Red Hat Single Sign-On 7.6

授权服务指南

用于 Red Hat Single Sign-On 7.6

Last Updated: 2024-06-05

Red Hat Single Sign-On 7.6 授权服务指南

用于 Red Hat Single Sign-On 7.6

法律通告

摘要

本指南包含用于 Red Hat Single Sign-On 7.6 的授权服务的信息。

目录

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 授权服务概述

Red Hat Single Sign-On 支持精细的授权策略，并可组合不同的访问控制机制，例如：

- **基于属性的访问控制(ABAC)**
- **基于角色的访问控制(RBAC)**
- **基于用户的访问控制(UBAC)**
- **基于上下文的访问控制(CBAC)**
- **基于规则的访问控制**
  - 使用 JavaScript
- **基于时间的访问控制**
- **通过服务提供商接口(SPI)支持自定义访问控制机制(ACM)**

Red Hat Single Sign-On 基于一组管理 UI 和 RESTful API，提供为受保护资源和范围创建权限所需的方法，将这些权限与授权策略与授权决策相关联，并在您的应用程序和服务中实施授权决策。

资源服务器（应用程序或服务为受保护的资源）通常依赖于某种信息来确定是否应向受保护的资源授予访问权限。对于基于 RESTful 的资源服务器，这些信息通常从安全令牌获取，通常在每个请求上作为 bearer 令牌发送到服务器。对于依赖会话验证用户的 Web 应用，该信息通常存储在用户的会话中，并从那里检索每个请求。

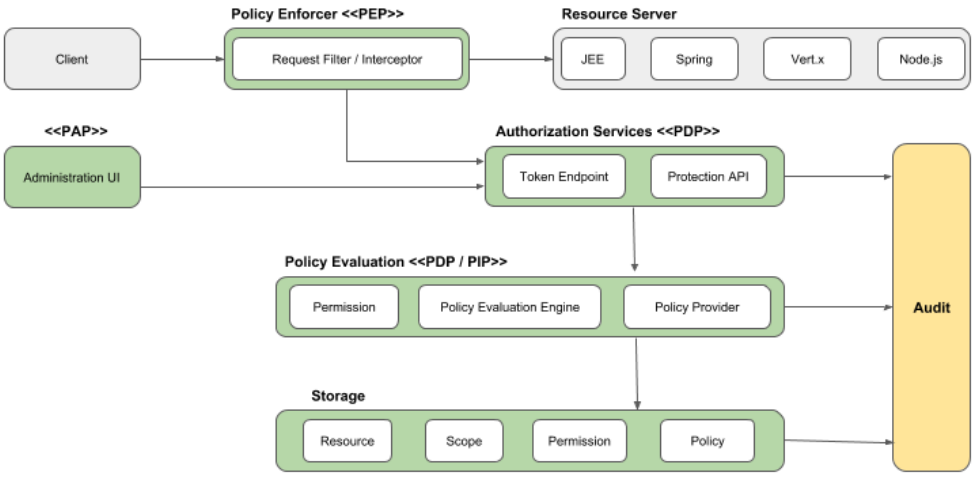
通常，资源服务器仅根据基于角色的访问控制(RBAC)执行授权决策，其中授予用户试图访问受保护的资源的角色会根据映射到这些相同资源的角色进行检查。虽然角色非常有用，并由应用程序使用，但它们有一些限制：

- 资源和角色与角色紧密耦合和更改角色（如添加、删除或更改访问上下文）可能会影响多个资源
- 对您安全要求的更改可能会对应用程序代码进行深度更改，以反映这些更改
- 根据您的应用程序大小，角色管理可能会变得困难且容易出错
- 它不是最灵活的访问控制机制。角色不表示您是和缺少上下文信息。如果您被授予角色，您至少具有一些访问权限。

我们认为，我们需要考虑在不同区域分发用户的异构环境、具有不同本地策略、使用不同的设备，以及大量信息共享的需求，红帽单点登录授权服务可以帮助您提高应用程序和服务的授权功能：

- 使用精细授权策略和不同的访问控制机制进行资源保护
- 集中资源、权限和策略管理
- 集中策略决策点
- 基于一组基于 REST 的授权服务进行 REST 安全性
- 授权工作流和用户管理的访问
- 基础架构，以帮助避免项目间的代码复制（并重新部署），并快速适应安全要求的变化。

1.1. 架构



从设计的角度，授权服务基于一组精心定义的授权模式，提供这些功能：

- **策略管理点(PAP)**
  提供一组基于 Red Hat Single Sign-On 管理控制台的 UI，以管理资源服务器、资源、范围、权限和策略。另外，这部分也可以通过使用 保护 API 进行远程完成。

- **策略决策点(PDP)**
  提供可分发策略决策点，以根据请求的权限相应地评估授权请求和策略的位置。如需更多信息，请参阅 获取权限。

- **策略实施点(PEP)**
  为不同的环境提供实现，在资源服务器端实际强制执行授权决策。Red Hat Single Sign-On 提供了一些内置 策略 Enforcers。

- **策略信息点(PIP)**
  根据 Red Hat Single Sign-On Authentication Server，您可以在评估授权策略期间从身份和运行时环境获取属性。

1.1.1. 授权过程

三个主要进程定义了了解如何使用 Red Hat Single Sign-On 为应用程序启用精细授权所需的步骤：

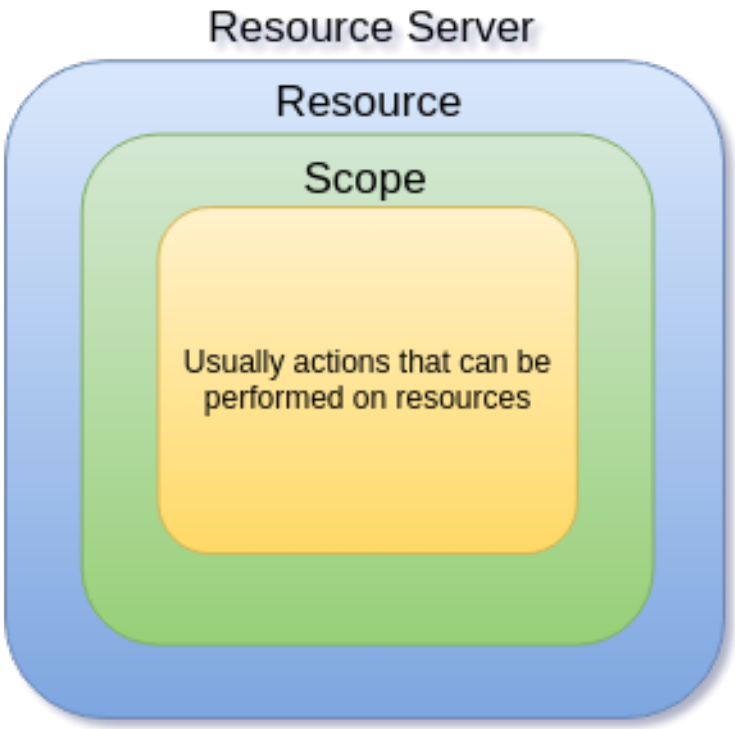
- **资源管理**
- **权限和策略管理**
- **策略强制**

1.1.1.1. 资源管理

资源管理 涉及所有必要的步骤来定义受保护的内容。

首先，您需要指定您要保护的 Red Hat Single Sign-On，它代表一个 Web 应用程序或一组或多个服务。有关资源服务器的更多信息，请参阅 术语。

资源服务器使用红帽单点登录管理控制台管理。您可以启用任何注册的客户端应用程序作为资源服务器，并开始管理您要保护的资源和范围。



资源可以是网页、RESTFul 资源、文件系统中的文件、EJB 等。它们可以代表一组资源（就像 Java 中的类）或代表一个和特定资源。

例如，您可能有一个代表所有 *银行* 帐户的银行帐户资源，并使用它来定义对所有银行帐户通用的授权策略。但是，您可能想为 *Alice 帐户* （属于客户的资源实例）定义特定的策略，其中只有所有者可以访问某些信息或执行操作。

可使用 Red Hat Single Sign-On 管理控制台或 保护 API 管理资源。在后者的情况下，资源服务器可以远程管理其资源。

范围通常代表可以对资源执行的操作，但它们不限于该资源。您还可以使用范围来代表资源中的一个或多个属性。

1.1.1.2. 权限和策略管理

在定义了资源服务器以及您要保护的所有资源后，您必须设置权限和策略。

此过程涉及所有必要步骤，以实际定义管理资源的安全和访问要求。

策略定义了访问或对某些对象（资源或范围）执行操作的条件，但它们不会与受到保护的内容关联。它们是通用的，可以重复利用以构建权限或更复杂的策略。

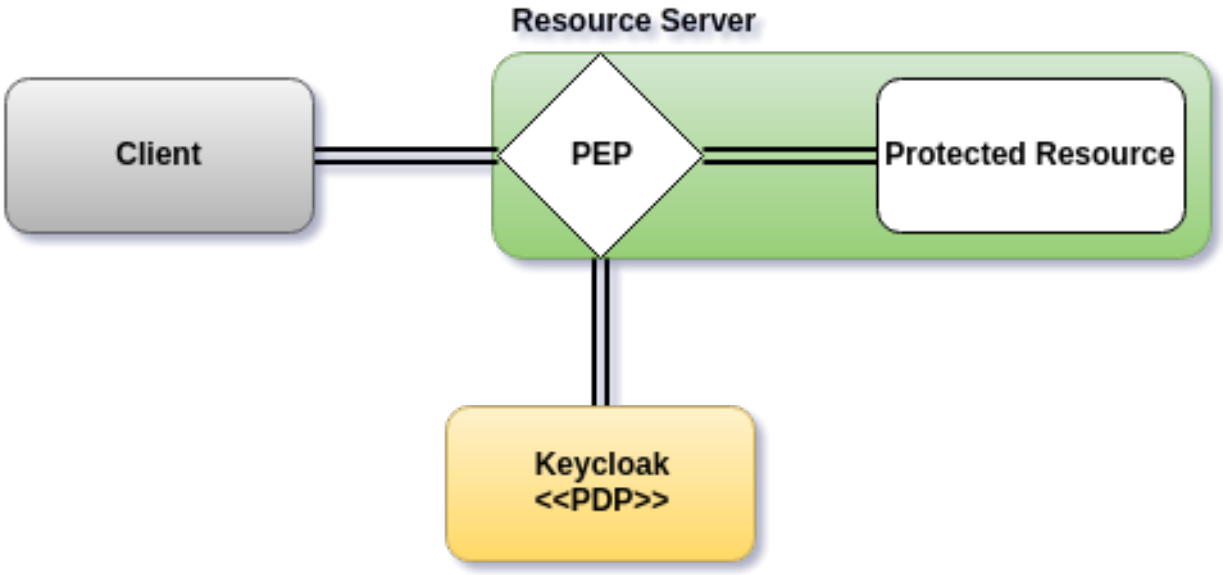
例如，若要只允许一组资源访问由角色"用户高级"授予的用户，您可以使用 RBAC （基于角色的访问控制）。

Red Hat Single Sign-On 提供一些内置策略类型（及其相应策略供应商）涵盖了最常见的访问控制机制。您甚至可以根据使用 JavaScript 编写的规则创建策略。

定义了策略后，您可以开始定义权限。权限与其保护的资源相结合。在这里，您需要指定您要保护的内容（资源或范围）以及必须满足授予或拒绝权限的策略。

1.1.1.3. 策略强制

策略强制 涉及实际对资源服务器实施授权决策的必要步骤。这可以通过在能够与授权服务器通信的资源服务器中启用 **策略强制点** 或 PEP 实现，请求授权数据并根据服务器返回的决策和权限控制对受保护资源的访问。



Red Hat Single Sign-On 提供了一些内置 策略强制实施，您可以根据它们运行的平台来保护应用程序。

1.1.2. 授权服务

授权服务包括以下 RESTFul 端点：

- **令牌端点**
- **资源管理端点**
- **权限管理端点**

这些服务各自提供一种特定的 API，涵盖了授权流程中涉及的不同步骤。

1.1.2.1. 令牌端点

OAuth2 客户端（如前端应用）可以使用令牌端点从服务器获取访问令牌，并将这些令牌用于访问由资源服务器（如后端服务）保护的资源。同样，Red Hat Single Sign-On Authorization Services 为 OAuth2 提供扩展，允许根据处理与请求资源关联的所有策略来发布访问令牌。这意味着资源服务器可以根据服务器获得的权限强制访问其受保护的资源，并由访问令牌持有。在 Red Hat Single Sign-On Authorization Services 中，具有权限的访问令牌被称为 Requesting unsupported Token 或 RPT for short。

其他资源

- 获取权限

1.1.2.2. 保护 API

Protection API 是一组 兼容 UMA 的端点，为资源服务器提供操作，以帮助它们管理与其关联的资源、范围、权限和策略。只有资源服务器可以访问此 API，它还需要 **uma_protection** 范围。

保护 API 提供的操作可以分为两个主要组：

- **资源管理**
  - 创建资源
  - 删除资源
  - Id 查找
  - 查询
- **权限管理**
  - 问题 Permission Tickets

注意

默认情况下启用 Remote Resource Management。您可以使用 Red Hat Single Sign-On 管理控制台进行更改，并只允许通过控制台进行资源管理。

使用 UMA 协议时，保护 API 的 Permission Tickets 的确是整个授权过程的重要组成部分。如后续小节中所述，它们代表客户端所请求的权限，并发送到服务器来获取最终令牌，以便在评估与请求的资源和范围关联的权限和策略时获得所有权限。

其他资源

- Protection API

1.2. 术语

在进一步之前，务必要了解 Red Hat Single Sign-On Authorization Services 中引入的这些术语和概念。

1.2.1. 资源服务器

根据 OAuth2 术语，资源服务器是托管受保护的资源的服务器，能够接受和响应受保护的资源请求。

资源服务器通常依赖于某种信息来确定是否应被授予对受保护资源的访问权限。对于基于 RESTful 的资源服务器，该信息通常采用安全令牌，通常作为 bearer 令牌与服务器的每个请求一起发送。依赖于会话对用户进行身份验证的 Web 应用通常会将该信息存储在用户的会话中，并从那里为每个请求检索信息。

在红帽单点登录中，任何 **机密** 客户端应用程序都可以充当资源服务器。此客户端的资源及其对应的范围受到一组授权策略的影响。

1.2.2. 资源

资源是应用程序和组织的资产的一部分。它可以是一个或多个端点、一个典型的 Web 资源，如 HTML 页面等。在授权策略术语中，资源是受保护的 *对象*。

每一资源具有可表示单个资源或一组资源的唯一标识符。例如，您可以管理一个 *代表所有银行帐户的银行客户资源* 并定义一组授权策略。但是，您也可以拥有另外一个名为 *Alice 的银行帐户* 资源，它代表了单个客户拥有的单一资源，这些资源可以拥有自己的授权策略。

1.2.3. 影响范围

资源的范围是可以在资源上执行的限额访问范围。在授权策略术语中，范围是可能的很多 *动词*，可逻辑上应用到资源。

它通常表示可通过给定资源执行哪些操作。范围示例为 view、edit、delete 等。但是，范围也可以与资源提供的特定信息相关。在这种情况下，您可以具有项目资源和成本范围，其中成本范围用于定义用户访问项目成本的具体策略和权限。

1.2.4. 权限

请考虑简单和非常常见的权限：

权限将对象与必须评估的策略关联，以确定是否授予访问权限。

- **X** 可在资源 **Z** 上执行 **Y**
    - 其中 ...
        - **X** 代表一个或多个用户、角色或组，或者两者的组合。您也可以在此处使用声明和上下文。
        - **Y** 代表要执行的操作，例如：写入、查看等等。
        - **z** 代表受保护的资源，如 "/accounts"。

Red Hat Single Sign-On 提供丰富的平台，用于构建一系列权限策略，范围从简单到非常复杂的、基于规则的动态权限。它提供了灵活性并帮助：

- 降低代码重构和权限管理成本
- 支持更灵活的安全模式，帮助您轻松适应安全要求的变化
- 在运行时进行更改；应用程序只关注其受保护的资源和范围，而不如何保护它们。

1.2.5. 策略

策略定义了必须满足以授予对象访问权限的条件。与权限不同，您不指定对象受到保护，而是对给定对象（如资源、范围或两者）满足的条件。策略强烈与可用于保护资源的不同访问控制机制(ACM)相关。通过策略，您可以实施基于属性的访问控制(ABAC)、基于角色的访问控制(RBAC)、基于上下文访问控制或这

些组合的任意组合。

Red Hat Single Sign-On 利用策略的概念，以及如何提供聚合策略的概念来定义它们，您可以在其中构建"策略策略"并仍控制评估行为。在 Red Hat Single Sign-On Authorization Services 中，在 Red Hat Single Sign-On Authorization Services 中实施策略，而不是在所有必须满足给定资源的条件下编写大型策略。也就是说，您可以创建单独的策略，然后使用不同的权限重新利用它们，并通过组合单个策略来构建更复杂的策略。

1.2.6. 策略供应商

策略提供程序是特定策略类型的实现。Red Hat Single Sign-On 提供内置策略，由相应的策略供应商支持，您可以创建自己的策略类型来支持您的特定要求。

Red Hat Single Sign-On 提供了一个 SPI （服务提供商接口），您可以使用它来插入您自己的策略供应商实施。

1.2.7. 权限 ticket

权限 ticket 是由 User-Managed Access (UMA)规范定义的特殊令牌类型，它提供由授权服务器决定的不透明结构。这个结构代表了客户端所请求的资源和/或范围、访问上下文以及必须应用到授权数据请求的策略（请求方令牌 [RPT]）。

在 UMA 中，权限票据对于支持个人共享以及个人组织共享至关重要。使用授权工作流的权限问题单可让一系列从简单到复杂情况，而资源所有者和资源服务器可根据管理这些资源访问的精细策略完全控制其资源。

在 UMA 工作流中，授权服务器向资源服务器发布权限票据，它会将权限票据返回到尝试访问受保护的资源的客户端。客户端收到票据后，它可以通过向授权服务器发送 ticket 来为 RPT （一个最终令牌保留授权数据）发出请求。

有关权限票据的更多信息，请参阅 用户管理的访问 和 UMA 规格。

第 2 章 开始使用

在使用本指南前，您需要完成安装 Red Hat Single Sign-On 并创建初始 admin 用户，如 入门指南 中所示。这里有一个注意事项。您必须在与 Red Hat Single Sign-On Server 相同的计算机上运行单独的 JBoss EAP 实例。此独立的实例将运行 Java Servlet 应用。因此，您需要在不同端口下运行 Red Hat Single Sign-On，以便在同一台机器中运行任何端口冲突。在命令行中使用 **jboss.socket.binding.port-offset** 系统属性。此属性的值是一个数字，将添加到由 Red Hat Sign-On Server 打开的每个端口的基本值中。

要引导 Red Hat Single Sign-On 服务器：

Linux/Unix

```
$ .../bin/standalone.sh -Djboss.socket.binding.port-offset=100
```

Windows

```
> ...\bin\standalone.bat -Djboss.socket.binding.port-offset=100
```

安装和引导这两个服务器后，您应能够访问位于 http://localhost:8180/auth/admin/ 的 Red Hat Single Sign-On Admin Console，以及 JBoss EAP 实例，网址为 http://localhost:8080。

其他资源

- 有关安装和配置 JBoss EAP 实例的详情，请参阅保护应用程序和服务指南。

2.1. 保护 SERVLET 应用程序

此入门指南的目的是尽快启动并运行，以便您可以试验和测试 Red Hat Single Sign-On 提供的各种授权功能。这个快速导览依赖于默认数据库和服务器配置，且不涵盖复杂的部署选项。有关功能或配置选项的更多信息，请参阅本文档中的相应章节。

本指南说明了有关 Red Hat Single Sign-On 授权服务的关键概念：

- 为客户端应用程序启用精细授权
- 将客户端应用程序配置为资源服务器，带有受保护的资源
- 定义用于监管对受保护资源的访问权限和授权策略
- 在应用程序中启用策略执行。

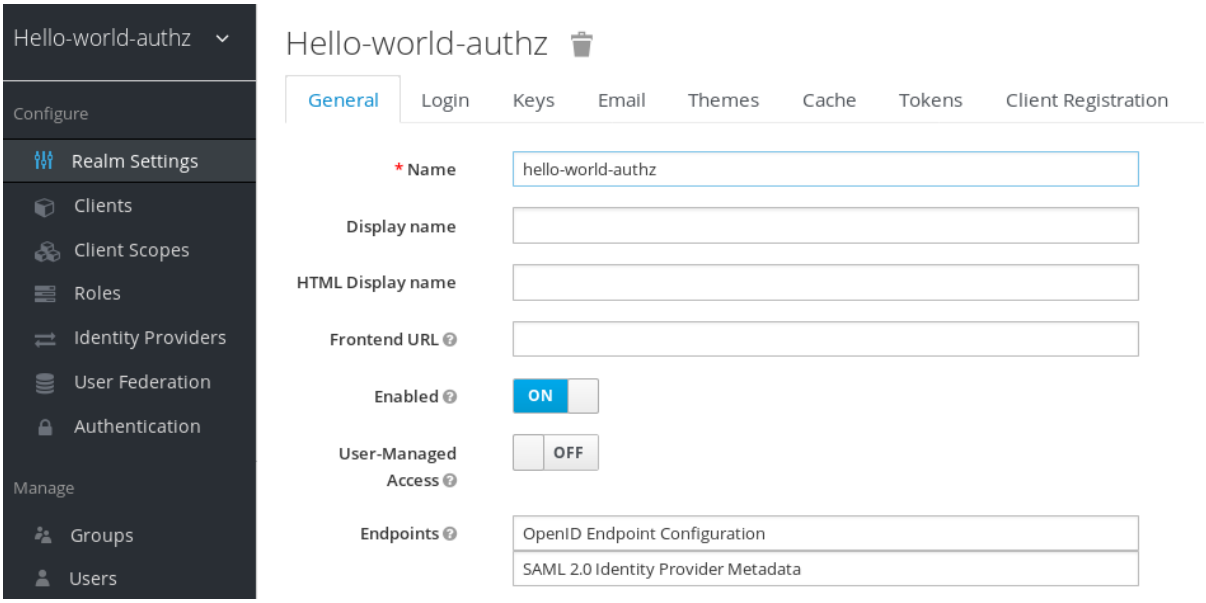
2.2. 创建域和用户

本教程中的第一步是在该域中创建一个域和用户。然后，在域中，我们将创建一个客户端应用程序，然后成为您需要启用授权服务 的资源服务器。

流程

1. 创建名为 **hello-world-authz** 的域。创建后，会显示类似如下的页面：

   realm hello-world-authz



2. 点 **Users**。
   用户列表页面显示您可以创建用户的位置。

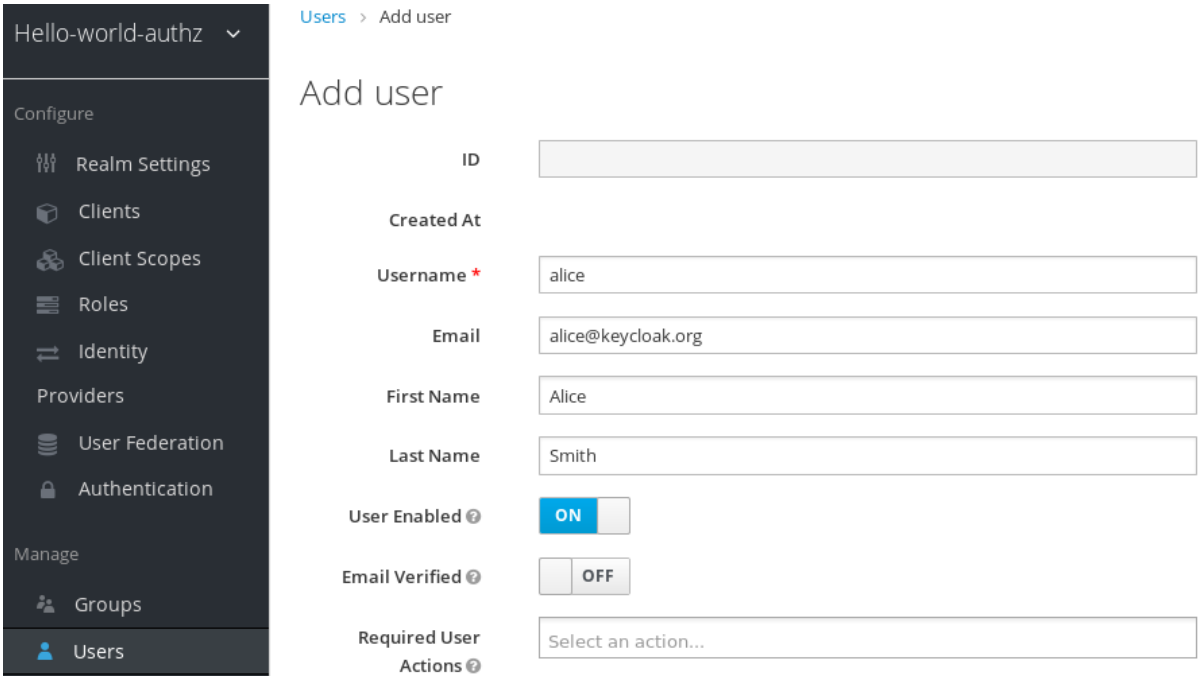
3. 单击 **添加用户**。

4. 填写 **Username**, **Email**, **First Name**, 和 **Last Name** 字段。

5. 将 **User Enabled** 切换到 **ON**

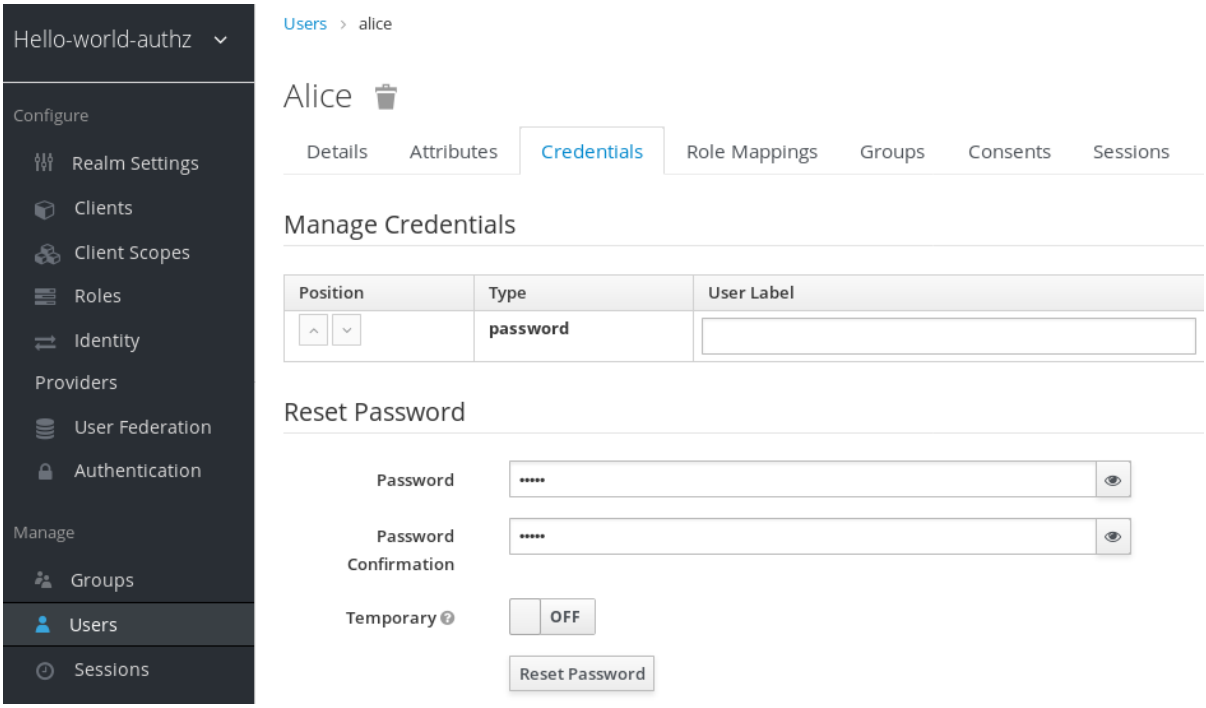
6. 点 **Save**。

   添加用户



7. 点 **Credentials** 选项卡为用户设置密码。

   设**置用**户**密**码



8. 使用密码完成 **New Password** 和 **Password Confirmation** 字段，然后单击 **Temporary** 开关为 **OFF**。

