Red Hat

Authenticating your applications using Red Hat Single Sign On

Luca Ferrari
EMEA Senior Solution Architect

What is Red Hat Single Sign-On

Red Hat Single Sign-On Architecture and Features

- Based on upstream project Keycloak
- Open source access and identity manager
- Identity Brokering
- User Federation with LDAP based directory services
- Client libraries for JavaEE, Spring, NodeJS, JS + more

Red Hat Single Sign-On Architecture and Features

Single-Sign On
Login once to multiple applications

Standard Protocols
OpenID Connect, OAuth 2.0 and SAML 2.0

Centralized Management
For admins and users

Adapters
Secure applications and services easily

LDAP and Active Directory
Connect to existing user directories

Social Login
Easily enable social login

Identity Brokering
OpenID Connect or SAML 2.0 IdPs

High Performance
Lightweight, fast and scalable

Clustering
For scalability and availability

Themes
Customize look and feel

Extensible
Customize through code

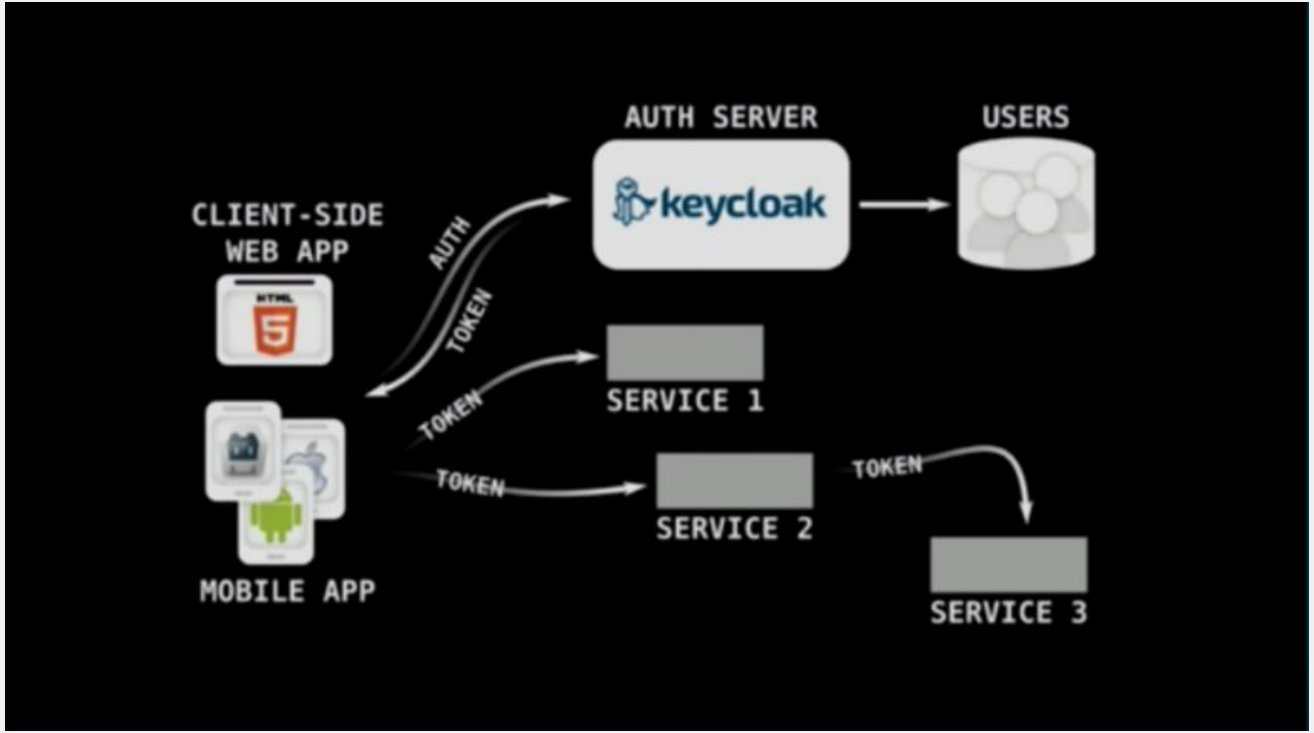
Password Policies
Customize password policies

Red Hat

Red Hat Single Sign-On Core Concepts

redhat.

Red Hat Single Sign-On Architecture and Features



Core Concepts

SSO Concepts I

- **Users:** entities that are able to log into your system. They can have attributes associated with themselves like email, username, address, phone number, and birthday.
- **Credentials:** pieces of data that Red Hat Single Sign-On uses to verify the identity of a user. Some examples are passwords, one-time-passwords, digital certificates, or even fingerprints.
- **Roles:** identify a type or category of user. Applications often assign access and permissions to specific roles rather than individual users as dealing with users can be too fine grained and hard to manage. There are ‘composite roles’
- **Groups:** Users that become members of a group inherit the attributes and role mappings that group defines

Core Concepts

SSO Concepts II

- **Authentication:** The process of identifying and validating a user.
- **Authorization**: The process of granting access to a user.
- **Clients:** entities that can request Red Hat Single Sign-On to authenticate a user. Most often, clients are applications and services that want to use Red Hat Single Sign-On to secure themselves and provide a single sign-on solution
- **Session:** When a user logs in, a session is created to manage the login session. A session contains information like when the user logged in and what applications have participated within single-sign on during that session. Both admins and users can view session information.

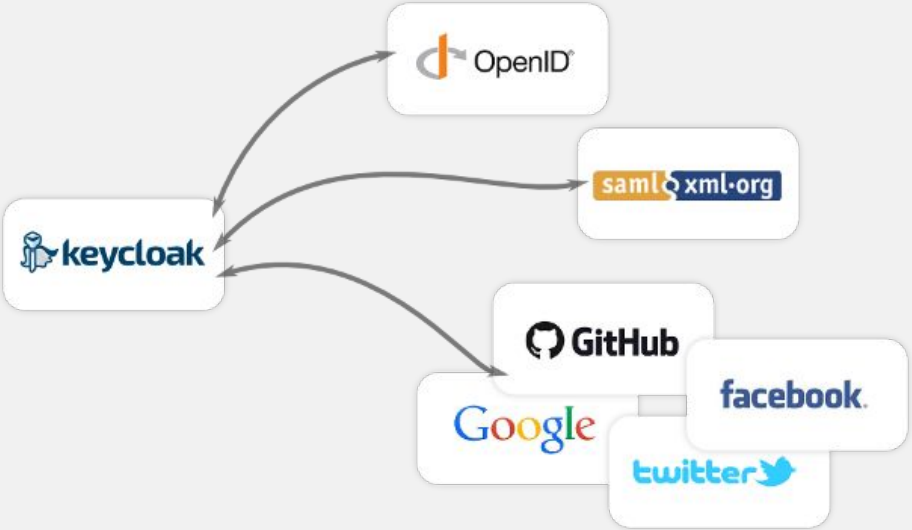
Core Concepts

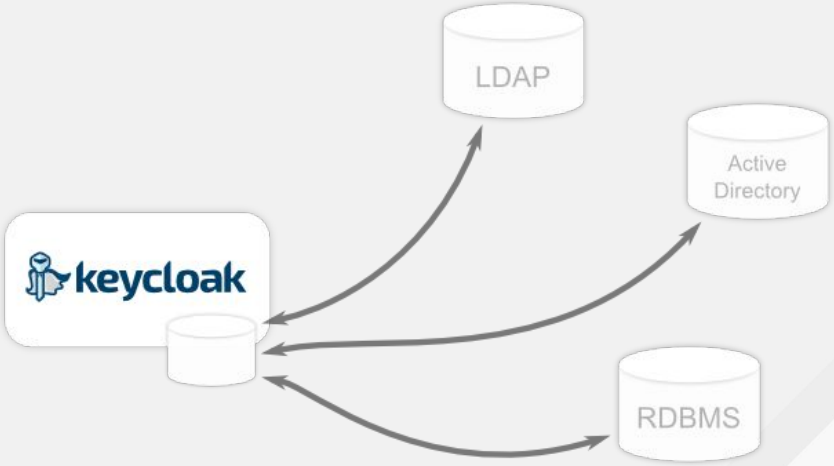
SSO Concepts III

- **Identity Manager (IdM) or Identity Access Manager (IAM):** Component controlling the task of controlling information about users and services on computers. Red Hat Single Sign-On is an IAM
- **Identity Provider (IdP):** service that can authenticate a user. Red Hat Single Sign-On is an IdP
- **Service Provider (SP):** System entity that receives and accepts authentication assets (SAML assertions, OIDC tokens). AKA, applications to protect.
- **Identity Provider Federation:** mechanism to delegate authentication (*identity brokering*) to one or more IDPs. Social login via Facebook or Google+ is an example of identity provider federation. You can also hook Red Hat Single Sign-On to delegate authentication to any other OpenID Connect or SAML 2.0 IDP
- **Identity Provider Mappers:** you can map incoming tokens and assertions to user and session attributes to propagate identity information

Core Concepts

Identity Brokering - User Federation





Red Hat

Core Concepts

Keycloak Concepts I

- **Realm:** manages a set of users, credentials, roles and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.
- **Client Adapters:** plugins that you install into your application environment to be able to communicate and be secured by Red Hat Single Sign-On.
    - Java
    - Javascript
    - Node.js
    - Generic OpenID Connect Resource Provider (RP) libraries: admin endpoints
- **User federation provider:** Red Hat Single Sign-On can store and manage users. You can point Red Hat Single Sign-On to validate credentials from external stores as LDAP or AD and pull in identity information. Custom ones can be implemented.

