



Red Hat Single Sign-On 7.6

Server Administration Guide

For Use with Red Hat Single Sign-On 7.6

Red Hat Single Sign-On 7.6 Server Administration Guide

For Use with Red Hat Single Sign-On 7.6

Legal Notice

Copyright © 2024 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Abstract

This guide consists of information for administrators to configure Red Hat Single Sign-On 7.6

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE	12
CHAPTER 1. RED HAT SINGLE SIGN-ON FEATURES AND CONCEPTS	13
1.1. FEATURES	13
1.2. BASIC RED HAT SINGLE SIGN-ON OPERATIONS	14
1.3. CORE CONCEPTS AND TERMS	14
CHAPTER 2. CREATING THE FIRST ADMINISTRATOR	17
2.1. CREATING THE ACCOUNT ON THE LOCAL HOST	17
2.2. CREATING THE ACCOUNT REMOTELY	17
CHAPTER 3. CONFIGURING REALMS	19
3.1. USING THE ADMIN CONSOLE	19
3.2. THE MASTER REALM	20
3.3. CREATING A REALM	21
3.4. CONFIGURING SSL FOR A REALM	22
3.5. CLEARING SERVER CACHES	23
3.6. CONFIGURING EMAIL FOR A REALM	24
3.7. CONFIGURING THEMES AND INTERNATIONALIZATION	25
3.7.1. Enabling internationalization	26
3.7.2. User locale selection	27
3.8. CONTROLLING LOGIN OPTIONS	27
3.8.1. Enabling forgot password	27
3.8.2. Enabling Remember Me	30
3.8.3. ACR to Level of Authentication (LoA) Mapping	31
3.8.3.1. Update Email Workflow (UpdateEmail)	31
3.9. CONFIGURING REALM KEYS	32
3.9.1. Rotating keys	32
3.9.2. Adding a generated keypair	33
3.9.3. Rotating keys by extracting a certificate	33
3.9.4. Adding an existing keypair and certificate	34
3.9.5. Loading keys from a Java Keystore	35
3.9.6. Making keys passive	35
3.9.7. Disabling keys	35
3.9.8. Compromised keys	36
CHAPTER 4. USING EXTERNAL STORAGE	37
4.1. ADDING A PROVIDER	37
4.2. DEALING WITH PROVIDER FAILURES	37
4.3. LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL (LDAP) AND ACTIVE DIRECTORY	38
4.3.1. Configuring federated LDAP storage	38
4.3.2. Storage mode	38
4.3.3. Edit mode	39
4.3.4. Other configuration options	40
4.3.5. Connecting to LDAP over SSL	40
4.3.6. Synchronizing LDAP users to Red Hat Single Sign-On	40
4.3.7. LDAP mappers	41
4.3.8. Password hashing	42
4.3.9. Troubleshooting	43
4.4. SSSD AND FREEIPA IDENTITY MANAGEMENT INTEGRATION	44
4.4.1. FreeIPA/IdM server	45
4.4.2. SSSD and D-Bus	46

4.4.3. Enabling the SSSD federation provider	47
4.5. CONFIGURING A FEDERATED SSSD STORE	47
4.6. CUSTOM PROVIDERS	47
CHAPTER 5. MANAGING USERS	48
5.1. CREATING USERS	48
5.2. DEFINING USER CREDENTIALS	48
5.2.1. Setting a password for a user	49
5.2.2. Creating an OTP	50
5.3. CONFIGURING USER ATTRIBUTES	51
5.4. ALLOWING USERS TO SELF-REGISTER	51
5.4.1. Enabling user registration	52
5.4.2. Registering as a new user	53
5.5. DEFINING ACTIONS REQUIRED AT LOGIN	54
5.5.1. Setting required actions for one user	54
5.5.2. Setting required actions for all users	55
5.5.3. Enabling terms and conditions as a required action	55
5.6. SEARCHING FOR A USER	56
5.7. DELETING A USER	56
5.8. ENABLING ACCOUNT DELETION BY USERS	57
5.8.1. Enabling the Delete Account Capability	57
5.8.2. Giving a user the delete-account role	57
5.8.3. Deleting your account	58
5.9. IMPERSONATING A USER	59
5.10. ENABLING RECAPTCHA	60
5.11. DEFINING A USER PROFILE	62
5.11.1. Enabling the User Profile	63
5.11.2. Managing the User Profile	64
5.11.3. Managing Attributes	65
5.11.3.1. Managing Permissions	67
5.11.3.2. Managing validations	68
5.11.3.2.1. Managing annotations	70
5.11.4. Managing Attribute Groups	70
5.11.5. Using the JSON configuration	72
5.11.5.1. Required property	73
5.11.5.2. Permissions property	74
5.11.5.3. Annotations property	74
5.11.6. Using dynamic forms	75
5.11.6.1. Ordering attributes	76
5.11.6.2. Grouping attributes	76
5.11.6.3. Configuring Form input filed for Attributes	78
5.11.6.3.1. Defining options for select and multiselect fields	81
5.11.7. Forcing User Profile compliance	85
5.11.8. Migrating to User Profile	86
5.12. PERSONAL DATA COLLECTED BY RED HAT SINGLE SIGN-ON	87
CHAPTER 6. MANAGING USER SESSIONS	88
6.1. ADMINISTERING SESSIONS	88
6.1.1. The Logout all Operation	88
6.1.2. Application navigation	89
6.1.3. User navigation	89
6.2. REVOCATION POLICIES	90
6.3. SESSION AND TOKEN TIMEOUTS	90

6.4. OFFLINE ACCESS	94
6.5. OFFLINE SESSIONS PRELOADING	95
6.6. TRANSIENT SESSIONS	96
CHAPTER 7. ASSIGNING PERMISSIONS AND ACCESS USING ROLES AND GROUPS	97
7.1. CREATING A REALM ROLE	97
7.2. CLIENT ROLES	98
7.3. CONVERTING A ROLE TO A COMPOSITE ROLE	98
7.4. ASSIGNING ROLE MAPPINGS	99
7.5. USING DEFAULT ROLES	100
7.6. ROLE SCOPE MAPPINGS	101
7.7. GROUPS	102
7.7.1. Groups compared to roles	104
7.7.2. Using default groups	104
CHAPTER 8. CONFIGURING AUTHENTICATION	106
8.1. PASSWORD POLICIES	106
8.1.1. Password policy types	107
8.1.1.1. Hashing algorithm	107
8.1.1.2. Hashing iterations	107
8.1.1.3. Digits	107
8.1.1.4. Lowercase characters	108
8.1.1.5. Uppercase characters	108
8.1.1.6. Special characters	108
8.1.1.7. Not username	108
8.1.1.8. Not email	108
8.1.1.9. Regular expression	108
8.1.1.10. Expire password	108
8.1.1.11. Not recently used	108
8.1.1.12. Password blacklist	108
8.2. ONE TIME PASSWORD (OTP) POLICIES	108
8.2.1. Time-based or counter-based one time passwords	109
8.2.2. TOTP configuration options	109
8.2.2.1. OTP hash algorithm	109
8.2.2.2. Number of digits	110
8.2.2.3. Look around window	110
8.2.2.4. OTP token period	110
8.2.3. HOTP configuration options	110
8.2.3.1. OTP hash algorithm	110
8.2.3.2. Number of digits	110
8.2.3.3. Look ahead window	110
8.2.3.4. Initial counter	110
8.3. AUTHENTICATION FLOWS	110
8.3.1. Built-in flows	111
8.3.1.1. Auth type	111
8.3.1.2. Requirement	112
8.3.1.2.1. Required	112
8.3.1.2.2. Alternative	112
8.3.1.2.3. Disabled	112
8.3.1.2.4. Conditional	112
8.3.2. Creating flows	112
8.3.3. Creating a password-less browser login flow	115
8.3.4. Creating a browser login flow with step-up mechanism	119

8.3.5. Configuring user session limits	124
8.4. KERBEROS	125
8.4.1. Setup of Kerberos server	126
8.4.2. Setup and configuration of Red Hat Single Sign-On server	126
8.4.2.1. Enabling SPNEGO processing	127
8.4.2.2. Configure Kerberos user storage federation providers	127
8.4.3. Setup and configuration of client machines	129
8.4.4. Credential delegation	129
8.4.5. Cross-realm trust	130
8.4.6. Troubleshooting	132
8.5. X.509 CLIENT CERTIFICATE USER AUTHENTICATION	132
8.5.1. Features	133
8.5.1.1. Regular expressions	133
8.5.1.1.1. Mapping certificate identity to an existing user	133
8.5.1.1.2. Extended certificate validation	134
8.5.2. Enable X.509 client certificate user authentication	134
8.5.2.1. Enable mutual SSL in JBoss EAP	134
8.5.2.2. Enable HTTPS listener	135
8.5.3. Adding X.509 client certificate authentication to browser flows	135
8.5.4. Configuring X.509 client certificate authentication	137
8.5.5. Adding X.509 Client Certificate Authentication to a Direct Grant Flow	140
8.5.6. Client certificate lookup	141
8.5.6.1. HAProxy certificate lookup provider	142
8.5.6.2. Apache certificate lookup provider	142
8.5.6.3. NGINX certificate lookup provider	142
8.5.6.4. Other reverse proxy implementations	143
8.5.7. Troubleshooting	143
8.6. W3C WEB AUTHENTICATION (WEBAUTHN)	145
8.6.1. Setup	145
8.6.1.1. Enable WebAuthn authenticator registration	146
8.6.1.2. Adding WebAuthn authentication to a browser flow	146
8.6.2. Authenticate with WebAuthn authenticator	148
8.6.3. Managing WebAuthn as an administrator	149
8.6.3.1. Managing credentials	149
8.6.3.2. Managing policy	149
8.6.4. Attestation statement verification	151
8.6.5. Managing WebAuthn credentials as a user	151
8.6.5.1. Register WebAuthn authenticator	151
8.6.5.2. New user	151
8.6.5.3. Existing user	152
8.6.6. Passwordless WebAuthn together with Two-Factor	152
8.6.6.1. Setup	152
8.6.7. LoginLess WebAuthn	153
8.6.7.1. Setup	154
8.6.7.2. Vendor specific remarks	154
8.6.7.2.1. Compatibility check list	154
8.6.7.2.2. Windows Hello	155
8.6.7.2.3. Supported security keys	155
8.7. RECOVERY CODES (RECOVERYCODES)	155
8.8. CONDITIONS IN CONDITIONAL FLOWS	155
8.8.1. Available conditions	155
8.8.2. Explicitly deny/allow access in conditional flows	156

CHAPTER 9. INTEGRATING IDENTITY PROVIDERS	158
9.1. BROKERING OVERVIEW	158
9.2. DEFAULT IDENTITY PROVIDER	160
9.3. GENERAL CONFIGURATION	160
9.4. SOCIAL IDENTITY PROVIDERS	164
9.4.1. Bitbucket	164
9.4.2. Facebook	165
9.4.3. GitHub	167
9.4.4. GitLab	168
9.4.5. Google	169
9.4.6. LinkedIn	170
9.4.7. Microsoft	171
9.4.8. OpenShift 3	171
9.4.9. OpenShift 4	173
9.4.10. PayPal	175
9.4.11. Stack overflow	176
9.4.12. Twitter	178
9.4.13. Instagram	179
9.5. OPENID CONNECT V1.0 IDENTITY PROVIDERS	183
9.6. SAML V2.0 IDENTITY PROVIDERS	187
9.6.1. Requesting specific AuthnContexts	191
9.6.2. SP Descriptor	191
9.6.3. Send subject in SAML requests	191
9.7. CLIENT-SUGGESTED IDENTITY PROVIDER	192
9.8. MAPPING CLAIMS AND ASSERTIONS	192
9.9. AVAILABLE USER SESSION DATA	193
9.10. FIRST LOGIN FLOW	194
9.10.1. Default first login flow authenticators	194
9.10.2. Automatically link existing first login flow	196
9.10.3. Disabling automatic user creation	196
9.10.4. Detect existing user first login flow	197
9.11. RETRIEVING EXTERNAL IDP TOKENS	197
9.12. IDENTITY BROKER LOGOUT	198
CHAPTER 10. SSO PROTOCOLS	199
10.1. OPENID CONNECT	199
10.1.1. OIDC auth flows	199
10.1.1.1. Authorization Code Flow	199
10.1.1.2. Implicit Flow	200
10.1.1.3. Resource owner password credentials grant (Direct Access Grants)	201
10.1.1.4. Client credentials grant	201
10.1.1.5. Device authorization grant	201
10.1.1.6. Client initiated backchannel authentication grant	201
10.1.1.6.1. CIBA Policy	202
10.1.1.6.2. Provider Setting	202
10.1.1.6.3. Authentication Channel Provider	203
10.1.1.6.4. User Resolver Provider	206
10.1.2. OIDC Logout	206
10.1.2.1. Session Management	207
10.1.2.2. RP-Initiated Logout	207
10.1.2.3. Frontchannel Logout	207
10.1.2.4. Backchannel Logout	208
10.1.3. Red Hat Single Sign-On server OIDC URI endpoints	208

10.2. SAML	208
10.2.1. SAML bindings	209
10.2.1.1. Redirect binding	209
10.2.1.2. POST binding	209
10.2.1.3. ECP	210
10.2.2. Red Hat Single Sign-On Server SAML URI Endpoints	210
10.3. OPENID CONNECT COMPARED TO SAML	210
10.4. DOCKER REGISTRY V2 AUTHENTICATION	210
10.4.1. Docker authentication flow	211
10.4.2. Red Hat Single Sign-On Docker Registry v2 Authentication Server URI Endpoints	211
CHAPTER 11. CONTROLLING ACCESS TO THE ADMIN CONSOLE	212
11.1. MASTER REALM ACCESS CONTROL	212
11.1.1. Global roles	212
11.1.2. Realm specific roles	212
11.2. DEDICATED REALM ADMIN CONSOLES	213
11.3. FINE GRAIN ADMIN PERMISSIONS	213
11.3.1. Managing one specific client	214
11.3.1.1. Permission setup	214
11.3.1.2. Testing it out	218
11.3.2. Restrict user role mapping	218
11.3.2.1. Testing it out	221
11.3.2.2. Per client map-roles shortcut	221
11.3.3. Full list of permissions	222
11.3.3.1. Role	222
11.3.3.2. Client	222
11.3.3.3. Users	223
11.3.3.4. Group	223
CHAPTER 12. MANAGING OPENID CONNECT AND SAML CLIENTS	225
12.1. OIDC CLIENTS	225
12.1.1. Creating an OpenID Connect Client	225
12.1.2. Basic settings	226
12.1.3. Advanced settings	228
12.1.4. Confidential client credentials	232
12.1.5. Client Secret Rotation	236
12.1.5.1. Rules for client secret rotation	237
12.1.6. Creating an OIDC Client Secret Rotation Policy	237
12.1.7. Using a service account	240
12.1.8. Audience support	242
12.1.8.1. Setup	243
12.1.8.2. Automatically add audience	243
12.1.8.3. Hardcoded audience	244
12.2. CREATING A SAML CLIENT	245
12.2.1. IDP Initiated login	249
12.2.2. Using an entity descriptor to create a client	250
12.3. CLIENT LINKS	251
12.4. OIDC TOKEN AND SAML ASSERTION MAPPINGS	251
12.4.1. Priority order	253
12.4.2. OIDC user session note mappers	253
12.4.3. Script mapper	254
12.5. GENERATING CLIENT ADAPTER CONFIG	254
12.6. CLIENT SCOPES	254

12.6.1. Protocol	255
12.6.2. Consent related settings	256
12.6.3. Link client scope with the client	256
12.6.3.1. Example	256
12.6.4. Evaluating Client Scopes	257
12.6.5. Client scopes permissions	258
12.6.6. Realm default client scopes	258
12.6.7. Scopes explained	259
12.7. CLIENT POLICIES	259
12.7.1. Use-cases	259
12.7.2. Protocol	260
12.7.3. Architecture	260
12.7.3.1. Condition	260
12.7.3.2. Executor	261
12.7.3.3. Profile	262
12.7.3.4. Policy	262
12.7.4. Configuration	262
12.7.5. Backward Compatibility	262
12.7.6. Client Secret Rotation Example	263
CHAPTER 13. USING A VAULT TO OBTAIN SECRETS	264
13.1. KUBERNETES / OPENSIFT FILES PLAIN-TEXT VAULT PROVIDER	264
13.2. ELYTRON CREDENTIAL STORE VAULT PROVIDER	265
13.3. KEY RESOLVERS	266
13.4. SAMPLE CONFIGURATION	267
13.4.1. Configuring the credential store and vault without a mask	267
13.4.2. Masking the password in the credential store and vault	269
CHAPTER 14. CONFIGURING AUDITING TO TRACK EVENTS	271
14.1. LOGIN EVENTS	271
14.1.1. Event types	273
14.1.2. Event listener	274
14.1.2.1. The logging event listener	274
14.1.2.2. The Email Event Listener	276
14.2. ADMIN EVENTS	276
CHAPTER 15. IMPORTING AND EXPORTING THE DATABASE	280
15.1. ADMIN CONSOLE EXPORT/IMPORT	282
CHAPTER 16. MITIGATING SECURITY THREATS	283
16.1. HOST	283
16.2. ADMIN ENDPOINTS AND ADMIN CONSOLE	283
16.2.1. IP restriction	283
16.2.2. Port restriction	284
16.3. BRUTE FORCE ATTACKS	285
16.3.1. Password policies	287
16.4. READ-ONLY USER ATTRIBUTES	287
16.5. CLICKJACKING	289
16.6. SSL/HTTPS REQUIREMENT	290
16.7. CSRF ATTACKS	290
16.8. UNSPECIFIC REDIRECT URIS	290
16.9. FAPI COMPLIANCE	290
16.10. COMPROMISED ACCESS AND REFRESH TOKENS	291
16.11. COMPROMISED AUTHORIZATION CODE	291

16.12. OPEN REDIRECTORS	291
16.13. PASSWORD DATABASE COMPROMISED	291
16.14. LIMITING SCOPE	292
16.15. LIMIT TOKEN AUDIENCE	292
16.16. LIMIT AUTHENTICATION SESSIONS	292
16.17. SQL INJECTION ATTACKS	293
CHAPTER 17. ACCOUNT CONSOLE	294
17.1. ACCESSING THE ACCOUNT CONSOLE	294
17.2. CONFIGURING WAYS TO SIGN IN	294
17.2.1. Two-factor authentication with OTP	295
17.2.2. Two-factor authentication with WebAuthn	296
17.2.3. Passwordless authentication with WebAuthn	297
17.3. VIEWING DEVICE ACTIVITY	298
17.4. ADDING AN IDENTITY PROVIDER ACCOUNT	298
17.5. ACCESSING OTHER APPLICATIONS	299
CHAPTER 18. ADMIN CLI	300
18.1. INSTALLING THE ADMIN CLI	300
18.2. USING THE ADMIN CLI	300
18.3. AUTHENTICATING	301
18.4. WORKING WITH ALTERNATIVE CONFIGURATIONS	302
18.5. BASIC OPERATIONS AND RESOURCE URIS	302
18.6. REALM OPERATIONS	304
Creating a new realm	304
Listing existing realms	304
Getting a specific realm	305
Updating a realm	305
Deleting a realm	305
Turning on all login page options for the realm	305
Listing the realm keys	305
Generating new realm keys	306
Adding new realm keys from a Java Key Store file	306
Making the key passive or disabling the key	307
Deleting an old key	307
Configuring event logging for a realm	307
Flushing the caches	309
Importing a realm from exported .json file	309
18.7. ROLE OPERATIONS	309
Creating a realm role	309
Creating a client role	309
Listing realm roles	310
Listing client roles	310
Getting a specific realm role	310
Getting a specific client role	310
Updating a realm role	311
Updating a client role	311
Deleting a realm role	311
Deleting a client role	311
Listing assigned, available, and effective realm roles for a composite role	311
Listing assigned, available, and effective client roles for a composite role	312
Adding realm roles to a composite role	312
Removing realm roles from a composite role	312

Adding client roles to a realm role	312
Adding client roles to a client role	312
Removing client roles from a composite role	313
Adding client roles to a group	313
Removing client roles from a group	313
18.8. CLIENT OPERATIONS	313
Creating a client	313
Listing clients	314
Getting a specific client	314
Getting the current secret for a specific client	314
Generate a new secret for a specific client	314
Updating the current secret for a specific client	314
Getting an adapter configuration file (keycloak.json) for a specific client	314
Getting a WildFly subsystem adapter configuration for a specific client	315
Getting a Docker-v2 example configuration for a specific client	315
Updating a client	315
Deleting a client	315
Adding or removing roles for client's service account	315
18.9. USER OPERATIONS	315
Creating a user	315
Listing users	316
Getting a specific user	316
Updating a user	316
Deleting a user	316
Resetting a user's password	317
Listing assigned, available, and effective realm roles for a user	317
Listing assigned, available, and effective client roles for a user	317
Adding realm roles to a user	318
Removing realm roles from a user	318
Adding client roles to a user	318
Removing client roles from a user	318
Listing a user's sessions	318
Logging out a user from a specific session	319
Logging out a user from all sessions	319
18.10. GROUP OPERATIONS	319
Creating a group	319
Listing groups	319
Getting a specific group	319
Updating a group	319
Deleting a group	320
Creating a subgroup	320
Moving a group under another group	320
Get groups for a specific user	320
Adding a user to a group	320
Removing a user from a group	320
Listing assigned, available, and effective realm roles for a group	321
Listing assigned, available, and effective client roles for a group	321
18.11. IDENTITY PROVIDER OPERATIONS	321
Listing available identity providers	321
Listing configured identity providers	322
Getting a specific configured identity provider	322
Removing a specific configured identity provider	322
Configuring a Keycloak OpenID Connect identity provider	322

Configuring an OpenID Connect identity provider	322
Configuring a SAML 2 identity provider	322
Configuring a Facebook identity provider	323
Configuring a Google identity provider	323
Configuring a Twitter identity provider	323
Configuring a GitHub identity provider	323
Configuring a LinkedIn identity provider	324
Configuring a Microsoft Live identity provider	324
Configuring a Stack Overflow identity provider	324
18.12. STORAGE PROVIDER OPERATIONS	324
Configuring a Kerberos storage provider	324
Configuring an LDAP user storage provider	325
Removing a user storage provider instance	325
Triggering synchronization of all users for a specific user storage provider	326
Triggering synchronization of changed users for a specific user storage provider	326
Test LDAP user storage connectivity	326
Test LDAP user storage authentication	326
18.13. ADDING MAPPERS	326
Adding a hard-coded role LDAP mapper	327
Adding an MS Active Directory mapper	327
Adding a user attribute LDAP mapper	327
Adding a group LDAP mapper	328
Adding a full name LDAP mapper	328
18.14. AUTHENTICATION OPERATIONS	328
Setting a password policy	328
Obtaining the current password policy	329
Listing authentication flows	329
Getting a specific authentication flow	329
Listing executions for a flow	330
Adding configuration to an execution	330
Getting configuration for an execution	330
Updating configuration for an execution	330
Deleting configuration for an execution	331

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see [our CTO Chris Wright's message](#).

CHAPTER 1. RED HAT SINGLE SIGN-ON FEATURES AND CONCEPTS

Red Hat Single Sign-On is a single sign on solution for web apps and RESTful web services. The goal of Red Hat Single Sign-On is to make security simple so that it is easy for application developers to secure the apps and services they have deployed in their organization. Security features that developers normally have to write for themselves are provided out of the box and are easily tailorable to the individual requirements of your organization. Red Hat Single Sign-On provides customizable user interfaces for login, registration, administration, and account management. You can also use Red Hat Single Sign-On as an integration platform to hook it into existing LDAP and Active Directory servers. You can also delegate authentication to third party identity providers like Facebook and Google.

1.1. FEATURES

Red Hat Single Sign-On provides the following features:

- Single-Sign On and Single-Sign Out for browser applications.
- OpenID Connect support.
- OAuth 2.0 support.
- SAML support.
- Identity Brokering - Authenticate with external OpenID Connect or SAML Identity Providers.
- Social Login - Enable login with Google, GitHub, Facebook, Twitter, and other social networks.
- User Federation - Sync users from LDAP and Active Directory servers.
- Kerberos bridge - Automatically authenticate users that are logged-in to a Kerberos server.
- Admin Console for central management of users, roles, role mappings, clients and configuration.
- Account Management console that allows users to centrally manage their account.
- Theme support - Customize all user facing pages to integrate with your applications and branding.
- Two-factor Authentication - Support for TOTP/HOTP via Google Authenticator or FreeOTP.
- Login flows - optional user self-registration, recover password, verify email, require password update, etc.
- Session management - Admins and users themselves can view and manage user sessions.
- Token mappers - Map user attributes, roles, etc. how you want into tokens and statements.
- Not-before revocation policies per realm, application and user.
- CORS support - Client adapters have built-in support for CORS.
- Client adapters for JavaScript applications, JBoss EAP, etc.
- Supports any platform/language that has an OpenID Connect Relying Party library or SAML 2.0 Service Provider library.

1.2. BASIC RED HAT SINGLE SIGN-ON OPERATIONS

Red Hat Single Sign-On is a separate server that you manage on your network. Applications are configured to point to and be secured by this server. Red Hat Single Sign-On uses open protocol standards like [OpenID Connect](#) or [SAML 2.0](#) to secure your applications. Browser applications redirect a user's browser from the application to the Red Hat Single Sign-On authentication server where they enter their credentials. This redirection is important because users are completely isolated from applications and applications never see a user's credentials. Applications instead are given an identity token or assertion that is cryptographically signed. These tokens can have identity information like username, address, email, and other profile data. They can also hold permission data so that applications can make authorization decisions. These tokens can also be used to make secure invocations on REST-based services.

1.3. CORE CONCEPTS AND TERMS

Consider these core concepts and terms before attempting to use Red Hat Single Sign-On to secure your web applications and REST services.

users

Users are entities that are able to log into your system. They can have attributes associated with themselves like email, username, address, phone number, and birth day. They can be assigned group membership and have specific roles assigned to them.

authentication

The process of identifying and validating a user.

authorization

The process of granting access to a user.

credentials

Credentials are 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

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 rather than individual users as dealing with users can be too fine grained and hard to manage.

user role mapping

A user role mapping defines a mapping between a role and a user. A user can be associated with zero or more roles. This role mapping information can be encapsulated into tokens and assertions so that applications can decide access permissions on various resources they manage.

composite roles

A composite role is a role that can be associated with other roles. For example a **superuser** composite role could be associated with the **sales-admin** and **order-entry-admin** roles. If a user is mapped to the **superuser** role they also inherit the **sales-admin** and **order-entry-admin** roles.

groups

Groups manage groups of users. Attributes can be defined for a group. You can map roles to a group as well. Users that become members of a group inherit the attributes and role mappings that group defines.

realms

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

clients

Clients are 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. Clients can also be entities that just want to request identity information or an access token so that they can securely invoke other services on the network that are secured by Red Hat Single Sign-On.

client adapters

Client adapters are plugins that you install into your application environment to be able to communicate and be secured by Red Hat Single Sign-On. Red Hat Single Sign-On has a number of adapters for different platforms that you can download. There are also third-party adapters you can get for environments that we don't cover.

consent

Consent is when you as an admin want a user to give permission to a client before that client can participate in the authentication process. After a user provides their credentials, Red Hat Single Sign-On will pop up a screen identifying the client requesting a login and what identity information is requested of the user. User can decide whether or not to grant the request.

client scopes

When a client is registered, you must define protocol mappers and role scope mappings for that client. It is often useful to store a client scope, to make creating new clients easier by sharing some common settings. This is also useful for requesting some claims or roles to be conditionally based on the value of **scope** parameter. Red Hat Single Sign-On provides the concept of a client scope for this.

client role

Clients can define roles that are specific to them. This is basically a role namespace dedicated to the client.

identity token

A token that provides identity information about the user. Part of the OpenID Connect specification.

access token

A token that can be provided as part of an HTTP request that grants access to the service being invoked on. This is part of the OpenID Connect and OAuth 2.0 specification.

assertion

Information about a user. This usually pertains to an XML blob that is included in a SAML authentication response that provided identity metadata about an authenticated user.

service account

Each client has a built-in service account which allows it to obtain an access token.

direct grant

A way for a client to obtain an access token on behalf of a user via a REST invocation.

protocol mappers

For each client you can tailor what claims and assertions are stored in the OIDC token or SAML assertion. You do this per client by creating and configuring protocol mappers.

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.

user federation provider

Red Hat Single Sign-On can store and manage users. Often, companies already have LDAP or Active Directory services that store user and credential information. You can point Red Hat Single Sign-On to validate credentials from those external stores and pull in identity information.

identity provider

An identity provider (IDP) is a service that can authenticate a user. Red Hat Single Sign-On is an IDP.

identity provider federation

Red Hat Single Sign-On can be configured to delegate authentication 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

When doing IDP federation you can map incoming tokens and assertions to user and session attributes. This helps you propagate identity information from the external IDP to your client requesting authentication.

required actions

Required actions are 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

Authentication flows are 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.

events

Events are audit streams that admins can view and hook into.

themes

Every screen provided by Red Hat Single Sign-On is backed by a theme. Themes define HTML templates and stylesheets which you can override as needed.

CHAPTER 2. CREATING THE FIRST ADMINISTRATOR

After installing Red Hat Single Sign-On, you need an administrator account that can act as a *super* admin with full permissions to manage all parts of Red Hat Single Sign-On. With this account, you can log into the Red Hat Single Sign-On Admin Console where you create realms and users and register applications that are secured by Red Hat Single Sign-On.

Prerequisites

- Perform the installation and configuration tasks defined in the [Server Installation and Configuration Guide](#) to the point that the Red Hat Single Sign-On server is running.

2.1. CREATING THE ACCOUNT ON THE LOCAL HOST

If your server is accessible from **localhost**, perform these steps.

Procedure

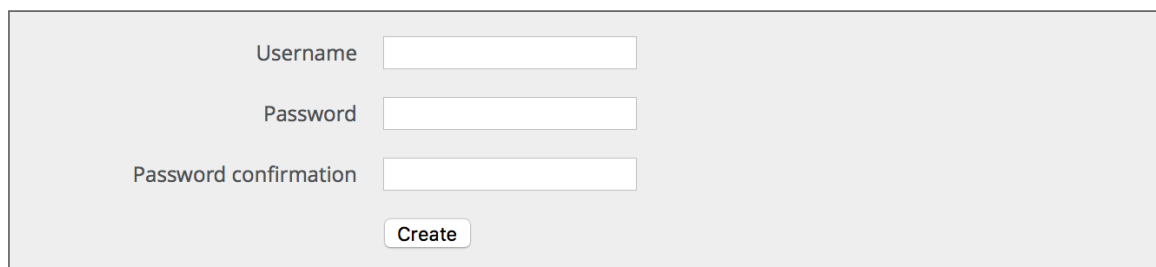
1. In a web browser, go to the <http://localhost:8080/auth> URL.
2. Supply a username and password that you can recall.

Welcome page

Welcome to Red Hat Single Sign-On

Your Red Hat Single Sign-On is running.

Please create an initial admin user to get started.



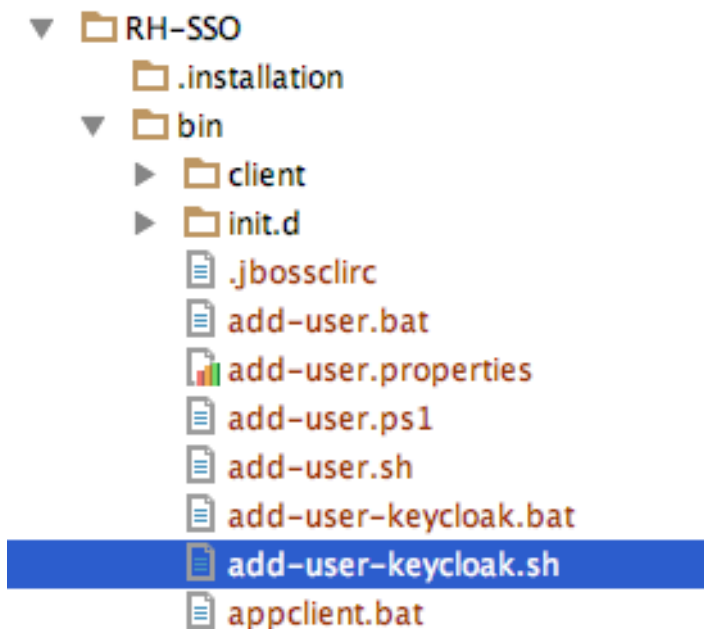
The screenshot shows a web form for creating an initial admin user. It has three input fields: 'Username', 'Password', and 'Password confirmation'. Below the fields is a 'Create' button. The form is set against a light gray background.

[Administration Console](#) | [Documentation](#)

2.2. CREATING THE ACCOUNT REMOTELY

If you cannot access the server from a **localhost** address, or just want to start Red Hat Single Sign-On from the command line, use the `.../bin/add-user-keycloak` script.

Add-user-keycloak script



The parameters are a little different depending if you are using the standalone operation mode or domain operation mode. For standalone mode, here is how you use the script.

Linux/Unix

```
$ ../bin/add-user-keycloak.sh -r master -u <username> -p <password>
```

Windows

```
> ...bin\add-user-keycloak.bat -r master -u <username> -p <password>
```

The generated file is owned by a different user than the Red Hat Single Sign-On running user. Use this command to set the permissions so the Red Hat Single Sign-On user can read the file upon restarting the server.

```
chgrp jboss /opt/rh/rh-sso7/root/usr/share/keycloak/standalone/configuration/keycloak-add-user.json
```

For domain mode, you have to point the script to one of your server hosts using the **-sc** switch.

Linux/Unix

```
$ ../bin/add-user-keycloak.sh --sc domain/servers/server-one/configuration -r master -u <username> -p <password>
```

Windows

```
> ...bin\add-user-keycloak.bat --sc domain/servers/server-one/configuration -r master -u <username> -p <password>
```

CHAPTER 3. CONFIGURING REALMS

Once you have an administrative account for the Admin Console, you can configure realms. A realm is a space where you manage objects, including users, applications, roles, and groups. A user belongs to and logs into a realm. One Red Hat Single Sign-On deployment can define, store, and manage as many realms as there is space for in the database.

3.1. USING THE ADMIN CONSOLE

You configure realms and perform most administrative tasks in the Red Hat Single Sign-On Admin Console.

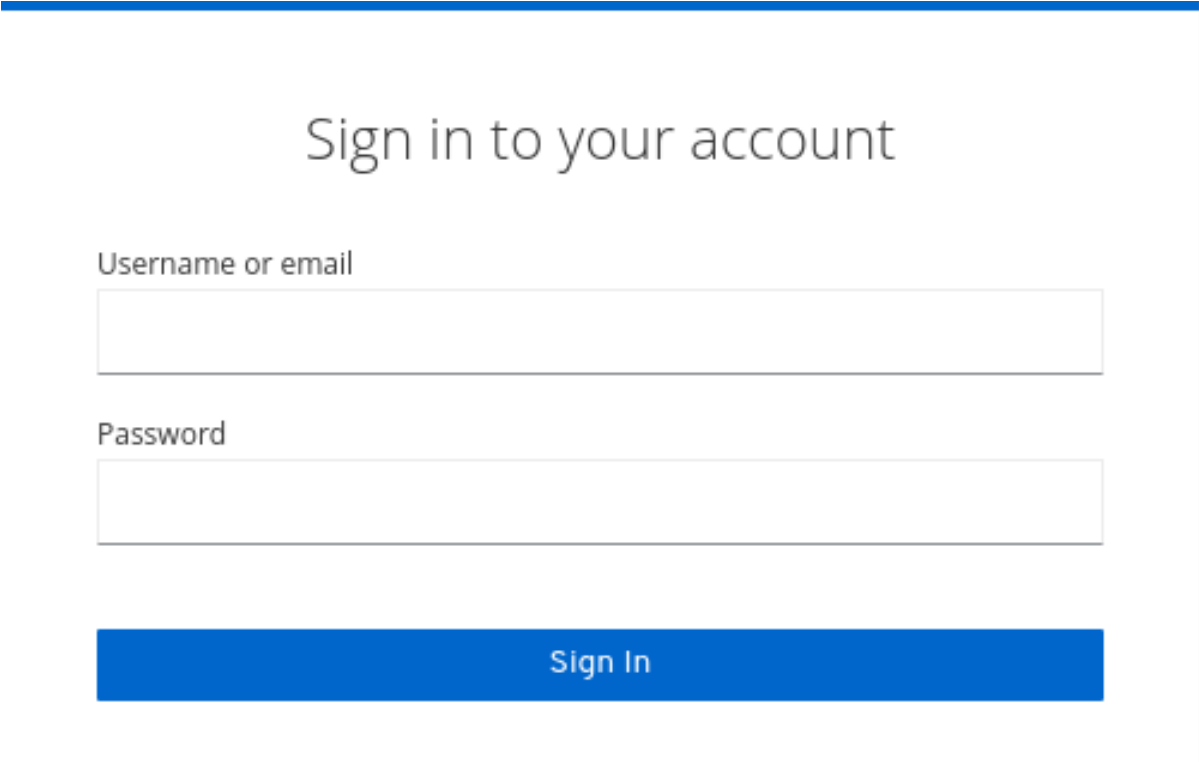
Prerequisites

- You need an administrator account. See [Creating the first administrator](#).

Procedure

1. Go to the URL for the Admin Console.
For example, for localhost, use this URL: <http://localhost:8080/auth/admin/>

Login page



Sign in to your account

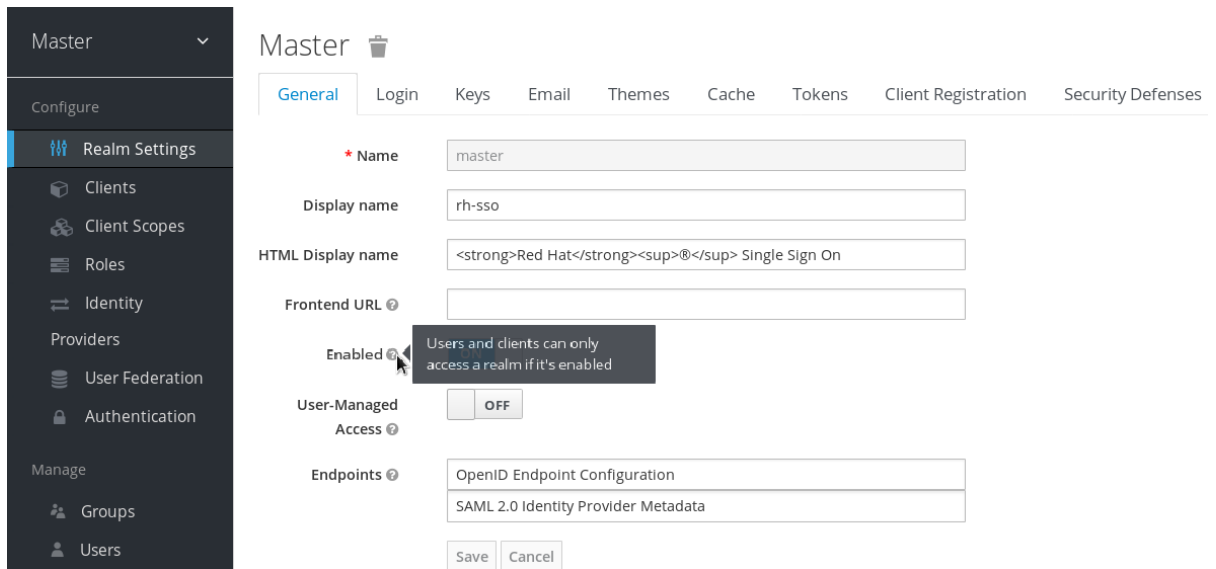
Username or email

Password

Sign In

2. Enter the username and password you created on the Welcome Page or the **add-user-keycloak** script in the bin directory. This action displays the Admin Console.

Admin Console



3. Note the menus and other options that you can use:

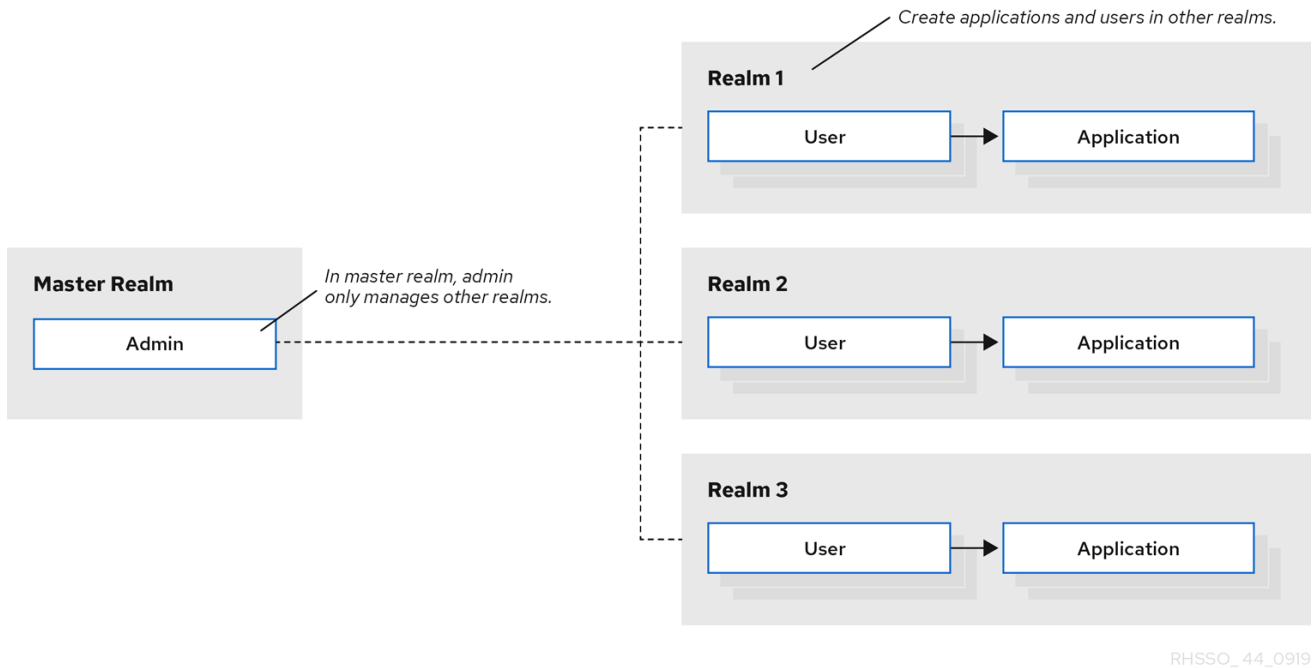
- Click the menu labeled **Master** to pick a realm you want to manage or to create a new one.
- Click the top right list to view your account or log out.
- Hover over a question mark ? icon to show a tooltip text that describes that field. The image above shows the tooltip in action.

3.2. THE MASTER REALM

In the Admin Console, two types of realms exist:

- **Master realm** - This realm was created for you when you first started Red Hat Single Sign-On. It contains the administrator account you created at the first login. Use the *master* realm only to create and manage the realms in your system.
- **Other realms** - These realms are created by the administrator in the master realm. In these realms, administrators manage the users in your organization and the applications they need. The applications are owned by the users.

Realms and applications



Realms are isolated from one another and can only manage and authenticate the users that they control. Following this security model helps prevent accidental changes and follows the tradition of permitting user accounts access to only those privileges and powers necessary for the successful completion of their current task.

Additional resources

- See [Dedicated Realm Admin Consoles](#) if you want to disable the *master* realm and define administrator accounts within any new realm you create. Each realm has its own dedicated Admin Console that you can log into with local accounts.

3.3. CREATING A REALM

You create a realm to provide a management space where you can create users and give them permissions to use applications. At first login, you are typically in the *master* realm, the top-level realm from which you create other realms.

When deciding what realms you need, consider the kind of isolation you want to have for your users and applications. For example, you might create a realm for the employees of your company and a separate realm for your customers. Your employees would log into the employee realm and only be able to visit internal company applications. Customers would log into the customer realm and only be able to interact with customer-facing apps.

Procedure

1. Point to the top of the left pane.
2. Click **Add Realm**.

Add realm menu

The screenshot shows the 'Master' realm configuration page. On the left is a dark sidebar with a 'Master' dropdown menu and an 'Add realm' button. Below that is a 'Realm Settings' menu with options: Clients, Client Scopes, Roles, Identity, Providers, User Federation, and Authentication. Under 'Manage' are Groups, Users, Sessions, Events, and Import. The main content area is titled 'Master' and has tabs for General, Login, Keys, Email, Themes, Cache, Tokens, Client Registration, and Security Defenses. The 'General' tab is active, showing fields for Name (master), Display name (rh-sso), HTML Display name (Red Hat[®] Single Sign On), Frontend URL, Enabled (ON), User-Managed Access (OFF), and Endpoints (OpenID Endpoint Configuration, SAML 2.0 Identity Provider Metadata). There are 'Save' and 'Cancel' buttons at the bottom.

3. Enter a name for the realm.
4. Click **Create**.

Create realm

The screenshot shows the 'Add realm' configuration page. On the left is a dark sidebar with a 'Select realm' dropdown menu. The main content area is titled 'Add realm' and has an 'Import' section with a 'Select file' button. Below that is a 'Name' field with a red asterisk, and an 'Enabled' toggle switch set to 'ON'. There are 'Create' and 'Cancel' buttons at the bottom.

The current realm is now set to the realm you just created. You can switch between managing different realms by pointing to the top left corner to click **Select Realm**.

3.4. CONFIGURING SSL FOR A REALM

Each realm has an associated SSL Mode, which defines the SSL/HTTPS requirements for interacting with the realm. Browsers and applications that interact with the realm honor the SSL/HTTPS requirements defined by the SSL Mode or they cannot interact with the server.



WARNING

Red Hat Single Sign-On generates a self-signed certificate the first time it runs. Please note that self-signed certificates are not secure, and should only be used for testing purposes. It is highly recommended that you install a CA-signed certificate on the Red Hat Single Sign-On server itself or on a reverse proxy in front of the Red Hat Single Sign-On server. See the [Server Installation and Configuration Guide](#).

Procedure

1. Click **Realm Settings** in the menu.
2. Click the **Login** tab.

Login tab

The screenshot displays the Admin Console interface. On the left, a dark sidebar contains a navigation menu with 'Configure' and 'Manage' sections. 'Configure' includes 'Realm Settings' (highlighted), 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. 'Manage' includes 'Groups', 'Users', and 'Sessions'. The main content area is titled 'Master' and has a trash icon. Below the title are tabs for 'General', 'Login' (selected), 'Keys', 'Email', 'Themes', 'Cache', 'Tokens', 'Client Registration', and 'Security Defenses'. The 'Login' tab contains the following settings:

- User registration: ON
- Email as username: OFF
- Edit username: OFF
- Forgot password: ON
- Remember Me: OFF
- Verify email: OFF
- Login with email: ON
- Require SSL: external requests (dropdown menu)

At the bottom of the settings are 'Save' and 'Cancel' buttons.

3. Set **Require SSL** to one of the following SSL modes:

- external requests:: Users can interact with Red Hat Single Sign-On without SSL so long as they stick to private IP addresses such as **localhost**, **127.0.0.1**, **10.x.x.x**, **192.168.x.x**, and **172.16.x.x**. If you try to access Red Hat Single Sign-On without SSL from a non-private IP address, you will get an error.
- none:: Red Hat Single Sign-On does not require SSL. This choice applies only in development when you are experimenting and do not plan to support this deployment.
- all requests:: Red Hat Single Sign-On requires SSL for all IP addresses.

3.5. CLEARING SERVER CACHES

Red Hat Single Sign-On caches everything it can in memory within the limits of your JVM and the limits you have configured. If the Red Hat Single Sign-On database is modified by a third party, such as a DBA, outside the scope of the server's REST APIs or Admin Console, parts of the in-memory cache could be stale. You can clear the realm cache, user cache or cache of external public keys, such as Public keys of external clients or Identity providers, which Red Hat Single Sign-On may use to verify signatures of particular external entity.

Procedure

1. Click **Realm Settings** in the menu.
2. Click the **Cache** tab.
3. Click **Clear** for the cache you want to evict.

Cache tab

The screenshot shows the 'Master' configuration page for the 'Cache' tab. The left sidebar is open to 'Realm Settings'. The main content area has tabs for 'General', 'Login', 'Keys', 'Email', 'Themes', 'Cache', 'Tokens', 'Client Registration', and 'Security Defenses'. Under the 'Cache' tab, there are three sections: 'Realm Cache', 'User Cache', and 'Keys Cache', each with a 'Clear' button.

3.6. CONFIGURING EMAIL FOR A REALM

Red Hat Single Sign-On sends emails to users to verify their email addresses, when they forget their passwords, or when an administrator needs to receive notifications about a server event. To enable Red Hat Single Sign-On to send emails, you provide Red Hat Single Sign-On with your SMTP server settings.