9. 单击 **Set Password** 来设置用户的密码。

2.3. 启用授权服务

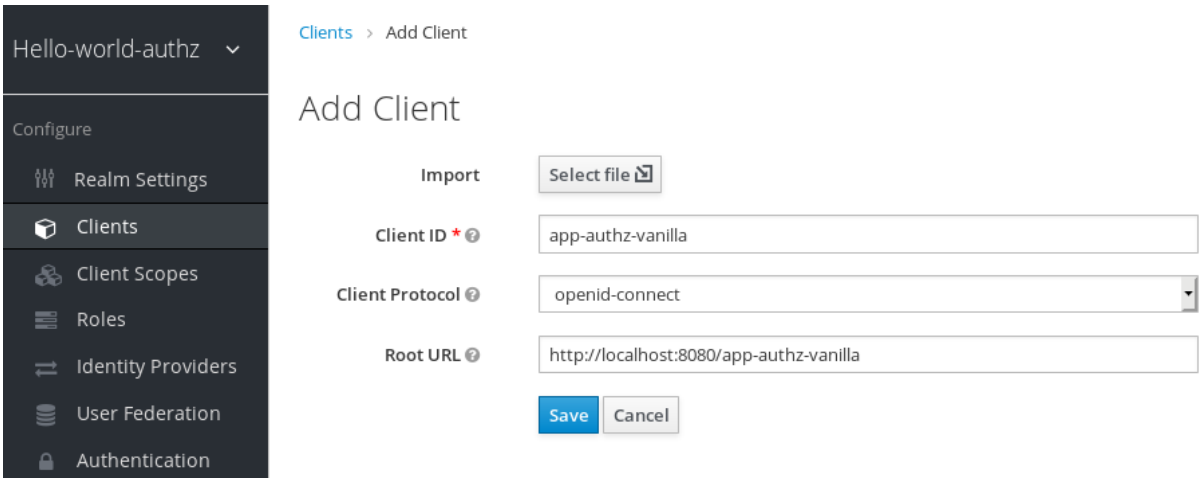
您可以在配置为使用 OpenID Connect 协议的现有客户端应用程序中启用授权服务。您还可以使用以下步骤创建客户端。

流程

1. 点 **Clients** 开始创建客户端应用程序。

2. 填写 **客户端 ID**、**客户端协议**和 **根 URL** 字段。

   创建客户端应用程序

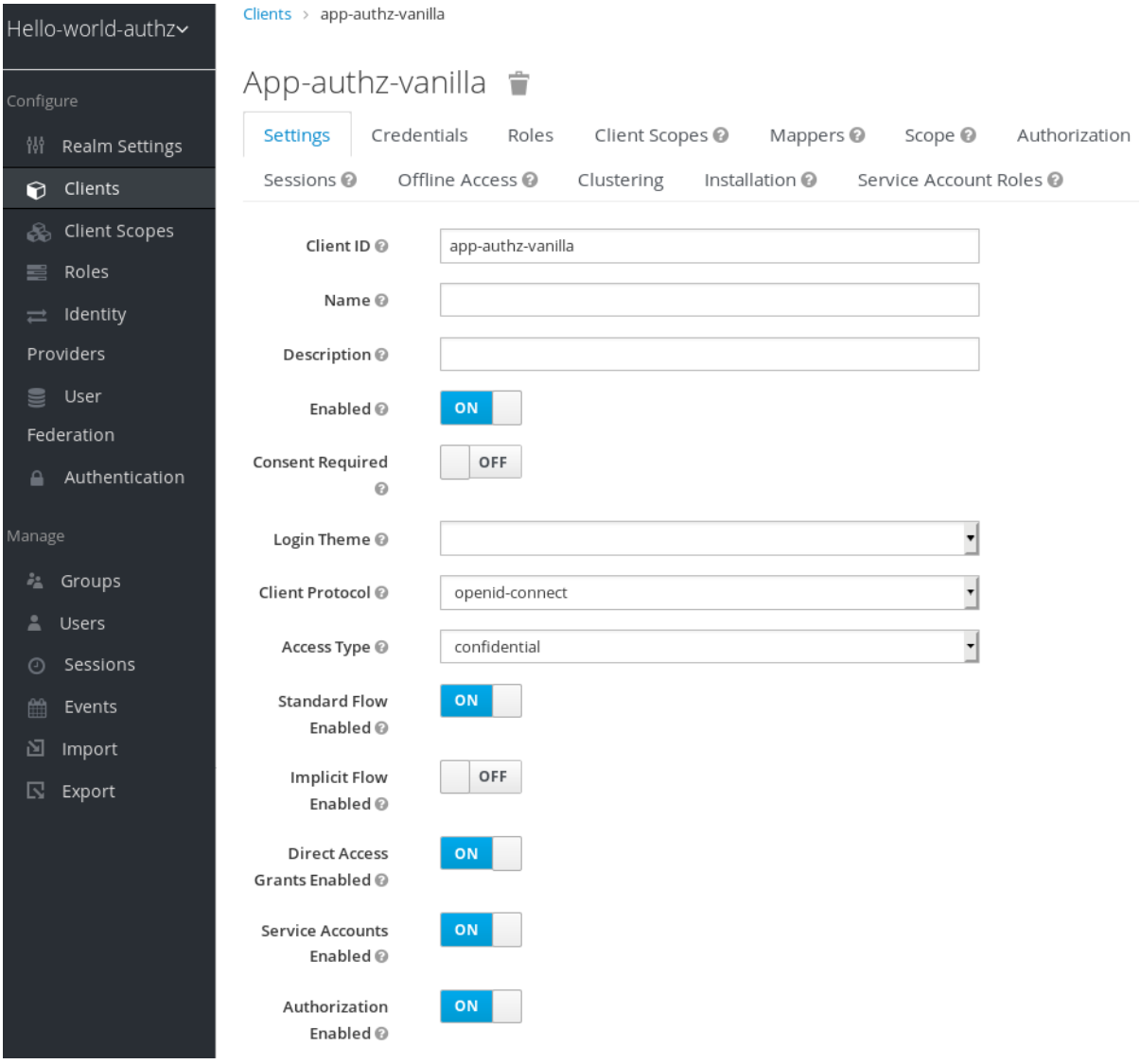


3. 点 **Save**。
   此时会显示 Client Settings 页面。

4. 在 **Access Type** 字段中选择 **confidential**，将 **Authorization Enabled** 切换到 **ON**

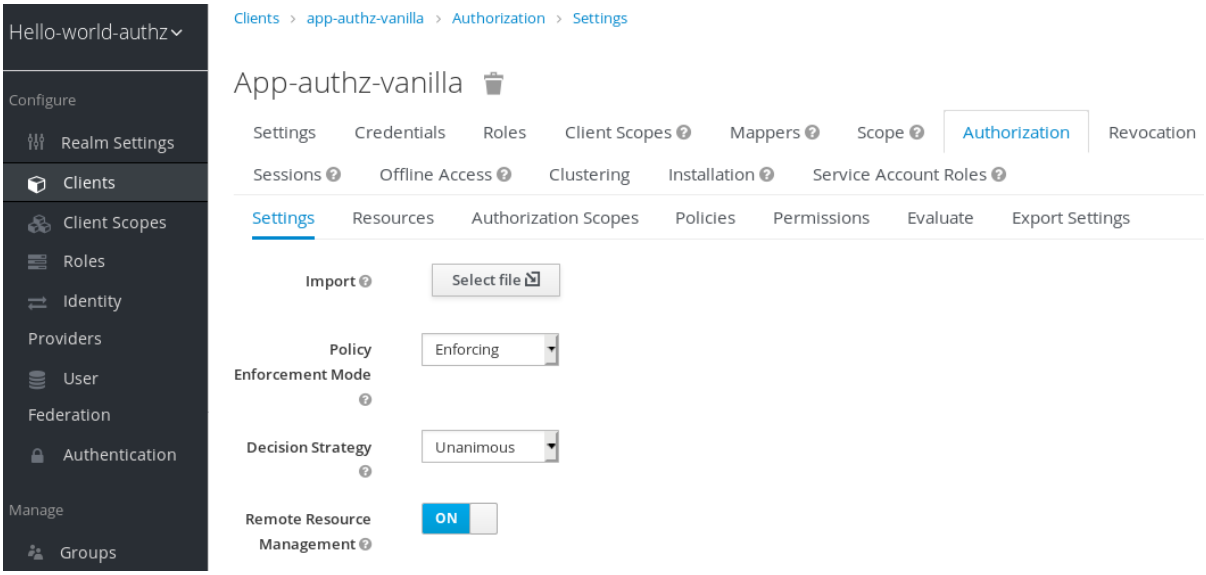
5. 点 **Save**。
   为客户端显示一个新的 **Authorization** 选项卡。

   客户端设置



6. 点 **Authorization** 选项卡。
   此时会显示类似如下的 Authorization Settings 页面：

   授权设置



当您为客户端应用程序启用授权服务时，Red Hat Single Sign-On 会自动为客户端授权配置创建几个默认设置。

其他资源

- 启用授权服务
- 默认配置

2.4. 构建、部署和测试您的应用程序

现在，**app-authz-vanilla** 资源服务器（或客户端）已正确配置和授权服务被启用，可以将其部署到服务器。

Red Hat Single Sign-On Quickstarts Repository 中提供了您要部署的应用程序的项目和代码。您需要在您的机器中安装以下内容，并在 PATH 中可用，然后才能继续：

- Java JDK 8
- Apache Maven 3.1.1 或更高版本
- Git

您可以通过克隆软件仓库（位于 https://github.com/redhat-developer/redhat-sso-quickstarts ）来获取代码。使用与正在使用的红帽单点登录版本匹配的分支。

按照以下步骤下载代码。

克隆项目

```
$ git clone https://github.com/redhat-developer/redhat-sso-quickstarts
```

我们即将构建和部署的应用位于

```
$ cd redhat-sso-quickstarts/app-authz-jee-vanilla
```

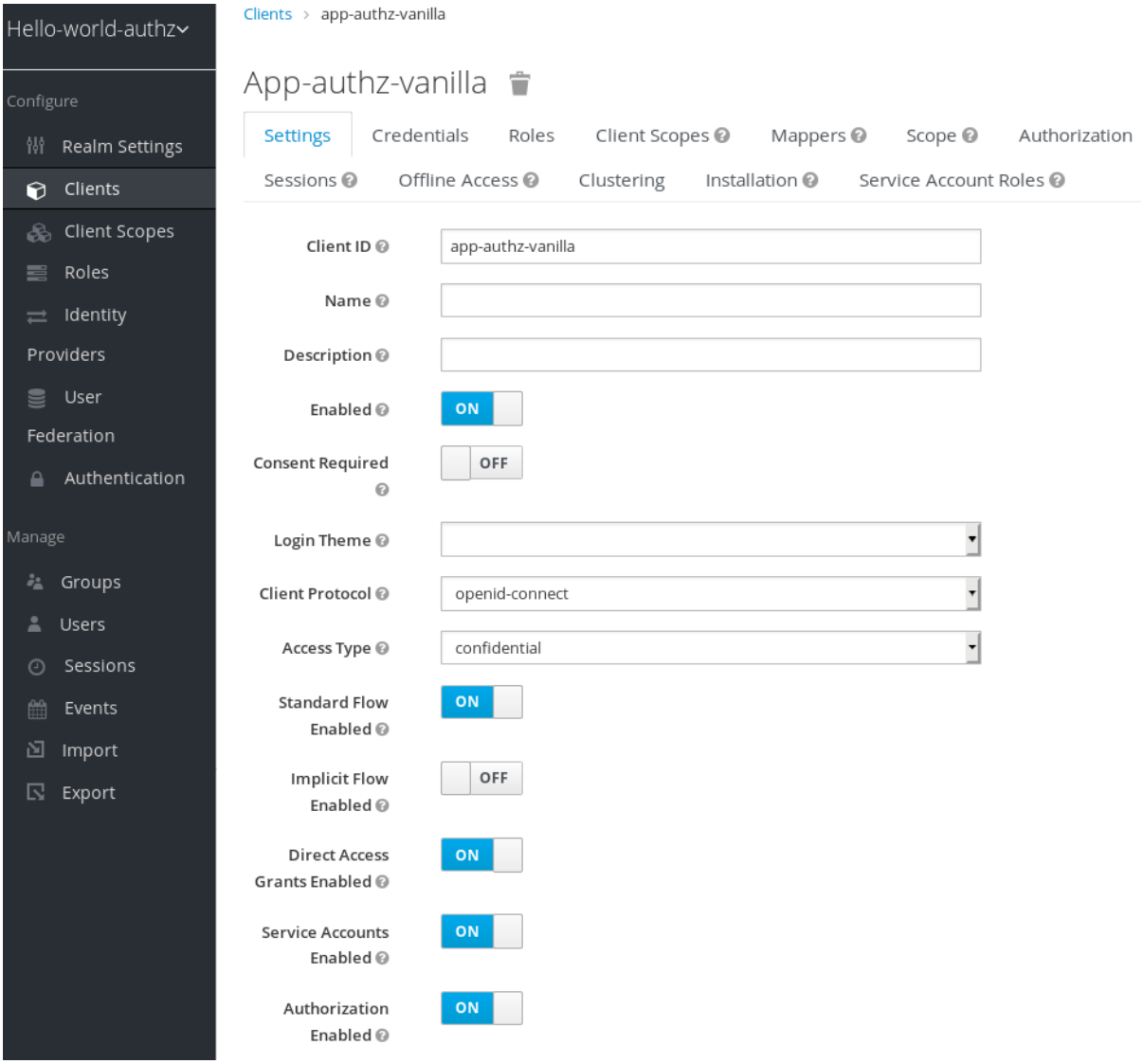
2.4.1. 获取适配器配置

在构建和部署应用程序前，您必须首先获取适配器配置。

流程

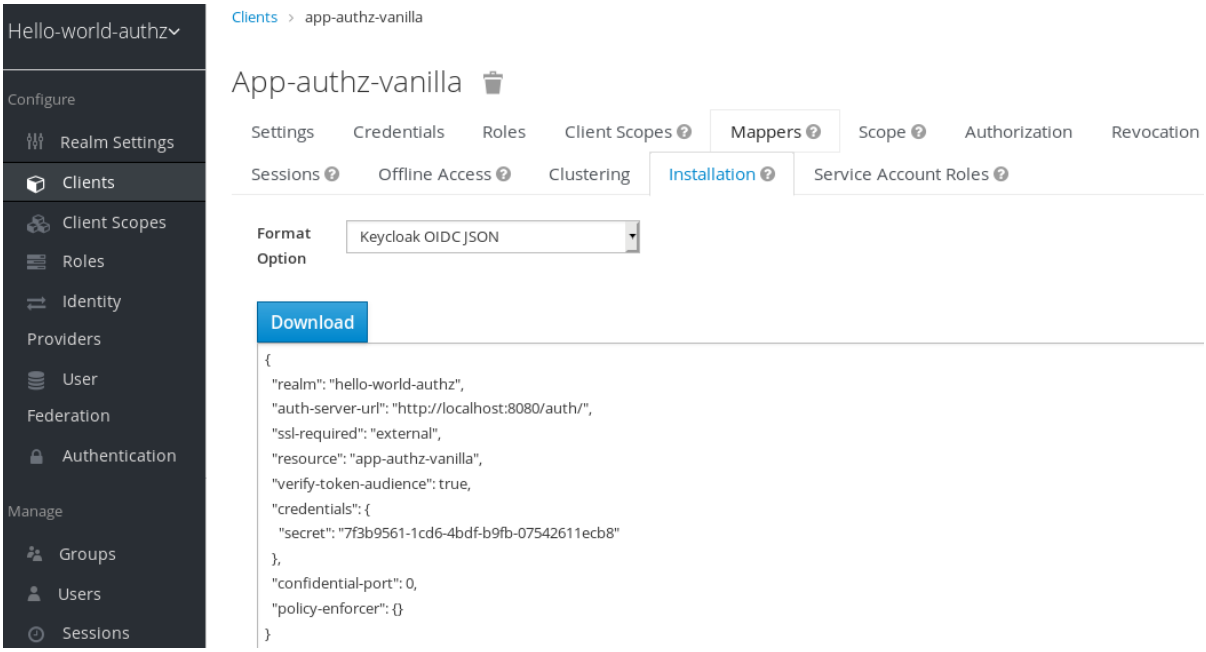
1. 登录 Admin 控制台。
2. 点菜单中的 **Clients**。
3. 在客户端列表中，单击 **app-authz-vanilla** 客户端应用。此时会打开 Client Settings 页面。

   客户端设置



4. 点 **Installation** 选项卡。
5. 在 Format Options 项中选择 **Keycloak OIDC JSON**。
   适配器配置以 JSON 格式显示。
6. 点 **Download**。

   适配器配置



7. 将文件 **keycloak.json** 移到 **app-authz-jee-vanilla/config** 目录。

8. （可选）指定重定向 URL。
默认情况下，当用户在资源服务器上访问受保护的资源的权限时，策略强制r 用 **403** 状态代码进行响应。但是，您还可以为未授权的用户指定重定向 URL。要指定重定向 URL，请编辑您更新的 **keycloak.json** 文件，并使用以下内容替换 **policy-enforcer** 配置：

```
"policy-enforcer": {
    "on-deny-redirect-to" : "/app-authz-vanilla/error.jsp"
}
```

如果用户没有访问受保护资源所需的权限，则这个更改会指定将用户重定向到 **/app-authz-vanilla/error.jsp** 页面，而不是返回一个 **403 Unauthorized** 信息。

2.4.2. 构建和部署应用程序

要构建和部署应用程序，请执行以下命令：

```
$ cd redhat-sso-quickstarts/app-authz-jee-vanilla
$ mvn clean package wildfly:deploy
```

2.4.3. 测试应用程序

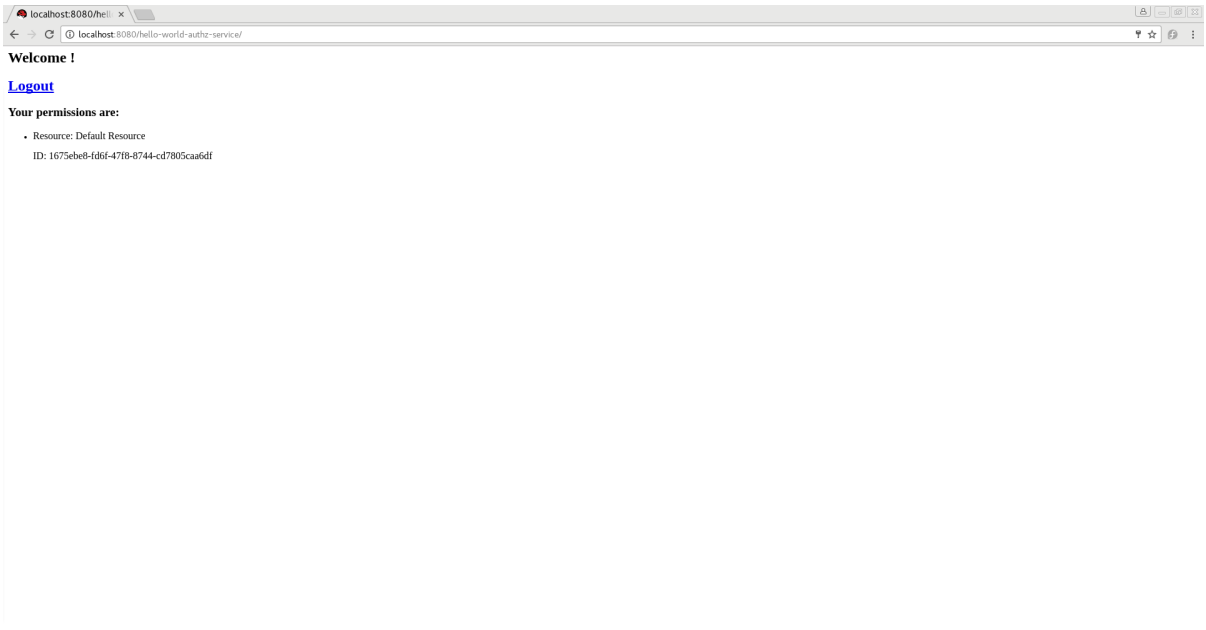
如果您的应用程序已被成功部署，您可以在 http://localhost:8080/app-authz-vanilla 访问它。Red Hat Single Sign-On Login 页面将打开。

登录页面

流程

1. 以 alice 身份登录，使用您为该用户指定的密码。此时会显示以下页面：

   hello World Authz 主页

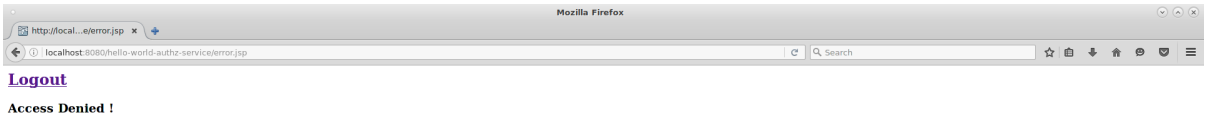
   

   当您为客户端应用程序启用授权服务时，红帽单点登录定义 的默认设置 提供了一个简单策略，始终授予对此策略保护的资源的访问权限。

您可以首先更改默认权限和策略，并测试应用程序如何响应，甚至使用 Red Hat Single Sign-On 提供的不同 策略类型 创建新策略。

现在您可以做很多事情来测试此应用程序。例如，您可以通过单击客户端的 **Authorization** 选项卡来更改默认策略，然后单击 **Policies** 选项卡中的客户端，然后单击列表中的 **Default Policy**。现在，我们将使用此页面中的下拉列表将 **Logic** 改为 **Negative**。

1. 从演示应用程序注销，然后再次登录。
   您无法再访问应用程序。



其他资源

- 策略类型

2.4.4. 后续步骤

您可以执行其他操作，例如：

- 创建范围，为它定义策略和权限，并在应用程序侧进行测试。用户可以执行操作（或由您创建的范围代表的任何其他项）？
- 创建不同类型的策略，并将这些策略与 **Default Permission** 相关联。
- 将多个策略应用到 **默认权限** 并测试其行为。例如，组合多个策略并相应地更改 **决策策略**。

其他资源

- 有关如何查看和测试应用程序内的权限的更多信息，请参阅 获取授权上下文。

2.5. 授权快速入门

除了上一节中作为示例应用程序使用的 **app-authz-jee-vanilla** quickstart 外， Red Hat Single Sign-On Quickstarts Repository 包含了使用本文档中描述的授权服务的其他应用。

授权快速入门已被设计，以便在不同的场景中显示授权服务，并使用不同的技术和集成。它并非是涉及授权的所有可能用例的综合用例，而是应该为有意了解授权服务在其自己的应用程序中使用授权服务的用户提供起点。

每个快速入门都有一个 **README** 文件，其中包含如何构建、部署和测试示例应用程序的说明。下表提供了可用授权快速入门的简要描述：

表 2.1. 授权快速入门

Name	描述
app-authz-jee-servlet	演示如何为 Jakarta EE 应用启用精细的授权，以保护特定资源并根据从红帽单点登录服务器获取的权限构建动态菜单。
app-authz-jee-vanilla	演示如何为 Jakarta EE 应用启用精细的授权，并使用默认授权设置来保护应用中的所有资源。
app-authz-rest-springboot	演示如何使用 Red Hat Single Sign-On Authorization Services 保护 SpringBoot REST 服务。
app-authz-springboot	演示如何编写 SpringBoot Web 应用程序，其中身份验证和授权方面都由 Red Hat Single Sign-On 管理。
app-authz-uma-photoz	基于 HTML5+AngularJS+JAX-RS 的简单应用，它演示了如何启用用户管理访问应用程序并允许用户管理其资源的权限。