Core Concepts

Keycloak Concepts II

- **Authentication Flows:** work flows a user must perform when interacting with certain aspects of the system. A login flow can define what credential types are required. A registration flow defines what profile information a user must enter and whether something like reCAPTCHA must be used to filter out bots. Credential reset flow defines what actions a user must do before they can reset their password.
- **Registration flow:** workflow a user must perform when Realm allows user registration
- **Required actions:** actions a user must perform during the authentication process. A user will not be able to complete the authentication process until these actions are complete. For example, an admin may schedule users to reset their passwords every month. An update password required action would be set for all these users.

Clients and Adapters

Client Adapters

- Way of integrate applications in SSO.
- Are libraries that makes it very easy to secure applications and services with RH-SSO
- You can always relay in standard protocols

Software Downloads

Product: Red Hat Single Sign-On

Version: 7.2

Releases (10) | Patches (9) | Security Advisories (9)

Download File	Release Date	
Red Hat Single Sign-On 7.2.0 Server	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 Client Adapter for JBoss EAP 6	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 Client Adapter for JBoss EAP 7	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 Client Adapter for Fuse	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 Node.js Adapter	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 JavaScript Adapter	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 SAML Adapter for JBoss EAP 6	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 SAML Adapter for JBoss EAP 7	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 Source Code	01/31/2018 05:44 AM EDT	Download
Red Hat Single Sign-On 7.2.0 Client Adapters Maven Repository	01/31/2018 05:44 AM EDT	Download

Client Adapters

SAML

- Java adapter for Red Hat JBoss EAP
- mod_auth_mellon: Apache httpd module for SAML

https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.2/html/securing_applications_and_services_guide/saml_2#

Client Adapters

OIDC

- Java adapters
  - JBoss EAP
  - JBoss Fuse
- Javascript adapter: to secure HTML5/JavaScript applications. Supports Cordova
- Node.js adapter: to secure server-side JavaScript apps
- Other OpenID Connect Libraries: Rest API. Standard OIDC and OAuth2 endpoints

https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.2/html/securing_applications_and_services_guide/openid_connect_3

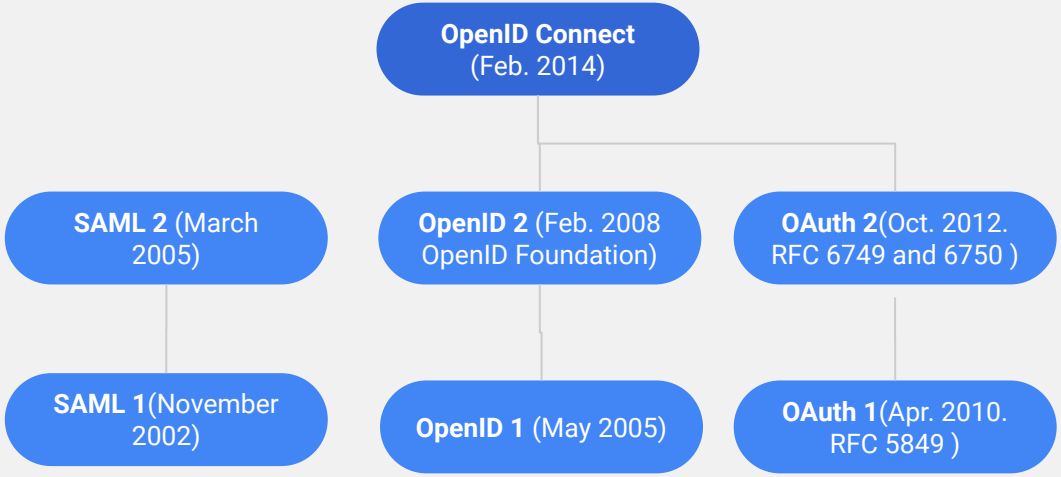
More about Clients

WAIT!!

Some theory about protocols and tokens needed...

OpenID Connect protocol

OIDC History



Main strengths of OIDC

- **Easy to consume identity tokens:** JSON Web Token (JWT)
- **Based on the OAuth 2.0 pseudo protocol:** way for Internet users to grant websites or applications access to their information on other websites but without giving them the password
- **Simpler** than other open alternatives: SAML, or closed implementations
- Ready both for **public** or **enterprise** environments

Identity Token

JWT

- Asserts the identity of the user, called subject in OpenID, and contains user attributes
- Specifies issuing authority (iss) and is generated for a particular audience, the client.
- Specifies issue and expiration dates (iat, exp)
- Digitally signed and, optionally, encrypted.
- JSON Web Token format


```
{
  "sub"        : "alice",
  "iss"        : "https://openid.c2id.com",
  "aud"        : "client-12345",
  "nonce"      : "n-0S6_WzA2Mj",
  "auth_time" : 1311280969,
  "acr"        : "c2id.loa.hisec",
  "iat"        : 1311280970,
  "exp"        : 1311281970,
}
```


After Base64 encode:


```
eyJhbGciOiJSUzI1NiIsImtpZCI6IjFlOWdkazcifQ.ewogImlzcyI6ICJodHRw
Oi8vc2VydmVyLmV4YW1wbGUuY29tIiwKICJzdWIiOiAiMjQ4Mjg5NzYxMDAxIiw
KICJhdWQiOiAiczZCaGRSa3F0MyIsCiAibm9uY2UiOiAibi0wUzZfV3pBMk1qIi
wKICJleHAiOiAxMzExMjgxOTcwLAogImlhdCI6IDEzMTEyODA5NzAKfQ.ggW8hZ
1EuVLuxNuuIJKX_V8a_OMXzR0EHR9R6jgdqrOOF4daGU96Sr_P6qJp6IcmD3HP9
9Obi1PRs-cwh3LO-p146waJ8IhehcwL7F09JdijmBqkvPeB2T9CJNqeGpe-gccM
g4vfKjkM8FcGvnzZUN4_KSP0aAp1tOJ1zZwgjxqGByKHiOtX7TpdQyHE5lcMiKP
XfEIQILVq0pc_E2DzL7emopWoaoZTF_m0_N0YzFC6g6EJbOEoRoSK5hoDalrcvR
YLSrQAZZKflyuVCyixEoV9GfNQC3_osjzw2PAithfubEEBLuVVk4XUVrWOLrLl0
nx7RkKU8NXNHq-rvKMzqg
```


Examples from: https://connect2id.com/learn/openid-connect

OIDC flows for obtaining ID tokens

OAuth defined some flows for obtaining ID tokens that OIDC has incorporated and RH-SSO implements:

- **Authorization Code Flow**: most recommended and common. More secure: tokens not revealed to browser
- **Implicit Flow**: For Javascript apps that have not a backend. Uses less tokens. ID token revealed to browser
- **Direct Access Grants** (Resource Owner Password Credentials Grant): REST clients that want to obtain a token on behalf of a user (with user/password)
- **Client Credentials Grant**: REST clients, but instead of obtaining a token that works on behalf of an external user, a token is created based on the metadata and permissions of a service account that is associated with the client

```
grant_types_supported:
  0:                                    "authorization_code"
  1:                                    "implicit"
  2:                                    "refresh_token"
  3:                                    "password"
  4:                                    "client_credentials"
```

