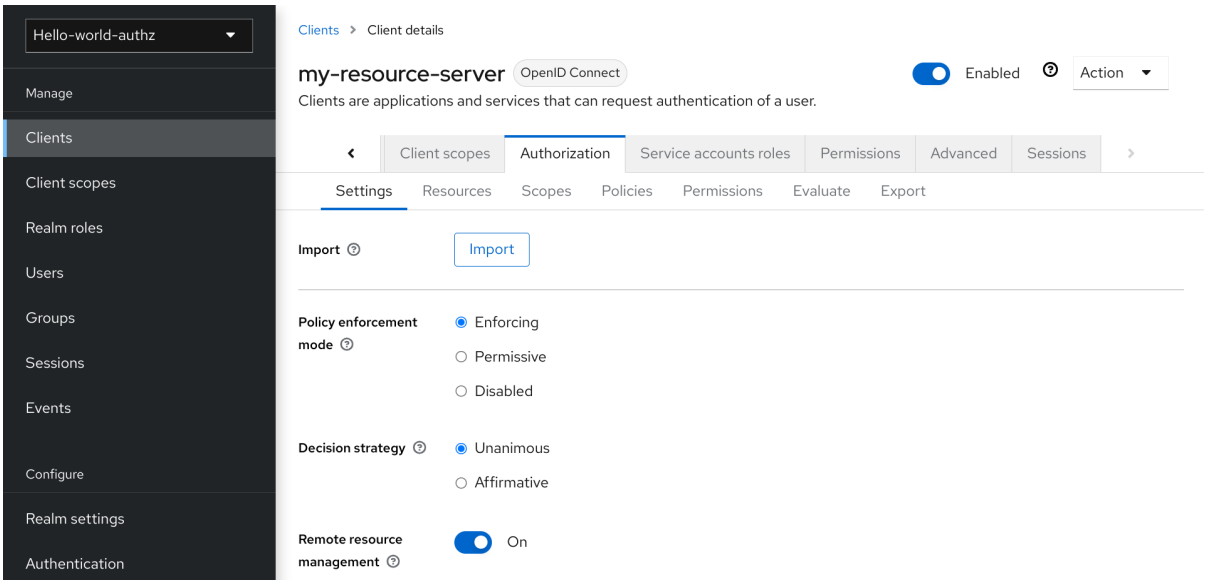
3.4.2. 設定ファイルのインポート

リソースサーバーの設定ファイルをインポートすることができます。

手順

1. **Resource Server Settings** ページに移動します。

インポート設定



2. **Import** をクリックし、インポートする設定が含まれるファイルを選択します。

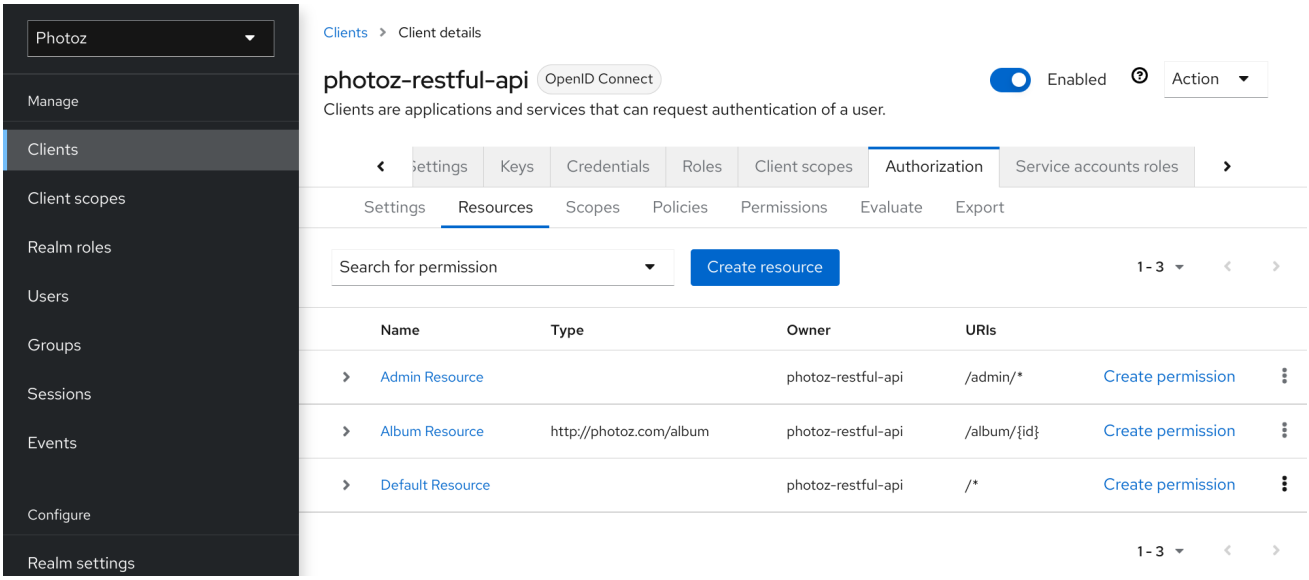
第4章 リソースおよびスコープの管理

リソース管理は簡単で汎用的です。リソースサーバーの作成後、保護するリソースおよびスコープの作成を開始できます。リソースおよびスコープは、それぞれ **Resource** タブおよび **Authorization Scopes** タブに移動して管理できます。

4.1. リソースの表示

Resource ページに、リソースサーバーに関連するリソースのリストが表示されます。

Resources



リソースリストは、以下のような保護されているリソースに関する情報を提供します。

- 種類
- URI
- Owner
- 関連するスコープ (存在する場合)
- 関連付けられたパーミッション

このリストから、パーミッションを作成するリソースの **Create Permission** をクリックして直接パーミッションを作成することができます。

注記

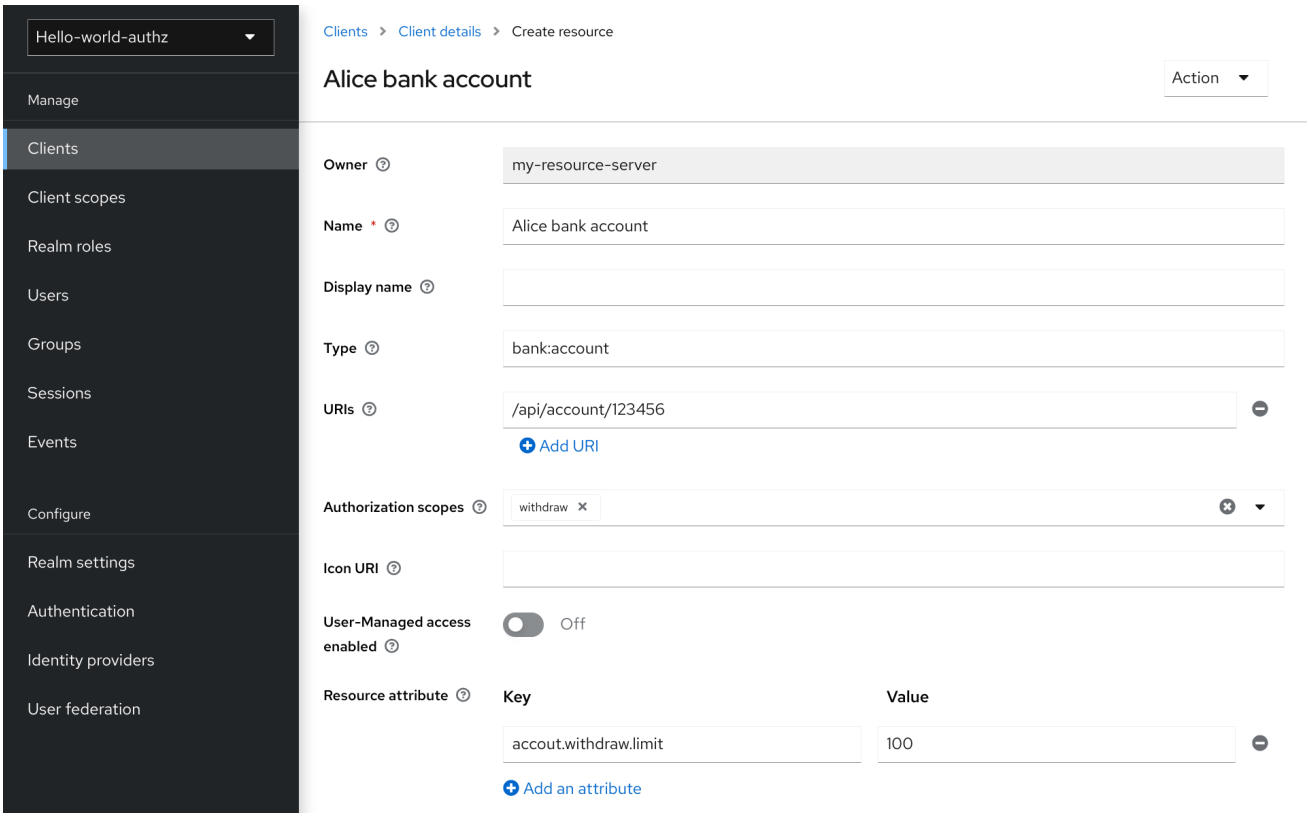
リソースのパーミッションを作成する前に、パーミッションに関連付けるポリシーがすでに定義されていることを確認してください。

4.2. リソースの作成

リソースの作成は簡単で汎用性があります。主な懸念は、作成するリソースの粒度です。つまり、1つ以上のリソースのセットを表すリソースを作成できます。また、リソースの定義方法は、パーミッションを管理する上で重要です。

新しいリソースを作成するには、**Create resource** をクリックします。

リソースの追加



Red Hat build of Keycloak では、リソースは、以下のような異なるタイプのリソースに共通する小さな情報セットを定義します。

- **名前**
  このリソースを説明する、人間が判読可能な一意の文字列。

- **種類**
  1 つ以上のリソースのセットのタイプを識別する文字列。種類は、異なるリソースインスタンスをグループ化するために使用される **文字列** です。たとえば、自動的に作成されるデフォルトリソースのデフォルトタイプは **urn:resource-server-name:resources:default** です。

- **URI**
  リソースの場所/アドレスを提供する URIS。通常、HTTP リソースの場合、URIS はこれらのリソースを提供するために使用される相対パスになります。

- **スコープ**
  リソースに関連付ける 1 つ以上のスコープ。

4.2.1. resource 属性

リソースには、関連する属性を指定できます。これらの属性を使用すると、リソースに関する追加の情報を提供でき、リソースに関連付けられたパーミッションを評価する際にポリシーに関する追加情報が提供されます。

各属性はキーと値のペアで、値は 1 つ以上の文字列のセットになります。各値をコンマで区切ることにより、属性に複数の値を定義できます。

4.2.2. 型付けされたリソース

リソースのタイプフィールドを使用してさまざまなリソースをグループ化できるため、共通のパーミッションセットを使用して保護できます。

4.2.3. リソースオーナー

リソースには所有者もあります。デフォルトでは、リソースはリソースサーバーによって所有されます。

ただし、リソースもユーザーに関連付けることができるため、リソースの所有者に基づいてパーミッションを作成できます。たとえば、リソースの所有者のみが指定のリソースを削除または更新できます。

4.2.4. リソースのリモート管理

また、リソース管理は 保護 API を通じて公開され、リソースサーバーはリソースをリモートで管理できます。

保護 API を使用する場合、ユーザーが所有するリソースを管理するためにリソースサーバーを実装できます。この場合は、ユーザー ID を指定して、特定のユーザーに属するリソースを設定できます。

注記

Red Hat build of Keycloakは、リソースサーバーがリソースを完全に制御できるようにします。今後は、特に UMA プロトコルを使用する場合に、ユーザーが独自のリソースを制御したり、認可要求を承認したり、パーミッションを管理したりできるようにする必要があります。

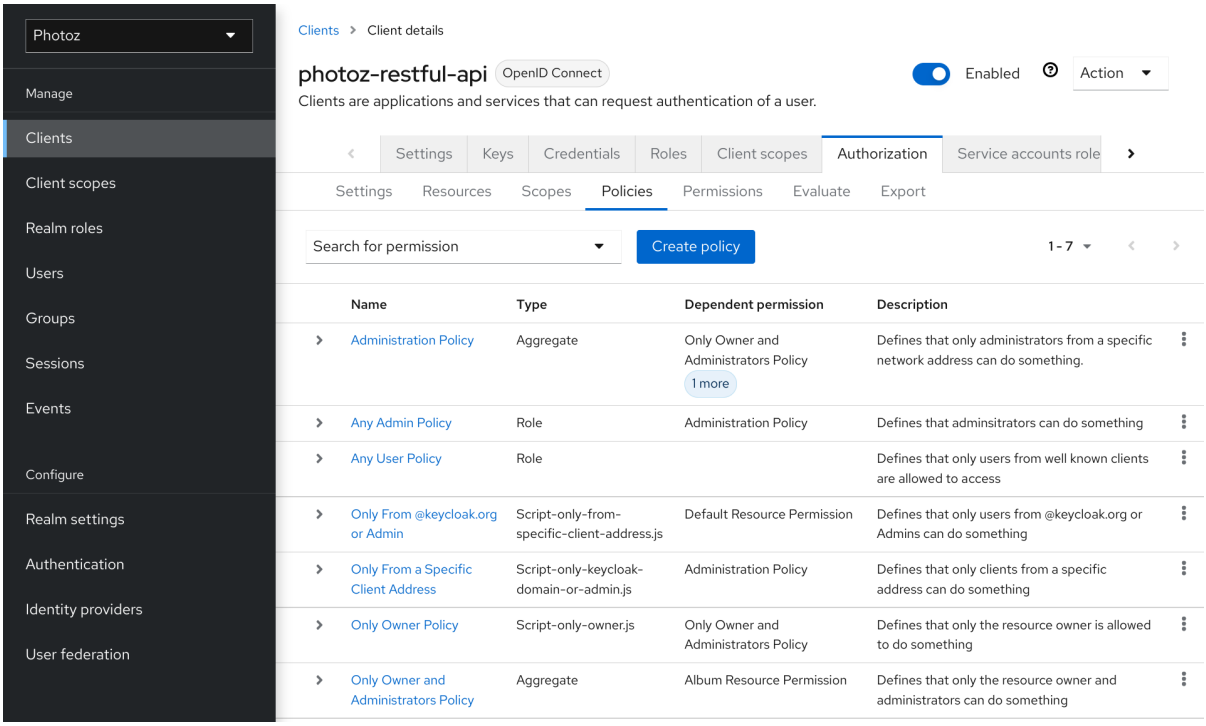
第5章 ポリシーの管理

前述のように、ポリシーは、オブジェクトへのアクセス権限を付与する前に満たさなければならない条件を定義します。

手順

1. **Policy** タブをクリックして、リソースサーバーに関連付けられたポリシーをすべて表示します。

ポリシー



このタブでは、以前に作成したポリシーのリストを表示し、ポリシーの作成および編集を行うことができます。

新しいポリシーを作成するには、**Create policy** をクリックし、リストからポリシータイプを選択します。

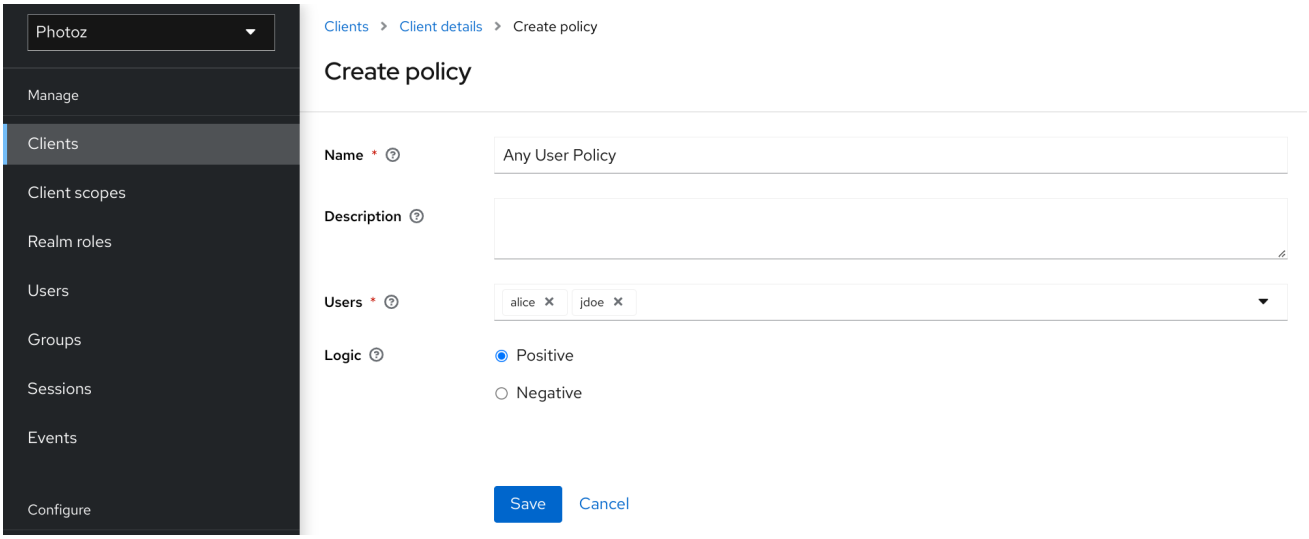
本セクションでは、各ポリシー種別の詳細を説明します。

5.1. ユーザーベースのポリシー

このタイプのポリシーを使用して、1 人以上のユーザーセットがオブジェクトにアクセスできるパーミッションの条件を定義できます。

新しいユーザーベースのポリシーを作成するには、ポリシーリストの右上隅にあるドロップダウンリストで **User** を選択します。

ユーザーポリシーの追加



5.1.1. 設定

- **Name**
  ポリシーを識別する、人間が判読可能な一意の文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このポリシーの詳細を含む文字列。

- **Users**
  このポリシーによりアクセスが付与されるユーザーを指定します。

- **Logic**
  他の条件を評価した後に適用する、このポリシーのロジック。

関連情報

- 正ロジックと負ロジック

5.2. ロールベースのポリシー

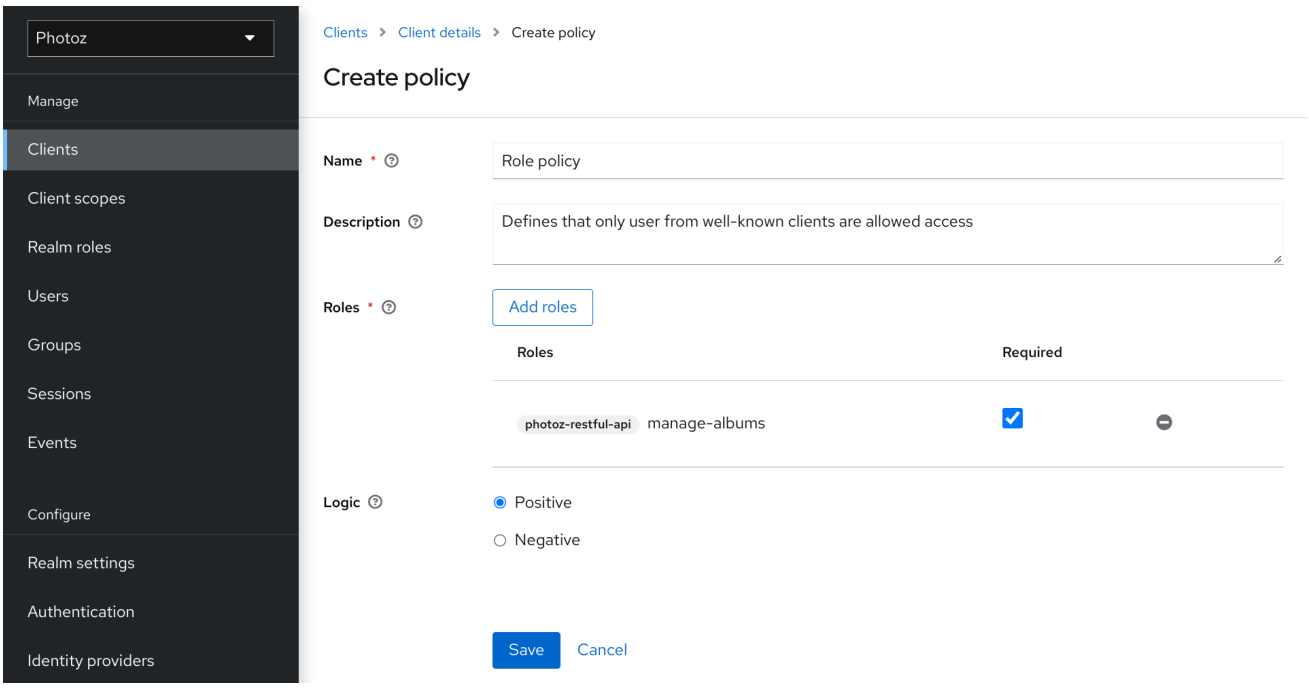
このタイプのポリシーを使用して、1 つ以上のロールセットがオブジェクトにアクセスできるパーミッションの条件を定義できます。