第 3 章 管理资源服务器

根据 OAuth2 规范，资源服务器是托管受保护的资源的服务器，能够接受和响应受保护的资源请求。

在 Red Hat Single Sign-On 中，提供丰富的平台，为受保护的资源提供精细的授权，根据不同的访问控制机制进行授权决策。

任何客户端应用程序都可以配置为支持精细的权限。这样，您概念将客户端应用程序转变为资源服务器。

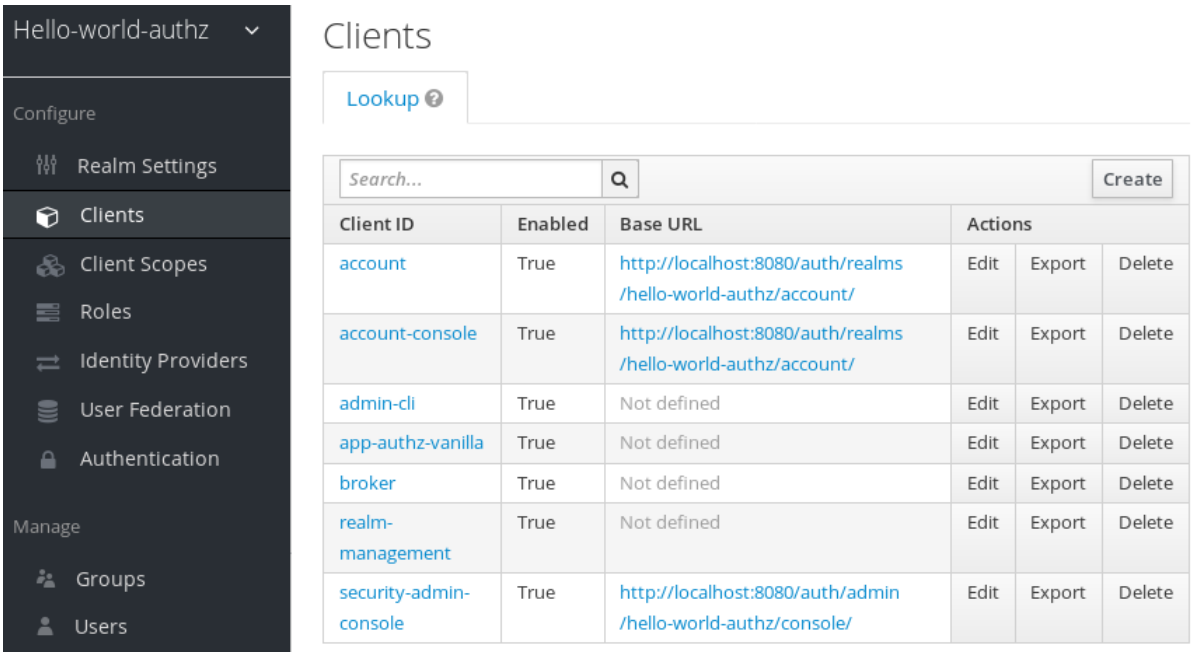
3.1. 创建客户端应用程序

启用 Red Hat Single Sign-On Authorization Services 的第一步是创建您要切换到资源服务器的客户端应用程序。

流程

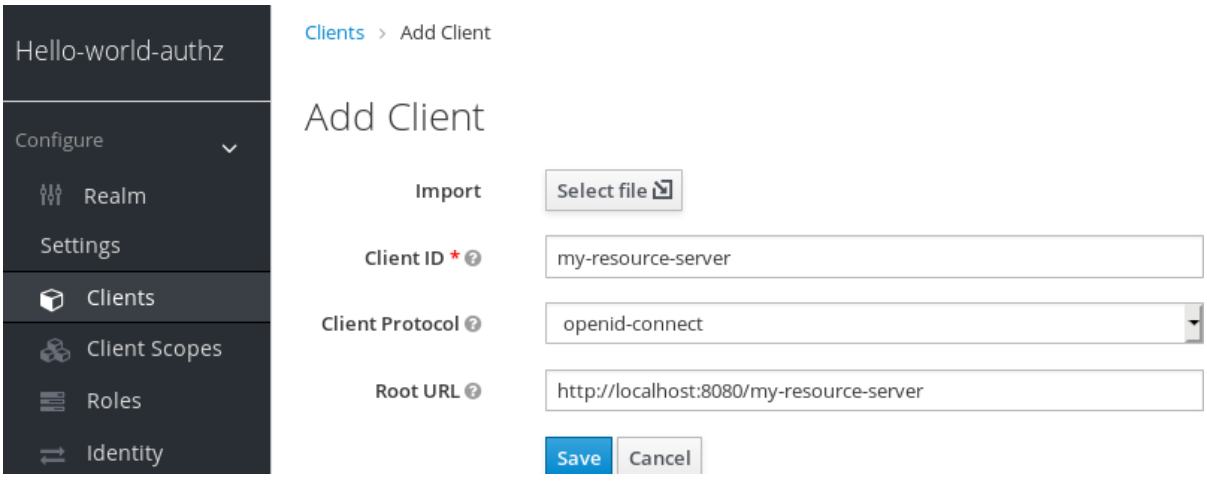
1. 点 **Clients**。

客户端



2. 在此页面上，单击 **Create**。

添加客户端



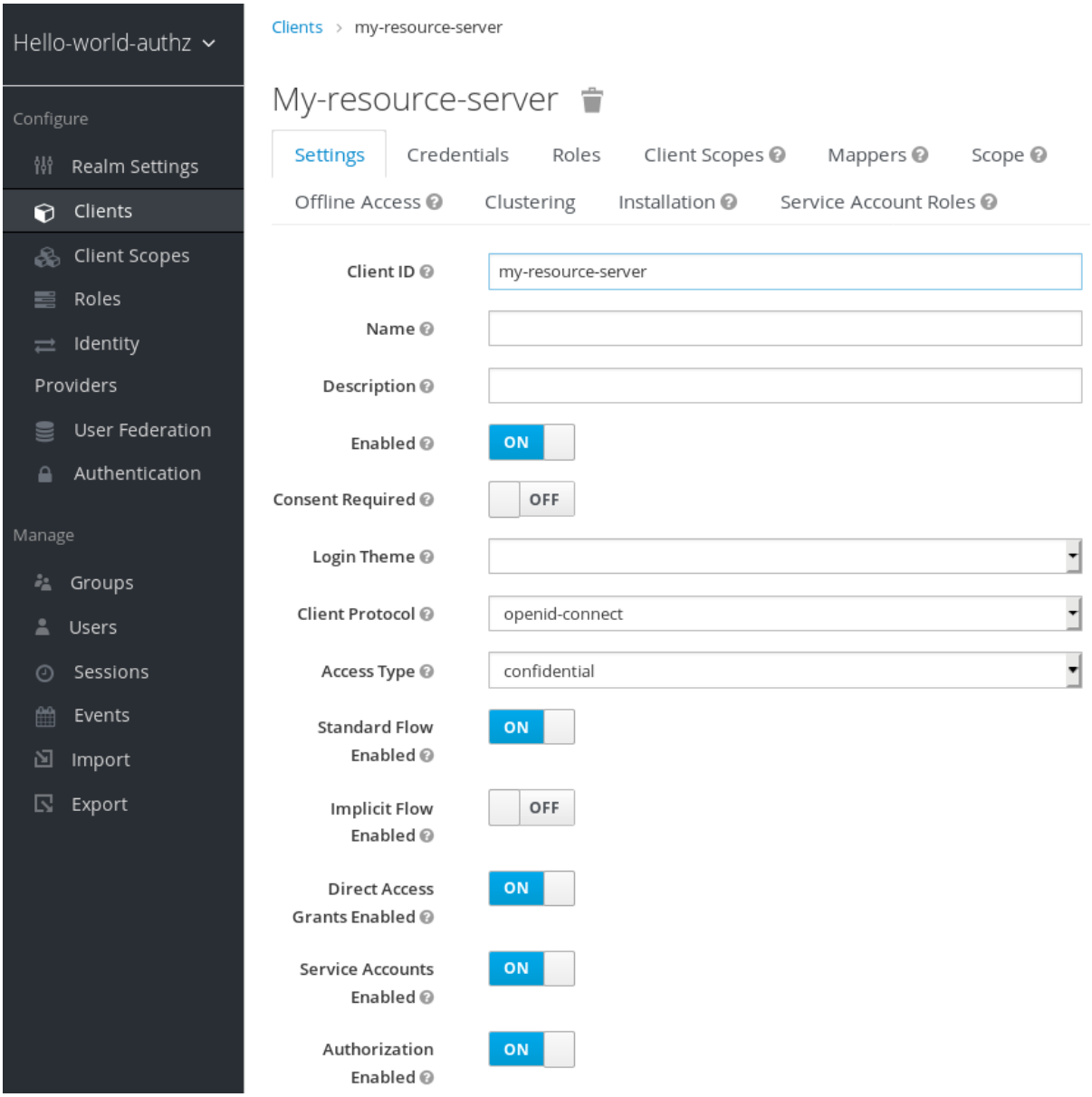
3. **键入客户端的客户端 ID**。例如，*my-resource-server*。

4. 输入应用程序的 **Root URL**。例如：

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

5. 点 **Save**。客户端已创建，客户端 Settings 页面将打开。此时会显示类似如下的页面：

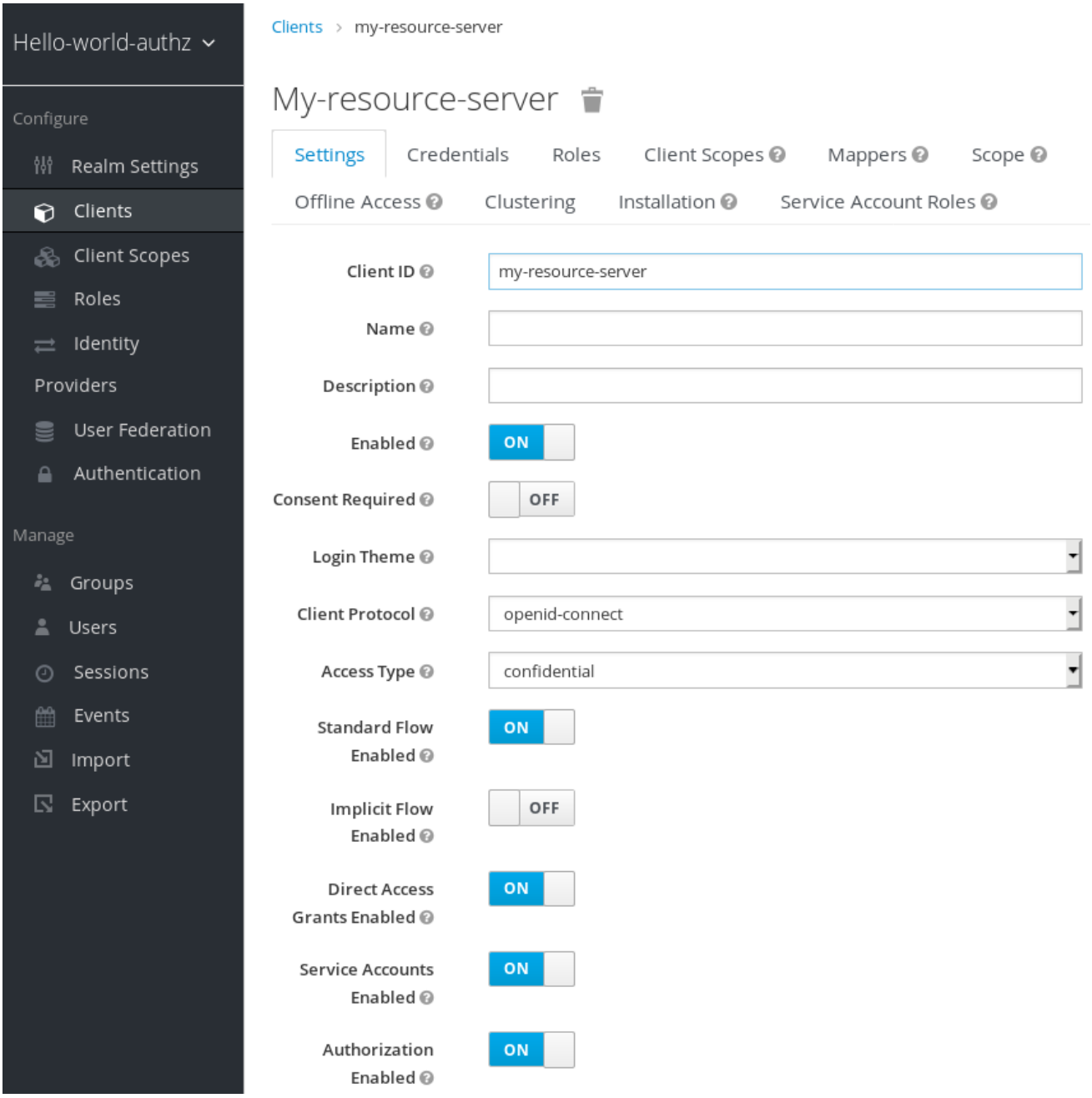
客户端设置



3.2. 启用授权服务

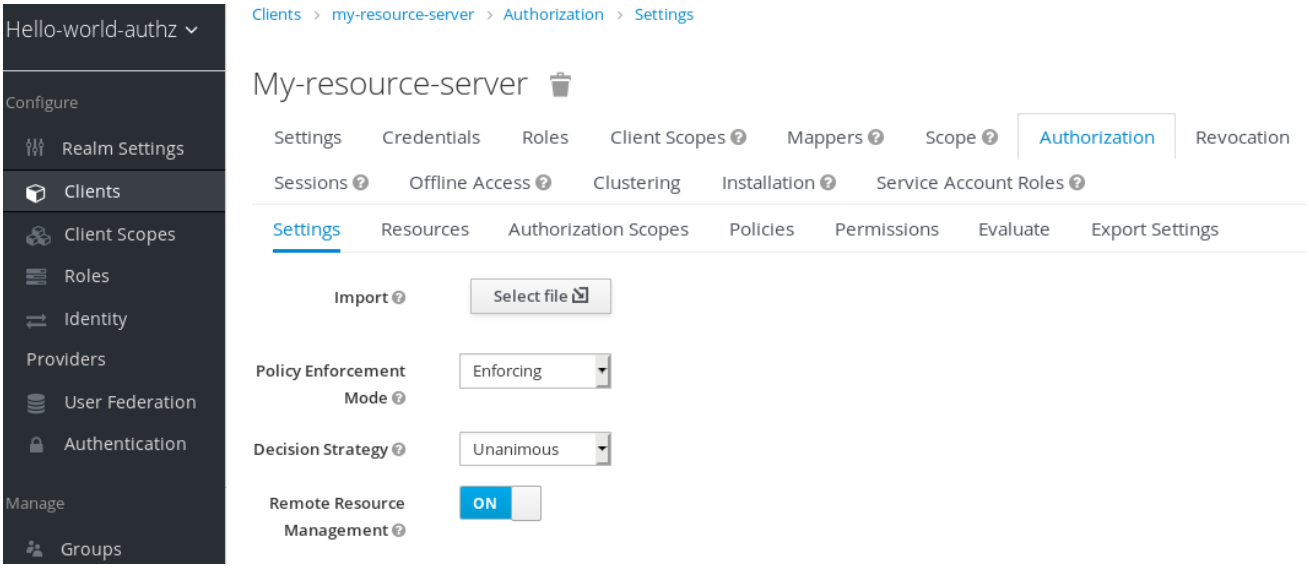
要将 OIDC Client Application 转变为资源服务器并启用精细的授权，请选择 **Access type confidential**，然后点击 **Authorization Enabled** switch to **ON**，然后点 **Save**。

启用授权服务



此客户端会显示一个新的 Authorization 选项卡。点击 **Authorization** 标签页，会显示类似如下的页面：

资**源服**务**器**设**置**



Authorization 选项卡包含额外的子选项卡，其中涵盖了您必须遵循的不同步骤来真正保护应用程序的资源。每个标签页分别在本文档的特定主题中单独介绍。但以下是每个描述的快速描述：

- **设置**
  资源服务器的常规设置。有关此页面的详情，请查看 Resource Server Settings 部分。
- **资源**
  在本页中，您可以管理 应用程序的资源。
- **授权范围**
  在此页面中，您可以管理 范围。
- **策略（policy）**
  在这个页面中，您可以管理 授权策略，并定义授予权限的条件。
- **权限**
  在这个页面中，您可以通过将保护的资源 的权限 与您创建的策略链接到您的保护资源和范围。
- **评估**
  在此页面中，您可以 模拟授权请求 并查看您定义的权限和授权策略评估结果。
- **导出设置**
  在此页面中，您可以将授权设置 导出到 JSON 文件。

3.2.1. 资源服务器设置

在 Resource Server Settings 页面中，您可以配置策略强制模式，允许远程资源管理，并导出授权配置设置。

- **策略强制模式**
  指定在处理发送到服务器的授权请求时强制实施策略。
  - **Enforcing**
    （默认模式）默认拒绝请求，即使没有与给定资源关联的策略。
  - **Permissive**
    即使没有与给定资源关联的策略，也会允许请求。
  - **Disabled**

禁用所有策略并允许访问所有资源。

- **决策策略**
  此配置改变了策略评估引擎如何根据所有评估权限的结果来决定资源或范围是否被授予相应。**Affirmative** 表示，至少一个权限必须评估积极决策，才能授予对资源及其范围的访问权限。**不确定** 所有权限必须评估积极决策才能最终决定。例如，如果相同资源或范围的两个权限都存在冲突（其中之一被授予访问权限，另一个则拒绝访问），如果选择策略为 **Affirmative**，则将授予其对资源或范围的权限。否则，任何权限中的单个拒绝也会拒绝对资源或范围的访问。

- **远程资源管理**
  指定资源是否可以由资源服务器远程管理。如果为 false，则只能从管理控制台管理资源。

3.3. 默认配置

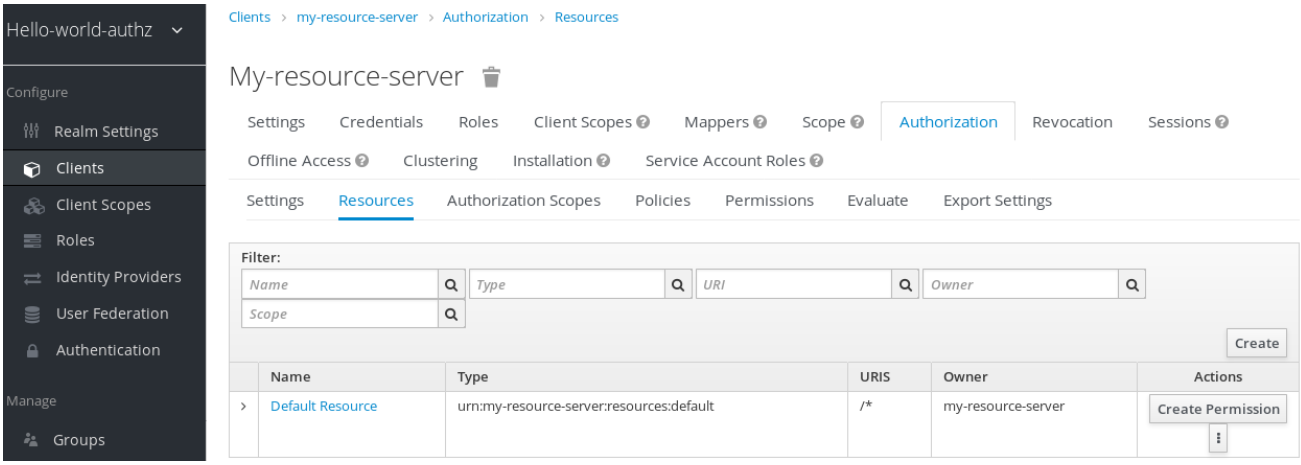
当您创建资源服务器时，Red Hat Single Sign-On 为新创建的资源服务器创建默认配置。

默认配置包括：

- 代表应用程序中的所有资源的默认保护资源。
- 始终授予对受此策略保护的资源的访问权限的策略。
- 根据默认策略管理对所有资源的访问权限的权限。

默认受保护的资源称为 **默认资源**，如果您导航到 **Resources** 选项卡，您可以查看它。

默认资源

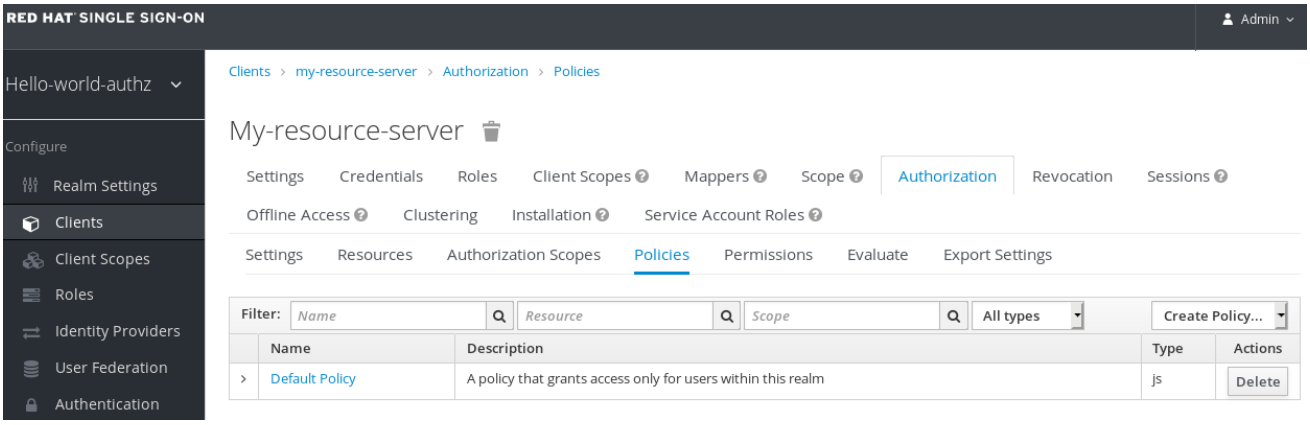


此资源定义一个 **Type**，即 **urn:my-resource-server:resources:default** 和一个 **URI** /*。这里的 **URI** 字段定义了一个通配符模式，它指示该资源代表应用程序中的所有路径的 Red Hat Single Sign-On。换句话说，当为应用程序启用 策略强制 时，将在授予访问权限前检查与该资源关联的所有权限。

以前提到的 **Type** 定义了一个值，可用于创建 类型的资源权限，该权限必须应用到默认资源或您使用相同的类型创建的任何其他资源。

默认策略称为 **唯一的域策略**，如果您进入到 **Policies** 选项卡，可以查看该策略。

默认策略

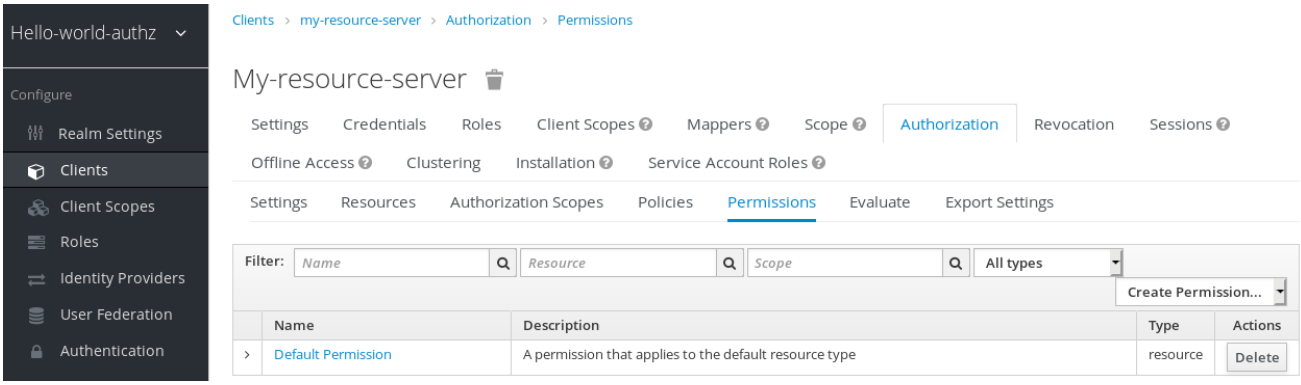


此策略是基于 JavaScript 的策略，它始终授予对此策略保护的资源的访问权限。如果点击此策略，可以看到它定义了规则，如下所示：

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

最后，默认权限称为**默认权限**，您可以通过 **Permissions** 标签页来查看它。

默认权限



此权限是基于资源的权限，定义一个或多个策略的集合，适用于具有给定类型的所有资源。

3.3.1. 更改默认配置

您可以通过删除默认资源、策略或权限定义并创建您自己的来更改默认配置。

默认资源使用 URI 创建，该 **URI** 使用 **/*** 模式映射到应用程序中的任何资源或路径。在创建属于您自己的资源、权限和策略之前，请确保默认配置不会与您自己的设置冲突。

注意

默认配置定义了一个资源，它映射到应用程序中的所有路径。如果要为自己的资源写入权限，请务必删除 **Default Resource**，或将其 **URIS** 字段更改为应用程序中更为具体的路径。否则，与默认资源关联的策略（默认情况下始终授予访问权限），Red Hat Single Sign-On 将允许 Red Hat Single Sign-On 授予任何受保护的资源的访问权限。

3.4. 导出和导入授权配置

可以导出和下载资源服务器（或客户端）的配置设置。您还可以为资源服务器导入现有的配置文件。当您想要为资源服务器创建初始配置或更新现有配置时，可以导入和导出配置文件。配置文件包含以下方面的定义：

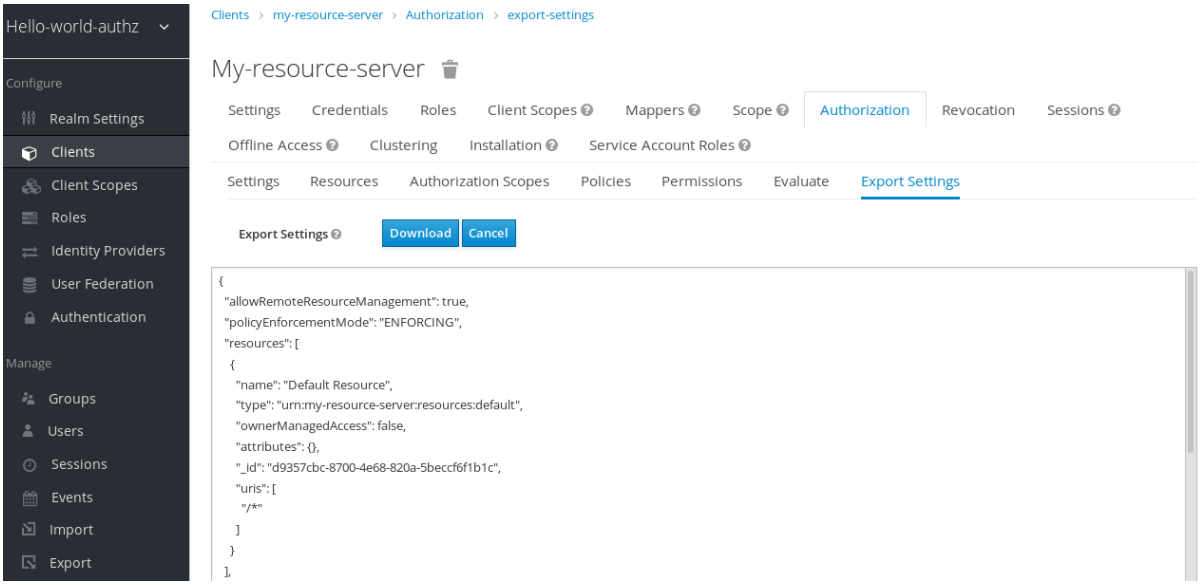
- 保护的资源和范围
- 策略（policy）
- 权限

3.4.1. 导出配置文件

流程

1. 导航到 **Resource Server Settings** 页面。
2. 点 **Export Settings** 选项卡。
3. 在此页面上，单击 **Export**。