Procedure

1. Click **Realm Settings** in the menu.
2. Click the **Email** tab.

Email tab

The screenshot shows the 'Master' configuration page for the 'Email' tab. The left sidebar is open to 'Realm Settings'. The main content area has tabs for 'General', 'Login', 'Keys', 'Email', 'Themes', 'Cache', 'Tokens', 'Client Registration', and 'Security Defenses'. Under the 'Email' tab, there are several configuration fields:

- Host**: SMTP Host (with a 'Test connection' button)
- Port**: SMTP Port (defaults to 25)
- From Display Name**: Display Name for Sender Email Address
- From**: Sender Email Address
- Reply To Display Name**: Display Name for Reply To Email Address
- Reply To**: Reply To Email Address
- Envelope From**: Sender Envelope Email Address
- Enable SSL**: OFF
- Enable StartTLS**: OFF
- Enable Authentication**: OFF

3. Fill in the fields and toggle the switches as needed.

Host

Host denotes the SMTP server hostname used for sending emails.

Port

Port denotes the SMTP server port.

From

From denotes the address used for the **From** SMTP-Header for the emails sent.

From Display Name

From Display Name allows to configure a user friendly email address aliases (optional). If not set the plain **From** email address will be displayed in email clients.

Reply To

Reply To denotes the address used for the **Reply-To** SMTP-Header for the mails sent (optional). If not set the plain **From** email address will be used.

Reply To Display Name

Reply To Display Name allows to configure a user friendly email address aliases (optional). If not set the plain **Reply To** email address will be displayed.

Envelope From

Envelope From denotes the [Bounce Address](#) used for the **Return-Path** SMTP-Header for the mails sent (optional).

Enable SSL and Enable StartTLS

Toggle one of these switches to **ON** to support sending emails for recovering usernames and passwords, especially if the SMTP server is on an external network. You will most likely need to change the **Port** to 465, the default port for SSL/TLS.

Enable Authentication

Set this switch to **ON** if your SMTP server requires authentication. When prompted, supply the **Username** and **Password**. The value of the **Password** field can refer a value from an external [vault](#).

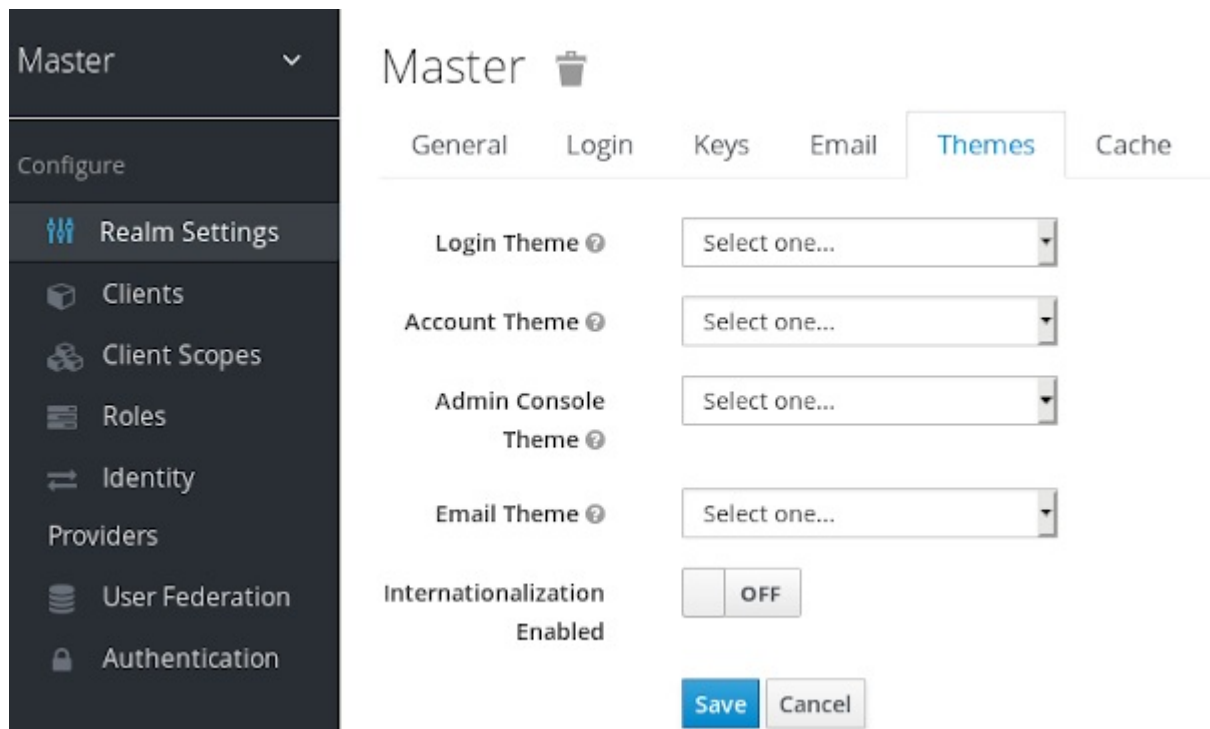
3.7. CONFIGURING THEMES AND INTERNATIONALIZATION

For a given realm, you can change the appearance of any UI, including the language that appears, in Red Hat Single Sign-On by using themes.

Procedure

1. Click **Realm Setting** in the menu.
2. Click the **Themes** tab.

Themes tab



- Pick the theme you want for each UI category and click **Save**.

Login Theme

Username password entry, OTP entry, new user registration, and other similar screens related to login.

Account Theme

Each user has an User Account Management UI.

Admin Console Theme

The skin of the Red Hat Single Sign-On Admin Console.

Email Theme

Whenever Red Hat Single Sign-On has to send out an email, it uses templates defined in this theme to craft the email.

Additional resources

- The [Server Developer Guide](#) describes how to create a new theme or modify existing ones.

3.7.1. Enabling internationalization

Every UI screen is internationalized in Red Hat Single Sign-On. The default language is English, but you can choose which locales you want to support and what the default locale will be.

Procedure

- Click **Realm Settings** in the menu.
- Click the **Theme** tab.
- Set **Internationalization** to **ON**.

The next time a user logs in, that user can choose a language on the login page to use for the login screens, Account Console, and Admin Console.

Additional resources

- The [Server Developer Guide](#) explains how you can offer additional languages. All internationalized texts which are provided by the theme can be overwritten by realm-specific texts on the **Localization** tab.

3.7.2. User locale selection

A locale selector provider suggests the best locale on the information available. However, it is often unknown who the user is. For this reason, the previously authenticated user's locale is remembered in a persisted cookie.

The logic for selecting the locale uses the first of the following that is available:

- User selected - when the user has selected a locale using the drop-down locale selector
- User profile - when there is an authenticated user and the user has a preferred locale set
- Client selected - passed by the client using for example `ui_locales` parameter
- Cookie - last locale selected on the browser
- Accepted language - locale from **Accept-Language** header
- Realm default
- If none of the above, fall back to English

When a user is authenticated an action is triggered to update the locale in the persisted cookie mentioned earlier. If the user has actively switched the locale through the locale selector on the login pages the users locale is also updated at this point.

If you want to change the logic for selecting the locale, you have an option to create custom **LocaleSelectorProvider**. For details, please refer to the [Server Developer Guide](#).

3.8. CONTROLLING LOGIN OPTIONS

Red Hat Single Sign-On includes several built-in login page features.

3.8.1. Enabling forgot password

If you enable **Forgot Password**, users can reset their login credentials if they forget their passwords or lose their OTP generator.

Procedure

1. Click **Realm Settings** in the menu.
2. Click the **Login** tab.

Login tab

Master

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

Configure

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

Manage

- Groups
- Users
- Sessions

User registration ON

Email as username OFF

Edit username OFF

Forgot password ON

Remember Me OFF

Verify email OFF

Login with email ON

Require SSL

Save Cancel

3. Toggle **Forgot Password** to **ON**.
A **forgot password** link displays in your login pages.

Forgot password link

Sign in to your account

Username or email

Password

[Forgot Password?](#)

Sign In

New user? [Register](#)

4. Click this link to bring users where they can enter their username or email address and receive an email with a link to reset their credentials.

Forgot password page

Forgot Your Password?

Username or email

[« Back to Login](#)

Submit

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

The text sent in the email is configurable. See [Server Developer Guide](#) for more information.

When users click the email link, Red Hat Single Sign-On asks them to update their password, and if they have set up an OTP generator, Red Hat Single Sign-On asks them to reconfigure the OTP generator. Depending on security requirements of your organization, you may not want users to reset their OTP generator through email.

To change this behavior, perform these steps:

Procedure

1. Click **Authentication** in the menu.
2. Click the **Flows** tab.
3. Select the **Reset Credentials** flow.

Reset credentials flow

The screenshot shows the 'Authentication' configuration page in the Red Hat Single Sign-On Administration Console. The 'Flows' tab is selected, and the 'Reset Credentials' flow is chosen. The table below shows the requirements for this flow.

Auth Type	Requirement	REQUIRED	ALTERNATIVE	DISABLED	CONDITIONAL
Choose User		<input checked="" type="radio"/>			
Send Reset Email		<input checked="" type="radio"/>			
Reset Password		<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	
Reset - Conditional OTP		<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
	Condition - User Configured	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	
	Reset OTP	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	

If you do not want to reset the OTP, set the **Reset OTP** requirement to **Disabled**.

4. Click the **Required Actions** tab. Ensure *Update Password* is enabled.

3.8.2. Enabling Remember Me

A logged-in user closing their browser destroys their session, and that user must log in again. You can set Red Hat Single Sign-On to keep the user's login session open if that user clicks the *Remember Me* checkbox upon login. This action turns the login cookie from a session-only cookie to a persistence cookie.

Procedure

1. Click **Realm Settings** in the menu.
2. Click the **Login** tab.
3. Toggle the **Remember Me** switch to **ON**.

Login tab

The screenshot displays the Red Hat Single Sign-On administration console. On the left is a dark sidebar menu with a 'Master' dropdown at the top. Under 'Configure', 'Realm Settings' is selected. Under 'Manage', 'Groups', 'Users', and 'Sessions' are listed. The main content area shows the 'Master' configuration page with a trash icon. The 'Login' tab is active, showing various settings: 'User registration' (ON), 'Email as username' (OFF), 'Edit username' (OFF), 'Forgot password' (ON), 'Remember Me' (OFF), 'Verify email' (OFF), 'Login with email' (ON), and 'Require SSL' (external requests). 'Save' and 'Cancel' buttons are at the bottom.

When you save this setting, a **remember me** checkbox displays on the realm's login page.

Remember Me

Sign in to your account

Username or email

Password

Remember me

Sign In

New user? [Register](#)

3.8.3. ACR to Level of Authentication (LoA) Mapping

In the login settings of a realm, you can define which **Authentication Context Class Reference (ACR)** value is mapped to which **Level of Authentication (LoA)**. The ACR can be any value, whereas the LoA must be numeric. The acr claim can be requested in the **claims** or **acr_values** parameter sent in the OIDC request and it is also included in the access token and ID token. The mapped number is used in the authentication flow conditions.

Mapping can be also specified at the client level in case that particular client needs to use different values than realm. However, a best practice is to stick to realm mappings.

ACR to LoA Mapping ?

silver	1	-
gold	2	-
ACR	LOA	+

[Save](#) [Cancel](#)

For further details see [Step-up Authentication](#) and [the official OIDC specification](#).

3.8.3.1. Update Email Workflow (UpdateEmail)

With this workflow, users will have to use an UPDATE_EMAIL action to change their own email address.

The action is associated with a single email input form. If the realm has email verification disabled, this action will allow to update the email without verification. If the realm has email verification enabled, the

action will send an email update action token to the new email address without changing the account email. Only the action token triggering will complete the email update.

Applications are able to send their users to the email update form by leveraging UPDATE_EMAIL as an AIA (Application Initiated Action).



NOTE

UpdateEmail is **Technology Preview** and is not fully supported. This feature is disabled by default.

To enable start the server with **-Dkeycloak.profile=preview** or **-Dkeycloak.profile.feature.update_email=enabled**. For more details see [Profiles](#).



NOTE

If you enable this feature and you are migrating from a previous version, enable the **Update Email** required action in your realms. Otherwise, users cannot update their email addresses.

3.9. CONFIGURING REALM KEYS

The authentication protocols that are used by Red Hat Single Sign-On require cryptographic signatures and sometimes encryption. Red Hat Single Sign-On uses asymmetric key pairs, a private and public key, to accomplish this.

Red Hat Single Sign-On has a single active keypair at a time, but can have several passive keys as well. The active keypair is used to create new signatures, while the passive keypairs can be used to verify previous signatures. This makes it possible to regularly rotate the keys without any downtime or interruption to users.

When a realm is created, a keypair and a self-signed certificate is automatically generated.

Procedure

1. Select the realm in the Admin Console.
2. Click **Realm settings**.
3. Click **Keys**.
4. Click **Passive** to view passive keys.
5. Click **Disabled** to view disabled keys.

A keypair can have the status **Active**, but still not be selected as the currently active keypair for the realm. The selected active pair which is used for signatures is selected based on the first key provider sorted by priority that is able to provide an active keypair.

3.9.1. Rotating keys

We recommend that you regularly rotate keys. Start by creating new keys with a higher priority than the existing active keys. You can instead create new keys with the same priority and making the previous keys passive.

Once new keys are available, all new tokens and cookies are signed with the new keys. When a user authenticates to an application, the SSO cookie is updated with the new signature. When OpenID Connect tokens are refreshed, new tokens are signed with the new keys. Eventually, all cookies and tokens use the new keys and after a while the old keys can be removed.

The frequency of deleting old keys is a tradeoff between security and making sure all cookies and tokens are updated. Consider creating new keys every three to six months and deleting old keys one to two months after you create the new keys. If a user was inactive in the period between the new keys being added and the old keys being removed, that user will have to re-authenticate.

Rotating keys also applies to offline tokens. To make sure they are updated, the applications need to refresh the tokens before the old keys are removed.

3.9.2. Adding a generated keypair

Use this procedure to generate a keypair including a self-signed certificate.

Procedure

1. Select the realm in the Admin Console.
2. Click **Realm settings**.
3. Click the **Keys** tab.
4. Click the **Providers** tab.
5. Click **Add keystore** and select **rsa-generated**.
6. Enter a number in the **Priority** field. This number determines if the new key pair becomes the active keypair. The highest number makes the keypair active.
7. Select a value for **keysize**.
8. Click **Save**.

Changing the priority for a provider will not cause the keys to be re-generated, but if you want to change the keysize you can edit the provider and new keys will be generated.

3.9.3. Rotating keys by extracting a certificate

You can rotate keys by extracting a certificate from an RSA generated keypair and using that certificate in a new keystore.

Prerequisites

- A generated keypair

Procedure

1. Select the realm in the Admin Console.
2. Click **Realm Settings**.
3. Click the **Keys** tab.
A list of **Active** keys appears.

4. On a row with an RSA key, click **Certificate** under **Public Keys**.
The certificate appears in text form.
5. Save the certificate to a file and enclose it in these lines.

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

6. Use the **keytool** command to convert the key file to PEM Format.
7. Remove the current RSA public key certificate from the keystore.

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

8. Import the new certificate into the keystore

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

9. Undeploy and rebuild the application.

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

3.9.4. Adding an existing keypair and certificate

To add a keypair and certificate obtained elsewhere select **Providers** and choose **rsa** from the dropdown. You can change the priority to make sure the new keypair becomes the active keypair.

Prerequisites

- A private key file. The file must be PEM formatted.

Procedure

1. Select the realm in the Admin Console.
2. Click **Realm settings**.
3. Click the **Keys** tab.
4. Click the **Providers** tab.
5. Click **Add keystore** and select **rsa**.
6. Enter a number in the **Priority** field. This number determines if the new key pair becomes the active key pair.
7. Click **Select file** beside **Private RSA Key** to upload the private key file.
8. If you have a signed certificate for your private key, click **Select file** beside **X509 Certificate** to upload the certificate file. Red Hat Single Sign-On automatically generates a self-signed certificate if you do not upload a certificate.

9. Click **Save**.

3.9.5. Loading keys from a Java Keystore

To add a keypair and certificate stored in a Java Keystore file on the host select **Providers** and choose **java-keystore** from the dropdown. You can change the priority to make sure the new keypair becomes the active keypair.

For the associated certificate chain to be loaded it must be imported to the Java Keystore file with the same **Key Alias** used to load the keypair.

Procedure

1. Select the realm in the Admin Console.
2. Click **Realm settings**.
3. Click the **Keys** tab.
4. Click the **Providers** tab.
5. Click **Add keystore** and select **java-keystore**.
6. Enter a number in the **Priority** field. This number determines if the new key pair becomes the active key pair.
7. Enter a value for **Keystore**.
8. Enter a value for **Keystore Password**.
9. Enter a value for **Key Alias**.
10. Enter a value for **Key Password**.
11. Click **Save**.

3.9.6. Making keys passive

Procedure

1. Select the realm in the Admin Console.
2. Click Realm settings.
3. Click the **Keys** tab.
4. Click the **Active** tab.
5. Click the provider of the key you want to make passive.
6. Toggle **Active** to **OFF**.
7. Click **Save**.

3.9.7. Disabling keys

Procedure

1. Select the realm in the Admin Console.
2. Click Realm settings.
3. Click the **Keys** tab.
4. Click the **Active** tab.
5. Click the provider of the key you want to make passive.
6. Toggle **Enabled** to **OFF**.
7. Click **Save**.

3.9.8. Compromised keys

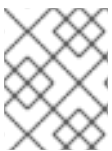
Red Hat Single Sign-On has the signing keys stored just locally and they are never shared with the client applications, users or other entities. However, if you think that your realm signing key was compromised, you should first generate new keypair as described above and then immediately remove the compromised keypair.

Alternatively, you can delete the provider from the **Providers** table.

Procedure

1. Click **Clients** in the menu.
2. Click **security-admin-console**.
3. Click the **Revocation** tab.
4. Click **Set to now**.
5. Click **Push**.

Pushing the not-before policy ensures that client applications do not accept the existing tokens signed by the compromised key. The client application is forced to download new key pairs from Red Hat Single Sign-On also so the tokens signed by the compromised key will be invalid.



NOTE

REST and confidential clients must set **Admin URL** so Red Hat Single Sign-On can send clients the pushed not-before policy request.

CHAPTER 4. USING EXTERNAL STORAGE

Organizations can have databases containing information, passwords, and other credentials. Typically, you cannot migrate existing data storage to a Red Hat Single Sign-On deployment so Red Hat Single Sign-On can federate existing external user databases. Red Hat Single Sign-On supports LDAP and Active Directory, but you can also code extensions for any custom user database by using the Red Hat Single Sign-On User Storage SPI.

When a user attempts to log in, Red Hat Single Sign-On examines that user's storage to find that user. If Red Hat Single Sign-On does not find the user, Red Hat Single Sign-On iterates over each User Storage provider for the realm until it finds a match. Data from the external data storage then maps into a standard user model the Red Hat Single Sign-On runtime consumes. This user model then maps to OIDC token claims and SAML assertion attributes.

External user databases rarely have the data necessary to support all the features of Red Hat Single Sign-On, so the User Storage Provider can opt to store items locally in Red Hat Single Sign-On user data storage. Providers can import users locally and sync periodically with external data storage. This approach depends on the capabilities of the provider and the configuration of the provider. For example, your external user data storage may not support OTP. The OTP can be handled and stored by Red Hat Single Sign-On, depending on the provider.

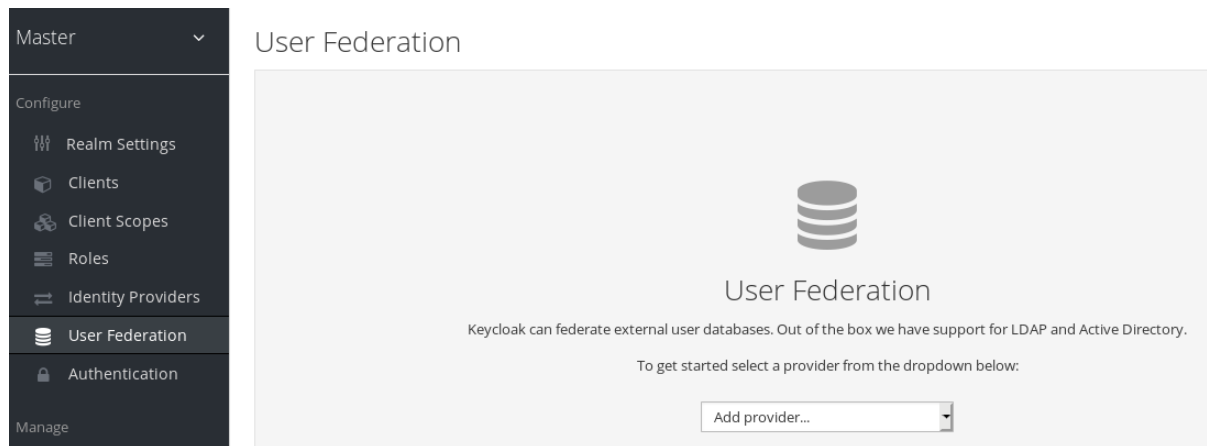
4.1. ADDING A PROVIDER

To add a storage provider, perform the following procedure:

Procedure

1. Click **User Federation** in the menu.

User federation



2. Select the provider type from the **Add Provider** list. Red Hat Single Sign-On brings you to that provider's configuration page.

4.2. DEALING WITH PROVIDER FAILURES

If a User Storage Provider fails, you may not be able to log in and view users in the Admin Console. Red Hat Single Sign-On does not detect failures when using a Storage Provider to look up a user, so it cancels the invocation. If you have a Storage Provider with a high priority that fails during user lookup, the login or user query fails with an exception and will not fail over to the next configured provider.

Red Hat Single Sign-On searches the local Red Hat Single Sign-On user database first to resolve users before any LDAP or custom User Storage Provider. Consider creating an administrator account stored in the local Red Hat Single Sign-On user database in case of problems connecting to your LDAP and back ends.

Each LDAP and custom User Storage Provider has an **enable** toggle on its Admin Console page. Disabling the User Storage Provider skips the provider when performing queries, so you can view and log in with user accounts in a different provider with lower priority. If your provider uses an **import** strategy and is disabled, imported users are still available for lookup in read-only mode.

When a Storage Provider lookup fails, Red Hat Single Sign-On does not fail over because user databases often have duplicate usernames or duplicate emails between them. Duplicate usernames and emails can cause problems because the user loads from one external data store when the admin expects them to load from another data store.

4.3. LIGHTWEIGHT DIRECTORY ACCESS PROTOCOL (LDAP) AND ACTIVE DIRECTORY

Red Hat Single Sign-On includes an LDAP/AD provider. You can federate multiple different LDAP servers in one Red Hat Single Sign-On realm and map LDAP user attributes into the Red Hat Single Sign-On common user model.

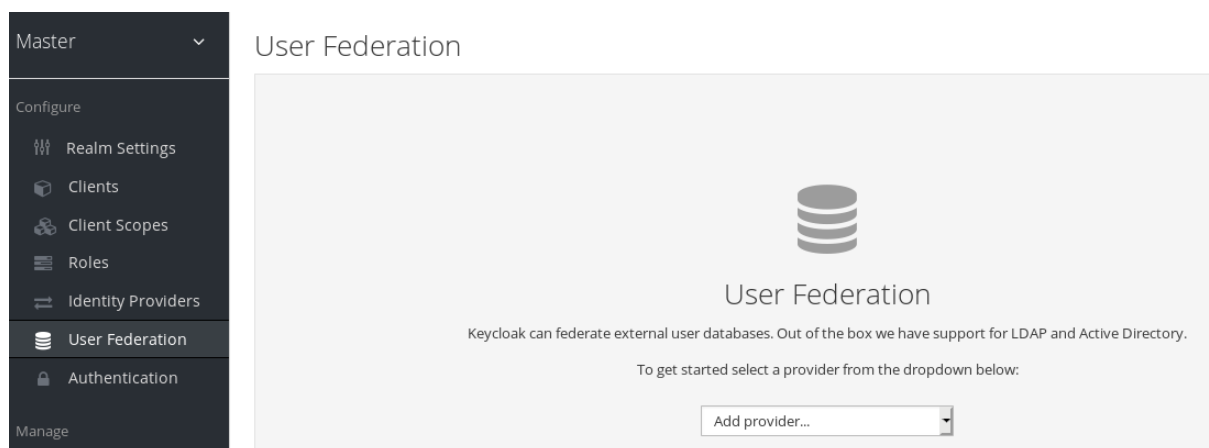
By default, Red Hat Single Sign-On maps the username, email, first name, and last name of the user account, but you can also configure additional [mappings](#). Red Hat Single Sign-On's LDAP/AD provider supports password validation using LDAP/AD protocols and storage, edit, and synchronization modes.

4.3.1. Configuring federated LDAP storage

Procedure

1. Click **User Federation** in the menu.

User federation



2. Select *ldap* from the **Add Provider** list. Red Hat Single Sign-On brings you to the LDAP configuration page.

4.3.2. Storage mode

Red Hat Single Sign-On imports users from LDAP into the local Red Hat Single Sign-On user database. This copy of the user database synchronizes on-demand or through a periodic background task. An

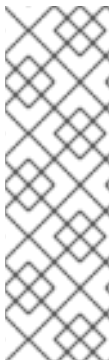
exception exists for synchronizing passwords. Red Hat Single Sign-On never imports passwords. Password validation always occurs on the LDAP server.

The advantage of synchronization is that all Red Hat Single Sign-On features work efficiently because any required extra per-user data is stored locally. The disadvantage is that each time Red Hat Single Sign-On queries a specific user for the first time, Red Hat Single Sign-On performs a corresponding database insert.

You can synchronize the import with your LDAP server. Import synchronization is unnecessary when LDAP mappers always read particular attributes from the LDAP rather than the database.

You can use LDAP with Red Hat Single Sign-On without importing users into the Red Hat Single Sign-On user database. The LDAP server backs up the common user model that the Red Hat Single Sign-On runtime uses. If LDAP does not support data that a Red Hat Single Sign-On feature requires, that feature will not work. The advantage of this approach is that you do not have the resource usage of importing and synchronizing copies of LDAP users into the Red Hat Single Sign-On user database.

The **Import Users** switch on the LDAP configuration page controls this storage mode. To import users, toggle this switch to **ON**.



NOTE

If you disable **Import Users**, you cannot save user profile attributes into the Red Hat Single Sign-On database. Also, you cannot save metadata except for user profile metadata mapped to the LDAP. This metadata can include role mappings, group mappings, and other metadata based on the LDAP mappers' configuration.

When you attempt to change the non-LDAP mapped user data, the user update is not possible. For example, you cannot disable the LDAP mapped user unless the user's **enabled** flag maps to an LDAP attribute.

4.3.3. Edit mode

Users and admins can modify user metadata, users through the [Account Console](#), and administrators through the Admin Console. The **Edit Mode** configuration on the LDAP configuration page defines the user's LDAP update privileges.

READONLY

You cannot change the username, email, first name, last name, and other mapped attributes. Red Hat Single Sign-On shows an error anytime a user attempts to update these fields. Password updates are not supported.

WRITABLE

You can change the username, email, first name, last name, and other mapped attributes and passwords and synchronize them automatically with the LDAP store.

UNSYNCED

Red Hat Single Sign-On stores changes to the username, email, first name, last name, and passwords in Red Hat Single Sign-On local storage, so the administrator must synchronize this data back to LDAP. In this mode, Red Hat Single Sign-On deployments can update user metadata on read-only LDAP servers. This option also applies when importing users from LDAP into the local Red Hat Single Sign-On user database.

**NOTE**

When Red Hat Single Sign-On creates the LDAP provider, Red Hat Single Sign-On also creates a set of initial [LDAP mappers](#). Red Hat Single Sign-On configures these mappers based on a combination of the **Vendor**, **Edit Mode**, and **Import Users** switches. For example, when edit mode is UNSYNCED, Red Hat Single Sign-On configures the mappers to read a particular user attribute from the database and not from the LDAP server. However, if you later change the edit mode, the mapper's configuration does not change because it is impossible to detect if the configuration changes changed in UNSYNCED mode. Decide the **Edit Mode** when creating the LDAP provider. This note applies to **Import Users** switch also.

4.3.4. Other configuration options

Console Display Name

The name of the provider to display in the admin console.

Priority

The priority of the provider when looking up users or adding a user.

Sync Registrations

Toggle this switch to **ON** if you want new users created by Red Hat Single Sign-On added to LDAP.

Allow Kerberos authentication

Enable Kerberos/SPNEGO authentication in the realm with user data provisioned from LDAP. For more information, see the [Kerberos section](#).

Other options

Hover the mouse pointer over the tooltips in the Admin Console to see more details about these options.

4.3.5. Connecting to LDAP over SSL

When you configure a secure connection URL to your LDAP store (for example, `ldaps://myhost.com:636`), Red Hat Single Sign-On uses SSL to communicate with the LDAP server. Configure a truststore on the Red Hat Single Sign-On server side so that Red Hat Single Sign-On can trust the SSL connection to LDAP.

Configure the global truststore for Red Hat Single Sign-On with the Truststore SPI. For more information about configuring the global truststore, see the [Server Installation and Configuration Guide](#). If you do not configure the Truststore SPI, the truststore falls back to the default mechanism provided by Java, which can be the file supplied by the `javax.net.ssl.trustStore` system property or the cacerts file from the JDK if the system property is unset.

The **Use Truststore SPI** configuration property, in the LDAP federation provider configuration, controls the truststore SPI. By default, Red Hat Single Sign-On sets the property to **Only for ldaps**, which is adequate for most deployments. Red Hat Single Sign-On uses the Truststore SPI if the connection URL to LDAP starts with **ldaps** only.

4.3.6. Synchronizing LDAP users to Red Hat Single Sign-On

If you set the **Import Users** option, the LDAP Provider handles importing LDAP users into the Red Hat Single Sign-On local database. The first time a user logs in, the LDAP provider imports the LDAP user into the Red Hat Single Sign-On database and validates the LDAP password. This first time a user logs in is the only time Red Hat Single Sign-On imports the user. If you click the **Users** menu in the Admin

Console and click the **View all users** button, you only see the LDAP users authenticated at least once by Red Hat Single Sign-On. Red Hat Single Sign-On imports users this way, so this operation does not trigger an import of the entire LDAP user database.

If you want to sync all LDAP users into the Red Hat Single Sign-On database, configure and enable the **Sync Settings** on the LDAP provider configuration page.

Two types of synchronization exist:

Periodic Full sync

This type synchronizes all LDAP users into the Red Hat Single Sign-On database. The LDAP users already in Red Hat Single Sign-On, but different in LDAP, directly update in the Red Hat Single Sign-On database.

Periodic Changed users sync

When synchronizing, Red Hat Single Sign-On creates or updates users created or updated after the last sync only.

The best way to synchronize is to click **Synchronize all users** when you first create the LDAP provider, then set up periodic synchronization of changed users.

4.3.7. LDAP mappers

LDAP mappers are **listeners** triggered by the LDAP Provider. They provide another extension point to LDAP integration. LDAP mappers are triggered when:

- Users log in by using LDAP.
- Users initially register.
- The Admin Console queries a user.

When you create an LDAP Federation provider, Red Hat Single Sign-On automatically provides a set of **mappers** for this provider. This set is changeable by users, who can also develop mappers or update/delete existing ones.

User Attribute Mapper

This mapper specifies which LDAP attribute maps to the attribute of the Red Hat Single Sign-On user. For example, you can configure the **mail** LDAP attribute to the **email** attribute in the Red Hat Single Sign-On database. For this mapper implementation, a one-to-one mapping always exists.

FullName Mapper

This mapper specifies the full name of the user. Red Hat Single Sign-On saves the name in an LDAP attribute (usually **cn**) and maps the name to the **firstName** and **lastName** attributes in the Red Hat Single Sign-On database. Having **cn** to contain the full name of the user is common for LDAP deployments.



NOTE

When you register new users in Red Hat Single Sign-On and **Sync Registrations** is ON for the LDAP provider, the `fullName` mapper permits falling back to the username. This fallback is useful when using Microsoft Active Directory (MSAD). The common setup for MSAD is to configure the `cn` LDAP attribute as `fullName` and, at the same time, use the `cn` LDAP attribute as the **RDN LDAP Attribute** in the LDAP provider configuration. With this setup, Red Hat Single Sign-On falls back to the username. For example, if you create Red Hat Single Sign-On user "john123" and leave `firstName` and `lastName` empty, then the `fullName` mapper saves "john123" as the value of the `cn` in LDAP. When you enter "John Doe" for `firstName` and `lastName` later, the `fullName` mapper updates LDAP `cn` to the "John Doe" value as falling back to the username is unnecessary.

Hardcoded Attribute Mapper

This mapper adds a hardcoded attribute value to each Red Hat Single Sign-On user linked with LDAP. This mapper can also force values for the **enabled** or **emailVerified** user properties.

Role Mapper

This mapper configures role mappings from LDAP into Red Hat Single Sign-On role mappings. A single role mapper can map LDAP roles (usually groups from a particular branch of the LDAP tree) into roles corresponding to a specified client's realm roles or client roles. You can configure more Role mappers for the same LDAP provider. For example, you can specify that role mappings from groups under **ou=main,dc=example,dc=org** map to realm role mappings, and role mappings from groups under **ou=finance,dc=example,dc=org** map to client role mappings of client **finance**.

Hardcoded Role Mapper

This mapper grants a specified Red Hat Single Sign-On role to each Red Hat Single Sign-On user from the LDAP provider.

Group Mapper

This mapper maps LDAP groups from a branch of an LDAP tree into groups within Red Hat Single Sign-On. This mapper also propagates user-group mappings from LDAP into user-group mappings in Red Hat Single Sign-On.

MSAD User Account Mapper

This mapper is specific to Microsoft Active Directory (MSAD). It can integrate the MSAD user account state into the Red Hat Single Sign-On account state, such as enabled account or expired password. This mapper uses the **userAccountControl**, and **pwdLastSet** LDAP attributes, specific to MSAD and are not the LDAP standard. For example, if the value of **pwdLastSet** is **0**, the Red Hat Single Sign-On user must update their password. The result is an `UPDATE_PASSWORD` required action added to the user. If the value of **userAccountControl** is **514** (disabled account), the Red Hat Single Sign-On user is disabled.

Certificate Mapper

This mapper maps X.509 certificates. Red Hat Single Sign-On uses it in conjunction with X.509 authentication and **Full certificate in PEM format** as an identity source. This mapper behaves similarly to the **User Attribute Mapper**, but Red Hat Single Sign-On can filter for an LDAP attribute storing a PEM or DER format certificate. Enable **Always Read Value From LDAP** with this mapper.

User Attribute mappers that map basic Red Hat Single Sign-On user attributes, such as username, firstname, lastname, and email, to corresponding LDAP attributes. You can extend these and provide your own additional attribute mappings. The Admin Console provides tooltips to help with configuring the corresponding mappers.

4.3.8. Password hashing

When Red Hat Single Sign-On updates a password, Red Hat Single Sign-On sends the password in

plain-text format. This action is different from updating the password in the built-in Red Hat Single Sign-On database, where Red Hat Single Sign-On hashes and salts the password before sending it to the database. For LDAP, Red Hat Single Sign-On relies on the LDAP server to hash and salt the password.

By default, LDAP servers such as MSAD, RHDS, or FreeIPA hash and salt passwords. Other LDAP servers such as OpenLDAP or ApacheDS store the passwords in plain-text unless you use the *LDAPv3 Password Modify Extended Operation* as described in [RFC3062](#). Enable the LDAPv3 Password Modify Extended Operation in the LDAP configuration page. See the documentation of your LDAP server for more details.



WARNING

Always verify that user passwords are properly hashed and not stored as plaintext by inspecting a changed directory entry using **ldapsearch** and base64 decode the **userPassword** attribute value.

4.3.9. Troubleshooting

It is useful to increase the logging level to TRACE for the category **org.keycloak.storage.ldap**. With this setting, many logging messages are sent to the server log in the **TRACE** level, including the logging for all queries to the LDAP server and the parameters, which were used to send the queries. When you are creating any LDAP question on user forum or JIRA, consider attaching the server log with enabled TRACE logging. If it is too big, the good alternative is to include just the snippet from server log with the messages, which were added to the log during the operation, which causes the issues to you.

- When you create LDAP provider, message appear in the server log in the INFO level starting with:

When you create LDAP provider, message appear in the server log in the INFO level starting with:

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

It shows the configuration of your LDAP provider. Before you are asking the questions or reporting bugs, it will be nice to include this message to show your LDAP configuration. Eventually feel free to replace some config changes, which you do not want to include, with some placeholder values. One example is **bindDn=some-placeholder**. For **connectionUrl**, feel free to replace it as well, but it is generally useful to include at least the protocol, which was used (**ldap** vs **ldaps**). Similarly it can be useful to include the details for configuration of your LDAP mappers, which are displayed with the message like this at the DEBUG level:

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

Note those messages are displayed just with the enabled DEBUG logging.

- For tracking the performance or connection pooling issues, consider setting the value of property **Connection Pool Debug Level** of

For tracking the performance or connection pooling issues, consider setting the value of property

Connection Pool Debug Level of the LDAP provider to value **all**. This will add lots of additional messages to server log with the included logging for the LDAP connection pooling. This can be used to track the issues related to connection pooling or performance.



NOTE

After changing the configuration of connection pooling, you may need to restart the Keycloak server to enforce re-initialization of the LDAP provider connection.

If no more messages appear for connection pooling even after server restart, it can indicate that connection pooling does not work with your LDAP server.

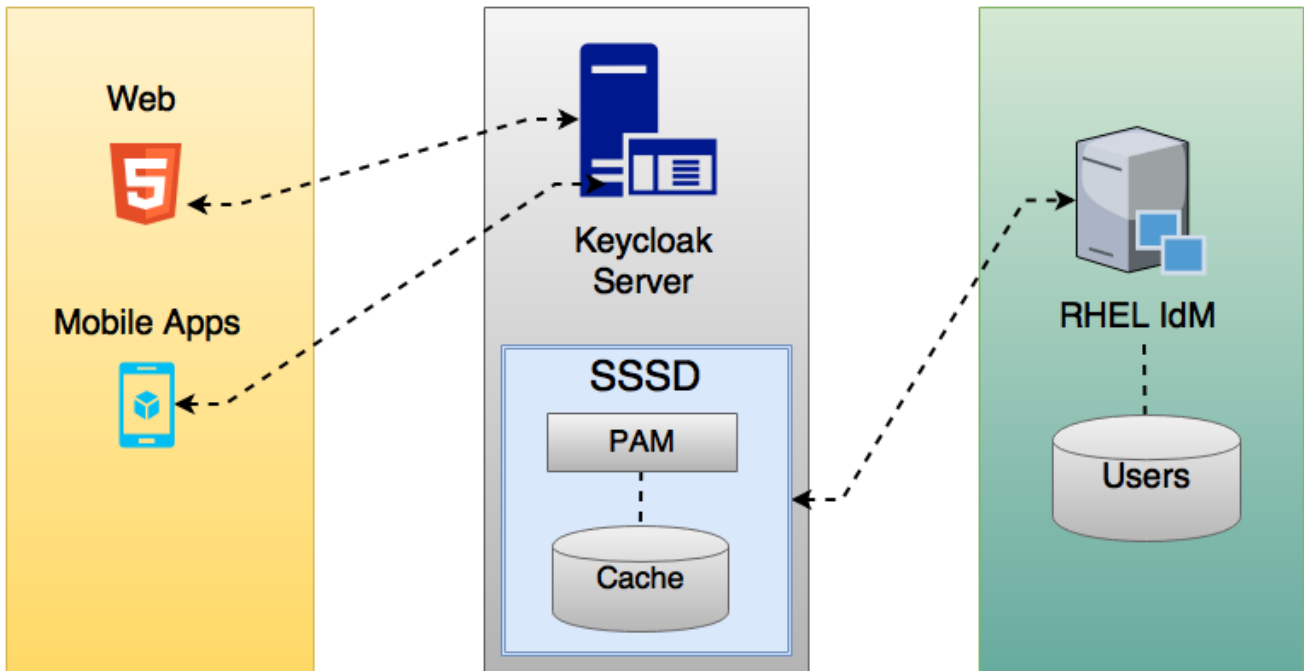
- For the case of reporting LDAP issue, you may consider to attach some part of your LDAP tree with the target data, which causes issues in your environment. For example if login of some user takes lot of time, you can consider attach his LDAP entry showing count of **member** attributes of various "group" entries. In this case, it might be useful to add if those group entries are mapped to some Group LDAP mapper (or Role LDAP Mapper) in Red Hat Single Sign-On etc.

For the case of reporting LDAP issue, you may consider to attach some part of your LDAP tree with the target data, which causes issues in your environment. For example if login of some user takes lot of time, you can consider attach his LDAP entry showing count of **member** attributes of various "group" entries. In this case, it might be useful to add if those group entries are mapped to some Group LDAP mapper (or Role LDAP Mapper) in Red Hat Single Sign-On and so on.

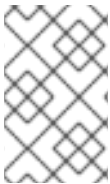
4.4. SSSD AND FREEIPA IDENTITY MANAGEMENT INTEGRATION

Red Hat Single Sign-On includes the [System Security Services Daemon \(SSSD\)](#) plugin. SSSD is part of the Fedora and Red Hat Enterprise Linux (RHEL), and it provides access to multiple identities and authentication providers. SSSD also provides benefits such as failover and offline support. For more information, see [the Red Hat Enterprise Linux Identity Management documentation](#) .

SSSD integrates with the FreeIPA identity management (IdM) server, providing authentication and access control. With this integration, Red Hat Single Sign-On can authenticate against privileged access management (PAM) services and retrieve user data from SSSD. For more information about using Red Hat Identity Management in Linux environments, see [the Red Hat Enterprise Linux Identity Management documentation](#).



Red Hat Single Sign-On and SSSD communicate through read-only D-Bus interfaces. For this reason, the way to provision and update users is to use the FreeIPA/IdM administration interface. By default, the interface imports the username, email, first name, and last name.



NOTE

Red Hat Single Sign-On registers groups and roles automatically but does not synchronize them. Any changes made by the Red Hat Single Sign-On administrator in Red Hat Single Sign-On do not synchronize with SSSD.

4.4.1. FreeIPA/IdM server

The [FreeIPA Docker image](#) is available in Docker Hub. To set up the FreeIPA server, see the [FreeIPA documentation](#).

Procedure

1. Run your FreeIPA server using this command:

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

The parameter **-h** with **server.freeipa.local** represents the FreeIPA/IdM server hostname. Change **YOUR_PASSWORD** to a password of your own.

2. After the container starts, change the **/etc/hosts** file to include:

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

If you do not make this change, you must set up a DNS server.

3. Use the following command to enroll your Linux server in the IPA domain so that the SSSD federation provider starts and runs on Red Hat Single Sign-On:

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

4. Run the following command on the client to verify the installation is working:

```
kinit admin
```

5. Enter your password.
6. Add users to the IPA server using this command:

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

7. Force set the user's password using kinit.

```
kinit <username>
```

8. Enter the following to restore normal IPA operation:

```
kdestroy -A  
kinit admin
```

4.4.2. SSSD and D-Bus

The federation provider obtains the data from SSSD using D-BUS. It authenticates the data using PAM.

Procedure

1. Install the `sssd-dbus` RPM.

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

2. Run the following provisioning script:

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

This script makes the following changes to `/etc/sssd/sssd.conf`:

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

3. Run **dbus-send** to ensure the setup is successful.

```
sudo dbus-send --print-reply --system --dest=org.freedesktop.sssd.infopipe
/org/freedesktop/sss/infopipe org.freedesktop.sssd.infopipe.GetUserGroups string:john
```

If the setup is successful, you see the user's group. If this command returns a timeout or an error, the federation provider running on Red Hat Single Sign-On cannot retrieve any data. This error usually happens because the server is not enrolled in the FreeIPA IdM server, or does not have permission to access the SSSD service.

If you do not have permission to access the SSSD service, ensure that the user running the Red Hat Single Sign-On server is in the `/etc/sss/sss.conf` file in the following section:

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

4.4.3. Enabling the SSSD federation provider

Red Hat Single Sign-On uses DBus-Java to communicate at a low level with D-Bus. D-Bus depends on the [Unix Sockets Library](#).

Before enabling the SSSD Federation provider, install the RPM for this library:

```
$ sudo yum install rh-sso7-libunix-dbus-java
```

Red Hat Single Sign-On uses JNA to authenticate with PAM. Ensure you have the JAN package installed.

```
$ sudo yum install jna
```

Use the `sssctl user-checks` command to validate your setup:

```
$ sudo sssctl user-checks admin -s keycloak
```

4.5. CONFIGURING A FEDERATED SSSD STORE

After the installation, configure a federated SSSD store.

Procedure

1. Click **User Federation** in the menu.
2. From the **Add Provider** list select `sss`. Red Hat Single Sign-On brings you to the `sss` configuration page.
3. Click **Save**.

You can now authenticate against Red Hat Single Sign-On using FreeIPA/IdM credentials.

4.6. CUSTOM PROVIDERS

Red Hat Single Sign-On does have a Service Provider Interface (SPI) for User Storage Federation to develop custom providers. You can find documentation on developing customer providers in the [Server Developer Guide](#).

CHAPTER 5. MANAGING USERS

From the Admin Console, you have a wide range of actions you can perform to manage users.

5.1. CREATING USERS

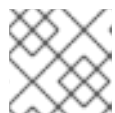
You create users in the realm where you intend to have applications needed by those users. Avoid creating users in the master realm, which is only intended for creating other realms.

Prerequisite

- You are in a realm other than the master realm.

Procedure

1. Click **Users** in the menu.
2. Click **Add User**.
3. Enter the details for the new user.



NOTE

Username is the only required field.

4. Click **Save**. After saving the details, the **Management** page for the new user is displayed.

5.2. DEFINING USER CREDENTIALS

You can manage credentials of a user in the **Credentials** tab.

Credential management

The screenshot shows the 'Credentials' tab for user 'Johndoe'. The interface includes a sidebar with navigation options such as 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication) and 'Manage' (Groups, Users, Sessions, Events, Import, Export). The main content area is titled 'Manage Credentials' and contains a table with the following columns: Position, Type, User Label, Data, and Actions. Below the table, there are two main sections: 'Set Password' and 'Credential Reset'. The 'Set Password' section includes two password input fields, a 'Temporary' toggle set to 'ON', and a 'Set Password' button. The 'Credential Reset' section includes a 'Reset Actions' dropdown menu, an 'Expires In' field set to 12 hours, and a 'Send email' button.

This tab includes the following fields:

Position

The arrow buttons in the **Position** column allow you to shift the priority of the credential for the user. The topmost credential has the highest priority. The priority determines which credential is displayed first after a user logs in.

Type

This column displays the type of credential, for example **password** or **OTP**.

User Label

This is an assignable label to recognize the credential when presented as a selection option during login. It can be set to any value to describe the credential.

Data

This is the non-confidential technical information about the credential. It is hidden, by default. You can click **Show data...** to display the data for a credential.

Actions

This column has two actions. Click **Save** to record the value or the user field. Click **Delete** to remove the credential.

You cannot configure other types of credentials for a specific user in the admin console; that task is the user's responsibility.

You can delete the credentials of a user in the event a user loses an OTP device or if credentials have been compromised. You can only delete credentials of a user in the **Credentials** tab.

5.2.1. Setting a password for a user

If a user does not have a password, or if the password has been deleted, the **Set Password** section is displayed.

If a user already has a password, it can be reset in the **Reset Password** section.

Procedure

1. Click **Users** in the menu. The **Users** page is displayed.
2. Select a user.
3. Click the **Credentials** tab.
4. Type a new password in the **Set Password** section.
5. Click **Set Password**.



NOTE

If **Temporary** is **ON**, the user must change the password at the first login. To allow users to keep the password supplied, set **Temporary** to **OFF**. The user must click **Set Password** to change the password.

6. Alternatively, you can send an email to the user that requests the user reset the password.
 - a. Navigate to the **Reset Actions** list under **Credential Reset**.
 - b. Select **Update Password** from the list.
 - c. Click **Send Email**. The sent email contains a link that directs the user to the **Update Password** window.
 - d. Optionally, you can set the validity of the email link. This is set to the default preset in the **Tokens** tab in **Realm Settings**.

5.2.2. Creating an OTP

If OTP is conditional in your realm, the user must navigate to Red Hat Single Sign-On Account Console to reconfigure a new OTP generator. If OTP is required, then the user must reconfigure a new OTP generator when logging in.

Alternatively, you can send an email to the user that requests the user reset the OTP generator. The following procedure also applies if the user already has an OTP credential.

Prerequisite

- You are logged in to the appropriate realm.