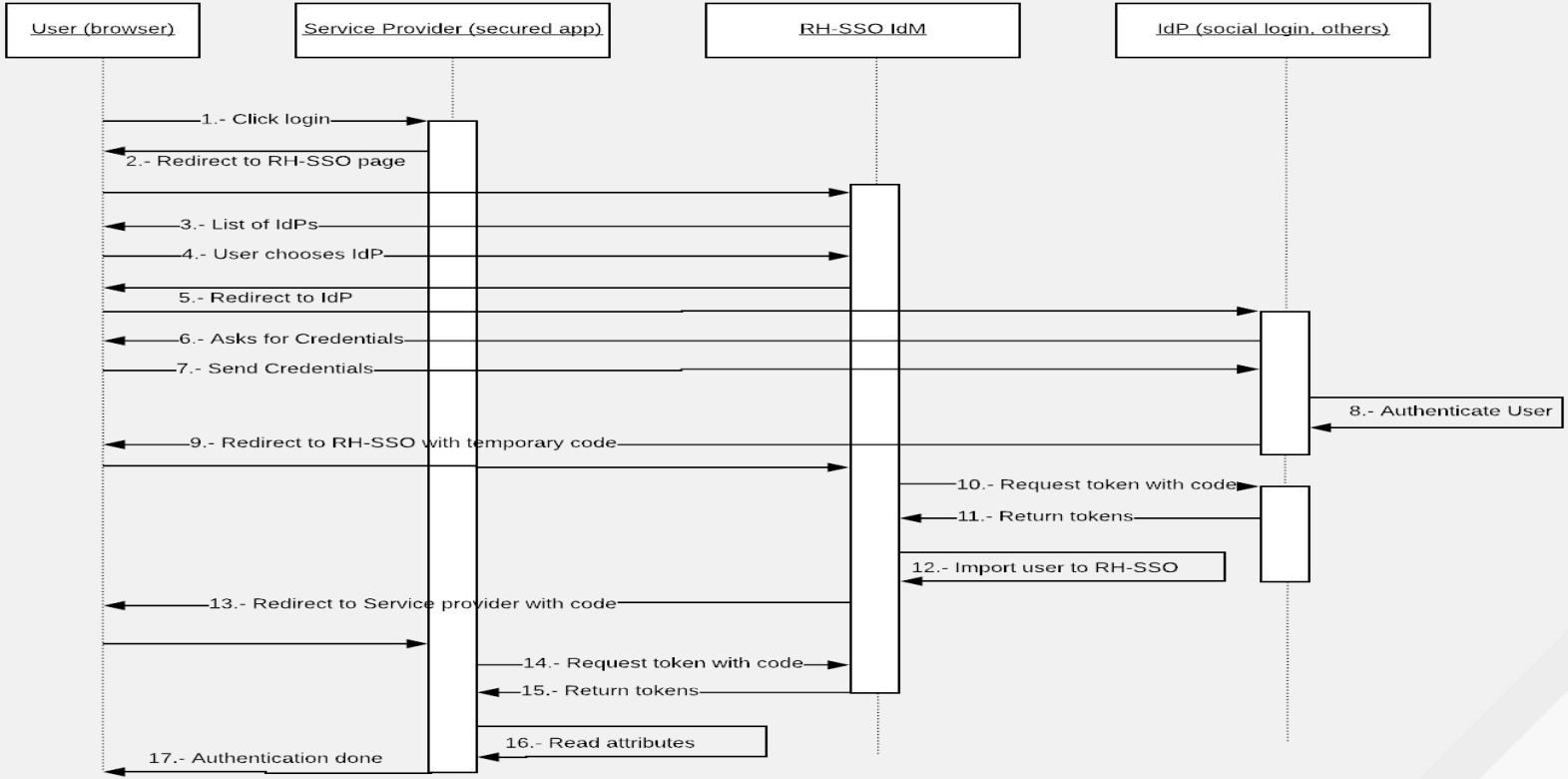
Authorization Flows

Token Types

- Identity Token: described before. Forced to be JWT
- Access Token: used in Auth header (Bearer token)
- Refresh token

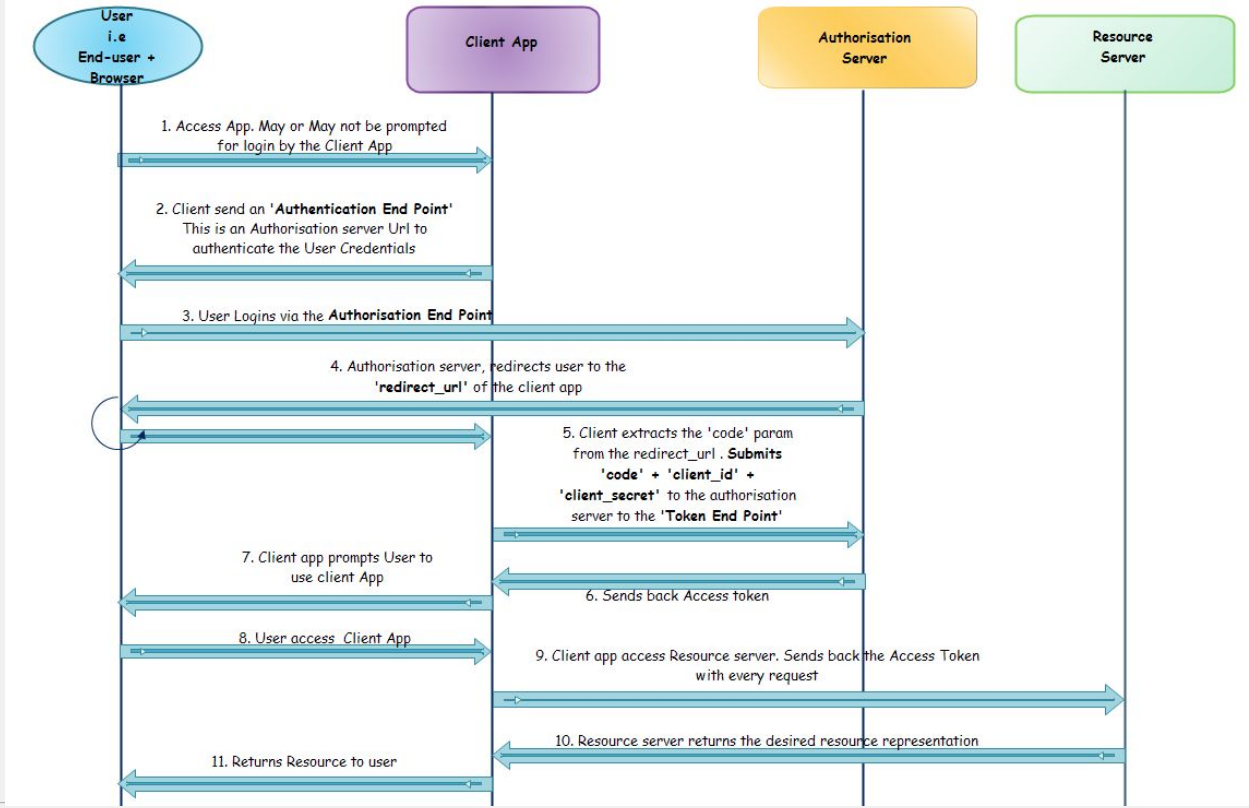
Authorization Code OIDC flow

Flow sequence diagram (with IdP brokering)



Authorization Code OIDC flow

Flow sequence diagram (without IdP brokering). Thanks to Dustin Minnich



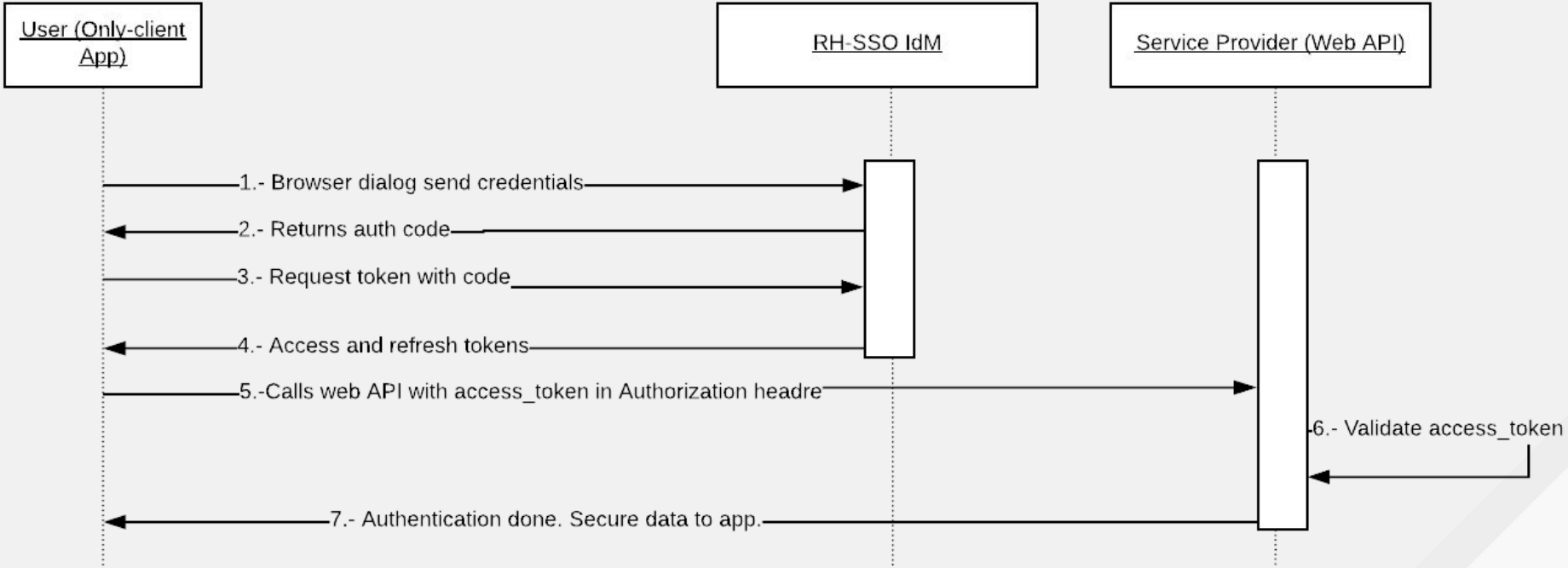
Authorization Code Flow

1. Browser visits application. Application redirects the browser to Red Hat Single Sign-On to be authenticated. There is a callback URL as parameter.