デフォルトでは、このポリシーに追加されたロールは必要に応じて指定されず、アクセスを要求するユーザーがこのロールが付与されている場合は、ポリシーはアクセスを付与します。ただし、特定のロールを強制する場合は、必要 に応じて特定のロールを指定することができます。レルムまたはクライアントロールのどちらであるかに関係なく、必須ロールと必須ロールを組み合わせることもできます。

ロールポリシーは、より多くの制限付きのロールベースのアクセス制御 (RBAC) が必要な場合に役に立ちます。オブジェクトへのアクセス権限を付与するために特定のロールを適用する必要がある場合に役立ちます。たとえば、ユーザーがユーザーのリソースにアクセスするために (ユーザーの代わりに) クライアントアプリケーションを許可する必要があります。Red Hat build of Keycloak の Client Scope Mapping を使用して競合ページを有効にしたり、クライアントが Red Hat build of Keycloak サーバーからアクセストークンを取得する際にスコープを明示的に指定することもできます。

新しいロールベースのポリシーを作成するには、ポリシータイプリストから **Role** を選択します。

ロールポリシーの追加



5.2.1. 設定

- **Name**
  ポリシーを記述する人間が判読できる一意な文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このポリシーの詳細を含む文字列。

- **Realm Roles**
  このポリシーで許可される **realm** ロールを指定します。

- **Client Roles**
  このポリシーで許可される **client** ロールを指定します。このフィールドを有効にするには、最初に **Client** を選択します。

- **Logic**
  他の条件を評価した後に適用する、このポリシーのロジック。

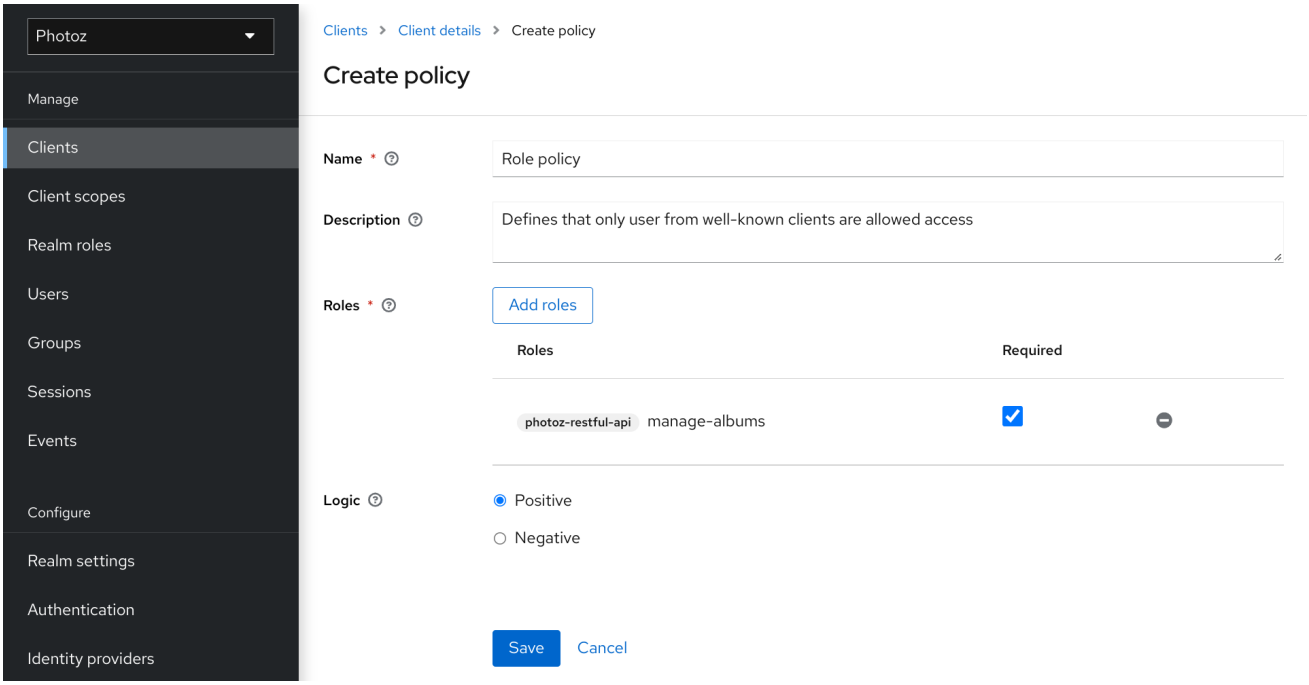
関連情報

- 正ロジックと負ロジック

5.2.2. ロールの定義 (任意)

ロールベースのポリシーの作成時に、特定のロールを **Required** として指定できます。これを実行する際に、アクセスを要求するユーザーが **すべて** の **必要な** ロールが付与されている場合にのみ、ポリシーはアクセスを付与します。レルムとクライアントロールの両方を設定できます。

必要なロールの例



必要に応じてロールを指定するには、必要に応じて設定するロールの **Required** チェックボックスを選択します。

必要なロールは、ポリシーに複数のロールを定義する場合に役に立ちますが、それらのサブセットのみが必須となります。この場合、レルムとクライアントロールを組み合わせて、アプリケーションに対してより粒度の細かいロールベースアクセス制御 (RBAC) モデルを有効にすることができます。たとえば、クライアントに固有のポリシーを持つことができ、そのクライアントに関連付けられた特定のクライアントロールが必要です。また、アクセスを強制できるのは、特定のレルムロールの有無のみです。また、両方のアプローチを同じポリシーに組み合わせることもできます。

5.3. JAVASCRIPT ベースのポリシー

警告

以下の例のように、ポリシーの実装で属性ベースのアクセス制御 (ABAC) を使用している場合は、ユーザーが保護された属性を編集できず、対応する属性が読み取り専用であることを確認してください。詳細は、脅威モデルの軽減 を参照してください。

このタイプのポリシーを使用して、JavaScript を使用してパーミッションの条件を定義することができます。Red Hat build of Keycloak でサポートされるルールベースのポリシータイプの 1 つで、評価 API に基づいて任意のポリシーを作成する柔軟性を提供します。

新しい JavaScript ベースのポリシーを作成するには、ポリシーリストの右上隅にある項目リストで **JavaScript** を選択します。

注記

デフォルトでは、JavaScript ポリシーはサーバーにアップロードできません。JavaScript プロバイダー で説明されているように、JS ポリシーを直接サーバーにデプロイすることが推奨されます。

5.3.1. デプロイ済みの JAR ファイルからの JS ポリシーの作成

Red Hat build of Keycloak では、JAR ファイルをデプロイして、スクリプトをサーバーにデプロイできます。詳細は、JavaScript プロバイダー を参照してください。

スクリプトをデプロイしたら、利用可能なポリシープロバイダーのリストからデプロイしたスクリプトを選択できるはずです。

5.3.2. 例

5.3.2.1. 評価コンテキストの属性の確認

以下は、属性ベースのアクセス制御 (ABAC) を使用して、実行コンテキストから取得した属性に基づいて条件を定義する JavaScript ベースのポリシーの簡単な例です。

```
const context = $evaluation.getContext();
const contextAttributes = context.getAttributes();

if (contextAttributes.containsValue('kc.client.network.ip_address', '127.0.0.1')) {
    $evaluation.grant();
}
```

5.3.2.2. 現在のアイデンティティーの属性の確認

以下は、属性ベースのアクセス制御 (ABAC) を使用して、現在のアイデンティティーに関連付けられた属性に基づいて条件を定義する JavaScript ベースのポリシーの簡単な例です。

```
const context = $evaluation.getContext();
const identity = context.getIdentity();
const attributes = identity.getAttributes();
const email = attributes.getValue('email').asString(0);

if (email.endsWith('@keycloak.org')) {
    $evaluation.grant();
}
```

これらの属性は、認可要求で使用されたトークンで定義された要求からマッピングされます。

5.3.2.3. 現在のアイデンティティーに付与されるロールの確認

ポリシーでロールベースアクセス制御 (RBAC) を使用することもできます。以下の例では、ユーザーに **keycloak_user** realm ロールが付与されているかどうかを確認します。

```
const context = $evaluation.getContext();
const identity = context.getIdentity();
```

```
if (identity.hasRealmRole('keycloak_user')) {
    $evaluation.grant();
}
```

または、ユーザーに **my-client-role** client ロールが付与されているかどうかを確認できます。ここでは、**my-client** はクライアントアプリケーションのクライアント ID になります。

```
const context = $evaluation.getContext();
const identity = context.getIdentity();

if (identity.hasClientRole('my-client', 'my-client-role')) {
    $evaluation.grant();
}
```

5.3.2.4. ユーザーに付与されているロールの確認

ユーザーに付与されたレルムロールを確認するには、以下を実行します。

```
const realm = $evaluation.getRealm();

if (realm.isUserInRealmRole('marta', 'role-a')) {
    $evaluation.grant();
}
```

ユーザーに付与されているクライアントロールの場合は、以下を実行します。

```
const realm = $evaluation.getRealm();

if (realm.isUserInClientRole('marta', 'my-client', 'some-client-role')) {
    $evaluation.grant();
}
```

5.3.2.5. グループに付与されているロールの確認

グループに付与されたレルムロールを確認するには、以下を実行します。

```
const realm = $evaluation.getRealm();

if (realm.isGroupInRole('/Group A/Group D', 'role-a')) {
    $evaluation.grant();
}
```

5.3.2.6. 任意のクレームをリソースサーバーへプッシュする

任意の要求をリソースサーバーにプッシュし、パーミッションの適用方法に関する追加情報を提供します。

```
const permission = $evaluation.getPermission();

// decide if permission should be granted

if (granted) {
    permission.addClaim('claim-a', 'claim-a');
```

```
    permission.addClaim('claim-a', 'claim-a1');
    permission.addClaim('claim-b', 'claim-b');
}
```

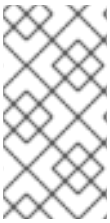
5.3.2.7. グループメンバーシップの確認

```
const realm = $evaluation.getRealm();

if (realm.isUserInGroup('marta', '/Group A/Group B')) {
    $evaluation.grant();
}
```

5.3.2.8. 異なるアクセス制御メカニズムの組み合わせ

複数のアクセス制御メカニズムの組み合わせを使用することもできます。以下の例は、同じポリシー内でロール (RBAC) および Request/attributes(ABAC) チェックを使用できる方法を示しています。この場合、ユーザーに **admin** ロールが付与されているか、または **keycloak.org** ドメインからの電子メールがあるかどうかを確認します。

```
const context = $evaluation.getContext();
const identity = context.getIdentity();
const attributes = identity.getAttributes();
const email = attributes.getValue('email').asString(0);

if (identity.hasRealmRole('admin') || email.endsWith('@keycloak.org')) {
    $evaluation.grant();
}
```



注記

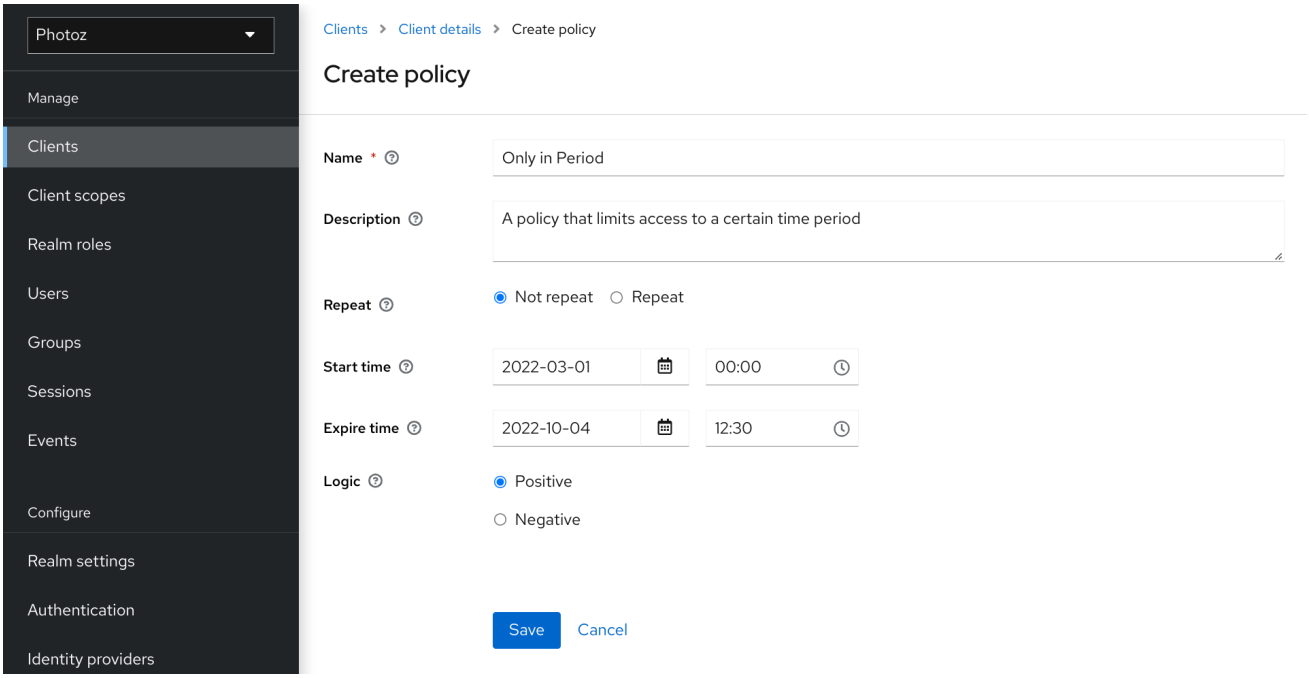
独自のルールを記述する際は、**$evaluation** オブジェクトが **org.keycloak.authorization.policy.evaluation.Evaluation** を実装するオブジェクトであることに留意してください。このインターフェイスからアクセスできる内容の詳細は、評価 API を参照してください。

5.4. 時間ベースのポリシー

このタイプのポリシーを使用して、パーミッションの時間条件を定義することができます。

新しいタイムベースのポリシーを作成するには、ポリシーリストの右上隅にある項目リストで **Time** を選択します。

時間ポリシーの追加



5.4.1. 設定

- **Name**
  ポリシーを記述する人間が判読できる一意な文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このポリシーの詳細を含む文字列。

- **開始時間**
  その前にアクセスを **許可してはならない** 時間を定義します。パーミッションには、現在の日付と時刻がこの値よりも後の日付/時刻が後で付与されます。

- **有効期限**
  アクセスを **許可してはならない** 時間を定義します。パーミッションには、現在の日付と時刻がこの値よりも前の日付またはそれ以上である場合のみ付与されます。**Repeat** を選択して、特定の **日付**、**月**、**年**、**時間**、または **分** に付与されるアクセスを繰り返します。

- **Day of Month**
  アクセスが付与される必要のある月の日付を定義します。日付の範囲を指定することもできます。この場合、パーミッションには、月の現在の日が指定された 2 つの値以下または等しい場合にのみ付与されます。

- **Month**
  アクセスを付与する必要のある月を定義します。また、月の範囲を指定することもできます。この場合、パーミッションは現在の月間または指定した 2 つの値の間のみ付与されます。

- **Year**
  アクセスを付与する必要のある年を定義します。また、年の範囲を指定することもできます。この場合、パーミッションは指定の 2 つの値の間の現在の年の間のみ付与されます。

- **Hour**
  アクセスを付与する必要のある時間を定義します。時間の範囲を指定することもできます。この場合、パーミッションは、現在の時間と指定された 2 つの値の間のみ付与されます。

- **Minute**
  アクセスを付与する必要のある分を定義します。分の範囲を指定することもできます。この場合、パーミッションは現在の分を指定した 2 つの値以上の間のみ付与されます。

- **Logic**
  他の条件を評価した後に適用する、このポリシーのロジック。

すべての条件が満たされる場合にのみ、アクセスが許可されます。Red Hat build of Keycloak は、各条件の結果に基づいて **AND** を実行します。

関連情報

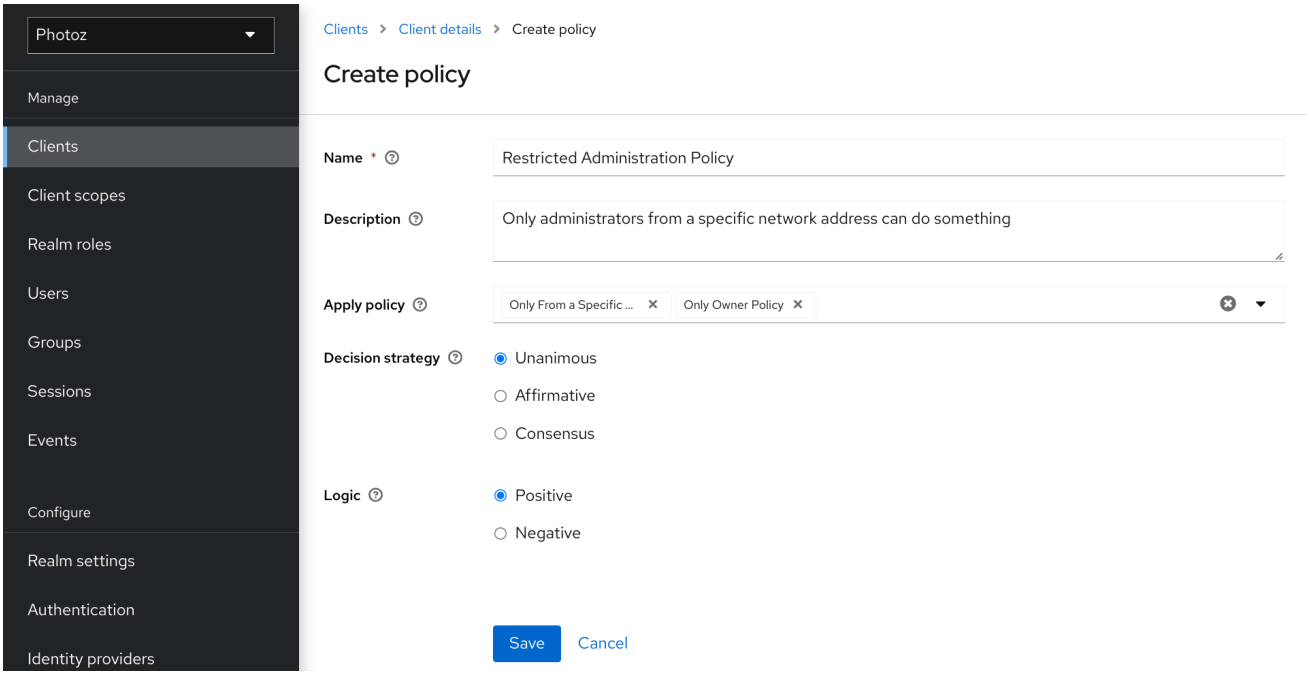
- 正ロジックと負ロジック

5.5. 集約ポリシー

前述したように、Red Hat build of Keycloak ではポリシーのポリシーを (ポリシー集約と呼ばれる概念) を作成できます。ポリシーアグリゲーションを使用して、既存のポリシーを再利用し、認可要求の処理中に評価されるポリシーからさらに分離したパーミッションを維持することができます。

新しい集約ポリシーを作成するには、ポリシーリストの右上隅にある項目リストで **集約** を選択します。

集約されたポリシーの追加



keycloak.org ドメインのユーザーのみが特定の IP アドレス範囲からアクセス可能な **Confidential Resource** というリソースがあるとします。両方の条件を指定してポリシーを 1 つ作成できます。ただし、このポリシーのドメイン部分を再利用して、送信元ネットワークに関係なく動作するパーミッションに適用します。

ドメインとネットワーク条件の両方に個別のポリシーを作成し、これら 2 つのポリシーの組み合わせに基づいて 3 つ目のポリシーを作成することができます。集約ポリシーでは、他のポリシーを自由に組み合わせ、新しい集約されたポリシーを任意のパーミッションに適用できます。

注記

集約されたポリシーを作成する場合には、ポリシー間の循環参照や依存関係が導入されていないことに注意してください。循環の依存関係が検出されると、ポリシーを作成または更新できません。

5.5.1. 設定

- **Name**
  ポリシーを記述する人間が判読できる一意な文字列。お客様のビジネスやセキュリティー要件と密接に関連した名前を使用することを強く推奨します。そうすれば、より簡単に識別でき、またその意味も理解できるでしょう。

- **Description**
  このポリシーの詳細情報が含まれる文字列。

- **Apply Policy**
  集約ポリシーに関連付ける 1 つ以上のポリシーのセットを定義します。ポリシーを関連付けるには、既存のポリシーを選択するか、作成するポリシーのタイプを選択して新規のポリシーを作成します。