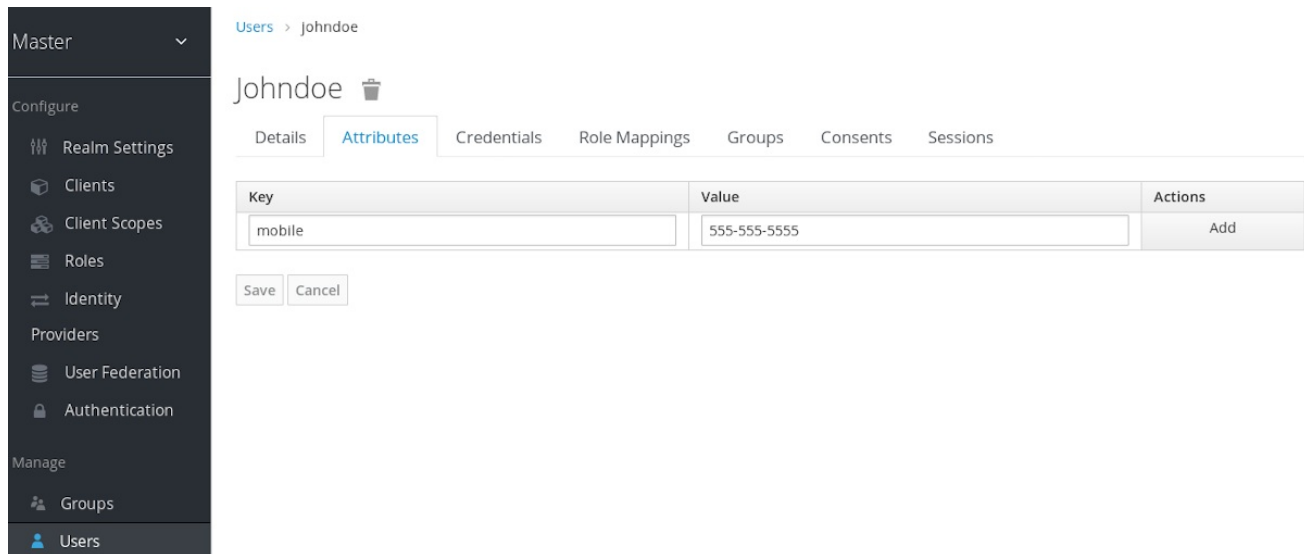
Procedure

1. Click **Users** in the main menu. The **Users** page is displayed.
2. Select a user.
3. Click the **Credentials** tab.
4. Navigate to the **Reset Actions** list.
5. Click **Configure OTP**.
6. Click **Send Email**. The sent email contains a link that directs the user to the **OTP setup page**.

5.3. CONFIGURING USER ATTRIBUTES

User attributes provide a customized experience for each user. You can create a personalized identity for each user in the console by configuring user attributes.

Users



The screenshot shows the user management interface. On the left is a navigation menu with sections: Master, Configure (Realm Settings, Clients, Client Scopes, Roles, Identity), Providers (User Federation, Authentication), and Manage (Groups, Users). The main area shows the user 'Johndoe' with tabs for Details, Attributes (selected), Credentials, Role Mappings, Groups, Consents, and Sessions. Below the tabs is a table for attributes:

Key	Value	Actions
mobile	555-555-5555	Add

Below the table are 'Save' and 'Cancel' buttons.

Prerequisite

- You are in the realm where the user exists.

Procedure

1. Click **Users** in the menu.
2. Select a user to manage.
3. Click the **Attributes** tab.
4. Enter the attribute name in the **Key** field.
5. Enter the attribute value in the **Value** field.
6. Click **Add**.
7. Click **Save**.



NOTE

Some read-only attributes are not supposed to be updated by the administrators. This includes attributes that are read-only by design like for example **LDAP_ID**, which is filled automatically by the LDAP provider. Some other attributes should be read-only for typical user administrators due to security reasons. See the details in the [Mitigating security threats](#) chapter.

5.4. ALLOWING USERS TO SELF-REGISTER

You can use Red Hat Single Sign-On as a third-party authorization server to manage application users, including users who self-register. If you enable self-registration, the login page displays a registration link so that user can create an account.

Registration link

Sign in to your account

Username or email

Password

Sign In

New user? [Register](#)

A user must add profile information to the registration form to complete registration. The registration form can be customized by removing or adding the fields that must be completed by a user.

Additional resources

- For more information on customizing user registration, see the [Server Developer Guide](#).

5.4.1. Enabling user registration

Enable users to self-register.

Procedure

1. Click **Realm Settings** in the main menu.
2. Click the **Login** tab.
3. Toggle **User Registration** to **ON**.
4. Click **Save**.

After you enable this setting, a **Register** link displays on the login page of the Admin Console.

5.4.2. Registering as a new user

As a new user, you must complete a registration form to log in for the first time. You add profile information and a password to register.

Registration form

Register

First name

Last name

Email

Username

Password

Confirm password

[« Back to Login](#)

Register

Prerequisite

- User registration is enabled.

Procedure

1. Click the **Register** link on the login page. The registration page is displayed.
2. Enter the user profile information.
3. Enter the new password.
4. Click **Save**.

5.5. DEFINING ACTIONS REQUIRED AT LOGIN

You can set the actions that a user must perform at the first login. These actions are required after the user provides credentials. After the first login, these actions are no longer required. You add required actions on the **Details** tab of that user.

The following are examples of required action types:

Update Password

The user must change their password.

Configure OTP

The user must configure a one-time password generator on their mobile device using either the Free OTP or Google Authenticator application.

Verify Email

The user must verify their email account. An email will be sent to the user with a validation link that they must click. Once this workflow is successfully completed, the user will be allowed to log in.

Update Profile

The user must update profile information, such as name, address, email, and phone number.

5.5.1. Setting required actions for one user

You can set the actions that are required for any user.

Procedure

1. Click **Users** in the menu.
2. Select a user from the list.
3. Navigate to the **Required User Actions** list.

The screenshot shows the user management interface for a user named 'Johndoe'. On the left is a dark sidebar menu with sections: 'Master' (dropdown), 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication), and 'Manage' (Groups, Users, Sessions, Events, Import, Export). The 'Users' option is selected. The main content area shows the breadcrumb 'Users > johndoe' and the user name 'Johndoe' with a trash icon. Below this are tabs for 'Details', 'Attributes', 'Credentials', 'Role Mappings', 'Groups', 'Consents', and 'Sessions'. The 'Details' tab is active, displaying the following fields and controls:

- ID:** 80589290-ee2d-4e22-9c15-73aa96642364
- Created At:** 2/13/20 3:53:18 PM
- Username:** johndoe
- Email:** johndoe@example.com
- First Name:** John
- Last Name:** Doe
- User Enabled:** ON (toggle)
- Email Verified:** OFF (toggle)
- Required User Actions:** Update Password (button with X icon)
- Impersonate user:** Impersonate (button)

4. Select all the actions you want to add to the account.
5. Click the **X** next to the action name to remove it.
6. Click **Save** after you select which actions to add.

5.5.2. Setting required actions for all users

You can specify what actions are required before the first login of all new users. The requirements apply to a user created by the **Add User** button on the **Users** page or the **Register** link on the login page.

Procedure

1. Click **Authentication** in the menu.
2. Click the **Required Actions** tab.
3. Click the checkbox in the **Default Action** column for one or more required actions. When a new user logs in for the first time, the selected actions must be executed.

5.5.3. Enabling terms and conditions as a required action

You can enable a required action that new users must accept the terms and conditions before logging in to Red Hat Single Sign-On for the first time.

Procedure

1. Click **Authentication** in the menu.
2. Click the **Required Actions** tab.
3. Enable the **Terms and Conditions** action.

4. Edit the **terms.ftl** file in the base login theme.

Additional resources

- For more information on extending and creating themes, see the [Server Developer Guide](#).

5.6. SEARCHING FOR A USER

Search for a user to view detailed information about the user, such as the user's groups and roles.

Prerequisite

- You are in the realm where the user exists.

Procedure

1. Click **Users** in the main menu. This **Users** page is displayed.
2. Type the full name, last name, first name, or email address of the user you want to search for in the search box. The search returns all users who match your criteria.
3. Alternatively, you can click **View all users** to list every user in the system.



NOTE

This action searches only the local Red Hat Single Sign-On database and not the federated database, such as LDAP. The backends for federated databases do not have a pagination mechanism that enables searching for users.

- a. To search users from a federated backend, the user list must be synced into the Red Hat Single Sign-On database. Adjust the search criteria to sync the backend users to the Red Hat Single Sign-On database.
- b. Alternatively, click the **User Federation** in the left menu.
 - i. To apply changes to a selected user, click **Sync changed users** on the page with your federation provider.
 - ii. To apply changes to all users in the database, click **Sync all users** on the page with your federation provider.

Additional resources

- For more information on user federation, see [User Federation](#).

5.7. DELETING A USER

You can delete a user, who no longer needs access to applications. If a user is deleted, the user profile and data is also deleted.

Procedure

1. Click **Users** in the menu. The **Users** page is displayed.

2. Click **View all users** to find a user to delete.



NOTE

Alternatively, you can use the search bar to find a user.

3. Click **Delete** next to the user you want to remove and confirm deletion.

5.8. ENABLING ACCOUNT DELETION BY USERS

End users and applications can delete their accounts in the Account Console if you enable this capability in the Admin Console. Once you enable this capability, you can give that capability to specific users.

5.8.1. Enabling the Delete Account Capability

You enable this capability on the **Required Actions** tab.

Procedure

1. Click **Authentication** in the menu.
2. Click the **Required Actions** tab.
3. Select **Enabled** on the **Delete Account** row.

Delete account on required actions tab

Authentication

Flows Bindings **Required Actions** Password Policy OTP Policy WebAuthn Policy WebAuthn Passwordless Policy

Required Action	Enabled	Default Action
Configure OTP	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Terms and Conditions	<input type="checkbox"/>	<input type="checkbox"/>
Update Password	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Update Profile	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Verify Email	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Update User Locale	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Delete Account	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Register

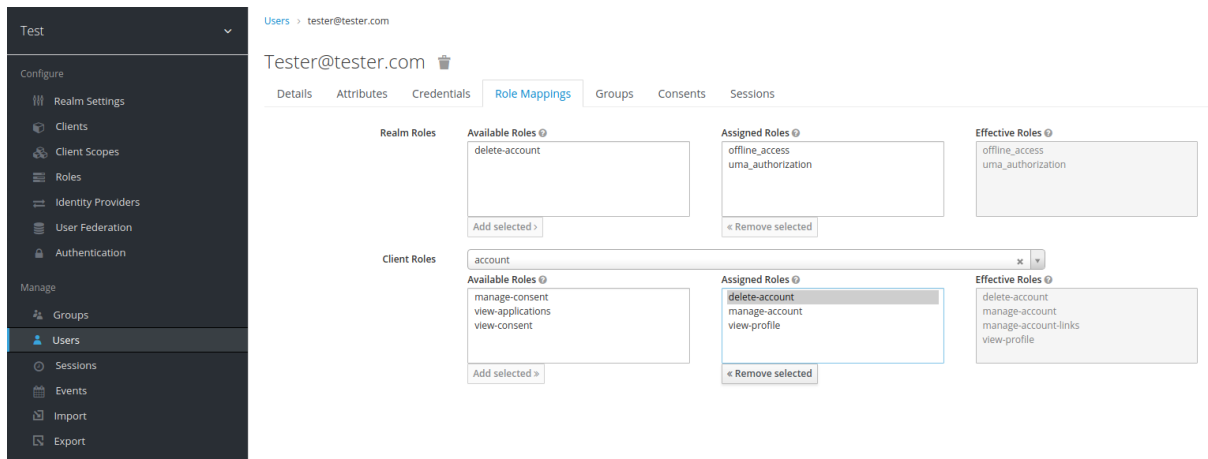
5.8.2. Giving a user the delete-account role

You can give specific users a role that allows account deletion.

Procedure

1. Click **Users** in the menu.
2. Select a user.
3. Click the **Role Mappings** tab.
4. From the **Client Roles** list, select **account**.
5. Under **Available Roles**, select **delete-account**.
6. Click **Add selected**.

Delete-account role

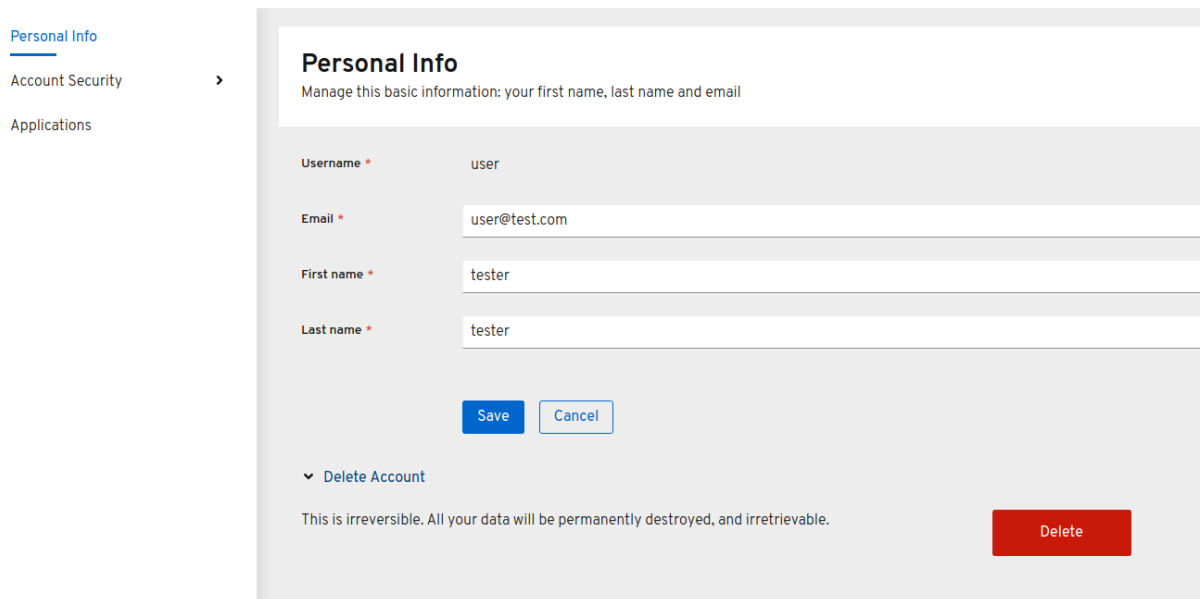


5.8.3. Deleting your account

Once you have the **delete-account** role, you can delete your own account.

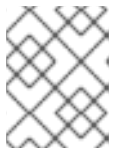
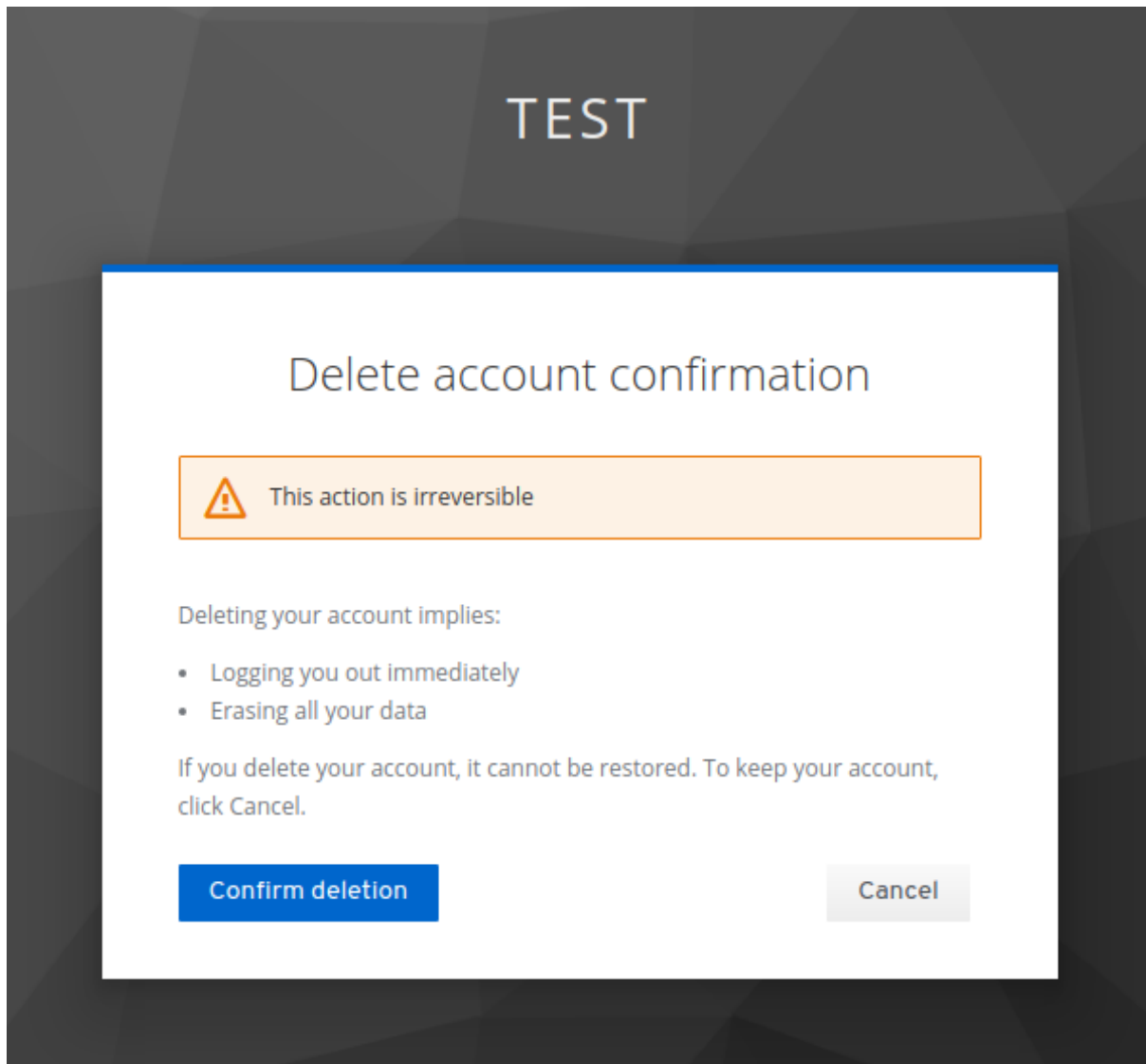
1. Log into the Account Console.
2. At the bottom of the **Personal Info** page, click **Delete Account**.

Delete account page



3. Enter your credentials and confirm the deletion.

Delete confirmation



NOTE

This action is irreversible. All your data in Red Hat Single Sign-On will be removed.

5.9. IMPERSONATING A USER

An administrator with the appropriate permissions can impersonate a user. For example, if a user experiences a bug in an application, an administrator can impersonate the user to investigate or duplicate the issue.

Any user with the **impersonation** role in the realm can impersonate a user.

The screenshot shows the 'Users' page for a user named 'Johndoe'. The left sidebar contains navigation options under 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication) and 'Manage' (Groups, Users, Sessions, Events, Import, Export). The 'Users' page has tabs for 'Details', 'Attributes', 'Credentials', 'Role Mappings', 'Groups', 'Consents', and 'Sessions'. The 'Details' tab is active, displaying the following information:

- ID:** 80589290-ee2d-4e22-9c15-73aa96642364
- Created At:** 2/13/20 3:53:18 PM
- Username:** johndoe
- Email:** johndoe@example.com
- First Name:** John
- Last Name:** Doe
- User Enabled:** ON
- Email Verified:** OFF
- Required User Actions:** Select an action...
- Impersonate user:** Impersonate

- If the administrator and the user are in the same realm, then the administrator will be logged out and automatically logged in as the user being impersonated.
- If the administrator and user are in different realms, the administrator will remain logged in, and additionally will be logged in as the user in that user's realm.

In both instances, the **User Account Management** page of the impersonated user is displayed.

You can access the **Impersonate** button from the **Details** tab on the **Users** page.

Additional resources

- For more information on assigning administration permissions, see the [Admin Console Access Control](#) chapter.

5.10. ENABLING RECAPTCHA

To safeguard registration against bots, Red Hat Single Sign-On has integration with Google reCAPTCHA.

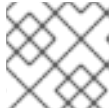
Once reCAPTCHA is enabled, you can edit **register.ftl** in your login theme to configure the placement and styling of the reCAPTCHA button on the registration page.

Procedure

1. Enter the following URL in a browser:

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

2. Create an API key to get your reCAPTCHA site key and secret. Note the reCAPTCHA site key and secret for future use in this procedure.

**NOTE**

The localhost works by default. You do not have to specify a domain.

3. Navigate to the Red Hat Single Sign-On admin console.
4. Click **Authentication** in the menu.
5. Click the **Flows** tab.
6. Select **Registration** from the drop down menu.
7. Set the **reCAPTCHA** requirement to **Required**. This enables reCAPTCHA.
8. Click **Actions** to the right of the reCAPTCHA flow entry.
9. Click the **Config** link.

Recaptcha config page

The screenshot shows the Red Hat Single Sign-On admin console. On the left is a dark sidebar with a navigation menu. The 'Authentication' section is highlighted. The main content area is titled 'Recaptcha' and shows the configuration for a specific flow. The breadcrumb navigation is 'Authentication Flows > Registration > recaptcha'. The configuration fields are:

- ID:** bbde9a8a-a005-4b27-a8db-0b1ddda9606c
- Alias:** recaptcha
- Recaptcha Site Key:** AAA0aY-SRkc3ksZyw4Aanqfa27Bn
- Recaptcha Secret:** 6LcFEAkTAAAAM0Ser
- use recaptcha.net:** OFF

- a. Enter the **Recaptcha Site Key** generated from the Google reCAPTCHA website.
 - b. Enter the **Recaptcha Secret** generated from the Google reCAPTCHA website.
10. Authorize Google to use the registration page as an iframe.

**NOTE**

In Red Hat Single Sign-On, websites cannot include a login page dialog in an iframe. This restriction is to prevent clickjacking attacks. You need to change the default HTTP response headers that is set in Red Hat Single Sign-On.

- a. Click **Realm Settings** in the menu.
- b. Click the **Security Defenses** tab.
- c. Enter <https://www.google.com> in the field for the **X-Frame-Options** header.
- d. Enter <https://www.google.com> in the field for the **Content-Security-Policy** header.

Additional resources

- For more information on extending and creating themes, see the [Server Developer Guide](#).

5.11. DEFINING A USER PROFILE

In Red Hat Single Sign-On a user is associated with a set of attributes. These attributes are used to better describe and identify users within Red Hat Single Sign-On as well as to pass over additional information about them to applications.

A user profile defines a well-defined schema for representing user attributes and how they are managed within a realm. By providing a consistent view over user information, it allows administrators to control the different aspects on how attributes are managed as well as to make a lot easier to extend Red Hat Single Sign-On to support additional attributes.

Among other capabilities, user profile enables administrators to:

- Define a schema for user attributes
- Define whether an attribute is required based on contextual information (e.g.: if required only for users, or admins, or both, or depending on the scope being requested.)
- Define specific permissions for viewing and editing user attributes, making possible to adhere to strong privacy requirements where some attributes can not be seen or be changed by third-parties (including administrators)
- Dynamically enforce user profile compliance so that user information is always updated and in compliance with the metadata and rules associated with attributes
- Define validation rules on a per-attribute basis by leveraging the built-in validators or writing custom ones
- Dynamically render forms that users interact with like registration, update profile, brokering, and personal information in the account console, according to the attribute definitions and without any need to manually change themes.

The User Profile capabilities are backed by the User Profile SPI. By default, these capabilities are disabled and realms are configured to use a default configuration that keeps backward compatibility with the legacy behavior.



NOTE

The legacy behavior is about keeping the default constraints used by Red Hat Single Sign-On when managing users root attributes such as username, email, first and last name, without any restriction on how custom attributes are managed. Regarding user flows such as registration, profile update, brokering, and managing accounts through the account console, users are restricted to use the attributes aforementioned with the possibility to change theme templates to support additional attributes.

If you are already using Red Hat Single Sign-On, the legacy behavior is what you have been using so far.

Differently than the legacy behavior, the declarative provider gives you a lot more flexibility to define the user profile configuration to a realm through the administration console and a well-defined JSON schema.

In the next sections, we'll be looking at how to use the declarative provider to define your own user profile configuration.



NOTE

In the future, the legacy behavior will no longer be supported in Red Hat Single Sign-On. Ideally, you should start looking at the new capabilities provided by the User Profile and migrate your realms accordingly.

5.11.1. Enabling the User Profile




NOTE

Declarative User Profile is **Technology Preview** and is not fully supported. This feature is disabled by default.

To enable start the server with **-Dkeycloak.profile=preview** or **-Dkeycloak.profile.feature.declarative_user_profile=enabled**. For more details see [Profiles](#).

In addition to enabling the **declarative_user_profile** feature, you should enable User Profile for a realm. To do that, click on the **Realm Settings** link on the left side menu and turn on the **User Profile Enabled** switch.

Myrealm 

General Login Keys Email Themes Localization Cache Tokens Client Registration Client Policies Security Defenses User Profile

* Name

Display name

HTML Display name

Frontend URL

Hostname

Enabled

User-Managed Access

User Profile Enabled

Endpoints

Once you enable it and click on the **Save** button, you can access the **User Profile** tab from where you can manage the configuration for user attributes.

By enabling the user profile for a realm, Red Hat Single Sign-On is going to impose additional constraints on how attributes are managed based on the user profile configuration. In summary, here is the list of what you should expect when the feature is enabled:

- From an administration point of view, the **Attributes** tab at the user details page will only show the attributes defined in the user profile configuration. The conditions defined on a per-attribute basis will also be taken into account when managing attributes.
- User facing forms like registration, update profile, brokering, and personal info in the account console, are going to be rendered dynamically based on the user profile configuration. For that, Red Hat Single Sign-On is going to rely on different templates to render these forms dynamically.

In the next topics, we'll be exploring how to manage the user profile configuration and how it affects your realm.

5.11.2. Managing the User Profile

The user profile configuration is managed on a per-realm basis. For that, click on the **Realm Settings** link on the left side menu and then click on the **User Profile** tab.

User Profile Tab

				Create	
Name	Display name	Attribute Group	Actions		
<input type="button" value="^"/> <input type="button" value="v"/> username	`\${username}`		Edit	Delete	
<input type="button" value="^"/> <input type="button" value="v"/> email	`\${email}`		Edit	Delete	
<input type="button" value="^"/> <input type="button" value="v"/> firstName	`\${firstName}`		Edit	Delete	
<input type="button" value="^"/> <input type="button" value="v"/> lastName	`\${lastName}`		Edit	Delete	

In the **Attributes** sub-tab you have a list of the attributes currently associated with the user profile. By default, the configuration is created based on the user root attributes and each attribute is configured with some defaults in terms of validation and permissioning.

In the **Attribute Groups** sub-tab you can manage attribute groups. An attribute group allows you to correlate attributes so that they are displayed together when rendering user facing forms.



NOTE

For now, attribute groups are only used for rendering purposes but in the future they should also enable defining top-level configurations to the attributes they are linked to.

In the **JSON Editor** sub-tab you can view and edit the configuration using a well-defined JSON schema. Any change you make when at any other tab are reflected in the JSON configuration shown at this tab.

In the next section, you are going to learn how to manage the configuration from the **Attributes** sub-tab.

5.11.3. Managing Attributes

At the **Attributes** sub-tab you can create, edit, and delete the attributes associated with the user profile.

To define a new attribute and associate it with the user profile, click on the **Create** button in the top-right corner of the attribute listing.

Attribute Configuration

Attribute configuration

Name ?	<input type="text"/>
Display name ?	<input type="text"/>
Attribute Group ?	<input type="text" value="v"/>
Enabled when scope ?	<input type="text" value="Select a scope..."/>
Required ?	<input type="checkbox"/> OFF

> Permission

> Validation

> Annotation

When configuring the attribute you can define the following settings:

Name

The name of the attribute.

Display name

A user-friendly name for the attribute, mainly used when rendering user-facing forms. It supports internationalization so that values can be loaded from message bundles.

Attribute Group

The attribute group to which the attribute belongs to, if any.

Enabled when scope

Allows you to define a list of scopes to dynamically enable an attribute. If not set, the attribute is always enabled and its constraints are always enforced when managing user profiles as well as when rendering user-facing forms. Otherwise, the same constraints only apply when any of the scopes in the list is requested by clients.

Required

Set the attribute as required. If not enabled, the attribute is optional. Otherwise, the attribute must be provided by users and administrators with the possibility to also make the attribute required only for users or administrators as well as based on the scopes requested by clients.

Permission

In this section, you can define read and write permissions for users and administrators.

Validation

In this section, you can define the validations that will be performed when managing the attribute value. Red Hat Single Sign-On provides a set of built-in validators you can choose from with the possibility to add your own.

Annotation

In this section, you can associate annotations to the attribute. Annotations are mainly useful to pass over additional metadata to frontends for rendering purposes.

5.11.3.1. Managing Permissions

In the **Permission** section, you can define the level of access users and administrators have to read and write to an attribute.

Attribute Permission

Attribute **birthDate** configuration

Name ?	<input type="text" value="birthDate"/>
Required ?	<input type="checkbox"/> OFF

▼ Permission

Can user view? ?	<input checked="" type="checkbox"/> ON
Can admin view? ?	<input checked="" type="checkbox"/> ON
Can user edit? ?	<input checked="" type="checkbox"/> ON
Can admin edit? ?	<input type="checkbox"/> OFF

For that, you can use the following settings:

Can user view?

If enabled, users can view the attribute. Otherwise, users don't have access to the attribute.

Can user edit?

If enabled, users can view and edit the attribute. Otherwise, users don't have access to write to the attribute.

Can admin view?

If enabled, administrators can view the attribute. Otherwise, administrators don't have access to the attribute.

Can admin edit?

If enabled, administrators can view and edit the attribute. Otherwise, administrators don't have access to write to the attribute.

**NOTE**

When you create an attribute, no permission is set to the attribute. Effectively, the attribute won't be accessible by either users or administrators. Once you create the attribute, make sure to set the permissions accordingly to that the attribute is only visible by the target audience.

Permissioning has a direct impact on how and who can manage the attribute, as well as on how the attribute is rendered in user-facing forms.

For instance, by marking an attribute as only viewable by users, the administrators won't have access to the attribute when managing users through the administration console (neither from the User API). Also, users won't be able to change the attribute when updating their profiles. An interesting configuration if user attributes are fetched from an existing identity store (federation) and you just want to make attributes visible to users without any possibility to update the attribute other than through the source identity store.

Similarly, you can also mark an attribute as writable only for administrators with read-only access for users. In this case, only administrators are going to be allowed to manage the attribute.

Depending on your privacy requirements, you might also want attributes inaccessible to administrators but with read-write permissions for users.


Make sure to set the correct permissions whenever you add a new attribute to the user profile configuration.

5.11.3.2. Managing validations

In the **Validation** section, you can choose from different forms of validation to make sure the attribute value conforms to specific rules.

Attribute Validation

Attribute **birthDate** configuration

Name 

Required  OFF

> Permission

∨ Validation

Add Validator 

Name	Config	Actions
local-date	{}	Delete

Red Hat Single Sign-On provides different validators out of the box:

Name	Description	Configuration
------	-------------	---------------

Name	Description	Configuration
length	Check the length of a string value based on a minimum and maximum length.	<p>min: an integer to define the minimum allowed length.</p> <p>max: an integer to define the maximum allowed length.</p> <p>trim-disabled: a boolean to define whether the value is trimmed prior to validation.</p>
integer	Check if the value is an integer and within a lower and/or upper range. If no range is defined, the validator only checks whether the value is a valid number.	<p>min: an integer to define the lower range.</p> <p>max: an integer to define the upper range.</p>
double	Check if the value is a double and within a lower and/or upper range. If no range is defined, the validator only checks whether the value is a valid number.	<p>min: an integer to define the lower range.</p> <p>max: an integer to define the upper range.</p>
uri	Check if the value is a valid URI.	None
pattern	Check if the value matches a specific RegEx pattern.	<p>pattern: the RegEx pattern to use when validating values.</p> <p>error-message: the key of the error message in i18n bundle. If not set a generic message is used.</p>
email	Check if the value has a valid e-mail format.	None
local-date	Check if the value has a valid format based on the realm and/or user locale.	None
person-name-prohibited-characters	Check if the value is a valid person name as an additional barrier for attacks such as script injection. The validation is based on a default RegEx pattern that blocks characters not common in person names.	error-message: the key of the error message in i18n bundle. If not set a generic message is used.


Name	Description	Configuration
username-prohibited-characters	Check if the value is a valid username as an additional barrier for attacks such as script injection. The validation is based on a default RegEx pattern that blocks characters not common in usernames.	error-message: the key of the error message in i18n bundle. If not set a generic message is used.
options	Check if the value is from the defined set of allowed values. Useful to validate values entered through select and multiselect fields.	options: array of strings containing allowed values.


5.11.3.2.1. Managing annotations

In order to pass additional information to frontends, attributes can be decorated with annotations to dictate how attributes are rendered. This capability is mainly useful when extending Red Hat Single Sign-On themes to render pages dynamically based on the annotations associated with attributes. This mechanism is used for example to configure Form input filed for attribute.

Attribute Annotation

Attribute **birthDate** configuration

Name 

Required  OFF

> Permission

> Validation

∨ Annotation

Key	Value	Actions
type	<input type="text" value="date"/>	Delete
<input type="text"/>	<input type="text"/>	Add

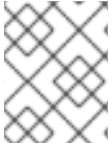
5.11.4. Managing Attribute Groups

At the **Attribute Groups** sub-tab you can create, edit, and delete attribute groups. An attribute group allows you to define a container for correlated attributes so that they are rendered together when at the user-facing forms.

Attribute Group List

Attributes [Attribute Groups](#) JSON Editor

				Create
Name	Display header	Display description	Actions	
personalInfo	Personal Information		Edit	Delete
addressInfo	Address Information		Edit	Delete



NOTE

You can't delete attribute groups that are bound to attributes. For that, you should first update the attributes to remove the binding.

To create a new group, click on the **Create** button in the top-right corner of the attribute groups listing.

Attribute Group Configuration

Attributes [Attribute Groups](#) JSON Editor

Attribute Group configuration

Name ?

Display header ?

Display
description ?

> Annotation

Save

Cancel

When configuring the group you can define the following settings:

Name

The name of the group.

Display name

A user-friendly name for the group, mainly used when rendering user-facing forms. It supports internationalization so that values can be loaded from message bundles.

Display description

A user-friendly text that will be displayed as a tooltip when rendering user-facing forms.

Annotation

In this section, you can associate annotations to the attribute. Annotations are mainly useful to pass over additional metadata to frontends for rendering purposes.

5.11.5. Using the JSON configuration

The user profile configuration is stored using a well-defined JSON schema. You can choose from editing the user profile configuration directly by clicking on the **JSON Editor** sub-tab.

JSON Configuration

Attributes [JSON Editor](#)

```
{
  "attributes": [
    {
      "name": "username"
    },
    {
      "name": "email"
    },
    {
      "name": "firstName",
      "required": {
        "roles": [
          "user"
        ]
      },
      "permissions": {
        "view": [
          "admin",
          "user"
        ],
        "edit": [
```

Save

Cancel

The JSON schema is defined as follows:

```
{
  "attributes": [
```

```

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

```

The schema supports as many attributes as you need.

For each attribute you should define a **name** and, optionally, the **required**, **permission**, and the **annotations** settings.

5.11.5.1. Required property

The **required** setting defines whether an attribute is required. Red Hat Single Sign-On allows you to set an attribute as required based on different conditions.

When the **required** setting is defined as an empty object, the attribute is always required.

```

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

```

On the other hand, you can choose to make the attribute required only for users, or administrators, or both. As well as mark the attribute as required only in case a specific scope is requested when the user is authenticating in Red Hat Single Sign-On.

To mark an attribute as required for a user and/or administrator, set the **roles** property as follows:

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

The **roles** property expects an array whose values can be either **user** or **admin**, depending on whether the attribute is required by the user or the administrator, respectively.

Similarly, you can choose to make the attribute required when a set of one or more scopes is requested by a client when authenticating a user. For that, you can use the **scopes** property as follows:

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

The **scopes** property is an array whose values can be any string representing a client scope.

5.11.5.2. Permissions property

The attribute-level **permissions** property can be used to define the read and write permissions to an attribute. The permissions are set based on whether these operations can be performed on the attribute by a user, or administrator, or both.

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

Both **view** and **edit** properties expect an array whose values can be either **user** or **admin**, depending on whether the attribute is viewable or editable by the user or the administrator, respectively.

When the **edit** permission is granted, the **view** permission is implicitly granted.

5.11.5.3. Annotations property

The attribute-level **annotation** property can be used to associate additional metadata to attributes. Annotations are mainly useful for passing over additional information about attributes to frontends rendering user attributes based on the user profile configuration. Each annotation is a key/value pair.

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

5.11.6. Using dynamic forms

One of the main capabilities of User Profile is the possibility to dynamically render user-facing forms based on attributes metadata. When you have the feature enabled to your realm, forms like registration and update profile are rendered using specific theme templates to dynamically render pages based on the user profile configuration.

That said, you shouldn't need to customize templates at all if the default rendering mechanisms serves to your needs. In case you still need customizations to themes, here are the templates you should be looking at:

Template	Description
base/login/update-user-profile.ftl	The template that renders the update profile page.
base/login/register-user-profile.ftl	The template that renders the registration page.
base/login/idp-review-user-profile.ftl	The template that renders the page to review/update the user profile when federating users through brokering.
base/login/user-profile-commons.ftl	The template that renders input fields in forms based on attributes configuration. Used from all three page templates described above. New input types can be implemented here.

The default rendering mechanism provides the following capabilities:

- Dynamically display fields based on the permissions set to attributes.
- Dynamically render markers for required fields based on the constraints set to the attributes.
- Dynamically render field input type (text, date, number, select, multiselect) set to an attribute.
- Dynamically render read-only fields depending on the permissions set to an attribute.
- Dynamically order fields depending on the order set to the attributes.



















- Dynamically group fields that belong to a same attribute group.

5.11.6.1. Ordering attributes

The attributes order is set by clicking on the up and down arrows when at the attribute listing page.

Ordering Attributes

[Attributes](#) [Attribute Groups](#) [JSON Editor](#)

						Create
Name	Display name	Attribute Group	Actions			
  username	\${username}		Edit	Delete		
  email	\${email}		Edit	Delete		
  firstName	\${firstName}	personalInfo	Edit	Delete		
  lastName	\${lastName}	personalInfo	Edit	Delete		
  birthDate	Date of Birth	personalInfo	Edit	Delete		
  address	Address	addressInfo	Edit	Delete		
  postalCode	Postal Code	addressInfo	Edit	Delete		
  city	City	addressInfo	Edit	Delete		
  Country	Country	addressInfo	Edit	Delete		

The order you set in this page is respected when fields are rendered in dynamic forms.

5.11.6.2. Grouping attributes

When dynamic forms are rendered, they will try to group together attributes that belong to a same attribute group.

Dynamic Update Profile Form

* Required fields

Update Account Information

Email *

Personal Information

First name *

Last name *

Date of Birth

Address Information

Address *

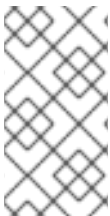


Please specify this field.

Postal Code *



Please specify this field.



NOTE

When attributes are linked to an attribute group, the attribute order is also important to make sure attributes within the same group are close together, within a same group header. Otherwise, if attributes within a group do not have a sequential order you might have the same group header rendered multiple times in the dynamic form.

5.11.6.3. Configuring Form input filed for Attributes

Red Hat Single Sign-On provides built-in annotations to configure which input type will be used for the attribute in dynamic forms and other aspects of it's visualization.

Available annotations are:

Name	Description
inputType	Type of the form input field. Available types are described in a table below.
inputHelperTextBefore	Helper text rendered before (above) the input field. Direct text or internationalization pattern (like #{i18n.key}) can be used here. Text is NOT html escaped when rendered into the page, so you can use html tags here to format the text, but you also have to correctly escape html control characters.
inputHelperTextAfter	Helper text rendered after (under) the input field. Direct text or internationalization pattern (like #{i18n.key}) can be used here. Text is NOT html escaped when rendered into the page, so you can use html tags here to format the text, but you also have to correctly escape html control characters.
inputOptionsFromValidation	Annotation for select and multiselect types. Optional name of custom attribute validation to get input options from. See detailed description below.
inputOptionLabelsI18nPrefix	Annotation for select and multiselect types. Internationalization key prefix to render options in UI. See detailed description below.
inputOptionLabels	Annotation for select and multiselect types. Optional map to define UI labels for options (directly or using internationalization). See detailed description below.
inputTypePlaceholder	HTML input placeholder attribute applied to the field - specifies a short hint that describes the expected value of an input field (e.g. a sample value or a short description of the expected format). The short hint is displayed in the input field before the user enters a value.
inputTypeSize	HTML input size attribute applied to the field - specifies the width, in characters, of an single line input field. For fields based on HTML select type it specifies number of rows with options shown. May not work, depending on css in used theme!

Name	Description
inputTypeCols	HTML input cols attribute applied to the field - specifies the width, in characters, for textarea type. May not work, depending on css in used theme!
inputTypeRows	HTML input rows attribute applied to the field - specifies the height, in characters, for textarea type. For select fields it specifies number of rows with options shown. May not work, depending on css in used theme!
inputTypePattern	HTML input pattern attribute applied to the field providing client side validation - specifies a regular expression that an input field's value is checked against. Useful for single line inputs.
inputTypeMaxLength	HTML input maxlength attribute applied to the field providing client side validation - maximal length of the text which can be entered into the input field. Useful for text fields.
inputTypeMinLength	HTML input minlength attribute applied to the field providing client side validation - minimal length of the text which can be entered into the input field. Useful for text fields.
inputTypeMax	HTML input max attribute applied to the field providing client side validation - maximal value which can be entered into the input field. Useful for numeric fields.
inputTypeMin	HTML input min attribute applied to the field providing client side validation - minimal value which can be entered into the input field. Useful for numeric fields.
inputTypeStep	HTML input step attribute applied to the field - Specifies the interval between legal numbers in an input field. Useful for numeric fields.



NOTE

Field types use HTML form field tags and attributes applied to them - they behave based on the HTML specifications and browser support for them.

Visual rendering also depends on css styles applied in the used theme.

Available **inputType** annotation values:

Name	Description	HTML tag used
text	Single line text input.	input
textarea	Multiple line text input.	textarea
select	Common single select input. See description how to configure options below.	select
select-radiobuttons	Single select input through group of radio buttons. See description how to configure options below.	group of input
multiselect	Common multiselect input. See description how to configure options below.	select
multiselect-checkboxes	Multiselect input through group of checkboxes. See description how to configure options below.	group of input
html5-email	Single line text input for email address based on HTML 5 spec.	input
html5-tel	Single line text input for phone number based on HTML 5 spec.	input
html5-url	Single line text input for URL based on HTML 5 spec.	input
html5-number	Single line input for number (integer or float depending on step) based on HTML 5 spec.	input
html5-range	Slider for number entering based on HTML 5 spec.	input
html5-datetime-local	Date Time input based on HTML 5 spec.	input
html5-date	Date input based on HTML 5 spec.	input
html5-month	Month input based on HTML 5 spec.	input
html5-week	Week input based on HTML 5 spec.	input

Name	Description	HTML tag used
html5-time	Time input based on HTML 5 spec.	input


5.11.6.3.1. Defining options for select and multiselect fields


Options for select and multiselect fields are taken from validation applied to the attribute to be sure validation and field options presented in UI are always consistent. By default, options are taken from built-in **options** validation.


You can use various ways to provide nice human readable labels for select and multiselect options. The simplest case is when attribute values are same as UI labels. No any extra configuration is necessary in this case.


Option values same as UI labels


Attribute **jobTitle** configuration

Name 

Display name 

Attribute Group 

Enabled when scope 

Required  OFF

> Permission

∨ Validation

Add Validator 

Name	Config	Actions
options	{"options":["SW Engineer","SW architect"]}	Delete

∨ Annotation

Key	Value	Actions
inputType	<input type="text" value="select"/>	Delete
<input type="text"/>	<input type="text"/>	Add

When attribute value is kind of ID not suitable for UI, you can use simple internationalization support provided by **inputOptionLabelsI18nPrefix** annotation. It defines prefix for internationalization keys, option value is dot appended to this prefix.

Simple internationalization for UI labels using i18n key prefix

Attribute **jobTitle** configuration

Name ⓘ

Display name ⓘ

Attribute Group ⓘ

Enabled when scope ⓘ

Required ⓘ OFF

> Permission

∨ Validation

Add Validator ⓘ

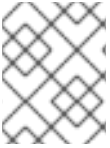
Name	Config	Actions
options	{"options":["sweng","swarch"]}	Delete

∨ Annotation

Key	Value	Actions
inputOptionLabelsI18nPrefix	<input type="text" value="userprofile.jobtitle"/>	Delete
inputType	<input type="text" value="select"/>	Delete
<input type="text"/>	<input type="text"/>	Add

Localized UI label texts for option value have to be provided by **userprofile.jobtitle.sweng** and **userprofile.jobtitle.swarch** keys then, using common localization mechanism.

You can also use **inputOptionLabels** annotation to provide labels for individual options. It contains map of labels for option - key in the map is option value (defined in validation), and value in the map is UI label text itself or its internationalization pattern (like **#{i18n.key}**) for that option.

**NOTE**

You have to use User Profile **JSON Editor** to enter map as **inputOptionLabels** annotation value.

Example of directly entered labels for individual options without internationalization:

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

Example of the internationalized labels for individual options:

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



Localized texts have to be provided by **jobtitle.swengineer** and **jobtitle.swarchitect** keys then, using common localization mechanism.


Custom validator can be used to provide options thanks to **inputOptionsFromValidation** attribute annotation. This validation have to have **options** config providing array of options. Internationalization works the same way as for options provided by built-in **options** validation.


Options provided by custom validator


[Attributes](#) Attribute Groups [JSON Editor](#)


Attribute **jobTitle** configuration

Name 

Display name 

Attribute Group 

Enabled when scope 

Required  OFF

> Permission

∨ Validation

Add Validator 

Name	Config	Actions
customOptionsValidator	{"options":["SW Engineer","SW architect"]}	Delete

∨ Annotation

Key	Value	Actions
inputOptionsFromValidation	<input type="text" value="customOptionsValidator"/>	Delete
inputType	<input type="text" value="select"/>	Delete
<input type="text" value=""/>	<input type="text" value=""/>	Add

5.11.7. Forcing User Profile compliance

In order to make sure user profiles are in compliance with the configuration, administrators may use the **VerifyProfile** required action to eventually force users to update their profiles when authenticating to Red Hat Single Sign-On.



NOTE

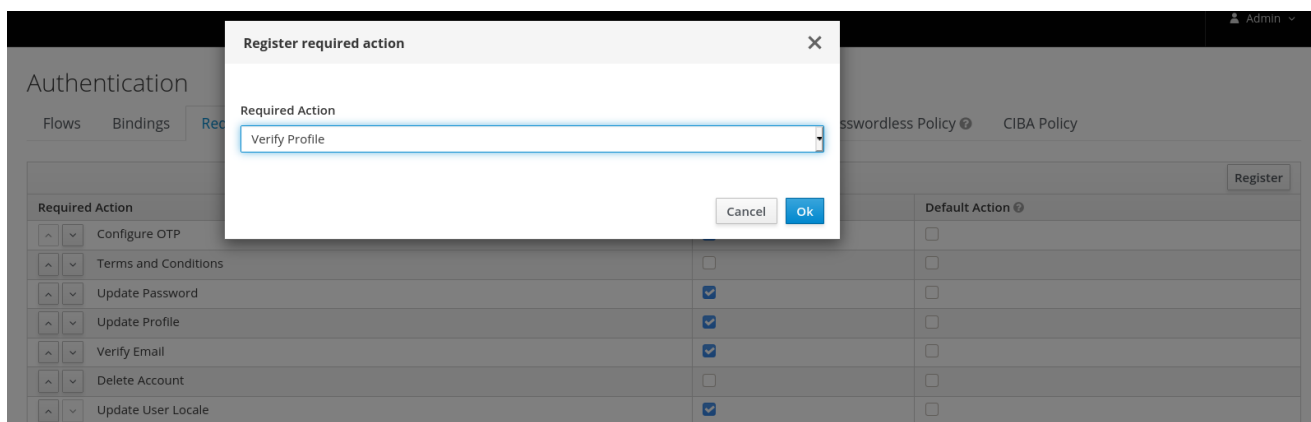
The **VerifyProfile** action is similar to the **UpdateProfile** action. However, it leverages all the capabilities provided by the user profile to automatically enforce compliance with the user profile configuration.

When enabled, the **VerifyProfile** action is going to perform the following steps when the user is authenticating:

- Check whether the user profile is fully compliant with the user profile configuration set to the realm.
- If not, perform an additional step during the authentication so that the user can update any missing or invalid attribute.
- If the user profile is compliant with the configuration, no additional step is performed, and the user continues with the authentication process.

By default, the **VerifyProfile** action is disabled. To enable it, click on the **Authentication** link on the left side menu and then click on the **Required Actions** tab. At this tab, click on the **Register** button and select the **VerifyProfile** action.