导**出**设**置**



配置文件以 JSON 格式导出，并显示在文本区域中，您可以复制和粘贴。您还可以点 **Download** 以下载配置文件并保存它。

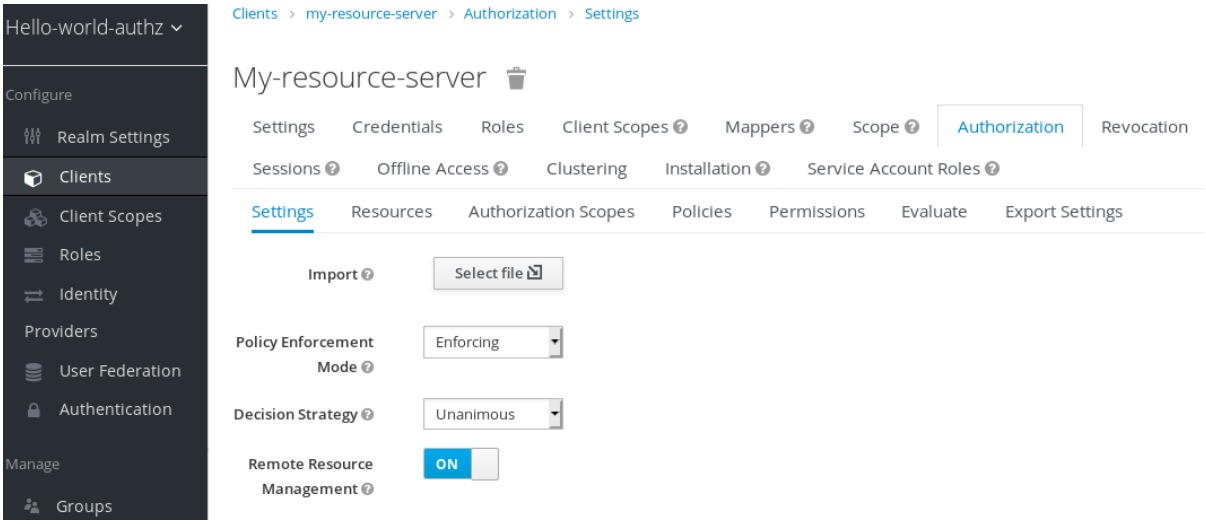
3.4.2. 导入配置文件

您可以为资源服务器导入配置文件。

流程

1. 导航到 **Resource Server Settings** 页面。

导**入**设**置**



2. 点 **Select file** 并选择一个包含您要导入的配置的文件。

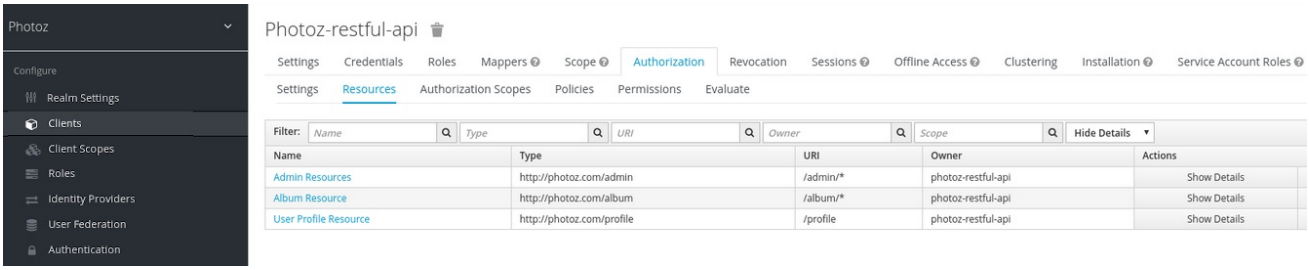
第 4 章 管理资源和范围

资源管理简单且通用。创建资源服务器后，您可以开始创建您要保护的资源和范围。可以通过分别导航到 **Resource** 和 **Authorization Scopes** 选项卡来管理资源和范围。

4.1. 查看资源

在 **Resource** 页面中，您会看到与资源服务器关联的资源列表。

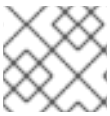
Resources



资源列表提供有关受保护的资源的信息，例如：

- 类型
- URIS
- 所有者
- 相关范围（若有）
- 关联的权限

从此列表中，您还可以直接通过点击您要为其创建权限的资源 **Create Permission** 来创建权限。



注意

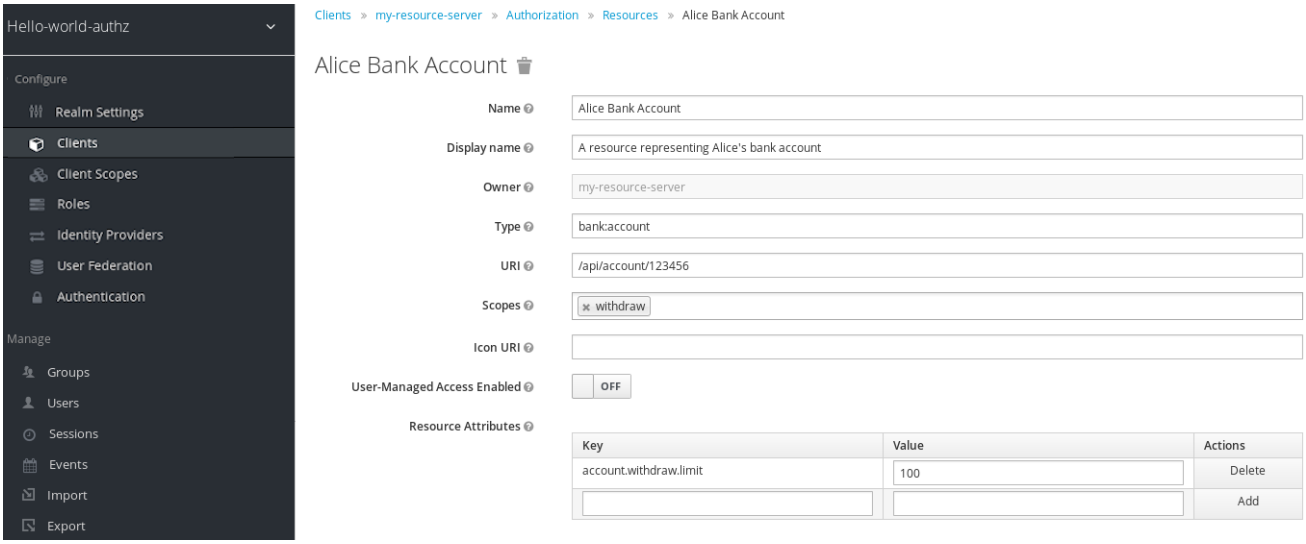
在为您的资源创建权限前，请确定您已定义了您要与权限关联的策略。

4.2. 创建资源

创建资源非常简单且通用。您主要关注的是您创建的资源。换句话说，可以创建资源来代表一个或多个资源的集合，而且您定义它们的方式对于管理权限至关重要。

若要创建新资源，可点资源列表右上角的 **Create**。

添加资源



在 Red Hat Single Sign-On 中，资源定义了一组常见不同类型的资源的信息，例如：

- **Name**
  描述此资源的人类可读和唯一字符串。
- **类型**
  一个字符串，用于唯一标识一组或多个资源类型。类型 *是一个字符串*，用于对不同的资源实例进行分组。例如，自动创建的默认资源的类型是 **urn:resource-server-name:resources:default**
- **URIS**
  为资源提供位置/地址的 URIS。对于 HTTP 资源，URIS 通常是用于提供这些资源的相对路径。
- **范围**
  与资源关联的一个或多个范围。

4.2.1. 资源属性

资源可以关联有属性。这些属性可用于提供有关资源的其他信息，并在评估与资源关联的权限时向策略提供额外的信息。

每个属性是一个键值对，值可以是一个或多个字符串的集合。可以通过逗号分隔每个值来为属性定义多个值。

4.2.2. 输入的资源

资源的 type 字段可用于将不同的资源分组在一起，从而可以使用一组常用权限进行保护。

4.2.3. 资源所有者

资源还具有所有者。默认情况下，资源归资源服务器所有。

但是，资源也可以与用户关联，因此您可以根据资源所有者创建权限。例如，只有资源所有者才可以删除或更新给定资源。

4.2.4. 远程管理资源

资源管理也可以通过 保护 API 公开，以允许资源服务器远程管理其资源。

在使用 Protection API 时，可以实施资源服务器来管理用户所拥有的资源。在这种情况下，您可以指定用户标识符将资源配置为属于特定用户。

注意

Red Hat Single Sign-On 提供资源服务器完全控制其资源。在未来，我们应当能够允许用户控制自己的资源，以及批准授权请求和管理权限，特别是在使用 UMA 协议时。

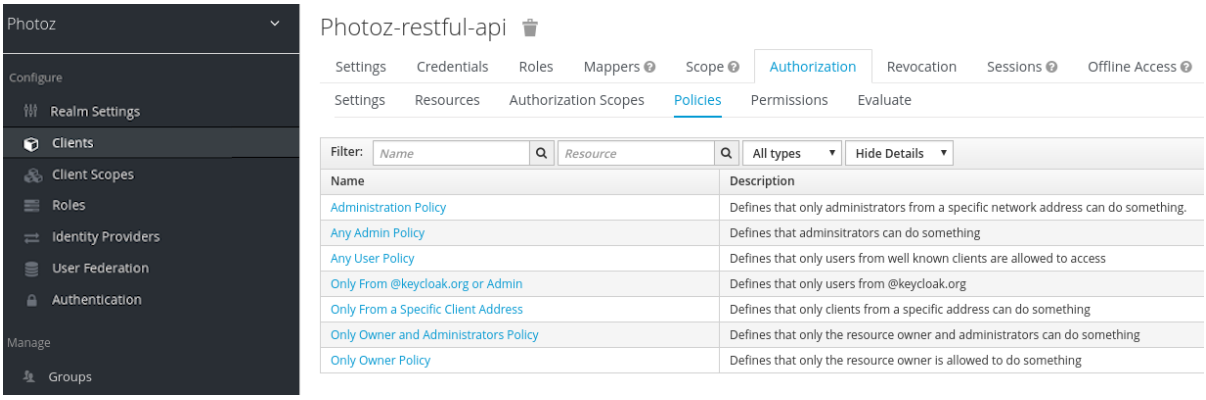
第 5 章 管理策略

如前所述，策略定义在授予对象访问权限前必须满足的条件。

流程

1. 点 **Policy** 选项卡查看与资源服务器关联的所有策略。

   策略（policy）



   在这个标签页中，您可以查看之前创建的策略列表，并创建和编辑策略。

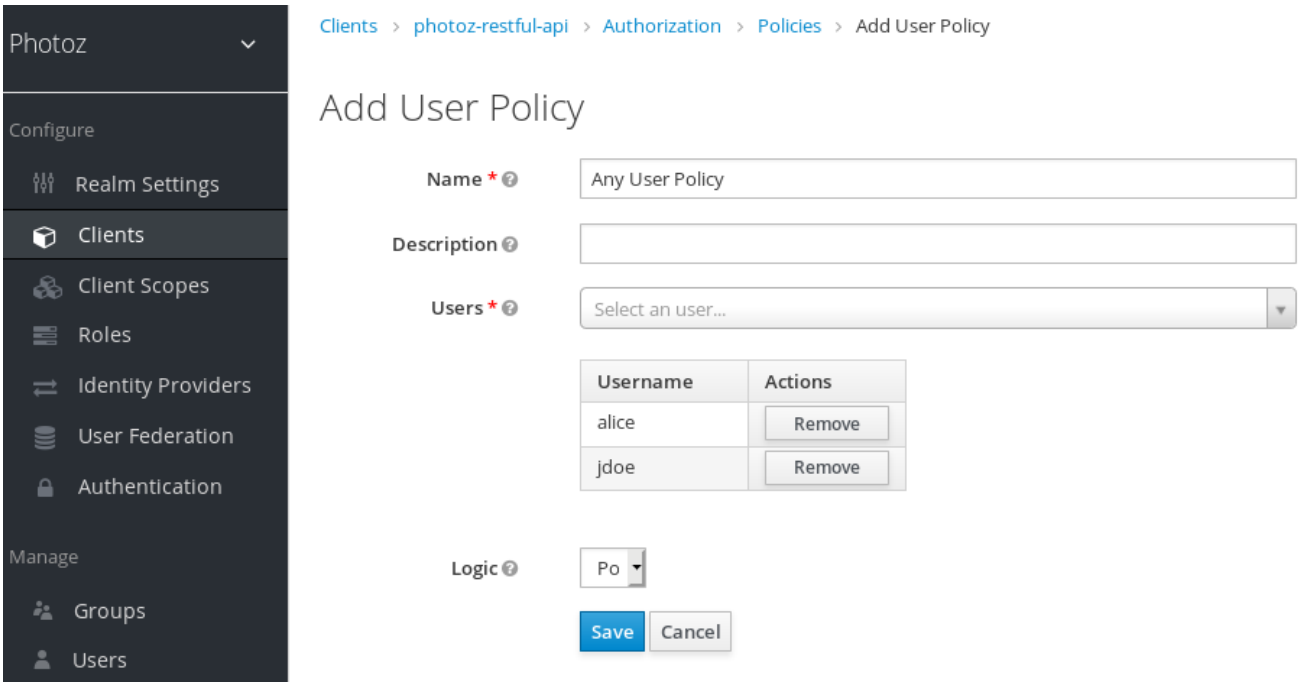
2. 要创建新策略，请从右上角的 **Create policy** item 列表中选择策略类型。
   本节描述了每种策略类型的详情。

5.1. 基于用户的策略

您可以使用这类策略为您的权限定义条件，其中允许一组用户或其他用户访问对象。

要创建基于用户的新策略，请在策略列表右上角的项列表中选择 **User**。

添加用户策略



5.1.1. Configuration

- **Name**
  标识策略的用户可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**
  包含此策略详情的字符串。

- **用户**
  指定此策略可以访问哪些用户。

- **逻辑**
  在评估其他条件后应用此策略的逻辑。

其他资源

- 正和负逻辑

5.2. 基于角色的策略

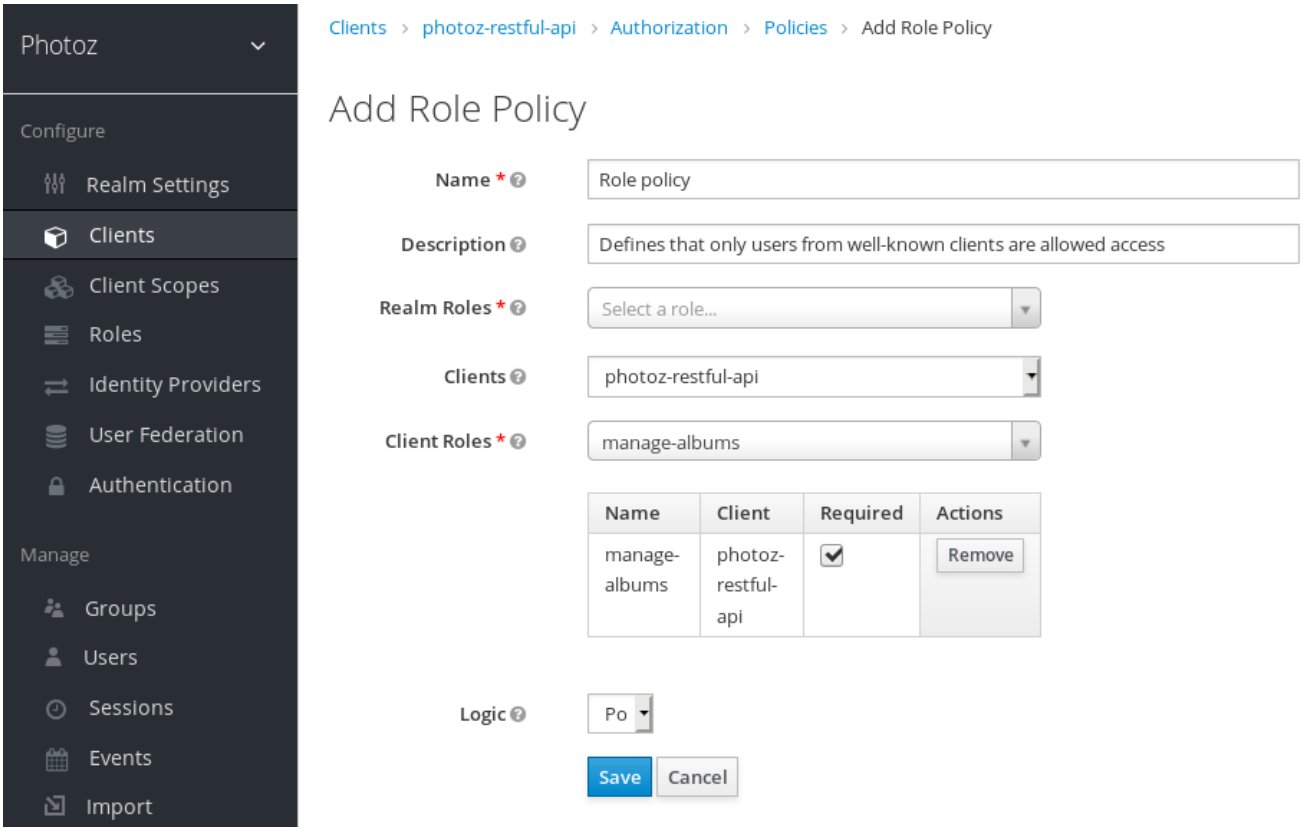
您可以使用这类策略为您的权限定义条件，其中允许一组或多个角色访问对象。

默认情况下，不会根据需要指定添加到此策略中的角色，如果被授予了任何这些角色的用户请求访问权限，策略将授予访问权限。但是，如果要强制执行特定的角色，您可以根据需要指定一个特定的角色为 required。您还可以组合所需的和非必需角色，无论它们是 realm 还是客户端角色。

当您需要更受限的基于角色的访问控制(RBAC)时，角色策略很有用，其中必须强制特定的角色才能授予对象访问权限。例如，您可以强制用户同意允许客户端应用程序（代表用户）访问用户的资源。您可以使用 Red Hat Single Sign-On Client Scope 映射来启用同意页面，甚至强制客户端在从 Red Hat Single Sign-On 服务器获取访问令牌时显式提供范围。

若要创建新的基于角色的策略，可在策略列表右上角的项列表中选择 **Role**。

添加角色策略



5.2.1. Configuration

- **Name**
  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**
  包含此策略详情的字符串。

- **realm Roles**
  指定哪些 **realm** 角色被这个策略允许。

- **客户端角色**
  指定哪些 **client** 角色被这个策略允许。要启用此字段，必须首先选择一个 **Client**。

- **逻辑**
  在评估其他条件后应用此策略的逻辑。

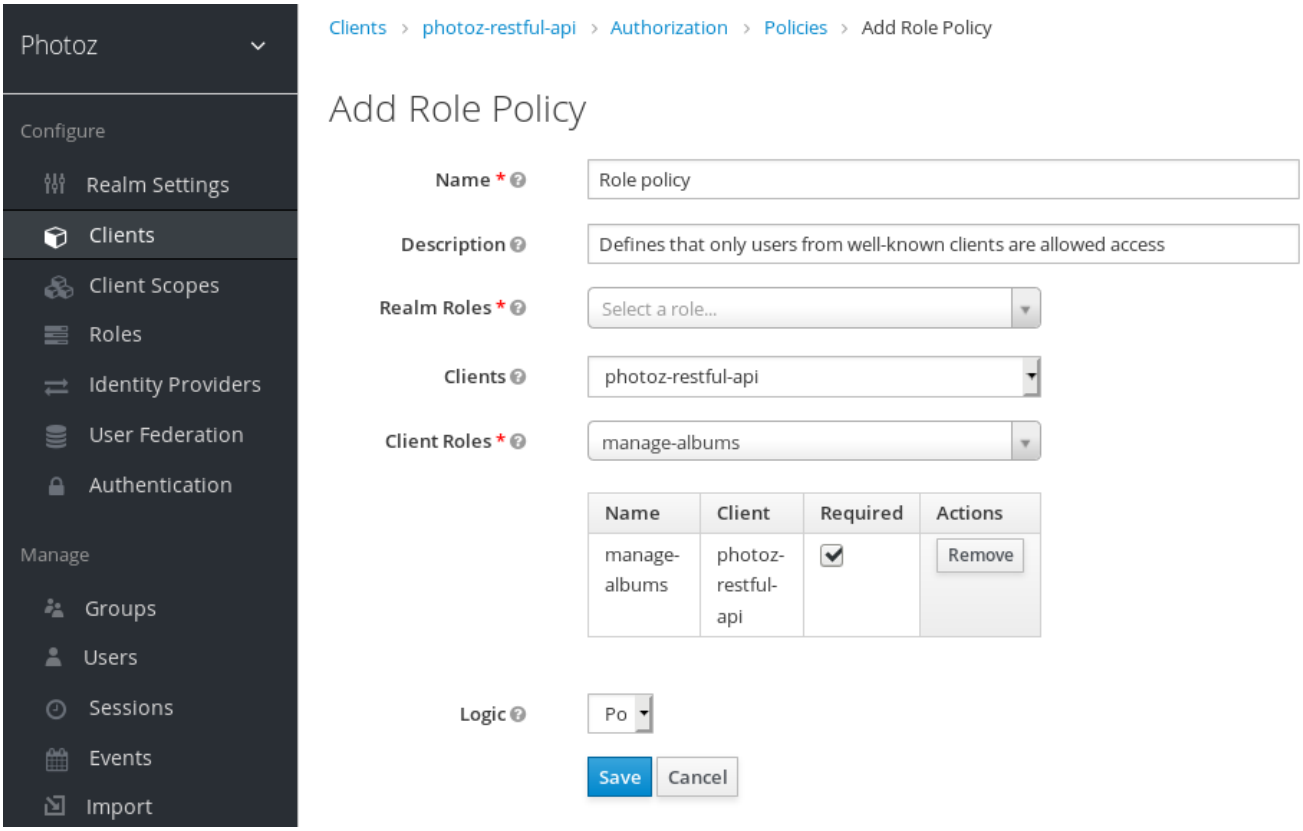
其他资源

- 正和负逻辑

5.2.2. 根据需要定义角色

在创建基于角色的策略时，您可以根据需要指定特定角色 。当这样做时，只有在用户请求访问被授予了**所有 required** 角色时才会授予访问权限。域和客户端角色都可以配置，例如：

所需角色示例



要根据需要指定角色，请根据需要选中您要配置的角色 **所需的** 复选框。

当您的策略定义了多个角色，但只强制要求其中的一部分时，所需的角色就很有用。在这种情况下，您可以组合域和客户端角色，为您的应用程序启用更精细的角色访问控制(RBAC)模型。例如，您可以有特定于客户端的策略，并需要与该客户端关联的特定客户端角色。或者，您可以强制仅被授予特定域角色的访问权限。您还可以在同一策略中组合这两种方法。

5.3. 基于 JAVASCRIPT 的策略

警告

如果您的策略实施基于属性的访问控制(ABAC)，如以下示例所示，请确保用户无法编辑受保护的属性，并且对应的属性是只读的。请参阅 Threat 模型缓解章节中 的详细信息。

您可以使用这类策略使用 JavaScript 为权限定义条件。它是红帽单点登录支持的基于规则的策略类型之一，并提供根据 评估 API 编写任何策略的灵活性。

要创建新的基于 JavaScript 的策略，请在策略列表右上角的 item 列表中选择 **JavaScript**。

注意

默认情况下，JavaScript Policies 无法上传到服务器。您应该希望直接将 JS 策略部署到服务器，如 JavaScript Providers 中所述。

5.3.1. 从部署的 JAR 文件创建 JS 策略

Red Hat Single Sign-On 允许您部署 JAR 文件，以将脚本部署到服务器。请参见 JavaScript Providers 以了解更多详情。

部署好脚本后，您应该能够从可用策略供应商列表中选择部署的脚本。

5.3.2. 例子

5.3.2.1. 从评估上下文中检查属性

以下是基于 JavaScript 的策略的简单示例，它使用基于属性的访问控制(ABAC)根据从执行上下文获取的属性来定义条件：

```
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. 从当前身份检查属性

以下是基于 JavaScript 的策略的简单示例，它使用基于属性的访问控制(ABAC)根据与当前身份关联的属性来定义条件：

```
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();
}
```

其中，这些属性从授权请求中使用的令牌中定义的任何声明映射。

5.3.2.3. 检查授予当前身份的角色

您还可以在策略中使用基于角色的访问控制(RBAC)。在以下示例中，我们检查是否被授予 **keycloak_user 域** 角色：

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

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

或者，您可以检查用户是否被授予 **my-client-role 客户端** 角色，其中 **my-client** 是客户端应用程序的客户端 ID：

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

```
const identity = context.getIdentity();

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

5.3.2.4. 检查授予用户的角色

检查授予用户的域角色：

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

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

或对于授予用户的客户端角色：

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

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

5.3.2.5. 检查授予组的角色

检查授予组的域角色：

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

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

5.3.2.6. 将任意声明推送到资源服务器

将任意声明推送到资源服务器，以提供有关如何强制实施权限的附加信息：

```
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. 检查组成员资格

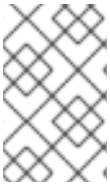
```
const realm = $evaluation.getRealm();
```

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

5.3.2.8. 混合访问控制机制

您还可以使用多种访问控制机制的组合。以下示例演示了如何在同一策略中使用角色(RBAC)和声明/attributes (ABAC)检查。在这种情况下，我们检查是否被授予 **admin** 角色，或者具有来自 **keycloak.org** 域中的电子邮件：