- **Decision Strategy**
  このパーミッションのデシジョンストラテジー。

- **Logic**
  他の条件を評価した後に適用する、このポリシーのロジック。

関連情報

- 正ロジックと負ロジック

5.5.2. 集計されたポリシーの Decision Strategy

集計ポリシーの作成時に、各ポリシーの結果に基づいて最終的な決定決定を行うために使用されるデシジョンストラテジーを定義することもできます。

- **Unanimous**
  指定がない場合はデフォルトのストラテジーです。この場合、最終的な決定を正にするには、**すべて** のポリシーが正の決定に評価される必要があります。

- **Affirmative**
  この場合、最終的な決定を正にするには、**少なくとも 1 つ** のポリシーを正の決定に評価する必要があります。

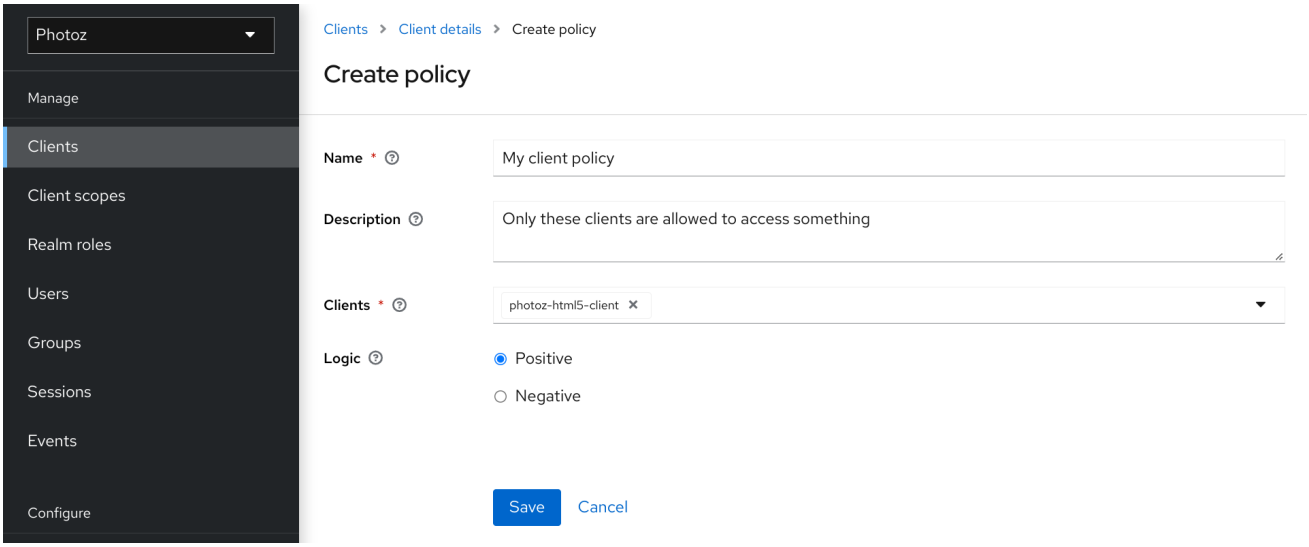
- **Consensus**
  この場合、正のデシジョン数は、負の数より大きくする必要があります。正と負のデシジョンが同じである場合、最終的な決定は負の値になります。

5.6. クライアントベースのポリシー

このタイプのポリシーを使用して、1 つ以上のクライアントセットがオブジェクトにアクセスできるパーミッションの条件を定義できます。

新しいクライアントベースのポリシーを作成するには、ポリシータイプリストから **Client** を選択します。

クライアントポリシーの追加



5.6.1. 設定

- **Name**
  ポリシーを識別する、人間が判読可能な一意の文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このポリシーの詳細を含む文字列。

- **Clients**
  このポリシーによってグループベースのポリシーアクセスが付与されたクライアントを示します。

- **Logic**
  他の条件を評価した後に適用する、このポリシーのロジック。

関連情報

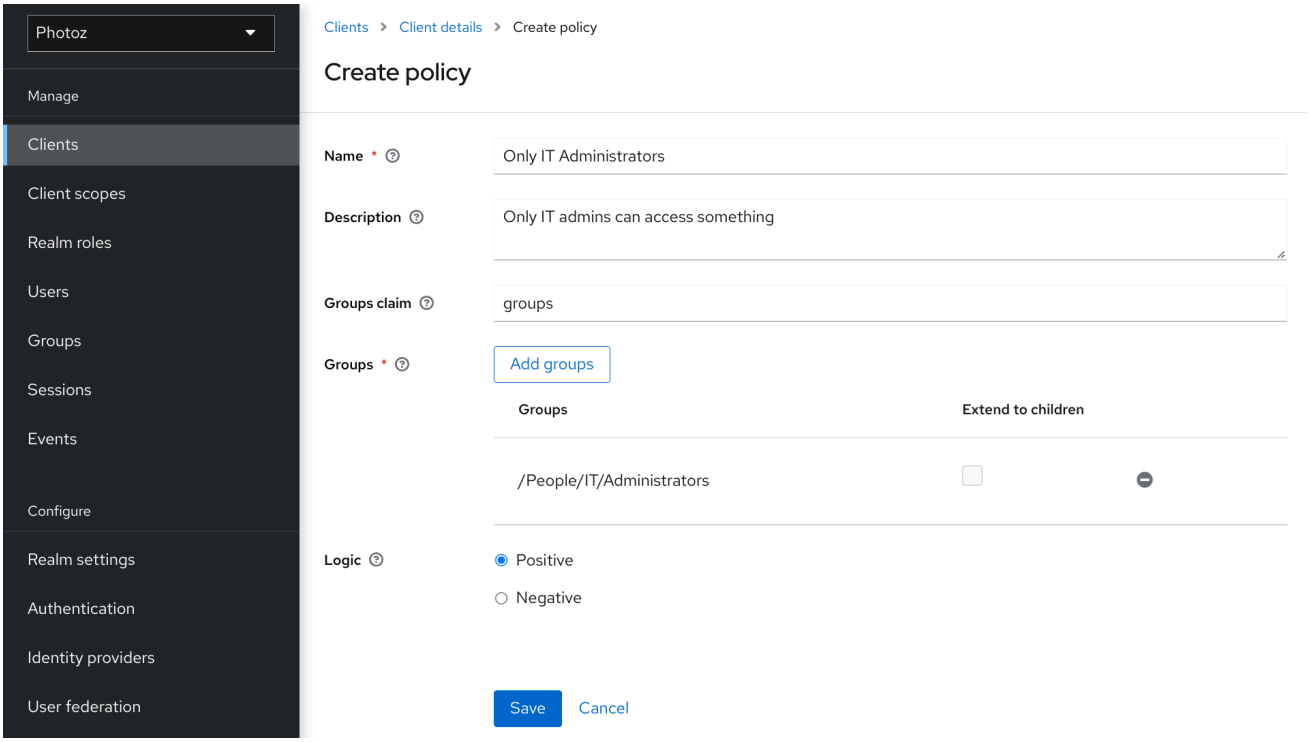
- 正ロジックと負ロジック

5.7. グループベースのポリシー

このタイプのポリシーを使用して、1つ以上のグループ (およびそれらの階層) セットがオブジェクトにアクセスできるパーミッションの条件を定義できます。

新しいグループベースのポリシーを作成するには、ポリシータイプリストから **Group** を選択します。

グループポリシー



5.7.1. 設定

- **Name**
  ポリシーを記述する人間が判読できる一意な文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このポリシーの詳細を含む文字列。

- **Groups Claim**
  グループ名またはパスを保持するトークン内の要求の名前を指定します。通常、認可要求は、これまで何らかのユーザーの代わりに動作しているクライアントに発行される ID トークンまたはアクセストークンを基に処理されます。定義された場合、トークンに、ユーザーがメンバーであるグループを取得する場合、トークンに要求を含める必要があります。定義されていない場合は、ユーザーのグループがレルム設定から取得されます。

- **Groups**
  パーミッションを評価する際に、このポリシーで強制する必要のあるグループを選択できます。グループを追加したら、**Extend to Children** チェックボックスに印を付けることで、グループの子へのアクセスを拡張できます。マークが付いていないままの場合、アクセス制限は選択したグループにのみ適用されます。

- **Logic**
  他の条件を評価した後に適用する、このポリシーのロジック。

関連情報

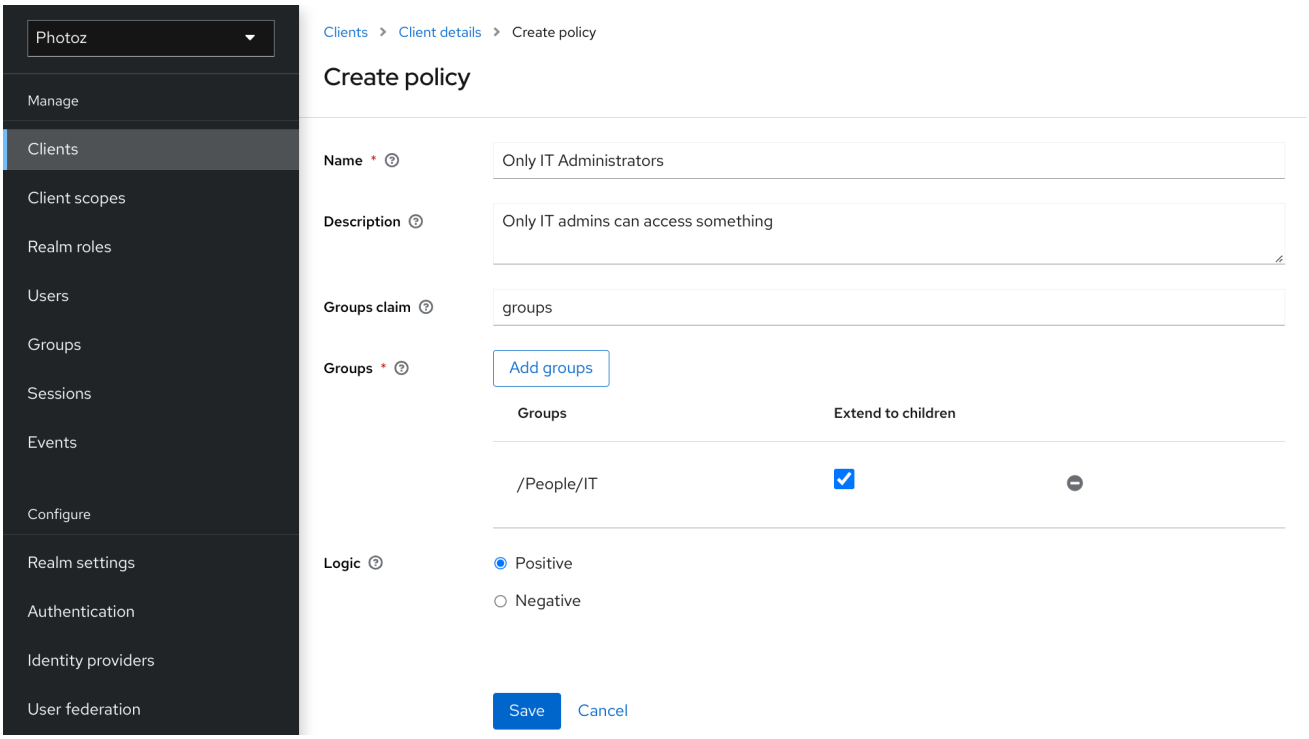
- 正ロジックと負ロジック

5.7.2. 子グループへのアクセスの拡張

デフォルトでは、このポリシーにグループを追加すると、アクセス制限が選択されたグループのメンバーにのみ適用されます。

状況によっては、グループ自体だけでなく、階層内の子グループにアクセスできなくなることがあります。追加されたグループで、子グループへのアクセスを拡張するために **Extend to Children** チェックボックスを選択できます。

子グループへのアクセスの拡張



上記の例では、このポリシーは **IT** またはその子のいずれかのユーザーメンバーにアクセス権限を付与しています。

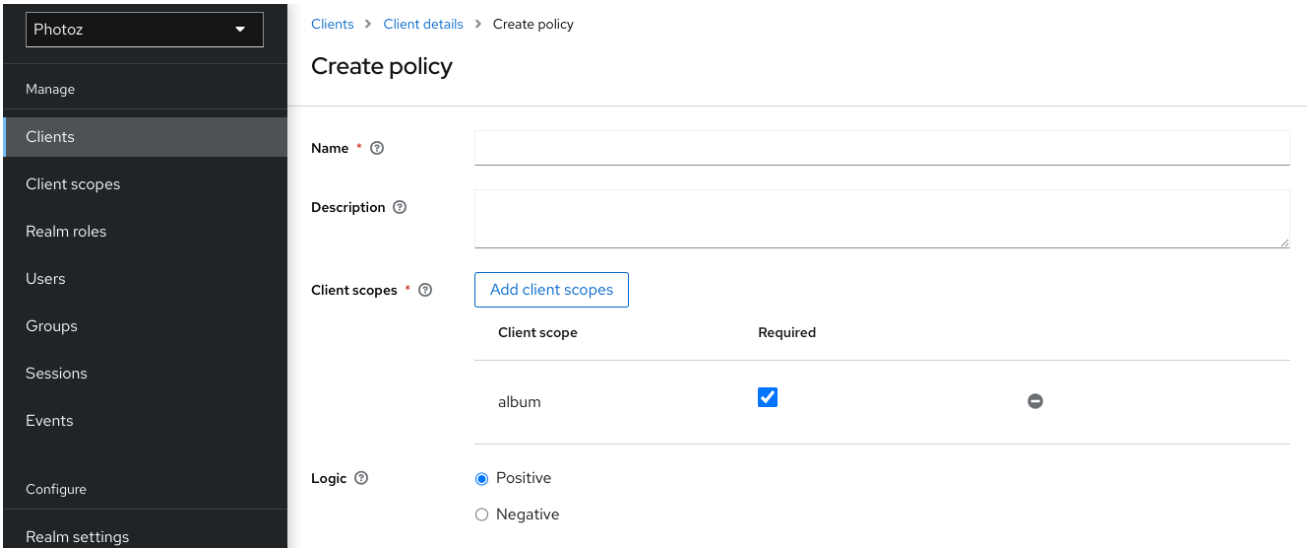
5.8. クライアントスコープベースのポリシー

このタイプのポリシーを使用して、1 つ以上のクライアントスコープのセットがオブジェクトにアクセスできるパーミッションの条件を定義できます。

デフォルトでは、このポリシーに追加されたクライアントスコープは必要に応じて指定されず、アクセスを要求するクライアントがこのクライアントスコープが付与されている場合は、ポリシーはアクセスを付与します。ただし、特定のクライアントスコープを適用する場合は、必要 に応じて特定のクライアントスコープを指定できます。

新しいクライアントスコープベースのポリシーを作成するには、ポリシータイプリストから **Client Scope** を選択します。

クライアントスコープポリシーの追加



5.8.1. 設定

- **Name**
  ポリシーを記述する人間が判読できる一意な文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このポリシーの詳細を含む文字列。

- **クライアントスコープ**
  このポリシーで許可されるクライアントスコープを指定します。

- **Logic**
  他の条件を評価した後に適用する、このポリシーのロジック。

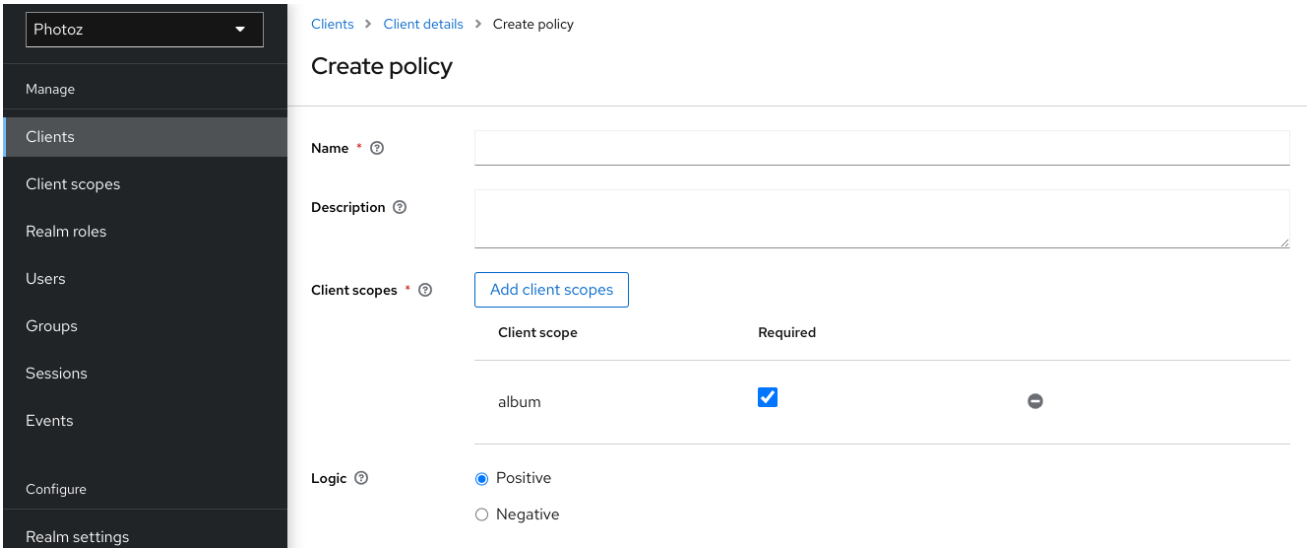
関連情報

- 正ロジックと負ロジック

5.8.2. クライアントスコープの定義 (任意)

クライアントスコープベースのポリシーを作成するときに、特定のクライアントスコープを **Required** として指定できます。これを実行する際に、アクセスを要求するクライアントが **すべて** の **必要な** クライアントスコープが付与されている場合にのみ、ポリシーはアクセスを付与します。

必要なクライアントスコープの例



必要に応じてクライアントスコープを指定するには、必要に応じて設定するクライアントスコープの **Required** チェックボックスを選択します。

必要なクライアントスコープは、ポリシーに複数のクライアントスコープを定義する場合に役に立ちますが、それらのサブセットのみが必須となります。

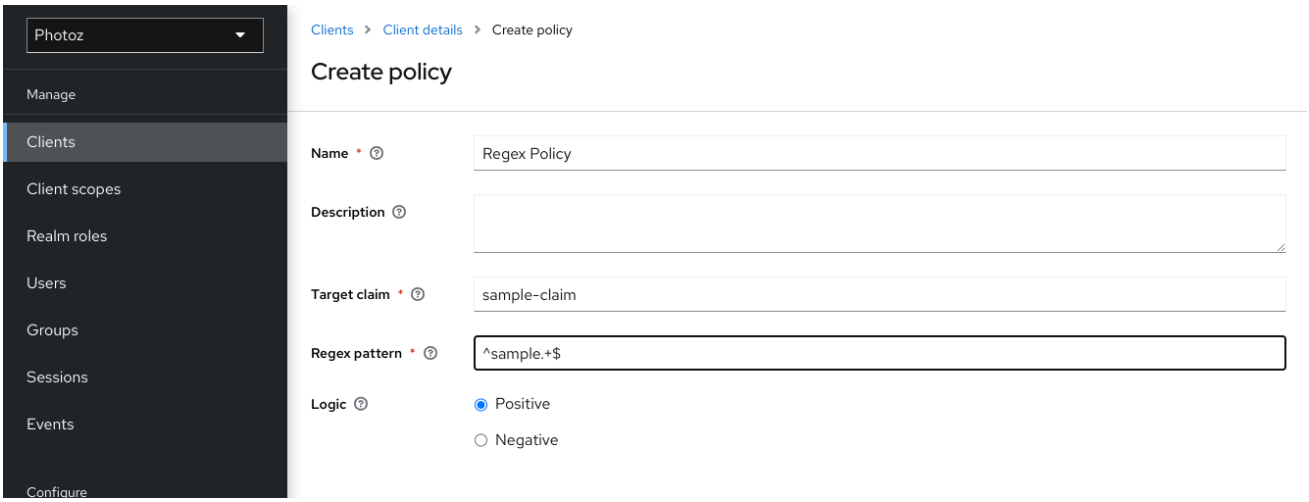
5.9. 正規表現ベースのポリシー

このタイプのポリシーを使用して、パーミッションの正規表現条件を定義することができます。

新しい正規表現ベースのポリシーを作成するには、ポリシータイプリストから **Regex** を選択します。

このポリシーは、現在のアイデンティティーから利用可能な属性を解決します。

正規表現ポリシーの追加



5.9.1. 設定

- **Name**
  ポリシーを記述する人間が判読できる一意な文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このポリシーの詳細を含む文字列。

- **Target Claim**
  トークン内のターゲットクレームの名前を指定します。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。ターゲットクレームが JSON オブジェクトを参照する場合、最初のパス (**contact** など) は、JSON オブジェクトを保持する属性名にマップする必要があります。