Registering the VerifyProfile Required Action



5.11.8. Migrating to User Profile

Before enabling the User Profile capabilities to a realm, there are some important considerations you should be aware of. By providing a single place to manage attribute metadata, the feature is very strict about the attributes that can be set to users and how they are managed.

In terms of user management, administrators are able to manage only the attributes defined in the user profile configuration. Any other attribute set to the user and not yet defined in the user profile configuration won't be accessible. It is recommended to update your user profile configuration with all the user attributes you want to expose either to users or administrators.

The same recommendation applies for those accessing the User REST API to query user information.

In regards to Red Hat Single Sign-On internal user attributes such as **LDAP_ID**, **LDAP_ENTRY_DN**, or **KERBEROS_PRINCIPAL**, if you want to be able to access those attributes you should have them as

attributes in your user profile configuration. The recommendation is to mark these attributes as viewable only to administrators so that you can look at them when managing the user attributes through the administration console or querying users via User API.

In regards to theming, if you already have customizations to the legacy templates (those hardcoded with user root attributes) your custom templates won't be used when rendering user-facing forms but the new templates that render these forms dynamically. Ideally, you should avoid having any customizations to templates and try to stick with the behavior provided by these new templates to dynamically render forms for you. If they are still not enough to address your requirements, you can either customize them or provide us with any feedback so that we discuss whether it makes sense to enhance the new templates.

5.12. PERSONAL DATA COLLECTED BY RED HAT SINGLE SIGN-ON

By default, Red Hat Single Sign-On collects the following data:

- Basic user profile data, such as the user email, first name, and last name.
- Basic user profile data used for social accounts and references to the social account when using a social login.
- Device information collected for audit and security purposes, such as the IP address, operating system name, and the browser name.

The information collected in Red Hat Single Sign-On is highly customizable. The following guidelines apply when making customizations:

- Registration and account forms can contain custom fields, such as birthday, gender, and nationality. An administrator can configure Red Hat Single Sign-On to retrieve data from a social provider or a user storage provider such as LDAP.
- Red Hat Single Sign-On collects user credentials, such as password, OTP codes, and WebAuthn public keys. This information is encrypted and saved in a database, so it is not visible to Red Hat Single Sign-On administrators. Each type of credential can include non-confidential metadata that is visible to administrators such as the algorithm that is used to hash the password and the number of hash iterations used to hash the password.
- With authorization services and UMA support enabled, Red Hat Single Sign-On can hold information about some objects for which a particular user is the owner.

CHAPTER 6. MANAGING USER SESSIONS

When users log into realms, Red Hat Single Sign-On maintains a user session for each user and remembers each client visited by the user within the session. Realm administrators can perform multiple actions on each user session:

- View login statistics for the realm.
- View active users and where they logged in.
- Log a user out of their session.
- Revoke tokens.
- Set up token timeouts.
- Set up session timeouts.

6.1. ADMINISTERING SESSIONS

To see a top-level view of the active clients and sessions in Red Hat Single Sign-On, click **Sessions** from the menu.

Sessions

The screenshot shows the 'Sessions' page in the Red Hat Single Sign-On administration console. The left sidebar contains a navigation menu with 'Sessions' highlighted. The main content area has a title 'Sessions' and two tabs: 'Realm Sessions' (selected) and 'Revocation'. Below the tabs is a table with the following data:

Client	Active Sessions	Offline Sessions
security-admin-console	1	0

A 'Logout all' button is located in the top right corner of the table area.

6.1.1. The Logout all Operation

You can log out all users in the realm by clicking the **Logout all** button.

When you click the **Logout all** button, all SSO cookies become invalid, and clients requesting authentication within active browser sessions must log in again. Red Hat Single Sign-On notifies clients by using the Red Hat Single Sign-On OIDC client adapter of the logout event. Client types such as SAML do not receive a back-channel logout request.



NOTE

Clicking **Logout all** does not revoke outstanding access tokens. Outstanding tokens must expire naturally. For clients using the Red Hat Single Sign-On OIDC client adapter, you can push a [revocation policy](#) to revoke the token, but this does not work for other adapters.

6.1.2. Application navigation

On the **Sessions** page, you can click on each client to go to that client's **Sessions** tab. Click the **Show Sessions** button there to see which users are in the application.

Application sessions

Master ▼

Configure

- Realm Settings
- Clients**
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

Clients > security-admin-console

Security-admin-console

Settings Roles Client Scopes [?] Mappers [?] Scope [?] Revocation

Sessions [?] Offline Access [?] Installation [?]

Active Sessions [?]

1

[Show Sessions](#)

User	From IP	Session Start
admin	127.0.0.1	Feb 21, 2020 12:53:01 PM

6.1.3. User navigation

If you go to the **Sessions** tab of an individual user, you can also view the user's session information.

User Sessions

Master ▼

Configure

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

Manage

- Groups
- Users**

Users > admin

Admin

Details Attributes Credentials Role Mappings Groups Consents

Sessions Identity Provider Links

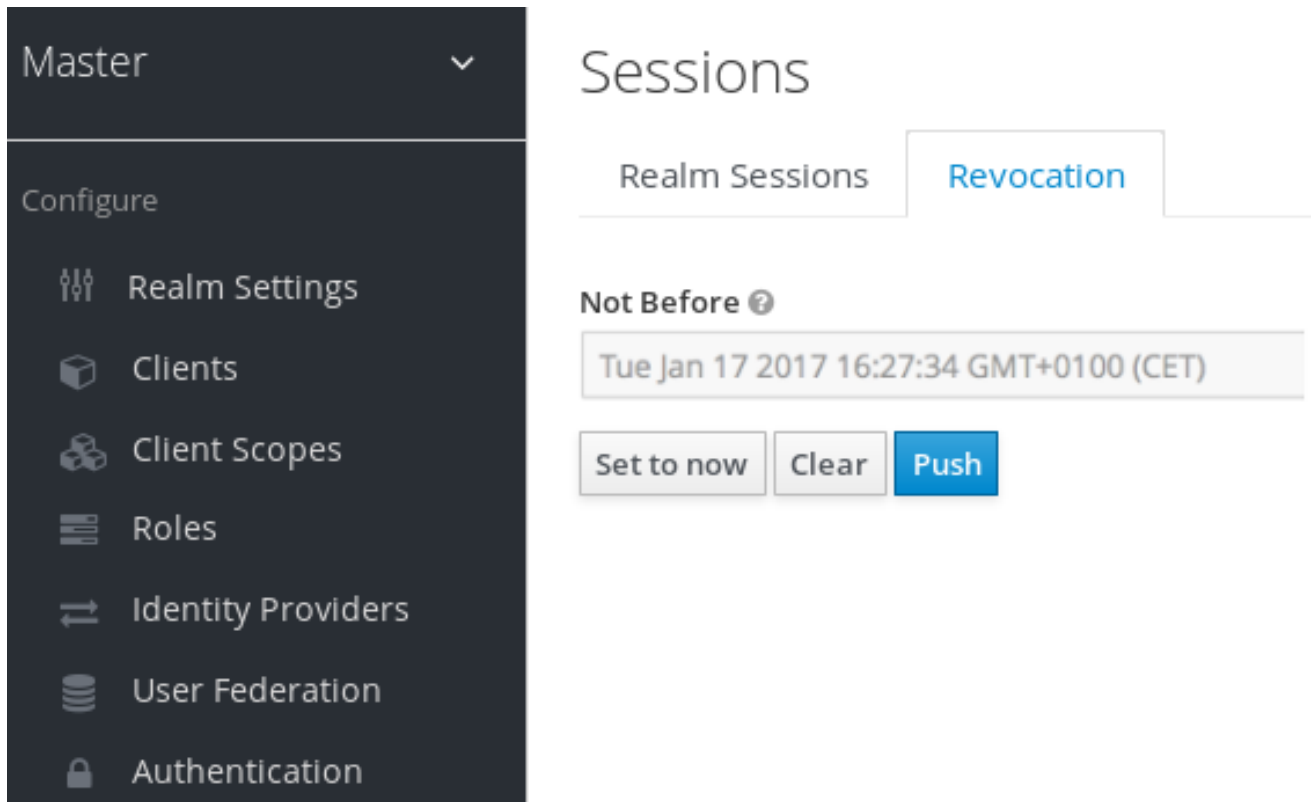
[Log out all sessions](#)

IP Address	Started	Last Access	Clients	Action
127.0.0.1	Feb 21, 2020 12:53:01 PM	Feb 21, 2020 3:30:58 PM	security-admin-console	Logout

6.2. REVOCATION POLICIES

If your system is compromised, you can revoke all active sessions and access tokens by clicking the **Sessions** screen **Revocation** tab.

Revocation



The screenshot shows the administration console interface. On the left is a dark sidebar menu with a 'Master' header and a dropdown arrow. Below it is a 'Configure' section with several menu items: 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. The main content area is titled 'Sessions' and has two tabs: 'Realm Sessions' and 'Revocation'. The 'Revocation' tab is active. Below the tabs, there is a 'Not Before' label with a help icon, followed by a text input field containing the date and time 'Tue Jan 17 2017 16:27:34 GMT+0100 (CET)'. At the bottom of this section are three buttons: 'Set to now', 'Clear', and 'Push'.

You can specify a time and date where sessions or tokens issued before that time and date are invalid using this console. Click **Set to now** to set the policy to the current time and date. Click **Push** to push this revocation policy to any registered OIDC client with the Red Hat Single Sign-On OIDC client adapter.

6.3. SESSION AND TOKEN TIMEOUTS

Red Hat Single Sign-On includes control of the session, cookie, and token timeouts through the **Tokens** tab in the **Realm Settings** menu.

Tokens tab

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

Configure

- ⚙️ Realm Settings
- 👤 Clients
- 🔑 Client Scopes
- 📄 Roles
- 👤 Identity Providers
- 👤 User Federation
- 🔒 Authentication

Manage

- 👤 Groups
- 👤 Users
- 🕒 Sessions
- 📅 Events
- 📄 Import
- 📄 Export

Default Signature Algorithm ?

Revoke Refresh Token ? OFF

SSO Session Idle ? Minutes

SSO Session Max ? Hours

SSO Session Idle Remember Me ? Minutes

SSO Session Max Remember Me ? Minutes

Offline Session Idle ? Days

Offline Session Max Limited ? OFF

Access Token Lifespan ? Minutes

Access Token Lifespan For Implicit Flow ? Minutes

Client login timeout ? Minutes

Login timeout ? Minutes

Login action timeout ? Minutes

User-Initiated Action Lifespan ? Minutes

Default Admin-Initiated Action Lifespan ? Hours

Override User-Initiated Action Lifespan ? Minutes

Configuration	Description
Default Signature Algorithm	The default algorithm used to assign tokens for the realm.
Revoke Refresh Token	When ON , Red Hat Single Sign-On revokes refresh tokens and issues another token that the client must use. This action applies to OIDC clients performing the refresh token flow.
SSO Session Idle	This setting is for OIDC clients only. If a user is inactive for longer than this timeout, the user session is invalidated. This timeout value resets when clients request authentication or send a refresh token request. Red Hat Single Sign-On adds a window of time to the idle timeout before the session invalidation takes effect. See the note later in this section.
SSO Session Max	The maximum time before a user session expires.

Configuration	Description
SSO Session Idle Remember Me	This setting is similar to the standard SSO Session Idle configuration but specific to logins with Remember Me enabled. Users can specify longer session idle timeouts when they click Remember Me when logging in. This setting is an optional configuration and, if its value is not greater than zero, it uses the same idle timeout as the SSO Session Idle configuration.
SSO Session Max Remember Me	This setting is similar to the standard SSO Session Max but specific to Remember Me logins. Users can specify longer sessions when they click Remember Me when logging in. This setting is an optional configuration and, if its value is not greater than zero, it uses the same session lifespan as the SSO Session Max configuration.
Offline Session Idle	This setting is for offline access . The amount of time the session remains idle before Red Hat Single Sign-On revokes its offline token. Red Hat Single Sign-On adds a window of time to the idle timeout before the session invalidation takes effect. See the note later in this section.
Offline Session Max Limited	This setting is for offline access . If this flag is ON , Offline Session Max can control the maximum time the offline token remains active, regardless of user activity. If the flag is OFF , offline sessions never expire by lifespan; these session only expire by being idle. Once this option is activated, you can configure the Offline Session Max (global option at realm level) and Client Offline Session Max (specific client level option in the Advanced Settings tab).
Offline Session Max	This setting is for offline access , and it is the maximum time before Red Hat Single Sign-On revokes the corresponding offline token. This option controls the maximum amount of time the offline token remains active, regardless of user activity.
Client Offline Session Idle	This setting is for offline access . If a user is inactive for longer than this timeout, offline token requests bump the idle timeout. This setting specifies a shorter idle timeout of an offline token than the offline session idle. Users can override this setting for individual clients. This setting is an optional configuration and, when set to zero, uses the same idle timeout in the Offline Session Idle configuration.

Configuration	Description
Client Offline Session Max	This setting is for offline access . The maximum time before an offline token expires and invalidates. This setting specifies a shorter token timeout than an offline session timeout, but users can override it for individual clients. This setting is an optional configuration and, when set to zero, uses the same idle timeout in the Offline Session Max configuration.
Client Session Idle	Idle timeout for the client session. If the user is inactive for longer than this timeout, the client session is invalidated and the refresh token requests bump the idle timeout. This setting never affects the general SSO user session, which is unique. Note the SSO user session is the parent of zero or more client sessions. One client session is created for each client app where the user logs in. This value should specify a shorter idle timeout than the SSO Session Idle . Users can override it for individual clients in the Advanced Settings client tab. This setting is an optional configuration and, when set to zero, uses the same idle timeout in the SSO Session Idle configuration.
Client Session Max	The maximum time for a client session and before a refresh token expires and invalidates. As in the previous option, this setting never affects the SSO user session and should specify a shorter value than the SSO Session Max . Users can override it for individual clients in the Advanced Settings client tab. This setting is an optional configuration and, when set to zero, uses the same max timeout in the SSO Session Max configuration.
Access Token Lifespan	When Red Hat Single Sign-On creates an OIDC access token, this value controls the lifetime of the token.
Access Token Lifespan For Implicit Flow	With the Implicit Flow, Red Hat Single Sign-On does not provide a refresh token. A separate timeout exists for access tokens created by the Implicit Flow.

Configuration	Description
Client login timeout	The maximum time before clients must finish the Authorization Code Flow in OIDC.
Login timeout	The total time a logging in must take. If authentication takes longer than this time, the user must start the authentication process again.
Login action timeout	The Maximum time users can spend on any one page during the authentication process.
User-Initiated Action Lifespan	The maximum time before a user's action permission expires. Keep this value short because users generally react to self-created actions quickly.
Default Admin-Initiated Action Lifespan	The maximum time before an action permission sent to a user by an administrator expires. Keep this value long to allow administrators to send e-mails to offline users. An administrator can override the default timeout before issuing the token.
Override User-Initiated Action Lifespan	Specifies independent timeouts per individual operation (for example, e-mail verification, forgot password, user actions, and Identity Provider E-mail Verification). This value defaults to the value configured at <i>User-Initiated Action Lifespan</i> .



NOTE

For idle timeouts, a two-minute window of time exists that the session is active. For example, when you have the timeout set to 30 minutes, it will be 32 minutes before the session expires.

This action is necessary for some scenarios in cluster and cross-data center environments where the token refreshes on one cluster node a short time before the expiration and the other cluster nodes incorrectly consider the session as expired because they have not yet received the message about a successful refresh from the refreshing node.

6.4. OFFLINE ACCESS

During [offline access](#) logins, the client application requests an offline token instead of a refresh token. The client application saves this offline token and can use it for future logins if the user logs out. This action is useful if your application needs to perform offline actions on behalf of the user even when the user is not online. For example, a regular data backup.

The client application is responsible for persisting the offline token in storage and then using it to retrieve new access tokens from the Red Hat Single Sign-On server.

The difference between a refresh token and an offline token is that an offline token never expires and is not subject to the **SSO Session Idle** timeout and **SSO Session Max** lifespan. The offline token is valid

after a user logout or server restart. You must use the offline token for a refresh token action at least once per thirty days or for the value of the [Offline Session Idle](#).

If you enable [Offline Session Max Limited](#), offline tokens expire after 60 days even if you use the offline token for a refresh token action. You can change this value, [Offline Session Max](#), in the Admin Console.

When using offline access, client idle and max timeouts can be overridden at the [client level](#). The options **Client Offline Session Idle** and **Client Offline Session Max**, in the client **Advanced Settings** tab, allow you to have a shorter offline timeout for a specific application. Note that client session values also control the refresh token expiration but they never affect the global offline user SSO session. The option **Client Offline Session Max** is only evaluated in the client if [Offline Session Max Limited](#) is **Enabled** at the realm level.

If you enable the [Revoke Refresh Token](#) option, you can use each offline token once only. After refresh, you must store the new offline token from the refresh response instead of the previous one.

Users can view and revoke offline tokens that Red Hat Single Sign-On grants them in the [User Account Console](#). Administrators can revoke offline tokens for individual users in the Admin Console in the **Consents** tab. Administrators can view all offline tokens issued in the **Offline Access** tab of each client. Administrators can revoke offline tokens by setting a [revocation policy](#).

To issue an offline token, users must have the role mapping for the realm-level **offline_access** role. Clients must also have that role in their scope. Clients must add an **offline_access** client scope as an **Optional client scope** to the role, which is done by default.

Clients can request an offline token by adding the parameter **scope=offline_access** when sending their authorization request to Red Hat Single Sign-On. The Red Hat Single Sign-On OIDC client adapter automatically adds this parameter when you use it to access your application's secured URL (such as, `http://localhost:8080/customer-portal/secured?scope=offline_access`). The Direct Access Grant and Service Accounts support offline tokens if you include **scope=offline_access** in the authentication request body.

6.5. OFFLINE SESSIONS PRELOADING

In addition to Infinispan caches, offline sessions are stored in a database which means they will be available even after server restart. By default, the offline sessions are not preloaded from the database into the Infinispan caches during the server startup, because this approach has a drawback if there are many offline sessions to be preloaded. It can significantly slow down the server startup time. Therefore, the offline sessions are lazily fetched from the database by default.

However, Red Hat Single Sign-On can be configured to preload the offline sessions from the database into the Infinispan caches during the server startup. It can be achieved by setting **preloadOfflineSessionsFromDatabase** property in the **userSessions** SPI to **true**.

The following example shows how to configure offline sessions preloading.

```
<subsystem xmlns="urn:jboss:domain:keycloak-server:1.2">
  ...
  <spi name="userSessions">
    <default-provider>infinispan</default-provider>
    <provider name="infinispan" enabled="true">
      <properties>
        <property name="preloadOfflineSessionsFromDatabase" value="true"/>
      </properties>
    </provider>
  </spi>
</subsystem>
```

```
</spi>  
...  
</subsystem>
```

Equivalent configuration using CLI commands:

```
/subsystem=keycloak-server/spi=userSessions:add(default-provider=infinispan)  
/subsystem=keycloak-server/spi=userSessions/provider=infinispan:add(properties=  
{preloadOfflineSessionsFromDatabase => "true"},enabled=true)
```

6.6. TRANSIENT SESSIONS

You can conduct transient sessions in Red Hat Single Sign-On. When using transient sessions, Red Hat Single Sign-On does not create a user session after successful authentication. Red Hat Single Sign-On creates a temporary, transient session for the scope of the current request that successfully authenticates the user. Red Hat Single Sign-On can run [protocol mappers](#) using transient sessions after authentication.

During transient sessions, the client application cannot refresh tokens, introspect tokens, or validate a specific session. Sometimes these actions are unnecessary, so you can avoid the additional resource use of persisting user sessions. This session saves performance, memory, and network communication (in cluster and cross-data center environments) resources.

CHAPTER 7. ASSIGNING PERMISSIONS AND ACCESS USING ROLES AND GROUPS

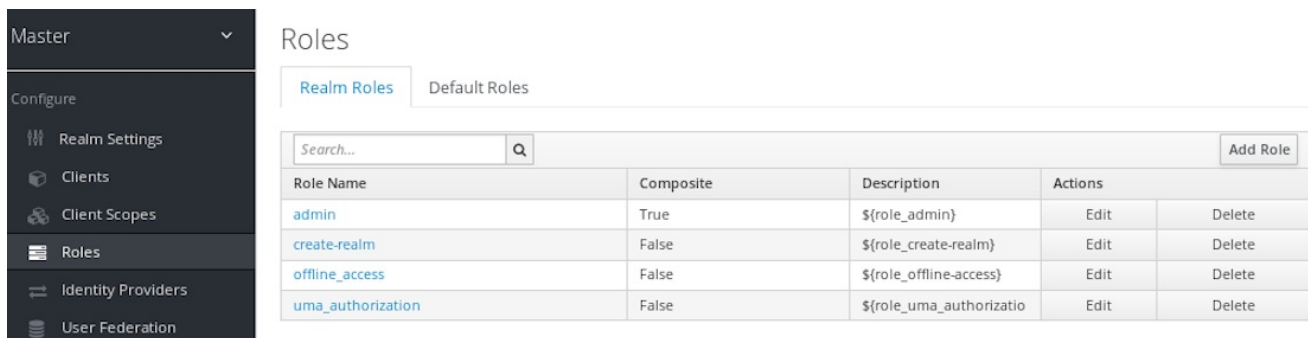
Roles and groups have a similar purpose, which is to give users access and permissions to use applications. Groups are a collection of users to which you apply roles and attributes. Roles define specific applications permissions and access control.

A role typically applies to one type of user. For example, an organization may include **admin**, **user**, **manager**, and **employee** roles. An application can assign access and permissions to a role and then assign multiple users to that role so the users have the same access and permissions. For example, the Admin Console has roles that give permission to users to access different parts of the Admin Console.

There is a global namespace for roles and each client also has its own dedicated namespace where roles can be defined.

7.1. CREATING A REALM ROLE

Realm-level roles are a namespace for defining your roles. To see the list of roles, click **Roles** in the menu.



Role Name	Composite	Description	Actions	
admin	True	`\${role_admin}`	Edit	Delete
create-realm	False	`\${role_create-realm}`	Edit	Delete
offline_access	False	`\${role_offline-access}`	Edit	Delete
uma_authorization	False	`\${role_uma_authorizatio}`	Edit	Delete

Procedure

1. Click **Add Role**.
2. Enter a **Role Name**.
3. Enter a **Description**.
4. Click **Save**.

Add role

The screenshot shows the 'Developer' role configuration page. The left sidebar has a 'Roles' menu item highlighted. The main content area has a breadcrumb 'Roles > developer' and a title 'Developer' with a trash icon. Below the title are three tabs: 'Details' (active), 'Permissions', and 'Users in Role'. The 'Details' tab contains a 'Role Name' field with the value 'developer', a 'Description' text area, and a 'Composite Roles' toggle switch set to 'OFF'. At the bottom are 'Save' and 'Cancel' buttons.

The **description** field can be localizable by specifying a substitution variable with `${var-name}` strings. The localized value is configured to your theme within the themes property files. See the [Server Developer Guide](#) for more details.

7.2. CLIENT ROLES

Client roles are namespaces dedicated to clients. Each client gets its own namespace. Client roles are managed under the **Roles** tab for each client. You interact with this UI the same way you do for realm-level roles.

7.3. CONVERTING A ROLE TO A COMPOSITE ROLE

Any realm or client level role can become a *composite role*. A *composite role* is a role that has one or more additional roles associated with it. When a composite role is mapped to a user, the user gains the roles associated with the composite role. This inheritance is recursive so users also inherit any composite of composites. However, we recommend that composite roles are not overused.

Procedure

1. Click **Roles** in the menu.
2. Click the role that you want to convert.
3. Toggle **Composite Roles** to **ON**.

Composite role

The role selection UI is displayed on the page and you can associate realm level and client level roles to the composite role you are creating.

In this example, the **employee** realm-level role is associated with the **developer** composite role. Any user with the **developer** role also inherits the **employee** role.



NOTE

When creating tokens and SAML assertions, any composite also has its associated roles added to the claims and assertions of the authentication response sent back to the client.

7.4. ASSIGNING ROLE MAPPINGS

You can assign role mappings to a user through the **Role Mappings** tab for that user.

Procedure

1. Click **Users** in the menu.
2. Click the user that you want to perform a role mapping on. If the user is not displayed, click **View all users**.
3. Click the **Role Mappings** tab.
4. Click the role you want to assign to the user in the **Available Roles** box.
5. Click **Add selected**.

Role mappings

The screenshot shows the user management interface for user 'johndoe'. The 'Role Mappings' tab is active. The 'Available Roles' list contains 'admin', 'create-realm', 'developer', and 'employee'. The 'Assigned Roles' list contains 'offline_access' and 'uma_authorization'. The 'Effective Roles' list contains 'offline_access' and 'uma_authorization'. The 'Client Roles' dropdown is set to 'Select a client...'.

In the preceding example, we are assigning the composite role **developer** to a user. That role was created in the [Composite Roles](#) topic.

Effective role mappings

The screenshot shows the user management interface for user 'johndoe'. The 'Role Mappings' tab is active. The 'Available Roles' list contains 'admin' and 'create-realm'. The 'Assigned Roles' list contains 'developer', 'offline_access', and 'uma_authorization'. The 'Effective Roles' list contains 'developer', 'employee', 'offline_access', and 'uma_authorization'. The 'Client Roles' dropdown is set to 'Select a client...'.

When the **developer** role is assigned, the **employee** role associated with the **developer** composite is displayed in the **Effective Roles** box. **Effective Roles** are the roles explicitly assigned to users and roles that are inherited from composites.

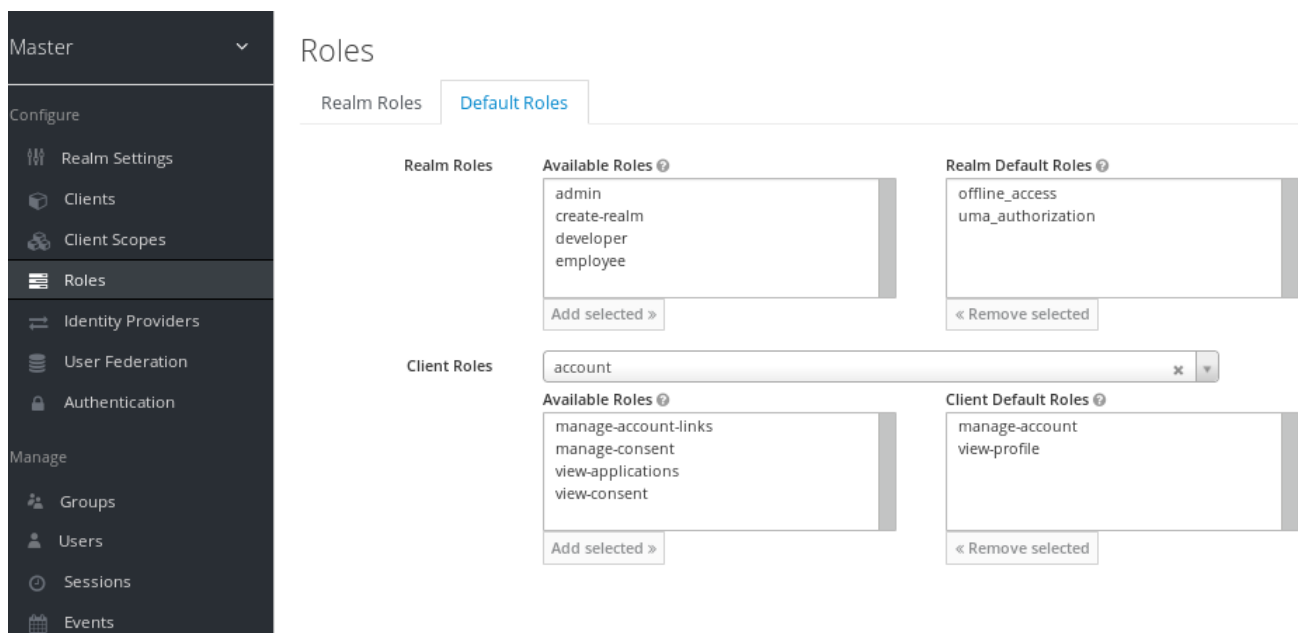
7.5. USING DEFAULT ROLES

Use default roles to automatically assign user role mappings when a user is created or imported through [Identity Brokering](#).

Procedure

1. Click **Roles** in the menu
2. Click the **Default Roles** tab.

Default roles



This screenshot shows that some *default roles* already exist.

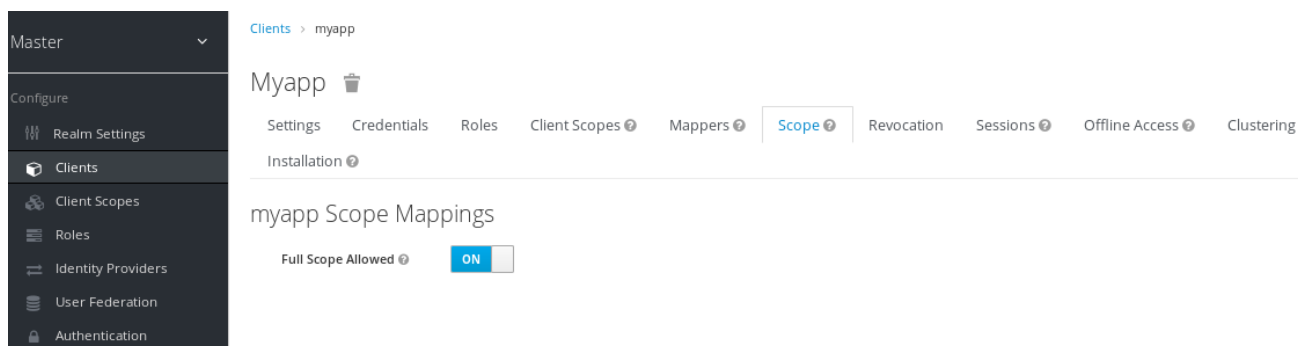
7.6. ROLE SCOPE MAPPINGS

On creation of an OIDC access token or SAML assertion, the user role mappings become claims within the token or assertion. Applications use these claims to make access decisions on the resources controlled by the application. Red Hat Single Sign-On digitally signs access tokens and applications re-use them to invoke remotely secured REST services. However, these tokens have an associated risk. An attacker can obtain these tokens and use their permissions to compromise your networks. To prevent this situation, use *Role Scope Mappings*.

Role Scope Mappings limit the roles declared inside an access token. When a client requests a user authentication, the access token they receive contains only the role mappings that are explicitly specified for the client's scope. The result is that you limit the permissions of each individual access token instead of giving the client access to all the users permissions.

By default, each client gets all the role mappings of the user. You can view the role mappings in the **Scope** tab of each client.

Full scope



By default, the effective roles of scopes are every declared role in the realm. To change this default behavior, toggle **Full Scope Allowed** to **ON** and declare the specific roles you want in each client. You can also use [client scopes](#) to define the same role scope mappings for a set of clients.

Partial scope

7.7. GROUPS

Groups in Red Hat Single Sign-On manage a common set of attributes and role mappings for each user. Users can be members of any number of groups and inherit the attributes and role mappings assigned to each group.

To manage groups, click **Groups** in the menu.

Groups

Groups are hierarchical. A group can have multiple subgroups but a group can have only one parent. Subgroups inherit the attributes and role mappings from their parent. Users inherit the attributes and role mappings from their parent as well.

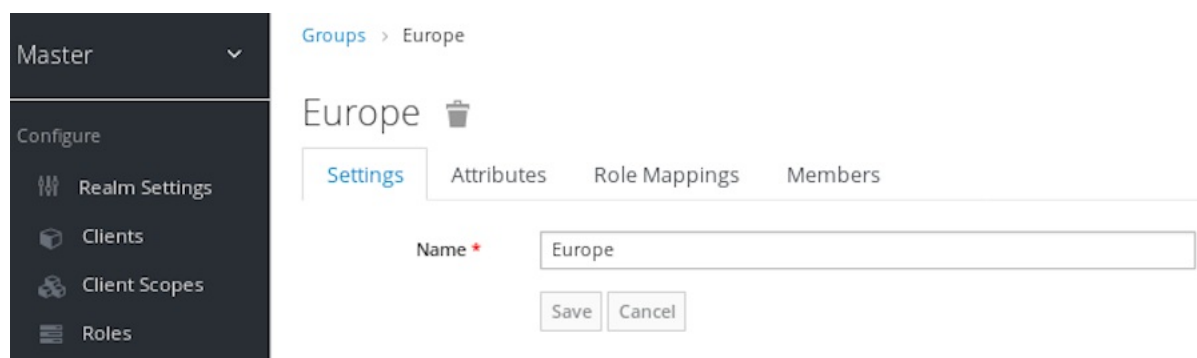
If you have a parent group and a child group, and a user that belongs only to the child group, the user in the child group inherits the attributes and role mappings of both the parent group and the child group.

The following example includes a top-level **Sales** group and a child **North America** subgroup.

To add a group:

1. Click the group.
2. Click **New**.
3. Select the **Groups** icon in the tree to make a top-level group.
4. Enter a group name in the **Create Group** screen.
5. Click **Save**.
The group management page is displayed.

Group

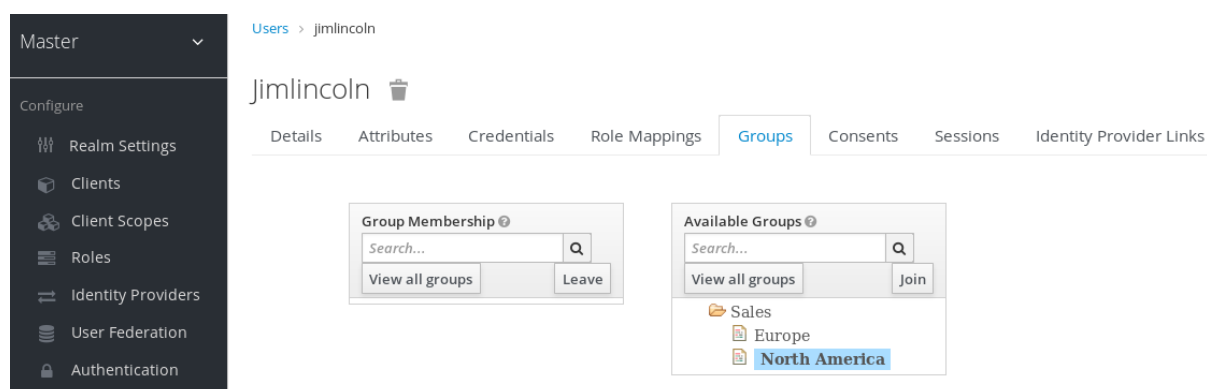


Attributes and role mappings you define are inherited by the groups and users that are members of the group.

To add a user to a group:

1. Click **Users** in the menu.
2. Click the user that you want to perform a role mapping on. If the user is not displayed, click **View all users**.
3. Click **Groups**.

User groups



4. Select a group from the **Available Groups** tree.
5. Click **Join**.

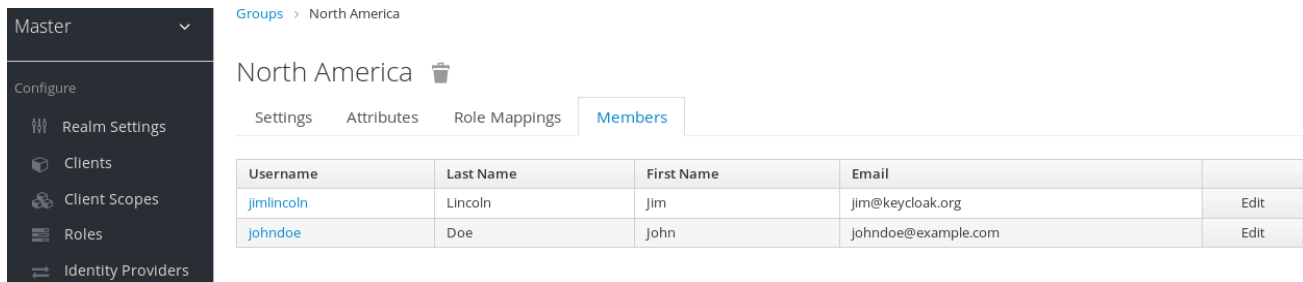
To remove a group from a user:

1. Select the group from the **Group Membership** tree.


2. Click **Leave**.

In this example, the user *jimlincoln* is in the *North America* group. You can see *jimlincoln* displayed under the **Members** tab for the group.

Group membership



Groups > North America

North America 

Settings Attributes Role Mappings **Members**

Username	Last Name	First Name	Email	
jimlincoln	Lincoln	Jim	jim@keycloak.org	Edit
johndoe	Doe	John	johndoe@example.com	Edit

7.7.1. Groups compared to roles

Groups and roles have some similarities and differences. In Red Hat Single Sign-On, groups are a collection of users to which you apply roles and attributes. Roles define types of users and applications assign permissions and access control to roles.

Composite Roles are similar to Groups as they provide the same functionality. The difference between them is conceptual. Composite roles apply the permission model to a set of services and applications. Use composite roles to manage applications and services.

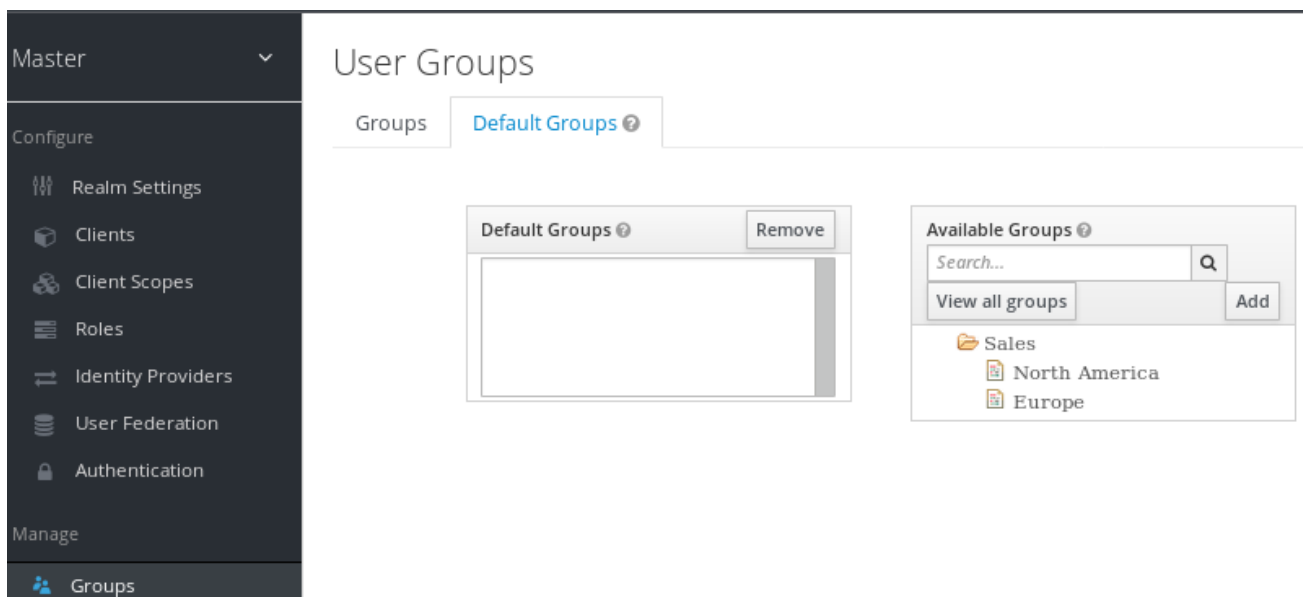
Groups focus on collections of users and their roles in an organization. Use groups to manage users.


7.7.2. Using default groups

To automatically assign group membership to any users who is created or who is imported through **Identity Brokering**, you use default groups.

1. Click **Groups** in the menu.
2. Click the **Default Groups** tab.

Default groups



Master 


Configure


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

Manage


- Groups**


User Groups

Groups **Default Groups** 




Default Groups 

[Remove](#)

Available Groups 

Search... 

[View all groups](#) [Add](#)

-  Sales
-  North America
-  Europe

This screenshot shows that some *default groups* already exist.

CHAPTER 8. CONFIGURING AUTHENTICATION

This chapter covers several authentication topics. These topics include:

- Enforcing strict password and One Time Password (OTP) policies.
- Managing different credential types.
- Logging in with Kerberos.
- Disabling and enabling built-in credential types.

8.1. PASSWORD POLICIES

When Red Hat Single Sign-On creates a realm, it does not associate password policies with the realm. You can set a simple password with no restrictions on its length, security, or complexity. Simple passwords are unacceptable in production environments. Red Hat Single Sign-On has a set of password policies available through the Admin Console.

Procedure

1. Click **Authentication** in the menu.
2. Click the **Password Policy** tab.
3. Select the policy to add in the **Add policy** drop-down box.
4. Enter a value for the **Policy Value** corresponding with the policy chosen.
5. Click **Save**.

Password policy

Master

Configure

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

Authentication

Flows Bindings Required Actions **Password Policy** OTP Policy WebAuthn Policy

Add policy...

Policy Type	Policy Value	Actions
Hashing Iterations	27500	Delete
Minimum Length	6	Delete

Save Cancel

After saving the policy, Red Hat Single Sign-On enforces the policy for new users and sets an Update Password action for existing users to ensure they change their password the next time they log in. For example:

Failed password policy

Update password



Invalid password: minimum length 6.

New Password

Confirm password

Submit

8.1.1. Password policy types

8.1.1.1. Hashing algorithm

Passwords are not stored in clear text. Before storage or validation, Red Hat Single Sign-On hashes passwords using standard hashing algorithms Red Hat Single Sign-On that support the PBKDF2, PBKDF2-SHA256 and PBKDF-SHA512 hashing algorithms.

8.1.1.2. Hashing iterations

Specifies the number of times Red Hat Single Sign-On hashes passwords before storage or verification. The default value is 27,500.

Red Hat Single Sign-On hashes passwords to ensure that hostile actors with access to the password database cannot read passwords through reverse engineering.



NOTE

A high hashing iteration value can impact performance as it requires higher CPU power.

8.1.1.3. Digits

The number of numerical digits required in the password string.

8.1.1.4. Lowercase characters

The number of lower case letters required in the password string.

8.1.1.5. Uppercase characters

The number of upper case letters required in the password string.

8.1.1.6. Special characters

The number of special characters required in the password string.

8.1.1.7. Not username

The password cannot be the same as the username.

8.1.1.8. Not email

The password cannot be the same as the email address of the user.

8.1.1.9. Regular expression

Password must match one or more defined regular expression patterns.

8.1.1.10. Expire password

The number of days the password is valid. When the number of days has expired, the user must change their password.

8.1.1.11. Not recently used

Password cannot be already used by the user. Red Hat Single Sign-On stores a history of used passwords. The number of old passwords stored is configurable in Red Hat Single Sign-On.

8.1.1.12. Password blacklist

Password must not be in a blacklist file.

- Blacklist files are UTF-8 plain-text files with Unix line endings. Every line represents a blacklisted password.
- Red Hat Single Sign-On compares passwords in a case-insensitive manner. All passwords in the blacklist must be lowercase.
- The value of the blacklist file must be the name of the blacklist file.
- Blacklist files resolve against **`${jboss.server.data.dir}/password-blacklists/`** by default. Customize this path using:
 - The **`keycloak.password.blacklists.path`** property.
 - The **`blacklistsPath`** property of the **`passwordBlacklist`** policy SPI configuration.

8.2. ONE TIME PASSWORD (OTP) POLICIES

Red Hat Single Sign-On has several policies for setting up a FreeOTP or Google Authenticator One-Time Password generator. Click the **Authentication** menu and click the **OTP Policy** tab.

Otp policy

The screenshot shows the 'Authentication' configuration page in Red Hat Single Sign-On. The left sidebar is open to the 'Authentication' menu. The main content area is titled 'Authentication' and has several tabs: 'Flows', 'Bindings', 'Required Actions', 'Password Policy', 'OTP Policy' (selected), and 'WebAuthn Policy'. Under the 'OTP Policy' tab, the following settings are visible:

- OTP Type**: Time Based
- OTP Hash Algorithm**: SHA1
- Number of Digits**: 6
- Look Ahead Window**: 1
- OTP Token Period**: 30
- Supported Applications**: FreeOTP, Google Authenticator

Red Hat Single Sign-On generates a QR code on the OTP set-up page, based on information configured in the **OTP Policy** tab. FreeOTP and Google Authenticator scan the QR code when configuring OTP.

8.2.1. Time-based or counter-based one time passwords

The algorithms available in Red Hat Single Sign-On for your OTP generators are time-based and counter-based.

With Time-Based One Time Passwords (TOTP), the token generator will hash the current time and a shared secret. The server validates the OTP by comparing the hashes within a window of time to the submitted value. TOTPs are valid for a short window of time.

With Counter-Based One Time Passwords (HOTP), Red Hat Single Sign-On uses a shared counter rather than the current time. The Red Hat Single Sign-On server increments the counter with each successful OTP login. Valid OTPs change after a successful login.

TOTP is more secure than HOTP because the matchable OTP is valid for a short window of time, while the OTP for HOTP is valid for an indeterminate amount of time. HOTP is more user-friendly than TOTP because no time limit exists to enter the OTP.

HOTP requires a database update every time the server increments the counter. This update is a performance drain on the authentication server during heavy load. To increase efficiency, TOTP does not remember passwords used, so there is no need to perform database updates. The drawback is that it is possible to re-use TOTPs in the valid time interval.

8.2.2. TOTP configuration options

8.2.2.1. OTP hash algorithm

The default algorithm is SHA1. The other, more secure options are SHA256 and SHA512.

8.2.2.2. Number of digits

The length of the OTP. Short OTP's are user-friendly, easier to type, and easier to remember. Longer OTP's are more secure than shorter OTP's.

8.2.2.3. Look around window

The number of intervals the server attempts to match the hash. This option is present in Red Hat Single Sign-On if the clock of the TOTP generator or authentication server becomes out-of-sync. The default value of 1 is adequate. For example, if the time interval for a token is 30 seconds, the default value of 1 means it will accept valid tokens in the 90-second window (time interval 30 seconds + look ahead 30 seconds + look behind 30 seconds). Every increment of this value increases the valid window by 60 seconds (look ahead 30 seconds + look behind 30 seconds).

8.2.2.4. OTP token period

The time interval in seconds the server matches a hash. Each time the interval passes, the token generator generates a TOTP.

8.2.3. HOTP configuration options

8.2.3.1. OTP hash algorithm

The default algorithm is SHA1. The other, more secure options are SHA256 and SHA512.

8.2.3.2. Number of digits

The length of the OTP. Short OTPs are user-friendly, easier to type, and easier to remember. Longer OTPs are more secure than shorter OTPs.

8.2.3.3. Look ahead window

The number of intervals the server attempts to match the hash. This option is present in Red Hat Single Sign-On if the clock of the TOTP generator or authentication server become out-of-sync. The default value of 1 is adequate. This option is present in Red Hat Single Sign-On to cover when the user's counter gets ahead of the server.

8.2.3.4. Initial counter

The value of the initial counter.

8.3. AUTHENTICATION FLOWS

An *authentication flow* is a container of authentications, screens, and actions, during log in, registration, and other Red Hat Single Sign-On workflows. To view all the flows, actions, and checks, each flow requires:

Procedure

1. Click **Authentication** in the menu.
2. Click the **Flows** tab.

8.3.1. Built-in flows

Red Hat Single Sign-On has several built-in flows. You cannot modify these flows, but you can alter the flow's requirements to suit your needs.

In the drop-down list, select **browser** to display the Browser Flow screen.

Browser flow

Auth Type		Requirement				
Cookie		<input type="radio"/> REQUIRED	<input checked="" type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED		
Kerberos		<input type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input checked="" type="radio"/> DISABLED		
Identity Provider Redirector		<input type="radio"/> REQUIRED	<input checked="" type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED		Actions ▾
Forms	Username Password Form	<input checked="" type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED	<input type="radio"/> CONDITIONAL	
	Browser - Conditional OTP	<input type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED	<input checked="" type="radio"/> CONDITIONAL	
	Condition - User Configured	<input checked="" type="radio"/> REQUIRED	<input type="radio"/> DISABLED			
	OTP Form	<input checked="" type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED		

Hover over the question-mark tooltip of the drop-down list to view a description of the flow. Two sections exist.

8.3.1.1. Auth type

The name of the authentication or the action to execute. If an authentication is indented, it is in a sub-flow. It may or may not be executed, depending on the behavior of its parent.

1. Cookie

The first time a user logs in successfully, Red Hat Single Sign-On sets a session cookie. If the cookie is already set, this authentication type is successful. Since the cookie provider returned success and each execution at this level of the flow is *alternative*, Red Hat Single Sign-On does not perform any other execution. This results in a successful login.

2. Kerberos

This authenticator is disabled by default and is skipped during the Browser Flow.

3. Identity Provider Redirector

This action is configured through the **Actions > Config** link. It redirects to another IdP for [identity brokering](#).