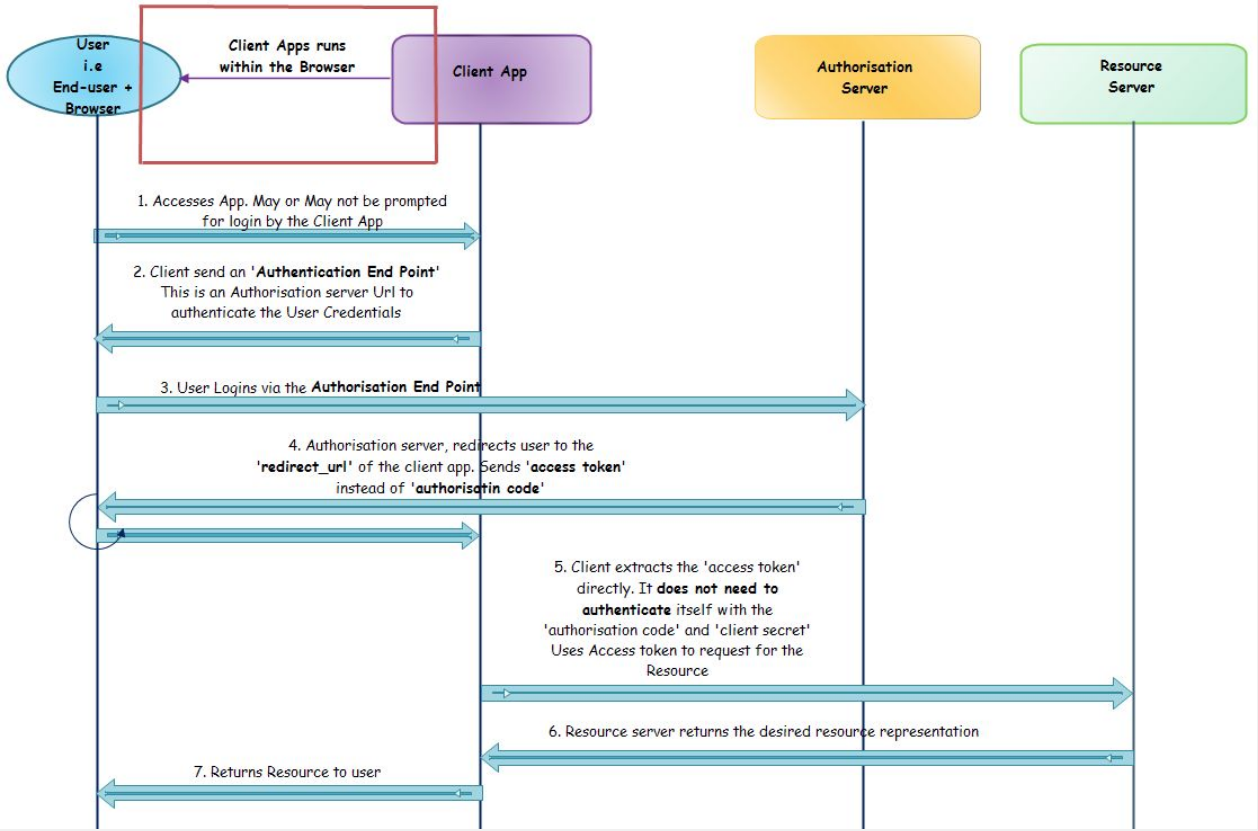
2. Red Hat Single Sign-On authenticates the user and creates a one-time, very short lived, temporary code, and redirect browser to callback URL with code as parameter.

3. The application extracts the temporary code and makes a background out of band REST invocation to Red Hat Single Sign-On to exchange the code for an identity, access and refresh token.

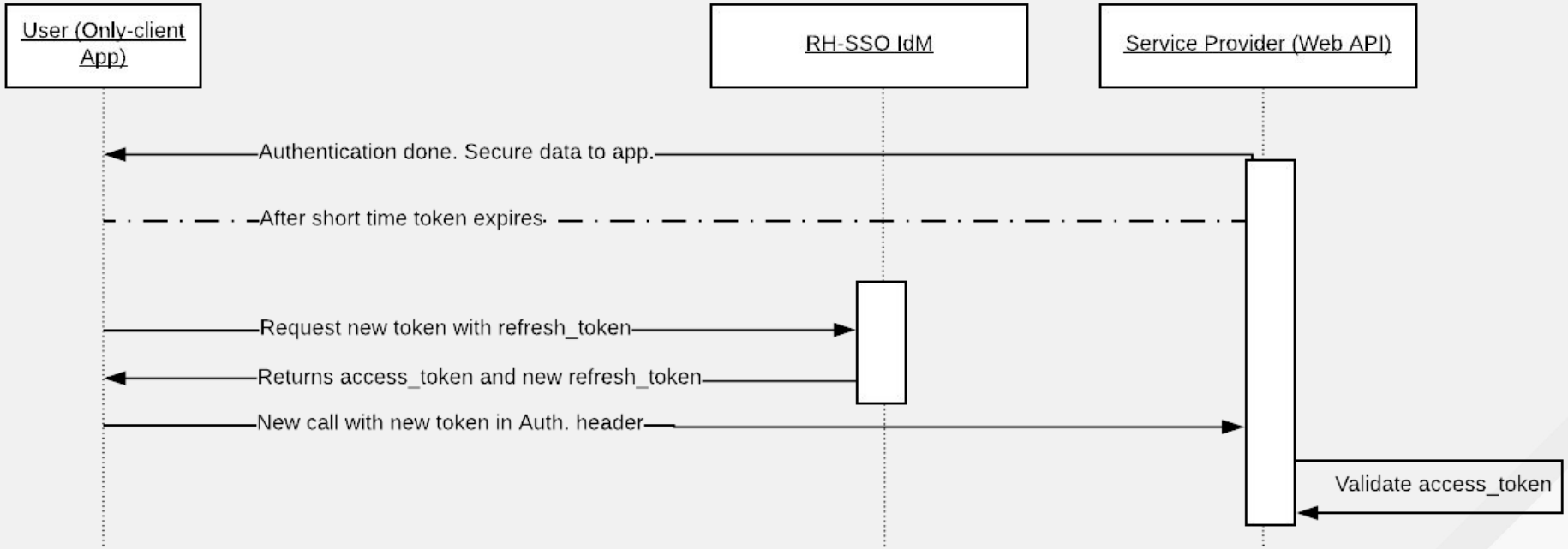
Implicit Flow



Implicit Flow (Dustin Minnich's version)



Refresh Flow



OIDC endpoints

localhost:8080/auth/realms/demo/.well-known/openid-configuration 150% activemq network of broker