- **Regex Pattern**
  正規表現パターンを指定します。

- **Logic**
  他の条件を評価した後に適用するこのポリシーの ロジック。

5.10. 正ロジックと負ロジック

ポリシーは、正または負のロジックで設定できます。簡単に使用して、このオプションを使用してポリシーの結果を保持するか、否定するかを定義することができます。

たとえば、特定のロールで **付与されていない** ユーザーにのみアクセスを付与するポリシーを作成するとします。この場合は、そのロールを使用してロールベースのポリシーを作成し、その **Logic** フィールドを **Negative** に設定できます。デフォルトの動作である **Positive** を保持すると、ポリシーの結果はそのまま保持されます。

5.11. ポリシー評価 API

JavaScript を使用してルールベースのポリシーを作成する場合、Red Hat build of Keycloak は、パーミッションを付与するかどうかを判断するために役立つ情報を提供する評価用 API を提供します。

この API は、以下のような情報へのアクセスを提供するいくつかのインターフェイスで設定されています。

- 評価されるパーミッション (要求されているリソースとスコープの両方)。
- 要求されるリソースに関連付けられた属性
- ランタイム環境と、実行コンテキストに関連付けられたその他の属性
- グループメンバーシップおよびロールなどのユーザーに関する情報

メインインターフェイスは **org.keycloak.authorization.policy.evaluation.Evaluation** で、以下のコントラクトを定義します。

```
public interface Evaluation {

  /**
   * Returns the {@link ResourcePermission} to be evaluated.
   *
   * @return the permission to be evaluated
   */
  ResourcePermission getPermission();

  /**
   * Returns the {@link EvaluationContext}. Which provides access to the whole evaluation runtime context.
   *
```

```
     * @return the evaluation context
     */
    EvaluationContext getContext();

    /**
     * Returns a {@link Realm} that can be used by policies to query information.
     *
     * @return a {@link Realm} instance
     */
    Realm getRealm();

    /**
     * Grants the requested permission to the caller.
     */
    void grant();

    /**
     * Denies the requested permission.
     */
    void deny();
}
```

認証リクエストを処理する際に、Red Hat build of Keycloak はポリシーを評価する前に **Evaluation** インスタンスを作成します。その後、このインスタンスは各ポリシーに渡され、アクセスが **GRANT** または **DENY** かどうかが判断されます。

ポリシーは、**Evaluation** インスタンスで **grant()** または **deny()** メソッドを呼び出してこれを決定します。デフォルトでは、**評価** インスタンスの状態は拒否されます。つまり、ポリシー評価エンジンに対してパーミッションを付与する必要があることを示す **grant()** メソッドを明示的に呼び出す必要があります。

関連情報

- JavaDoc ドキュメント

5.11.1. 評価コンテキスト

評価コンテキストは、評価時にポリシーに有用な情報を提供します。

```
public interface EvaluationContext {

    /**
     * Returns the {@link Identity} that represents an entity (person or non-person) to which the
permissions must be granted, or not.
     *
     * @return the identity to which the permissions must be granted, or not
     */
    Identity getIdentity();

    /**
     * Returns all attributes within the current execution and runtime environment.
     *
     * @return the attributes within the current execution and runtime environment
```

```
     */
    Attributes getAttributes();
}
```

このインターフェイスから、ポリシーは以下を取得できます。

- 認証された **ID**
- 実行コンテキストおよびランタイム環境に関する情報

Identity は、認可要求と共に送信された OAuth2 アクセストークンに基づいてビルドされ、このコンストラクトは元のトークンから抽出されたすべての要求にアクセスできます。たとえば、**プロトコルマッパー** を使用して OAuth2 アクセストークンにカスタム要求を組み込む場合は、ポリシーからこの要求にアクセスし、これを使用して条件を構築することもできます。

EvaluationContext は、実行環境とランタイム環境の両方に関連する属性へのアクセスも提供します。ここでは、いくつかの組み込み属性のみがあります。

表5.1 実行およびランタイム属性

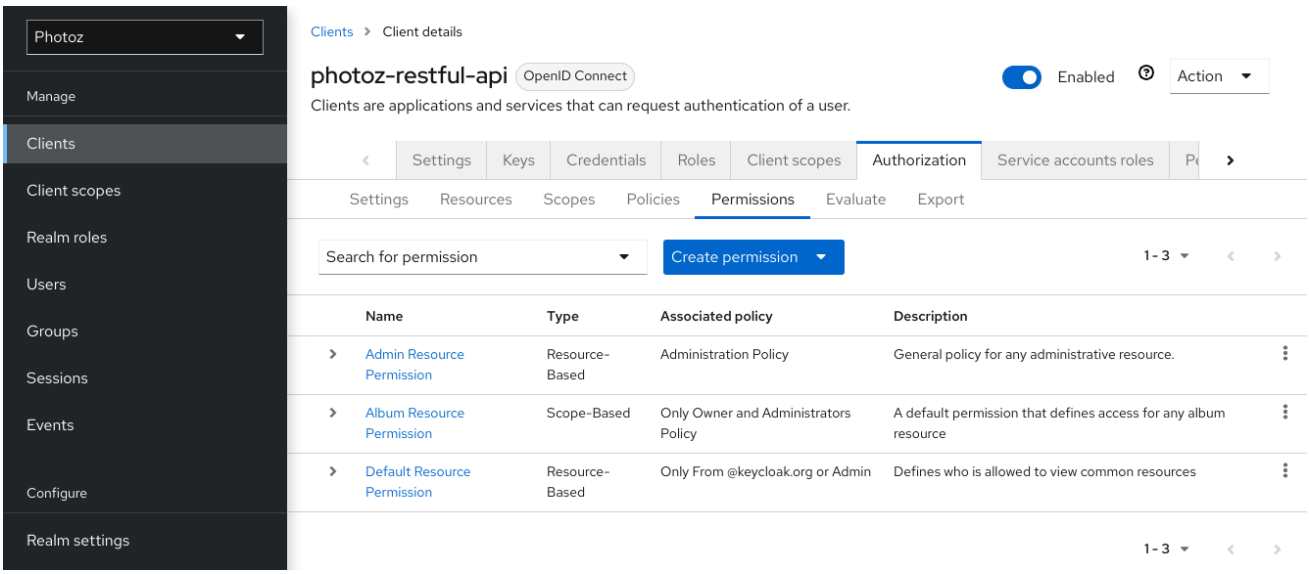
名前	説明	種類
kc.time.date_time	現在の日時	文字列。**MM/dd/yyyy hh:mm:ss** 形式。
kc.client.network.ip_address	クライアントの IPv4 アドレス	文字列
kc.client.network.host	クライアントのホスト名	文字列
kc.client.id	クライアント ID	文字列
kc.client.user_agent	User-AgentHTTP ヘッダーの値	String[]
kc.realm.name	レルムの名前	String

第6章 パーミッションの管理

パーミッションは、保護されているオブジェクトと、アクセスが付与されるかを決定するために評価する必要のあるポリシーを関連付けます。

保護するリソースや、これらのリソースを保護するために使用するポリシーを作成した後、パーミッションの管理を開始できます。パーミッションを管理するには、リソースサーバーを編集するときに **Permissions** タブをクリックします。

権限



パーミッションを作成して、2 つの主要なタイプのオブジェクトを保護することができます。

- **リソース**
- **スコープ**

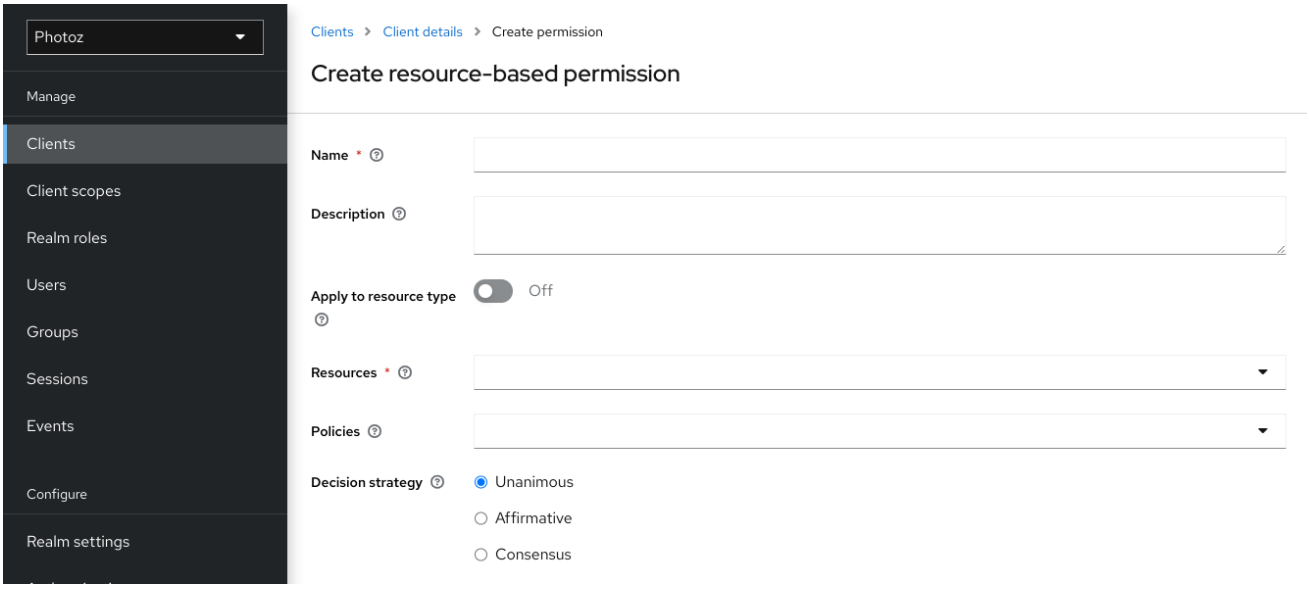
パーミッションを作成するには、パーミッションリストの右上にある項目リストから作成するパーミッションタイプを選択します。以下のセクションでは、これらの 2 つのタイプのオブジェクトについて詳しく説明します。

6.1. リソースベースのパーミッションの作成

リソースベースのパーミッションは、1 つ以上の認可ポリシーを使用して保護する 1 つ以上のリソースのセットを定義します。

新しいリソースベースのパーミッションを作成するには、**Create permission** ドロップダウンから **Create resource-based permission** を選択します。

リソースパーミッションの追加



6.1.1. 設定

- **名前**
  パーミッションを説明する、人間が判読可能な一意の文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このパーミッションの詳細を含む文字列。

- **Apply To Resource Type**
  パーミッションが指定されたタイプのすべてのリソースに適用されるかどうかを指定します。このフィールドを選択すると、保護するリソースタイプを入力するように求められます。

  - リソースタイプ
    保護するリソースタイプを定義します。定義すると、このパーミッションは、そのタイプに一致するすべてのリソースに対して評価されます。

- **Resources**
  保護する 1 つ以上のリソースのセットを定義します。

- **ポリシー**
  パーミッションに関連付ける 1 つ以上のポリシーのセットを定義します。ポリシーを関連付けるには、既存のポリシーを選択するか、作成するポリシーのタイプを選択して新規のポリシーを作成します。

- **Decision Strategy**
  このパーミッションの 決定ストラテジー。

6.1.2. 型が指定されたリソースのパーミッション

リソースパーミッションを使用して、特定の タイプ を持つすべてのリソースに適用されるポリシーを定義することもできます。このフォームは、共通のアクセス要件と制約を共有するリソースがある場合に便利です。

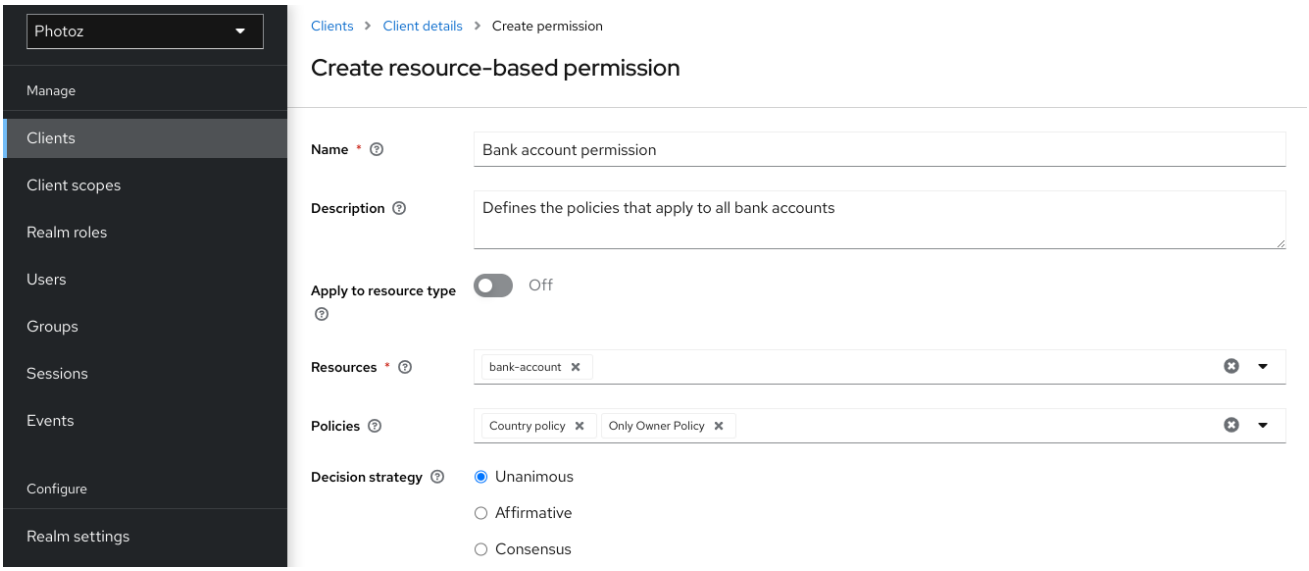
多くの場合、アプリケーション内のリソースは、カプセル化するデータまたは提供する機能に基づいて分類 (またはタイプ化) できます。たとえば、金融関係のアプリケーションは、特定の顧客に属する異なる銀行取引アカウントを管理できます。銀行のアカウントは異なりますが、銀行取引組織がグローバル

に定義されている共通のセキュリティー要件と制約を共有します。タイプされたリソースパーミッションでは、以下のように、すべての銀行のアカウントに適用する共通ポリシーを定義できます。

- アカウントを管理できるのは所有者のみです
- 所有者の国や地域からのアクセスのみを許可
- 特定の認証方法の実施

タイプが指定されたリソースパーミッションを作成するには、新しいリソースベースのパーミッションの作成時に Apply to Resource Type をクリックします。**Apply to Resource Type** を **On** に設定した状態で、保護するタイプと、指定したタイプですべてのリソースへのアクセスを制御するポリシーを指定できます。

タイプが指定されたリソースパーミッションの例

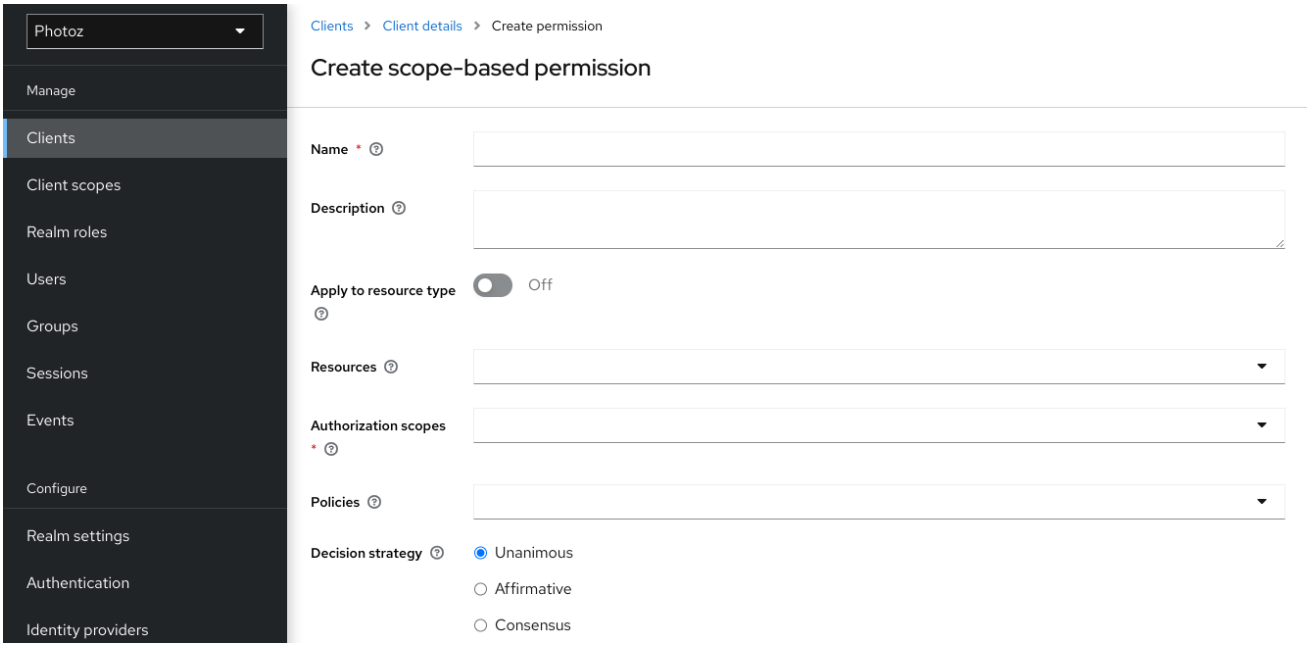


6.2. スコープベースのパーミッションの作成

スコープベースのパーミッションは、1つ以上の認可ポリシーを使用して保護する1つ以上のスコープのセットを定義します。リソースベースのパーミッションとは異なり、このパーミッションタイプを使用して、リソースだけでなく、リソースを管理するパーミッションやそれらに対して実行できるパーミッションを定義する際に、さらに粒度を細かく設定することができます。

新しいスコープベースのパーミッションを作成するには、**Create permission** ドロップダウンから **Create scope-based permission** を選択します。

スコープパーミッションの追加



6.2.1. 設定

- **名前**
  パーミッションを説明する、人間が判読可能な一意の文字列。ベストプラクティスは、ビジネス要件とセキュリティー要件に密接に関連する名前を使用して、それらをより簡単に識別できるようにすることです。

- **Description**
  このパーミッションの詳細を含む文字列。

- **リソース**
  選択されたリソースに関連付けられたスコープを制限します。何も選択しないと、すべてのスコープが利用可能になります。

- **Scopes**
  保護する 1 つ以上のスコープのセットを定義します。

- **ポリシー**
  パーミッションに関連付ける 1 つ以上のポリシーのセットを定義します。ポリシーを関連付けるには、既存のポリシーを選択するか、作成するポリシーのタイプを選択して新規のポリシーを作成します。

- **Decision Strategy**
  このパーミッションの 決定ストラテジー。

6.3. ポリシーデシジョンストラテジー

ポリシーをパーミッションに関連付ける際には、意思決定ストラテジーを定義して、関連付けられたポリシーの結果を評価してアクセスする方法を指定することもできます。

- **Unanimous**
  指定がない場合はデフォルトのストラテジーです。この場合、最終的な決定を正にするには、**すべて** のポリシーが正の決定に評価される必要があります。