4. Forms

Since this sub-flow is marked as *alternative*, it will not be executed if the **Cookie** authentication type passed. This sub-flow contains an additional authentication type that needs to be executed. Red Hat Single Sign-On loads the executions for this sub-flow and processes them.

The first execution is the **Username Password Form**, an authentication type that renders the username and password page. It is marked as *required*, so the user must enter a valid username and password.

The second execution is the **Browser - Conditional OTP** sub-flow. This sub-flow is *conditional* and executes depending on the result of the **Condition - User Configured** execution. If the result is true, Red Hat Single Sign-On loads the executions for this sub-flow and processes them.

The next execution is the **Condition - User Configured** authentication. This authentication checks if Red Hat Single Sign-On has configured other executions in the flow for the user. The **Browser - Conditional OTP** sub-flow executes only when the user has a configured OTP credential.

The final execution is the **OTP Form**. Red Hat Single Sign-On marks this execution as *required* but it runs only when the user has an OTP credential set up because of the setup in the *conditional* sub-flow. If not, the user does not see an OTP form.

8.3.1.2. Requirement

A set of radio buttons that control the execution of an action executes.

8.3.1.2.1. Required

All *Required* elements in the flow must be successfully sequentially executed. The flow terminates if a required element fails.

8.3.1.2.2. Alternative

Only a single element must successfully execute for the flow to evaluate as successful. Because the *Required* flow elements are sufficient to mark a flow as successful, any *Alternative* flow element within a flow containing *Required* flow elements will not execute.

8.3.1.2.3. Disabled

The element does not count to mark a flow as successful.

8.3.1.2.4. Conditional

This requirement type is only set on sub-flows.

- A *Conditional* sub-flow contains executions. These executions must evaluate to logical statements.
- If all executions evaluate as *true*, the *Conditional* sub-flow acts as *Required*.
- If all executions evaluate as *false*, the *Conditional* sub-flow acts as *Disabled*.
- If you do not set an execution, the *Conditional* sub-flow acts as *Disabled*.
- If a flow contains executions and the flow is not set to *Conditional*, Red Hat Single Sign-On does not evaluate the executions, and the executions are considered functionally *Disabled*.

8.3.2. Creating flows

Important functionality and security considerations apply when you design a flow.

To create a flow, perform the following:

Procedure

1. Click **Authentication** in the menu.
2. Click **New**.



NOTE

You can copy and then modify an existing flow. Select a flow, click **Copy**, and enter a name for the new flow.

When creating a new flow, you must create a top-level flow first with the following options:

Alias

The name of the flow.

Description

The description you can set to the flow.

Top-Level Flow Type

The type of flow. The type **client** is used only for the authentication of clients (applications). For all other cases, choose **generic**.

Create a top-level flow

When Red Hat Single Sign-On has created the flow, Red Hat Single Sign-On displays the **Delete**, **Add execution**, and **Add flow** buttons.

An empty new flow

Three factors determine the behavior of flows and sub-flows.

- The structure of the flow and sub-flows.
- The executions within the flows
- The requirements set within the sub-flows and the executions.

Executions have a wide variety of actions, from sending a reset email to validating an OTP. Add executions with the **Add execution** button. Hover over the question mark next to **Provider**, to see a description of the execution.

Adding an authentication execution

The screenshot shows the 'Create Authenticator Execution' page in the Red Hat Single Sign-On 7.6 administration console. The left sidebar is open to the 'Authentication' section. The main content area has tabs for 'Flows', 'Bindings', 'Required Actions', 'Password Policy', 'OTP Policy', and 'WebAuthn Policy'. The 'Flows' tab is active, and a 'Provider' dropdown menu is open, showing a list of execution types. The 'Choose User' option is highlighted in blue. Other options in the list include 'Browser Redirect/Refresh', 'Automatically Set Existing User', 'Basic Auth Challenge', 'Basic Auth Password+OTP', 'Conditional Block - User Configured', 'Conditional Block - User Role', 'Conditional OTP Form', 'Confirm Link Existing Account', 'Cookie', 'Create User If Unique', 'Docker Authenticator', 'HTTP Basic Authentication', 'Identity Provider Redirector', 'Kerberos', 'OTP', 'OTP Form', 'Password', 'Password Form', and 'Reset OTP'.

Two types of executions exist, *automatic executions* and *interactive executions*. *Automatic executions* are similar to the **Cookie** execution and will automatically perform their action in the flow. *Interactive executions* halt the flow to get input. Executions executing successfully set their status to *success*. For a flow to complete, it needs at least one execution with a status of *success*.

You can add sub-flows to top-level flows with the **Add flow** button. The **Add flow** button displays the **Create Execution Flow** page. This page is similar to the **Create Top Level Form** page. The difference is that the **Flow Type** can be **generic** (default) or **form**. The **form** type constructs a sub-flow that generates a form for the user, similar to the built-in **Registration** flow. Sub-flows success depends on how their executions evaluate, including their contained sub-flows. See the [execution requirements section](#) for an in-depth explanation of how sub-flows work.



NOTE

After adding an execution, check the requirement has the correct value.

All elements in a flow have a **Delete** option in the **Actions** menu. This action removes the element from the flow. Executions have a **Config** menu option to configure the execution. It is also possible to add executions and sub-flows to sub-flows with the **Add execution** and **Add flow** menu options.

Since the order of execution is important, you can move executions and sub-flows up and down within their flows using the up and down buttons beside their names.

**WARNING**

Make sure to properly test your configuration when you configure the authentication flow to confirm that no security holes exist in your setup. We recommend that you test various corner cases. For example, consider testing the authentication behavior for a user when you remove various credentials from the user's account before authentication.

As an example, when 2nd-factor authenticators, such as OTP Form or WebAuthn Authenticator, are configured in the flow as **REQUIRED** and the user does not have credential of particular type, the user will be able to setup the particular credential during authentication itself. This situation means that the user does not authenticate with this credential as he setup it right during the authentication. So for browser authentication, make sure to configure your authentication flow with some 1st-factor credentials such as Password or WebAuthn Passwordless Authenticator.

8.3.3. Creating a password-less browser login flow

To illustrate the creation of flows, this section describes creating an advanced browser login flow. The purpose of this flow is to allow a user a choice between logging in using a password-less manner with [WebAuthn](#), or two-factor authentication with a password and OTP.

Procedure

1. Click **Authentication** in the menu.
2. Click the **Flows** tab.
3. Click **New**.
4. Enter **Browser Password-less** as an alias.
5. Click **Save**.
6. Click **Add execution**.
7. Select **Cookie** from the drop-down list.
8. Click **Save**.
9. Click **Alternative** for the **Cookie** authentication type to set its requirement to alternative.
10. Click **Add execution**.
11. Select **Kerberos** from the drop-down list.
12. Click **Add execution**.
13. Select **Identity Provider Redirector** from the drop-down list.
14. Click **Save**.

15. Click **Alternative** for the **Identity Provider Redirector** authentication type to set its requirement to alternative.
16. Click **Add flow**.
17. Enter **Forms** as an alias.
18. Click **Save**.
19. Click **Alternative** for the **Forms** authentication type to set its requirement to alternative.

The common part with the browser flow

Authentication

Authentication					
Flows					
Browser Password-less					
New Copy Delete Add execution Add flow					
Auth Type	Requirement				
Cookie	<input type="radio"/> REQUIRED	<input checked="" type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED		Actions
Kerberos	<input type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input checked="" type="radio"/> DISABLED		Actions
Identity Provider Redirector	<input type="radio"/> REQUIRED	<input checked="" type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED		Actions
Forms	<input type="radio"/> REQUIRED	<input checked="" type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED	<input type="radio"/> CONDITIONAL	Actions

20. Click **Actions** for the **Forms** execution.
21. Select **Add execution**.
22. Select **Username Form** from the drop-down list.
23. Click **Save**.
24. Click **Required** for the **Username Form** authentication type to set its requirement to required.

At this stage, the form requires a username but no password. We must enable password authentication to avoid security risks.

1. Click **Actions** for the **Forms** sub-flow.
2. Click **Add flow**.
3. Enter **Authentication** as an alias.
4. Click **Save**.
5. Click **Required** for the **Authentication** authentication type to set its requirement to required.
6. Click **Actions** for the **Authentication** sub-flow.
7. Click **Add execution**.
8. Select **Webauthn Passwordless Authenticator** from the drop-down list.
9. Click **Save**.
10. Click **Alternative** for the **Webauthn Passwordless Authenticator** authentication type to set its requirement to alternative.
11. Click **Actions** for the **Authentication** sub-flow.

12. Click **Add flow**.
13. Enter **Password with OTP** as an alias.
14. Click **Save**.
15. Click **Alternative** for the **Password with OTP** authentication type to set its requirement to alternative.
16. Click **Actions** for the **Password with OTP** sub-flow.
17. Click **Add execution**.
18. Select **Password Form** from the drop-down list.
19. Click **Save**.
20. Click **Required** for the **Password Form** authentication type to set its requirement to required.
21. Click **Actions** for the **Password with OTP** sub-flow.
22. Click **Add execution**.
23. Select **OTP Form** from the drop-down list.
24. Click **Save**.
25. Click **Required** for the **OTP Form** authentication type to set its requirement to required.

Finally, change the bindings.

1. Click the **Bindings** tab.
2. Click the **Browser Flow** drop-down list.
3. Select **Browser Password-less** from the drop-down list.
4. Click **Save**.

A password-less browser login

Authentication

Browser Password-less					New	Copy	Delete	Add execution	Add flow
Auth Type				Requirement					
<input type="checkbox"/>	<input type="checkbox"/>	Cookie		<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>			Actions
				REQUIRED	ALTERNATIVE	DISABLED			▼
<input type="checkbox"/>	<input type="checkbox"/>	Kerberos		<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>			Actions
				REQUIRED	ALTERNATIVE	DISABLED			▼
<input type="checkbox"/>	<input type="checkbox"/>	Identity Provider Redirector		<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>			Actions
				REQUIRED	ALTERNATIVE	DISABLED			▼
<input type="checkbox"/>	<input type="checkbox"/>	Forms		<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>		Actions
				REQUIRED	ALTERNATIVE	DISABLED	CONDITIONAL		▼
	<input type="checkbox"/>	Username Form		<input checked="" type="radio"/>					Actions
				REQUIRED					▼
	<input type="checkbox"/>	Authentication		<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>		Actions
				REQUIRED	ALTERNATIVE	DISABLED	CONDITIONAL		▼
		<input type="checkbox"/>	WebAuthn Passwordless Authenticator	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>			Actions
				REQUIRED	ALTERNATIVE	DISABLED			▼
		<input type="checkbox"/>	Password With OTP	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>		Actions
				REQUIRED	ALTERNATIVE	DISABLED	CONDITIONAL		▼
		<input type="checkbox"/>	Password Form	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>			Actions
				REQUIRED	ALTERNATIVE	DISABLED			▼
		<input type="checkbox"/>	OTP Form	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>			Actions
				REQUIRED	ALTERNATIVE	DISABLED			▼

After entering the username, the flow works as follows:

If users have WebAuthn passwordless credentials recorded, they can use these credentials to log in directly. This is the password-less login. The user can also select **Password with OTP** because the **WebAuthn Passwordless** execution and the **Password with OTP** flow are set to **Alternative**. If they are set to **Required**, the user has to enter WebAuthn, password, and OTP.

If the user selects the **Try another way** link with **WebAuthn passwordless** authentication, the user can choose between **Password** and **Security Key** (WebAuthn passwordless). When selecting the password, the user will need to continue and log in with the assigned OTP. If the user has no WebAuthn credentials, the user must enter the password and then the OTP. If the user has no OTP credential, they will be asked to record one.



NOTE

Since the WebAuthn Passwordless execution is set to **Alternative** rather than **Required**, this flow will never ask the user to register a WebAuthn credential. For a user to have a Webauthn credential, an administrator must add a required action to the user. Do this by:

1. Enabling the **Webauthn Register Passwordless** required action in the realm (see the [WebAuthn](#) documentation).
2. Setting the required action using the **Credential Reset** part of a user's [Credentials](#) management menu.

Creating an advanced flow such as this can have side effects. For example, if you enable the ability to reset the password for users, this would be accessible from the password form. In the default **Reset Credentials** flow, users must enter their username. Since the user has already entered a username earlier in the **Browser Password-less** flow, this action is unnecessary for Red Hat Single Sign-On and sub-optimal for user experience. To correct this problem, you can:

- Copy the **Reset Credentials** flow. Set its name to **Reset Credentials for password-less**, for example.
- Select **Delete** in the **Actions** menu of the **Choose user** execution.
- In the **Bindings** menu, change the reset credential flow from **Reset Credentials** to **Reset Credentials for password-less**

8.3.4. Creating a browser login flow with step-up mechanism

This section describes how to create advanced browser login flow using the step-up mechanism. The purpose of step-up authentication is to allow access to clients or resources based on a specific authentication level of a user.

Procedure

1. Click **Authentication** in the menu.
2. Click the **Flows** tab.
3. Click **New**.
4. Enter **Browser Incl Step up Mechanism** as an alias.
5. Click **Save**.
6. Click **Add execution**.
7. Select **Cookie** from the item list.
8. Click **Save**.
9. Click **Alternative** for the **Cookie** authentication type to set its requirement to alternative.
10. Click **Add flow**.
11. Enter **Auth Flow** as an alias.

12. Click **Save**.
13. Click **Alternative** for the **Auth Flow** authentication type to set its requirement to alternative.

Now you configure the flow for the first authentication level.

1. Click **Actions** for the **Auth Flow**.
2. Click **Add flow**.
3. Enter **1st Condition Flow** as an alias.
4. Click **Save**.
5. Click **Conditional** for the **1st Condition Flow** authentication type to set its requirement to conditional.
6. Click **Actions** for the **1st Condition Flow**.
7. Click **Add execution**.
8. Select **Conditional - Level Of Authentication** from the item list.
9. Click **Save**.
10. Click **Required** for the **Conditional - Level Of Authentication** authentication type to set its requirement to required.
11. Click **Actions** for the **Conditional - Level Of Authentication**.
12. Click **Config**.
13. Enter **Level 1** as an alias.
14. Enter **1** for the Level of Authentication (LoA).
15. Set Max Age to **36000**. This value is in seconds and it is equivalent to 10 hours, which is the default **SSO Session Max** timeout set in the realm. As a result, when a user authenticates with this level, subsequent SSO logins can re-use this level and the user does not need to authenticate with this level until the end of the user session, which is 10 hours by default.
16. Click **Save**.

Configure the condition for the first authentication level

Create authenticator config

Alias ?	<input type="text" value="Level1"/>
Level of Authentication (LoA) ?	<input type="text" value="1"/>
Max Age ?	<input type="text" value="36000"/>
	<input type="button" value="Save"/> <input type="button" value="Cancel"/>

17. Click **Actions** for the **1st Condition Flow**.
18. Click **Add execution**.

19. Select **Username Password Form** from the item list.
20. Click **Save**.
21. Click **Required** for the **Username Password Form** authentication type to set its requirement to required.

Now you configure the flow for the second authentication level.

1. Click **Actions** for the **Auth Flow**.
2. Click **Add flow**.
3. Enter **2nd Condition Flow** as an alias.
4. Click **Save**.
5. Click **Conditional** for the **2nd Condition Flow** authentication type to set its requirement to conditional.
6. Click **Actions** for the **2nd Condition Flow**.
7. Click **Add execution**.
8. Select **Conditional - Level Of Authentication** from the item list.
9. Click **Save**.
10. Click **Required** for the **Conditional - Level Of Authentication** authentication type to set its requirement to required.
11. Click **Actions** for the **Conditional - Level Of Authentication**.
12. Click **Config**.
13. Enter **Level 2** as an alias.
14. Enter **2** for the Level of Authentication (LoA).
15. Set Max Age to **0**. As a result, when a user authenticates, this level is valid just for the current authentication, but not any subsequent SSO authentications. So the user will always need to authenticate again with this level when this level is requested.
16. Click **Save**

Configure the condition for the second authentication level

Create authenticator config

Alias ?	<input type="text" value="Level2"/>
Level of Authentication (LoA) ?	<input type="text" value="2"/>
Max Age ?	<input type="text" value="0"/>
	<input type="button" value="Save"/> <input type="button" value="Cancel"/>

17. Click **Actions** for the **2nd Condition Flow**.

18. Click **Add execution**.
19. Select **OTP Form** from the item list.
20. Click **Save**.
21. Click **Required** for the **OTP Form** authentication type to set its requirement to required.

Finally, change the bindings.

1. Click the **Bindings** tab.
2. Click the **Browser Flow** item list.
3. Select **Browser Incl Step up Mechanism** from the item list.
4. Click **Save**.

Browser login with step-up mechanism

Authentication

Browser Incl Step Up Mechanism		New	Copy	Delete	Edit Flow	Add execution	Add flow
Auth Type		Requirement					
Cookie		<input type="radio"/> REQUIRED	<input checked="" type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED			Actions
Auth Flow		<input type="radio"/> REQUIRED	<input checked="" type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED	<input type="radio"/> CONDITIONAL		Actions
	1st Condition Flow	<input type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED	<input checked="" type="radio"/> CONDITIONAL		Actions
	Condition - Level Of Authentication (Level 1)	<input checked="" type="radio"/> REQUIRED	<input type="radio"/> DISABLED				Actions
	Username Password Form	<input checked="" type="radio"/> REQUIRED					Actions
	2nd Condition Flow	<input type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED	<input checked="" type="radio"/> CONDITIONAL		Actions
	Condition - Level Of Authentication (Level 2)	<input checked="" type="radio"/> REQUIRED	<input type="radio"/> DISABLED				Actions
	OTP Form	<input checked="" type="radio"/> REQUIRED	<input type="radio"/> ALTERNATIVE	<input type="radio"/> DISABLED			Actions

Request a certain authentication level

To use the step-up mechanism, you specify a requested level of authentication (LoA) in your authentication request. The **claims** parameter is used for this purpose:

```
https://{DOMAIN}/auth/realms/{REALMNAME}/protocol/openid-connect/auth?client_id={CLIENT-ID}&redirect_uri={REDIRECT-URI}&scope=openid&response_type=code&response_mode=query&nonce=exg16fxdjc&claims=%7B%22id_token%22%3A%7B%22acr%22%3A%7B%22essential%22%3Atrue%2C%22values%22%3A%5B%22gold%22%5D%7D%7D%7D
```

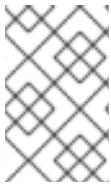
The **claims** parameter is specified in a JSON representation:

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

The Red Hat Single Sign-On javascript adapter has support for easy construct of this JSON and sending it in the login request. See [Javascript adapter documentation](#) for more details.

You can also use simpler parameter **acr_values** instead of **claims** parameter to request particular levels as non-essential. This is mentioned in the OIDC specification.

You can also configure the default level for the particular client, which is used when the parameter **acr_values** or the parameter **claims** with the **acr** claim is not present. For further details, see [Client ACR configuration](#)).



NOTE

To request the **acr_values** as text (such as **gold**) instead of a numeric value, you configure the mapping between the ACR and the LoA. It is possible to configure it at the realm level (recommended) or at the client level. For configuration see [ACR to LoA Mapping](#).

For more details see the [official OIDC specification](#).

Flow logic

The logic for the previous configured authentication flow is as follows:

If a client request a high authentication level, meaning Level of Authentication 2 (LoA 2), a user has to perform full 2-factor authentication: Username/Password + OTP. However, if a user already has a session in Keycloak, that was logged in with username and password (LoA 1), the user is only asked for the second authentication factor (OTP).

The option **Max Age** in the condition determines how long (how much seconds) the subsequent authentication level is valid. This settings helps to decide whether the user will be asked to present the authentication factor again during a subsequent authentication. If the particular level X is requested by the **claims** or **acr_values** parameter and user already authenticated with level X, but it is expired (for example max age is configured to 300 and user authenticated before 310 seconds) then the user will be asked to re-authenticate again with the particular level. However if the level is not yet expired, the user will be automatically considered as authenticated with that level.

Using **Max Age** with the value 0 means, that particular level is valid just for this single authentication. Hence every re-authentication requesting that level will need to authenticate again with that level. This is useful for operations that require higher security in the application (e.g. send payment) and always require authentication with the specific level.



WARNING

Note that parameters such as **claims** or **acr_values** might be changed by the user in the URL when the login request is sent from the client to the Red Hat Single Sign-On via the user's browser. This situation can be mitigated if client uses PAR (Pushed authorization request), a request object, or other mechanisms that prevents the user from rewrite the parameters in the URL. Hence after the authentication, clients are encouraged to check the ID Token to doublecheck that **acr** in the token corresponds to the expected level.

If no explicit level is requested by parameters, the Red Hat Single Sign-On will require the authentication with the first LoA condition found in the authentication flow, such as the Username/Password in the preceding example. When a user was already authenticated with that level

and that level expired, the user is not required to re-authenticate, but **acr** in the token will have the value 0. This result is considered as authentication based solely on **long-lived browser cookie** as mentioned in the section 2 of OIDC Core 1.0 specification.



NOTE

A conflict situation may arise when an admin specifies several flows, sets different LoA levels to each, and assigns the flows to different clients. However, the rule is always the same: if a user has a certain level, it needs only have that level to connect to a client. It's up to the admin to make sure that the LoA is coherent.

Example scenario

1. Max Age is configured as 300 seconds for level 1 condition.
2. Login request is sent without requesting any acr. Level 1 will be used and the user needs to authenticate with username and password. The token will have **acr=1**.
3. Another login request is sent after 100 seconds. The user is automatically authenticated due to the SSO and the token will return **acr=1**.
4. Another login request is sent after another 201 seconds (301 seconds since authentication in point 2). The user is automatically authenticated due to the SSO, but the token will return **acr=0** due the level 1 is considered expired.
5. Another login request is sent, but now it will explicitly request ACR of level 1 in the **claims** parameter. User will be asked to re-authenticate with username/password and then **acr=1** will be returned in the token.

ACR claim in the token

ACR claim is added to the token by the **acr loa level** protocol mapper defined in the **acr** client scope. This client scope is the realm default client scope and hence will be added to all newly created clients in the realm.

In case you do not want **acr** claim inside tokens or you need some custom logic for adding it, you can remove the client scope from your client.

Note when the login request initiates a request with the **claims** parameter requesting **acr** as **essential** claim, then Red Hat Single Sign-On will always return one of the specified levels. If it is not able to return one of the specified levels (For example if the requested level is unknown or bigger than configured conditions in the authentication flow), then Red Hat Single Sign-On will throw an error.

8.3.5. Configuring user session limits

Limits on the number of session that a user can have can be configured. Sessions can be limited per realm or per client.

To add session limits to a flow, perform the following steps.

1. Click **Add execution** for the flow.
2. Select **User Session Count Limiter** from the item list.
3. Click **Save**.

4. Click **Required** for the **User Session Count Limiter** authentication type to set its requirement to required.
5. Click **Actions** for the **User Session Count Limiter**.
6. Click **Config**.
7. Enter an alias for this config.
8. Enter the required maximum number of sessions a user can have in this realm. If a value of 0 is used, this check is disabled.
9. Enter the required maximum number of sessions a user can have for the client. If a value of 0 is used, this check is disabled.
10. Select the behavior that is required when the user tries to create a session after the limit is reached. Available behaviors are:

Deny new session - when a new session is requested and the session limit is reached, no new sessions can be created.

Terminate oldest session - when a new session is requested and the session limit has been reached, the oldest session will be removed and the new session created.

11. Optionally, add a custom error message to be displayed when the limit is reached.

Note that the user session limits should be added to your bound **Browser Flow**, **Direct Grant Flow**, **Reset Credentials** and also to any **Post Login Flow** on any configured Identity Providers. Currently, the administrator is responsible for maintaining consistency between the different configurations.

Note also that the user session limit feature is not available for CIBA.

8.4. KERBEROS

Red Hat Single Sign-On supports login with a Kerberos ticket through the Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO) protocol. SPNEGO authenticates transparently through the web browser after the user authenticates the session. For non-web cases, or when a ticket is not available during login, Red Hat Single Sign-On supports login with Kerberos username and password.

A typical use case for web authentication is the following:

1. The user logs into the desktop.
2. The user accesses a web application secured by Red Hat Single Sign-On using a browser.
3. The application redirects to Red Hat Single Sign-On login.
4. Red Hat Single Sign-On renders the HTML login screen with status 401 and HTTP header **WWW-Authenticate: Negotiate**
5. If the browser has a Kerberos ticket from desktop login, the browser transfers the desktop sign-on information to Red Hat Single Sign-On in header **Authorization: Negotiate 'spnego-token'**. Otherwise, it displays the standard login screen, and the user enters the login credentials.
6. Red Hat Single Sign-On validates the token from the browser and authenticates the user.

7. If using LDAPFederationProvider with Kerberos authentication support, Red Hat Single Sign-On provisions user data from LDAP. If using KerberosFederationProvider, Red Hat Single Sign-On lets the user update the profile and pre-fill login data.
8. Red Hat Single Sign-On returns to the application. Red Hat Single Sign-On and the application communicate through OpenID Connect or SAML messages. Red Hat Single Sign-On acts as a broker to Kerberos/SPNEGO login. Therefore Red Hat Single Sign-On authenticating through Kerberos is hidden from the application.

Perform the following steps to set up Kerberos authentication:

1. The setup and configuration of the Kerberos server (KDC).
2. The setup and configuration of the Red Hat Single Sign-On server.
3. The setup and configuration of the client machines.

8.4.1. Setup of Kerberos server

The steps to set up a Kerberos server depends on the operating system (OS) and the Kerberos vendor. Consult Windows Active Directory, MIT Kerberos, and your OS documentation for instructions on setting up and configuring a Kerberos server.

During setup, perform these steps:

1. Add some user principals to your Kerberos database. You can also integrate your Kerberos with LDAP, so user accounts provision from the LDAP server.
2. Add service principal for "HTTP" service. For example, if the Red Hat Single Sign-On server runs on **www.mydomain.org**, add the service principal **HTTP/www.mydomain.org@<kerberos realm>**.
On MIT Kerberos, you run a "kadmin" session. On a machine with MIT Kerberos, you can use the command:

```
sudo kadmin.local
```

Then, add HTTP principal and export its key to a keytab file with commands such as:

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

Ensure the keytab file **/tmp/http.keytab** is accessible on the host where Red Hat Single Sign-On is running.

8.4.2. Setup and configuration of Red Hat Single Sign-On server

Install a Kerberos client on your machine.

Procedure

1. Install a Kerberos client. If your machine runs Fedora, Ubuntu, or RHEL, install the [freeipa-client](#) package, containing a Kerberos client and other utilities.
2. Configure the Kerberos client (on Linux, the configuration settings are in the [/etc/krb5.conf](#) file).

Add your Kerberos realm to the configuration and configure the HTTP domains your server runs on.

For example, for the MYDOMAIN.ORG realm, you can configure the **domain_realm** section like this:

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

- Export the keytab file with the HTTP principal and ensure the file is accessible to the process running the Red Hat Single Sign-On server. For production, ensure that the file is readable by this process only.

For the MIT Kerberos example above, we exported keytab to the **/tmp/http.keytab** file. If your *Key Distribution Centre (KDC)* and Red Hat Single Sign-On run on the same host, the file is already available.

8.4.2.1. Enabling SPNEGO processing

By default, Red Hat Single Sign-On disables SPNEGO protocol support. To enable it, go to the [browser flow](#) and enable **Kerberos**.

Browser flow

Auth Type	Requirement
Cookie	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED
Kerberos	<input type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input checked="" type="radio"/> DISABLED
Identity Provider Redirector	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED
Forms	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input type="radio"/> CONDITIONAL
Username Password Form	<input checked="" type="radio"/> REQUIRED
Browser - Conditional OTP	<input type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input checked="" type="radio"/> CONDITIONAL
Condition - User Configured	<input checked="" type="radio"/> REQUIRED <input type="radio"/> DISABLED
OTP Form	<input checked="" type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED

Set the **Kerberos** requirement from *disabled* to *alternative* (Kerberos is optional) or *required* (browser must have Kerberos enabled). If you have not configured the browser to work with SPNEGO or Kerberos, Red Hat Single Sign-On falls back to the regular login screen.

8.4.2.2. Configure Kerberos user storage federation providerxs

You must now use [User Storage Federation](#) to configure how Red Hat Single Sign-On interprets Kerberos tickets. Two different federation providers exist with Kerberos authentication support.

To authenticate with Kerberos backed by an LDAP server, configure the [LDAP Federation Provider](#).

Procedure

- Go to the configuration page for your LDAP provider.

Ldap kerberos integration

Connection Pooling ON

Pagination ON

Kerberos Integration

Allow Kerberos authentication OFF

Use Kerberos For Password Authentication OFF

Sync Settings

Batch Size

Periodic Full Sync OFF

Periodic Changed Users Sync OFF

Cache Settings

Cache Policy

2. Toggle **Allow Kerberos authentication** to **ON**

Allow Kerberos authentication makes Red Hat Single Sign-On use the Kerberos principal access user information so information can import into the Red Hat Single Sign-On environment.

If an LDAP server is not backing up your Kerberos solution, use the **Kerberos** User Storage Federation Provider.

Procedure

1. Click **User Federation** in the menu.
2. Select **Kerberos** from the **Add provider** select box.

Kerberos user storage provider

The screenshot shows the configuration page for adding a user storage provider. The left sidebar is a dark navigation menu with the following sections: Master (dropdown), Configure (Realm Settings, Clients, Client Scopes, Roles, Identity, Providers), User Federation (selected), Authentication, and Manage (Groups, Users, Sessions, Events, Import, Export). The main content area is titled 'User Federation > Add user storage provider'. It is divided into two sections: 'Required Settings' and 'Cache Settings'. In the 'Required Settings' section, 'Enabled' is a toggle switch set to 'ON'. 'Console Display Name' is a text input field containing 'kerberos'. 'Priority' is a text input field containing '0'. 'Kerberos Realm', 'Server Principal', and 'KeyTab' are text input fields, each with a red asterisk indicating they are required. 'Debug', 'Allow Password Authentication', and 'Update Profile First Login' are toggle switches, all set to 'OFF'. The 'Cache Settings' section has a 'Cache Policy' dropdown menu set to 'DEFAULT'. At the bottom of the form are 'Save' and 'Cancel' buttons.

The **Kerberos** provider parses the Kerberos ticket for simple principal information and imports the information into the local Red Hat Single Sign-On database. User profile information, such as first name, last name, and email, are not provisioned.

8.4.3. Setup and configuration of client machines

Client machines must have a Kerberos client and set up the **krb5.conf** as described [above](#). The client machines must also enable SPNEGO login support in their browser. See [configuring Firefox for Kerberos](#) if you are using the Firefox browser.

The **.mydomain.org** URI must be in the **network.negotiate-auth.trusted-uris** configuration option.

In Windows domains, clients do not need to adjust their configuration. Internet Explorer and Edge can already participate in SPNEGO authentication.

8.4.4. Credential delegation

Kerberos supports the credential delegation. Applications may need access to the Kerberos ticket so they can re-use it to interact with other services secured by Kerberos. Because the Red Hat Single Sign-On server processed the SPNEGO protocol, you must propagate the GSS credential to your application within the OpenID Connect token claim or a SAML assertion attribute. Red Hat Single Sign-On transmits this to your application from the Red Hat Single Sign-On server. To insert this claim into the token or assertion, each application must enable the built-in protocol mapper **gss delegation credential**. This mapper is available in the **Mappers** tab of the application's client page. See [Protocol Mappers](#) chapter for more details.

Applications must deserialize the claim it receives from Red Hat Single Sign-On before using it to make GSS calls against other services. When you deserialize the credential from the access token to the `GSSCredential` object, create the `GSSContext` with this credential passed to the **`GSSManager.createContext`** method. For example:

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

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

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

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



NOTE

Configure **forwardable** Kerberos tickets in **`krb5.conf`** file and add support for delegated credentials to your browser.



WARNING

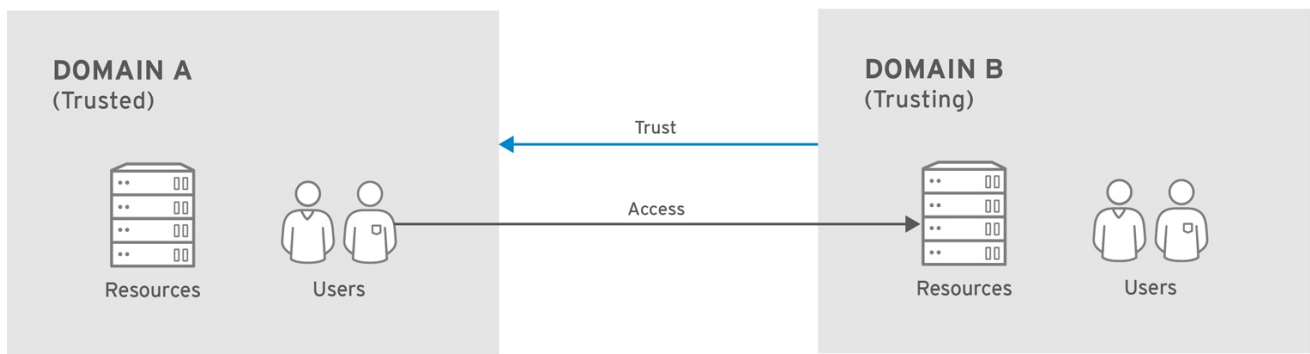
Credential delegation has security implications, so use it only if necessary and only with HTTPS. See [this article](#) for more details and an example.

8.4.5. Cross-realm trust

In the Kerberos protocol, the **realm** is a set of Kerberos principals. The definition of these principals exists in the Kerberos database, which is typically an LDAP server.

The Kerberos protocol allows cross-realm trust. For example, if 2 Kerberos realms, A and B, exist, then cross-realm trust will allow the users from realm A to access realm B's resources. Realm B trusts realm A.

Kerberos cross-realm trust



RHEL_404973_0516

The Red Hat Single Sign-On server supports cross-realm trust. To implement this, perform the following:

- Configure the Kerberos servers for the cross-realm trust. Implementing this step depends on the Kerberos server implementations. This step is necessary to add the Kerberos principal **krbtgt/B@A** to the Kerberos databases of realm A and B. This principal must have the same keys on both Kerberos realms. The principals must have the same password, key version numbers, and ciphers in both realms. Consult the Kerberos server documentation for more details.



NOTE

The cross-realm trust is unidirectional by default. You must add the principal **krbtgt/A@B** to both Kerberos databases for bidirectional trust between realm A and realm B. However, trust is transitive by default. If realm B trusts realm A and realm C trusts realm B, then realm C trusts realm A without the principal, **krbtgt/C@A**, available. Additional configuration (for example, **capaths**) may be necessary on the Kerberos client-side so clients can find the trust path. Consult the Kerberos documentation for more details.

- Configure Red Hat Single Sign-On server
 - When using an LDAP storage provider with Kerberos support, configure the server principal for realm B, as in this example: **HTTP/mydomain.com@B**. The LDAP server must find the users from realm A if users from realm A are to successfully authenticate to Red Hat Single Sign-On, because Red Hat Single Sign-On must perform the SPNEGO flow and then find the users.

For example, Kerberos principal user **john@A** must be available in the LDAP under an LDAP DN such as **uid=john,ou=People,dc=example,dc=com**. If you want users from realm A and B to authenticate, ensure that LDAP can find users from both realms A and B.

- When using a Kerberos user storage provider (typically, Kerberos without LDAP integration), configure the server principal as **HTTP/mydomain.com@B**, and users from Kerberos realms A and B must be able to authenticate.



WARNING

When using the Kerberos user storage provider, there cannot be conflicting users among Kerberos realms. If conflicting users exist, Red Hat Single Sign-On maps them to the same user.

8.4.6. Troubleshooting

If you have issues, enable additional logging to debug the problem:

- Enable **Debug** flag in the Admin Console for Kerberos or LDAP federation providers
- Enable TRACE logging for category **org.keycloak** to receive more information in server logs
- Add system properties **-Dsun.security.krb5.debug=true** and **-Dsun.security.spnego.debug=true**

8.5. X.509 CLIENT CERTIFICATE USER AUTHENTICATION

Red Hat Single Sign-On supports logging in with an X.509 client certificate if you have configured the server to use mutual SSL authentication.

A typical workflow:

- A client sends an authentication request over SSL/TLS channel.
- During the SSL/TLS handshake, the server and the client exchange their x.509/v3 certificates.
- The container (JBoss EAP) validates the certificate PKIX path and the certificate expiration date.
- The x.509 client certificate authenticator validates the client certificate by using the following methods:
 - Checks the certificate revocation status by using CRL or CRL Distribution Points.
 - Checks the Certificate revocation status by using OCSP (Online Certificate Status Protocol).
 - Validates whether the key in the certificate matches the expected key.
 - Validates whether the extended key in the certificate matches the expected extended key.
- If any of the these checks fail, the x.509 authentication fails. Otherwise, the authenticator extracts the certificate identity and maps it to an existing user.

When the certificate maps to an existing user, the behavior diverges depending on the authentication flow:

- In the Browser Flow, the server prompts users to confirm their identity or sign in with a username and password.

- In the Direct Grant Flow, the server signs in the user.



IMPORTANT

Note that it is the responsibility of the web container to validate certificate PKIX path. X.509 authenticator on the Red Hat Single Sign-On side provides just the additional support for check the certificate expiration, certificate revocation status and key usage. If you are using Red Hat Single Sign-On deployed behind reverse proxy, make sure that your reverse proxy is configured to validate PKIX path. If you do not use reverse proxy and users directly access the JBoss EAP, you should be fine as JBoss EAP makes sure that PKIX path is validated as long as it is configured as described below.

8.5.1. Features

Supported Certificate Identity Sources:

- Match SubjectDN by using regular expressions
- X500 Subject's email attribute
- X500 Subject's email from Subject Alternative Name Extension (RFC822Name General Name)
- X500 Subject's other name from Subject Alternative Name Extension. This other name is the User Principal Name (UPN), typically.
- X500 Subject's Common Name attribute
- Match IssuerDN by using regular expressions
- Certificate Serial Number
- Certificate Serial Number and IssuerDN
- SHA-256 Certificate thumbprint
- Full certificate in PEM format

8.5.1.1. Regular expressions

Red Hat Single Sign-On extracts the certificate identity from Subject DN or Issuer DN by using a regular expression as a filter. For example, this regular expression matches the email attribute:

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

The regular expression filtering applies if the **Identity Source** is set to either **Match SubjectDN using regular expression** or **Match IssuerDN using regular expression**.

8.5.1.1.1. Mapping certificate identity to an existing user

The certificate identity mapping can map the extracted user identity to an existing user's username, email, or a custom attribute whose value matches the certificate identity. For example, setting **Identity source** to *Subject's email* or **User mapping method** to *Username or email* makes the X.509 client certificate authenticator use the email attribute in the certificate's Subject DN as the search criteria when searching for an existing user by username or by email.



IMPORTANT

- If you disable **Login with email** at realm settings, the same rules apply to certificate authentication. Users are unable to log in by using the email attribute.
- Using **Certificate Serial Number and IssuerDN** as an identity source requires two custom attributes for the serial number and the IssuerDN.
- **SHA-256 Certificate thumbprint** is the lowercase hexadecimal representation of SHA-256 certificate thumbprint.
- Using **Full certificate in PEM format** as an identity source is limited to the custom attributes mapped to external federation sources, such as LDAP. Red Hat Single Sign-On cannot store certificates in its database due to length limitations, so in the case of LDAP, you must enable **Always Read Value From LDAP**.

8.5.1.1.2. Extended certificate validation

- Revocation status checking using CRL.
- Revocation status checking using CRL/Distribution Point.
- Revocation status checking using OCSP/Responder URI.
- Certificate KeyUsage validation.
- Certificate ExtendedKeyUsage validation.

8.5.2. Enable X.509 client certificate user authentication

The following sections describe how to configure JBoss EAP/Undertow and the Red Hat Single Sign-On Server to enable X.509 client certificate authentication.

8.5.2.1. Enable mutual SSL in JBoss EAP

See [Enable SSL](#) for the instructions to enable SSL in JBoss EAP.

- Open `RHSSO_HOME/standalone/configuration/standalone.xml` and add a new realm:

```
<security-realms>
  <security-realm name="ssl-realm">
    <server-identities>
      <ssl>
        <keystore path="servercert.jks"
          relative-to="jboss.server.config.dir"
          keystore-password="servercert password"/>
      </ssl>
    </server-identities>
    <authentication>
      <truststore path="truststore.jks"
        relative-to="jboss.server.config.dir"
        keystore-password="truststore password"/>
    </authentication>
  </security-realm>
</security-realms>
```

ssl/keystore

The **ssl** element contains the **keystore** element that contains the details to load the server public key pair from a JKS keystore.

ssl/keystore/path

The path to the JKS keystore.

ssl/keystore/relative-to

The path that the keystore path is relative to.

ssl/keystore/keystore-password

The password to open the keystore.

ssl/keystore/alias (optional)

The alias of the entry in the keystore. Set if the keystore contains multiple entries.

ssl/keystore/key-password (optional)

The private key password, if different from the keystore password.

authentication/truststore

Defines how to load a trust store to verify the certificate presented by the remote side of the inbound/outgoing connection. Typically, the truststore contains a collection of trusted CA certificates.

authentication/truststore/path

The path to the JKS keystore containing the certificates of the trusted certificate authorities.

authentication/truststore/relative-to

The path that the truststore path is relative to.

authentication/truststore/keystore-password

The password to open the truststore.

8.5.2.2. Enable HTTPS listener

See [HTTPS Listener](#) for the instructions to enable HTTPS in WildFly.

- Add the `<https-listener>` element.