JSON | Datos sin procesar | Cabeceras

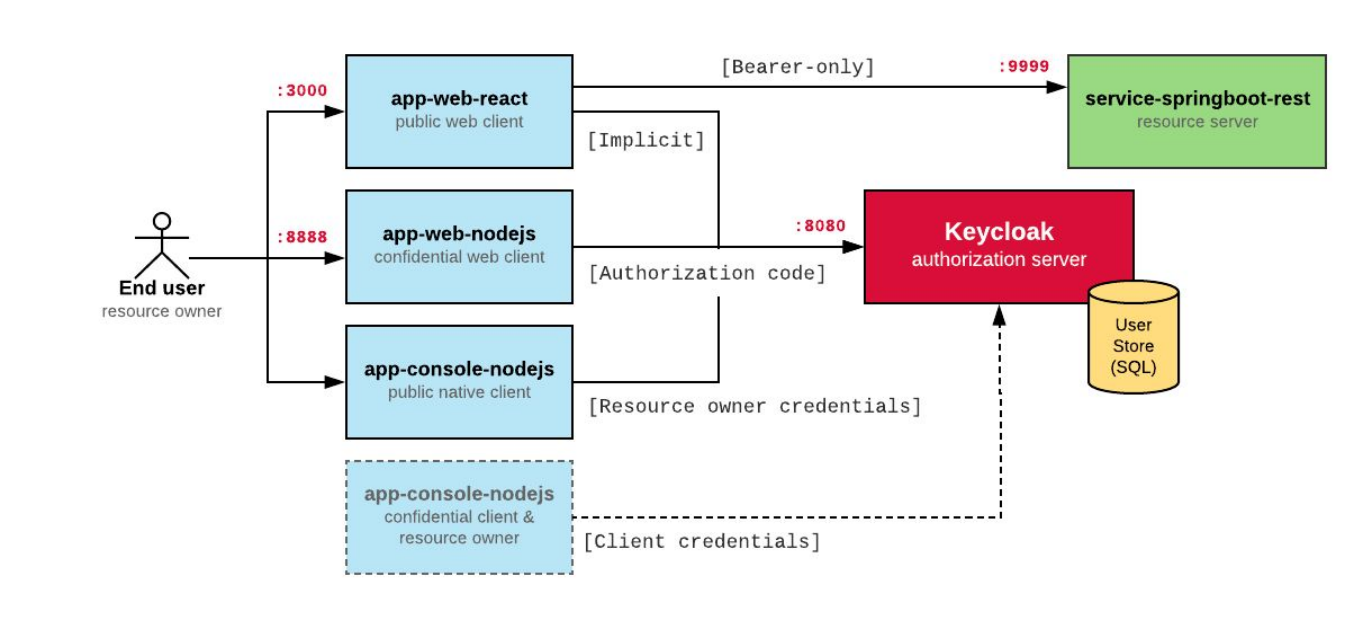
Guardar Copiar Filtrar JSON


```
issuer:                                        "http://localhost:8080/auth/realms/demo"
authorization_endpoint:                        "http://localhost:8080/auth/realms/demo/protocol/openid-connect/auth"
token_endpoint:                                "http://localhost:8080/auth/realms/demo/protocol/openid-connect/token"
token_introspection_endpoint:                  "http://localhost:8080/auth/realms/demo/protocol/openid-connect/token/introspect"
userinfo_endpoint:                             "http://localhost:8080/auth/realms/demo/protocol/openid-connect/userinfo"
end_session_endpoint:                          "http://localhost:8080/auth/realms/demo/protocol/openid-connect/logout"
jwks_uri:                                      "http://localhost:8080/auth/realms/demo/protocol/openid-connect/certs"
check_session_iframe:                          "http://localhost:8080/auth/realms/demo/protocol/openid-connect/login-status-iframe.html"
grant_types_supported:                         […]
response_types_supported:                      […]
subject_types_supported:                       […]
id_token_signing_alg_values_supported:         […]
userinfo_signing_alg_values_supported:         […]
request_object_signing_alg_values_supported:   […]
response_modes_supported:                      […]
registration_endpoint:                         "http://localhost:8080/auth/realms/demo/clients-registrations/openid-connect"
token_endpoint_auth_methods_supported:         […]
token_endpoint_auth_signing_alg_values_supported: […]
claims_supported:                              […]
claim_types_supported:                         […]
claims_parameter_supported:                    false
scopes_supported:                              […]
request_parameter_supported:                   true
request_uri_parameter_supported:               true
```


Red Hat

Demo

Demo



[source: https://github.com/novomatic-tech/keycloak-examples/tree/master/app-web-nodejs]

Demo

1. Login to RH-SSO using the username and password in the Env variables
2. Import the realm from github
3. Show the clients and the users configured inside RH-SSO
4. Navigate to app-web-nodejs
5. npm install
6. Change application.yaml pointing to RHMI SSO
7. npm start
8. Sign in on http://localhost:8888 with jsnow:jsnow
9. Navigate to restricted part of the app

[source: https://github.com/novomatic-tech/keycloak-examples/tree/master/app-web-nodejs]

THANK YOU


plus.google.com/+RedHat


facebook.com/redhatinc


linkedin.com/company/red-hat


twitter.com/RedHat


youtube.com/user/RedHatVideos

BACKUP

IdM Administration
Identity Brokering

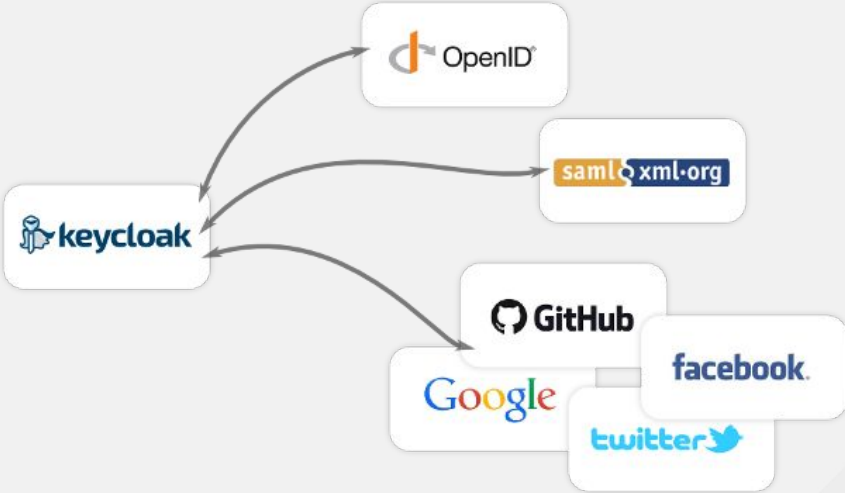
redhat.

Identity Brokering

- An identity Broker can be configured to **delegate authentication** to one or more IDPs. Social login via Facebook or Google+ is an example of identity provider federation.

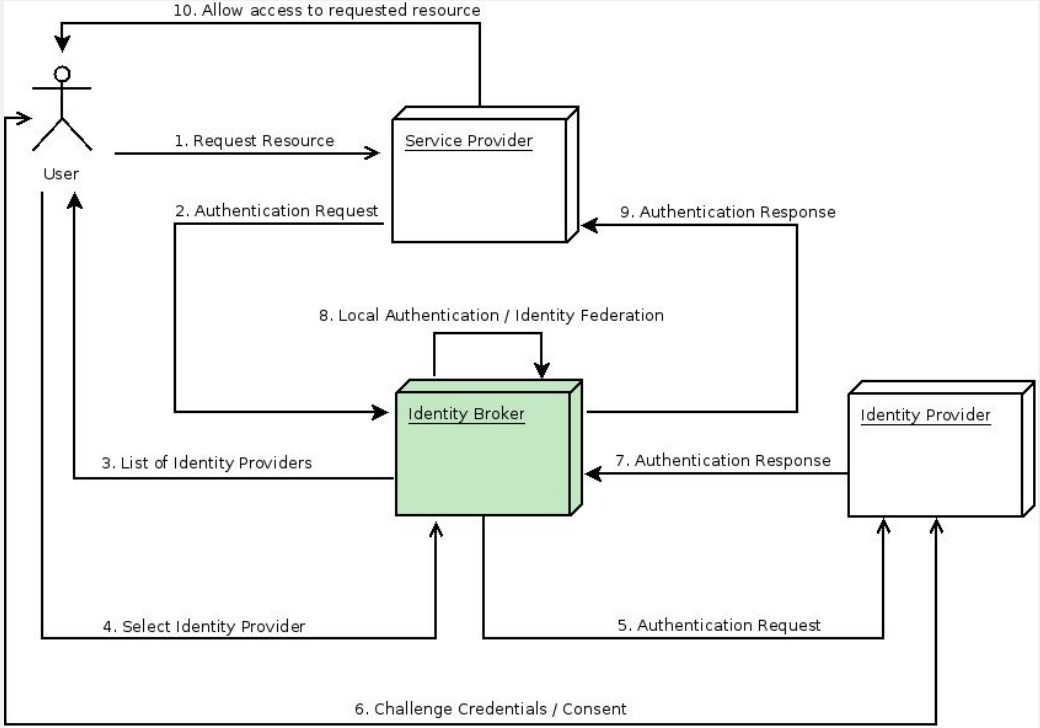
- An Identity Broker is an **intermediary service** that connects multiple service providers with different identity providers

- An identity provider is usually based on a **specific protocol** that is used to authenticate and communicate authentication and authorization information to their users.