- **Affirmative**

この場合、最終的な決定を正にするには、**少なくとも 1つ** のポリシーを正の決定に評価する必要があります。

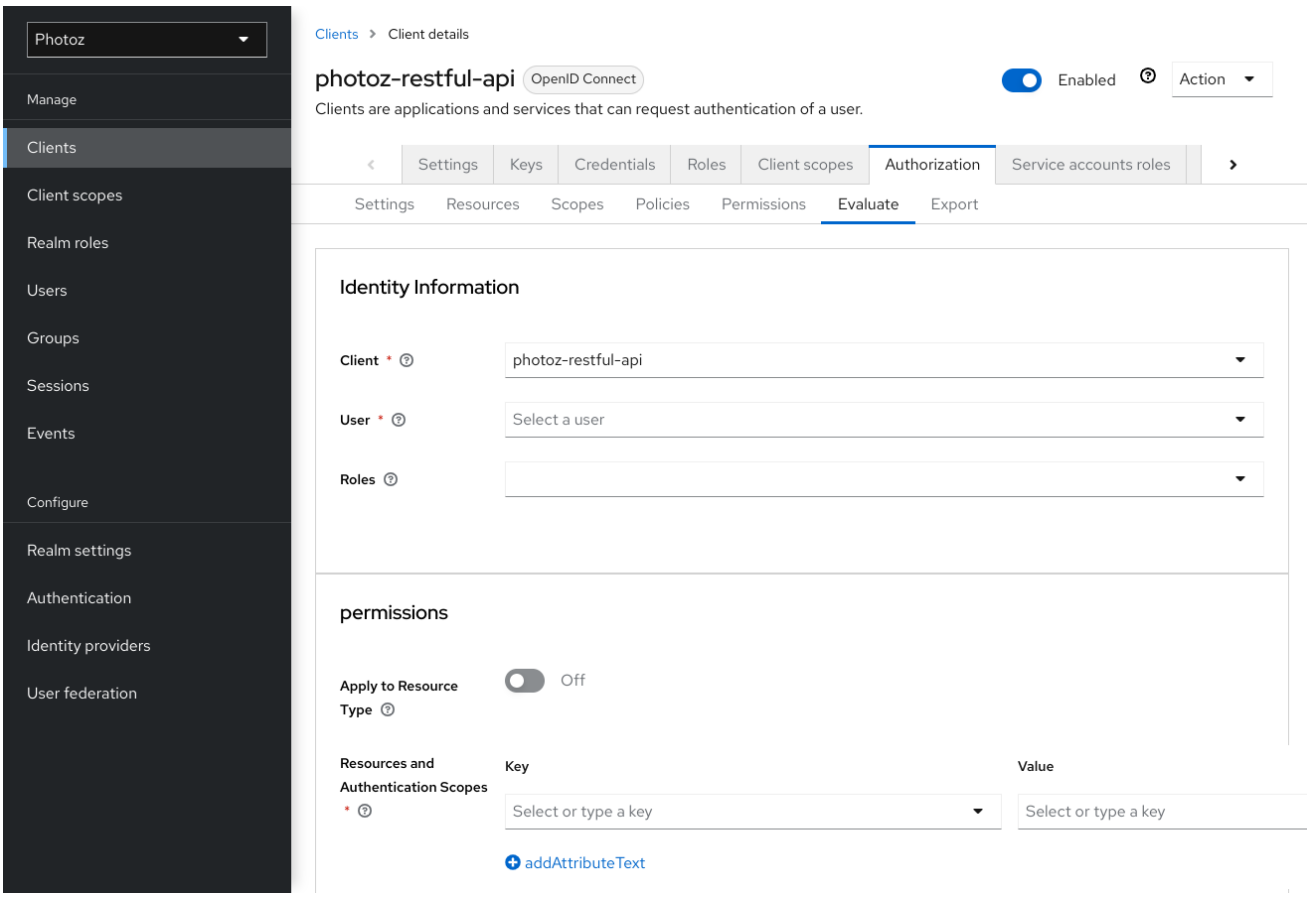
- Consensus
  この場合、正のデシジョン数は、負の数より大きくする必要があります。正と負のデシジョンの数が等しい場合は、最終的な決定が負の値になります。

第7章 ポリシーの評価およびテスト

ポリシーを設計する際に、認可要求をシミュレートして、ポリシーがどのように評価されるかテストすることができます。

リソースサーバーの編集時に **Evaluate** タブをクリックしてポリシー評価ツールにアクセスできます。実際の認可要求をシミュレートし、ポリシーの影響をテストするために異なる入力を指定できます。

ポリシー評価ツール



7.1. ID 情報の提供

Identity Information フィルターを使用して、パーミッションを要求するユーザーを指定できます。

7.2. コンテキスト情報の提供

Contextual Information フィルターを使用して評価コンテキストに追加の属性を定義して、ポリシーが同じ属性を取得できるようにすることができます。

7.3. パーミッションの指定

パーミッション フィルターは、認可要求を作成するために使用できます。1つ以上のリソースおよびスコープのセットのパーミッションを要求できます。すべての保護リソースおよびスコープに基づいて認可要求をシミュレーションする場合は、**Resources** または **Scopes** を指定せずに **Add** をクリックします。

必要な値を指定したら、**Evaluate** をクリックします。

第8章 認可サービス

Red Hat build of Keycloak の Authorization Services は、OAuth2 や User-Managed Access 仕様などのよく知られた標準規格の上にビルドされています。

OAuth2 クライアント (フロントエンドアプリケーションなど) は、トークンエンドポイントを使用してサーバーからアクセストークンを取得し、その同じトークンを使用してリソースサーバー (バックエンドサービスなど) で保護されたリソースにアクセスすることができます。同様に、Red Hat build of Keycloak の Authorization Services は OAuth2 の拡張機能を提供し、要求されたリソースまたはスコープに関連するすべてのポリシーの処理に基づいてアクセストークンを発行できます。つまり、リソースサーバーは、サーバーによって付与されるパーミッションに基づいて保護されているリソースへのアクセスを強制でき、アクセストークンによって保持されます。Red Hat build of Keycloak の Authorization Services では、パーミッションのあるアクセストークンは Requesting Party Token (RPT) と呼ばれます。

RPT の発行に加えて、Red Hat build of Keycloak の Authorization Services は、リソースサーバーが保護されたリソース、スコープ、パーミッション、ポリシーを管理できるようにする一連の RESTful エンドポイントも提供します。これは、きめ細かな認可をサポートするために、開発者がこれらの機能をアプリケーションに拡張または統合する際に役立ちます。

8.1. 認可サービスエンドポイントおよびメタデータの検出

Red Hat build of Keycloak では、ディスカバリードキュメントが提供されています。クライアントは、エンドポイントの場所や機能などを含め、Red Hat build of Keycloak Authorization Services と対話するために必要なすべての情報をここから取得できます。

検出ドキュメントは、以下から取得できます。

```
curl -X GET \
  http://${host}:${port}/realms/${realm}/.well-known/uma2-configuration
```

${host}:${port} は、Red Hat build of Keycloak が実行されているホスト名 (または IP アドレス) とポートで、**${realm}** は Red Hat build of Keycloak のレルムの名前です。

そのため、以下のような応答が返されるはずです。

```
{

    // some claims are expected here

    // these are the main claims in the discovery document about Authorization Services endpoints
location
    "token_endpoint": "http://${host}:${port}/realms/${realm}/protocol/openid-connect/token",
    "token_introspection_endpoint": "http://${host}:${port}/realms/${realm}/protocol/openid-
connect/token/introspect",
    "resource_registration_endpoint":
"http://${host}:${port}/realms/${realm}/authz/protection/resource_set",
    "permission_endpoint": "http://${host}:${port}/realms/${realm}/authz/protection/permission",
    "policy_endpoint": "http://${host}:${port}/realms/${realm}/authz/protection/uma-policy"
}
```

これらの各エンドポイントは、特定の機能セットを公開します。

- **token_endpoint**

urn:ietf:params:oauth:grant-type:uma-ticket 付与タイプをサポートする OAuth2 準拠のトークンエンドポイント。このエンドポイントを介して、クライアントは認可要求を送信し、Red Hat build of Keycloak によって付与されたすべての権限を持つ RPT を取得できます。

- **token_introspection_endpoint**
  OAuth2 準拠のトークンイントロスペクションエンドポイント。クライアントがサーバーにクエリーして RPT のアクティブな状態を判断し、Red Hat build of Keycloak が付与するパーミッションなどのトークンに関連付けられた他の情報を決定します。

- **resource_registration_endpoint**
  リソースサーバーが保護されるリソースおよびスコープを管理するために使用できる UMA 準拠の Resource Registration Endpoint。このエンドポイントは、Red Hat build of Keycloak のリソースおよびスコープの作成、読み取り、更新、削除の操作を行います。

- **permission_endpoint**
  リソースサーバーがパーミッションチケットを管理するのに使用できる UMA 準拠のパーミッションエンドポイント。このエンドポイントは、Red Hat build of Keycloak でパーミッションチケットを作成、読み取り、更新、削除する操作を提供します。

8.2. パーミッションの取得

Red Hat build of Keycloak からパーミッションを取得するには、認可要求をトークンエンドポイントに送信します。これにより、Red Hat build of Keycloak はリソースに関連付けられたすべてのポリシーを評価し、スコープが要求され、サーバーで付与されるすべてのパーミッションで RPT を発行します。

以下のパラメーターを使用して、クライアントはトークンエンドポイントに認可要求を送信できます。

- **grant_type**
  このパラメーターは **必須** です。**urn:ietf:params:oauth:grant-type:uma-ticket** である必要があります。

- **ticket**
  このパラメーターは **任意** です。UMA 認可プロセスの一部として、クライアントが受信する最新のパーミッションチケットです。

- **claim_token**
  このパラメーターは **任意** です。要求されるリソースおよびスコープのパーミッションを評価する際にサーバーによって考慮される必要のある追加の要求を表す文字列。このパラメーターにより、クライアントは要求を Red Hat build of Keycloak にプッシュできます。サポートされるすべてトークン形式に関する詳細は、**claim_token_format** パラメーターを参照してください。

- **claim_token_format**
  このパラメーターは **任意** です。**claim_token** パラメーターで指定されたトークンの形式を示す文字列。Red Hat build of Keycloak は、**urn:ietf:params:oauth:token-type:jwt** および **https://openid.net/specs/openid-connect-core-1_0.html#IDToken** の 2 つのトークン形式をサポートします。**urn:ietf:params:oauth:token-type:jwt** 形式は、**claim_token** パラメーターがアクセストークンを参照することを示します。**https://openid.net/specs/openid-connect-core-1_0.html#IDToken** は、**claim_token** パラメーターが OpenID Connect ID トークンを参照することを示しています。

- **rpt**
  このパラメーターは **任意** です。以前は、パーミッションを評価する RPT を発行して、新しいパーミッションに追加する必要がある。このパラメーターにより、クライアントが RPT の所有しているクライアントに対して、パーミッションがオンデマンドで追加される増分認可を実行できます。

- permission
  このパラメーターは **任意** です。クライアントがアクセスを求めている 1 つまたは複数のリソースとスコープのセットを表す文字列です。このパラメーターは、複数のリソースおよびスコープのパーミッションを要求するために複数回定義できます。このパラメーターは、クライアントがパーミッションチケットなしで認可要求を送信できるようにする **urn:ietf:params:oauth:grant-type:uma-ticket** 付与タイプの拡張です。文字列の形式は **RESOURCE_ID#SCOPE_ID** にする必要があります。たとえば、**Resource A#Scope A**、**Resource A#Scope A, Scope B, Scope C**、**Resource A**、**#Scope A** です。

- permission_resource_format
  このパラメーターは **任意** です。**permission** パラメーター内のリソースを示す形式を表現する文字列。可能な値は **id** と **uri** です。**id** は、形式が **RESOURCE_ID** であることを示します。**uri** は、形式が **URI** であることを示します。指定しない場合、デフォルトは **id** です。

- permission_resource_matching_uri
  このパラメーターは **任意** です。**permission** パラメーターでリソースを URI 形式で表すときに、パスマッチングを使用するかどうかを示すブール値。指定しない場合、デフォルトは false です。

- audience
  このパラメーターは **任意** です。クライアントがアクセスを表示しているリソースサーバーのクライアント識別子。**permission** パラメーターが定義されている場合、このパラメーターは必須です。これは、Red Hat build of Keycloak へのヒントとして機能し、パーミッションの評価に使用するコンテキストを示します。

- response_include_resource_name
  このパラメーターは **任意** です。リソース名を RPT のパーミッションに含めるかどうかを示すブール値。false の場合、リソース識別子のみが含まれます。

- response_permissions_limit
  このパラメーターは **任意** です。RPT が持つパーミッションの量の制限を定義する整数 N。**rpt** パラメーターとともに使用する場合は、最後に要求されたパーミッションのみが RPT に保持されます。

- submit_request
  このパラメーターは **任意** です。サーバーがリソースに対するパーミッション要求を作成するかどうか、およびパーミッションチケットによって参照されるスコープを許可するかどうかを示すブール値。このパラメーターは、UMA 認可プロセスの一部として **ticket** パラメーターとともに使用する場合にのみ有効です。

- response_mode
  このパラメーターは **任意** です。サーバーが認可要求に応答する方法を示す文字列値。このパラメーターは、標準の OAuth2 応答ではなく、サーバー全体またはサーバーによって付与されたパーミッションに主に関心がある場合に特に便利です。可能な値は次のとおりです。

  - **decision**
    サーバーからの応答は、以下の形式で JSON を返すことで全体的な決定内容のみを表す必要があることを示します。

    ```
    {
        'result': true
    }
    ```

    認可要求がパーミッションにマップされない場合、代わりに **403** HTTP ステータスコードが返されます。

- **permissions**
  以下の形式で JSON を返して、サーバーからの応答に、サーバーによって付与されたパーミッションが含まれることを示します。

```
[
   {
      'rsid': 'My Resource'
      'scopes': ['view', 'update']
   },

   ...
]
```

  認可要求がパーミッションにマップされない場合、代わりに **403** HTTP ステータスコードが返されます。

リソースサーバーが保護する 2 つのリソースへのアクセスをクライアントがシークしている場合の認可要求の例。

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "audience={resource_server_client_id}" \
  --data "permission=Resource A#Scope A" \
  --data "permission=Resource B#Scope B"
```

クライアントがすべてのリソースへのアクセスを生み、リソースサーバーによって保護されるスコープを指定する際の認可要求の例。注記: すべてのリソースのパーミッションが評価されるわけではありません。代わりに、リソースサーバーが所有するリソース、要求側ユーザーが所有するリソース、および他の所有者によって要求側ユーザーに明示的に付与されたリソースのパーミッションが評価されます。

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "audience={resource_server_client_id}"
```

認可プロセスの一環として、リソースサーバーからパーミッションチケットを受信した後に、クライアントが UMA 保護リソースにアクセスできるようにする場合に認可要求の例:

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "ticket=${permission_ticket}
```

Red Hat build of Keycloak アセスメントプロセスでパーミッションが発行されると、パーミッションが関連付けられている RPT が発行されます。

Red Hat build of Keycloak は RPT でクライアントに応答します

```
HTTP/1.1 200 OK
Content-Type: application/json
```

```
...
{
    "access_token": "${rpt}",
}
```

サーバーからの応答は、他の付与タイプを使用する場合にトークンエンドポイントからの他の応答と同様になります。RPT は **access_token** 応答パラメーターから取得できます。クライアントが認可されていない場合、Red Hat build of Keycloak は **403** HTTP ステータスコードで応答します。

Red Hat build of Keycloak は認可要求を拒否します

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
...
{
    "error": "access_denied",
    "error_description": "request_denied"
}
```

8.2.1. クライアント認証方法

RPT を取得するには、クライアントがトークンエンドポイントに対して認証する必要があります。**urn:ietf:params:oauth:grant-type:uma-ticket** タイプを使用すると、クライアントは以下の認証方法のいずれかを使用できます。

- **ベアラートークン**
  クライアントは、アクセストークンを HTTP Authorization ヘッダーの Bearer 認証情報としてトークンエンドポイントに送信する必要があります。

  例: アクセストークンを使用してトークンエンドポイントに対して認証を行う認可要求

  ```
  curl -X POST \
    http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
    -H "Authorization: Bearer ${access_token}" \
    --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket"
  ```

  この方法は、クライアントがユーザーの代わりに動作している場合などに便利です。この場合、ベアラートークンは、Red Hat build of Keycloak が以前、ユーザーの代わりに (または自身の代わりに) クライアントに発行したアクセストークンです。パーミッションは、アクセストークンで表されるアクセスコンテキストを考慮します。たとえば、ユーザー A に代わってアクセストークンがクライアント A に発行された場合、リソースおよびユーザー A がアクセス可能なスコープに応じてパーミッションが付与されます。

- **クライアント認証情報**
  クライアントは、Red Hat build of Keycloak でサポートされている任意のクライアント認証方法を使用できます。たとえば、client_id/client_secret または JWT などがあります。

  例: クライアント ID およびクライアントシークレットを使用してトークンエンドポイントに対して認証を行う認可要求

  ```
  curl -X POST \
    http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
    -H "Authorization: Basic cGhvdGg6L7Jl13RmfWgtkk==pOnNlY3JldA==" \
    --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket"
  ```

8.2.2. クレームのプッシュ

サーバーからパーミッションを取得する場合、任意の要求をプッシュし、パーミッションの評価時にこれらの要求をポリシーに対して利用可能にすることができます。

パーミッションチケット (UMA フロー) を **使用せず** にサーバーからパーミッションを取得する場合は、以下のように認可要求をトークンエンドポイントに送信できます。

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "claim_token=ewogICAib3JnYW5pemF0aW9uIjogWyJhY21lIl0KfQ==" \
  --data "claim_token_format=urn:ietf:params:oauth:token-type:jwt" \
  --data "client_id={resource_server_client_id}" \
  --data "client_secret={resource_server_client_secret}" \
  --data "audience={resource_server_client_id}"
```

claim_token パラメーターには、以下の例のような形式を持つ、BASE64 でエンコードされた JSON が必要になります。

```
{
   "organization" : ["acme"]
}
```

この形式では、各要求の値が文字列の配列である必要がある 1 つ以上の要求を想定します。

8.2.2.1. UMA を使用した要求のプッシュ

UMA およびパーミッションチケットの使用時に要求をプッシュする方法は、パーミッション API を参照してください。

8.3. ユーザー管理のアクセス

Red Hat build of Keycloak Authorization Services は、ユーザー管理アクセス (略して UMA) に基づいています。UMA は、以下の方法で OAuth2 機能を強化する仕様です。

- **プライバシー**
  今日、ユーザーのプライバシーは、より多くのデータやデバイスが利用でき、クラウドに接続しているため、ユーザーのプライバシーの懸念となります。UMA および Red Hat build of Keycloak を使用すると、リソースサーバーの機能が強化されます。ユーザーが定義したポリシーに基づきパーミッションが許可され、ユーザープライバシーの観点からリソースの保護方法が向上します。

- **サードパーティー間の認可**
  リソースの所有者 (通常のエンドユーザー) は、リソースへのアクセスを管理し、他のリソースにアクセスするために他のユーザーが (例: 通常のエンドユーザー) を認可できます。これは、ユーザーの代わりに動作しているクライアントアプリケーションに、UMA リソースの所有者が完全に非同期でアクセスすることが許可される OAuth2 とは異なります。

- **リソース共有**
  リソースの所有者はリソースに対するパーミッションを管理し、特定のリソースにアクセスできるユーザーや方法を決定できます。その後、Red Hat build of Keycloak は、リソース所有者がリソースを管理できる共有管理サービスとして機能します。

Red Hat build of Keycloak は、ほとんどの UMA 機能を提供する、UMA 2.0 準拠の認可サーバーです。

たとえば、Alice というユーザー (リソース所有者) は、インターネットバンキングサービス (リソースサーバー) を使用して自分の銀行口座 (リソース) を管理しているとします。ある日、Alice は自分の銀行口座を会計専門家の Bob (要求側) に開示することにしました。ただし、Bob は Alice の口座の閲覧権限 (スコープ) しか持っていません。

リソースサーバーとして、インターネットバンキングサービスは Alice の銀行口座を保護できる必要があります。このため、Red Hat build of Keycloak の Resource Registration Endpoint を使用して、Alice の銀行口座を表すサーバーでリソースを作成します。

現在、Bob が Alice の銀行口座にアクセスしようとすると、アクセスは拒否されます。インターネットバランシングサービスは、銀行取引アカウントのデフォルトポリシーをいくつか定義します。その1つは、所有者 (この場合は Alice) のみが銀行取引アカウントにアクセスできます。

ただし、インターネットボーシングサービスは、Alice のプライバシーに伴い、銀行取引アカウントの特定のポリシーを変更することもできます。変更可能なこれらのポリシーの1つとして、銀行取引アカウントを閲覧できるユーザーを定義することです。このため、インターネットバンキングサービスは Red Hat build of Keycloak を使用して、Alice が誰にどのような操作 (またはデータ) の権限を許可するかを選択できる領域を提供します。いつでも、Alice は追加のパーミッションを Bob に付与するか、付与できます。

8.3.1. 認可プロセス

UMA では、クライアントが UMA 保護リソースサーバーにアクセスしようとすると、認可プロセスが起動します。

UMA 保護リソースサーバーでは、トークンが RPT である要求内のベアラートークンが必要です。クライアントが RPT を使用せずにリソースサーバーでリソースを要求する場合:

クライアントで、RPT を送信せずに保護されているリソースを要求

```
curl -X GET \
  http://${host}:${port}/my-resource-server/resource/1bfdfe78-a4e1-4c2d-b142-fc92b75b986f
```

リソースサーバーは、パーミッション **ticket** と、RPT 取得のためのチケット送信先である Red Hat build of Keycloak サーバーの場所が指定された **as_uri** パラメーターを含む応答を、クライアントに送信します。

リソースサーバーはパーミッションチケットで応答

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="${realm}",
    as_uri="https://${host}:${port}/realms/${realm}",
    ticket="016f84e8-f9b9-11e0-bd6f-0021cc6004de"
```

パーミッションチケットは、Red Hat build of Keycloak Permission API が発行する特殊なトークンです。これらは、要求されるパーミッション (リソースやスコープなど) や、要求に関連するその他の情報を表します。リソースサーバーのみがこれらのトークンを作成できます。