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
....
  <server name="default-server">
    <https-listener name="default"
      socket-binding="https"
      security-realm="ssl-realm"
      verify-client="REQUESTED"/>
  </server>
</subsystem>
```

https-listener/security-realm

This value must match the name of the realm from the previous section.

https-listener/verify-client

If set to **REQUESTED**, the server optionally asks for a client certificate. If set to **REQUIRED**, the server refuses inbound connections if no client certificate has been provided.

8.5.3. Adding X.509 client certificate authentication to browser flows

1. Click **Authentication** in the menu.
2. Click the "Browser" flow.
3. Click **Copy** to make a copy of the built-in "Browser" flow.
4. Enter a name for the copy.
5. Click **Ok**.
6. Click the copy in the **Add policy** drop-down box.
7. Click **Add execution**.
8. Click "X509/Validate Username Form".
9. Click **Save**.

X509 execution

The screenshot shows the 'Create Authenticator Execution' dialog. On the left is a sidebar menu with 'Master' at the top and 'Configure' below it, containing 'Realm Settings', 'Clients', and 'Client Scopes'. The main area has tabs for 'Flows', 'Bindings', 'Required Actions', 'Password Policy', 'OTP Policy', and 'WebAuthn Policy'. The 'Provider' dropdown is set to 'X509/Validate Username Form'. Below the dropdown are 'Save' and 'Cancel' buttons.

10. Click the up/down arrow buttons to move the "X509/Validate Username Form" over the "Browser Forms" execution.
11. Set the requirement to "ALTERNATIVE".

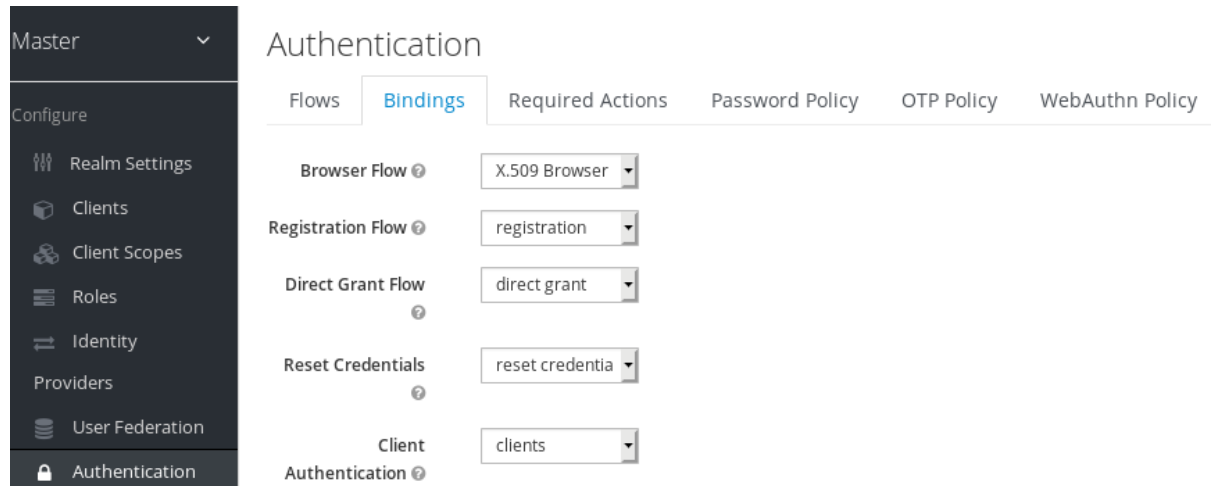
X509 browser flow

The screenshot shows the 'Authentication' configuration page. The sidebar menu is expanded to 'Authentication'. The main area has tabs for 'Flows', 'Bindings', 'Required Actions', 'Password Policy', 'OTP Policy', 'WebAuthn Policy', and 'WebAuthn Passwordless Policy'. A table lists execution requirements for the 'X509 Browser' flow. The 'X509/Validate Username Form' requirement is selected with the 'ALTERNATIVE' radio button.

Auth Type	Requirement
Cookie	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED
Kerberos	<input type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input checked="" type="radio"/> DISABLED
Identity Provider Redirector	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED
X509/Validate Username Form	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED
X.509 Browser Forms	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input type="radio"/> CONDITIONAL

12. Click the **Bindings** tab.
13. Click the **Browser Flow** drop-down list.
14. Click the copy of the browser flow from the drop-down list.
15. Click **Save**.


















X509 browser flow bindings



8.5.4. Configuring X.509 client certificate authentication

X509 configuration

X509-form-config

ID	<input type="text" value="915754bc-6012-4969-8e7c-7831da072ca6"/>
Alias 	<input type="text" value="x509-form-config"/>
User Identity Source 	<input type="text" value="Subject's e-mail"/> 
Canonical DN representation enabled 	<input type="checkbox"/> OFF
Enable Serial Number hexadecimal representation 	<input type="checkbox"/> OFF
A regular expression to extract user identity 	<input type="text" value="(.*)\{?:\$"/>
User mapping method 	<input type="text" value="Username or Email"/> 
A name of user attribute 	<input type="text" value="usercertificate"/>
CRL Checking Enabled 	<input type="checkbox"/> OFF
Enable CRL Distribution Point to check certificate revocation status 	<input type="checkbox"/> OFF
CRL Path 	<input type="text" value="crl.pem"/>
OCSP Checking Enabled 	<input type="checkbox"/> OFF
OCSP Responder Uri 	<input type="text"/>
OCSP Responder Certificate 	<input type="text"/>
Validate Key Usage 	<input type="text"/>
Validate Extended Key Usage 	<input type="text"/>
Bypass identity confirmation 	<input type="checkbox"/> OFF

User Identity Source

Defines the method for extracting the user identity from a client certificate.

Canonical DN representation enabled

Defines whether to use canonical format to determine a distinguished name. The official [Java API documentation](#) describes the format. This option affects the two User Identity Sources *Match SubjectDN using regular expression* and *Match IssuerDN using regular expression* only. Enable this option when you set up a new Red Hat Single Sign-On instance. Disable this option to retain backward compatibility with existing Red Hat Single Sign-On instances.

Enable Serial Number hexadecimal representation

Represent the [serial number](#) as hexadecimal. The serial number with the sign bit set to 1 must be left padded with 00 octet. For example, a serial number with decimal value *161*, or *a1* in hexadecimal representation is encoded as *00a1*, according to RFC5280. See [RFC5280, appendix-B](#) for more details.

A regular expression

A regular expression to use as a filter for extracting the certificate identity. The expression must contain a single group.

User Mapping Method

Defines the method to match the certificate identity with an existing user. *Username or email* searches for existing users by username or email. *Custom Attribute Mapper* searches for existing users with a custom attribute that matches the certificate identity. The name of the custom attribute is configurable.

A name of user attribute

A custom attribute whose value matches against the certificate identity. Use multiple custom attributes when attribute mapping is related to multiple values, For example, 'Certificate Serial Number and IssuerDN'.

CRL Checking Enabled

Check the revocation status of the certificate by using the Certificate Revocation List. The location of the list is defined in the **CRL file path** attribute.

Enable CRL Distribution Point to check certificate revocation status

Use CDP to check the certificate revocation status. Most PKI authorities include CDP in their certificates.

CRL file path

The path to a file containing a CRL list. The value must be a path to a valid file if the **CRL Checking Enabled** option is enabled.

OCSP Checking Enabled

Checks the certificate revocation status by using Online Certificate Status Protocol.

OCSP Fail-Open Behavior

By default the OCSP check must return a positive response in order to continue with a successful authentication. Sometimes however this check can be inconclusive: for example, the OCSP server could be unreachable, overloaded, or the client certificate may not contain an OCSP responder URI. When this setting is turned ON, authentication will be denied only if an explicit negative response is received by the OCSP responder and the certificate is definitely revoked. If a valid OCSP response is not available the authentication attempt will be accepted.

OCSP Responder URI

Override the value of the OCSP responder URI in the certificate.

Validate Key Usage

Verifies the certificate's KeyUsage extension bits are set. For example,

"digitalSignature,KeyEncipherment" verifies if bits 0 and 2 in the KeyUsage extension are set. Leave this parameter empty to disable the Key Usage validation. See [RFC5280, Section-4.2.1.3](#) for more information. Red Hat Single Sign-On raises an error when a key usage mismatch occurs.

Validate Extended Key Usage

Verifies one or more purposes defined in the Extended Key Usage extension. See [RFC5280, Section-4.2.1.12](#) for more information. Leave this parameter empty to disable the Extended Key Usage validation. Red Hat Single Sign-On raises an error when flagged as critical by the issuing CA and a key usage extension mismatch occurs.

Validate Certificate Policy

Verifies one or more policy OIDs as defined in the Certificate Policy extension. See [RFC5280, Section-4.2.1.4](#). Leave the parameter empty to disable the Certificate Policy validation. Multiple policies should be separated using a comma.

Certificate Policy Validation Mode

When more than one policy is specified in the **Validate Certificate Policy** setting, it decides whether the matching should check for all requested policies to be present, or one match is enough for a successful authentication. Default value is **All**, meaning that all requested policies should be present in the client certificate.

Bypass identity confirmation

If enabled, X.509 client certificate authentication does not prompt the user to confirm the certificate identity. Red Hat Single Sign-On signs in the user upon successful authentication.

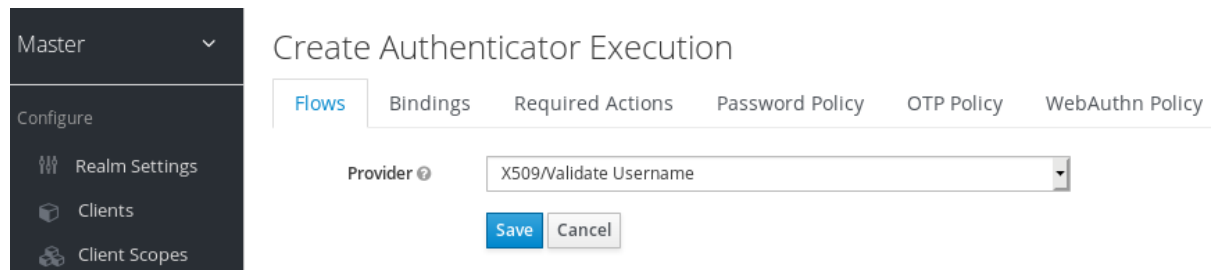
Revalidate client certificate

If set, the client certificate trust chain will be always verified at the application level using the certificates present in the configured trust store. This can be useful if the underlying web server does not enforce client certificate chain validation, for example because it is behind a non-validating load balancer or reverse proxy, or when the number of allowed CAs is too large for the mutual SSL negotiation (most browsers cap the maximum SSL negotiation packet size at 32767 bytes, which corresponds to about 200 advertised CAs). By default this option is off.

8.5.5. Adding X.509 Client Certificate Authentication to a Direct Grant Flow

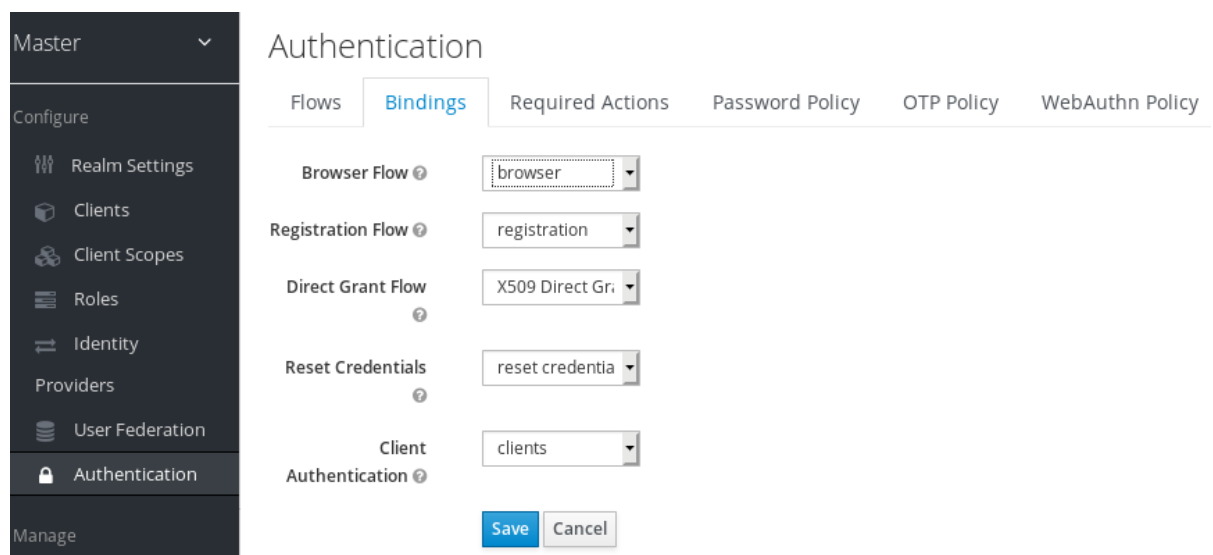
1. Click **Authentication** in the menu.
2. Click the "Direct Grant" flow.
3. Click **Copy** to make a copy of the "Direct Grant" flow.
4. Enter a name for the copy.
5. Click **Ok**.
6. Click on the **Actions** link for "Username Validation" and click **Delete**.
7. Click **Delete**.
8. Click on the **Actions** link for "Password" and click **Delete**.
9. Click **Delete**.
10. Click **Add execution**.
11. Click "X509/Validate Username".
12. Click **Save**.

X509 direct grant execution



13. Set up the x509 authentication configuration by following the steps described in the [x509 Browser Flow](#) section.
14. Click the **Bindings** tab.
15. Click the **Direct Grant Flow** drop-down list.
16. Click the newly created "x509 Direct Grant" flow.
17. Click **Save**.

X509 direct grant flow bindings



8.5.6. Client certificate lookup

When the Red Hat Single Sign-On server receives a direct HTTP request, the JBoss EAP undertow subsystem establishes an SSL handshake and extracts the client certificate. The JBoss EAP saves the client certificate to the `javax.servlet.request.X509Certificate` attribute of the HTTP request, as specified in the servlet specification. The Red Hat Single Sign-On X509 authenticator can look up the certificate from this attribute.

However, when the Red Hat Single Sign-On server listens to HTTP requests behind a load balancer or reverse proxy, the proxy server may extract the client certificate and establish a mutual SSL connection. A reverse proxy generally puts the authenticated client certificate in the HTTP header of the underlying request. The proxy forwards the request to the back end Red Hat Single Sign-On server. In this case, Red Hat Single Sign-On must look up the X.509 certificate chain from the HTTP headers rather than the attribute of the HTTP request.

If Red Hat Single Sign-On is behind a reverse proxy, you generally need to configure the alternative provider of the `x509cert-lookup` SPI in `RHSSO_HOME/standalone/configuration/standalone.xml`. With

the **default** provider looking up the HTTP header certificate, two additional built-in providers exist: **haproxy** and **apache**.

8.5.6.1. HAProxy certificate lookup provider

You use this provider when your Red Hat Single Sign-On server is behind an HAProxy reverse proxy. Use the following configuration for your server:

```
<spi name="x509cert-lookup">
  <default-provider>haproxy</default-provider>
  <provider name="haproxy" enabled="true">
    <properties>
      <property name="sslClientCert" value="SSL_CLIENT_CERT"/>
      <property name="sslCertChainPrefix" value="CERT_CHAIN"/>
      <property name="certificateChainLength" value="10"/>
    </properties>
  </provider>
</spi>
```

In this example configuration, the client certificate is looked up from the HTTP header, **SSL_CLIENT_CERT**, and the other certificates from its chain are looked up from HTTP headers such as **CERT_CHAIN_0** through **CERT_CHAIN_9**. The attribute **certificateChainLength** is the maximum length of the chain so the last attribute is **CERT_CHAIN_9**.

Consult the HAProxy documentation for the details of configuring the HTTP Headers for the client certificate and client certificate chain.

8.5.6.2. Apache certificate lookup provider

You can use this provider when your Red Hat Single Sign-On server is behind an Apache reverse proxy. Use the following configuration for your server:

```
<spi name="x509cert-lookup">
  <default-provider>apache</default-provider>
  <provider name="apache" enabled="true">
    <properties>
      <property name="sslClientCert" value="SSL_CLIENT_CERT"/>
      <property name="sslCertChainPrefix" value="CERT_CHAIN"/>
      <property name="certificateChainLength" value="10"/>
    </properties>
  </provider>
</spi>
```

This configuration is the same as the **haproxy** provider. Consult the Apache documentation on [mod_ssl](#) and [mod_headers](#) for details on how the HTTP Headers for the client certificate and client certificate chain are configured.

8.5.6.3. NGINX certificate lookup provider

You can use this provider when your Red Hat Single Sign-On server is behind an NGINX reverse proxy. Use the following configuration for your server:

```
<spi name="x509cert-lookup">
  <default-provider>nginx</default-provider>
```

```

<provider name="nginx" enabled="true">
  <properties>
    <property name="sslClientCert" value="ssl-client-cert"/>
    <property name="sslCertChainPrefix" value="USELESS"/>
    <property name="certificateChainLength" value="2"/>
  </properties>
</provider>
</spi>

```

**NOTE**

The NGINX [SSL/TLS module](#) does not expose the client certificate chain. Red Hat Single Sign-On's NGINX certificate lookup provider rebuilds it by using the [Keycloak truststore](#). Populate the Red Hat Single Sign-On truststore by using the keytool CLI with all root and intermediate CA's for rebuilding client certificate chain.

Consult the NGINX documentation for the details of configuring the HTTP Headers for the client certificate.

Example of NGINX configuration file :

```

...
server {
  ...
  ssl_client_certificate      trusted-ca-list-for-client-auth.pem;
  ssl_verify_client          optional_no_ca;
  ssl_verify_depth           2;
  ...
  location / {
    ...
    proxy_set_header ssl-client-cert    $ssl_client_escaped_cert;
    ...
  }
  ...
}

```

**NOTE**

All certificates in trusted-ca-list-for-client-auth.pem must be added to [Keycloak truststore](#).

8.5.6.4. Other reverse proxy implementations

Red Hat Single Sign-On does not have built-in support for other reverse proxy implementations. However, you can make other reverse proxies behave in a similar way to **apache** or **haproxy**. If none of these work, create your implementation of the **org.keycloak.services.x509.X509ClientCertificateLookupFactory** and **org.keycloak.services.x509.X509ClientCertificateLookup** providers. See the [Server Developer Guide](#) for details on how to add your provider.

8.5.7. Troubleshooting

Dumping HTTP headers


```
HNILCJwcmVmZXJyZWRfdXNlcm5hbWUiOiJ1c2VyMSIsImVtYWlsIjoiaXNlckByZWRoYXQuY29tIn0.CC
tltEkmlTloDpqU5alq4U1JopqEJVeogIT-
wA43edQ_DfeWSgefL0BlrPlt1SKhFMOVitywq_9XZvfiS5ZiObE33cDmhr6eohbUtDPibU2GuEIYP9WjIV
pZDMaSKQVU5SwM91m6yei22PtH-
ApPOBeG4Ru0xZtNXjwGQpuIJEi_H1rZdPY3I4U2IPuQo4Uono5gnF7re_nUvf90FJi0uaOOrsvUhUkj1x
EwQ0Diy1olymcbrDL0Ek7B30StBcjm-fe3-
0GpLttLQju0OGTkwd7Eb0UWTKoWAwspMlgpf9NalGj8rmBsz6eBIGIGWBN2Qg6v3PzbJ2NXKvq435f
9Zg",
"expires_in": 300,
"refresh_expires_in": 1800,
"refresh_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3N1bWU6IjE6ImVtYWlsIjoiaXNlckByZWRoYXQuY29tIn0.CC
WZhNGZhODVhMTEifQ.eyJleHAiOiJlcnR5cCI6IkpzZW50L3N1bWU6IjE6ImVtYWlsIjoiaXNlckByZWRoYXQuY29tIn0.CC
hMzYtMWEzMS00ZGEzLWlxMGEtNmY1ODkxYmI0MzlhliwiaXNzIjoiaHR0cHM6Ly9sb2NhbGhvc3Q6O
DQ0My9hdXRoL3JlYWxtcy9YNTA5X2RlZW8iLCJhdWQiOiJodHRwczovL2xvY2FsaG9zdDo4NDQzL2F
1dGgvcvVhbG1zL1g1MDlfZGVtbyIsInN1bWU6IjE6ImVtYWlsIjoiaXNlckByZWRoYXQuY29tIn0.CC
1dGgvcvVhbG1zL1g1MDlfZGVtbyIsInN1bWU6IjE6ImVtYWlsIjoiaXNlckByZWRoYXQuY29tIn0.CC
MGJkMCIsInR5cCI6IkpzZW50L3N1bWU6IjE6ImVtYWlsIjoiaXNlckByZWRoYXQuY29tIn0.CC
5YmEtNGQ0Ni1iNDNlTzkMTM0MGJmNTA5OCIsInN1bWU6IjE6ImVtYWlsIjoiaXNlckByZWRoYXQuY29tIn0.CC
wYjNiMTJjLTM5YmEtNGQ0Ni1iNDNlTzkMTM0MGJmNTA5OCJ9.MubgR9rvyrmSOcaq5ce-
qVTPenVQye1KsEHJr7nh9-A",
"token_type": "Bearer",
"not-before-policy": 0,
"session_state": "c0b3b12c-39ba-4d46-b43e-6d1340bf5098",
"scope": "profile email"
}
```

[host][:port]

The host and the port number of the remote Red Hat Single Sign-On server.

user_cert.crt

A public key for the user.

user_cert.key

A private key for the user. This key verifies that the public key is not forged. The private key points to the same hash as the public key.

CLIENT_ID

The client id.

CLIENT_SECRET

For confidential clients, a client secret.

8.6. W3C WEB AUTHENTICATION (WEBAUTHN)

Red Hat Single Sign-On provides support for [W3C Web Authentication \(WebAuthn\)](#). Red Hat Single Sign-On works as a WebAuthn's [Relying Party \(RP\)](#).

**NOTE**

WebAuthn's operations success depends on the user's WebAuthn supporting authenticator, browser, and platform. Make sure your authenticator, browser, and platform support the WebAuthn specification.

8.6.1. Setup

The setup procedure of WebAuthn support for 2FA is the following :

8.6.1.1. Enable WebAuthn authenticator registration

1. Click **Authentication** in the menu.
2. Click the **Required Actions** tab.
3. Click **Register**.
4. Click the **Required Action** drop-down list.
5. Click **Webauthn Register**.
6. Click **Ok**.

Mark the **Default Action** checkbox if you want all new users to be required to register their WebAuthn credentials.

8.6.1.2. Adding WebAuthn authentication to a browser flow

1. Click **Authentication** in the menu.
2. Click the **Browser** flow.
3. Click **Copy** to make a copy of the built-in **Browser** flow.
4. Enter a name for the copy.
5. Click **Ok**.
6. Click the *Actions* link for **WebAuthn Browser Browser - Conditional OTP** and click *Delete*.
7. Click **Delete**.

If you require WebAuthn for all users:

1. Click on the *Actions* link for **WebAuthn Browser Forms**.
2. Click **Add execution**.
3. Click the **Provider** drop-down list.
4. Click **WebAuthn Authenticator**.
5. Click **Save**.
6. Click *REQUIRED* for **WebAuthn Authenticator**

Authentication

Flows Bindings Required Actions Password Policy OTP Policy WebAuthn Policy

WebAuthn Browser ⓘ

Auth Type	Requirement
Cookie	<input type="radio"/> REQUIRED
Kerberos	<input type="radio"/> REQUIRED
Identity Provider Redirector	<input type="radio"/> REQUIRED
WebAuthn Browser Forms	<input type="radio"/> REQUIRED
Username Password Form	<input checked="" type="radio"/> REQUIRED
WebAuthn Authenticator	<input checked="" type="radio"/> REQUIRED

7. Click the **Bindings** tab.
8. Click the **Browser Flow** drop-down list.
9. Click **WebAuthn Browser**.
10. Click **Save**.

**NOTE**

If a user does not have WebAuthn credentials, the user must register WebAuthn credentials.

Users can log in with WebAuthn if they have a WebAuthn credential registered only. So instead of adding the **WebAuthn Authenticator** execution, you can:

Procedure

1. Click on the *Actions* link for **WebAuthn Browser Forms**.
2. Click **Add flow**.
3. Enter "Conditional 2FA" for the *Alias* field.
4. Click **Save**.
5. Click **CONDITIONAL** for **Conditional 2FA**
6. Click on the *Actions* link for **Conditional 2FA**.
7. Click **Add execution**.
8. Click the **Provider** drop-down list.
9. Click **Condition - User Configured**.
10. Click **Save**.
11. Click **REQUIRED** for **Conditional 2FA**

12. Click on the *Actions* link for **Conditional 2FA**.
13. Click **Add execution**.
14. Click the **Provider** drop-down list.
15. Click **WebAuthn Authenticator**.
16. Click **Save**.
17. Click *ALTERNATIVE* for **Conditional 2FA**

Authentication

Flows Bindings Required Actions Password Policy OTP Policy WebAuthn Policy

WebAuthn Browser

Auth Type	Requirement	Actions
Cookie	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	Actions
Kerberos	<input type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input checked="" type="radio"/> DISABLED	Actions
Identity Provider Redirector	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	Actions
WebAuthn Browser Forms	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input type="radio"/> CONDITIONAL	Actions
Username Password Form	<input checked="" type="radio"/> REQUIRED	Actions
Conditional 2FA	<input type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input checked="" type="radio"/> CONDITIONAL	Actions
Condition - User Configured	<input checked="" type="radio"/> REQUIRED <input type="radio"/> DISABLED	Actions
WebAuthn Authenticator	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	Actions

The user can choose between using WebAuthn and OTP for the second factor:

Procedure

1. Click the *Actions* link for **Conditional 2FA**.
2. Click **Add execution**.
3. Click the **Provider** drop-down list.
4. Click **ITP Form**.
5. Click **Save**.
6. Click *ALTERNATIVE* for **Conditional 2FA**

Authentication

Flows Bindings Required Actions Password Policy OTP Policy WebAuthn Policy

WebAuthn Browser

Auth Type	Requirement	Actions
Cookie	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	Actions
Kerberos	<input type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input checked="" type="radio"/> DISABLED	Actions
Identity Provider Redirector	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	Actions
WebAuthn Browser Forms	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input type="radio"/> CONDITIONAL	Actions
Username Password Form	<input checked="" type="radio"/> REQUIRED	Actions
Conditional 2FA	<input type="radio"/> REQUIRED <input type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED <input checked="" type="radio"/> CONDITIONAL	Actions
Condition - User Configured	<input checked="" type="radio"/> REQUIRED <input type="radio"/> DISABLED	Actions
WebAuthn Authenticator	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	Actions
OTP Form	<input type="radio"/> REQUIRED <input checked="" type="radio"/> ALTERNATIVE <input type="radio"/> DISABLED	Actions

8.6.2. Authenticate with WebAuthn authenticator

After registering a WebAuthn authenticator, the user carries out the following operations:

- Open the login form. The user must authenticate with a username and password.

- The user's browser asks the user to authenticate by using their WebAuthn authenticator.

8.6.3. Managing WebAuthn as an administrator

8.6.3.1. Managing credentials

Red Hat Single Sign-On manages WebAuthn credentials similarly to other credentials from [User credential management](#):

- Red Hat Single Sign-On assigns users a required action to create a WebAuthn credential from the **Reset Actions** list and select **Webauthn Register**.
- Administrators can delete a WebAuthn credential by clicking **Delete**.
- Administrators can view the credential's data, such as the AAGUID, by selecting **Show data...**
- Administrators can set a label for the credential by setting a value in the **User Label** field and saving the data.

8.6.3.2. Managing policy

Administrators can configure WebAuthn related operations as **WebAuthn Policy** per realm.

Procedure

1. Click **Authentication** in the menu.
2. Click the **WebAuthn Policy** tab.
3. Configure the items within the policy (see description below).
4. Click **Save**.

The configurable items and their description are as follows:

Configuration	Description
Relying Party Entity Name	The readable server name as a WebAuthn Relying Party. This item is mandatory and applies to the registration of the WebAuthn authenticator. The default setting is "keycloak". For more details, see WebAuthn Specification .
Signature Algorithms	The algorithms telling the WebAuthn authenticator which signature algorithms to use for the Public Key Credential . Red Hat Single Sign-On uses the Public Key Credential to sign and verify Authentication Assertions . If no algorithms exist, the default ES256 is adapted. ES256 is an optional configuration item applying to the registration of WebAuthn authenticators. For more details, see WebAuthn Specification .

Configuration	Description
Relying Party ID	The ID of a WebAuthn Relying Party that determines the scope of Public Key Credentials . The ID must be the origin's effective domain. This ID is an optional configuration item applied to the registration of WebAuthn authenticators. If this entry is blank, Red Hat Single Sign-On adapts the host part of Red Hat Single Sign-On's base URL. For more details, see WebAuthn Specification .
Attestation Conveyance Preference	The WebAuthn API implementation on the browser (WebAuthn Client) is the preferential method to generate Attestation statements. This preference is an optional configuration item applying to the registration of the WebAuthn authenticator. If no option exists, its behavior is the same as selecting "none". For more details, see WebAuthn Specification .
Authenticator Attachment	The acceptable attachment pattern of a WebAuthn authenticator for the WebAuthn Client. This pattern is an optional configuration item applying to the registration of the WebAuthn authenticator. For more details, see WebAuthn Specification .
Require Resident Key	The option requiring that the WebAuthn authenticator generates the Public Key Credential as Client-side-resident Public Key Credential Source . This option applies to the registration of the WebAuthn authenticator. If left blank, its behavior is the same as selecting "No". For more details, see WebAuthn Specification .
User Verification Requirement	The option requiring that the WebAuthn authenticator confirms the verification of a user. This is an optional configuration item applying to the registration of a WebAuthn authenticator and the authentication of a user by a WebAuthn authenticator. If no option exists, its behavior is the same as selecting "preferred". For more details, see WebAuthn Specification for registering a WebAuthn authenticator and WebAuthn Specification for authenticating the user by a WebAuthn authenticator .

Configuration	Description
Timeout	The timeout value, in seconds, for registering a WebAuthn authenticator and authenticating the user by using a WebAuthn authenticator. If set to zero, its behavior depends on the WebAuthn authenticator's implementation. The default value is 0. For more details, see WebAuthn Specification for registering a WebAuthn authenticator and WebAuthn Specification for authenticating the user by a WebAuthn authenticator .
Avoid Same Authenticator Registration	If enabled, Red Hat Single Sign-On cannot re-register an already registered WebAuthn authenticator.
Acceptable AAGUIDs	The white list of AAGUIDs which a WebAuthn authenticator must register against.

8.6.4. Attestation statement verification

When registering a WebAuthn authenticator, Red Hat Single Sign-On verifies the trustworthiness of the attestation statement generated by the WebAuthn authenticator. Red Hat Single Sign-On requires the trust anchor's certificates for this. Red Hat Single Sign-On uses the [Keycloak truststore](#), so you must import these certificates into it in advance.

To omit this validation, disable this truststore or set the WebAuthn policy's configuration item "Attestation Conveyance Preference" to "none".

8.6.5. Managing WebAuthn credentials as a user

8.6.5.1. Register WebAuthn authenticator

The appropriate method to register a WebAuthn authenticator depends on whether the user has already registered an account on Red Hat Single Sign-On.

8.6.5.2. New user

If the **WebAuthn Register** required action is **Default Action** in a realm, new users must set up the WebAuthn security key after their first login.

Procedure

1. Open the login form.
2. Click **Register**.
3. Fill in the items on the form.
4. Click **Register**.

After successfully registering, the browser asks the user to enter the text of their WebAuthn authenticator's label.

8.6.5.3. Existing user

If **WebAuthn Authenticator** is set up as required as shown in the first example, then when existing users try to log in, they are required to register their WebAuthn authenticator automatically:

Procedure

1. Open the login form.
2. Enter the items on the form.
3. Click **Save**.
4. Click **Login**.

After successful registration, the user's browser asks the user to enter the text of their WebAuthn authenticator's label.

8.6.6. Passwordless WebAuthn together with Two-Factor

Red Hat Single Sign-On uses WebAuthn for two-factor authentication, but you can use WebAuthn as the first-factor authentication. In this case, users with **passwordless** WebAuthn credentials can authenticate to Red Hat Single Sign-On without a password. Red Hat Single Sign-On can use WebAuthn as both the passwordless and two-factor authentication mechanism in the context of a realm and a single authentication flow.

An administrator typically requires that Security Keys registered by users for the WebAuthn passwordless authentication meet different requirements. For example, the security keys may require users to authenticate to the security key using a PIN, or the security key attests with a stronger certificate authority.

Because of this, Red Hat Single Sign-On permits administrators to configure a separate **WebAuthn Passwordless Policy**. There is a required **Webauthn Register Passwordless** action of type and separate authenticator of type **WebAuthn Passwordless Authenticator**.

8.6.6.1. Setup

Procedure

Set up WebAuthn passwordless support as follows:

1. Register a new required action for WebAuthn passwordless support. Use the steps described in [Enable WebAuthn Authenticator Registration](#). Register the **Webauthn Register Passwordless** action.
2. Configure the policy. You can use the steps and configuration options described in [Managing Policy](#). Perform the configuration in the Admin Console in the tab **WebAuthn Passwordless Policy**. Typically the requirements for the security key will be stronger than for the two-factor policy. For example, you can set the **User Verification Requirement** to **Required** when you configure the passwordless policy.
3. Configure the authentication flow. Use the **WebAuthn Browser** flow described in [Adding WebAuthn Authentication to a Browser Flow](#). Configure the flow as follows:

- The **WebAuthn Browser Forms** subflow contains **Username Form** as the first authenticator. Delete the default **Username Password Form** authenticator and add the **Username Form** authenticator. This action requires the user to provide a username as the first step.
- There will be a required subflow, which can be named **Passwordless Or Two-factor**, for example. This subflow indicates the user can authenticate with Passwordless WebAuthn credential or with Two-factor authentication.
- The flow contains **WebAuthn Passwordless Authenticator** as the first alternative.
- The second alternative will be a subflow named **Password And Two-factor Webauthn**, for example. This subflow contains a **Password Form** and a **WebAuthn Authenticator**.

The final configuration of the flow looks similar to this:

PasswordLess flow

Authentication

WebAuthn Browser				Requirement				New Copy Delete Add execution Add flow	
Auth Type				REQUIRED	ALTERNATIVE	DISABLED	CONDITIONAL	Actions	
Cookie				<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>		Actions	
Kerberos				<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>		Actions	
Identity Provider Redirector				<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>		Actions	
WebAuthn Browser Forms				<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Actions	
	Username Form			<input checked="" type="radio"/>				Actions	
	Passwordless Or Two-factor			<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Actions	
		WebAuthn Passwordless Authenticator		<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>		Actions	
		Password And Two-factor Webauthn		<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Actions	
			Password Form	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>		Actions	
			WebAuthn Authenticator	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>		Actions	

You can now add **WebAuthn Register Passwordless** as the required action to a user, already known to Red Hat Single Sign-On, to test this. During the first authentication, the user must use the password and second-factor WebAuthn credential. The user does not need to provide the password and second-factor WebAuthn credential if they use the WebAuthn Passwordless credential.

8.6.7. LoginLess WebAuthn

Red Hat Single Sign-On uses WebAuthn for two-factor authentication, but you can use WebAuthn as the first-factor authentication. In this case, users with **passwordless** WebAuthn credentials can authenticate to Red Hat Single Sign-On without submitting a login or a password. Red Hat Single Sign-On can use WebAuthn as both the loginless/passwordless and two-factor authentication mechanism in the context of a realm.

An administrator typically requires that Security Keys registered by users for the WebAuthn loginless authentication meet different requirements. Loginless authentication requires users to authenticate to the security key (for example by using a PIN code or a fingerprint) and that the cryptographic keys associated with the loginless credential are stored physically on the security key. Not all security keys meet that kind of requirements. Check with your security key vendor if your device supports 'user verification' and 'resident key'. See [Supported Security Keys](#).

Red Hat Single Sign-On permits administrators to configure the **WebAuthn Passwordless Policy** in a way that allows loginless authentication. Note that loginless authentication can only be configured with **WebAuthn Passwordless Policy** and with **WebAuthn Passwordless** credentials. WebAuthn loginless authentication and WebAuthn passwordless authentication can be configured on the same realm but will share the same policy **WebAuthn Passwordless Policy**.

8.6.7.1. Setup

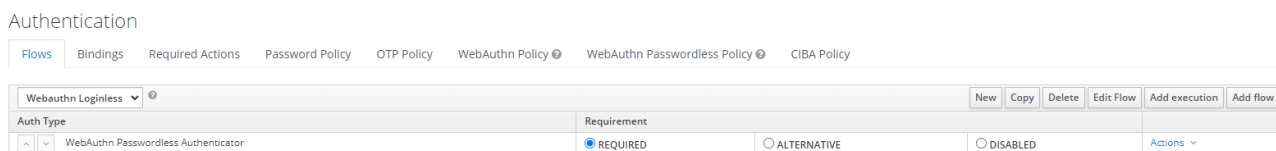
Procedure

Set up WebAuthn Loginless support as follows:

1. Register a new required action for WebAuthn passwordless support. Use the steps described in [Enable WebAuthn Authenticator Registration](#). Register the **Webauthn Register Passwordless** action.
2. Configure the **WebAuthn Passwordless Policy**. Perform the configuration in the Admin Console, **Authentication** section, in the tab **WebAuthn Passwordless Policy**. You have to set **User Verification Requirement** to **required** and **Require Resident Key** to **Yes** when you configure the policy for loginless scenario. Note that since there isn't a dedicated Loginless policy it won't be possible to mix authentication scenarios with user verification=no/resident key=no and loginless scenarios (user verification=yes/resident key=yes). Storage capacity is usually very limited on security keys meaning that you won't be able to store many resident keys on your security key.
3. Configure the authentication flow. Create a new authentication flow, add the "WebAuthn Passwordless" execution and set the Requirement setting of the execution to **Required**

The final configuration of the flow looks similar to this:

LoginLess flow



You can now add the required action **WebAuthn Register Passwordless** to a user, already known to Red Hat Single Sign-On, to test this. The user with the required action configured will have to authenticate (with a username/password for example) and will then be prompted to register a security key to be used for loginless authentication.

8.6.7.2. Vendor specific remarks

8.6.7.2.1. Compatibility check list

Loginless authentication with Red Hat Single Sign-On requires the security key to meet the following features

- FIDO2 compliance: not to be confused with FIDO/U2F
- User verification: the ability for the security key to authenticate the user (prevents someone finding your security key to be able to authenticate loginless and passwordless)

- Resident key: the ability for the security key to store the login and the cryptographic keys associated with the client application

8.6.7.2.2. Windows Hello

To use Windows Hello based credentials to authenticate against Red Hat Single Sign-On, configure the **Signature Algorithms** setting of the **WebAuthn Passwordless Policy** to include the **RS256** value. Note that some browsers don't allow access to platform security key (like Windows Hello) inside private windows.

8.6.7.2.3. Supported security keys

The following security keys have been successfully tested for loginless authentication with Red Hat Single Sign-On:

- Windows Hello (Windows 10 21H1/21H2)
- Yubico Yubikey 5 NFC
- Feitian ePass FIDO-NFC

8.7. RECOVERY CODES (RECOVERYCODES)

You can configure Recovery codes for two-factor authentication by adding 'Recovery Authentication Code Form' as a two-factor authenticator to your authentication flow. For an example of configuring this authenticator, see [WebAuthn](#).



NOTE

RecoveryCodes is **Technology Preview** and is not fully supported. This feature is disabled by default.

To enable start the server with **-Dkeycloak.profile=preview** or **-Dkeycloak.profile.feature.recovery_codes=enabled**. For more details see [Profiles](#).

8.8. CONDITIONS IN CONDITIONAL FLOWS

As was mentioned in [Execution requirements](#), *Condition* executions can be only contained in *Conditional* subflow. If all *Condition* executions evaluate as true, then the *Conditional* sub-flow acts as *Required*. You can process the next execution in the *Conditional* sub-flow. If some executions included in the *Conditional* sub-flow evaluate as false, then the whole sub-flow is considered as *Disabled*.

8.8.1. Available conditions

Condition - User Role

This execution has the ability to determine if the user has a role defined by *User role* field. If the user has the required role, the execution is considered as true and other executions are evaluated. The administrator has to define the following fields:

Alias

Describes a name of the execution, which will be shown in the authentication flow.

User role

Role the user should have to execute this flow. To specify an application role the syntax is **appname.approle** (for example **myapp.myrole**).

Condition - User Configured

This checks if the other executions in the flow are configured for the user. The Execution requirements section includes an example of the OTP form.

Condition - User Attribute

This checks if the user has set up the required attribute. There is a possibility to negate output, which means the user should not have the attribute. The [User Attributes](#) section shows how to add a custom attribute. You can provide these fields:

Alias

Describes a name of the execution, which will be shown in the authentication flow.

Attribute name

Name of the attribute to check.

Expected attribute value

Expected value in the attribute.

Negate output

You can negate the output. In other words, the attribute should not be present.

8.8.2. Explicitly deny/allow access in conditional flows

You can allow or deny access to resources in a conditional flow. The two authenticators **Deny Access** and **Allow Access** control access to the resources by conditions.

Allow Access

Authenticator will always successfully authenticate. This authenticator is not configurable.

Deny Access

Access will always be denied. You can define an error message, which will be shown to the user. You can provide these fields:

Alias

Describes a name of the execution, which will be shown in the authentication flow.

Error message

Error message which will be shown to the user. The error message could be provided as a particular message or as a property in order to use it with localization. (i.e. "You do not have the role 'admin'", *my-property-deny* in messages properties) Leave blank for the default message defined as property **access-denied**.

Here is an example how to deny access to all users who do not have the role **role1** and show an error message defined by a property **deny-role1**. This example includes **Condition - User Role** and **Deny Access** executions.

Browser flow

Conditions Forms	Execution	REQUIRED	ALTERNATIVE	DISABLED	CONDITIONAL	Actions
Username Form		<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Actions
Access By Role		<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	Actions
Condition - User Role (Must not have role1)		<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Actions
Deny Access (role1-alias)		<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Actions
Password Form		<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Actions

Condition - user role configuration

Create authenticator config

Alias  Must not have role1

User role  role1 

Negate output  ON

Configuration of the **Deny Access** is really easy. You can specify an arbitrary Alias and required message like this:

Create authenticator config

Alias  role1-alias

Error message  deny-role1 

The last thing is defining the property with an error message in the login theme **messages_en.properties** (for English):

deny-role1 = You do not have required role!

CHAPTER 9. INTEGRATING IDENTITY PROVIDERS

An Identity Broker is an intermediary service connecting service providers with identity providers. The identity broker creates a relationship with an external identity provider to use the provider's identities to access the internal services the service provider exposes.

From a user perspective, identity brokers provide a user-centric, centralized way to manage identities for security domains and realms. You can link an account with one or more identities from identity providers or create an account based on the identity information from them.

An identity provider derives from a specific protocol used to authenticate and send authentication and authorization information to users. It can be:

- A social provider such as Facebook, Google, or Twitter.
- A business partner whose users need to access your services.
- A cloud-based identity service you want to integrate.

Typically, Red Hat Single Sign-On bases identity providers on the following protocols:

- **SAML v2.0**
- **OpenID Connect v1.0**
- **OAuth v2.0**

9.1. BROKERING OVERVIEW

When using Red Hat Single Sign-On as an identity broker, Red Hat Single Sign-On does not force users to provide their credentials to authenticate in a specific realm. Red Hat Single Sign-On displays a list of identity providers from which they can authenticate.

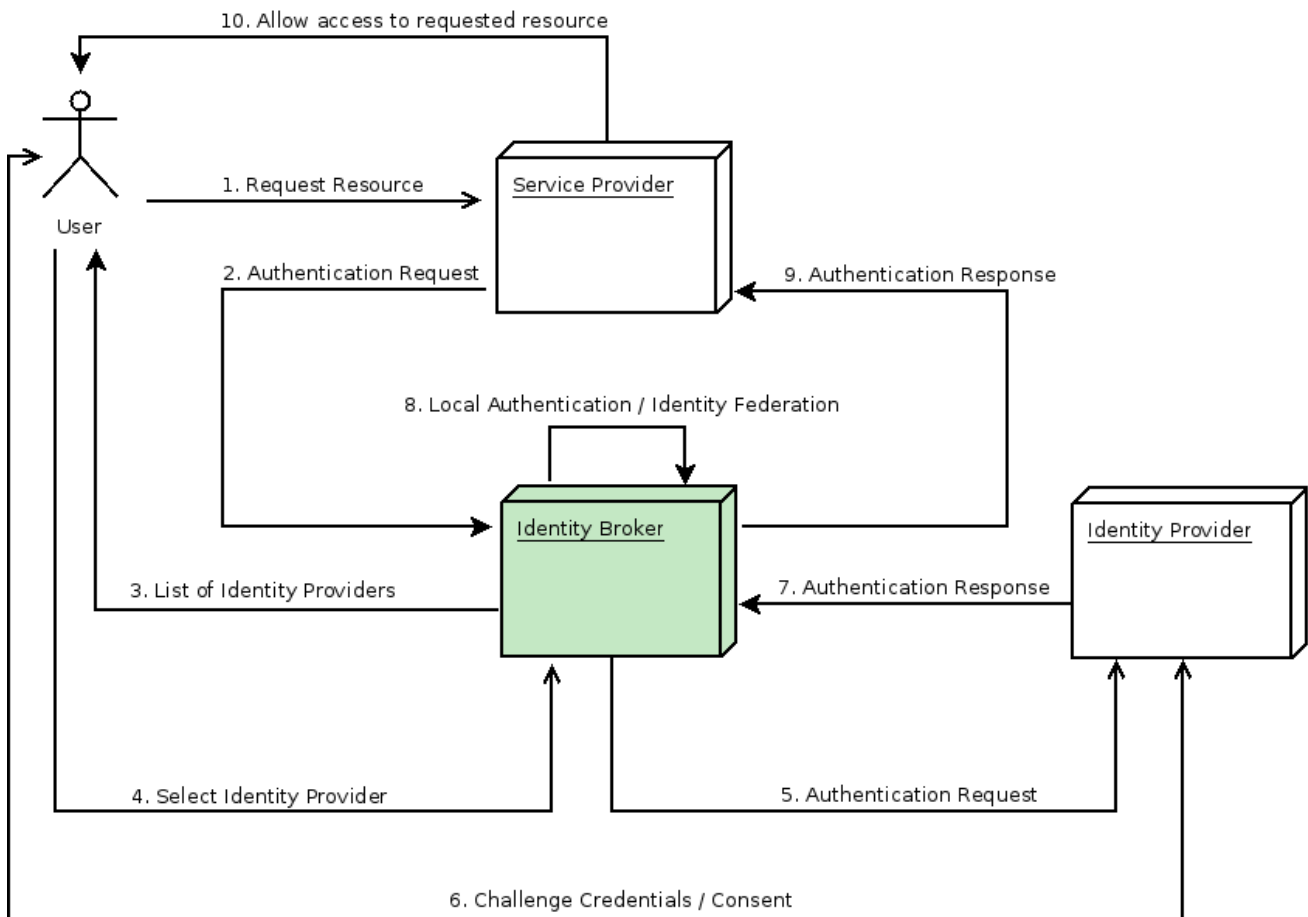
If you configure a default identity provider, Red Hat Single Sign-On redirects users to the default provider.



NOTE

Different protocols may require different authentication flows. All the identity providers supported by Red Hat Single Sign-On use the following flow.

Identity broker flow



1. The unauthenticated user requests a protected resource in a client application.
2. The client application redirects the user to Red Hat Single Sign-On to authenticate.
3. Red Hat Single Sign-On displays the login page with a list of identity providers configured in a realm.
4. The user selects one of the identity providers by clicking its button or link.
5. Red Hat Single Sign-On issues an authentication request to the target identity provider requesting authentication and redirects the user to the identity provider's login page. The administrator has already set the connection properties and other configuration options for the Admin Console's identity provider.
6. The user provides credentials or consents to authenticate with the identity provider.
7. Upon successful authentication by the identity provider, the user redirects back to Red Hat Single Sign-On with an authentication response. Usually, the response contains a security token used by Red Hat Single Sign-On to trust the identity provider's authentication and retrieve user information.
8. Red Hat Single Sign-On checks if the response from the identity provider is valid. If valid, Red Hat Single Sign-On imports and creates a user if the user does not already exist. Red Hat Single Sign-On may ask the identity provider for further user information if the token does not contain that information. This behavior is *identity federation*. If the user already exists, Red Hat Single Sign-On may ask the user to link the identity returned from the identity provider with the existing account. This behavior is *account linking*. With Red Hat Single Sign-On, you can configure *Account linking* and specify it in the [First Login Flow](#). At this step, Red Hat Single Sign-On authenticates the user and issues its token to access the requested resource in the service provider.

9. When the user authenticates, Red Hat Single Sign-On redirects the user to the service provider by sending the token previously issued during the local authentication.
10. The service provider receives the token from Red Hat Single Sign-On and permits access to the protected resource.

Variations of this flow are possible. For example, the client application can request a specific identity provider rather than displaying a list of them, or you can set Red Hat Single Sign-On to force users to provide additional information before federating their identity.

At the end of the authentication process, Red Hat Single Sign-On issues its token to client applications. Client applications are separate from the external identity providers, so they cannot see the client application's protocol or how they validate the user's identity. The provider only needs to know about Red Hat Single Sign-On.

9.2. DEFAULT IDENTITY PROVIDER

Red Hat Single Sign-On can redirect to an identity provider rather than displaying the login form. To enable this redirection:

Procedure

1. Click **Authentication** in the menu.
2. Click the **Browser** flow.
3. Select **Identity Provider Redirector** from the drop-down list.
4. Set **Default Identity Provider** to the identity provider you want to redirect users to.

If Red Hat Single Sign-On does not find the configured default identity provider, the login form is displayed.

This authenticator is responsible for processing the **kc_idp_hint** query parameter. See the [client suggested identity provider](#) section for more information.

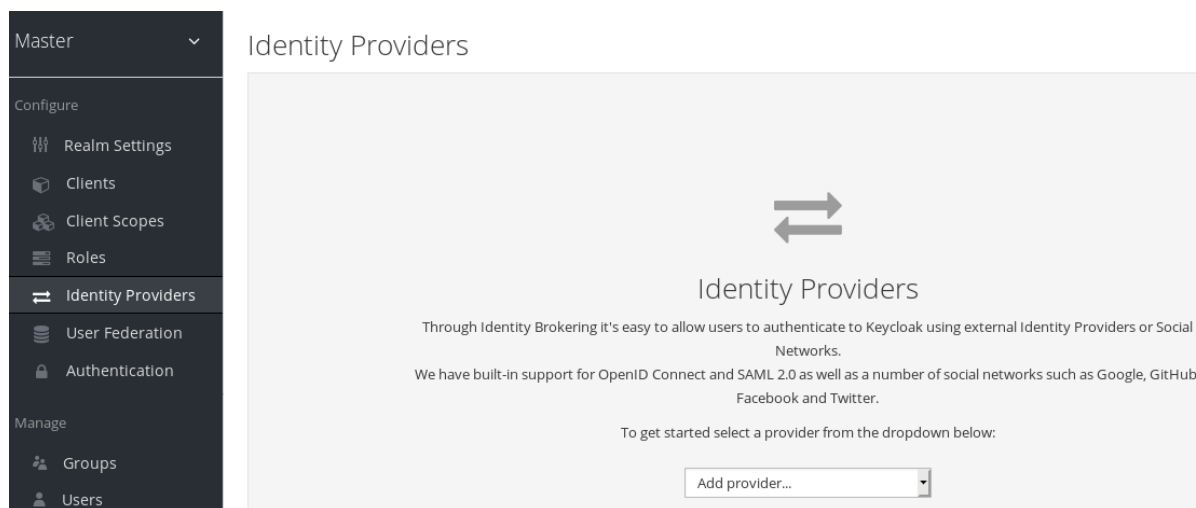
9.3. GENERAL CONFIGURATION

The foundations of the identity broker configuration are identity providers (IDPs). Red Hat Single Sign-On creates identity providers for each realm and enables them for every application by default. Users from a realm can use any of the registered identity providers when signing in to an application.

Procedure

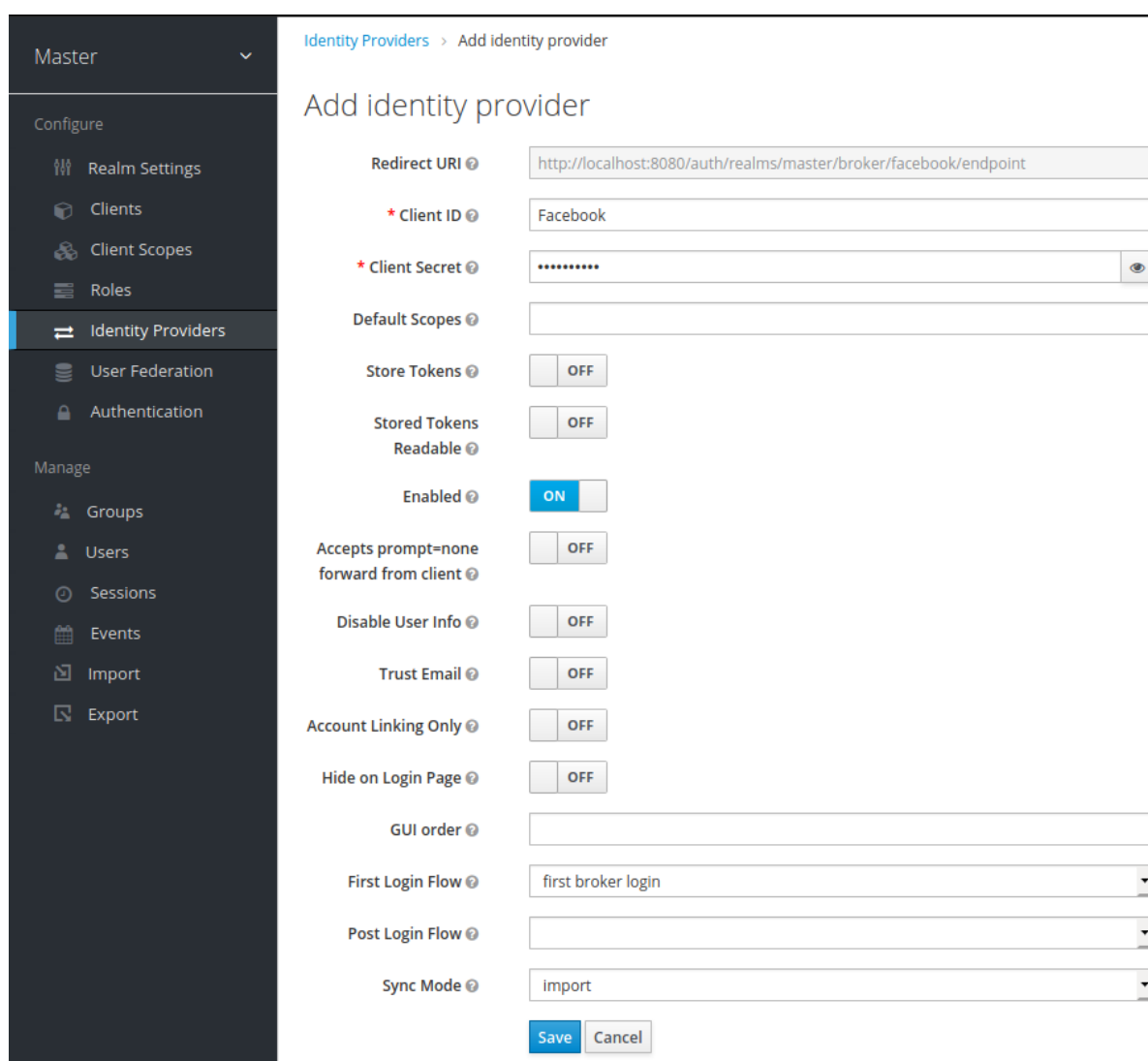
1. Click **Identity Providers** in the menu.

Identity Providers



- From the **Add provider** list, select the identity provider you want to add. Red Hat Single Sign-On displays the configuration page for the identity provider you selected.

Add facebook identity Provider



When you configure an identity provider, the identity provider appears on the Red Hat Single Sign-On login page as an option. You can place custom icons on the login screen for each identity provider. See [custom icons](#) for more information.

IDP login page

Sign in to your account

Username or email

Password

Remember me

Sign In

Or sign in with



Facebook

New user? [Register](#)

Social

Social providers enable social authentication in your realm. With Red Hat Single Sign-On, users can log in to your application using a social network account. Supported providers include Twitter, Facebook, Google, LinkedIn, Instagram, Microsoft, PayPal, Openshift v3, GitHub, GitLab, Bitbucket, and Stack Overflow.

Protocol-based

Protocol-based providers rely on specific protocols to authenticate and authorize users. Using these providers, you can connect to any identity provider compliant with a specific protocol. Red Hat Single Sign-On provides support for SAML v2.0 and OpenID Connect v1.0 protocols. You can configure and broker any identity provider based on these open standards.

Although each type of identity provider has its configuration options, all share a common configuration. The following configuration options are available:

Table 9.1. Common Configuration

Configuration	Description
Alias	The alias is a unique identifier for an identity provider and references an internal identity provider. Red Hat Single Sign-On uses the alias to build redirect URIs for OpenID Connect protocols that require a redirect URI or callback URL to communicate with an identity provider. All identity providers must have an alias. Alias examples include facebook , google , and idp.acme.com .
Enabled	Toggles the provider ON or OFF.
Hide on Login Page	When ON , Red Hat Single Sign-On does not display this provider as a login option on the login page. Clients can request this provider by using the 'kc_idp_hint' parameter in the URL to request a login.
Account Linking Only	When ON , Red Hat Single Sign-On links existing accounts with this provider. This provider cannot log users in, and Red Hat Single Sign-On does not display this provider as an option on the login page.
Store Tokens	When ON , Red Hat Single Sign-On stores tokens from the identity provider.
Stored Tokens Readable	When ON , users can retrieve the stored identity provider token. This action also applies to the <i>broker</i> client-level role <i>read token</i> .
Trust Email	When ON , Red Hat Single Sign-On trusts email addresses from the identity provider. If the realm requires email validation, users that log in from this identity provider do not need to perform the email verification process.
GUI Order	The sort order of the available identity providers on the login page.
First Login Flow	The authentication flow Red Hat Single Sign-On triggers when users use this identity provider to log into Red Hat Single Sign-On for the first time.
Post Login Flow	The authentication flow Red Hat Single Sign-On triggers when a user finishes logging in with the external identity provider.

Configuration	Description
Sync Mode	Strategy to update user information from the identity provider through mappers. When choosing legacy , Red Hat Single Sign-On used the current behavior. Import does not update user data and force updates user data when possible. See Identity Provider Mappers for more information.

9.4. SOCIAL IDENTITY PROVIDERS

A social identity provider can delegate authentication to a trusted, respected social media account. Red Hat Single Sign-On includes support for social networks such as Google, Facebook, Twitter, GitHub, LinkedIn, Microsoft, and Stack Overflow.

9.4.1. Bitbucket

To log in with Bitbucket, perform the following procedure.

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **Bitbucket**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page in the Red Hat Single Sign-On administration console. The left sidebar is dark with a 'Master' dropdown and a menu with options like 'Configure', 'Identity Providers', 'User Federation', and 'Authentication'. The main content area is titled 'Add identity provider' and contains the following configuration fields:

- Redirect URI**:
- * Consumer Key**:
- * Consumer Secret**:
- Default Scopes**:
- Store Tokens**: OFF
- Stored Tokens Readable**: OFF
- Enabled**: ON
- Trust Email**: OFF
- Account Linking Only**: OFF
- Hide on Login Page**: OFF
- GUI order**:
- First Login Flow**:
- Post Login Flow**:

3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, perform the [OAuth on Bitbucket Cloud](#) process. When you click **Add Consumer**:
 - a. Paste the value of **Redirect URI** into the **Callback URL** field.
 - b. Ensure you select **Email** and **Read** in the **Account** section to permit your application to read email.
5. Note the **Key** and **Secret** values Bitbucket displays when you create your consumer.
6. In Red Hat Single Sign-On, paste the value of the **Key** into the **Client ID** field.
7. In Red Hat Single Sign-On, paste the value of the **Secret** into the **Client Secret** field.
8. Click **Save**.

9.4.2. Facebook

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **Facebook**. Red Hat Single Sign-On displays the configuration page for the Facebook identity provider.

Add identity provider

The screenshot displays the configuration page for a Facebook Identity Provider in the Red Hat Single Sign-On console. The left-hand navigation menu is visible, with 'Identity Providers' selected. The main content area shows the 'Settings' tab for the Facebook provider. The configuration fields are as follows:

- Redirect URI:** `http://localhost:8080/auth/realms/master/broker/facebook/endpoint`
- * Client ID:** Facebook
- * Client Secret:** [Redacted]
- Default Scopes:** [Empty field]
- Store Tokens:** OFF
- Stored Tokens Readable:** OFF
- Enabled:** ON
- Accepts prompt=none forward from client:** OFF
- Disable User Info:** OFF
- Trust Email:** OFF
- Account Linking Only:** OFF
- Hide on Login Page:** OFF
- GUI order:** [Empty field]
- First Login Flow:** first broker login
- Post Login Flow:** [Empty field]

3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, follow the [Facebook Developer Guide's](#) instructions to create a project and client in Facebook.
 - a. Ensure your app is a website-type app.
 - b. Enter the **Redirect URI's** value into the **Site URL** of the Facebook **Website** settings block.
 - c. Ensure the app is public.
5. Enter the **Client ID** and **Client Secret** values from your Facebook app into the **Client ID** and **Client Secret** fields in Red Hat Single Sign-On.
6. Enter the required scopes into the **Default Scopes** field. By default, Red Hat Single Sign-On uses the **email** scope. See [Graph API](#) for more information about Facebook scopes.

Red Hat Single Sign-On sends profile requests to **graph.facebook.com/me?fields=id,name,email,first_name,last_name** by default. The response contains the id, name, email, first_name, and last_name fields only. To fetch additional fields from the Facebook profile, add a corresponding scope and add the field name in the **Additional user's profile fields** configuration option field.

9.4.3. GitHub

To log in with Github, perform the following procedure.

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **GitHub**.

Add identity provider

The screenshot displays the 'Add identity provider' configuration page. The left sidebar is dark-themed and contains a navigation menu with the following items: Master (dropdown), Configure (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication), and Manage (Groups, Users, Sessions, Events, Import, Export). The 'Identity Providers' menu item is highlighted. The main content area is titled 'Add identity provider' and contains the following configuration fields:

- Redirect URI**:
- * Client ID**:
- * Client Secret**:
- Default Scopes**:
- Store Tokens**:
- Stored Tokens Readable**:
- Enabled**:
- Accepts prompt=none forward from client**:
- Disable User Info**:
- Trust Email**:
- Account Linking Only**:
- Hide on Login Page**:
- GUI order**:
- First Login Flow**:
- Post Login Flow**:

3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, [create an OAUTH app](#).
 - a. Enter the value of **Redirect URI** into the **Authorization callback URL** field when creating the app.
 - b. Note the Client ID and Client secret on the management page of your OAUTH app.
5. In Red Hat Single Sign-On, paste the value of the **Client ID** into the **Client ID** field.

6. In Red Hat Single Sign-On, paste the value of the **Client Secret** into the **Client Secret** field.
7. Click **Save**.

9.4.4. GitLab

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **GitLab**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page in Red Hat Single Sign-On. The left sidebar is dark grey with a white navigation menu. The main content area is white and contains the configuration form. The breadcrumb 'Identity Providers > Add identity provider' is visible at the top of the main area. The form includes the following fields and controls:

- Redirect URI**: Text input field containing 'http://localhost:8080/auth/realms/master/broker/github/endpoint'.
- * Client ID**: Empty text input field.
- * Client Secret**: Empty text input field with an eye icon for visibility toggle.
- Default Scopes**: Empty text input field.
- Store Tokens**: Toggle switch set to OFF.
- Stored Tokens Readable**: Toggle switch set to OFF.
- Enabled**: Toggle switch set to ON.
- Accepts prompt=none forward from client**: Toggle switch set to OFF.
- Disable User Info**: Toggle switch set to OFF.
- Trust Email**: Toggle switch set to OFF.
- Account Linking Only**: Toggle switch set to OFF.
- Hide on Login Page**: Toggle switch set to OFF.
- GUI order**: Empty text input field.
- First Login Flow**: Dropdown menu with 'first broker login' selected.
- Post Login Flow**: Empty dropdown menu.

3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, [add a new GitLab application](#) .
 - a. Use the **Redirect URI** in your clipboard as the **Redirect URI**.
 - b. Note the **Client ID** and **Client Secret** when you save the application.
5. In Red Hat Single Sign-On, paste the value of the **Client ID** into the **Client ID** field.

6. In Red Hat Single Sign-On, paste the value of the **Client Secret** into the **Client Secret** field.
7. Click **Save**.

9.4.5. Google

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **Google**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page in Red Hat Single Sign-On. The left sidebar is open, showing the 'Identity Providers' menu item. The main content area is titled 'Add identity provider' and contains the following configuration options:

- Redirect URI**:
- * Client ID**:
- * Client Secret**:
- Hosted Domain**:
- Use userIp Param**: OFF
- Request refresh token**: OFF
- Default Scopes**:
- Store Tokens**: OFF
- Stored Tokens Readable**: OFF
- Enabled**: ON

3. In a separate browser tab open [the Google Cloud Platform console](#).
4. In the Google dashboard for your Google app, click the **OAuth consent screen** menu. Create a consent screen, ensuring that the user type of the consent screen is external.
5. In the Google dashboard:
 - a. Click the **Credentials** menu.
 - b. Click **CREATE CREDENTIALS - OAuth Client ID**.
 - c. From the **Application type** list, select **Web application**.
 - d. Click **Create**.
 - e. Note **Your Client ID** and **Your Client Secret**.
6. In Red Hat Single Sign-On, paste the value of the **Your Client ID** into the **Client ID** field.

7. In Red Hat Single Sign-On, paste the value of the **Your Client Secret** into the **Client Secret** field.
8. Enter the required scopes into the **Default Scopes** field. By default, Red Hat Single Sign-On uses the following scopes: **openid profile email**. See the [OAuth Playground](#) for a list of Google scopes.
9. To restrict access to your GSuite organization's members only, enter the G Suite domain into the **Hosted Domain** field.
10. Click **Save**.

9.4.6. LinkedIn

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **LinkedIn**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page in Red Hat Single Sign-On. The sidebar menu on the left includes 'Master', 'Configure' (with sub-items: Realm Settings, Clients, Client Scopes, Roles), 'Identity Providers' (selected), 'User Federation', 'Authentication', and 'Manage' (with sub-items: Groups, Users, Sessions, Events, Import). The main content area is titled 'Add identity provider' and contains the following fields and controls:

- Redirect URI**:
- * Client ID**:
- * Client Secret**:
- Hosted Domain**:
- Use userIp Param**: OFF
- Request refresh token**: OFF
- Default Scopes**:
- Store Tokens**: OFF
- Stored Tokens Readable**: OFF
- Enabled**: ON

3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, [create an app](#).
 - a. After you create the app, click the **Auth** tab.
 - b. Enter the value of **Redirect URI** into the **Authorized redirect URLs for your app** field.
 - c. Note **Your Client ID** and **Your Client Secret**.
5. In Red Hat Single Sign-On, paste the value of the **Client ID** into the **Client ID** field.

6. In Red Hat Single Sign-On, paste the value of the **Client Secret** into the **Client Secret** field.
7. Click **Save**.

9.4.7. Microsoft

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **Microsoft**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page in Red Hat Single Sign-On. The left sidebar is open, showing the 'Identity Providers' menu item. The main content area is titled 'Add identity provider' and contains the following configuration options:

- Redirect URI**:
- * Client ID**:
- * Client Secret**:
- Hosted Domain**:
- Use userIp Param**: OFF
- Request refresh token**: OFF
- Default Scopes**:
- Store Tokens**: OFF
- Stored Tokens Readable**: OFF
- Enabled**: ON

3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, [create a Microsoft app](#).
 - a. Click **Add URL** to add the redirect URL to the Microsoft app.
 - b. Note the **Application ID** and **Application Secret**.
5. In Red Hat Single Sign-On, paste the value of the **Application ID** into the **Client ID** field.
6. In Red Hat Single Sign-On, paste the value of the **Application Secret** into the **Client Secret** field.
7. Click **Save**.

9.4.8. OpenShift 3

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **Openshift**.

Add identity provider

[Identity Providers](#) > Add identity provider

Add identity provider

Redirect URI ⓘ	<input type="text" value="http://keycloak-keycloak.192.168.42.122.xip.io/auth/realms/master/broker/openshift-v3/endpoint"/>
* Client ID ⓘ	<input type="text" value="admin"/>
* Client Secret ⓘ	<input type="text" value="....."/>
* Base URL ⓘ	<input type="text"/>
Default Scopes ⓘ	<input type="text"/>
Store Tokens ⓘ	<input type="checkbox"/> OFF
Stored Tokens Readable ⓘ	<input type="checkbox"/> OFF
Enabled ⓘ	<input checked="" type="checkbox"/> ON
Disable User Info ⓘ	<input type="checkbox"/> OFF
Trust Email ⓘ	<input type="checkbox"/> OFF
GUI order ⓘ	<input type="text"/>
First Login Flow ⓘ	<input type="text" value="first broker login"/>
Post Login Flow ⓘ	<input type="text"/>
	<input type="button" value="Save"/> <input type="button" value="Cancel"/>

3. Copy the value of **Redirect URI** to your clipboard.
4. Register your client using the **oc** command-line tool.

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: v1
metadata:
  name: kc-client 1
  secret: "..." 2
  redirectURIs:
  - "http://www.example.com/" 3
grantMethod: prompt 4
')
```