```
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();
}
```



注意

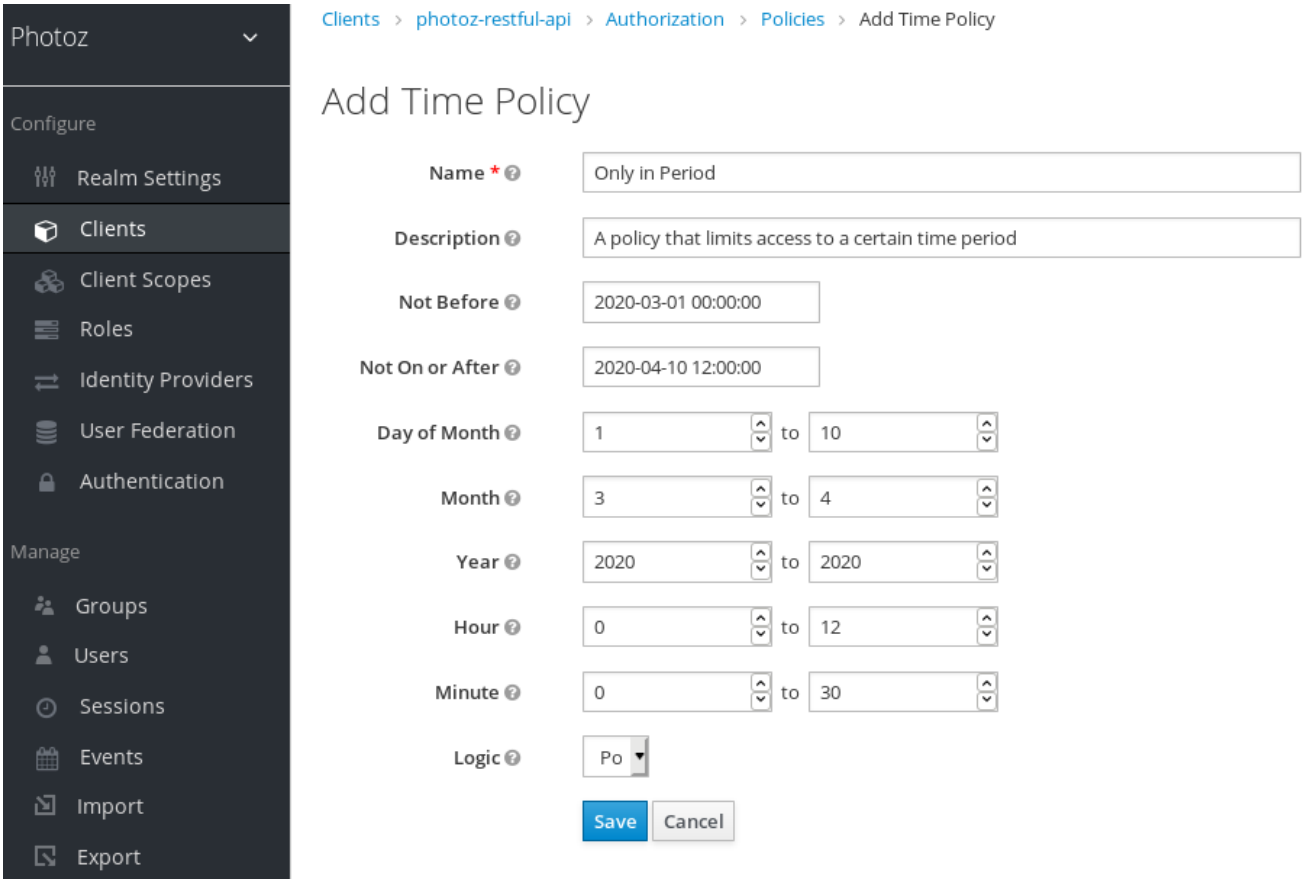
编写自己的规则时，请记住 **$evaluation** 对象是实施 **org.keycloak.authorization.policy.evaluation.Evaluation** 的对象。有关您可以从此接口访问的内容的更多信息，请参阅 评估 API。

5.4. 基于时间的策略

您可以使用这类策略为您的权限定义时间条件。

要创建基于时间的新策略，请在策略列表右上角的项列表中选择 **Time**。

添加时间策略



5.4.1. Configuration

- **Name**
  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**
  包含此策略详情的字符串。

- **不是 Before**
  定义 **不能** 授予访问权限前的时间。只有当前的日期/时间早于或等于这个值时才授予权限。

- **Not On 或 After**
  定义 **不能** 授予访问权限的时间。只有在当前日期/时间早于或等于这个值时才授予权限。

- **几号**
  定义必须授予访问权限的几号。您还可以指定日期范围。在这种情况下，只有在月的当前日期之间或等于指定两个值时才授予权限。

- **月**
  定义必须授予访问权限的月份。您还可以指定范围几个月。在这种情况下，只有在当前月份介于或等于指定两个值时，才会授予权限。

- **年**
  定义必须授予访问权限的年份。您还可以指定一系列年份。在这种情况下，只有在当前年之间或等于指定两个值时才授予权限。

- **hour**
  定义必须授予访问权限的小时。您还可以指定一系列小时。在这种情况下，只有在当前小时之间或等于指定两个值时才授予权限。

- **minute**
  定义必须授予访问权限的分钟。您还可以指定分钟范围。在这种情况下，只有在当前一分钟之间或等于指定两个值时才授予权限。

- **逻辑**
  在评估其他条件后应用此策略的逻辑。

只有满足所有条件时，才会授予访问权限。Red Hat Single Sign-On 将根据每个状况的结果来执行 *AND* 逻辑。

其他资源

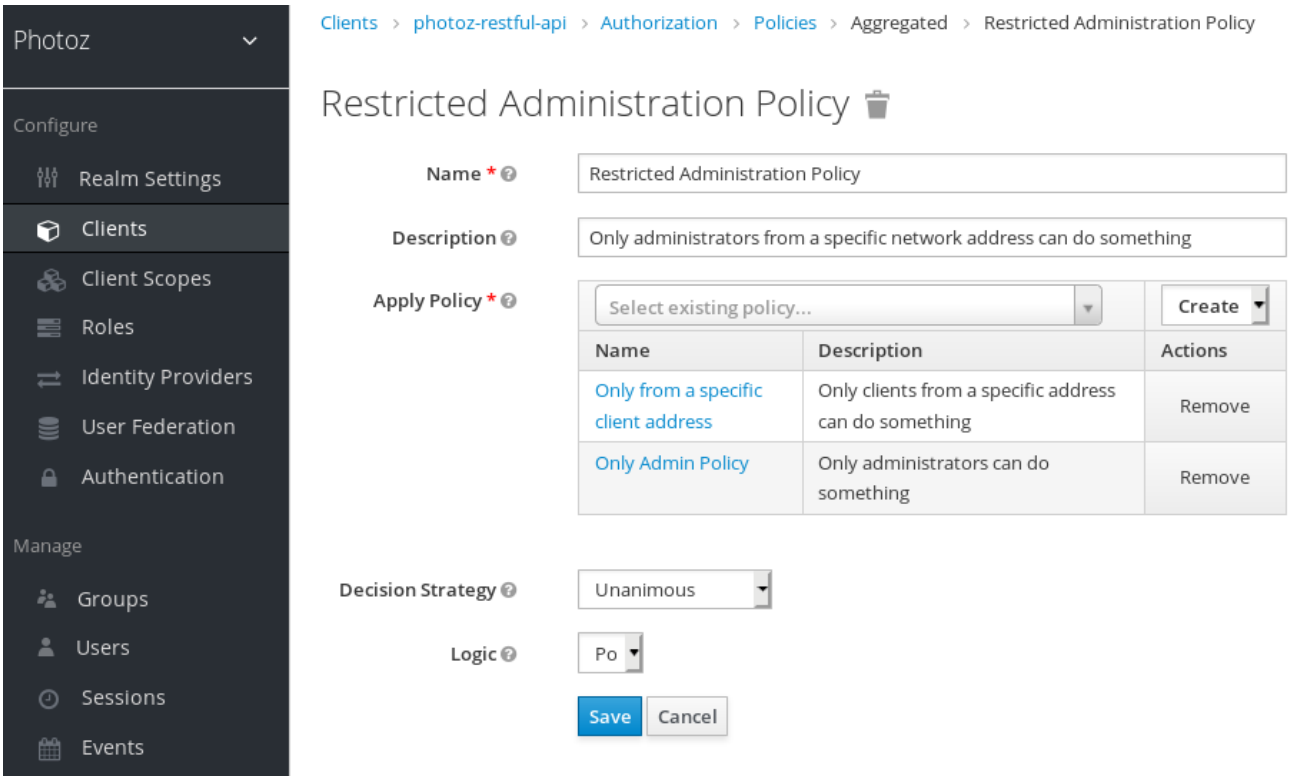
- 正和负逻辑

5.5. 聚合策略

如前文所述，Red Hat Single Sign-On 允许您构建一个策略策略，这种策略概念被称为策略聚合。您可以使用策略聚合来重复使用现有策略来构建更复杂的策略，并使权限与处理授权请求期间评估的策略更为分离。

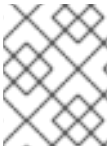
要创建新的聚合策略，请在策略列表右上角的项列表中选择 **Aggregated**。

添加聚合策略



假设您有名为 *Confidential Resource* 的资源，这些资源只能由 *keycloak.org* 域中的用户以及一定范围的 IP 地址进行访问。您可以使用这两个条件创建单个策略。但是，您想要重复使用此策略的域部分，以应用到运行的权限，而不考虑原始网络。

您可以为域和网络条件创建单独的策略，并根据这两个策略的组合创建第三个策略。通过聚合策略，您可以自由组合其他策略，然后将新的聚合策略应用到您想要的任何权限。



注意

在创建聚合策略时，请注意，您不会引入策略之间的循环参考或依赖项。如果检测到循环依赖关系，则无法创建或更新策略。

5.5.1. Configuration

- **Name**
  描述该策略的人类可读和唯一字符串。我们强烈建议您使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们，并知道它们的含义。

- **描述**
  包含此策略的更多详细信息的字符串。

- **应用策略**
  定义与聚合策略关联的一个或多个策略的集合。要关联策略，您可以选择现有策略或通过选择您要创建的策略类型来创建新策略。

- **决策策略**
  此权限的决策策略。

- **逻辑**
  在评估其他条件后应用此策略的逻辑。

其他资源

- 正和负逻辑

5.5.2. 用于聚合策略的决策策略

在创建聚合策略时，您还可以定义根据每个策略的结果来决定最终的决策策略。

- **unanimous**
  如果未提供任何策略，则默认策略。在这种情况下，*所有* 策略都必须评估最终决策的积极决定。

- **Affirmative**
  在这种情况下，*至少有一个* 策略必须评估积极的决定，才能做出最终决定。

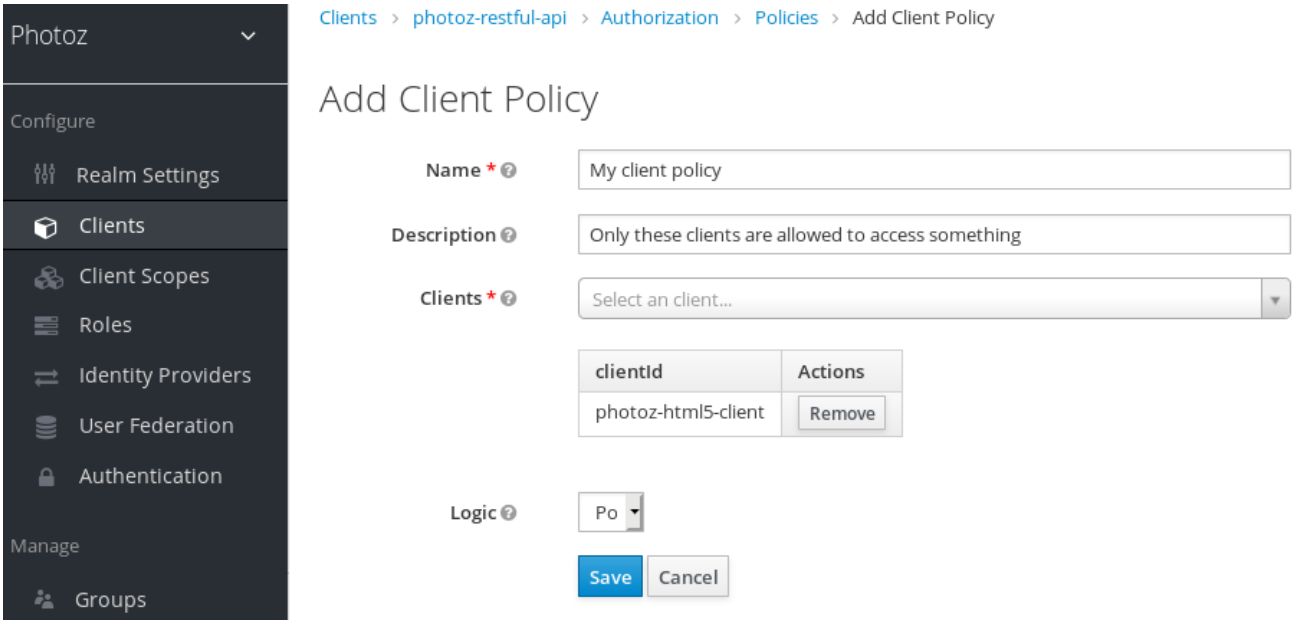
- **consensus**
  在这种情况下，主动决策的数量必须大于负决策的数量。如果正和负决策的数量相同，则最终决定为负数。

5.6. 基于客户端的策略

您可以使用这类策略为您的权限定义一组或多个客户端访问对象的条件。

要创建基于客户端的新策略，请在策略列表右上角的项列表中选择 **Client**。

添加客户端策略



5.6.1. Configuration

- **Name**
  标识策略的用户可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。
- **描述**
  包含此策略详情的字符串。
- **客户端**
  指定此策略可以访问哪些客户端。
- **逻辑**
  在评估其他条件后应用此策略的逻辑。

其他资源

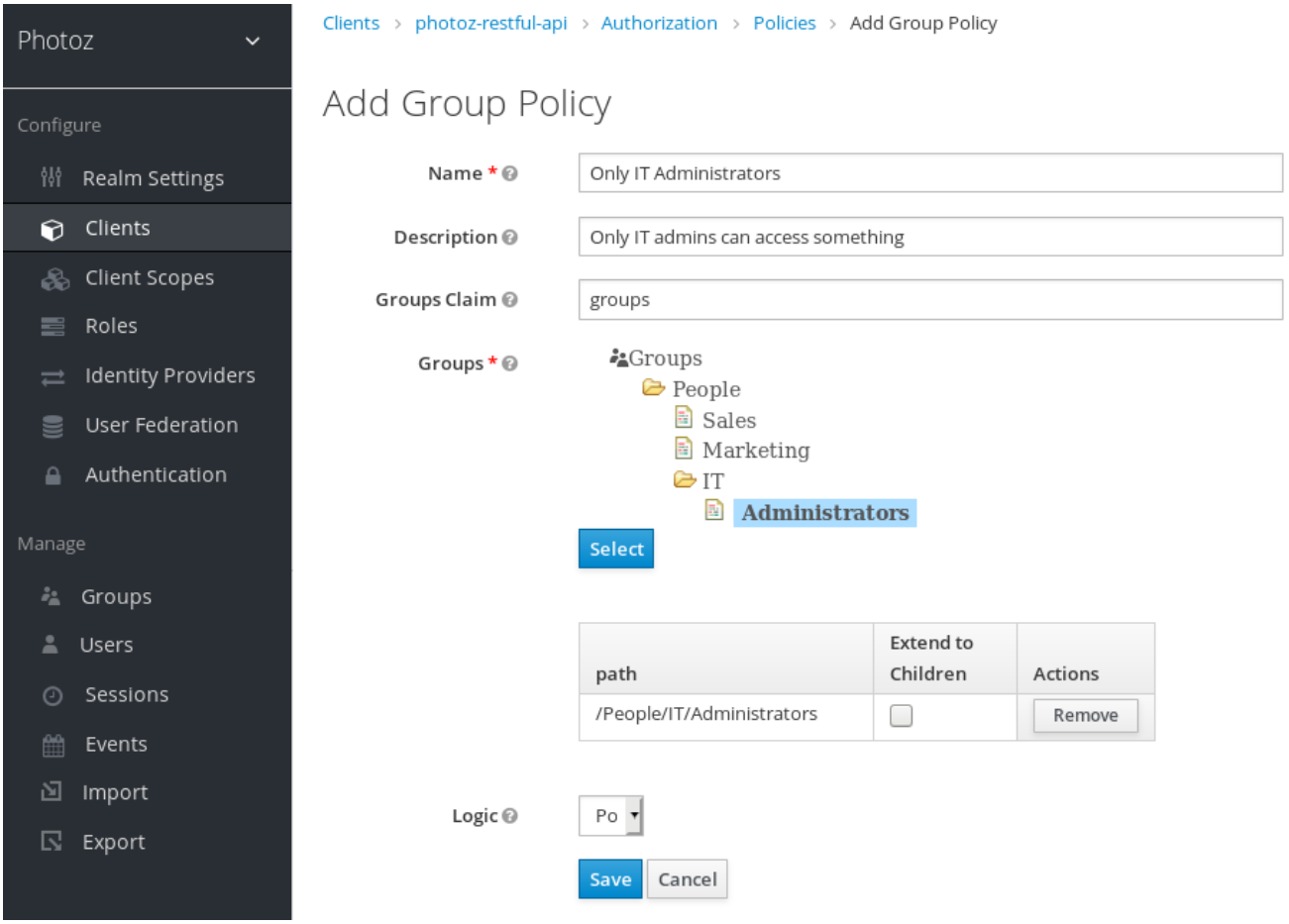
- 正和负逻辑

5.7. 基于组的策略

您可以使用这类策略为您的权限定义一组或多个组（及其层次结构）访问对象的条件。

要创建基于组的新策略，请在策略列表右上角的项列表中选择 **Group**。

组策略



5.7.1. Configuration

- **Name**
  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**
  包含此策略详情的字符串。

- **组声明**
  指定令牌中的声明名称（包含组名称和/或路径）。通常，授权请求基于之前发给给某些用户的客户端的 ID 令牌或访问令牌进行处理。如果定义，令牌必须包含从哪里获取此政策的声明，以便用户所属的组。如果未定义，则从您的域配置获取用户的组。

- **组**
  允许您选择在评估权限时此策略应强制实施的组。添加组后，您可以通过标记复选框 **扩展至** childrenren 来扩展组子的访问权限。如果保留未标记，则访问限制仅适用于所选组。

- **逻辑**
  在评估其他条件后应用此策略的逻辑。

其他资源

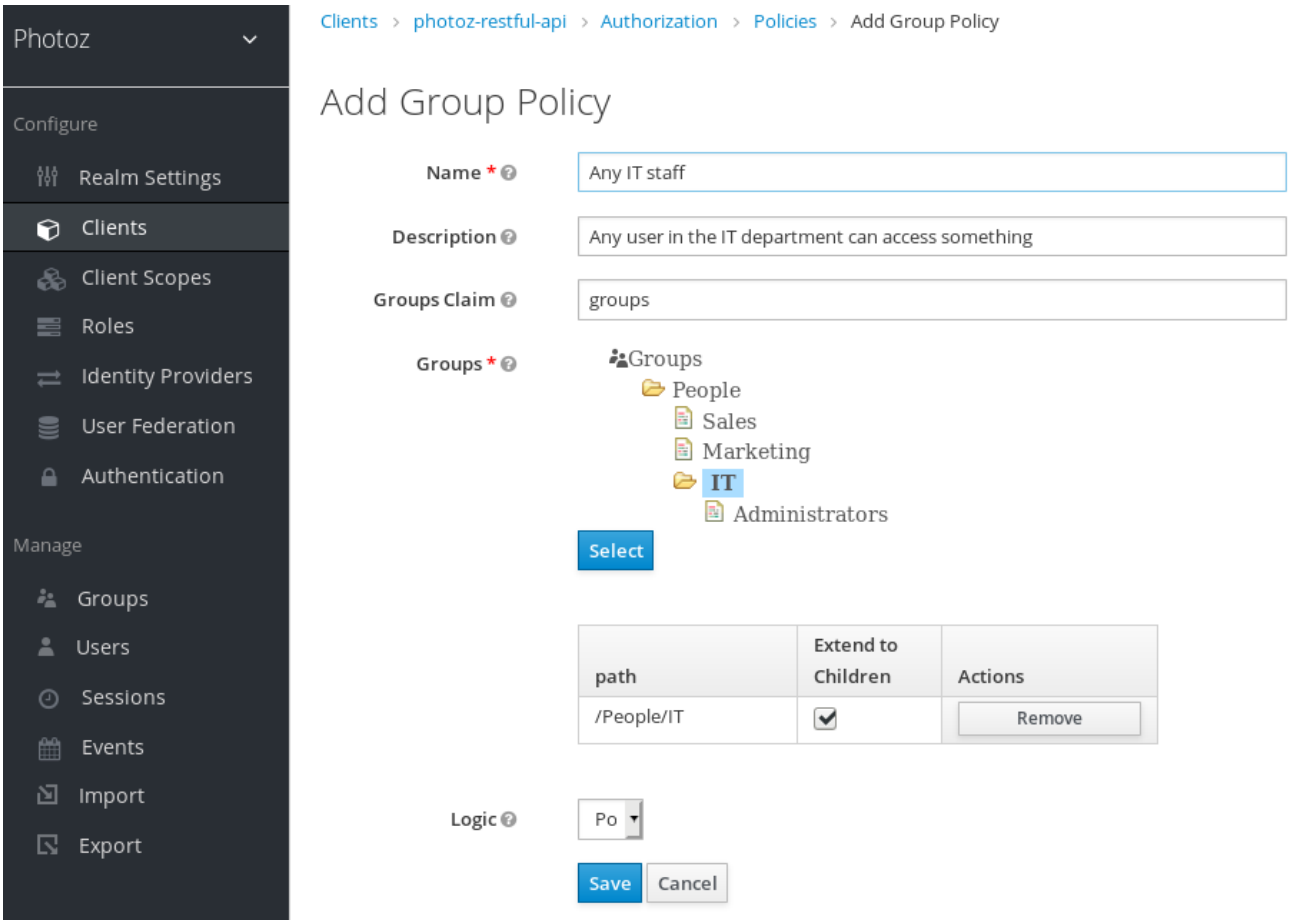
- 正和负逻辑

5.7.2. 扩展对子组的访问

默认情况下，当您将组添加到此策略时，访问限制将仅应用到所选组的成员。

在某些情况下，可能不需要允许访问组本身，而是允许访问层次结构中的任何子组。对于添加的任何组，您可以标记一个复选框 **可扩展到** 子进程，从而扩展对子组的访问权限。

扩展对子组的访问



在上面的示例中，策略是为 **IT** 或其任何成员的任何用户授予访问权限。

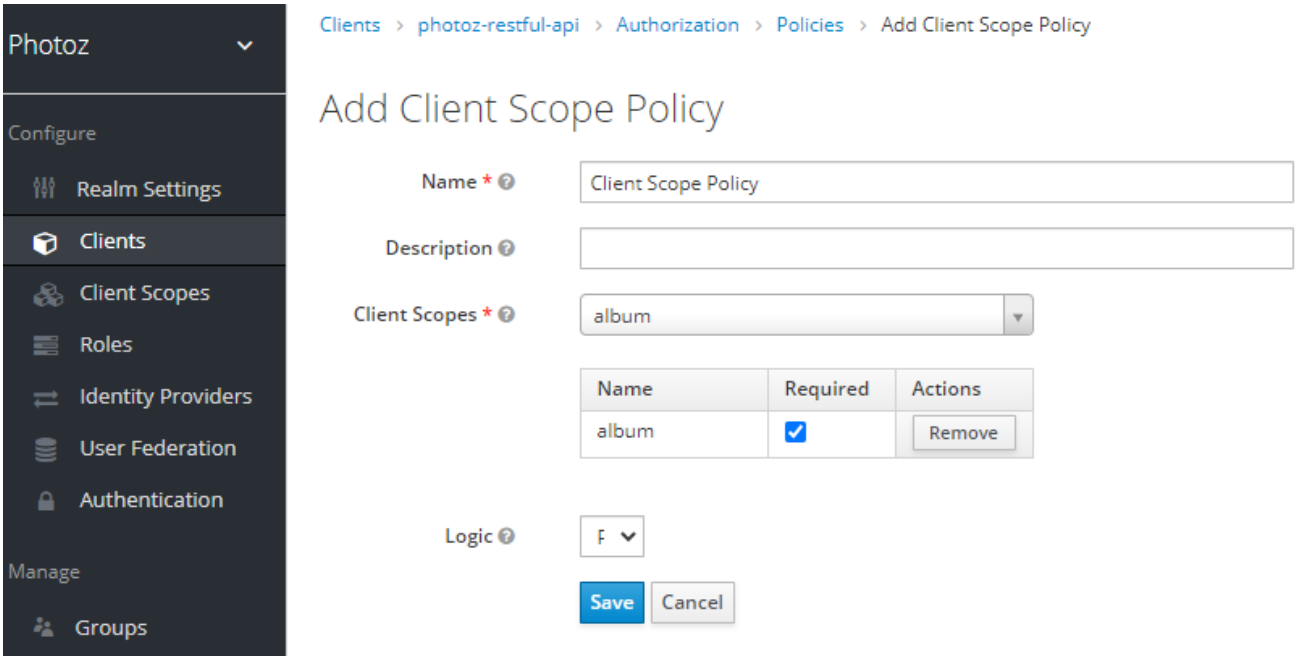
5.8. 基于客户端范围的策略

您可以使用这类策略为您的权限定义一组或多个客户端范围访问对象的条件。

默认情况下，添加到此策略中的客户端范围不根据需要指定，如果请求访问的客户端已被授予其中任何客户端范围，策略将授予访问权限。但是，如果要强制执行特定的客户端范围，您可以指定一个特定的客户端范围为 required。

要创建新的基于客户端范围的策略，请在策略列表右上角的项列表中选择 **Client Scope**。

添加客户端范围策略



5.8.1. Configuration

- **Name**
  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。
- **描述**
  包含此策略详情的字符串。
- **客户端范围**
  指定此策略允许哪些客户端范围。
- **逻辑**
  在评估其他条件后应用此策略的逻辑。

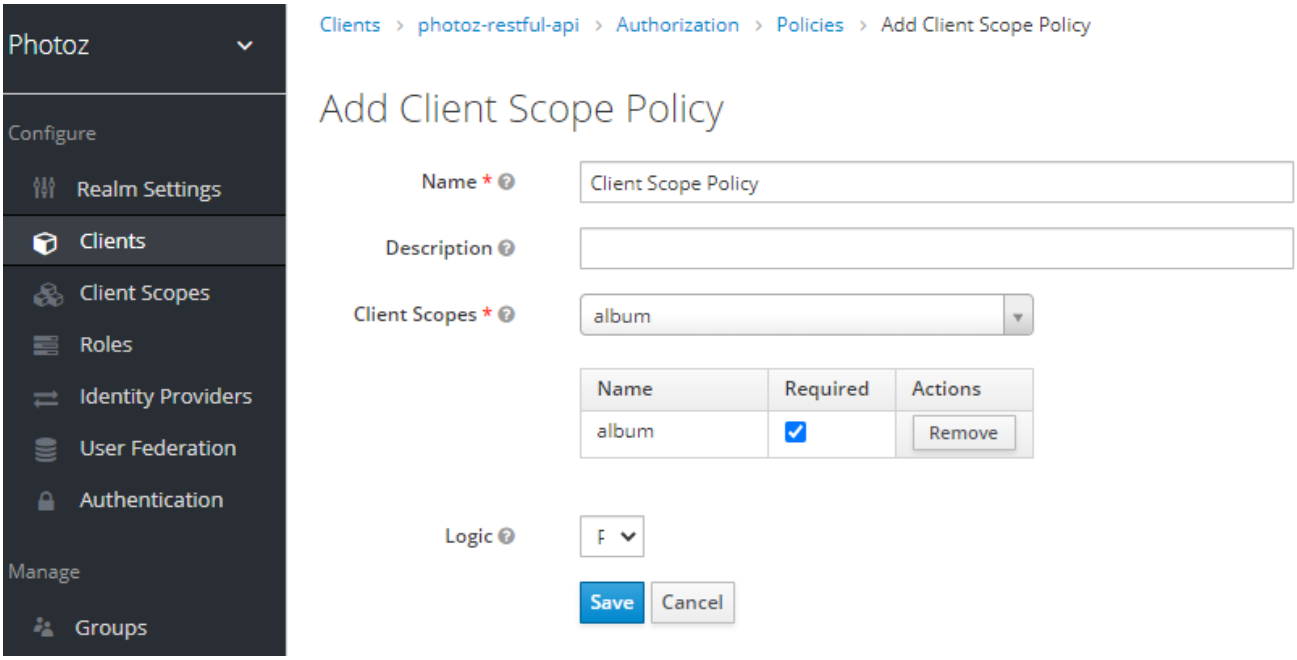
其他资源

- 正和负逻辑

5.8.2. 根据需要定义客户端范围

在创建基于客户端范围的策略时，您可以将特定的客户端范围指定为 **Required**。当这样做时，只有在客户端请求访问被授予了**所有 required** 客户端范围时才会授予访问权限。

所需的客户端范围示例



要根据需要指定客户端范围，请根据需要选中您要配置的客户端范围 **所需的** 复选框。

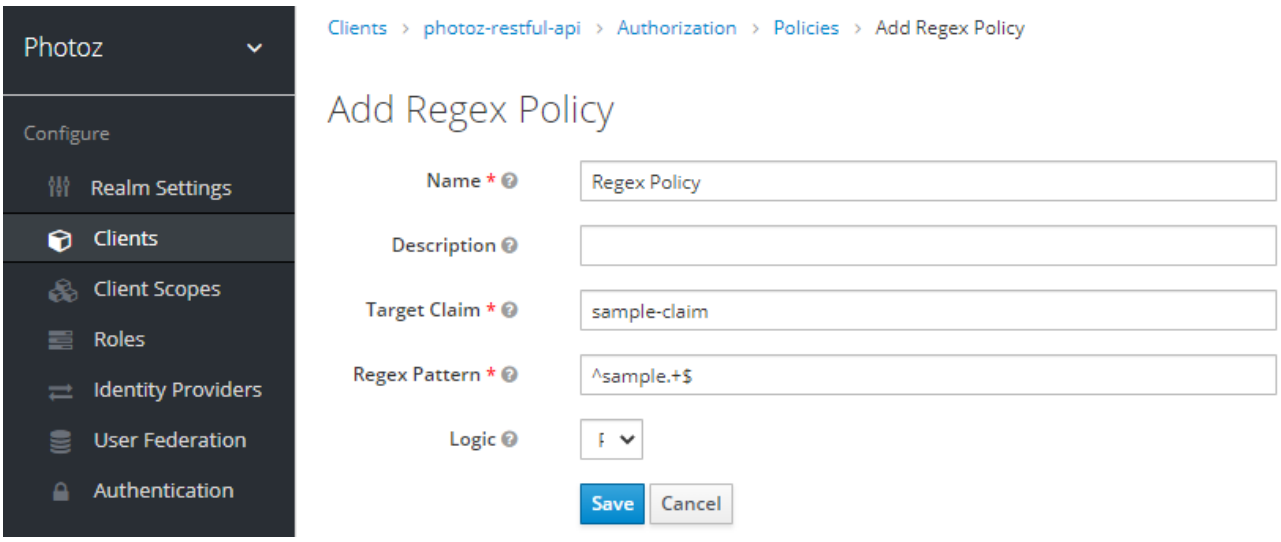
当您的策略定义了多个客户端范围时，所需的客户端范围会很有用，但只有它们的子集才是必需的。

5.9. 基于 REGEX 的策略

您可以使用这种类型的策略为您的权限定义 regex 条件。

要创建基于 regex 的新策略，请在策略列表右上角的项列表中选择 **Regex**。

添加 Regex 策略



5.9.1. Configuration

- **Name**
  描述该策略的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**
  包含此策略详情的字符串。

- **目标申索**
  指定令牌中的目标声明的名称。

- **正则表达式模式**
  指定 regex 模式。

- **逻辑**
  评估其他条件后要应用此策略的 Logic。

5.10. 正和负逻辑

策略可以使用正或负逻辑配置。简而言之，您可以使用这个选项来定义策略结果是否应保留不变。

例如，假设您要创建一个策略，**其中只** 只授予用户授予特定角色的用户。在这种情况下，您可以使用该角色创建基于角色的策略，并将其 **Logic** 字段设置为 **Negative**。如果您 **保留** Positive （这是默认行为），策略结果将保留原样。

5.11. 策略评估 API

使用 JavaScript 编写基于规则的策略时，红帽单点登录提供了评估 API，提供有用的信息来帮助确定是否应授予权限。

此 API 由几个接口组成，它们可让您访问信息，例如

- 要评估的权限，代表所请求的资源和范围。
- 与请求的资源关联的属性
- 运行时环境以及与执行上下文关联的任何其他属性
- 有关组成员和角色等用户的信息

主接口是 **org.keycloak.authorization.policy.evaluation.Evaluation**，它定义了以下合同：