これで、クライアントはパーミッションチケットと Red Hat build of Keycloak サーバーの場所情報を取得したため、ディスカバリードキュメントを使用してトークンエンドポイントの場所を取得し、認可要求を送信できます。

クライアントは、認可要求をトークンエンドポイントに送信して RPT を取得

```
curl -X POST \
```

```
http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
-H "Authorization: Bearer ${access_token}" \
--data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
--data "ticket=${permission_ticket}
```

Red Hat build of Keycloak アセスメントプロセスでパーミッションが発行されると、パーミッションが関連付けられている RPT が発行されます。

Red Hat build of Keycloak は RPT でクライアントに応答します

```
HTTP/1.1 200 OK
Content-Type: application/json
...
{
    "access_token": "${rpt}",
}
```

サーバーからの応答は、他の付与タイプを使用する場合にトークンエンドポイントからの他の応答と同様になります。RPT は **access_token** 応答パラメーターから取得できます。クライアントがパーミッションを持つことを認可されていない場合、Red Hat build of Keycloak は **403** HTTP ステータスコードで応答します。

Red Hat build of Keycloak は認可要求を拒否します

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
...
{
    "error": "access_denied",
    "error_description": "request_denied"
}
```

8.3.2. パーミッション要求の送信

認可プロセスの一環として、クライアントはまず Red Hat build of Keycloak の Token Endpoint で RPT と交換するために、UMA で保護されたリソースサーバーからパーミッションチケットを取得する必要があります。

デフォルトでは、Red Hat build of Keycloak は **403** HTTP ステータスコードで応答し、クライアントに RPT を発行できない場合は **request_denied** エラーを返します。

Red Hat build of Keycloak は認可要求を拒否します

```
HTTP/1.1 403 Forbidden
Content-Type: application/json
...
{
    "error": "access_denied",
    "error_description": "request_denied"
}
```

このような応答は、Red Hat build of Keycloak がパーミッションチケットで表されるパーミッションで RPT を発行できなかったことを意味します。

場合によっては、クライアントアプリケーションは非同期認可フローを開始し、アクセスを付与するべきかどうかを判断するリソースの所有者をリクエストする場合があります。そのため、クライアントは、トークンエンドポイントへの認可要求と共に **submit_request** 要求パラメーターを使用できます。

```
curl -X POST \
  http://${host}:${port}/realms/${realm}/protocol/openid-connect/token \
  -H "Authorization: Bearer ${access_token}" \
  --data "grant_type=urn:ietf:params:oauth:grant-type:uma-ticket" \
  --data "ticket=${permission_ticket} \
  --data "submit_request=true"
```

submit_request パラメーターを使用すると、Red Hat build of Keycloak はアクセスが拒否されたリソースごとにパーミッション要求を永続化します。作成されると、リソースの所有者は、アカウントを確認し、パーミッション要求を管理できます。

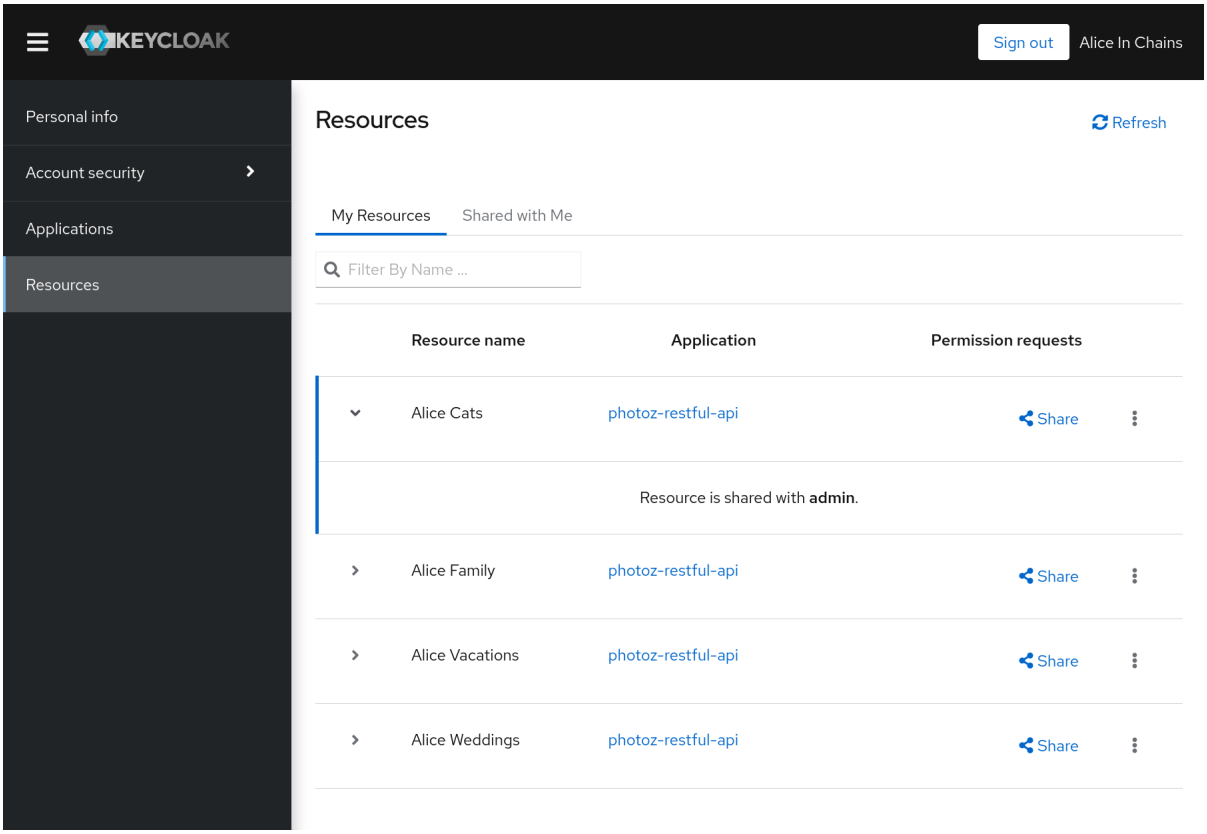
この機能はアプリケーションの **Request Access** ボタンと考えることができます。ここでは、他のユーザーにリソースへのアクセスを要求できます。

8.3.3. ユーザーリソースへのアクセス管理

ユーザーは、Red Hat build of Keycloak の アカウントコンソールを使用して、リソースへのアクセスを管理できます。この機能を有効にするには、最初にレルムのユーザー管理アクセスを有効にする必要があります。

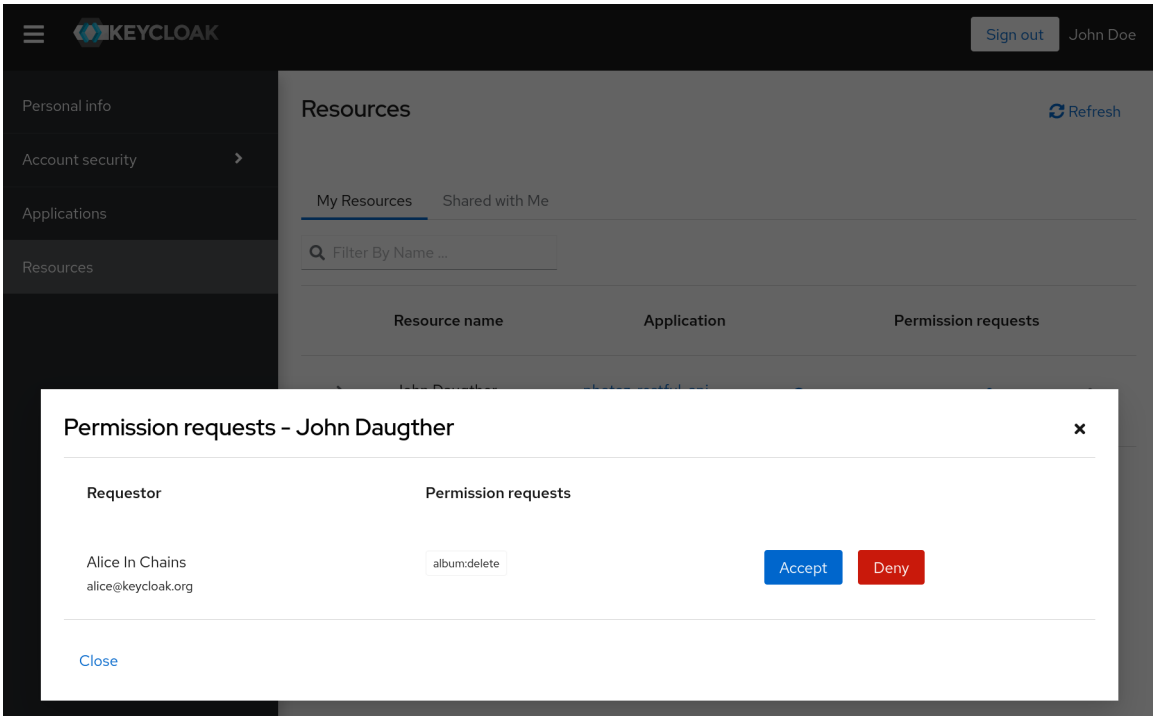
手順

1. 管理コンソールにログインします。
2. メニューで **Realm Settings** をクリックします。
3. **ユーザー管理アクセス** を **ON** に切り替えます。
4. 管理コンソールの右上にあるユーザー名をクリックし、**Manage Account** を選択します。



5. メニューオプションの **My Resources** をクリックします。以下のオプションを含むページが表示されます。

- **マイリソース** の管理
  このセクションには、ユーザーが所有するすべてのリソースのリストが含まれます。ユーザーは、リソースをクリックして詳細情報を検索し、リソースを他のユーザーと共有できます。承認待ちのパーミッション要求がある場合、リソースの名前の横にアイコンが表示されます。これらの要求は、特定のリソースへのアクセスを要求するユーザー (ユーザー) に接続されます。ユーザーはこれらのリクエストを承認または拒否できます。その場合はアイコンをクリックします。



- **共有されたリソース** の管理
  このセクションでは、ユーザーと共有する全リソースのリストを示します。

- **このリソースへのアクセス権限のあるユーザー** の管理
  本セクションでは、このリソースへのアクセスを持つユーザーのリストを説明します。ユーザーは、**Revoke** ボタンをクリックするか、特定の **Permission** を削除してアクセスを取り消すことができます。

- リソースを他のユーザーと共有する
  別のユーザーのユーザー名またはメールアドレスを入力し、ユーザーはリソースを共有し、アクセスを付与するパーミッションを選択できます。

8.4. 保護 API

この保護 API は、以下を提供する UMA 準拠のエンドポイントセットを提供します。

- **リソースの管理**
  このエンドポイントでは、リソースサーバーはリソースをリモートで管理し、ポリシーエンフォーサー が保護する必要のあるリソースについてサーバーをクエリーできるようにします。

- **パーミッションの管理**
  UMA プロトコルでは、リソースサーバーがこのエンドポイントにアクセスしてパーミッションチケットを作成します。Red Hat build of Keycloak は、パーミッションおよびクエリーパーミッションの状態を管理するエンドポイントも提供します。

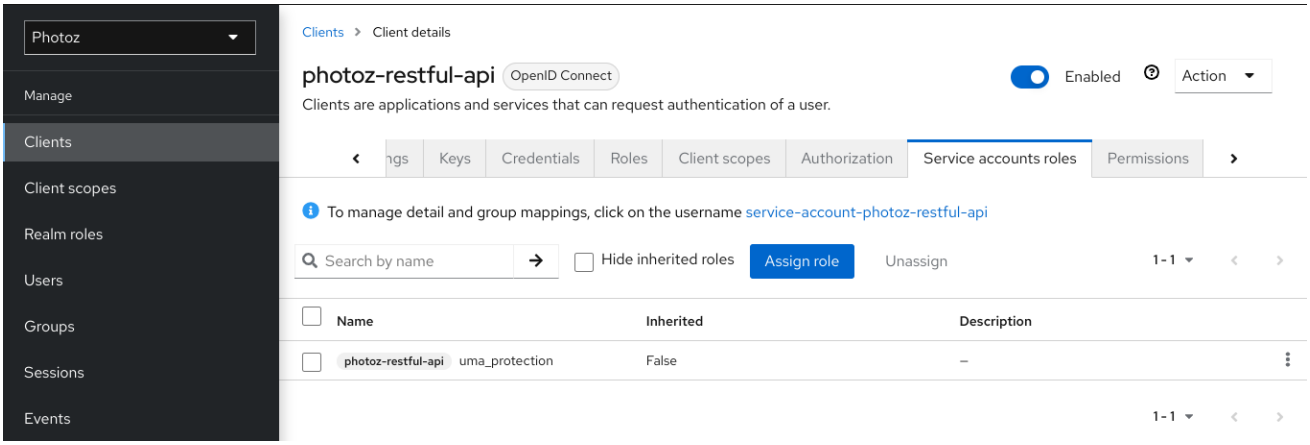
- **Policy API**
  Red Hat build of Keycloak は UMA Protection API を利用して、リソースサーバーがユーザーのパーミッションを管理できるようにします。Red Hat build of Keycloak は、Resource API や Permission API の他に、ユーザーの代わりにリソースサーバーがパーミッションを設定できる Policy API を提供します。

この API の重要な要件は、保護 API トークン (PAT) と呼ばれる特別な OAuth2 アクセストークンを使用して、リソースサーバー **のみ** がエンドポイントにアクセスできることです。UMA では、PAT はスコープの **uma_protection** を持つトークンです。

8.4.1. PAT および取得法

保護 API トークン (PAT) は、スコープが **uma_protection** として定義されている特別な OAuth2 アクセストークンです。リソースサーバーを作成すると、Red Hat build of Keycloak は対応するクライアントアプリケーションのロール **uma_protection** を自動的に作成し、それをクライアントのサービスアカウントに関連付けます。

uma_protection ロールで付与されたサービスアカウント



リソースサーバーは、他の OAuth2 アクセストークンと同様に、Red Hat build of Keycloak から PAT を取得できます。curl の使用例:

```
curl -X POST \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d 'grant_type=client_credentials&client_id=${client_id}&client_secret=${client_secret}' \
    "http://${host}:${port}/realms/${realm_name}/protocol/openid-connect/token"
```

上記の例は、**client_credentials** 付与タイプを使用してサーバーから PAT を取得します。その結果、サーバーは以下のような応答を返します。

```
{
  "access_token": ${PAT},
  "expires_in": 300,
  "refresh_expires_in": 1800,
  "refresh_token": ${refresh_token},
  "token_type": "bearer",
  "id_token": ${id_token},
  "not-before-policy": 0,
  "session_state": "ccea4a55-9aec-4024-b11c-44f6f168439e"
}
```

注記

Red Hat build of Keycloak は、さまざまな方法でクライアントアプリケーションを認証できます。分かりやすくするために、**client_credentials** 付与タイプを使用します。これには **client_id** と **client_secret** が必要です。サポート対象の認証方法を選択できます。

8.4.2. リソースの管理

リソースサーバーは、UMA 準拠のエンドポイントを使用してリソースをリモートで管理できます。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set
```

このエンドポイントは、以下のように要約された操作を提供します (分かりやすくするために終了となるパスを省略します)。

- リソースセットの説明の作成: POST /resource_set
- リソースセットの説明の読み取り: GET /resource_set/{_id}

- リソースセットの説明の更新: PUT /resource_set/{_id}
- リソースセットの説明の削除: DELETE /resource_set/{_id}
- リソースセットの説明のリスト表示: GET /resource_set

各操作のコントラクトに関する詳細は、UMA リソース登録 API を参照してください。

8.4.2.1. リソースの作成

リソースを作成するには、以下のように HTTP POST リクエストを送信する必要があります。

```
curl -v -X POST \
  http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '{
    "name":"Tweedl Social Service",
    "type":"http://www.example.com/rsrcs/socialstream/140-compatible",
    "icon_uri":"http://www.example.com/icons/sharesocial.png",
    "resource_scopes":[
        "read-public",
        "post-updates",
        "read-private",
        "http://www.example.com/scopes/all"
     ]
  }'
```

デフォルトでは、リソースの所有者はリソースサーバーです。特定のユーザーなど、異なる所有者を定義する場合は、以下のようにリクエストを送信できます。

```
curl -v -X POST \
  http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '{
    "name":"Alice Resource",
    "owner": "alice"
  }'
```

プロパティーの **owner** はユーザーのユーザー名または識別子で設定できます。

8.4.2.2. ユーザー管理リソースの作成

デフォルトでは、保護 API で作成されたリソースは、アカウントコンソール を通じてリソースの所有者によって管理できません。

リソースを作成し、リソースの所有者がこれらのリソースを管理できるようにするには、**ownerManagedAccess** プロパティーを以下のように設定する必要があります。

```
curl -v -X POST \
  http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '{
```

```
    "name":"Alice Resource",
    "owner": "alice",
    "ownerManagedAccess": true
  }'
```

8.4.2.3. リソースの更新

既存のリソースを更新するには、以下のように HTTP PUT 要求を送信します。

```
curl -v -X PUT \
  http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set/{resource_id} \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '{
    "_id": "Alice Resource",
    "name":"Alice Resource",
    "resource_scopes": [
      "read"
    ]
  }'
```

8.4.2.4. リソースの定義

既存のリソースを削除するには、以下のように HTTP DELETE リクエストを送信します。

```
curl -v -X DELETE \
  http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set/{resource_id} \
  -H 'Authorization: Bearer '$pat
```

8.4.2.5. リソースのクエリー

id 別にリソースをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set/{resource_id}
```

name を指定してリソースをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?name=Alice Resource
```

デフォルトでは、**name** フィルターは指定されたパターンのすべてのリソースと一致します。完全に一致するリソースのみを返すようにクエリーを制限するには、次を使用します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?name=Alice
Resource&exactName=true
```

uri を指定してリソースをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?uri=/api/alice
```

owner を指定してリソースをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?owner=alice
```

タイプ のあるリソースをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?type=albums
```

scope を指定してリソースをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/resource_set?scope=read
```

サーバーに対してパーミッションをクエリーする場合は、**first** パラメーターおよび **max** パラメーターを使用すると結果が制限されます。

8.4.3. パーミッション要求の管理

UMA プロトコルを使用するリソースサーバーは、特定のエンドポイントを使用してパーミッション要求を管理できます。このエンドポイントは、パーミッション要求を登録し、パーミッションチケットを取得するための UMA 準拠のフローを提供します。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/permission
```

パーミッションチケット は、パーミッション要求を表す特別なセキュリティートークンタイプです。UMA 仕様では、パーミッションチケットは以下のようになります。

認可サーバーからリソースサーバーへ、リソースサーバーからクライアントへ、最終的にはクライアントから認可サーバーに返信される相関ハンドル。これにより、認可サーバーは正しいポリシーを評価し、認可データのリクエストに適用されます。

ほとんどの場合、このエンドポイントを直接処理する必要はありません。Red Hat build of Keycloak は、リソースサーバーに対して UMA を有効にする ポリシーエンフォーサー を提供します。これにより、認可サーバーからパーミッションチケットを取得し、そのチケットをクライアントアプリケーションに返し、最終的な要求者トークン (RPT) に基づいき認可について決定できます。

Red Hat build of Keycloak からパーミッションチケットを取得するプロセスは、通常のクライアントアプリケーションではなくリソースサーバーによって実行されます。その場合、クライアントが適切なアクセス権を持たないまま保護されたリソースにアクセスしようとすると、パーミッションチケットが取得されます。パーミッションチケットの発行は、リソースサーバーを許可する UMA を使用する場合の重要な要素です。

- リソースサーバーが保護するリソースに関連付けられたデータをクライアントから抽象化します。
- Red Hat build of Keycloak 認可要求に登録します。これは後のワークフローで、リソース所有者の同意に基づきアクセス権を付与するために使用できます。
- 認可サーバーからリソースサーバーを分離し、異なる認可サーバーを使用してリソースを保護します。

クライアントについては、パーミッションチケットには重要な要素があり、その重要な側面は重要なものであり、以下の重要なものが重要となります。

- クライアントは、保護リソースと認可データがどのように関連付けられているのかを把握する必要はありません。パーミッションチケットはクライアントに完全に不透明です。
- クライアントは、異なるリソースサーバーのリソースにアクセスでき、異なる認可サーバーによって保護されます。

これは、UMA の他の側面がパーミッションチケットに基づいて設計されているという利点があります。これは、リソースへのプライバシーやユーザー管理に関しては、パーミッションチケットに基づいて強く使用されます。

8.4.3.1. パーミッションチケットの作成

パーミッションチケットを作成するには、以下のように HTTP POST リクエストを送信します。

```
curl -X POST \
  http://${host}:${port}/realms/${realm_name}/authz/protection/permission \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '[
  {
    "resource_id": "{resource_id}",
    "resource_scopes": [
      "view"
    ]
  }
]'
```