- 1 The **name** of your OAuth client. Passed as **client_id** request parameter when making requests to **<openshift_master>/oauth/authorize** and **<openshift_master>/oauth/token**.
- 2 The **secret** Red Hat Single Sign-On uses for the **client_secret** request parameter.

- 3 The `redirect_uri` parameter specified in requests to `<openshift_master>/oauth/authorize` and `<openshift_master>/oauth/token` must be equal to (or prefixed by) one of the URIs in
- 4 The `grantMethod` Red Hat Single Sign-On uses to determine the action when this client requests tokens but has not been granted access by the user.
 1. In Red Hat Single Sign-On, paste the value of the **Client ID** into the **Client ID** field.
 2. In Red Hat Single Sign-On, paste the value of the **Client Secret** into the **Client Secret** field.
 3. Click **Save**.

9.4.9. OpenShift 4

Prerequisites

1. Installation of `jq`.
2. **X509_CA_BUNDLE** configured in the container and set to `/var/run/secrets/kubernetes.io/serviceaccount/ca.crt`.

Procedure

1. Run the following command on the command line and note the OpenShift 4 API URL output.

```
curl -s -k -H "Authorization: Bearer $(oc whoami -t)" https://<openshift-user-facing-api-url>/apis/config.openshift.io/v1/infrastructures/cluster | jq ".status.apiServerURL"
```

2. Click **Identity Providers** in the Red Hat Single Sign-On menu.
3. From the **Add provider** list, select **Openshift**.

Add identity provider

Openshift v4

Settings

Mappers

Redirect URI ?	<input type="text" value="http://keycloak-myproject.apps.cluster-lipniki-e986.lipniki-e986.openshiftworkshop.com/auth/realm/quarkus-quickstart/broker/opensh"/>
* Alias ?	<input type="text" value="openshift-v4"/>
Display Name ?	<input type="text"/>
* Client ID ?	<input type="text" value="keycloak-broker"/>
* Client Secret ?	<input type="password" value="*****"/>
* Base Url ?	<input type="text" value="https://api.cluster-lipniki-e986.lipniki-e986.openshiftworkshop.com:6443"/>
Default Scopes ?	<input type="text" value="user:full"/>
Store Tokens ?	<input type="checkbox" value="OFF"/>
Stored Tokens Readable ?	<input type="checkbox" value="OFF"/>
Enabled ?	<input checked="" type="checkbox" value="ON"/>
Disable User Info ?	<input type="checkbox" value="OFF"/>
Trust Email ?	<input type="checkbox" value="OFF"/>
Account Linking Only ?	<input type="checkbox" value="OFF"/>
Hide on Login Page ?	<input type="checkbox" value="OFF"/>
GUI order ?	<input type="text"/>
First Login Flow ?	<input type="text" value="first broker login"/>
Post Login Flow ?	<input type="text"/>
	<input type="button" value="Save"/> <input type="button" value="Cancel"/>

4. Copy the value of **Redirect URI** to your clipboard.
5. Register your client using the **oc** command-line tool.

```
$ oc create -f <(echo '
kind: OAuthClient
apiVersion: oauth.openshift.io/v1
metadata:
  name: keycloak-broker 1
  secret: "..." 2
  redirectURIs:
    - "<copy pasted Redirect URI from OpenShift 4 Identity Providers page>" 3
  grantMethod: prompt 4
')
```

- 1 The **name** of your OAuth client. Passed as **client_id** request parameter when making requests to **<openshift_master>/oauth/authorize** and **<openshift_master>/oauth/token**. The **name** parameter must be the same in the **OAuthClient** object and the Red Hat Single Sign-On configuration.
- 2 The **secret** Red Hat Single Sign-On uses as the **client_secret** request parameter.
- 3 The **redirect_uri** parameter specified in requests to **<openshift_master>/oauth/authorize** and **<openshift_master>/oauth/token** must be equal to (or prefixed by) one of the URIs in **redirectURIs**. The easiest way to configure it correctly is to copy-paste it from Red Hat Single Sign-On OpenShift 4 Identity Provider configuration page (**Redirect URI** field).

- 4 The **grantMethod** Red Hat Single Sign-On uses to determine the action when this client requests tokens but has not been granted access by the user.
 1. In Red Hat Single Sign-On, paste the value of the **Client ID** into the **Client ID** field.
 2. In Red Hat Single Sign-On, paste the value of the **Client Secret** into the **Client Secret** field.
 3. Click **Save**.

See [official OpenShift documentation](#) for more information.

9.4.10. PayPal

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **PayPal**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page. The sidebar on the left includes a 'Master' dropdown and a menu with sections 'Configure' and 'Manage'. Under 'Configure', 'Identity Providers' is selected. The main content area has the title 'Add identity provider' and the breadcrumb 'Identity Providers > Add identity provider'. The configuration fields are as follows:

- Redirect URI**:
- * Client ID**:
- * Client Secret**:
- Hosted Domain**:
- Use userIp Param**: OFF
- Request refresh token**: OFF
- Default Scopes**:
- Store Tokens**: OFF
- Stored Tokens Readable**: OFF
- Enabled**: ON

3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, open the [PayPal Developer applications area](#) .
 - a. Click **Create App** to create a PayPal app.
 - b. Note the Client ID and Client Secret. Click the **Show** link to view the secret.
 - c. Ensure **Connect with PayPal** is checked.

- d. Set the value of the **Return URL** field to the value of **Redirect URI** from Red Hat Single Sign-On.
5. In Red Hat Single Sign-On, paste the value of the **Client ID** into the **Client ID** field.
6. In Red Hat Single Sign-On, paste the value of the **Client Secret** into the **Client Secret** field.
7. Click **Save**.

9.4.11. Stack overflow

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **Stack Overflow**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page in the Red Hat Single Sign-On administration console. The left sidebar is expanded to 'Identity Providers'. The main content area is titled 'Add identity provider' and contains the following fields and controls:

- Redirect URI**:
- * Client ID**:
- * Client Secret**: (with an eye icon for visibility toggle)
- Hosted Domain**:
- Use userIp Param**: OFF
- Request refresh token**: OFF
- Default Scopes**:
- Store Tokens**: OFF
- Stored Tokens Readable**: OFF
- Enabled**: ON

3. In a separate browser tab, log into [registering your application on Stack Apps](#).

Register application

StackExchange 1 help Search Q&A

stackapps Questions Tags Users Badges Unanswered Ask Question

Register Your V2.0 Application

Application Name
Be Unique! Avoid implying an official Stack Exchange relationship

Description

OAuth Domain
example.com, subdomains will be automatically whitelisted

Application Website
Where users can go to read about your application

Application Icon (optional)
Must be hosted by the Stack Exchange Imgur account Enable Client Side OAuth Flow

Register Your Application

Why Register?

Because it's the neighborly thing to do. We like to know who is using our API, and how, so we can have the metrics we need to support your application and improve the API together.

Once it's ready for public consumption, we'll help you promote your registered application here on Stack Apps.

Upon registering, you'll be provided an API key which grants your app a much larger per-day request quota than using the API anonymously.

You'll also receive parameters for authenticating users via OAuth 2.0.

- Enter your application name into the **Application Name** field.
- Enter the OAuth domain into the **OAuth Domain** field.
- Click **Register Your Application**.

Settings

stackapps Questions Tags Users Badges Unanswered Ask Question

Keycloak

Client Id
7209
This Id identifies your application to the Stack Exchange API. Your application client id is **not** secret, and may be safely embedded in distributed binaries.
Pass this as `client_id` in our [OAuth 2.0 flow](#).

Client Secret (reset)
A8M5pezJvqp9G)Nfx6aw9A((
Pass this as `client_secret` in our [OAuth 2.0 flow](#) if your app uses the explicit path.
This **must be** kept secret. Do not embed it in client side code or binaries you intend to distribute. If you need client side authentication, use the implicit OAuth 2.0 flow.

Key
sZA2ICUcqAr6ZkBikpss4w((
Pass this as `key` when making requests against the Stack Exchange API to receive a [higher request quota](#).
This is not considered a secret, and may be safely embed in client side code or distributed binaries.

Description
Keycloak
This **text-only** blurb will be shown to users during authentication.

OAuth Domain

More Info

- [Authentication Statistics](#)
- [API Documentation](#)

- Note the **Client ID** and **Client Secret**.
- In Red Hat Single Sign-On, paste the value of the **Client ID** into the **Client ID** field.

- In Red Hat Single Sign-On, paste the value of the **Client Secret** into the **Client Secret** field.
- Click **Save**.

9.4.12. Twitter

Prerequisites

- A Twitter developer account.

Procedure

- Click **Identity Providers** in the menu.
- From the **Add provider** list, select **Twitter**.

Add identity provider

The screenshot shows the 'Add identity provider' configuration page in the Red Hat Single Sign-On administration console. The left sidebar is dark with a 'Master' dropdown and a menu with options: Configure (Realm Settings, Clients, Client Scopes, Roles, Identity Providers), User Federation, Authentication, and Manage (Groups, Users, Sessions, Events, Import, Export). The main content area is titled 'Identity Providers > Add identity provider' and 'Add identity provider'. It contains several configuration fields and toggle switches:

- Redirect URI**:
- * Client ID**:
- * Client Secret**: (with an eye icon to toggle visibility)
- Default Scopes**:
- Store Tokens**: OFF
- Stored Tokens Readable**: OFF
- Enabled**: ON
- Accepts prompt=none forward from client**: OFF
- Disable User Info**: OFF
- Trust Email**: OFF
- Account Linking Only**: OFF
- Hide on Login Page**: OFF
- GUI order**:
- First Login Flow**:
- Post Login Flow**:

- Copy the value of **Redirect URI** to your clipboard.
- In a separate browser tab, create an app in [Twitter Application Management](#).
 - Enter any value for name and description.

- b. The value for **Website** can be any valid URL except **localhost**.
 - c. Paste the value of the **Redirect URL** into the **Callback URL** field.
 - d. When you create your Twitter app, note the value of **Consumer Key** and **Consumer Secret** in the **Keys and Access Tokens** section.
5. In Red Hat Single Sign-On, paste the value of the **Consumer Key** into the **Client ID** field.
 6. In Red Hat Single Sign-On, paste the value of the **Consumer Secret** into the **Client Secret** field.
 7. Click **Save**.

9.4.13. Instagram

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **Instagram**. Red Hat Single Sign-On displays the configuration page for the Instagram identity provider.

Add identity provider

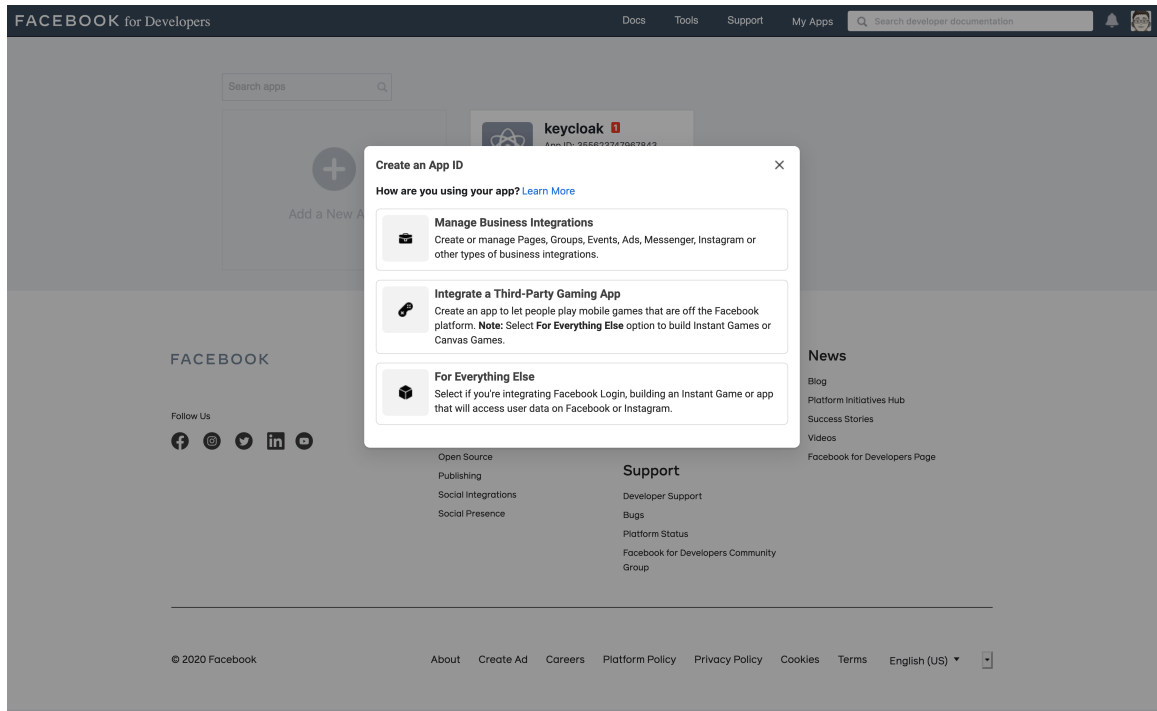
The screenshot shows the 'Add identity provider' configuration page for Instagram in Red Hat Single Sign-On. The page is divided into a left sidebar menu and a main configuration area. The sidebar menu includes options like 'Configure' (Realm Settings, Clients, Client Scopes, Roles) and 'Manage' (User Federation, Authentication, Groups, Users, Sessions, Events, Import, Export). The main configuration area is titled 'Add identity provider' and contains the following fields and controls:

- Redirect URI**:
- Client ID**:
- Client Secret**:
- Default Scopes**:
- Store Tokens**:
- Stored Tokens Readable**:
- Enabled**:
- Accepts prompt=none forward from client**:
- Disable User Info**:
- Trust Email**:
- Account Linking Only**:
- Hide on Login Page**:
- GUI order**:
- First Login Flow**:
- Post Login Flow**:
- Sync Mode**:

At the bottom of the configuration area, there are **Save** and **Cancel** buttons.

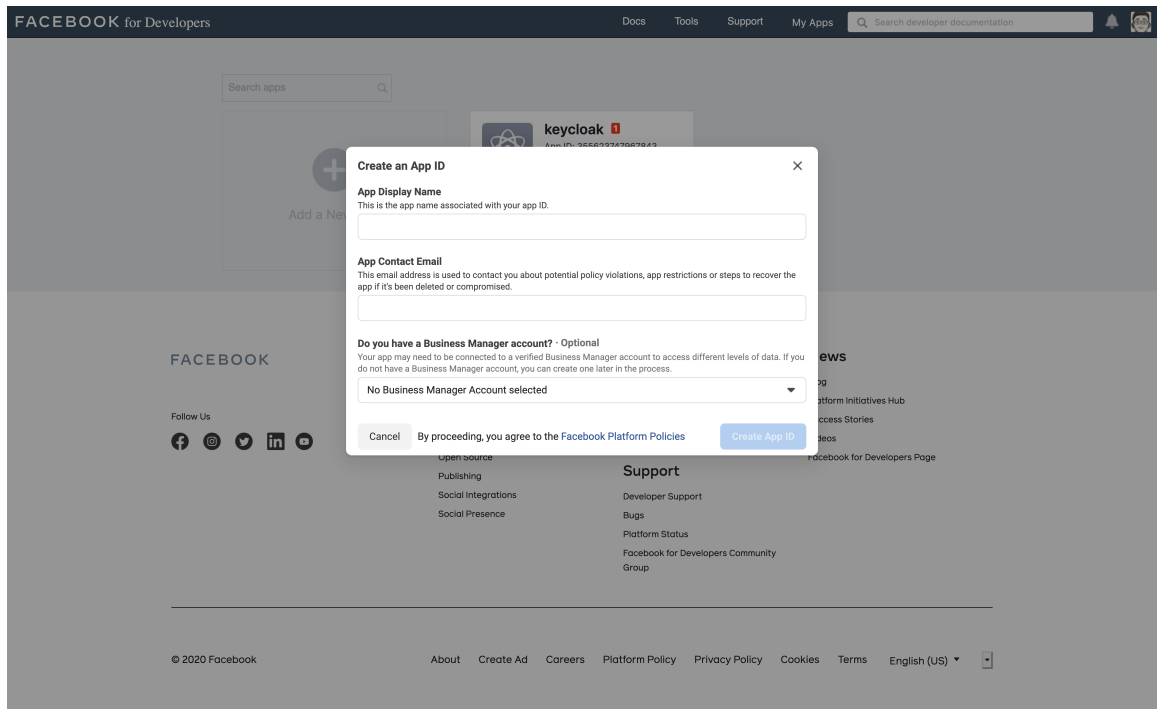
3. Copy the value of **Redirect URI** to your clipboard.
4. In a separate browser tab, open the [Facebook Developer Console](#).
 - a. Click **My Apps**.
 - b. Select **Add a New App**.

Add a new app



- c. Select **For Everything Else**.

Create a new app ID



- d. Fill in all required fields.
- e. Click **Create App**. Facebook then brings you to the dashboard.
- f. In the navigation panel, select **Settings - Basic**.

Add platform

FACEBOOK for Developers Docs Tools Support My Apps Search developer documentation

Test App ID: 274628007307826 In development View Analytics Help

Data Protection Officer Contact Information

The General Data Protection Regulation (GDPR) requires certain companies doing business in the European Union to designate a Data Protection Officer who people can contact for information about how their data is being processed. This contact information will be available to people on Facebook along with other information about your app or website. [Learn More.](#)

Name (optional)

Email

Address

Street Address

Apt/Suite/Other (Optional)

City/District

State/Province/Region ZIP/Postal Code

Country

United States

+ Add Platform

Discard Save Changes

- g. Select **+ Add Platform**.
- h. Click **[Website]**.
- i. Enter a URL for your site.

Add a product

FACEBOOK for Developers Docs Tools Support My Apps Search developer documentation

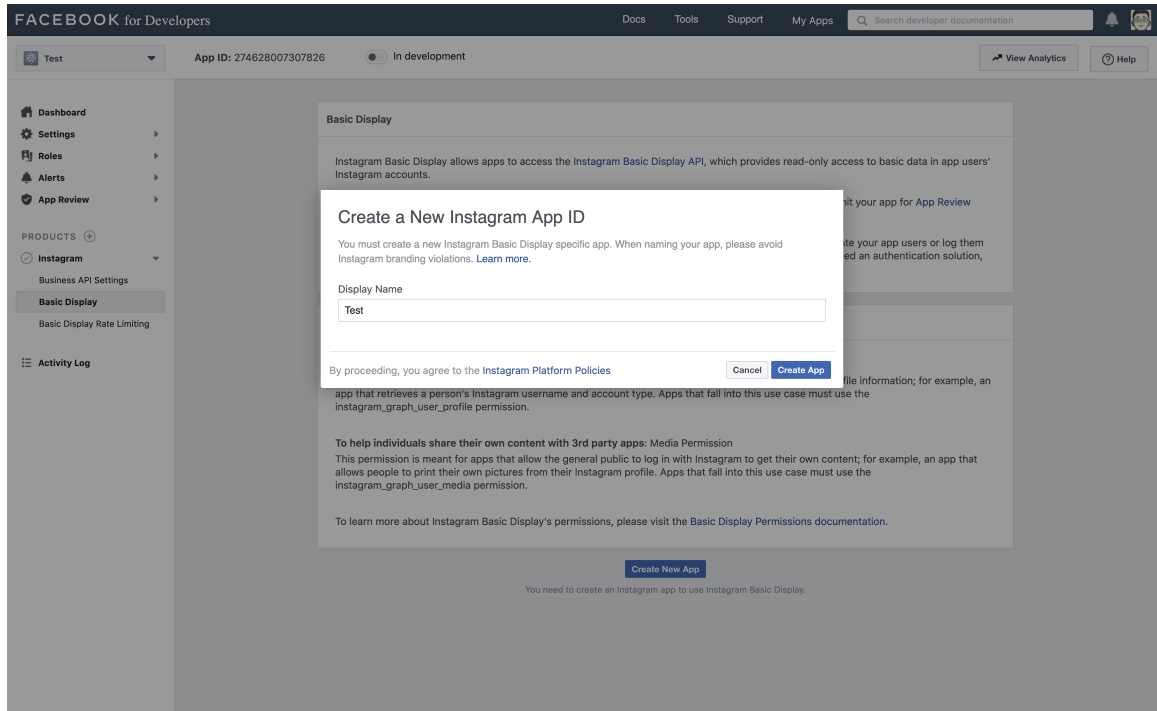
Test App ID: 274628007307826 In development View Analytics Help

Add a Product

<p>Instagram</p> <p>Integrate your app with the Instagram API to let businesses use your app with their Instagram accounts.</p> <p>Read Docs Set Up</p>	<p>Facebook Login</p> <p>The world's number one social login product.</p> <p>Read Docs Set Up</p>	<p>Audience Network</p> <p>Monetize your mobile app or website with native ads from 3 million Facebook advertisers.</p> <p>Read Docs Set Up</p>
<p>Analytics</p> <p>Understand how people engage with your business across apps, devices, platforms and websites.</p> <p>Read Docs Set Up</p>	<p>Messenger</p> <p>Customize the way you interact with people on Messenger.</p> <p>Read Docs Set Up</p>	<p>Webhooks</p> <p>Subscribe to changes and receive updates in real time without calling the API.</p> <p>Read Docs Set Up</p>
<p>Instant Games</p> <p>Create a cross platform HTML5 game hosted on Facebook</p> <p>Read Docs Set Up</p>	<p>Marketing API</p> <p>Integrate Facebook Marketing API with your app.</p> <p>Read Docs Set Up</p>	<p>App Center</p> <p>Get your game discovered by being listed as a featured game on Facebook.</p> <p>Read Docs Set Up</p>

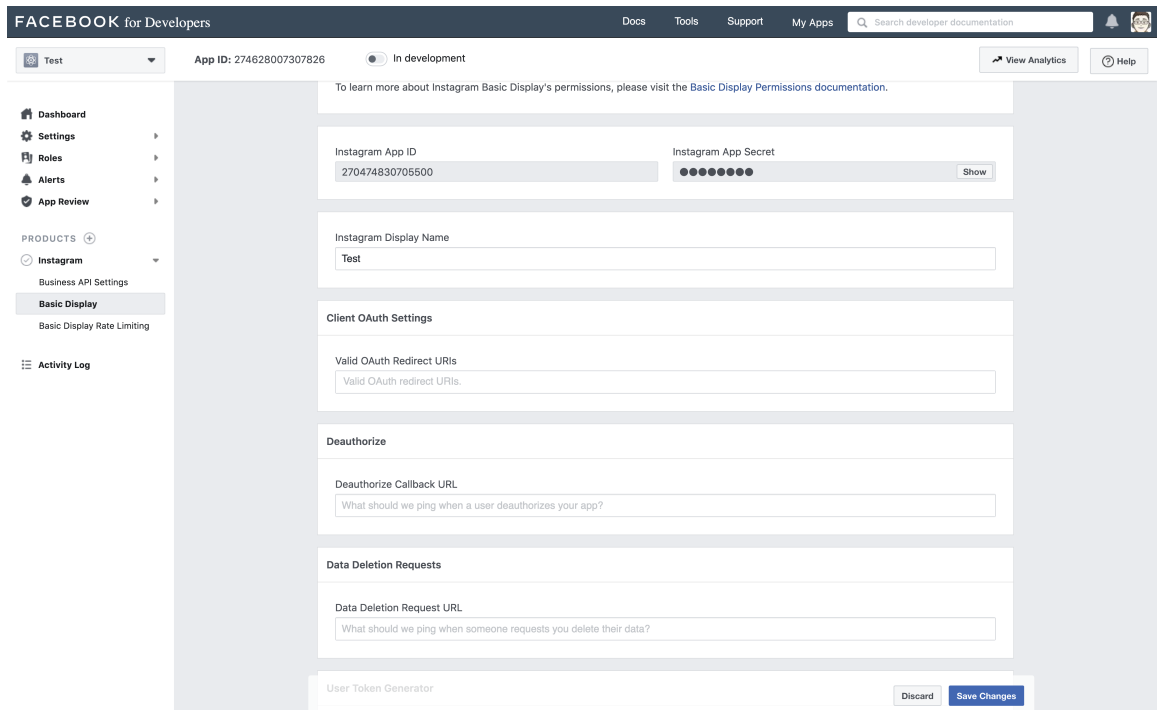
- j. Select **Dashboard** from the menu.
- k. Click **Set Up** in the Instagram box.
- l. Select **Instagram - Basic Display** from the menu.
- m. Click **Create New App**.

Create a new Instagram app ID



- n. Enter a value into **Display Name**.

Setup the app



- o. Paste the **Redirect URL** from Red Hat Single Sign-On into the **Valid OAuth Redirect URIs** field.
- p. Paste the **Redirect URL** from Red Hat Single Sign-On into the **Deauthorize Callback URL** field.
- q. Paste the **Redirect URL** from Red Hat Single Sign-On into the **Data Deletion Request URL** field.

- r. Click **Show** in the **Instagram App Secret** field.
 - s. Note the **Instagram App ID** and the **Instagram App Secret**
 - t. Click **App Review - Requests**.
 - u. Follow the instructions on the screen.
5. In Red Hat Single Sign-On, paste the value of the **Instagram App ID** into the **Client ID** field.
 6. In Red Hat Single Sign-On, paste the value of the **Instagram App Secret** into the **Client Secret** field.
 7. Click **Save**.

9.5. OPENID CONNECT V1.0 IDENTITY PROVIDERS

Red Hat Single Sign-On brokers identity providers based on the OpenID Connect protocol. These identity providers (IDPs) must support the [Authorization Code Flow](#) defined in the specification to authenticate users and authorize access.

Procedure

1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **OpenID Connect v1.0**.

Add identity provider

Master ▼

Configure

- Realm Settings
- Clients
- Client Scopes
- Roles
- Identity Providers**
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions
- Events
- Import
- Export

Identity Providers > Add identity provider

Add identity provider

Redirect URI

* Alias

Display Name

Enabled ON

Store Tokens OFF

Stored Tokens Readable OFF

Trust Email OFF

Account Linking Only OFF

Hide on Login Page OFF

GUI order

First Login Flow

Post Login Flow

OpenID Connect Config

* Authorization URL

Pass login_hint OFF

- Enter your initial configuration options. See [General IDP Configuration](#) for more information about configuration options.

Table 9.2. OpenID connect config

Configuration	Description
Authorization URL	The authorization URL endpoint the OIDC protocol requires.
Token URL	The token URL endpoint the OIDC protocol requires.
Logout URL	The logout URL endpoint in the OIDC protocol. This value is optional.
Backchannel Logout	A background, out-of-band, REST request to the IDP to log out the user. Some IDPs perform logout through browser redirects only, as they may identify sessions using a browser cookie.

Configuration	Description
User Info URL	An endpoint the OIDC protocol defines. This endpoint points to user profile information.
Client Authentication	Defines the Client Authentication method Red Hat Single Sign-On uses with the Authorization Code Flow. In the case of JWT signed with a private key, Red Hat Single Sign-On uses the realm private key. In the other cases, define a client secret. See the Client Authentication specifications for more information.
Client ID	A realm acting as an OIDC client to the external IDP. The realm must have an OIDC client ID if you use the Authorization Code Flow to interact with the external IDP.
Client Secret	Client secret from an external vault . This secret is necessary if you are using the Authorization Code Flow.
Client Assertion Signature Algorithm	Signature algorithm to create JWT assertion as client authentication. In the case of JWT signed with private key or Client secret as jwt, it is required. If no algorithm is specified, the following algorithm is adapted. RS256 is adapted in the case of JWT signed with private key. HS256 is adapted in the case of Client secret as jwt.
Issuer	Red Hat Single Sign-On validates issuer claims, in responses from the IDP, against this value.
Default Scopes	A list of OIDC scopes Red Hat Single Sign-On sends with the authentication request. The default value is openid . A space separates each scope.
Prompt	The prompt parameter in the OIDC specification. Through this parameter, you can force re-authentication and other options. See the specification for more details.

Configuration	Description
Accepts prompt=none forward from client	<p>Specifies if the IDP accepts forwarded authentication requests containing the prompt=none query parameter. If a realm receives an auth request with prompt=none, the realm checks if the user is currently authenticated and returns a login_required error if the user has not logged in. When Red Hat Single Sign-On determines a default IDP for the auth request (using the kc_idp_hint query parameter or having a default IDP for the realm), you can forward the auth request with prompt=none to the default IDP. The default IDP checks the authentication of the user there. Because not all IDPs support requests with prompt=none, Red Hat Single Sign-On uses this switch to indicate that the default IDP supports the parameter before redirecting the authentication request.</p> <p>If the user is unauthenticated in the IDP, the client still receives a login_required error. If the user is authentic in the IDP, the client can still receive an interaction_required error if Red Hat Single Sign-On must display authentication pages that require user interaction. This authentication includes required actions (for example, password change), consent screens, and screens set to display by the first broker login flow or post broker login flow.</p>
Validate Signatures	<p>Specifies if Red Hat Single Sign-On verifies signatures on the external ID Token signed by this IDP. If ON, Red Hat Single Sign-On must know the public key of the external OIDC IDP. For performance purposes, Red Hat Single Sign-On caches the public key of the external OIDC identity provider. If your identity provider's private key is compromised, update your keys and clear the keys cache. See Clearing the cache section for more details.</p>
Use JWKS URL	<p>This switch is applicable if Validate Signatures is ON. If Use JWKS URL is ON, Red Hat Single Sign-On downloads the IDP's public keys from the JWKS URL. New keys download when the identity provider generates a new keypair. If OFF, Red Hat Single Sign-On uses the public key (or certificate) from its database, so when the IDP keypair changes, import the new key to the Red Hat Single Sign-On database as well.</p>

Configuration	Description
JWKS URL	The URL pointing to the location of the IDP JWK keys. For more information, see the JWK specification . If you use an external Red Hat Single Sign-On as an IDP, you can use a URL such as http://broker-keycloak:8180/auth/realms/test/protocol/openid-connect/certs if your brokered Red Hat Single Sign-On is running on http://broker-keycloak:8180 and its realm is test .
Validating Public Key	The public key in PEM format that Red Hat Single Sign-On uses to verify external IDP signatures. This key applies if Use JWKS URL is OFF .
Validating Public Key Id	This setting applies if Use JWKS URL is OFF . This setting specifies the ID of the public key in PEM format. Because there is no standard way for computing key ID from the key, external identity providers can use different algorithms from what Red Hat Single Sign-On uses. If this field's value is not specified, Red Hat Single Sign-On uses the validating public key for all requests, regardless of the key ID sent by the external IDP. When ON , this field's value is the key ID used by Red Hat Single Sign-On for validating signatures from providers and must match the key ID specified by the IDP.

You can import all this configuration data by providing a URL or file that points to OpenID Provider Metadata. If you connect to a Red Hat Single Sign-On external IDP, you can import the IDP settings from `<root>/auth/realms/{realm-name}/well-known/openid-configuration`. This link is a JSON document describing metadata about the IDP.

9.6. SAML V2.0 IDENTITY PROVIDERS
















Red Hat Single Sign-On can broker identity providers based on the SAML v2.0 protocol.

Procedure


1. Click **Identity Providers** in the menu.
2. From the **Add provider** list, select **SAML v2.0**.

Add identity provider

▼ SAML Config

* Single Sign-On Service URL 	<input type="text"/>
Single Logout Service URL 	<input type="text"/>
Backchannel Logout 	<input type="checkbox"/> OFF
NameID Policy Format 	Persistent 
Principal Type 	Subject NameID 
HTTP-POST Binding Response 	<input type="checkbox"/> OFF
HTTP-POST Binding for AuthnRequest 	<input type="checkbox"/> OFF
HTTP-POST Binding Logout 	<input type="checkbox"/> OFF
Want AuthnRequests Signed 	<input type="checkbox"/> OFF
Want Assertions Signed 	<input type="checkbox"/> OFF
Want Assertions Encrypted 	<input type="checkbox"/> OFF
Force Authentication 	<input type="checkbox"/> OFF
Validate Signature 	<input type="checkbox"/> OFF

▼ Import External IDP Config

Import from URL 	<input type="text"/>
Import from file	<input data-bbox="726 1086 746 1115" file="" icon"="" type="button" value="Select file
<input type="button" value="Save"/> <input type="button" value="Cancel"/>	

3. Enter your initial configuration options. See [General IDP Configuration](#) for more information about configuration options. .SAML Config

Configuration	Description
Service Provider Entity ID	The SAML Entity ID that the remote Identity Provider uses to identify requests from this Service Provider. By default, this setting is set to the realms base URL <root>/auth/realms/{realm-name} .
Identity Provider Entity ID	The SAML Entity ID used to validate the Issuer element in received SAML assertions. If empty, no Issuer validation is performed. If your SAML IDP publishes an IDP entity descriptor, the value of this field is specified there.
Single Sign-On Service URL	The SAML endpoint that starts the authentication process. If your SAML IDP publishes an IDP entity descriptor, the value of this field is specified there.

Configuration	Description
Single Logout Service URL	The SAML logout endpoint. If your SAML IDP publishes an IDP entity descriptor, the value of this field is specified there.
Backchannel Logout	Toggle this switch to ON if your SAML IDP supports back channel logout.
NameID Policy Format	The URI reference corresponding to a name identifier format. By default, Red Hat Single Sign-On sets it to urn:oasis:names:tc:SAML:2.0:nameid-format:persistent .
Principal Type	Specifies which part of the SAML assertion will be used to identify and track external user identities. Can be either Subject NameID or SAML attribute (either by name or by friendly name). Subject NameID value can not be set together with 'urn:oasis:names:tc:SAML:2.0:nameid-format:transient' NameID Policy Format value.
Principal Attribute	If a Principal type is non-blank, this field specifies the name ("Attribute [Name]") or the friendly name ("Attribute [Friendly Name]") of the identifying attribute.
Allow create	Allow the external identity provider to create a new identifier to represent the principal.
HTTP-POST Binding Response	Controls the SAML binding in response to any SAML requests sent by an external IDP. When OFF , Red Hat Single Sign-On uses Redirect Binding.
HTTP-POST Binding for AuthnRequest	Controls the SAML binding when requesting authentication from an external IDP. When OFF , Red Hat Single Sign-On uses Redirect Binding.
Want AuthnRequests Signed	When ON , Red Hat Single Sign-On uses the realm's keypair to sign requests sent to the external SAML IDP.
Signature Algorithm	If Want AuthnRequests Signed is ON , the signature algorithm to use.

Configuration	Description
SAML Signature Key Name	Signed SAML documents sent using POST binding contain the identification of signing key in KeyName element, which, by default, contains the Red Hat Single Sign-On key ID. External SAML IDPs can expect a different key name. This switch controls whether KeyName contains: * KEY_ID - Key ID. * CERT_SUBJECT - the subject from the certificate corresponding to the realm key. Microsoft Active Directory Federation Services expect CERT_SUBJECT . * NONE - Red Hat Single Sign-On omits the key name hint from the SAML message.
Force Authentication	The user must enter their credentials at the external IDP even when the user is already logged in.
Validate Signature	When ON , the realm expects SAML requests and responses from the external IDP to be digitally signed.
Validating X509 Certificate	The public certificate Red Hat Single Sign-On uses to validate the signatures of SAML requests and responses from the external IDP.
Sign Service Provider Metadata	When ON , Red Hat Single Sign-On uses the realm's key pair to sign the SAML Service Provider Metadata descriptor .
Pass subject	Controls if Red Hat Single Sign-On forwards a login_hint query parameter to the IDP. Red Hat Single Sign-On adds this field's value to the login_hint parameter in the AuthnRequest's Subject so destination providers can pre-fill their login form.
Attribute Consuming Service Index	Identifies the attribute set to request to the remote IDP. Red Hat Single Sign-On automatically adds the attributes mapped in the identity provider configuration to the autogenerated SP metadata document.
Attribute Consuming Service Name	A descriptive name for the set of attributes that are advertised in the autogenerated SP metadata document.