```
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();
}
```

在处理授权请求时，Red Hat Single Sign-On 会创建 **评估** 实例，然后再评估任何策略。然后，此实例将传递到每个策略，以确定访问权限是否为 **GRANT** 还是 **DENY**。

通过调用 **Evaluation** 实例上的 **grant （）** 或 **deny （）** 方法来确定策略。默认情况下，**评估** 实例的状态被拒绝，这意味着您的策略必须显式调用 **grant （）** 方法，以指示应被授予权限的策略评估引擎。

其他资源

- JavaDocs 文档.

5.11.1. 评估上下文

评估上下文在评估期间为策略提供了有用的信息。

```
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();
}
```

在这个界面中，策略可以获取：

- 经过身份验证的 **身份**
- 有关执行上下文和运行时环境的信息

身份 根据与授权请求一起发送的 OAuth2 访问令牌构建，并且此构造可以访问从原始令牌中提取的所有声明。例如，如果您使用 *协议映射程序* 在 OAuth2 访问令牌中包含自定义声明，您也可以从策略访问此声明并使用它来构建您的条件。

EvaluationContext 还可让您访问与执行环境和运行时环境相关的属性。现在，只有几个内置属性。

表 5.1. 执行和运行时属性

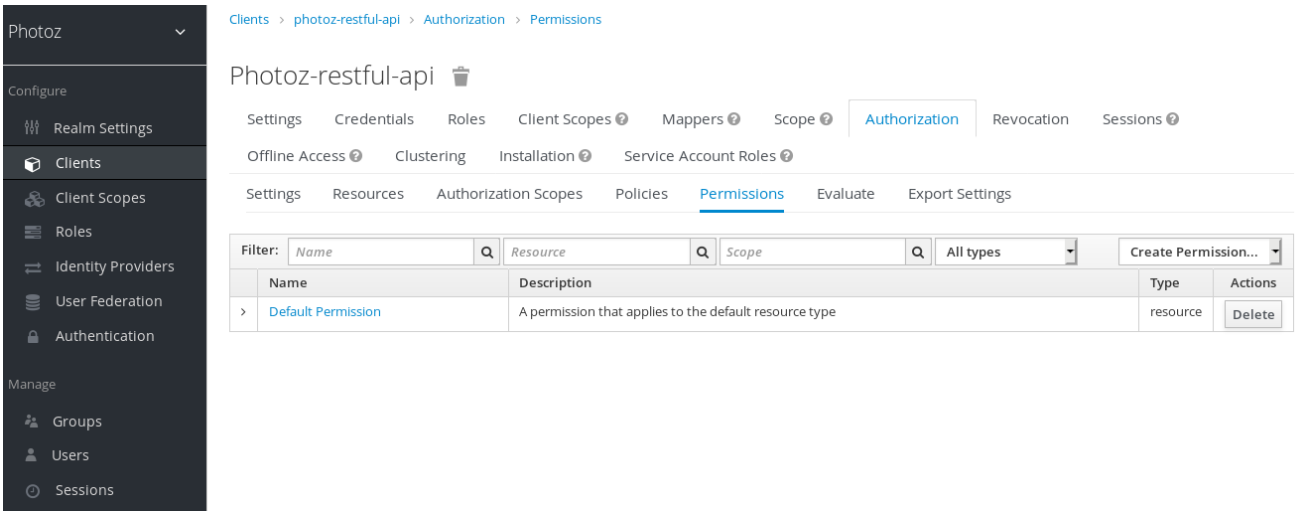
Name	描述	类型
kc.time.date_time	当前日期和时间	字符串.格式 **MM/dd/yyy hh:mm:ss**
kc.client.network.ip_address	客户端的 IPv4 地址	字符串
kc.client.network.host	客户端的主机名	字符串
kc.client.id	客户端 ID	字符串
kc.client.user_agent	'User-Agent' HTTP 标头的值	String[]
kc.realm.name	域名称	字符串

第 6 章 管理权限

权限将对象与受保护的对象相关联，以及必须接受相应策略来决定是否应授予访问权限。

创建您要保护的资源以及用于保护这些资源的策略后，您可以开始管理权限。要管理权限，在编辑资源服务器时点击 **Permissions** 选项卡。

权限



可以创建权限来保护主要类型的对象：

- **Resources**
- **范围**

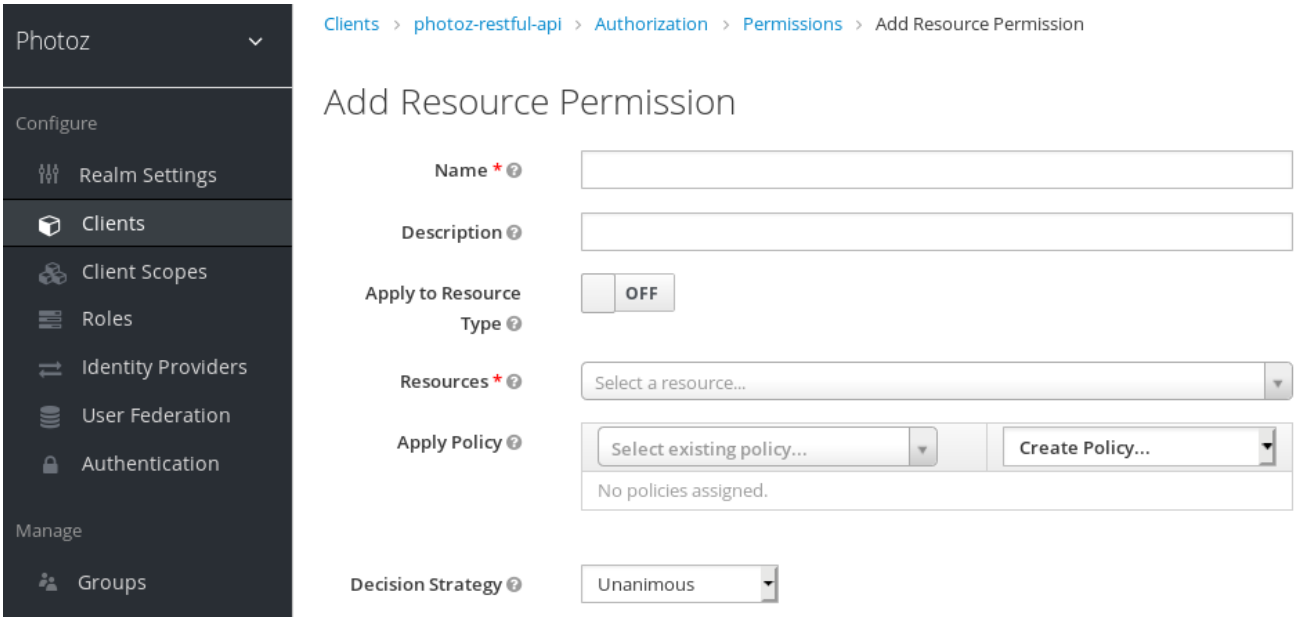
要创建权限，请从权限列表右上角的项目列表中选择您要创建的权限类型。以下小节更详细地描述了这两类对象。

6.1. 创建基于资源的权限

基于资源的权限定义了一组一个或多个资源，以防止使用一组或多个授权策略进行保护。

要创建基于资源的新权限，请在权限列表右上角的项列表中选择 **基于 Resource**。

添加资源权限



6.1.1. Configuration

- **Name**
  描述该权限的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**
  包含此权限详情的字符串。

- **应用 To 资源类型**
  指定权限是否应用到具有指定类型的所有资源。选择此字段时，会提示您输入要保护的资源类型。

  - 资源类型
    定义要保护的资源类型。定义后，此权限将针对与该类型匹配的所有资源进行评估。

- **Resources**
  定义一个或多个用于保护的资源的集合。

- **应用策略**
  定义与权限关联的一个或多个策略的集合。要关联策略，您可以选择现有策略或通过选择您要创建的策略类型来创建新策略。

- **决策策略**
  此权限的决策策略。

6.1.2. 输入的资源权限

资源权限也可用于定义要应用到给定类型的所有资源的策略。当您有共享共同访问要求和限制的资源时，这种基于资源的权限非常有用。

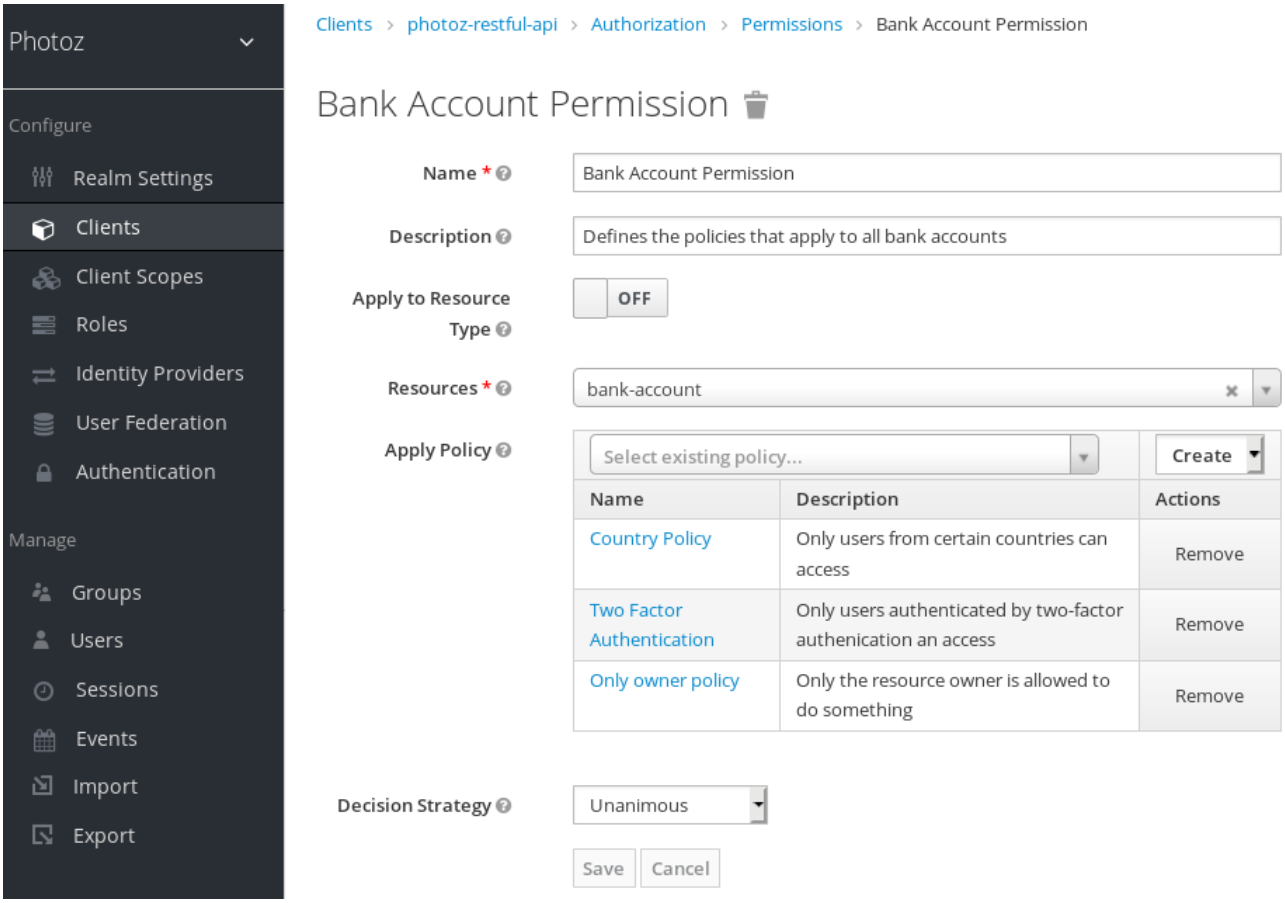
通常，应用中的资源可以根据它们封装或提供的功能来分类（或键入）。例如，一个金融应用程序可以管理不同的银行客户，各家都属于特定客户。虽然它们是不同的银行帐户，但它们共享银行组织在全球定义的常见安全要求和限制。使用键入的资源权限，您可以定义适用于所有银行帐户的通用策略，例如：

- 只有所有者可以管理他的帐户

- 仅允许来自所有者国家和/或地区的访问权限
- 强制执行特定的验证方法

要创建类型的资源权限，请在创建新基于资源的权限时点击 Apply to Resource Type。通过将 **Apply to Resource Type** 设置为 **On**，您可以指定要保护的类型以及要应用到指定类型的所有资源的策略。

输入的资源权限示例

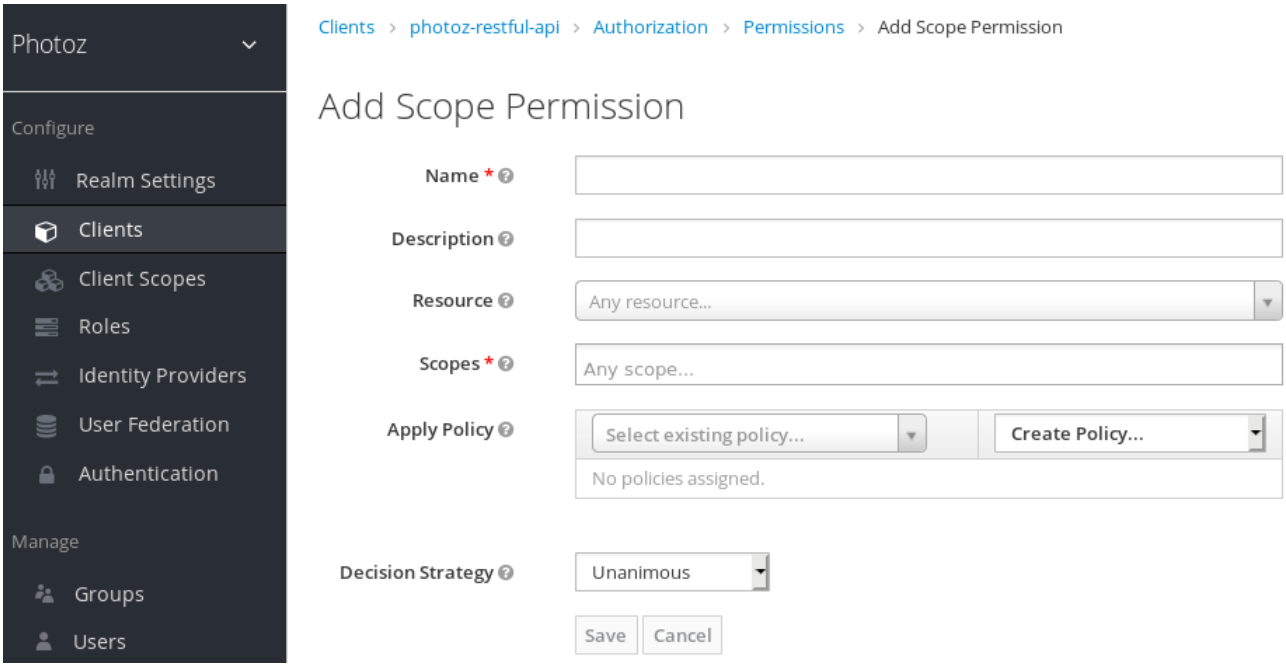


6.2. 创建基于范围的权限

基于范围的权限定义了一组或多个范围来保护使用一个或多个授权策略。与基于资源的权限不同，您可以使用此权限类型只为资源创建权限，也针对与资源关联的范围，在定义资源的权限以及可在其上执行的操作时提供更多粒度。

要创建基于范围的新权限，请在权限列表右上角的项列表中选择 **基于 Scope**。

添加范围权限



6.2.1. Configuration

- **Name**
  描述该权限的人类可读和唯一字符串。最佳实践是使用与您的业务和安全要求密切相关的名称，以便您可以更轻松地识别它们。

- **描述**
  包含此权限详情的字符串。

- **资源**
  将范围限制为与所选资源关联的范围。如果没有选择，则所有范围都可用。

- **范围**
  定义一个或多个要保护的范围。

- **应用策略**
  定义与权限关联的一个或多个策略的集合。要关联策略，您可以选择现有策略或通过选择您要创建的策略类型来创建新策略。

- **决策策略**
  此权限的决策策略。

6.3. 策略决策策略

将策略与权限关联时，您还可以定义决策策略来指定如何评估相关策略以确定访问权限的结果。

- **unanimous**
  如果未提供任何策略，则默认策略。在这种情况下，*所有* 策略都必须评估最终决策的积极决定。

- **Affirmative**
  在这种情况下，*至少有一个* 策略必须评估最终决定也是正面的积极决定。

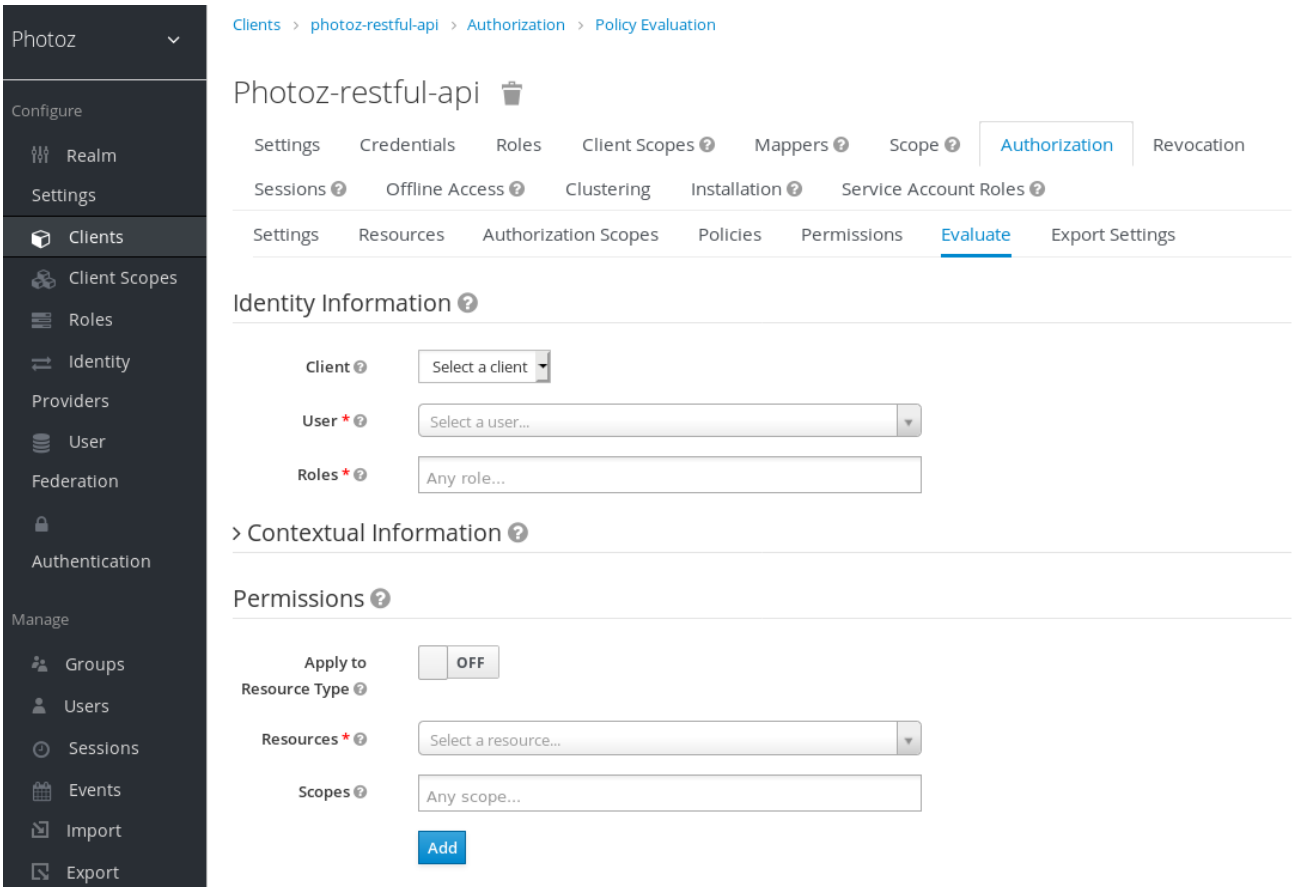
- **consensus**
  在这种情况下，主动决策的数量必须大于负决策的数量。如果正和负决策的数量相等，则最终决定为负数。

第 7 章 评估和测试策略

在设计策略时，您可以模拟授权请求来测试如何评估您的策略。

在编辑资源服务器时，您可以点击 **Evaluate** 选项卡来访问 Policy Evaluation Tool。您可以指定不同的输入来模拟实际授权请求并测试策略的影响。

策略评估工具



7.1. 提供身份信息

Identity Information 过滤器可用于指定请求权限的用户。

7.2. 提供上下文信息

上下文信息 过滤器可用于定义评估上下文的额外属性，以便策略能够获取这些相同的属性。

7.3. 提供权限

Permissions 过滤器可用于构建授权请求。您可以为一个或多个资源和范围请求权限。如果要根据所有受保护资源和范围模拟授权请求，请单击 **Add**，而不指定任何 **Resources** 或 **Scopes**。

当您指定了所需的值时，点 **Evaluate**。

第 8 章 授权服务

Red Hat Single Sign-On Authorization Services 基于已知的标准构建，如 OAuth2 和用户管理的访问规格。

OAuth2 客户端（如前端应用）可以使用令牌端点从服务器获取访问令牌，并将这些令牌用于访问由资源服务器（如后端服务）保护的资源。同样，Red Hat Single Sign-On Authorization Services 为 OAuth2 提供扩展，允许根据处理与请求资源关联的所有策略来发布访问令牌。这意味着资源服务器可以根据服务器获得的权限强制访问其受保护的资源，并由访问令牌持有。在 Red Hat Single Sign-On Authorization Services 中，具有权限的访问令牌被称为 Requesting unsupported Token 或 RPT for short。

除了 RPTs 的经验，Red Hat Single Sign-On Authorization Services 还提供一组 RESTful 端点，允许资源服务器管理其受保护的资源、范围、权限和策略，帮助开发人员将这些功能扩展到其应用程序中，以支持精细的授权。

8.1. 发现授权服务端点和元数据

Red Hat Single Sign-On 提供发现文档，客户端可从中获取所有必要信息，以便与红帽单点登录授权服务（包括端点位置和功能）进行交互。

发现文档可从以下位置获取：

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

其中 **${host}:${port}** 是运行 Red Hat Single Sign-On 的主机名（或 IP 地址）和端口，而 **${realm}** 是 Red Hat Single Sign-On 中域的名称。

因此，您应该获得如下响应：

```
{

    // some claims are expected here

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

每个端点都公开一组特定的功能：

- **token_endpoint**
  支持 **urn:ietf:params:oauth:grant-type:uma-ticket** 授权类型的 OAuth2 兼容令牌端点。通过此端点客户端可以发送授权请求，并通过 Red Hat Single Sign-On 授予的所有权限获取 RPT。

- **token_introspection_endpoint**
  OAuth2-compliant Token Introspection Endpoint，客户端可以用来查询服务器来确定 RPT 的活动状态，并确定与令牌关联的任何其他信息，如 Red Hat Single Sign-On 所授予的权限。

- resource_registration_endpoint
  与 UMA 兼容的资源注册端点，资源服务器可用于管理受保护的资源和范围。此端点提供 Red Hat Single Sign-On 中的操作创建、读取、更新和删除资源和范围。

- permission_endpoint
  与 UMA 兼容的 Permission Endpoint，资源服务器可用于管理权限票据。此端点在 Red Hat Single Sign-On 中提供操作创建、读取、更新和删除权限问题单。

8.2. 获取权限

要从 Red Hat Single Sign-On 获取权限，您需要将授权请求发送到令牌端点。因此，Red Hat Single Sign-On 将评估与请求资源相关的所有策略和范围，并发出包含服务器授予的所有权限的 RPT。

客户端可使用以下参数将授权请求发送到令牌端点：

- grant_type
  此参数是必需的。必须是 **urn:ietf:params:oauth:grant-type:uma-ticket**。

- ticket
  这个参数是**可选的**。客户端收到的最新权限票据，作为 UMA 授权过程的一部分。

- claim_token
  这个参数是**可选的**。代表在评估请求资源的权限时，服务器应考虑的额外声明的字符串。这个参数允许客户端将声明推送到 Red Hat Single Sign-On。有关所有支持的令牌格式的详情，请参阅 **claim_token_format** 参数。

- claim_token_format
  这个参数是**可选的**。指示 **claim_token** 参数中指定的令牌格式的字符串。Red Hat Single Sign-On 支持两种令牌格式：ur **n:ietf:params:oauth:token-type:jwt** 和 **https://openid.net/specs/openid-connect-core-1_0.html#IDToken**。**urn:ietf:params:oauth:token-type:jwt** 格式表示 **claim_token** 参数引用访问令牌。**https://openid.net/specs/openid-connect-core-1_0.html#IDToken** 表示 **claim_token** 参数引用 OpenID Connect ID Token。

- rpt
  这个参数是**可选的**。之前发布的 RPT 还应在一个新的中评估和添加权限。这个参数允许包含 RPT 的客户端执行增量授权，其中根据需要添加权限。

- **权限**
  这个参数是**可选的**。代表一个或多个资源和范围的一组字符串，客户端正在寻求访问。这个参数可以多次定义，以便请求多个资源和范围的权限。这个参数是对 **urn:ietf:params:oauth:grant-type:uma-ticket** 授权类型的扩展，以允许客户端在没有权限票据的情况下发送授权请求。字符串的格式必须是： **RESOURCE_ID#SCOPE_ID**。例如： **Resource A#Scope A**,**Resource A#Scope A, Scope B, Scope C**,**Resource A**,**#Scope A**.