チケットの作成時に、任意の要求をプッシュし、これらの要求をチケットに関連付けることもできます。

```
curl -X POST \
  http://${host}:${port}/realms/${realm_name}/authz/protection/permission \
  -H 'Authorization: Bearer '$pat \
  -H 'Content-Type: application/json' \
  -d '[
  {
    "resource_id": "{resource_id}",
    "resource_scopes": [
      "view"
    ],
    "claims": {
        "organization": ["acme"]
    }
  }
]'
```

これらの要求は、パーミッションチケットに関連付けられたリソースおよびスコープのパーミッションの評価時にポリシーで利用できます。

8.4.3.2. UMA 以外のエンドポイント

8.4.3.2.1. パーミッションチケットの作成

リソースの所有者が以下のように HTTP POST 要求を送信するため、ID {user_id} の特定のリソースに ID {user_id} の特定のリソースのパーミッションを付与するには、以下を実行します。

```
curl -X POST \
    http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket \
    -H 'Authorization: Bearer '$access_token \
    -H 'Content-Type: application/json' \
```

```
    -d '{
      "resource": "{resource_id}",
      "requester": "{user_id}",
      "granted": true,
      "scopeName": "view"
    }'
```

8.4.3.2.2. パーミッションチケットの取得

```
curl http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket \
    -H 'Authorization: Bearer '$access_token
```

これらのクエリーパラメーターのいずれかを使用することができます。

- **scopeId**
- **resourceId**
- **owner**
- **requester**
- **granted**
- **returnNames**
- **first**
- **max**

8.4.3.2.3. パーミッションチケットの更新

```
curl -X PUT \
    http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket \
    -H 'Authorization: Bearer '$access_token \
    -H 'Content-Type: application/json' \
    -d '{
      "id": "{ticket_id}"
      "resource": "{resource_id}",
      "requester": "{user_id}",
      "granted": false,
      "scopeName": "view"
    }'
```

8.4.3.2.4. パーミッションチケットの削除

```
curl -X DELETE
http://${host}:${port}/realms/${realm_name}/authz/protection/permission/ticket/{ticket_id} \
    -H 'Authorization: Bearer '$access_token
```

8.4.4. Policy API を使用したリソースパーミッションの管理

Red Hat build of Keycloak は UMA Protection API を利用して、リソースサーバーがユーザーのパーミッションを管理できるようにします。Red Hat build of Keycloak は、Resource API や Permission API の他に、ユーザーの代わりにリソースサーバーがパーミッションを設定できる Policy API を提供します。

Policy API は以下で使用できます。

```
http://${host}:${port}/realms/${realm_name}/authz/protection/uma-policy/{resource_id}
```

この API はベアラートークンで保護されています。ベアラートークンは、ユーザーに代わって権限を管理するためにユーザーがリソースサーバーに付与した同意を表す必要があります。ベアラートークンは、以下を使用してトークンエンドポイントから取得した通常のアクセストークンになります。

- リソースオーナーパスワードの認証情報の付与タイプ
- オーディエンスがリソースサーバーである場合に、一部のクライアント (パブリッククライアント) に付与されるアクセストークンを交換するトークン Exchange

8.4.4.1. リソースへのパーミッションの関連付け

パーミッションを特定リソースに関連付けるには、以下のように HTTP POST 要求を送信する必要があります。

```
curl -X POST \
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
  -H 'Authorization: Bearer '$access_token \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '{
 "name": "Any people manager",
 "description": "Allow access to any people manager",
 "scopes": ["read"],
 "roles": ["people-manager"]
}'
```

上記の例では、**resource_id** で表されるリソースに新しいパーミッションを作成し、関連付けています。ここでは、**people-manager** ロールを持つすべてのユーザーに **読み取り** スコープが付与されます。

グループの使用など、他のアクセス制御メカニズムを使用してポリシーを作成することもできます。

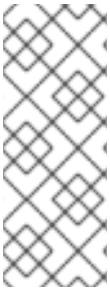
```
curl -X POST \
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
  -H 'Authorization: Bearer '$access_token \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '{
 "name": "Any people manager",
 "description": "Allow access to any people manager",
 "scopes": ["read"],
 "groups": ["/Managers/People Managers"]
}'
```

または特定のクライアント:

```
curl -X POST \
```

```
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
  -H 'Authorization: Bearer '$access_token \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '{
 "name": "Any people manager",
 "description": "Allow access to any people manager",
 "scopes": ["read"],
 "clients": ["my-client"]
}'
```

または、JavaScript を使用してカスタムポリシーを使用する場合でも、以下を実行します。



注記

アップロードスクリプトは **非推奨** となり、今後のリリースで削除されます。この機能はデフォルトでは無効になっています。

-Dkeycloak.profile.feature.upload_scripts=enabled でサーバーの起動を有効にするには、次のコマンドを実行します。詳細は、機能の有効化と無効化 ガイドを参照してください。

```
curl -X POST \
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{resource_id} \
  -H 'Authorization: Bearer '$access_token \
  -H 'Cache-Control: no-cache' \
  -H 'Content-Type: application/json' \
  -d '{
 "name": "Any people manager",
 "description": "Allow access to any people manager",
 "scopes": ["read"],
 "condition": "my-deployed-script.js"
}'
```

これらのアクセス制御メカニズムの組み合わせを設定することもできます。

既存のパーミッションを更新するには、以下のように HTTP PUT 要求を送信します。

```
curl -X PUT \
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{permission_id} \
  -H 'Authorization: Bearer '$access_token \
  -H 'Content-Type: application/json' \
  -d '{
    "id": "21eb3fed-02d7-4b5a-9102-29f3f09b6de2",
    "name": "Any people manager",
    "description": "Allow access to any people manager",
    "type": "uma",
    "scopes": [
        "album:view"
    ],
    "logic": "POSITIVE",
    "decisionStrategy": "UNANIMOUS",
    "owner": "7e22131a-aa57-4f5f-b1db-6e82babcd322",
    "roles": [
```

```
        "user"
    ]
}'
```

8.4.4.2. パーミッションの削除

リソースに関連付けられたパーミッションを削除するには、以下のように HTTP DELETE リクエストを送信します。

```
curl -X DELETE \
  http://localhost:8180/realms/photoz/authz/protection/uma-policy/{permission_id} \
  -H 'Authorization: Bearer '$access_token
```

8.4.4.3. パーミッションのクエリー

リソースに関連付けられたパーミッションをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy?resource={resource_id}
```

その名前のパーミッションをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy?name=Any people manager
```

特定のスコープに関連付けられたパーミッションをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy?scope=read
```

すべてのパーミッションをクエリーするには、以下のように HTTP GET リクエストを送信します。

```
http://${host}:${port}/realms/${realm}/authz/protection/uma-policy
```

サーバーに対してパーミッションをクエリーする場合は、**first** パラメーターおよび **max** パラメーターを使用すると結果が制限されます。

8.5. 参加者トークンの要求

要求者トークン (RPT) は、JSON Web 署名 (JWS) を使用してデジタル署名された JSON Web トークン (JWT) です。トークンは、Red Hat build of Keycloak が特定のクライアントに対して以前に発行した OAuth2 アクセストークンに基づき、ユーザーまたは自身の代わりに作成されます。

RPT をデコードすると、以下のようなペイロードが表示されます。

```
{
  "authorization": {
     "permissions": [
      {
        "resource_set_id": "d2fe9843-6462-4bfc-baba-b5787bb6e0e7",
        "resource_set_name": "Hello World Resource"
      }
     ]
```

```
  },
  "jti": "d6109a09-78fd-4998-bf89-95730dfd0892-1464906679405",
  "exp": 1464906971,
  "nbf": 0,
  "iat": 1464906671,
  "sub": "f1888f4d-5172-4359-be0c-af338505d86c",
  "typ": "kc_ett",
  "azp": "hello-world-authz-service"
}
```

このトークンから、サーバーから付与されたすべてのパーミッションを **permissions** 要求から取得できます。

また、パーミッションは、実際に付与して、これらの同じ権限を発行するために使用されたアクセス制御メソッドから保護および完全に分離しているリソース/スコープと直接関連する点にご留意ください。

8.5.1. 要求するトークンのイントロスペクション

場合によっては、リクエスト側のトークン (RPT) をイントロスペクションして、トークン内のパーミッションを確認するか、リソースサーバー側の認可決定を実施したい場合があります。

トークンイントロスペクションが役立つ主なユースケースは 2 つあります。

- クライアントアプリケーションがトークンの有効性をクエリーして、同じまたは追加のパーミッションを持つ新しいパーミッションを取得する必要がある場合は、
- リソースサーバーで認可の決定を強制する場合 (特に組み込みの ポリシーエンフォーサー がお使いのアプリケーションに適さない場合)

8.5.2. RPT に関する情報の取得

トークンイントロスペクションは基本的に OAuth2 トークンイントロスペクション に準拠するエンドポイントであり、そこから RPT に関する情報を取得できます。

```
http://${host}:${port}/realms/${realm_name}/protocol/openid-connect/token/introspect
```

このエンドポイントを使用して RPT をイントロスペクションするには、以下のようにサーバーにリクエストを送信します。

```
curl -X POST \
    -H "Authorization: Basic aGVsbG8td29ybGQtYXV0aHotc2VydmljZTpzZWNyZXQ=" \
    -H "Content-Type: application/x-www-form-urlencoded" \
    -d 'token_type_hint=requesting_party_token&token=${RPT}' \
    "http://localhost:8080/realms/hello-world-authz/protocol/openid-connect/token/introspect"
```

注記

上記の要求では HTTP BASIC を使用して、クライアントのクレデンシャル (クライアント ID およびシークレット) を渡して、トークンのイントロスペクションを試みるクライアントを認証していますが、Red Hat build of Keycloak でサポートされる他のクライアント認証方法を使用できます。

イントロスペクションエンドポイントには 2 つのパラメーターが必要です。

- **token_type_hint**
  このパラメーターの値として **requesting_party_token** を使用します。これは、RPT をイントロスペクションする必要があることを示します。

- **token**
  このパラメーターの値として、認可プロセス中にサーバーが返したためトークン文字列を使用します。

その結果、サーバーの応答は以下のようになります。

```
{
  "permissions": [
    {
      "resource_id": "90ccc6fc-b296-4cd1-881e-089e1ee15957",
      "resource_name": "Hello World Resource"
    }
  ],
  "exp": 1465314139,
  "nbf": 0,
  "iat": 1465313839,
  "aud": "hello-world-authz-service",
  "active": true
}
```

RPT がアクティブでない場合は、代わりに以下の応答が返されます。

```
{
  "active": false
}
```

8.5.3. RPT をイントロスペクションするたびにサーバーを呼び出す必要がありますか?

いいえ。Red Hat build of Keycloak サーバーが発行した通常のアクセストークンと同様に、RPT はデフォルトの形式として JSON Web トークン (JWT) 仕様も使用します。

リモートイントロスペクションエンドポイントへの呼び出しなしにこれらのトークンを検証する場合は、RPT をデコードし、その妥当性についてクエリーできます。トークンをデコードしたら、トークン内のパーミッションを使用して認可決定を実施することもできます。

これは基本的に、ポリシーエンフォーサーの動作です。必ず以下を行なってください。

- RPT の署名を検証します (レルムの公開鍵に基づく)
- **exp**、**iat**、および **aud** の要求に基づくトークンの有効性についてクエリー

関連情報

- JSON Web トークン (JWT)
- ポリシーエンフォーサー

8.6. AUTHORIZATION CLIENT JAVA API

要件によっては、リソースサーバーはリソースをリモートで管理したり、パーミッションをプログラム的にチェックしたりできるはずです。Java を使用している場合は、認可クライアント API を使用して Red Hat build of Keycloak Authorization Services にアクセスできます。

トークンエンドポイント、リソース、パーミッション管理エンドポイントなど、サーバーが提供する異なるエンドポイントにアクセスするリソースサーバーを対象としています。

8.6.1. Maven 依存関係

```
<dependencies>
    <dependency>
        <groupId>org.keycloak</groupId>
        <artifactId>keycloak-authz-client</artifactId>
        <version>${KEYCLOAK_VERSION}</version>
    </dependency>
</dependencies>
```

8.6.2. 設定

クライアント設定は、以下のように **keycloak.json** ファイルで定義されます。

```
{
  "realm": "hello-world-authz",
  "auth-server-url" : "http://localhost:8080",
  "resource" : "hello-world-authz-service",
  "credentials": {
    "secret": "secret"
  }
}
```

- **realm** (必須)
  レルムの名前。

- **auth-server-url** (必須)
  Red Hat build of Keycloak サーバーのベース URL。他のすべての Red Hat build of Keycloak ページと REST サービスエンドポイントは、ここから派生します。通常の形式は https://host:port です。

- **resource** (必須)
  アプリケーションの client-id各アプリケーションには、アプリケーションを識別するために使用される client-id があります。

- **credentials** (必須)
  アプリケーションの認証情報を指定します。これは、キーが認証情報タイプで、値は認証情報タイプの値です。

設定ファイルは通常、クライアントが **keycloak.json** ファイルを見つけようとするデフォルトの場所であるアプリケーションのクラスパスに置かれます。

8.6.3. 認可クライアントの作成

クラスパスに **keycloak.json** ファイルがあることを考慮すると、以下のように **AuthzClient** インスタンスを作成できます。

```
    // create a new instance based on the configuration defined in a keycloak.json located in your
classpath
    AuthzClient authzClient = AuthzClient.create();
```

8.6.4. ユーザーエンタイトルメントの取得

以下に、ユーザーエンタイトルメントの取得方法を示す例を示します。

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// create an authorization request
AuthorizationRequest request = new AuthorizationRequest();

// send the entitlement request to the server in order to
// obtain an RPT with all permissions granted to the user
AuthorizationResponse response = authzClient.authorization("alice", "alice").authorize(request);
String rpt = response.getToken();

System.out.println("You got an RPT: " + rpt);

// now you can use the RPT to access protected resources on the resource server
```

1 つ以上のリソースのセットでユーザーエンタイトルメントを取得する方法を示す例を以下に示します。

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// create an authorization request
AuthorizationRequest request = new AuthorizationRequest();

// add permissions to the request based on the resources and scopes you want to check access
request.addPermission("Default Resource");

// send the entitlement request to the server in order to
// obtain an RPT with permissions for a single resource
AuthorizationResponse response = authzClient.authorization("alice", "alice").authorize(request);
String rpt = response.getToken();

System.out.println("You got an RPT: " + rpt);

// now you can use the RPT to access protected resources on the resource server
```

8.6.5. 保護 API を使用したリソースの作成

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// create a new resource representation with the information we want
ResourceRepresentation newResource = new ResourceRepresentation();

newResource.setName("New Resource");
```

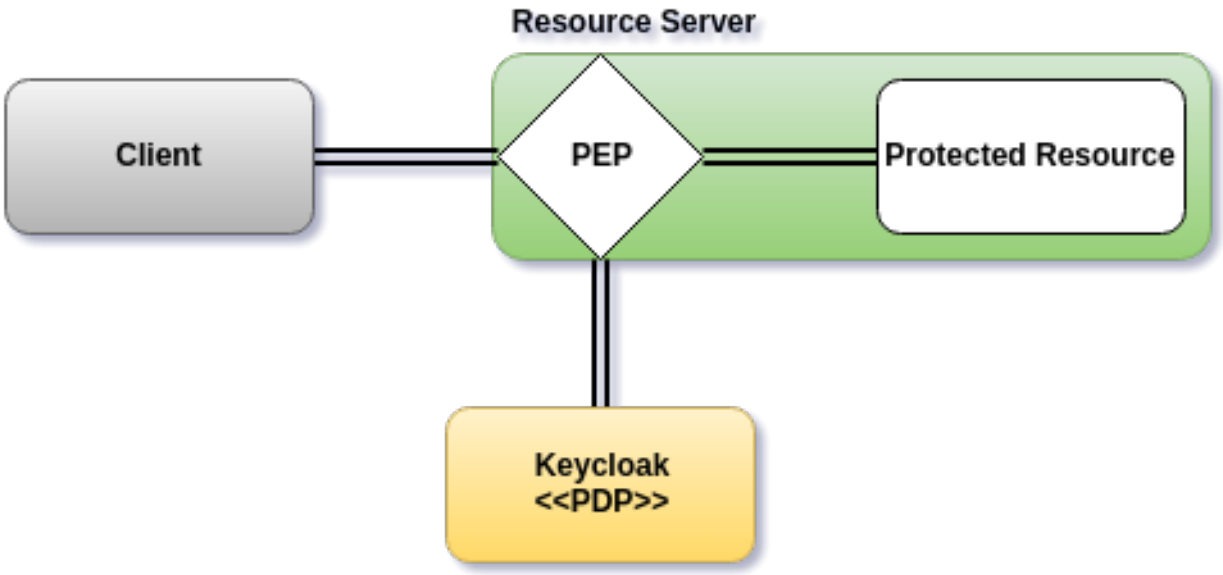
```
newResource.setType("urn:hello-world-authz:resources:example");

newResource.addScope(new ScopeRepresentation("urn:hello-world-authz:scopes:view"));

ProtectedResource resourceClient = authzClient.protection().resource();
ResourceRepresentation existingResource = resourceClient.findByName(newResource.getName());

if (existingResource != null) {
    resourceClient.delete(existingResource.getId());
}

// create the resource on the server
ResourceRepresentation response = resourceClient.create(newResource);
String resourceId = response.getId();

// query the resource using its newly generated id
ResourceRepresentation resource = resourceClient.findById(resourceId);

System.out.println(resource);
```

8.6.6. RPT のイントロスペクション

```
// create a new instance based on the configuration defined in keycloak.json
AuthzClient authzClient = AuthzClient.create();

// send the authorization request to the server in order to
// obtain an RPT with all permissions granted to the user
AuthorizationResponse response = authzClient.authorization("alice", "alice").authorize();
String rpt = response.getToken();

// introspect the token
TokenIntrospectionResponse requestingPartyToken =
authzClient.protection().introspectRequestingPartyToken(rpt);

System.out.println("Token status is: " + requestingPartyToken.getActive());
System.out.println("Permissions granted by the server: ");

for (Permission granted : requestingPartyToken.getPermissions()) {
    System.out.println(granted);
}
```

第9章 ポリシーエンフォーサー

Policy Enforcement Point (PEP: Policy Enforcement Point) は、異なる方法で実装できる設計パターンです。Red Hat build of Keycloak は、さまざまなプラットフォーム、環境、およびプログラミング言語に PEP を実装するのに必要な方法をすべて提供します。Red Hat build of Keycloak Authorization Services は RESTful API を提供し、一元化された認可サーバーを使用して詳細な認可に OAuth2 認可機能を活用します。



PEP は、保護されているリソースに関連付けられたポリシーを評価して、Red Hat build of Keycloak サーバーからアクセス決定を強制します。これは、保護されたリソースへの特定の要求を満たすかどうかをチェックするためにアプリケーションのフィルターやインターセプターとして機能します。この決定によって付与されるパーミッションに基づいて、保護されるリソースへの特定のリクエストを満たすことができるかどうかを確認します。

Red Hat build of Keycloak は、JakartaEE 準拠のフレームワークと Web コンテナーを保護するために、Java アプリケーションに対する **Red Hat build of Keycloak ポリシーエンフォーサー**を有効にするビルトインサポートを提供します。Maven を使用している場合は、プロジェクトに次の依存関係を設定する必要があります。

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-policy-enforcer</artifactId>
    <version>${keycloak.version}</version>
</dependency>
```

ポリシーエンフォーサーを有効にすると、アプリケーションに送信されたリクエストはすべてインターセプトされ、そのリクエストを送信しているアイデンティティーに対して Red Hat build of Keycloak が付与したパーミッションに応じて、保護されたリソースへのアクセスが付与されます。

ポリシーの適用は、アプリケーションのパスと、Red Hat build of Keycloak の管理コンソールを使用してリソースサーバー用に作成した リソース に強く連動しています。デフォルトではリソースサーバーを作成すると、ポリシーの適用をすぐに有効にできるように、Red Hat build of Keycloak はリソースサーバーの デフォルト設定 を作成します。

9.1. 設定

ポリシーエンフォーサの設定では JSON 形式が使用されます。リソースサーバーから利用可能なリソースに基づいて保護されたパスを自動的に解決する場合、ほとんどの場合は何も設定する必要はありません。

保護するリソースを手動で定義する場合は、もう少し詳細な形式を使用できます。