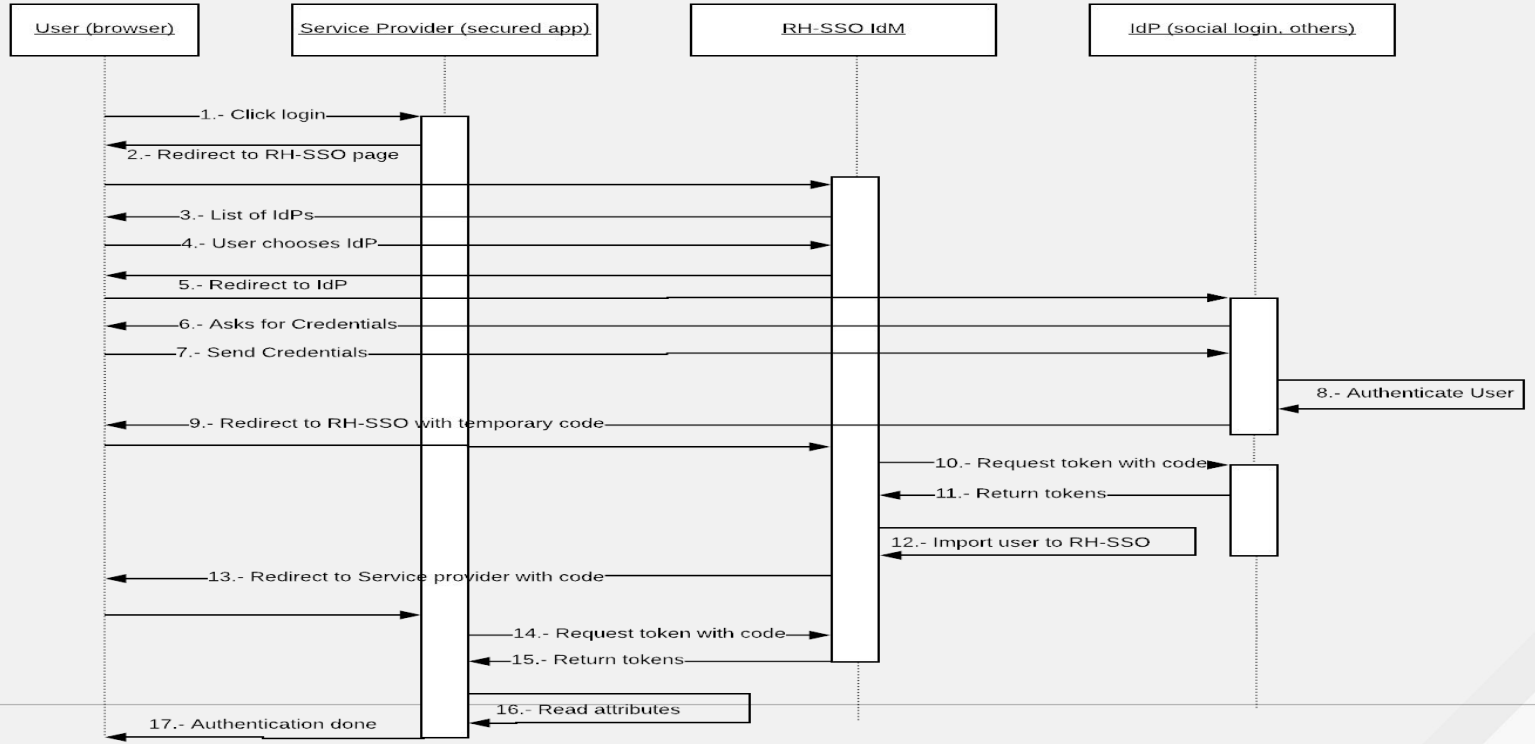
Red Hat

Identity Brokering



Identity Brokering

Sequence



Supported Identity Providers

- Can be resumed in SAML, OpenID Connect or Social Login (through OIDC or OAuth2)

OpenID Connect Config

* Authorization URL
* Token URL
Logout URL
Backchannel Logout OFF
Disable User Info OFF
User Info URL
* Client ID
* Client Secret
Issuer
Default Scopes
Prompt unspecified
Validate Signatures OFF

SAML Config

* Single Sign-On Service URL
Single Logout Service URL
Backchannel Logout OFF
NameID Policy Format Persistent
HTTP-POST Binding Response OFF
HTTP-POST Binding for AuthnRequest OFF
Want AuthnRequests Signed OFF
Force Authentication OFF
Validate Signature OFF

Import External IDP Config

Import from URL
Import from file Select file
Save Cancel

Red Hat

IdM Administration
Authentication & Registration

redhat.

Authentication

Policies

- Password policies shared by realm.
- Policies for One-Time Passwords (2-Factor Authentication)

RED HAT SINGLE SIGN-ON

Admin

Demo

Configure

- Realm Settings
- Clients
- Client Templates
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users

Authentication

Flows | Bindings | Required Actions | Password Policy | OTP Policy

Add policy...

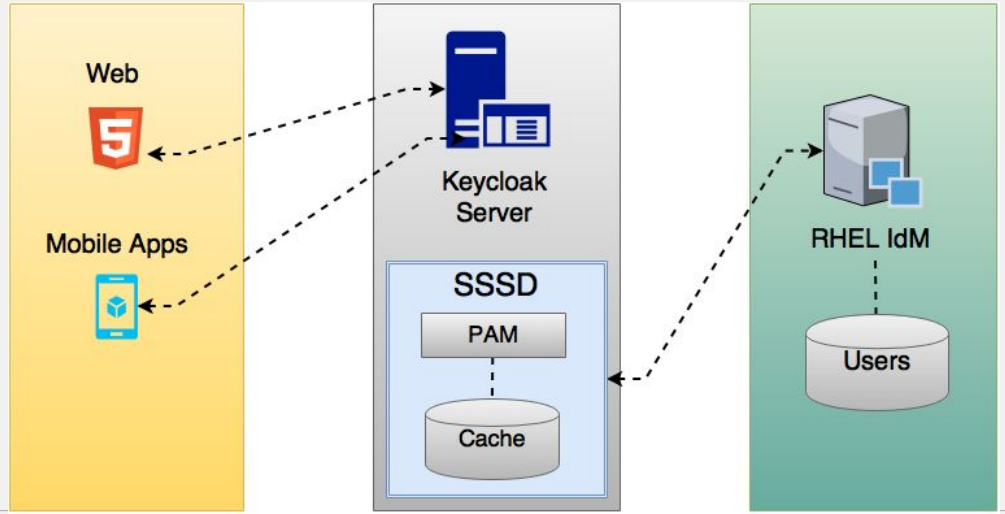
Policy Type	Policy Value	Actions
Hashing Iterations	20000	Delete
Not Username		Delete
Uppercase Characters	1	Delete
Lowercase Characters	1	Delete
Digits	1	Delete
Expire Password	90	Delete

Save Cancel

Authentication

Kerberos

- RH-SSO can use Kerberos for SSO from OS credentials of user: Ticket SPNEGO.
- Kerberos Server can be another User Provider



IdM Flows and Required Actions

- **Authentication Flows:** A RH-SSO authentication flow (not OIDC flows) is a container for all authentications, screens and actions that must happen during login, registration and other Red Hat Single Sign-On workflows
- **Required actions:** actions a user must perform during the authentication process. A user will not be able to complete the authentication process until these actions are complete. For example, an admin may schedule users to reset their passwords every month. An update password required action would be set for all these users.

Authentication Flows

- **Existing authentication flows:** Browser, Direct Grant, Registration, Reset Credentials, Clients, First Broker Login.

- **Auth types:** each authentication flow involve one or more Auth Types: authentication or action that will be executed. Some Auth Type has nested actions

- **Requirement**: each Auth Type can be ‘alternative’, ‘required’ or ‘disabled’. At least one ‘alternative’ flow must match

Registration Service I

- **Activated in Login Tab** of realm

- **Registration service** apply Registration Policies
  - ***Anonymous Access Policies***: used when Client Registration Service is invoked by unauthenticated request. No Initial Access Token nor Bearer Token.
  - ***Authenticated Access Policies***: used when Client Registration Service is invoked by authenticated request. This means request contains Initial Access Token or Bearer Token.

Registration Service II

Demo

Configure

- Realm Settings
- Clients
- Client Templates
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions
- Events
- Import

Demo

General | Login | Keys | Email | Themes | Cache | Tokens | Client Registration | Security Defenses

Initial Access Tokens | Client Registration Policies

Anonymous Access Policies

Add provider...

Policy Name	Provider ID	Actions	
Trusted Hosts	trusted-hosts	Edit	Delete
Consent Required	consent-required	Edit	Delete
Full Scope Disabled	scope	Edit	Delete
Max Clients Limit	max-clients	Edit	Delete
Allowed Protocol Mapper Types	allowed-protocol-mappers	Edit	Delete
Allowed Client Templates	allowed-client-templates	Edit	Delete

Authenticated Access Policies

Add provider...

Policy Name	Provider ID	Actions	
Allowed Protocol Mapper Types	allowed-protocol-mappers	Edit	Delete
Allowed Client Templates	allowed-client-templates	Edit	Delete

Red Hat

Required Actions

- **Tasks** that a user must finish before they are allowed to log in. Can be
  - Configure OTP
  - Terms and Conditions
  - Update Password
  - Update Profile
  - Verify Email

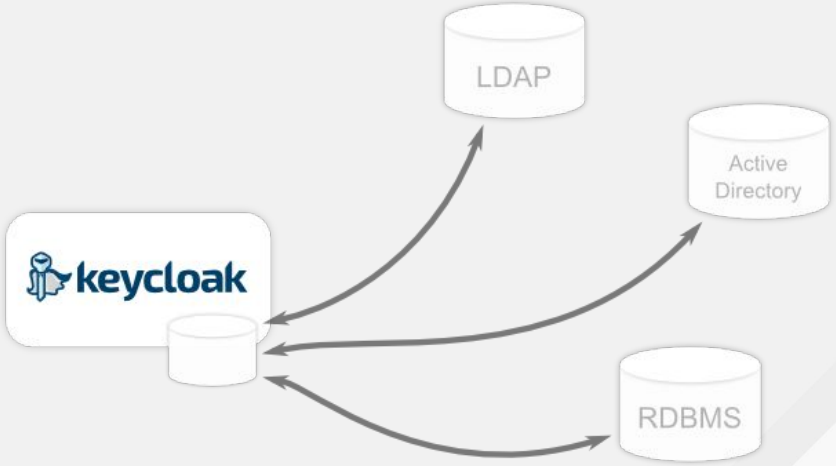
- **User level or default for all realm**

IdM Administration
User Federation

redhat.