- audience
  这个参数是**可选的**。客户端查找访问的资源服务器的客户端标识符。如果定义了 **权限** 参数，则此参数是必须的。它充当 Red Hat Single Sign-On 的提示，以指示应评估其权限的上下文。

- response_include_resource_name
  这个参数是**可选的**。代表服务器是否应该包含在 RPT 权限中的布尔值。如果为 false，则仅包括资源标识符。

- response_permissions_limit
  这个参数是**可选的**。为 RPT 的权限数量定义限值的整数 N。与 **rpt** 参数一同使用时，只有最后 N 请求的权限才会保存在 RPT 中。

- **submit_request**
  这个参数是**可选的**。布尔值指示服务器是否应创建对权限票据引用的资源和范围的权限。这个参数仅在将 **ticket** 参数用作 UMA 授权进程的一部分时才有效。

- **response_mode**
  这个参数是**可选的**。一个字符串值，代表服务器应如何响应授权请求。当您主要对总体决策或服务器授予的权限有关时，这个参数特别有用，而不是标准的 OAuth2 响应。可能的值有：

  - **decision**
    表示来自服务器的响应应该只通过返回带有以下格式的 JSON 来代表总体决策：

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

    如果授权请求没有映射到任何权限，则返回 **403** HTTP 状态代码。

  - **权限**
    表示来自服务器的响应应该包含服务器授予的任何权限，方法是返回带有以下格式的 JSON：

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

        ...
    ]
    ```

    如果授权请求没有映射到任何权限，则返回 **403** HTTP 状态代码。

当客户端想要访问由资源服务器保护的两个资源时，授权请求的示例。

```
curl -X POST \
  http://${host}:${port}/auth/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"
```

当客户端想要访问由资源服务器保护的资源和范围时，授权请求的示例。

```
curl -X POST \
  http://${host}:${port}/auth/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}"
```

当客户端在作为授权过程中从资源服务器接收权限票据后，查询访问 UMA 保护的资源的示例：

```
curl -X POST \
  http://${host}:${port}/auth/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}
```

如果 Red Hat Single Sign-On 评估过程会产生权限，则发出 RPT 与其关联了权限：

Red Hat Single Sign-On 使用 RPT 响应客户端

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

服务器的响应与使用某些其他授权类型时来自令牌端点的任何其他响应一样。RPT 可以从 **access_token** 响应参数获取。如果客户端没有获得授权，Red Hat Single Sign-On 会使用 **403** HTTP 状态代码进行响应：

Red Hat Single Sign-On 拒绝授权请求

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

8.2.1. 客户端验证方法

客户端需要向令牌端点进行身份验证，以获取 RPT。当使用 **urn:ietf:params:oauth:grant-type:uma-ticket** grant 类型时，客户端可以使用任何这些验证方法：

- **bearer 令牌**
  客户端应该在 HTTP 身份验证标头中将访问令牌作为 Bearer 凭据发送到令牌端点。

  示例：使用访问令牌向令牌端点进行身份验证的授权请求

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

  当客户端代表用户时，此方法特别有用。在这种情况下，bearer 令牌是之前由 Red Hat Single Sign-On 颁发给一些代表用户（或代表自己）发布的访问令牌。将评估权限，以考虑访问令牌所代表的访问上下文。例如，如果向客户端 A 签发了访问令牌，代表 User A，则根据用户 A 可以访问的资源和范围授予权限。

- **客户端凭证**
  客户端可以使用 Red Hat Single Sign-On 支持的任何客户端身份验证方法。例如，client_id/client_secret 或 JWT。

  示例：使用客户端 ID 和客户端 secret 进行令牌端点的授权请求

```
curl -X POST \
  http://${host}:${port}/auth/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. 推送声明

从服务器获取权限时，您可以推送任意声明，以便在评估权限时将这些声明提供给您的策略。

如果您在不使用权限票据(UMA 流) **的情况下** 从服务器获取权限，您可以按照以下方法向令牌端点发送授权请求：

```
curl -X POST \
  http://${host}:${port}/auth/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 参数需要一个类似于以下示例的 BASE64 编码 JSON：

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

格式需要一个或多个声明，其中每个声明的值都必须是一个字符串数组。

8.2.2.1. 使用 UMA 推送声明

有关如何使用 UMA 和权限票据时如何推送声明的更多详细信息，请参阅 权限 API

8.3. 用户管理的访问

Red Hat Single Sign-On Authorization Services 基于用户管理的访问或 UMA （简称为 ）UMA 是以以下方式增强了 OAuth2 功能的规格：

- **隐私性**
  现在，用户隐私已成为巨大的问题，因为更多的数据和设备可用并连接到云。借助 UMA 和红帽单点登录，资源服务器可以增强其功能，以改进根据用户定义的政策保护其资源保护。

- **第三方授权**
  资源所有者（例如：常规最终用户）可以管理对其资源的访问，授权其他方（例如：常规最终用户）以访问这些资源。这与 OAuth2 不同，如果代表用户为客户端应用程序授予权限，则允许 UMA 资源限制以完全异步方式访问其他用户。

- **资源共享**
  资源所有者可以管理其资源的权限，并决定谁可以访问特定资源以及如何。红帽单点登录可充当共享管理服务，资源所有者可从其中管理其资源。

Red Hat Single Sign-On 是一个兼容 UMA 2.0 的授权服务器，提供大多数 UMA 功能。

例如，请考虑使用互联网银行服务（资源服务器）来管理她银行帐户（资源服务器）的用户 Alice （资源所有者）。Alice Alice 决定开放她的银行账户给 Bob （请求方）是一个会计师。然而，Bob 应该只能访问查看（范围） Alice 的帐户。

作为资源服务器，互联网银行服务必须能够保护 Alice 的银行帐户。为此，它依赖于 Red Hat Single Sign-On Resource Registration Endpoint 在代表 Alice 银行帐户的服务器中创建资源。

目前，如果 Bob 尝试访问 Alice 银行帐户，则访问将被拒绝。互联网银行服务为银行帐户定义了几个默认策略。其中之一是，只有所有者（本例中为 Alice）被允许访问她银行帐户。

但是，针对 Alice 隐私权的 Internet 银行服务也允许她更改银行账户的特定政策。她可以改变的其中一个策略就是定义允许哪些人查看其银行帐户。为此，互联网银行服务依赖 Red Hat Single Sign-On 为 Alice 提供一个空间，以便她可以选择他们可访问的个人和操作（或数据）。随时，Alice 可以撤销访问权限或向 Bob 授予额外的权限。

8.3.1. 授权过程

在 UMA 中，当客户端尝试访问 UMA 保护的资源服务器时，授权进程会启动。

UMA 保护的资源服务器在请求中需要一个 bearer 令牌，其中令牌是 RPT。当客户端在没有 RPT 的情况下在资源服务器中请求资源时：

客户端在没有发送 RPT 的情况下请求受保护的资源

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

资源服务器使用权限 **票据** 和 **as_uri** 参数向客户端发送响应，使用 Red Hat Single Sign-On 服务器的位置将响应发送到该请求以获取 RPT。

资源服务器以权限票据响应

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

权限票据是由 Red Hat Single Sign-On Permission API 发布的一种特殊令牌类型。它们表示所请求的权限（例如：资源和范围）以及与请求关联的任何其他信息。只有资源服务器被允许创建这些令牌。

现在客户端有权限票据以及 Red Hat Single Sign-On 服务器的位置，客户端可以使用发现文档来获取令牌端点的位置并发送授权请求。

客户端向令牌端点发送授权请求以获取 RPT

```
curl -X POST \
  http://${host}:${port}/auth/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}
```

如果 Red Hat Single Sign-On 评估过程会产生权限，则发出 RPT 与其关联了权限：

Red Hat Single Sign-On 使用 RPT 响应客户端

-

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

服务器的响应与使用某些其他授权类型时来自令牌端点的任何其他响应一样。RPT 可以从 **access_token** 响应参数获取。如果客户端没有授权具有 Red Hat Single Sign-On 使用 **403** HTTP 状态代码的权限：

Red Hat Single Sign-On 拒绝授权请求

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

8.3.2. 提交权限请求

作为授权流程的一部分，客户端首先需要从 UMA 保护的资源服务器获取权限票据，以便在 Red Hat Single Sign-On Token Endpoint 中通过 RPT 对其进行交换。

默认情况下，Red Hat Single Sign-On 会响应 **403** HTTP 状态代码和 **request_denied** 错误。如果客户端无法使用 RPT 发布。

Red Hat Single Sign-On 拒绝授权请求

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

此类响应表示 Red Hat Single Sign-On 无法发出 RPT，其权限由权限票据代表。

在某些情况下，客户端应用程序可能希望启动异步授权流，并让所请求的资源所有者决定是否应授予访问权限。为此，客户端可以使用 **submit_request** 请求参数以及到令牌端点的授权请求：

```
curl -X POST \
  http://${host}:${port}/auth/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"
```

使用 commit **_request** 参数时，Red Hat Single Sign-On 将为访问被拒绝的每个资源保留权限请求。创建后，资源所有者可以检查其帐户并管理其权限请求。

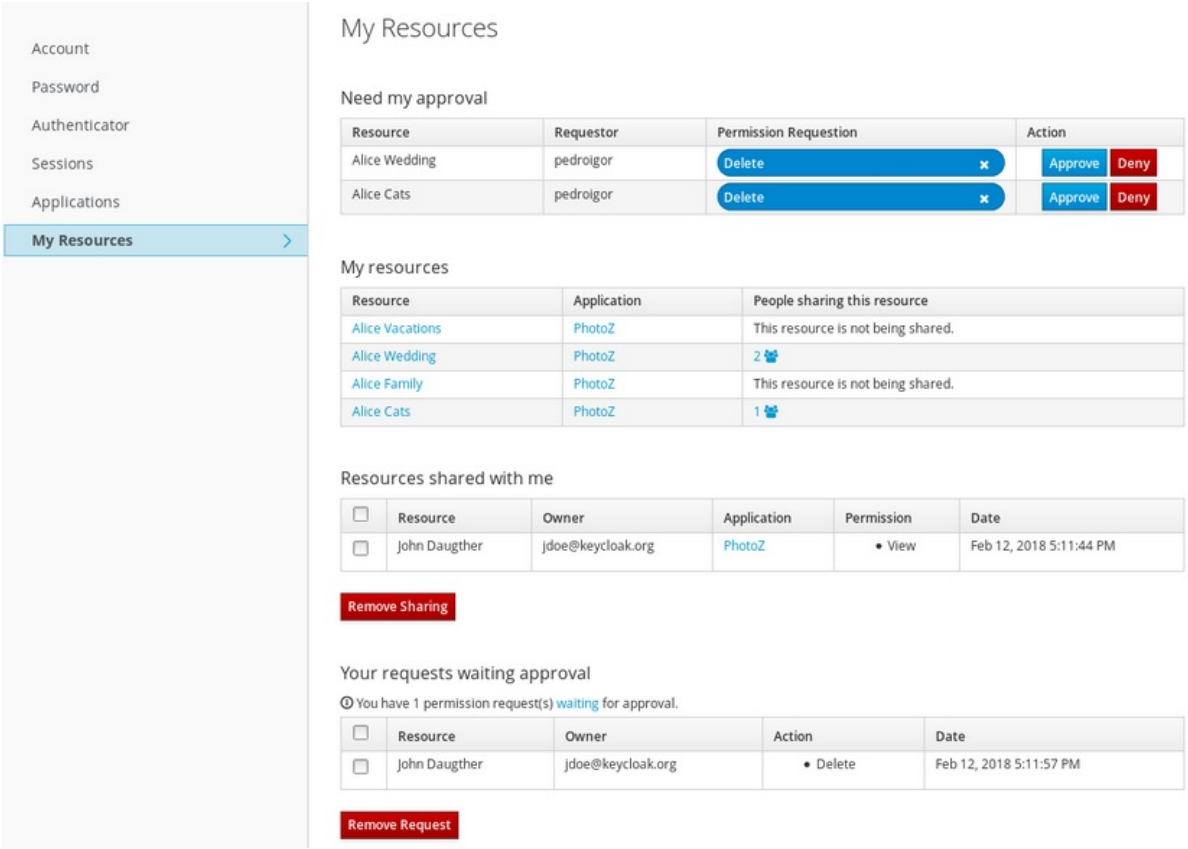
您可以将此功能视为应用程序中的 **Request Access** 按钮，其中用户可以要求其他用户访问其资源。

8.3.3. 管理对用户资源的访问

用户可以通过 Red Hat Single Sign-On 用户帐户服务来管理其资源的访问权限。要启用这个功能，您必须首先为您的域启用 User-Managed Access。

流程

1. 登录 Admin 控制台。
2. 点菜单中的 **Realm Settings**。
3. 将 **用户管理的访问权限切换为 ON**。



4. 在菜单选项中，单击 **My Resources**。此时将显示一个页面，其中包含以下选项：

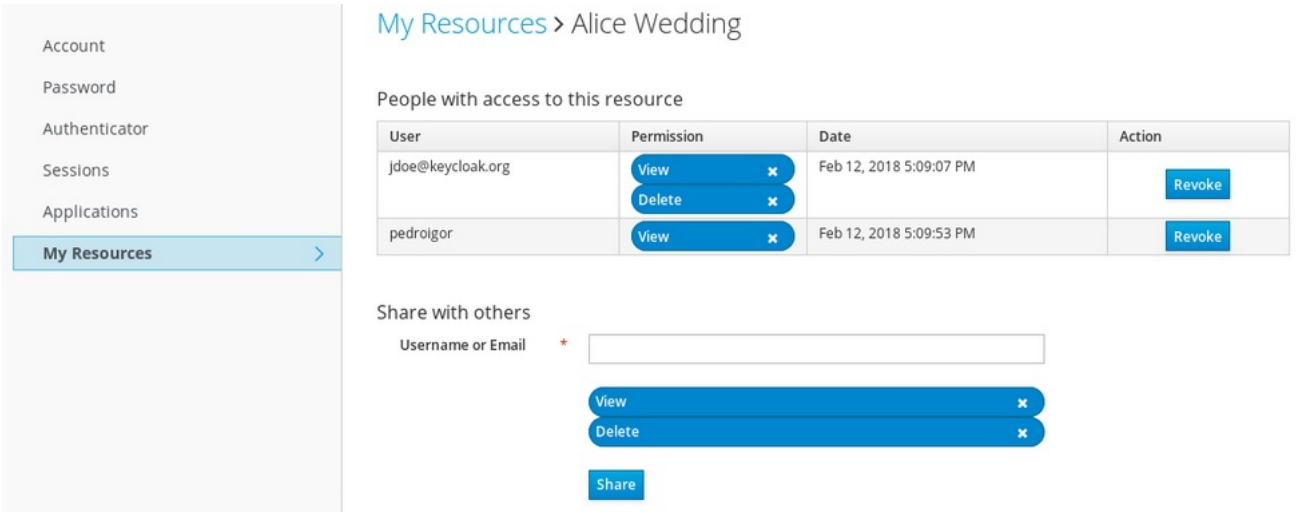
   - 管理 **需要我的批准的**Permission Requests
     本节包含等待批准的所有权限请求的列表。这些请求与请求访问特定资源的各方（用户）连接。用户可以批准或拒绝这些请求。

   - 管理 **我的资源**
     本节包含由用户拥有的所有资源的列表。用户可以单击资源以获取更多详情并与他人共享资源。

   - 管理 **与我共享的资源**
     本节包含与用户共享的所有资源的列表。

   - **管理等待批准的请求**
     本节包含等待其他用户或资源所有者批准的用户发送的权限请求列表。

当您点击特定资源以进行更改时，会显示以下页面：



本页提供以下选项：

- **管理对此资源的访问人员**
  本节包含有权访问此资源的人员的列表。用户可以通过单击 Revoke 按钮或删除特定权限来 **撤销访问权限**。

- 与他人共享资源
  通过输入其他用户的用户名或电子邮件，用户可以共享资源并选择想要授予访问权限的权限。

8.4. 保护 API

Protection API 提供了与 UMA 兼容的端点集合：

- **资源管理**
  使用这个端点，资源服务器可以远程管理其资源，并启用 策略强制 程序查询服务器是否有需要保护的资源。

- **权限管理**
  在 UMA 协议中，资源服务器访问此端点以创建权限票据。Red Hat Single Sign-On 还提供端点来管理权限状态和查询权限。

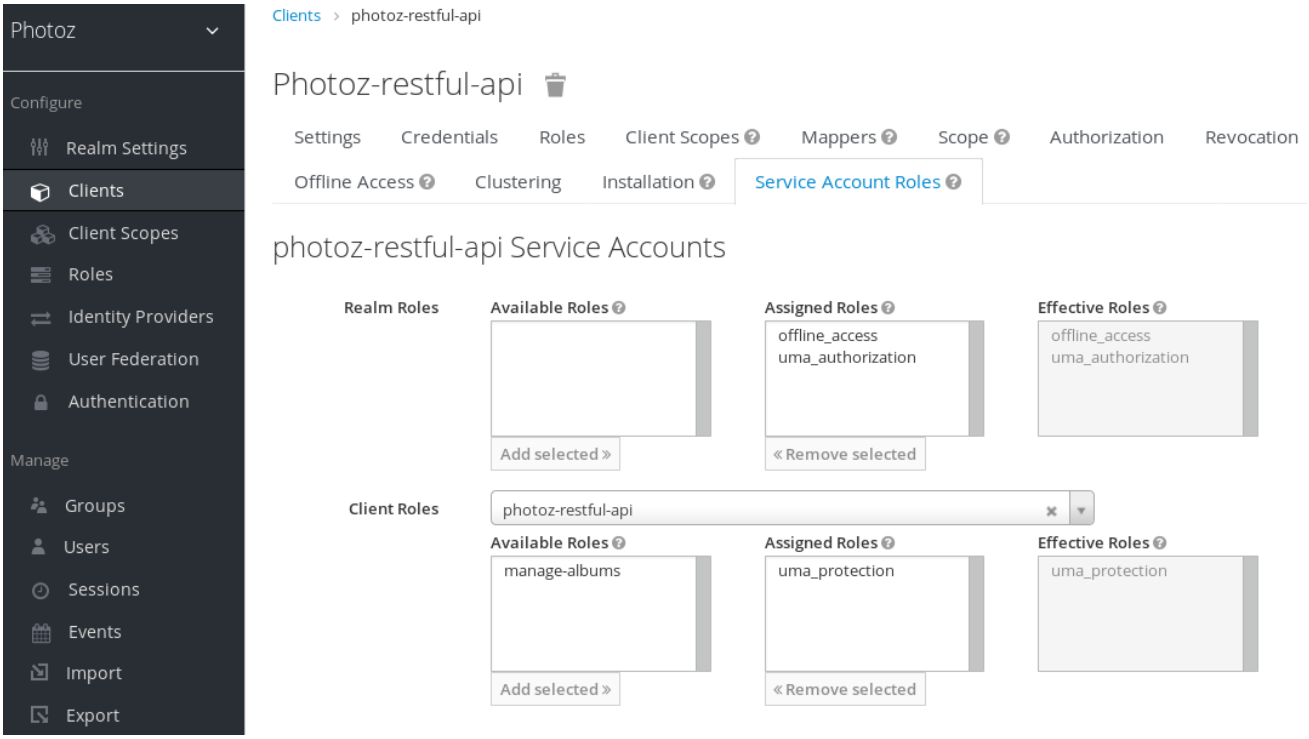
- **Policy API**
  Red Hat Single Sign-On 利用 UMA Protection API 允许资源服务器管理其用户的权限。除了 Resource 和 Permission API 外，Red Hat Single Sign-On 还提供了 Policy API，其中权限可以通过代表他们的用户的资源服务器设置为资源。

此 API 的一个重要要求是，*仅允许* 资源服务器使用名为保护 API 令牌(PAT)的特殊 OAuth2 访问令牌访问其端点。在 UMA 中，PAT 是范围为 **uma_protection** 的令牌。

8.4.1. 什么是 PAT 以及如何获取它

保护 API 令牌 (PAT)是一种特殊的 OAuth2 访问令牌，其范围定义为 **uma_protection**。当您创建资源服务器时，Red Hat Single Sign-On 会自动创建一个角色 *uma_protection* （对应的客户端应用程序），并将它与客户端的服务帐户相关联。

通过 uma_protection 角色授予的服务帐户



与任何其他 OAuth2 访问令牌一样，资源服务器可以从 Red Hat Single Sign-On 获取 PAT。例如，使用 curl：

```
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://localhost:8080/auth/realms/${realm_name}/protocol/openid-connect/token"
```

上面的示例是使用 **client_credentials** 授权类型从服务器获取 PAT。因此，服务器会返回类似如下的响应：

```
{
  "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"
}
```

注意

Red Hat Single Sign-On 可以以不同的方式验证您的客户端应用程序。为了简单起见，此处使用了 **client_credentials** 授权类型，它需要一个 *client_id* 和 *client_secret*。您可以选择使用任何支持的身份验证方法。

8.4.2. 管理资源

资源服务器可以使用兼容 UMA 的端点远程管理其资源。

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

此端点提供如下操作（为清晰起见，忽略路径）：

- 创建资源限制描述：POST /resource_set
- 读取资源设置描述： GET /resource_set/{_id}
- 更新资源设置描述： PUT /resource_set/{_id}
- Delete resource set description: DELETE /resource_set/{_id}
- 列出资源集合描述： GET /resource_set

有关这些操作的合同的更多信息，请参阅 UMA 资源注册 API。

8.4.2.1. 创建资源

要创建资源，您必须发送 HTTP POST 请求，如下所示：

```
curl -v -X POST \
  http://${host}:${port}/auth/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"
      ]
  }'
```

默认情况下，资源的所有者是资源服务器。如果要定义不同的所有者，如特定用户，您可以按照以下方式发送请求：

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

可以使用用户名或用户标识符来设置属性 **所有者**。

8.4.2.2. 创建用户管理的资源

默认情况下，通过保护 API 创建的资源不能由资源所有者通过 用户帐户 服务进行管理。

要创建资源并允许资源所有者管理这些资源，您必须按照如下所示设置 **ownerManagedAccess** 属性：

```
curl -v -X POST \
  http://${host}:${port}/auth/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. 更新资源

要更新现有资源，请按如下所示发送 HTTP PUT 请求：

```
curl -v -X PUT \
  http://${host}:${port}/auth/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. 删除资源

要删除现有资源，请按如下所示发送 HTTP DELETE 请求：

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

8.4.2.5. 查询资源

要根据 **id** 查询资源，请按如下所示发送 HTTP GET 请求：

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

要查询 **名称** 的资源，请按如下所示发送 HTTP GET 请求：

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

默认情况下，**名称** 过滤器将与给定模式匹配的任何资源。要把查询限制为只返回具有相同匹配项的资源，请使用：

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

要查询一个 **uri** 的资源，请按如下所示发送 HTTP GET 请求：

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

要查询一个 **所有者** 的资源，请按如下所示发送 HTTP GET 请求：

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

要查询某个 **类型的资源**，请按如下所示发送 HTTP GET 请求：

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

要查询 **范围** 的资源，请按如下所示发送 HTTP GET 请求：

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

查询服务器以获取权限时，使用 **前** 和 **最大** 结果的参数来限制结果。

8.4.3. 管理权限请求

使用 UMA 协议的资源服务器可以使用特定的端点来管理权限请求。此端点提供了一个与 UMA 兼容的流，用于注册权限请求并获取一个权限票据。

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

权限票据 是一个代表权限请求的特殊安全令牌类型。根据 UMA 规格，权限 ticket 为：

关联句柄，该处理从授权服务器发布到资源服务器（从资源服务器到客户端），并最终从客户端返回到授权服务器，以启用授权服务器来评估正确的策略以应用到授权数据请求。

在大多数情况下，您不需要直接处理此端点。Red Hat Single Sign-On 提供了一个 策略强制 程序，它为您的资源服务器启用 UMA，以便它从授权服务器获取权限票据，返回此票据到客户端应用程序，并根据最终请求方令牌(RPT)强制授权决策。

从 Red Hat Single Sign-On 获取权限票据的过程由资源服务器而不是常规客户端应用程序执行，当客户端试图访问受保护的资源时，则会获得权限票据，而无需必要授权访问该资源。使用 UMA 时，权限问题单是一个重要事项，因为它允许资源服务器实现：