```
{
  "enforcement-mode" : "ENFORCING",
  "paths": [
   {
     "path" : "/users/*",
     "methods" : [
      {
        "method": "GET",
        "scopes" : ["urn:app.com:scopes:view"]
      },
      {
        "method": "POST",
        "scopes" : ["urn:app.com:scopes:create"]
      }
     ]
   }
  ]
}
```

以下は、各設定オプションの説明です。

- **enforcement-mode**
  ポリシーの適用方法を指定します。
  - **ENFORCING**
    (デフォルトモード) 指定のリソースにポリシーが関連付けられていない場合を含め、リクエストはデフォルトで拒否されます。
  - **PERMISSIVE**
    指定のリソースにポリシーが関連付けられていない場合を含め、リクエストは許可されます。
  - **DISABLED**
    ポリシーの評価を完全に無効にし、すべてのリソースにアクセスできるようにします。**enforcement-mode** が **DISABLED** の場合、アプリケーションは 認可コンテキスト を介して Red Hat build of Keycloak が付与したすべてのパーミッションを取得できます。
- **on-deny-redirect-to**
  "access denied" メッセージがサーバーから取得される際に、クライアントリクエストがリダイレクトされる URL を定義します。デフォルトでは、アダプターは 403 HTTP ステータスコードで応答します。
- **path-cache**
  ポリシーエンフォーサーが、Red Hat build of Keycloak で定義したアプリケーションとリソース間の関連付けを追跡する方法を定義します。このキャッシュは、パスと保護リソース間の関連付けをキャッシュすることで、Red Hat build of Keycloak サーバーへの不要なリクエストを回避するために必要です。
  - **有効期間**

エントリーの期限が切れる時間 (ミリ秒単位) を定義します。指定しないと、デフォルト値は 30000 になります。0 に等しい値を設定して、キャッシュを完全に無効にすることができます。-1 に等しい値を設定して、キャッシュの有効期限を無効にすることができます。

- max-entries
  キャッシュに保持される必要があるエントリーの制限を定義します。指定しないと、デフォルト値は 1000 になります。

- paths
  保護するパスを指定します。この設定はオプションです。定義されていない場合、ポリシーエンフォーサーは Red Hat build of Keycloak でアプリケーションに定義したリソースを取得してすべてのパスを検出します。これらのリソースは、アプリケーション内の一部のパスを表す **URIS** で定義されます。

  - name
    指定のパスに関連付けられるサーバー上のリソースの名前。**path** と併用される場合、ポリシーエンフォーサーはリソースの **URIS** プロパティーを無視し、代わりに指定したパスを使用します。

  - path
    (必須) アプリケーションのコンテキストパスに相対する URI。このオプションが指定されていると、ポリシーは、同じ値を持つ **URI** でリソースについて、サーバーに対してクエリーを実行します。現在、パス照合に非常に基本的なロジックがサポートされています。有効なパスの例は以下のとおりです。

    - ワイルドカード: /*
    - 接尾辞: /***.html**
    - サブパス: /**path**/*
    - パスパラメーター: /resource/{id}
    - 完全一致: /resource
    - パターン: /{version}/resource, /api/{version}/resource, /api/{version}/resource/*

  - methods
    HTTP メソッド (GET、POST、PATCH など) は、サーバーの特定リソースのスコープとどのように関連付けされるかを指定します。

    - method
      HTTP メソッドの名前。

    - scopes
      メソッドに関連付けられたスコープを持つ文字列の配列。スコープを特定のメソッドに関連付ける際に、保護されたリソース (またはパス) へのアクセスを試みるクライアントが、リストで指定されたすべてのスコープにパーミッションを付与する RPT を提供する必要があります。たとえば、スコープ **作成** でメソッド **POST** を定義する場合、パスへの POST を実行する際に、RPT には **create** スコープへのアクセスを付与するパーミッションが含まれている必要があります。

    - scopes-enforcement-mode
      メソッドに関連付けられたスコープの強制モードを参照する文字列。値は、**ALL** または **ANY** にすることができます。**ALL** の場合は、このメソッドを使用してリソースにアクセスするために定義されたすべてのスコープが付与される必要があります。**ANY** の

場合は、このメソッドを使用してリソースにアクセスするには、少なくとも 1 つのスコープを付与する必要があります。デフォルトでは、強制モードは **ALL** に設定されます。

- **enforcement-mode**
  ポリシーの適用方法を指定します。
  - **ENFORCING**
    (デフォルトモード) 指定されたリソースに関連付けられたポリシーがない場合でも、要求はデフォルトで拒否されます。
  - **DISABLED**
- **claim-information-point**
  これらの要求をポリシーで利用可能にするために解決し、Red Hat build of Keycloak サーバーにプッシュする必要のある 1 つ以上の要求のセットを定義します。詳細は、要求情報ポイント を参照してください。

- **lazy-load-paths**
  アプリケーションがパスに関連付けられたリソースのサーバーを取得する方法を指定します。**true** の場合、ポリシーエンフォーサーは要求されるパスに応じてオンデマンドでリソースをフェッチします。この設定は、デプロイメント中にサーバーからすべてのリソースを取得する場合 (**paths** を指定しなかった場合) や、**paths** のサブセットのみが定義されていて、他をオンデマンドで取得する場合などに特に便利です。

- **http-method-as-scope**
  スコープを HTTP メソッドにマッピングする方法を指定します。**true** に設定すると、ポリシーエンフォーサーは現在のリクエストから HTTP メソッドを使用して、アクセスを付与すべきかどうかを確認します。有効にした場合、Red Hat build of Keycloak のリソースが保護している各 HTTP メソッドを表すスコープに関連付けられていることを確認してください。

- **claim-information-point**
  これらの要求をポリシーで利用可能にするために、解決して Red Hat build of Keycloak サーバーにプッシュする必要がある 1 つ以上の **グローバル** 要求のセットを定義します。詳細は、要求情報ポイント を参照してください。

9.2. 要求情報ポイント

Claim Information Point (CIP) は、ポリシーへのアクセスコンテキストに関する詳細情報を提供するために、要求を解決し、これらの要求を Red Hat build of Keycloak サーバーにプッシュします。policy-enforcer の設定オプションとして定義して、以下のような異なるソースから要求を解決できます。

- HTTP リクエスト (パラメーター、ヘッダー、ボディーなど)
- 外部 HTTP サービス
- 設定で定義された静的値
- Claim Information Provider SPI を実装するその他のソース

Red Hat build of Keycloak サーバーに要求をプッシュする場合、ポリシーは、ユーザーが誰であるかだけでなく、指定されたトランザクションの who、what、why、when、where、which に基づき、コンテキストとコンテンツを考慮して決定できます。コンテキストベースの認可や、粒度の細かい認可の決定をサポートするためにランタイム情報を使用する方法のみです。

9.2.1. HTTP リクエストからの情報の取得

HTTP 要求から要求を抽出する方法を示す例を以下に示します。

keycloak.json

```
{
  "paths": [
    {
      "path": "/protected/resource",
      "claim-information-point": {
        "claims": {
          "claim-from-request-parameter": "{request.parameter['a']}",
          "claim-from-header": "{request.header['b']}",
          "claim-from-cookie": "{request.cookie['c']}",
          "claim-from-remoteAddr": "{request.remoteAddr}",
          "claim-from-method": "{request.method}",
          "claim-from-uri": "{request.uri}",
          "claim-from-relativePath": "{request.relativePath}",
          "claim-from-secure": "{request.secure}",
          "claim-from-json-body-object": "{request.body['/a/b/c']}",
          "claim-from-json-body-array": "{request.body['/d/1']}",
          "claim-from-body": "{request.body}",
          "claim-from-static-value": "static value",
          "claim-from-multiple-static-value": ["static", "value"],
          "param-replace-multiple-placeholder": "Test {keycloak.access_token['/custom_claim/0']} and
{request.parameter['a']}"
        }
      }
    }
  ]
}
```

9.2.2. 外部 HTTP サービスからの情報の取得

以下は、外部 HTTP サービスから要求を抽出する方法を示しています。

keycloak.json

```
{
  "paths": [
    {
      "path": "/protected/resource",
      "claim-information-point": {
        "http": {
          "claims": {
            "claim-a": "/a",
            "claim-d": "/d",
            "claim-d0": "/d/0",
            "claim-d-all": [
              "/d/0",
              "/d/1"
            ]
          },
          "url": "http://mycompany/claim-provider",
```

```
          "method": "POST",
          "headers": {
            "Content-Type": "application/x-www-form-urlencoded",
            "header-b": [
              "header-b-value1",
              "header-b-value2"
            ],
            "Authorization": "Bearer {keycloak.access_token}"
          },
          "parameters": {
            "param-a": [
              "param-a-value1",
              "param-a-value2"
            ],
            "param-subject": "{keycloak.access_token['/sub']}",
            "param-user-name": "{keycloak.access_token['/preferred_username']}",
            "param-other-claims": "{keycloak.access_token['/custom_claim']}"
          }
        }
      }
    }
  ]
}
```

9.2.3. 静的要求

keycloak.json

```
{
  "paths": [
    {
      "path": "/protected/resource",
      "claim-information-point": {
        "claims": {
          "claim-from-static-value": "static value",
          "claim-from-multiple-static-value": ["static", "value"]
        }
      }
    }
  ]
}
```

9.2.4. クレーム情報プロバイダー SPI

Claim Information Provider SPI は、組み込みプロバイダーが要件に対応するために十分ではない場合に開発者が異なる要求情報ポイントをサポートするために使用できます。

たとえば、新しい CIP プロバイダーを実装するには、**org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory** および **ClaimInformationPointProvider** and also provide the file **META-INF/services/org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory** をアプリケーションのクラスパスに指定する必要があります。

org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory の例:

```
public class MyClaimInformationPointProviderFactory implements
ClaimInformationPointProviderFactory<MyClaimInformationPointProvider> {

    @Override
    public String getName() {
        return "my-claims";
    }

    @Override
    public void init(PolicyEnforcer policyEnforcer) {

    }

    @Override
    public MyClaimInformationPointProvider create(Map<String, Object> config) {
        return new MyClaimInformationPointProvider(config);
    }
}
```

すべての CIP プロバイダーは、**MyClaimInformationPointProviderFactory.getName** メソッドで上記で定義されている名前に関連付ける必要があります。この名前は、**policy-enforcer** 設定の **claim-information-point** セクションから実装に設定をマップするために使用されます。

リクエストの処理時に、ポリシーエンフォーサーは MyClaimInformationPointProviderFactory.create メソッドを呼び出して MyClaimInformationPointProvider のインスタンスを取得します。呼び出されると、この特定の CIP プロバイダーに定義されたすべての設定がマップとして渡されます (request-information-point により)。

ClaimInformationPointProvider の例:

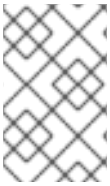
```
public class MyClaimInformationPointProvider implements ClaimInformationPointProvider {

    private final Map<String, Object> config;

    public MyClaimInformationPointProvider(Map<String, Object> config) {
        this.config = config;
    }

    @Override
    public Map<String, List<String>> resolve(HttpFacade httpFacade) {
        Map<String, List<String>> claims = new HashMap<>();

        // put whatever claim you want into the map

        return claims;
    }
}
```

9.3. 認可コンテキストの取得

ポリシーの適用が有効になると、サーバーから取得したパーミッションは **org.keycloak.AuthorizationContext** で利用できます。このクラスは複数のメソッドを提供し、特定のリソースまたはスコープにパーミッションが付与されたかどうかにかかわらず、パーミッションの取得に使用できます。

サーブレットコンテナーでの認可コンテキストの取得

```
    HttpServletRequest request = // obtain javax.servlet.http.HttpServletRequest
    AuthorizationContext authzContext = (AuthorizationContext)
request.getAttribute(AuthorizationContext.class.getName());
```



注記

認可コンテキストを使用すると、サーバーによる決定と返される決定をより細かく制御できます。たとえば、これを使用して、リソースまたはスコープに関連付けられたパーミッションに応じて、項目の表示や表示が動的メニューをビルドできます。

```
if (authzContext.hasResourcePermission("Project Resource")) {
    // user can access the Project Resource
}

if (authzContext.hasResourcePermission("Admin Resource")) {
    // user can access administration resources
}

if (authzContext.hasScopePermission("urn:project.com:project:create")) {
    // user can create new projects
}
```

AuthorizationContext は、Red Hat build of Keycloak Authorization Services の主要機能の 1 つを表します。上記の例では、保護されたリソースがそれらを管理するポリシーに直接関連付けられていないことを確認できます。

ロールベースアクセス制御 (RBAC) を使用する同様のコードについて考えてみましょう。

```
if (User.hasRole('user')) {
    // user can access the Project Resource
}

if (User.hasRole('admin')) {
    // user can access administration resources
}

if (User.hasRole('project-manager')) {
    // user can create new projects
}
```

いずれの例も同じ要件に対応しますが、これらはさまざまな方法で対処します。RBAC では、ロールはリソースのアクセスを **暗黙的** に定義します。Red Hat build of Keycloak では、RBAC、属性ベースのアクセス制御 (ABAC)、またはその他の BAC バリアントを使用している場合でも、リソースに直接焦点をあてる管理可能なコードを作成できます。指定されたリソースまたはスコープのパーミッションの有無は関係ありません。

セキュリティー要件が変更され、プロジェクトマネージャーに加えて、PMO も新しいプロジェクトを作成できるようになりました。

セキュリティー要件が変更されても、Red Hat build of Keycloak を使用している場合は、新しい要件を指すためにアプリケーションコードを変更する必要はありません。アプリケーションがリソースとスコープ識別子に基づいていると、認可サーバーの特定のリソースに関連付けられたパーミッションまた

はポリシーの設定のみを変更する必要があります。この場合、**プロジェクトリソース** やスコープ **urn:project.com:project:create** に関連付けられたパーミッションおよびポリシーが変更されます。

9.4. AUTHORIZATIONCONTEXT を使用した認可クライアントインスタンスの取得

AuthorizationContext を使用して、アプリケーションに設定された Authorization Client API への参照を取得することもできます。

```
ClientAuthorizationContext clientContext = ClientAuthorizationContext.class.cast(authzContext);
AuthzClient authzClient = clientContext.getClient();
```

ポリシーエンフォーバーで保護されているリソースサーバーは、認可サーバーによって提供される API にアクセスする必要があります。手動で **AuthzClient** インスタンスを使用すると、リソースサーバーはリソースの作成やプログラムによる特定のパーミッションのチェックを行うためにサーバーと対話できます。

9.5. JAVASCRIPT 統合

Red Hat build of Keycloak サーバーには JavaScript ライブラリーが含まれており、このライブラリーを使用して、ポリシーエンフォーサーで保護されたリソースサーバーとの対話に使用できます。このライブラリーは、Red Hat build of Keycloak JavaScript アダプターをベースとしています。これを統合して、クライアントが Red Hat build of Keycloak サーバーからパーミッションを取得できるようにすることができます。

Web ページに以下の **script** タグを含めることで、稼働中の Red Hat build of Keycloak インスタンスからこのライブラリーを取得できます。

```
<script src="http://.../js/keycloak-authz.js"></script>
```

次に、以下のように **KeycloakAuthorization** インスタンスを作成できます。

```
const keycloak = ... // obtain a Keycloak instance from keycloak.js library
const authorization = new KeycloakAuthorization(keycloak);
```

keycloak-authz.js ライブラリーは、2 つの主要機能を提供します。

- UMA 保護リソースサーバーにアクセスする場合、パーミッションチケットを使用してサーバーからパーミッションを取得します。
- リソースを送信し、アプリケーションがアクセスできるスコープを指定して、サーバーからパーミッションを取得します。

どちらの場合も、ライブラリーを使用することで、リソースサーバーと Red Hat build of Keycloak Authorization Services の両方と簡単に対話して、クライアントがベアラートークンとして使用してリソースサーバー上で保護されているリソースにアクセスするためのパーミッションを持つトークンを取得できます。

9.5.1. UMA-Protected Resource Server からの認可応答の処理

ポリシーエンフォーバーによってリソースサーバーが保護されている場合、ベアラートークンとともに実行されたパーミッションに基づいてクライアント要求に応答します。通常、保護されているリソースへのアクセスパーミッションがないベアラートークンを持つリソースサーバーにアクセスしようとする

と、リソースサーバーは 401 ステータスコードと **WWW-Authenticate** ヘッダーを返します。

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: UMA realm="${realm}",
   as_uri="https://${host}:${port}/realms/${realm}",
   ticket="016f84e8-f9b9-11e0-bd6f-0021cc6004de"
```

詳細は、UMA 認可プロセス を参照してください。

クライアントが必要とする内容は、リソースサーバーから返された **WWW-Authenticate** ヘッダーからパーミッションチケットを抽出し、以下のようにライブラリーを使用して認可要求を送信します。

```
// prepare a authorization request with the permission ticket
const authorizationRequest = {};
authorizationRequest.ticket = ticket;

// send the authorization request, if successful retry the request
Identity.authorization.authorize(authorizationRequest).then(function (rpt) {
   // onGrant
}, function () {
   // onDeny
}, function () {
   // onError
});
```

authorize 機能は完全に非同期で、サーバーから通知を受信するいくつかのコールバック機能をサポートします。

- **onGrant**: 関数の最初の引数。認可に成功し、サーバーは要求されたパーミッションを持つ RPT を返す場合、コールバックは RPT を受け取ります。
- **onDeny**: 関数の 2 つ目の引数。サーバーが認可要求を拒否した場合にのみ呼び出されます。
- **onError**: 関数の 3 つ目の引数。サーバーが予期せず応答する場合にのみ呼び出されます。

ほとんどのアプリケーションは、**onGrant** コールバックを使用して、401 応答の後にリクエストを再試行する必要があります。後続の要求には、再試行のベアラートークンとして RPT を含める必要があります。

9.5.2. エンタイトルメントの取得

keycloak-authz.js ライブラリーは、クライアントがアクセスするリソースとスコープを提供することでサーバーから RPT を取得するために使用できる **エンタイトルメント** 機能を提供します。

すべてのリソースのパーミッションと、ユーザーがアクセスできるスコープが設定された RPT を取得する方法の例

```
authorization.entitlement('my-resource-server-id').then(function (rpt) {
   // onGrant callback function.
   // If authorization was successful you'll receive an RPT
   // with the necessary permissions to access the resource server
});
```

特定のリソースおよびスコープのパーミッションのある RPT を取得する方法の例

-

```
authorization.entitlement('my-resource-server', {
    "permissions": [
        {
            "id" : "Some Resource"
        }
    ]
}).then(function (rpt) {
    // onGrant
});
```

エンタイトルメント 機能を使用する場合は、アクセスするリソースサーバーの **client_id** を指定する必要があります。

エンタイトルメント 機能は完全に非同期で、サーバーから通知を受信するためのコールバック関数をいくつかサポートします。

- **onGrant**: 関数の最初の引数。認可に成功し、サーバーは要求されたパーミッションを持つ RPT を返す場合、コールバックは RPT を受け取ります。
- **onDeny**: 関数の 2 つ目の引数。サーバーが認可要求を拒否した場合にのみ呼び出されます。
- **onError**: 関数の 3 つ目の引数。サーバーが予期せず応答する場合にのみ呼び出されます。

9.5.3. 認可要求

認可 および **エンタイトルメント** 機能はいずれも、認可リクエストオブジェクトを受け入れます。このオブジェクトは以下のプロパティーで設定できます。

- permissions
  リソースとスコープを表すオブジェクトの配列。以下に例を示します。

  ```
  const authorizationRequest = {
    "permissions": [
      {
        "id" : "Some Resource",
        "scopes" : ["view", "edit"]
      }
    ]
  }
  ```

- metadata
  プロパティーが、サーバーによる認可要求の処理方法を定義するオブジェクト。

  - response_include_resource_name
    リソース名を RPT のパーミッションに含める必要があるかどうかを示すブール値。false の場合、リソース識別子のみが含まれます。

  - response_permissions_limit
    RPT が持つパーミッションの量の制限を定義する整数 N。**rpt** パラメーターとともに使用する場合は、最後に要求された N 個のパーミッションのみが RPT に保持されます。

- submit_request
  サーバーがリソースに対するパーミッション要求を作成するかどうか、およびパーミッションチケットによって参照されるスコープを許可するかどうかを示すブール値。このパラメーターは、UMA 認可プロセスの一部として **ticket** パラメーターと共に使用する場合にのみ有効にな

ります。

9.5.4. RPT の取得

ライブラリーが提供するいずれかの認可機能を使用して RPT をすでに取得している場合は、認可オブジェクトからフォローするように常に RPT を取得できます (上記で説明した技術のいずれかによって初期化されていることを前提とします)。

```
const rpt = authorization.rpt;
```

9.6. TLS/HTTPS の設定

サーバーが HTTPS を使用している場合は、ポリシーエンフォーサが次のように設定されていることを確認してください。

```
{
  "truststore": "path_to_your_trust_store",
  "truststore-password": "trust_store_password"
}
```

上記の設定により、認可クライアントへの TLS/HTTPS が有効になり、HTTPS スキームを使用して Red Hat build of Keycloak サーバーにリモートでアクセスできるようになります。

注記

Red Hat build of Keycloak サーバーのエンドポイントにアクセスするときは、TLS/HTTPS を有効にすることが強く推奨されます。