User storage federation

- **RH-SSO** can federate multiple external databases containing users. Most common: LDAP and Active Directory
- But RH-SSO maintains its own identity relational database
- RH-SSO provides an API for creating custom plugins for User federation, the [User Storage SPI](#).



Sequence for users lookup

1. RH-SSO needs to look up an user.
2. Search in ‘users’ cache. If not found,
3. Search in local database. If not found,
4. Loops through User Storage SPI provider implementations to perform the user query until one of them returns the user

Local user storage

- RH-SSO always imports users from external providers to its own local storage

- How much metadata: depends on the underlying federation plugin:
  a. Some plugins only import the username
  b. Others import more data (name, address, and phone number)
  c. Some import Credentials to local storage, some others don't

- RH-SSO is an identity layer that can add features to external Users storage

User federation from LDAP I

- LDAP and AD are most common use cases of User Storage Federation in RH-SSO
- You can federate more than one LDAP user providers in same Realm
- You can map LDAP user attributes into the RH-SSO common user model (mappings). By default, it maps:
    a. username
    b. email
    c. first name
    d. last name

User federation from LDAP II

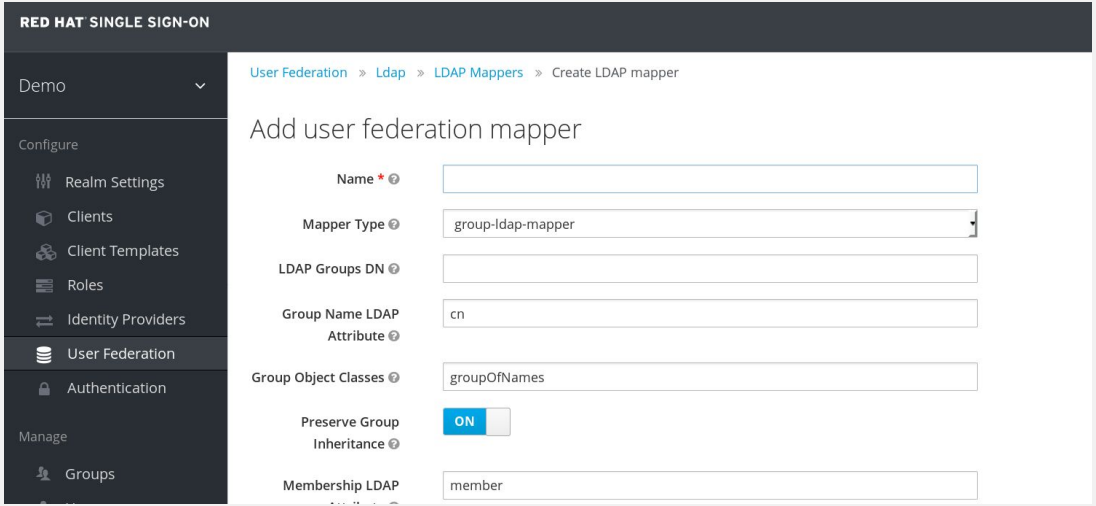
Edition Modes and Synchronization

- 3 **edition modes** of LDAP providers:
  a. ***READONLY***: Username, email, first name, last name and other mapped attributes will be unchangeable. Red Hat Single Sign-On will show an error anytime anybody tries to update these fields. Also password updates will not be supported.
  b. ***WRITABLE***: Username, email, first name, last name and other mapped attributes and passwords can all be updated and will be synchronized automatically with your LDAP store.
  c. ***UNSYNCED***: Any changes to username, email, first name, last name and passwords will be stored in Red Hat Single Sign-On local storage
- 2 types of **synchronization:**
  a. **Periodic Full Sync**: Those LDAP users, which already exist in Red Hat Single Sign-On and were changed in LDAP directly, will be updated in Red Hat Single Sign-On DB
  b. **Periodic changed user Sync**: Only those users that were created or updated after the last sync will be updated and/or imported

User federation from LDAP III

Mappers

- Triggered when a user logs in via LDAP and needs to be imported
- 7 Role Mappers and Group Mappers



Managing Clients

redhat.

SAML and OIDC

```
<samlp:Response xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion" ID="_8e8dc5f0
IssueInstant="2014-07-17T01:01:48Z" Destination="http://sp.example.com/demo1/index.php?acs" InResponseTo="ONELOGIN_4fee3b046395c4e7
  <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer>
  <samlp:Status>
    <samlp:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
  </samlp:Status>
  <saml:Assertion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema" ID="pfx5039e92e
    <saml:Issuer>http://idp.example.com/metadata.php</saml:Issuer><ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
  <ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
    <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
  <ds:Reference URI="#pfx5039e92e-0ec2-015f-5a8e-96219add2782"><ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2000/09/xm
xml-exc-c14n#"/></ds:Transforms><ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><ds:DigestValue>oeJ2C1c+kzgoUX
ds:SignedInfo><ds:SignatureValue>uVLAairZdrTgIk5/0rlI4XDmbnDHh9ZWS+Uqkgpw+vZA6kWUukY28pcMdfFNNPs7Fl0nXvDhVOv/SxWamvKbU9Ez2vhiauvrNT
ds:SignatureValue>
<ds:KeyInfo><ds:X509Data><ds:X509Certificate>MIICajCCAdOgAwIBAgIBADANBgkqhkiG9w0BAQ0FADBSMQswCQYDVQ(
+ON4IUoIWxgukTb1tOiX3bMYzYQiwWPUNMp+Fq82xoNogso2bykZG0yiJm5o8zv/sd6pGouayMgkx/2FSOdc36T0jGbCHuRSbtia
d8wIDAQABo1AwTjAdBgNVHQ4EFgQUGHxYqZYyX7cTxKVODVgZwSTdCnwwHwYDVR0jBBgwFoAUGHxYqZYyX7cTxKVODVgZwSTdCnw
DEe98mXpowhTkC04ENpr0yXi7ZbUqiicF89uAGyt1oqgTUCD1VsLahqIcmrzgumNyTwLGWo17WDAa1/usDhetWAMhgzF/Cnf5ek(
    <saml:Subject>
      <saml:NameID SPNameQualifier="http://sp.example.com/demo1/metadata.php" Format="urn:oasis:name
      <saml:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
        <saml:SubjectConfirmationData NotOnOrAfter="2024-01-18T06:21:48Z" Recipient="http://sp.examp
      </saml:SubjectConfirmation>
    </saml:Subject>
    <saml:Conditions NotBefore="2014-07-17T01:01:18Z" NotOnOrAfter="2024-01-18T06:21:48Z">
      <saml:AudienceRestriction>
        <saml:Audience>http://sp.example.com/demo1/metadata.php</saml:Audience>
      </saml:AudienceRestriction>
    </saml:Conditions>
    <saml:AuthnStatement AuthnInstant="2014-07-17T01:01:48Z" SessionNotOnOrAfter="2024-07-17T09:01:
      <saml:AuthnContext>
        <saml:AuthnContextClassRef>urn:oasis:names:tc:SAML:2.0:ac:classes:Password</saml:AuthnConte
      </saml:AuthnContext>
    </saml:AuthnStatement>
    <saml:AttributeStatement>
      <saml:Attribute Name="uid" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="mail" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">test@example.com</saml:AttributeValue>
      </saml:Attribute>
      <saml:Attribute Name="eduPersonAffiliation" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:basic">
        <saml:AttributeValue xsi:type="xs:string">users</saml:AttributeValue>
        <saml:AttributeValue xsi:type="xs:string">examplerole1</saml:AttributeValue>
      </saml:Attribute>
    </saml:AttributeStatement>
  </saml:Assertion>
</samlp:Response>
```

```
{
  "version":"3.0",
  "issuer":"https:\/\/localhost:9031",
  "authorization_endpoint":"https:\/\/localhost:9031\/as\/authorization.oauth2",
  "token_endpoint":"https:\/\/localhost:9031\/as\/token.oauth2",
  "userinfo_endpoint":"https:\/\/localhost:9031\/idp\/userinfo.openid",
  "jwks_uri":"https:\/\/localhost:9031\/pf\/JWKS",
  "scopes_supported":["phone","address","email","admin","edit","openid","profile"],
  "response_types_supported":["code","token","id_token","code token",
    "code id_token","token id_token","code token id_token"],
  "subject_types_supported":["public"],
  "id_token_signing_alg_values_supported":["none","HS256","HS384","HS512","RS256",
    "RS384","RS512","ES256","ES384","ES512"],
  "token_endpoint_auth_methods_supported":["client_secret_basic","client_secret_post"],
  "claim_types_supported":["normal"],
  "claims_parameter_supported":false,
  "request_parameter_supported":false,
  "request_uri_parameter_supported":false
}
```