- 来自与资源服务器保护的资源相关的数据的客户端摘要
- 在 Red Hat Single Sign-On 授权请求中注册，稍后可在工作流中使用，以根据资源的所有者同意授予访问权限
- 将资源服务器与授权服务器分离，并允许使用不同的授权服务器保护和管理其资源

客户端明智之，权限票据也很重要，需要强调：

- 客户端不需要了解授权数据如何与受保护的资源关联。对客户端而言完全不透明权限票据。
- 客户端可以访问不同资源服务器上的资源，并由不同的授权服务器进行保护

这些仅仅是 UMA 带来的一些好处，其中 UMA 的其他方面很严重基于权限票据，特别考虑隐私和用户控制对其资源的访问。

8.4.3.1. 创建权限票据

要创建权限 ticket，请按如下所示发送 HTTP POST 请求：

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

在创建问题单时，您还可以推送任意声明，并将这些声明与 ticket 关联：

```
curl -X POST \
  http://${host}:${port}/auth/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"]
    }
  }
]'
```

在评估与权限票据关联的资源和范围时，这些声明可用于您的策略。

8.4.3.2. 其他兼容 UMA 的端点

8.4.3.2.1. 创建权限票据

将带有 id {resource_id} 的特定资源的权限授予具有 id {user_id} 的用户，作为资源发送 HTTP POST 请求的所有者，如下所示：

```
curl -X POST \
    http://${host}:${port}/auth/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. 获取权限问题单

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

您可以使用任何这些查询参数：

- **scopeId**
- **resourceId**
- **owner**
- **requester**
- **授权**
- **returnNames**
- **First**
- **max**

8.4.3.2.3. 更新权限票据

```
curl -X PUT \
    http://${host}:${port}/auth/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. 删除权限票据

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

8.4.4. 使用 Policy API 管理资源权限

Red Hat Single Sign-On 利用 UMA Protection API 允许资源服务器管理其用户的权限。除了 Resource 和 Permission API 外，Red Hat Single Sign-On 还提供了 Policy API，其中权限可以通过代表他们的用户的资源服务器设置为资源。

Policy API 位于：

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

此 API 受一个 bearer 令牌保护，该令牌必须表示用户授权给资源服务器以代表其管理权限。bearer 令牌可以是从令牌端点获取的定期访问令牌：

- 资源所有者密码凭证授予类型
- 令牌交换，为使用者是资源服务器的令牌（公共客户端）交换授予某些客户端（公共客户端）的访问令牌

8.4.4.1. 将权限与资源关联

要将权限与特定资源关联，您必须发送 HTTP POST 请求，如下所示：

```
curl -X POST \
  http://localhost:8180/auth/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"]
}'
```

在上面的示例中，我们会创建新权限并将其与 **resource_id** 表示的资源关联，其中任何具有 role **people-manager** 的用户都应该被 **读取** 范围授予。

您还可以使用其他访问控制机制创建策略，比如使用组：

```
curl -X POST \
  http://localhost:8180/auth/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"]
}'
```

或者一个特定的客户端：

```
curl -X POST \
  http://localhost:8180/auth/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"]
}'
```

或者使用 JavaScript 来使用自定义策略：

注意

上传脚本 **已弃用**，并将在以后的版本中删除。此功能默认为禁用。

使用 **-Dkeycloak.profile.feature.upload_scripts=enabled** 来启用服务器。如需了解更多详细信息，请参阅 配置文件。

```
curl -X POST \
  http://localhost:8180/auth/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"
}'
```

也可以设置这些访问控制机制的任意组合。

要更新现有权限，请按如下所示发送 HTTP PUT 请求：

```
curl -X PUT \
  http://localhost:8180/auth/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. 删除权限

要删除与资源关联的权限，请按如下所示发送 HTTP DELETE 请求：

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

8.4.4.3. 查询权限

要查询与资源关联的权限，请按如下所示发送 HTTP GET 请求：

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

要查询给定其名称的权限，请按如下所示发送 HTTP GET 请求：

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

要查询与特定范围关联的权限，请按如下所示发送 HTTP GET 请求：

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

要查询所有权限，请按如下所示发送 HTTP GET 请求：

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

查询服务器以获取权限时，使用 **前** 和 **最大** 结果的参数来限制结果。

8.5. 请求方令牌

请求方令牌 (RPT) 是使用 JSON web signature (JWS) 进行了数字签名的 JSON web token (JWT)。该令牌基于之前由 Red Hat Single Sign-On 发布的 OAuth2 访问令牌来构建，该令牌代表用户或自行代表。

当解码 RPT 时，您会看到类似如下的有效负载：

```
{
  "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"
}
```

通过此令牌，您可以从 **权限** 声明获取服务器授予的所有权限。

另请注意，权限与您所使用的资源/范围直接相关，并完全脱离用于实际授予和签发相同的权限的访问控制方法。

8.5.1. 内省请求方令牌

有时，您可能希望内省请求方令牌(RPT)来检查其有效期，或获取令牌中的权限，以便在资源服务器端强制执行授权决策。

有两个主要用例，令牌内省可帮助您：

- 当客户端应用程序需要查询令牌有效期以获取具有相同或额外权限的新值时
- 在资源服务器端 强制实施授权决策时，特别是当没有内置策略强制器 适合应用程序时

8.5.2. 获取有关 RPT 的信息

令牌内省本质上是 OAuth2 令牌内省- 兼容端点，您可以获取有关 RPT 的信息。

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

要使用此端点内省 RPT，您可以按照如下方式向服务器发送请求：

```
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/auth/realms/hello-world-authz/protocol/openid-connect/token/introspect"
```

注意

以上请求使用 HTTP BASIC 并通过客户端凭证（客户端 ID 和 secret）来验证客户端试图内省令牌，但您可以使用 Red Hat Single Sign-On 支持的任何其他客户端身份验证方法。

内省端点需要两个参数：

- **token_type_hint**
  使用 **request_party_token** 作为此参数的值，这表示您要内省 RPT。
- **token**
  使用令牌字符串，因为服务器在授权过程中返回，作为此参数的值。

因此，服务器响应是：

```
{
  "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
}
```

如果 RPT 没有被激活，则会返回这个响应：

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

8.5.3. 每次要内省 RPT 时，我是否需要调用服务器？

不如 Red Hat Single Sign-On 服务器发布的常规访问令牌，RPTs 也使用 JSON Web 令牌(JWT)规范作为默认格式。

如果要在没有调用远程内省端点的情况下验证这些令牌，您可以在本地解码 RPT 和查询其有效性。对令牌进行解码后，您还可以使用令牌中的权限来强制执行授权决策。

这基本上是策略执行者的作用。确保：

- 验证 RPT 的签名（基于域的公钥）
- 基于它的 *exp*, *iat*, 和 *aud* 声明来查询令牌的有效性。

其他资源

- JSON Web 令牌(JWT)
- 策略强制器

8.6. 授权客户端 JAVA API

根据您的要求，资源服务器应能够远程管理资源，甚至可以以编程方式检查权限。如果使用 Java，您可以使用授权客户端 API 访问 Red Hat Single Sign-On Authorization Services。

它的目标是希望访问服务器提供的不同端点（如 Token Endpoint、资源和 Permission 管理端点）的资源服务器。

8.6.1. Maven 依赖项

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

8.6.2. Configuration

客户端配置在 **keycloak.json** 文件中定义，如下所示：

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

```
    "secret": "secret"
  }
}
```

- **realm** （必需）
  域的名称。

- **auth-server-url** （必需）
  Red Hat Single Sign-On 服务器的基本 URL。所有其他 Red Hat Single Sign-On 页面和 REST 服务端点都来自于此目的。它通常格式为 https://host:port/auth。

- **resource** （必需）
  应用程序的 client-id。每个应用程序都有一个客户端 ID，用于识别应用程序。

- **凭证** （必需）
  指定应用程序的凭证。这是一个对象表示法，其中键是凭证类型，值是凭证类型的值。

配置文件通常位于应用程序的类路径中，客户端将尝试从中查找 **keycloak.json** 文件的默认位置。

8.6.3. 创建授权客户端

您考虑在类路径中有一个 **keycloak.json** 文件，您可以按照如下所示创建一个新的 **AuthzClient** 实例：

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

8.6.4. 获取用户权利

下面是一个演示了如何获取用户权利的示例：

```
// 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
```

下面是一个示例，它演示了如何为一个或多个资源获取一组用户权利：

```
// 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. 使用保护 API 创建资源

```
// 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);
```

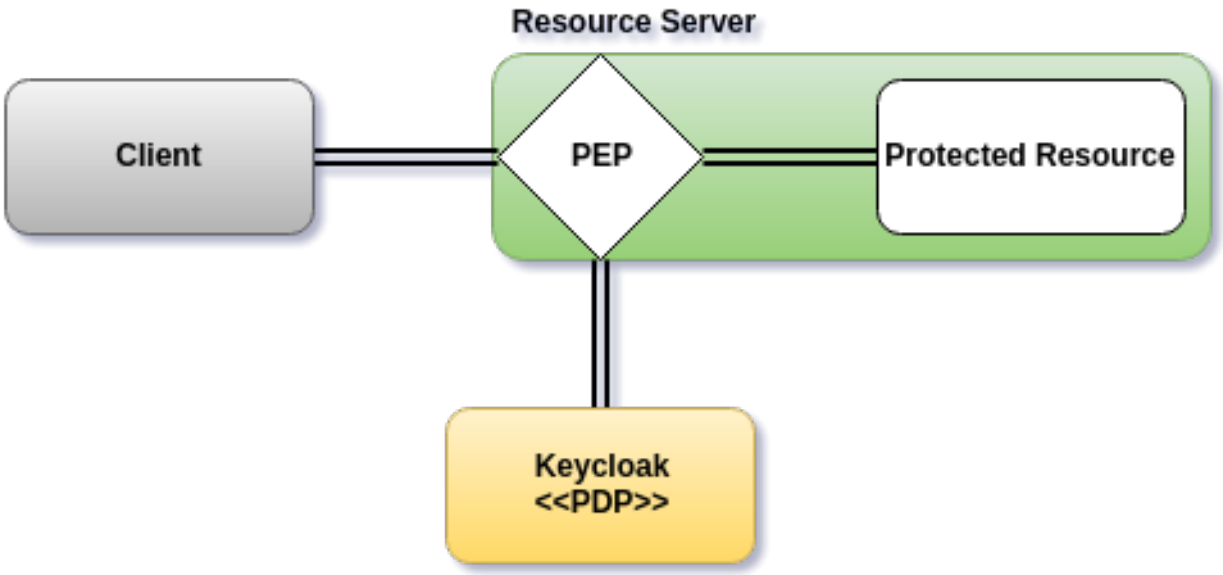
8.6.6. 内省 RPT

```
// 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 策略 ENFORCERS

策略强制点(PEP)是一种设计模式，因此您可以使用不同的方法实施它。Red Hat Single Sign-On 提供了为不同平台、环境和编程语言实施 PEP 所需的所有方法。红帽单点登录授权服务提供了 RESTful API，利用 OAuth2 授权功能使用集中授权服务器进行精细的授权。



PEP 负责评估与受保护资源关联的策略，从红帽单点登录服务器实施这些决策。它充当应用程序中的过滤器或拦截器，以便根据这些决策所授予的权限，检查对受保护资源的特定请求。

根据您使用的协议强制实施权限。使用 UMA 时，策略强制器始终需要一个 RPT 作为 bearer 令牌来确定是否可以提供请求。这意味着，客户端应首先从 Red Hat Single Sign-On 获取 RPT，然后向资源服务器发送请求。

但是，如果您不使用 UMA，您也可以将常规访问令牌发送到资源服务器。在这种情况下，策略 enforcer 将试图直接从服务器获得权限。

如果您使用任何 Red Hat Single Sign-On OIDC 适配器，可以通过在 **keycloak.json** 文件中添加以下属性来轻松启用策略 enforcer：

keycloak.json

```
{
 "policy-enforcer": {}
}
```

当您启用策略强制时，会将发送的所有请求拦截，并根据 Red Hat Single Sign-On 向身份发出的身份授予受保护资源对保护资源的访问。

策略强制使用 Red Hat Single Sign-On 管理控制台与应用程序的路径和您为资源服务器创建的资源相关联。默认情况下，当您创建资源服务器时，Red Hat Single Sign-On 会为资源服务器创建一个 默认配置，以便您可以快速启用策略强制。

9.1. CONFIGURATION

要为应用程序启用策略强制，请在 **keycloak.json** 文件中添加以下属性：

keycloak.json

-

```
{
  "policy-enforcer": {}
}
```

如果要手动定义受保护的资源，请更详细：

```
{
  "policy-enforcer": {
    "user-managed-access" : {},
    "enforcement-mode" : "ENFORCING",
    "paths": [
      {
        "path" : "/someUri/*",
        "methods" : [
          {
            "method": "GET",
            "scopes" : ["urn:app.com:scopes:view"]
          },
          {
            "method": "POST",
            "scopes" : ["urn:app.com:scopes:create"]
          }
        ]
      },
      {
        "name" : "Some Resource",
        "path" : "/usingPattern/{id}",
        "methods" : [
          {
            "method": "DELETE",
            "scopes" : ["urn:app.com:scopes:delete"]
          }
        ]
      },
      {
        "path" : "/exactMatch"
      },
      {
        "name" : "Admin Resources",
        "path" : "/usingWildCards/*"
      }
    ]
  }
}
```

下面是每个配置选项的描述：

- **policy-enforcer**
  指定用来定义如何强制策略以及您要保护的路径（可选）的配置选项。如果没有指定，策略强制查询服务器以获取受保护的资源，以获取与受保护的资源服务器关联的所有资源。在这种情况下，您需要确保资源已使用与您要保护的路径匹配的 URIS 属性正确配置。

  - **user-managed-access**
    指定适配器使用 UMA 协议。如果指定，适配器查询服务器进行权限问题单，并根据 UMA 规格将其返回到客户端。如果没有指定，策略强制器将可以根据常规访问令牌或 RPTs 来强制

实施权限。在这种情况下，在令牌缺少权限时拒绝访问资源，策略强制程序会尝试直接从服务器获得权限。

- enforcement-mode
  指定如何强制实施策略。
  - ENFORCING
    （默认模式）默认拒绝请求，即使没有与给定资源关联的策略。
  - PERMISSIVE
    即使没有与给定资源关联的策略，也会允许请求。
  - DISABLED
    完全禁用策略评估并允许访问任何资源。当 **enforcement-mode** 为 **DISABLED** 应用程序时，仍然可以通过授权 上下文获取 Red Hat Single Sign-On 授予的所有权限
- on-deny-redirect-to
  定义 URL，在从服务器获取"访问被拒绝"消息时会重定向客户端请求。默认情况下，适配器以 403 HTTP 状态代码进行响应。
- path-cache
  定义策略执行程序如何跟踪您在红帽单点登录中定义的应用程序和资源中的路径之间的关联。缓存需要通过缓存路径和受保护的资源之间的关联来避免对 Red Hat Single Sign-On 服务器的不必要的请求。
  - lifespan
    定义条目应过期的时间（毫秒）。如果没有提供，则默认值为 30000。等于 0 的值可以设置为完全禁用缓存。这样设置一个等于 -1 的值来禁用缓存的过期。
  - max-entries
    定义应在缓存中保留的条目的限制。如果没有提供，则默认值为 1000。
- paths
  指定保护的路径。此配置是可选的。如果没有定义，策略强制器将通过获取您在红帽单点登录中定义为应用程序的资源来发现所有路径，其中使用代表应用程序中的一些路径的 **URIS** 定义这些资源。
  - name
    要与给定路径关联的服务器中资源的名称。与 **路径** 结合使用时，策略强制器忽略资源的 **URIS** 属性，并使用您提供的路径。
  - path
    （必需）相对于应用程序的上下文路径的 URI。如果指定了这个选项，策略强制器会查询带有相同值的 **URI** 的服务器。目前支持为路径匹配提供非常基本的逻辑。有效路径示例包括：
    - 通配符： /*
    - 后缀 ：/*.html
    - sub-paths: /**path**/*
    - path 参数：/resource/{id}
    - 确切匹配：/resource
    - pattern: /{version}/resource, /api/{version}/resource, /api/{version}/resource/*

- **方法**
  HTTP 方法（如 GET、POST、PATCH）来保护以及它们如何与服务器中给定资源的范围相关联。
  - **方法**
    HTTP 方法的名称。
  - **范围**
    与方法关联的范围的字符串数组。当您将范围与特定方法关联时，尝试访问受保护的资源（或路径）的客户端必须提供向列表中指定的所有范围授予权限的 RPT。例如，如果您使用范围 *创建* 定义了方法 *POST*，则 RPT 必须包含在向路径执行 POST 到路径时授予 *create* 范围的访问权限。
  - **scopes-enforcement-mode**
    引用与方法关联的范围实施模式的字符串。值可以是 **ALL** 或 **ANY**。**如果所有**，则必须授予所有定义的范围，才能使用该方法访问资源。如果 **ANY**，应至少输入一个范围来访问使用该方法的资源。默认情况下，执行模式设置为 **ALL**。
- **enforcement-mode**
  指定如何强制实施策略。
  - **ENFORCING**
    （默认模式）默认拒绝请求，即使没有与给定资源关联的策略。
  - **DISABLED**
- **claim-information-point**
  定义必须解析并推送到 Red Hat Single Sign-On 服务器的一组或多个声明，以便这些声明可供策略使用。如需了解更多详细信息，请参阅声明信息点。

- **lazy-load-paths**
  指定适配器应该如何为与应用程序中的路径关联的资源获取服务器。如果为 **true**，则策略 enforcer 会根据请求的路径相应地获取资源。当您不想在部署过程中从服务器获取所有资源时（您没有提供 **paths**）或您只定义了 **paths** 的一个子集，并按需获取其他路径，则此配置特别有用。
- **http-method-as-scope**
  指定范围应如何映射到 HTTP 方法。如果设置为 **true**，则策略强制器将使用当前请求的 HTTP 方法来检查是否应授予访问权限。启用后，请确保您的 Red Hat Single Sign-On 中的资源与代表您保护的每个 HTTP 方法的范围相关联。
- **claim-information-point**
  定义一个或多个 **全局** 声明，必须解析并推送到 Red Hat Single Sign-On 服务器，以便这些声明可用于策略。如需了解更多详细信息，请参阅声明信息点。

9.2. 声明信息点

申索信息点(CIP)负责解决这些声明并将这些声明推送到红帽单点登录服务器，以提供访问上下文到策略的更多信息。它们可以定义为 policy-enforcer 的配置选项，以便解析来自不同来源的声明，例如：

- HTTP 请求（参数、标头、正文等）
- 外部 HTTP 服务
- 配置中定义的静态值

- 实施 Claim Information Provider SPI

在向 Red Hat Single Sign-On 服务器推送声明时，策略只能针对用户是谁，也可以通过将上下文和内容纳入帐户，具体取决于什么人、原因、何时以及给定事务。它们都是基于上下文的授权，以及如何使用运行时信息来支持精细的授权决策。

9.2.1. 从 HTTP 请求获取信息

下面是几个示例演示了如何从 HTTP 请求中提取声明：

keycloak.json

```
"policy-enforcer": {
   "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. 从外部 HTTP 服务获取信息

下面是几个示例演示了如何从外部 HTTP 服务提取声明：

keycloak.json

```
"policy-enforcer": {
   "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. 静态声明

keycloak.json

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

9.2.4. 声明信息供应商 SPI

如果任何内置供应商都足够满足要求，则开发人员可使用 Claim Information Provider SPI 来支持不同的声明信息点。

例如，要实施新的 CIP 提供程序，您需要实施 **org.keycloak.adapters.authorization.ClaimInformationPointProvidery** 和 **ClaimInformationPointProvider**，同时还在应用程序类的 **META-INF/services/org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory**.

org.keycloak.adapters.authorization.ClaimInformationPointProviderFactory 示例：

```
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);
    }
}
```

每个 CIP 供应商必须与一个名称关联，如 **MyClaimInformationPointProviderFactory.getName** 方法所定义。name 将用于从 **policy-enforcer** 配置的 **claim-information-point** 部分映射到实施。

在处理请求时，策略 enforcer 将调用 MyClaimInformationPointProviderFactory.create 方法来获取 MyClaimInformationPointProvider 实例。调用时，为这个特定 CIP 提供程序（通过 claim-information-point）定义的配置都会作为映射传递。

ClaimInformationPointProvider 示例：

```
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. 获取授权上下文

启用策略强制后，可通过 **org.keycloak.AuthorizationContext** 获得从服务器获取的权限。此类提供了多种方法，您可以使用 获取权限，以及是否被授予特定资源或范围的权限。

在 Servlet 容器中获取授权上下文

```
    HttpServletRequest request = ... // obtain javax.servlet.http.HttpServletRequest
    KeycloakSecurityContext keycloakSecurityContext =
        (KeycloakSecurityContext) request
```

```
        .getAttribute(KeycloakSecurityContext.class.getName());
AuthorizationContext authzContext =
    keycloakSecurityContext.getAuthorizationContext();
```

注意

有关如何获取 **KeycloakSecurityContext** 的详情，请参考适配器配置。使用由 Red Hat Single Sign-On 支持的任何 servlet 容器运行应用程序时，应该有足够的上下文。

授权上下文可帮助您更好地控制服务器所做的决策和返回。例如，您可以根据与资源或范围关联的权限，使用它来构建隐藏或显示项目的动态菜单。

```
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 代表 Red Hat Single Sign-On 授权服务的主要功能之一。在上面的例子中，您可以看到受保护的资源没有直接与管理它们的策略关联。

使用基于角色的访问控制(RBAC)为例一些类似的代码：

```
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
}
```

虽然这两个示例都满足相同的要求，但它们以不同的方式实现。在 RBAC 中，角色仅 *隐式* 定义其资源的访问权限。通过 Red Hat Single Sign-On，您可以创建更可管理的代码，直接侧重于您的资源，无论您使用 RBAC、基于属性的访问控制(ABAC)，还是其它 BAC 变体。您没有给定资源或范围的权限，或者您没有。

现在，假设您的安全要求已经有所变化，除了项目经理外，PMO 也能够创建新项目。

安全要求改变，但红帽单点登录不需要更改应用程序代码来满足新的要求。应用基于资源和范围标识符后，您只需要更改与授权服务器中特定资源关联的权限或策略的配置。在本例中，与**项目资源**关联的权限和策略，或范围 **urn:project.com:project:create** 将被更改。

9.4. 使用 AUTHORIZATIONCONTEXT 获取授权客户端实例

AuthorizationContext 还可以用来获取对应用程序配置的 授权客户端 API 的引用：

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

在某些情况下，由策略强制保护的资源服务器需要访问授权服务器提供的 API。使用 **AuthzClient** 实例，资源服务器可以与服务器交互，以便以编程方式创建资源或检查特定权限。

9.5. JAVASCRIPT 集成

Red Hat Single Sign-On Server 附带一个 JavaScript 库，可用于与策略 enforcer 保护的资源服务器交互。这个程序库基于 Red Hat Single Sign-On JavaScript 适配器，它可集成来允许您的客户端从 Red Hat Single Sign-On Server 获取权限。

您可以通过在 web 页面中包括 **以下脚本** 标签，从正在运行的 Red Hat Single Sign-On Server 实例获取这个库：

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

完成此操作后，您可以创建一个 **KeycloakAuthorization** 实例，如下所示：

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

keycloak-authz.js 库提供两个主要功能：

- 如果您要访问 UMA 保护的资源服务器，请使用权限 ticket 从服务器获取权限。
- 通过发送资源和范围来从服务器获取权限。

在这两种情况下，库都允许您轻松与资源服务器和 Red Hat Single Sign-On 授权服务交互，以获取您的客户端可以用作 bearer 令牌来访问资源服务器上的受保护资源。

9.5.1. 处理由 UMA-Protected 资源服务器的授权响应

如果资源服务器受策略强制保护，它将根据与 bearer 令牌一起执行的权限响应客户端请求。通常，当您尝试使用 bearer 令牌访问访问受保护资源的权限的资源服务器时，资源服务器会以 401 状态代码和 **WWW-Authenticate** 标头响应。

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

如需更多信息，请参阅 UMA 授权进程。

您的客户端需要从资源服务器返回的 **WWW-Authenticate** 标头中提取权限票据，并使用库发送授权请求，如下所示：

```
// 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
});
```

授权 功能完全异步，支持几个回调功能从服务器接收通知：

- **onGrant** ：功能的第一个参数。如果授权成功，并且服务器返回了具有请求权限的 RPT，则回调将接收 RPT。
- **onDeny** ：功能的第二个参数。只有服务器拒绝授权请求时调用。
- **onError** ：功能的第三个参数。只有服务器意外响应时才调用。

大多数应用都应该使用 **onGrant** 回调在 401 响应后重试请求。后续请求应包含 RPT 作为 bearer 令牌进行重试。

9.5.2. 获取权利

keycloak-authz.js 库提供了一个 **授权** 功能，您可以通过提供您客户端想要访问的资源和范围从服务器获取 RPT。

如何获取具有所有资源权限的 RPT 示例，以及用户可以访问的范围

```
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
});
```

有关如何使用特定资源和范围的权限获取 RPT 的示例

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

使用 **授权** 功能时，您必须提供您要访问的资源服务器的 *client_id*。

权利 功能完全异步，并支持几个回调功能从服务器接收通知：

- **onGrant** ：功能的第一个参数。如果授权成功，并且服务器返回了具有请求权限的 RPT，则回调将接收 RPT。
- **onDeny** ：功能的第二个参数。只有服务器拒绝授权请求时调用。
- **onError** ：功能的第三个参数。只有服务器意外响应时才调用。

9.5.3. 授权请求

授权 和授权 功能都接受授权请求对象。这个对象可使用以下属性设置：

- **权限**
  代表资源和范围的对象数组。例如：

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

- **metadata**
  其属性在其属性的位置，定义服务器应当如何处理授权请求。

  - **response_include_resource_name**
    指示服务器是否应该包含在 RPT 权限中的布尔值。如果为 false，则仅包括资源标识符。

  - **response_permissions_limit**
    为 RPT 的权限数量定义限值的整数 N。与 **rpt** 参数一同使用时，只有最后 N 请求的权限才会保存在 RPT 中

- **submit_request**
  布尔值指示服务器是否应创建对权限票据引用的资源和范围的权限。这个参数仅在与 **ticket** 参数一起使用时作为 UMA 授权进程的一部分时才生效。

9.5.4. 获取 RPT

如果您使用库提供的任何授权功能获得了 RPT，则始终可从授权对象获取 RPT （假设它已被前面显示的技术初始化）：

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

9.6. 配置 TLS/HTTPS

当服务器使用 HTTPS 时，请确保您的适配器配置如下：

keycloak.json

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

以上配置可让 TLS/HTTPS 授权客户端启用 TLS/HTTPS，从而可以使用 HTTPS 方案远程访问红帽单点登录服务器。

注意

强烈建议您在访问 Red Hat Single Sign-On Server 端点时启用 TLS/HTTPS。