Managing OIDC clients I

Demo

Configure

- Realm Settings
- Clients
- Client Templates
- Roles
- Identity Providers

Clients

Search... Create

Client ID	Enabled	Base URL	Actions		
account	True	/auth/realms/demo/account	Edit	Export	Delete
admin-cli	True	Not defined	Edit	Export	Delete
broker	True	Not defined	Edit	Export	Delete
realm-management	True	Not defined	Edit	Export	Delete
security-admin-console	True	/auth/admin/demo/console/index.html	Edit	Export	Delete

Red Hat

Managing OIDC clients II

Demo

Configure
- Realm Settings
- Clients
- Client Templates
- Roles
- Identity Providers
- User Federation
- Authentication

Manage
- Groups
- Users
- Sessions
- Events
- Import

Clients » vanilla

Vanilla

Settings | Roles | Mappers | Scope | Revocation | Sessions | Offline Access | Installation

Client ID	vanilla
Name	
Description	
Enabled	ON
Consent Required	OFF
Client Protocol	openid-connect
Client Template	
Access Type	public
Standard Flow Enabled	ON
Implicit Flow Enabled	OFF
Direct Access Grants Enabled	ON
Root URL	http://localhost:8180/vanilla
* Valid Redirect URIs	http://localhost:8180/vanilla/*

Red Hat

Managing OIDC clients III

Access Types

- ***confidential***:
  server-side clients that need to perform a browser login and require a client secret when they turn an access code into an access token (see Access Token Request in the OAuth 2.0 spec for more details). This type should be used for server-side applications.

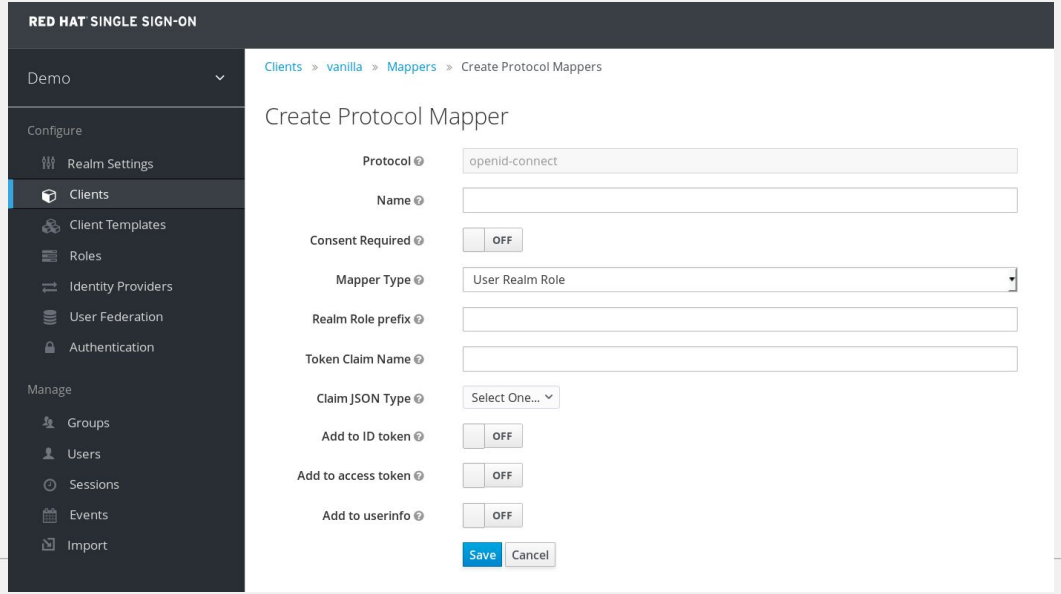
- ***public:***
  For client-side clients that need to perform a browser login. With a client-side application there is no way to keep a secret safe. Instead it is very important to restrict access by configuring correct redirect URIs for the client.

- ***bearer-only:***
  Bearer-only access type means that the application only allows bearer token requests. If this is turned on, this application cannot participate in browser logins. 'Bearer-only' clients are web services that never initiate a login.

Managing Clients

Protocol Mappers

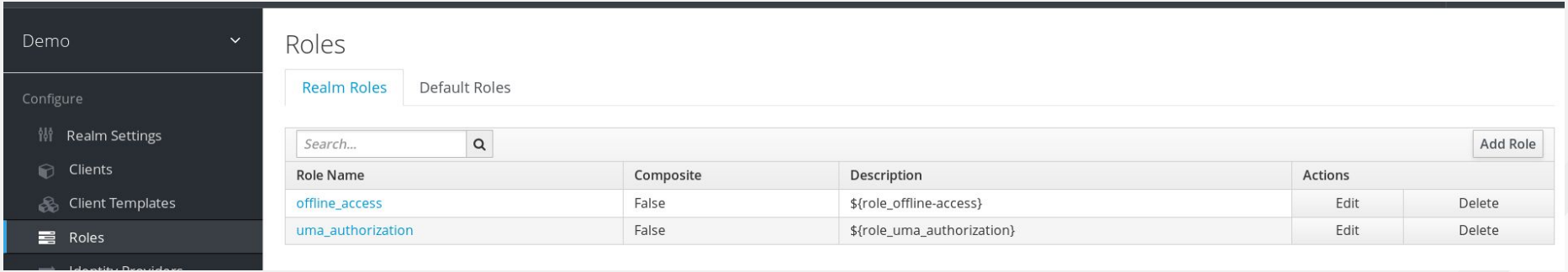
- Useful for add metadata and roles to JWT OIDC token or to SAML assertions
- LDAP mappers can be used with Client Mappers: useful for propagate LDAP roles to clients.



Roles

Realm Level Roles

- Roles identify a type or category of user. Admin, user, manager and employee are all typical roles that may exist in an organization. Applications often assign access and permissions to specific roles
- Realm roles are a global namespace to define roles



Client Level Roles

- One for each client

RED HAT SINGLE SIGN-ON

Admin

Demo

Configure

Realm Settings

Clients

Client Templates

Roles

Identity Providers

Clients » account

Account

Settings | Credentials | Roles | Mappers | Scope | Revocation | Sessions | Offline Access | Clustering | Installation

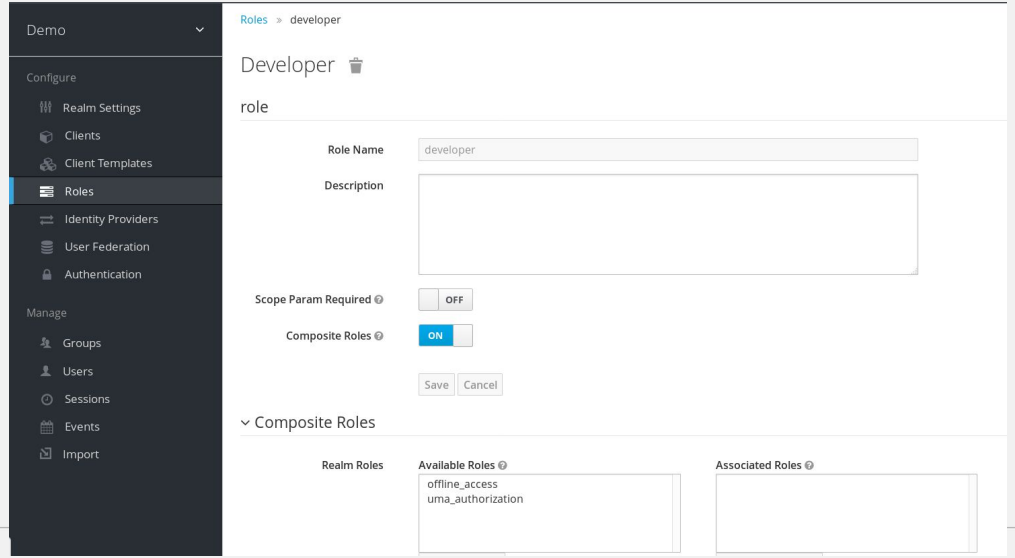
Add Role

Role Name	Composite	Description	Actions	
view-profile	False	${role_view-profile}	Edit	Delete
manage-account	False	${role_manage-account}	Edit	Delete

Red Hat

Composite Roles

- Any realm or client level role can be turned into a composite role. A composite role is a role that has one or more additional roles associated with it
- Users with a granted composited role get permissions from all involved roles



Composite Roles vs Groups

- Use groups to manage users.
- Use composite roles to manage applications and services.

Default Roles

- Are roles automatically assigned to new users when first time login