You can import all configuration data by providing a URL or a file pointing to the SAML IDP entity descriptor of the external IDP. If you are connecting to a Red Hat Single Sign-On external IDP, you can import the IDP settings from the URL `<root>/auth/realms/{realm-name}/protocol/saml/descriptor`. This link is an XML document describing metadata about the IDP. You can also import all this configuration data by providing a URL or XML file pointing to the external SAML IDP's entity descriptor to connect to.

9.6.1. Requesting specific AuthnContexts

Identity Providers facilitate clients specifying constraints on the authentication method verifying the user identity. For example, asking for MFA, Kerberos authentication, or security requirements. These constraints use particular AuthnContext criteria. A client can ask for one or more criteria and specify how the Identity Provider must match the requested AuthnContext, exactly, or by satisfying other equivalents.

You can list the criteria your Service Provider requires by adding ClassRefs or DeclRefs in the Requested AuthnContext Constraints section. Usually, you need to provide either ClassRefs or DeclRefs, so check with your Identity Provider documentation which values are supported. If no ClassRefs or DeclRefs are present, the Identity Provider does not enforce additional constraints.

Table 9.3. Requested AuthnContext Constraints

Configuration	Description
Comparison	The method the Identity Provider uses to evaluate the context requirements. The available values are Exact , Minimum , Maximum , or Better . The default value is Exact .
AuthnContext ClassRefs	The AuthnContext ClassRefs describing the required criteria.
AuthnContext DeclRefs	The AuthnContext DeclRefs describing the required criteria.

9.6.2. SP Descriptor

When you access the provider's SAML SP metadata, look for the **Endpoints** item in the identity provider configuration settings. It contains a **SAML 2.0 Service Provider Metadata** link which generates the SAML entity descriptor for the Service Provider. You can download the descriptor or copy its URL and then import it into the remote Identity Provider.

This metadata is also available publicly by going to the following URL:

```
http[s]://{host:port}/auth/realms/{realm-name}/broker/{broker-alias}/endpoint/descriptor
```

Ensure you save any configuration changes before accessing the descriptor.

9.6.3. Send subject in SAML requests

By default, a social button pointing to a SAML Identity Provider redirects the user to the following login URL:

```
http[s]://{host:port}/auth/realms/${realm-name}/broker/{broker-alias}/login
```

Adding a query parameter named **login_hint** to this URL adds the parameter's value to SAML request as a Subject attribute. If this query parameter is empty, Red Hat Single Sign-On does not add a subject to the request.

Enable the "Pass subject" option to send the subject in SAML requests.

9.7. CLIENT-SUGGESTED IDENTITY PROVIDER

OIDC applications can bypass the Red Hat Single Sign-On login page by hinting at the identity provider they want to use. You can enable this by setting the **kc_idp_hint** query parameter in the Authorization Code Flow authorization endpoint.

With Red Hat Single Sign-On OIDC client adapters, you can specify this query parameter when you access a secured resource in the application.

For example:

```
GET /myapplication.com?kc_idp_hint=facebook HTTP/1.1
Host: localhost:8080
```

In this case, your realm must have an identity provider with a **facebook** alias. If this provider does not exist, the login form is displayed.

If you are using the **keycloak.js** adapter, you can also achieve the same behavior as follows:

```
const keycloak = new Keycloak('keycloak.json');

keycloak.createLoginUrl({
  idpHint: 'facebook'
});
```

With the **kc_idp_hint** query parameter, the client can override the default identity provider if you configure one for the **Identity Provider Redirector** authenticator. The client can disable the automatic redirecting by setting the **kc_idp_hint** query parameter to an empty value.

9.8. MAPPING CLAIMS AND ASSERTIONS

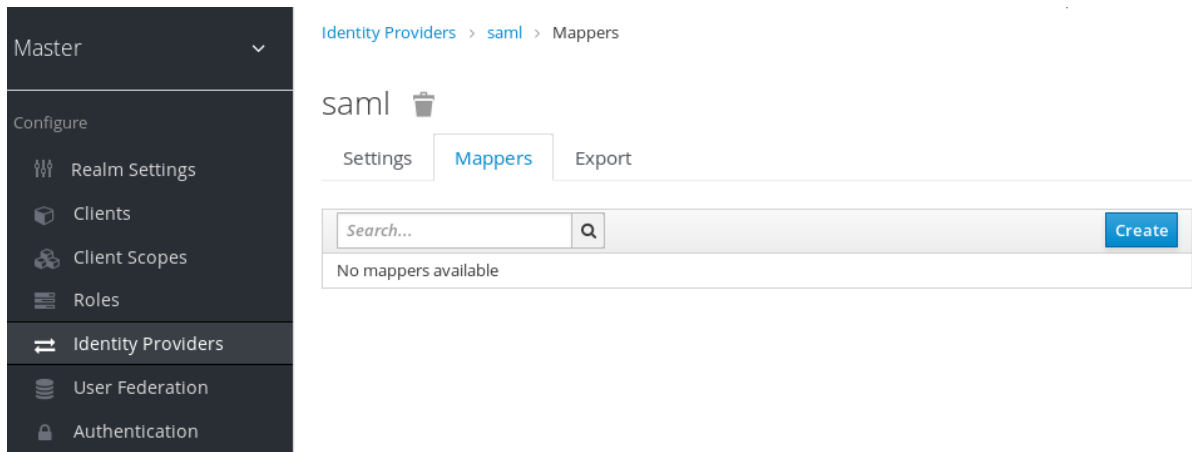
You can import the SAML and OpenID Connect metadata, provided by the external IDP you are authenticating with, into the realm. After importing, you can extract user profile metadata and other information, so you can make it available to your applications.

Each user logging into your realm using an external identity provider has an entry in the local Red Hat Single Sign-On database, based on the metadata from the SAML or OIDC assertions and claims.

Procedure

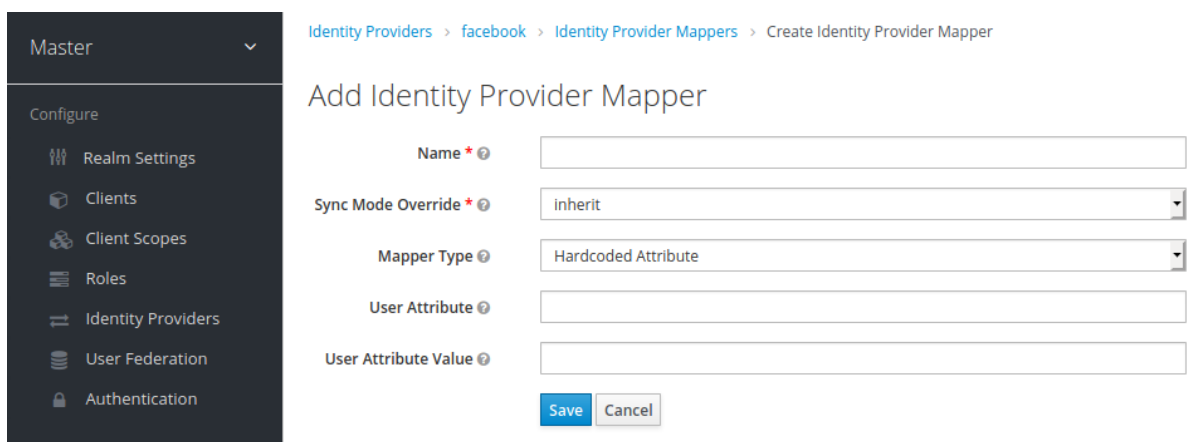
1. Click **Identity Providers** in the menu.
2. Select one of the identity providers in the list.
3. Click the **Mappers** tab.

Identity provider mappers



4. Click **Create**.

Identity provider mapper



5. Select a value for **Sync Mode Override**. The mapper updates user information when users log in repeatedly according to this setting.
 - a. Select **legacy** to use the behavior of the previous Red Hat Single Sign-On version.
 - b. Select **import** to import data from when the user was first created in Red Hat Single Sign-On during the first login to Red Hat Single Sign-On with a particular identity provider.
 - c. Select **force** to update user data at each user login.
 - d. Select **inherit** to use the sync mode configured in the identity provider. All other options will override this sync mode.
6. Select a mapper from the **Mapper Type** list. Hover over the **Mapper Type** for a description of the mapper and configuration to enter for the mapper.
7. Click **Save**.

For JSON-based claims, you can use dot notation for nesting and square brackets to access array fields by index. For example, **contact.address[0].country**.

To investigate the structure of user profile JSON data provided by social providers, you can enable the **DEBUG** level logger **org.keycloak.social.user_profile_dump** in the server's app-server configuration file (domain.xml or standalone.xml).

9.9. AVAILABLE USER SESSION DATA

After a user login from an external IDP, Red Hat Single Sign-On stores user session note data that you can access. This data can be propagated to the client requesting log in using the token or SAML assertion passed back to the client using an appropriate client mapper.

identity_provider

The IDP alias of the broker used to perform the login.

identity_provider_identity

The IDP username of the currently authenticated user. Often, but not always, the same as the Red Hat Single Sign-On username. For example, Red Hat Single Sign-On can link a user john to a Facebook user **john123@gmail.com**. In that case, the value of the user session note is **john123@gmail.com**.

You can use a [Protocol Mapper](#) of type **User Session Note** to propagate this information to your clients.

9.10. FIRST LOGIN FLOW

When users log in through identity brokering, Red Hat Single Sign-On imports and links aspects of the user within the realm's local database. When Red Hat Single Sign-On successfully authenticates users through an external identity provider, two situations can exist:

- Red Hat Single Sign-On has already imported and linked a user account with the authenticated identity provider account. In this case, Red Hat Single Sign-On authenticates as the existing user and redirects back to the application.
- No account exists for this user in Red Hat Single Sign-On. Usually, you register and import a new account into the Red Hat Single Sign-On database, but there may be an existing Red Hat Single Sign-On account with the same email address. Automatically linking the existing local account to the external identity provider is a potential security hole. You cannot always trust the information you get from the external identity provider.

Different organizations have different requirements when dealing with some of these situations. With Red Hat Single Sign-On, you can use the **First Login Flow** option in the IDP settings to choose a [workflow](#) for a user logging in from an external IDP for the first time. By default, the **First Login Flow** option points to the **first broker login** flow, but you can use your flow or different flows for different identity providers.

The flow is in the Admin Console under the **Authentication** tab. When you choose the **First Broker Login** flow, you see the authenticators used by default. You can re-configure the existing flow. For example, you can disable some authenticators, mark some of them as **required**, or configure some authenticators.

9.10.1. Default first login flow authenticators

Review Profile

- This authenticator displays the profile information page, so the users can review their profile that Red Hat Single Sign-On retrieves from an identity provider.
- You can set the **Update Profile On First Login** option in the **Actions** menu.
- When **ON**, users are presented with the profile page requesting additional information to federate the user's identities.

- When **missing**, users are presented with the profile page if the identity provider does not provide mandatory information, such as email, first name, or last name.
- When **OFF**, the profile page does not display unless the user clicks in a later phase on the **Review profile info** link in the page displayed by the **Confirm Link Existing Account** authenticator.

Create User If Unique

This authenticator checks if there is already an existing Red Hat Single Sign-On account with the same email or username like the account from the identity provider. If it's not, then the authenticator just creates a new local Red Hat Single Sign-On account and links it with the identity provider and the whole flow is finished. Otherwise it goes to the next **Handle Existing Account** subflow. If you always want to ensure that there is no duplicated account, you can mark this authenticator as **REQUIRED**. In this case, the user will see the error page if there is an existing Red Hat Single Sign-On account and the user will need to link the identity provider account through Account management.

- This authenticator verifies that there is already a Red Hat Single Sign-On account with the same email or username as the identity provider's account.
- If an account does not exist, the authenticator creates a local Red Hat Single Sign-On account, links this account with the identity provider, and terminates the flow.
- If an account exists, the authenticator implements the next **Handle Existing Account** subflow.
- To ensure there is no duplicated account, you can mark this authenticator as **REQUIRED**. The user sees the error page if a Red Hat Single Sign-On account exists, and users must link their identity provider account through Account management.

Confirm Link Existing Account

- On the information page, users see a Red Hat Single Sign-On account with the same email. Users can review their profile again and use a different email or username. The flow restarts and goes back to the **Review Profile** authenticator.
- Alternatively, users can confirm that they want to link their identity provider account with their existing Red Hat Single Sign-On account.
- Disable this authenticator if you do not want users to see this confirmation page and go straight to linking identity provider account by email verification or re-authentication.

Verify Existing Account By Email

- This authenticator is **ALTERNATIVE** by default. Red Hat Single Sign-On uses this authenticator if the realm has an SMTP setup configured.
- The authenticator sends an email to users to confirm that they want to link the identity provider with their Red Hat Single Sign-On account.
- Disable this authenticator if you do not want to confirm linking by email, but want users to reauthenticate with their password.

Verify Existing Account By Re-authentication

- Use this authenticator if the email authenticator is not available. For example, you have not configured SMTP for your realm. This authenticator displays a login screen for users to authenticate to link their Red Hat Single Sign-On account with the Identity Provider.
- Users can also re-authenticate with another identity provider already linked to their Red Hat Single Sign-On account.
- You can force users to use OTP. Otherwise, it is optional and used if you have set OTP for the user account.

9.10.2. Automatically link existing first login flow



WARNING

The AutoLink authenticator is dangerous in a generic environment where users can register themselves using arbitrary usernames or email addresses. Do not use this authenticator unless you are carefully curating user registration and assigning usernames and email addresses.

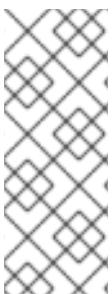
To configure a first login flow that links users automatically without prompting, create a new flow with the following two authenticators:

Create User If Unique

This authenticator ensures Red Hat Single Sign-On handles unique users. Set the authenticator requirement to **Alternative**.

Automatically Set Existing User

This authenticator sets an existing user to the authentication context without verification. Set the authenticator requirement to "Alternative".



NOTE

This setup is the simplest setup available, but it is possible to use other authenticators. For example: * You can add the Review Profile authenticator to the beginning of the flow if you want end users to confirm their profile information. * You can add authentication mechanisms to this flow, forcing a user to verify their credentials. Adding authentication mechanisms requires a complex flow. For example, you can set the "Automatically Set Existing User" and "Password Form" as "Required" in an "Alternative" sub-flow.

9.10.3. Disabling automatic user creation

The Default first login flow looks up the Red Hat Single Sign-On account matching the external identity and offers to link them. If no matching Red Hat Single Sign-On account exists, the flow automatically creates one.

This default behavior may be unsuitable for some setups. One example is when you use a read-only LDAP user store, where all users are pre-created. In this case, you must switch off automatic user creation.

To disable user creation:

Procedure

1. Click **Authentication** in the menu.
2. Select **First Broker Login** from the list.
3. Set **Create User If Unique** to **DISABLED**.
4. Set **Confirm Link Existing Account** to **DISABLED**.

This configuration also implies that Red Hat Single Sign-On itself won't be able to determine which internal account would correspond to the external identity. Therefore, the **Verify Existing Account By Re-authentication** authenticator will ask the user to provide both username and password.

9.10.4. Detect existing user first login flow

In order to configure a first login flow in which:

- only users already registered in this realm can log in,
- users are automatically linked without being prompted,

create a new flow with the following two authenticators:

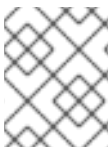
Detect Existing Broker User

This authenticator ensures that unique users are handled. Set the authenticator requirement to **Mandatory**.

Automatically Set Existing User

Automatically sets an existing user to the authentication context without any verification. Set the authenticator requirement to **Mandatory**.

You have to set the **First Login Flow** of the identity provider configuration to that flow. You could set the also set **Sync Mode** to **force** if you want to update the user profile (Last Name, First Name...) with the identity provider attributes.



NOTE

This flow can be used if you want to delegate the identity to other identity providers (such as github, facebook ...) but you want to manage which users that can log in.

With this configuration, Red Hat Single Sign-On is unable to determine which internal account corresponds to the external identity. The **Verify Existing Account By Re-authentication** authenticator asks the provider for the username and password.

9.11. RETRIEVING EXTERNAL IDP TOKENS

With Red Hat Single Sign-On, you can store tokens and responses from the authentication process with the external IDP using the **Store Token** configuration option on the IDP's settings page.

Application code can retrieve these tokens and responses to import extra user information or to request the external IDP securely. For example, an application can use the Google token to use other Google services and REST APIs. To retrieve a token for a particular identity provider, send a request as follows:

```
GET /auth/realms/{realm}/broker/{provider_alias}/token HTTP/1.1  
Host: localhost:8080  
Authorization: Bearer <KEYCLOAK ACCESS TOKEN>
```

An application must authenticate with Red Hat Single Sign-On and receive an access token. This access token must have the **broker** client-level role **read-token** set, so the user must have a role mapping for this role, and the client application must have that role within its scope. In this case, since you are accessing a protected service in Red Hat Single Sign-On, send the access token issued by Red Hat Single Sign-On during the user authentication. You can assign this role to newly imported users in the broker configuration page by setting the **Stored Tokens Readable** switch to **ON**.

These external tokens can be re-established by logging in again through the provider or using the client-initiated account linking API.

9.12. IDENTITY BROKER LOGOUT

When logging out, Red Hat Single Sign-On sends a request to the external identity provider that is used to log in initially and logs the user out of this identity provider. You can skip this behavior and avoid logging out of the external identity provider. See [adapter logout documentation](#) for more information.

CHAPTER 10. SSO PROTOCOLS

This section discusses authentication protocols, the Red Hat Single Sign-On authentication server and how applications, secured by the Red Hat Single Sign-On authentication server, interact with these protocols.

10.1. OPENID CONNECT

[OpenID Connect](#) (OIDC) is an authentication protocol that is an extension of [OAuth 2.0](#).

OAuth 2.0 is a framework for building authorization protocols and is incomplete. OIDC, however, is a full authentication and authorization protocol that uses the [Json Web Token](#) (JWT) standards. The JWT standards define an identity token JSON format and methods to digitally sign and encrypt data in a compact and web-friendly way.

In general, OIDC implements two use cases. The first case is an application requesting that a Red Hat Single Sign-On server authenticates a user. Upon successful login, the application receives an *identity token* and an *access token*. The *identity token* contains user information including user name, email, and profile information. The realm digitally signs the *access token* which contains access information (such as user role mappings) that applications use to determine the resources users can access in the application.

The second use case is a client accessing remote services.

- The client requests an *access token* from Red Hat Single Sign-On to invoke on remote services on behalf of the user.
- Red Hat Single Sign-On authenticates the user and asks the user for consent to grant access to the requesting client.
- The client receives the *access token* which is digitally signed by the realm.
- The client makes REST requests on remote services using the *access token*.
- The remote REST service extracts the *access token*.
- The remote REST service verifies the tokens signature.
- The remote REST service decides, based on access information within the token, to process or reject the request.

10.1.1. OIDC auth flows

OIDC has several methods, or flows, that clients or applications can use to authenticate users and receive *identity* and *access* tokens. The method depends on the type of application or client requesting access.

10.1.1.1. Authorization Code Flow

The Authorization Code Flow is a browser-based protocol and suits authenticating and authorizing browser-based applications. It uses browser redirects to obtain *identity* and *access* tokens.

1. A user connects to an application using a browser. The application detects the user is not logged into the application.

2. The application redirects the browser to Red Hat Single Sign-On for authentication.
3. The application passes a callback URL as a query parameter in the browser redirect. Red Hat Single Sign-On uses the parameter upon successful authentication.
4. Red Hat Single Sign-On authenticates the user and creates a one-time, short lived, temporary code.
5. Red Hat Single Sign-On redirects to the application using the callback URL and adds the temporary code as a query parameter in the callback URL.
6. The application extracts the temporary code and makes a background REST invocation to Red Hat Single Sign-On to exchange the code for an *identity* and *access* and *refresh* token. To prevent replay attacks, the temporary code cannot be used more than once.



NOTE

A system is vulnerable to a stolen token for the lifetime of that token. For security and scalability reasons, access tokens are generally set to expire quickly so subsequent token requests fail. If a token expires, an application can obtain a new access token using the additional *refresh* token sent by the login protocol.

Confidential clients provide client secrets when they exchange the temporary codes for tokens. *Public* clients are not required to provide client secrets. *Public* clients are secure when HTTPS is strictly enforced and redirect URIs registered for the client are strictly controlled. HTML5/JavaScript clients have to be *public* clients because there is no way to securely transmit the client secret to HTML5/JavaScript clients. For more details, see the [Managing Clients](#) chapter.

Red Hat Single Sign-On also supports the [Proof Key for Code Exchange](#) specification.

10.1.1.2. Implicit Flow

The Implicit Flow is a browser-based protocol. It is similar to the Authorization Code Flow but with fewer requests and no refresh tokens.



NOTE

The possibility exists of access tokens leaking in the browser history when tokens are transmitted via redirect URIs (see below).

Also, this flow does not provide clients with refresh tokens. Therefore, access tokens have to be long-lived or users have to re-authenticate when they expire.

We do not advise using this flow. This flow is supported because it is in the OIDC and OAuth 2.0 specification.

The protocol works as follows:

1. A user connects to an application using a browser. The application detects the user is not logged into the application.
2. The application redirects the browser to Red Hat Single Sign-On for authentication.
3. The application passes a callback URL as a query parameter in the browser redirect. Red Hat Single Sign-On uses the query parameter upon successful authentication.

4. Red Hat Single Sign-On authenticates the user and creates an *identity* and *access* token. Red Hat Single Sign-On redirects to the application using the callback URL and additionally adds the *identity* and *access* tokens as a query parameter in the callback URL.
5. The application extracts the *identity* and *access* tokens from the callback URL.

10.1.1.3. Resource owner password credentials grant (Direct Access Grants)

Direct Access Grants are used by REST clients to obtain tokens on behalf of users. It is a HTTP POST request that contains:

- The credentials of the user. The credentials are sent within form parameters.
- The id of the client.
- The clients secret (if it is a confidential client).

The HTTP response contains the *identity*, *access*, and *refresh* tokens.

10.1.1.4. Client credentials grant

The *Client Credentials Grant* creates a token based on the metadata and permissions of a service account associated with the client instead of obtaining a token that works on behalf of an external user. *Client Credentials Grants* are used by REST clients.

See the [Service Accounts](#) chapter for more information.

10.1.1.5. Device authorization grant

This is used by clients running on internet-connected devices that have limited input capabilities or lack a suitable browser. Here's a brief summary of the protocol:

1. The application requests Red Hat Single Sign-On a device code and a user code. Red Hat Single Sign-On creates a device code and a user code. Red Hat Single Sign-On returns a response including the device code and the user code to the application.
2. The application provides the user with the user code and the verification URI. The user accesses a verification URI to be authenticated by using another browser.
3. The application repeatedly polls Red Hat Single Sign-On to find out if the user completed the user authorization. If user authentication is complete, the application exchanges the device code for an *identity*, *access* and *refresh* token.

10.1.1.6. Client initiated backchannel authentication grant

This feature is used by clients who want to initiate the authentication flow by communicating with the OpenID Provider directly without redirect through the user's browser like OAuth 2.0's authorization code grant. Here's a brief summary of the protocol:

1. The client requests Red Hat Single Sign-On an *auth_req_id* that identifies the authentication request made by the client. Red Hat Single Sign-On creates the *auth_req_id*.
2. After receiving this *auth_req_id*, this client repeatedly needs to poll Red Hat Single Sign-On to obtain an Access Token, Refresh Token and ID Token from Red Hat Single Sign-On in return for the *auth_req_id* until the user is authenticated.

An administrator can configure Client Initiated Backchannel Authentication (CIBA) related operations as **CIBA Policy** per realm.

Also please refer to other places of Red Hat Single Sign-On documentation like [Backchannel Authentication Endpoint section](#) of Securing Applications and Services Guide and [Client Initiated Backchannel Authentication Grant section](#) of Securing Applications and Services Guide.

10.1.1.6.1. CIBA Policy

An administrator carries out the following operations on the **Admin Console** :

- Open the **Authentication** → **CIBA Policy** tab.
- Configure items and click **Save**.

The configurable items and their description follow.

Configuration	Description
Backchannel Token Delivery Mode	Specifying how the CD (Consumption Device) gets the authentication result and related tokens. There are three modes, "poll", "ping" and "push". Red Hat Single Sign-On only supports "poll". The default setting is "poll". This configuration is required. For more details, see CIBA Specification .
Expires In	The expiration time of the "auth_req_id" in seconds since the authentication request was received. The default setting is 120. This configuration is required. For more details, see CIBA Specification .
Interval	The interval in seconds the CD (Consumption Device) needs to wait for between polling requests to the token endpoint. The default setting is 5. This configuration is optional. For more details, see CIBA Specification .
Authentication Requested User Hint	The way of identifying the end-user for whom authentication is being requested. The default setting is "login_hint". There are three modes, "login_hint", "login_hint_token" and "id_token_hint". Red Hat Single Sign-On only supports "login_hint". This configuration is required. For more details, see CIBA Specification .

10.1.1.6.2. Provider Setting

The CIBA grant uses the following two providers.

1. Authentication Channel Provider : provides the communication between Red Hat Single Sign-On and the entity that actually authenticates the user via AD (Authentication Device).

2. User Resolver Provider : get **UserModel** of Red Hat Single Sign-On from the information provided by the client to identify the user.

Red Hat Single Sign-On has both default providers. However, the administrator needs to set up Authentication Channel Provider like this:

```
<spi name="ciba-auth-channel">
  <default-provider>ciba-http-auth-channel</default-provider>
  <provider name="ciba-http-auth-channel" enabled="true">
    <properties>
      <property name="httpAuthenticationChannelUri"
value="https://backend.internal.example.com/auth"/>
    </properties>
  </provider>
</spi>
```

The configurable items and their description follow.

Configuration	Description
httpAuthenticationChannelUri	Specifying URI of the entity that actually authenticates the user via AD (Authentication Device).

10.1.1.6.3. Authentication Channel Provider

CIBA standard document does not specify how to authenticate the user by AD. Therefore, it might be implemented at the discretion of products. Red Hat Single Sign-On delegates this authentication to an external authentication entity. To communicate with the authentication entity, Red Hat Single Sign-On provides Authentication Channel Provider.

Its implementation of Red Hat Single Sign-On assumes that the authentication entity is under the control of the administrator of Red Hat Single Sign-On so that Red Hat Single Sign-On trusts the authentication entity. It is not recommended to use the authentication entity that the administrator of Red Hat Single Sign-On cannot control.

Authentication Channel Provider is provided as SPI provider so that users of Red Hat Single Sign-On can implement their own provider in order to meet their environment. Red Hat Single Sign-On provides its default provider called HTTP Authentication Channel Provider that uses HTTP to communicate with the authentication entity.

If a user of Red Hat Single Sign-On user want to use the HTTP Authentication Channel Provider, they need to know its contract between Red Hat Single Sign-On and the authentication entity consisting of the following two parts.

Authentication Delegation Request/Response

Red Hat Single Sign-On sends an authentication request to the authentication entity.

Authentication Result Notification/ACK

The authentication entity notifies the result of the authentication to Red Hat Single Sign-On.

Authentication Delegation Request/Response consists of the following messaging.

Authentication Delegation Request

The request is sent from Red Hat Single Sign-On to the authentication entity to ask it for user authentication by AD.

POST [delegation_reception]

- Headers

Name	Value	Description
Content-Type	application/json	The message body is json formatted.
Authorization	Bearer [token]	The [token] is used when the authentication entity notifies the result of the authentication to Red Hat Single Sign-On.

- Parameters

Type	Name	Description
Path	delegation_reception	The endpoint provided by the authentication entity to receive the delegation request

- Body

Name	Description
login_hint	It tells the authentication entity who is authenticated by AD. By default, it is the user's "username". This field is required and was defined by CIBA standard document.
scope	It tells which scopes the authentication entity gets consent from the authenticated user. This field is required and was defined by CIBA standard document.
is_consent_required	It shows whether the authentication entity needs to get consent from the authenticated user about the scope. This field is required.

Name	Description
binding_message	Its value is intended to be shown in both CD and AD's UI to make the user recognize that the authentication by AD is triggered by CD. This field is optional and was defined by CIBA standard document.
acr_values	It tells the requesting Authentication Context Class Reference from CD. This field is optional and was defined by CIBA standard document.

Authentication Delegation Response

The response is returned from the authentication entity to Red Hat Single Sign-On to notify that the authentication entity received the authentication request from Red Hat Single Sign-On.

- Responses

HTTP Status Code	Description
201	It notifies Red Hat Single Sign-On of receiving the authentication delegation request.

Authentication Result Notification/ACK consists of the following messaging.

Authentication Result Notification

The authentication entity sends the result of the authentication request to Red Hat Single Sign-On.

POST /auth/realms/[realm]/protocol/openid-connect/ext/ciba/auth/callback

- Headers

Name	Value	Description
Content-Type	application/json	The message body is json formatted.
Authorization	Bearer [token]	The [token] must be the one the authentication entity has received from Red Hat Single Sign-On in Authentication Delegation Request.

- Parameters

Type	Name	Description
Path	realm	The realm name

- Body

Name	Description
status	It tells the result of user authentication by AD. It must be one of the following status. SUCCEED : The authentication by AD has been successfully completed. UNAUTHORIZED : The authentication by AD has not been completed. CANCELLED : The authentication by AD has been cancelled by the user.

Authentication Result ACK

The response is returned from Red Hat Single Sign-On to the authentication entity to notify Red Hat Single Sign-On received the result of user authentication by AD from the authentication entity.

- Responses

HTTP Status Code	Description
200	It notifies the authentication entity of receiving the notification of the authentication result.

10.1.1.6.4. User Resolver Provider

Even if the same user, its representation may differ in each CD, Red Hat Single Sign-On and the authentication entity.

For CD, Red Hat Single Sign-On and the authentication entity to recognize the same user, this User Resolver Provider converts their own user representations among them.

User Resolver Provider is provided as SPI provider so that users of Red Hat Single Sign-On can implement their own provider in order to meet their environment. Red Hat Single Sign-On provides its default provider called Default User Resolver Provider that has the following characteristics.

- Only support **login_hint** parameter and is used as default.
- **username** of UserModel in Red Hat Single Sign-On is used to represent the user on CD, Red Hat Single Sign-On and the authentication entity.

10.1.2. OIDC Logout

OIDC has four specifications relevant to logout mechanisms. These specifications are in draft status:

1. [Session Management](#)
2. [RP-Initiated Logout](#)
3. [Front-Channel Logout](#)
4. [Back-Channel Logout](#)

Again since all of this is described in the OIDC specification we will only give a brief overview here.

10.1.2.1. Session Management

This is a browser-based logout. The application obtains session status information from Red Hat Single Sign-On at a regular basis. When the session is terminated at Red Hat Single Sign-On the application will notice and trigger it's own logout.

10.1.2.2. RP-Initiated Logout

This is also a browser-based logout where the logout starts by redirecting the user to a specific endpoint at Red Hat Single Sign-On. This redirect usually happens when the user clicks the **Log Out** link on the page of some application, which previously used Red Hat Single Sign-On to authenticate the user.

Once the user is redirected to the logout endpoint, Red Hat Single Sign-On is going to send logout requests to clients to let them to invalidate their local user sessions, and potentially redirect the user to some URL once the logout process is finished. The user might be optionally requested to confirm the logout in case the `id_token_hint` parameter was not used. After logout, the user is automatically redirected to the specified `post_logout_redirect_uri` as long as it is provided as a parameter. Note that you need to include either the `client_id` or `id_token_hint` parameter in case the `post_logout_redirect_uri` is included. Also the `post_logout_redirect_uri` parameter needs to match one of the **Valid Post Logout Redirect URIs** specified in the client configuration.

Depending on the client configuration, logout requests can be sent to clients through the front-channel or through the back-channel. For the frontend browser clients, which rely on the Session Management described in the previous section, Red Hat Single Sign-On does not need to send any logout requests to them; these clients automatically detect that SSO session in the browser is logged out.

10.1.2.3. Frontchannel Logout

To configure clients to receive logout requests through the front-channel, look at the [Front-Channel Logout](#) client setting. When using this method, consider the following:

- Logout requests sent by Red Hat Single Sign-On to clients rely on the browser and on embedded **iframes** that are rendered for the logout page.
- By being based on **iframes**, front-channel logout might be impacted by Content Security Policies (CSP) and logout requests might be blocked.
- If the user closes the browser prior to rendering the logout page or before logout requests are actually sent to clients, their sessions at the client might not be invalidated.



NOTE

Consider using Back-Channel Logout as it provides a more reliable and secure approach to log out users and terminate their sessions on the clients.

If the client is not enabled with front-channel logout, then Red Hat Single Sign-On is going to try first to send logout requests through the back-channel using the [Back-Channel Logout URL](#). If not defined, the server is going to fall back to using the [Admin URL](#).

10.1.2.4. Backchannel Logout

This is a non browser-based logout that uses direct backchannel communication between Red Hat Single Sign-On and clients. Red Hat Single Sign-On sends a HTTP POST request containing a logout token to all clients logged into Red Hat Single Sign-On. These requests are sent to a registered backchannel logout URLs at Red Hat Single Sign-On and are supposed to trigger a logout at client side.

10.1.3. Red Hat Single Sign-On server OIDC URI endpoints

The following is a list of OIDC endpoints that Red Hat Single Sign-On publishes. These endpoints can be used when a non-Red Hat Single Sign-On client adapter uses OIDC to communicate with the authentication server. They are all relative URLs. The root of the URL consists of the HTTP(S) protocol, hostname, and optionally the path: For example

```
https://localhost:8080/auth
```

/realms/{realm-name}/protocol/openid-connect/auth

Used for obtaining a temporary code in the Authorization Code Flow or obtaining tokens using the Implicit Flow, Direct Grants, or Client Grants.

/realms/{realm-name}/protocol/openid-connect/token

Used by the Authorization Code Flow to convert a temporary code into a token.

/realms/{realm-name}/protocol/openid-connect/logout

Used for performing logouts.

/realms/{realm-name}/protocol/openid-connect/userinfo

Used for the User Info service described in the OIDC specification.

/realms/{realm-name}/protocol/openid-connect/revoke

Used for OAuth 2.0 Token Revocation described in [RFC7009](#).

/realms/{realm-name}/protocol/openid-connect/certs

Used for the JSON Web Key Set (JWKS) containing the public keys used to verify any JSON Web Token (jwks_uri)

/realms/{realm-name}/protocol/openid-connect/auth/device

Used for Device Authorization Grant to obtain a device code and a user code.

/realms/{realm-name}/protocol/openid-connect/ext/ciba/auth

This is the URL endpoint for Client Initiated Backchannel Authentication Grant to obtain an auth_req_id that identifies the authentication request made by the client.

/realms/{realm-name}/protocol/openid-connect/logout/backchannel-logout

This is the URL endpoint for performing backchannel logouts described in the OIDC specification.

In all of these, replace {realm-name} with the name of the realm.

10.2. SAML

[SAML 2.0](#) is a similar specification to OIDC but more mature. It is descended from SOAP and web service messaging specifications so is generally more verbose than OIDC. SAML 2.0 is an authentication protocol that exchanges XML documents between authentication servers and applications. XML

signatures and encryption are used to verify requests and responses.

In general, SAML implements two use cases.

The first use case is an application that requests the Red Hat Single Sign-On server authenticates a user. Upon successful login, the application will receive an XML document. This document contains an SAML assertion that specifies user attributes. The realm digitally signs the the document which contains access information (such as user role mappings) that applications use to determine the resources users are allowed to access in the application.

The second use case is a client accessing remote services. The client requests a SAML assertion from Red Hat Single Sign-On to invoke on remote services on behalf of the user.

10.2.1. SAML bindings

Red Hat Single Sign-On supports three binding types.

10.2.1.1. Redirect binding

Redirect binding uses a series of browser redirect URIs to exchange information.

1. A user connects to an application using a browser. The application detects the user is not authenticated.
2. The application generates an XML authentication request document and encodes it as a query parameter in a URI. The URI is used to redirect to the Red Hat Single Sign-On server. Depending on your settings, the application can also digitally sign the XML document and include the signature as a query parameter in the redirect URI to Red Hat Single Sign-On. This signature is used to validate the client that sends the request.
3. The browser redirects to Red Hat Single Sign-On.
4. The server extracts the XML auth request document and verifies the digital signature, if required.
5. The user enters their authentication credentials.
6. After authentication, the server generates an XML authentication response document. The document contains a SAML assertion that holds metadata about the user, including name, address, email, and any role mappings the user has. The document is usually digitally signed using XML signatures, and may also be encrypted.
7. The XML authentication response document is encoded as a query parameter in a redirect URI. The URI brings the browser back to the application. The digital signature is also included as a query parameter.
8. The application receives the redirect URI and extracts the XML document.
9. The application verifies the realm's signature to ensure it is receiving a valid authentication response. The information inside the SAML assertion is used to make access decisions or display user data.

10.2.1.2. POST binding

POST binding is similar to *Redirect* binding but *POST* binding exchanges XML documents using *POST* requests instead of using *GET* requests. *POST* Binding uses JavaScript to make the browser send a

POST request to the Red Hat Single Sign-On server or application when exchanging documents. HTTP responds with an HTML document which contains an HTML form containing embedded JavaScript. When the page loads, the JavaScript automatically invokes the form.

POST binding is recommended due to two restrictions:

- **Security** – With *Redirect* binding, the SAML response is part of the URL. It is less secure as it is possible to capture the response in logs.
- **Size** – Sending the document in the HTTP payload provides more scope for large amounts of data than in a limited URL.

10.2.1.3. ECP

Enhanced Client or Proxy (ECP) is a SAML v.2.0 profile which allows the exchange of SAML attributes outside the context of a web browser. It is often used by REST or SOAP-based clients.

10.2.2. Red Hat Single Sign-On Server SAML URI Endpoints

Red Hat Single Sign-On has one endpoint for all SAML requests.

`http(s)://authserver.host/auth/realms/{realm-name}/protocol/saml`

All bindings use this endpoint.

10.3. OPENID CONNECT COMPARED TO SAML

The following lists a number of factors to consider when choosing a protocol.

For most purposes, Red Hat Single Sign-On recommends using OIDC.

OIDC

- OIDC is specifically designed to work with the web.
- OIDC is suited for HTML5/JavaScript applications because it is easier to implement on the client side than SAML.
- OIDC tokens are in the JSON format which makes them easier for Javascript to consume.
- OIDC has features to make security implementation easier. For example, see the [iframe trick](#) that the specification uses to determine a users login status.

SAML

- SAML is designed as a layer to work on top of the web.
- SAML can be more verbose than OIDC.
- Users pick SAML over OIDC because there is a perception that it is mature.
- Users pick SAML over OIDC existing applications that are secured with it.

10.4. DOCKER REGISTRY V2 AUTHENTICATION

**NOTE**

Docker authentication is disabled by default. To enable docker authentication, see [Profiles](#).

[Docker Registry V2 Authentication](#) is a protocol, similar to OIDC, that authenticates users against Docker registries. Red Hat Single Sign-On's implementation of this protocol lets Docker clients use a Red Hat Single Sign-On authentication server to authenticate against a registry. This protocol uses standard token and signature mechanisms but it does deviate from a true OIDC implementation. It deviates by using a very specific JSON format for requests and responses as well as mapping repository names and permissions to the OAuth scope mechanism.

10.4.1. Docker authentication flow

The authentication flow is described in the [Docker API documentation](#). The following is a summary from the perspective of the Red Hat Single Sign-On authentication server:

- Perform a **docker login**.
- The Docker client requests a resource from the Docker registry. If the resource is protected and no authentication token is in the request, the Docker registry server responds with a 401 HTTP message with some information on the permissions that are required and the location of the authorization server.
- The Docker client constructs an authentication request based on the 401 HTTP message from the Docker registry. The client uses the locally cached credentials (from the **docker login** command) as part of the [HTTP Basic Authentication](#) request to the Red Hat Single Sign-On authentication server.
- The Red Hat Single Sign-On authentication server attempts to authenticate the user and return a JSON body containing an OAuth-style Bearer token.
- The Docker client receives a bearer token from the JSON response and uses it in the authorization header to request the protected resource.
- The Docker registry receives the new request for the protected resource with the token from the Red Hat Single Sign-On server. The registry validates the token and grants access to the requested resource (if appropriate).

**NOTE**

Red Hat Single Sign-On does not create a browser SSO session after successful authentication with the Docker protocol. The browser SSO session does not use the Docker protocol as it cannot refresh tokens or obtain the status of a token or session from the Red Hat Single Sign-On server; therefore a browser SSO session is not necessary. For more details, see the [transient session](#) section.

10.4.2. Red Hat Single Sign-On Docker Registry v2 Authentication Server URI Endpoints

Red Hat Single Sign-On has one endpoint for all Docker auth v2 requests.

`http(s)://authserver.host/auth/realms/{realm-name}/protocol/docker-v2`

CHAPTER 11. CONTROLLING ACCESS TO THE ADMIN CONSOLE

Each realm created on the Red Hat Single Sign-On has a dedicated Admin Console from which that realm can be managed. The **master** realm is a special realm that allows admins to manage more than one realm on the system. You can also define fine-grained access to users in different realms to manage the server. This chapter goes over all the scenarios for this.

11.1. MASTER REALM ACCESS CONTROL

The **master** realm in Red Hat Single Sign-On is a special realm and treated differently than other realms. Users in the Red Hat Single Sign-On **master** realm can be granted permission to manage zero or more realms that are deployed on the Red Hat Single Sign-On server. When a realm is created, Red Hat Single Sign-On automatically creates various roles that grant fine-grain permissions to access that new realm. Access to The Admin Console and Admin REST endpoints can be controlled by mapping these roles to users in the **master** realm. It's possible to create multiple super users, as well as users that can only manage specific realms.

11.1.1. Global roles

There are two realm-level roles in the **master** realm. These are:

- admin
- create-realm

Users with the **admin** role are super users and have full access to manage any realm on the server. Users with the **create-realm** role are allowed to create new realms. They will be granted full access to any new realm they create.

11.1.2. Realm specific roles

Admin users within the **master** realm can be granted management privileges to one or more other realms in the system. Each realm in Red Hat Single Sign-On is represented by a client in the **master** realm. The name of the client is **<realm name>-realm**. These clients each have client-level roles defined which define varying level of access to manage an individual realm.

The roles available are:

- view-realm
- view-users
- view-clients
- view-events
- manage-realm
- manage-users
- create-client
- manage-clients

- manage-events
- view-identity-providers
- manage-identity-providers
- impersonation

Assign the roles you want to your users and they will only be able to use that specific part of the administration console.



IMPORTANT

Admins with the **manage-users** role will only be able to assign admin roles to users that they themselves have. So, if an admin has the **manage-users** role but doesn't have the **manage-realm** role, they will not be able to assign this role.

11.2. DEDICATED REALM ADMIN CONSOLES

Each realm has a dedicated Admin Console that can be accessed by going to the url **/auth/admin/{realm-name}/console**. Users within that realm can be granted realm management permissions by assigning specific user role mappings.

Each realm has a built-in client called **realm-management**. You can view this client by going to the **Clients** left menu item of your realm. This client defines client-level roles that specify permissions that can be granted to manage the realm.

- view-realm
- view-users
- view-clients
- view-events
- manage-realm
- manage-users
- create-client
- manage-clients
- manage-events
- view-identity-providers
- manage-identity-providers
- impersonation

Assign the roles you want to your users and they will only be able to use that specific part of the administration console.

11.3. FINE GRAIN ADMIN PERMISSIONS

**NOTE**

Fine Grain Admin Permissions is **Technology Preview** and is not fully supported. This feature is disabled by default.

To enable start the server with **-Dkeycloak.profile=preview** or **-Dkeycloak.profile.feature.admin_fine_grained_authz=enabled**. For more details see [Profiles](#).

Sometimes roles like **manage-realm** or **manage-users** are too coarse grain and you want to create restricted admin accounts that have more fine grain permissions. Red Hat Single Sign-On allows you to define and assign restricted access policies for managing a realm. Things like:

- Managing one specific client
- Managing users that belong to a specific group
- Managing membership of a group
- Limited user management.
- Fine grain impersonation control
- Being able to assign a specific restricted set of roles to users.
- Being able to assign a specific restricted set of roles to a composite role.
- Being able to assign a specific restricted set of roles to a client's scope.
- New general policies for viewing and managing users, groups, roles, and clients.

There are some important things to note about fine grain admin permissions:

- Fine grain admin permissions were implemented on top of [Authorization Services](#). It is highly recommended that you read up on those features before diving into fine grain permissions.
- Fine grain permissions are only available within [dedicated admin consoles](#) and admins defined within those realms. You cannot define cross-realm fine grain permissions.
- Fine grain permissions are used to grant additional permissions. You cannot override the default behavior of the built in admin roles.

11.3.1. Managing one specific client

Let's look first at allowing an admin to manage one client and one client only. In our example, we have a realm called **test** and a client called **sales-application**. In the realm **test** we will give a user in that realm permission to only manage that application.

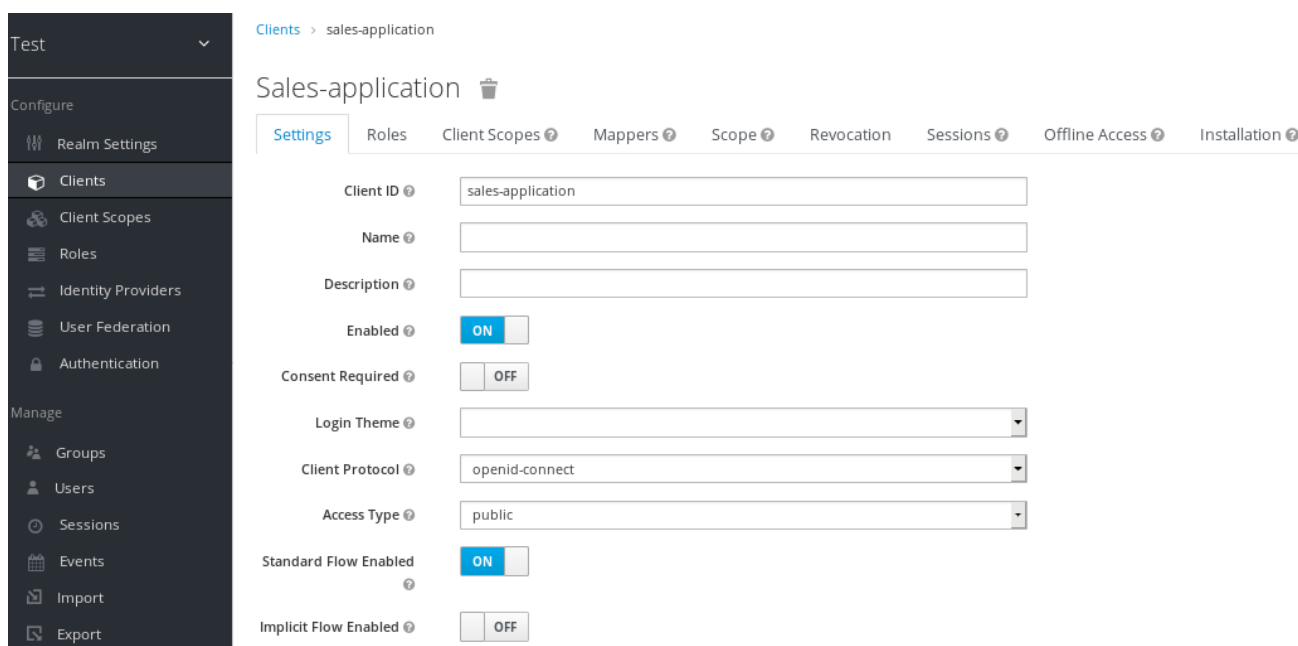
**IMPORTANT**

You cannot do cross realm fine grain permissions. Admins in the **master** realm are limited to the predefined admin roles defined in previous chapters.

11.3.1.1. Permission setup

The first thing we must do is login to the Admin Console so we can set up permissions for that client. We navigate to the management section of the client, we want to define fine-grain permissions for.

Client management



Test

Configure

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

Manage

- Groups
- Users
- Sessions
- Events
- Import
- Export

Clients > sales-application

Sales-application

Settings Roles Client Scopes Mappers Scope Revocation Sessions Offline Access Installation

Client ID sales-application

Name

Description

Enabled ON

Consent Required OFF

Login Theme

Client Protocol openid-connect

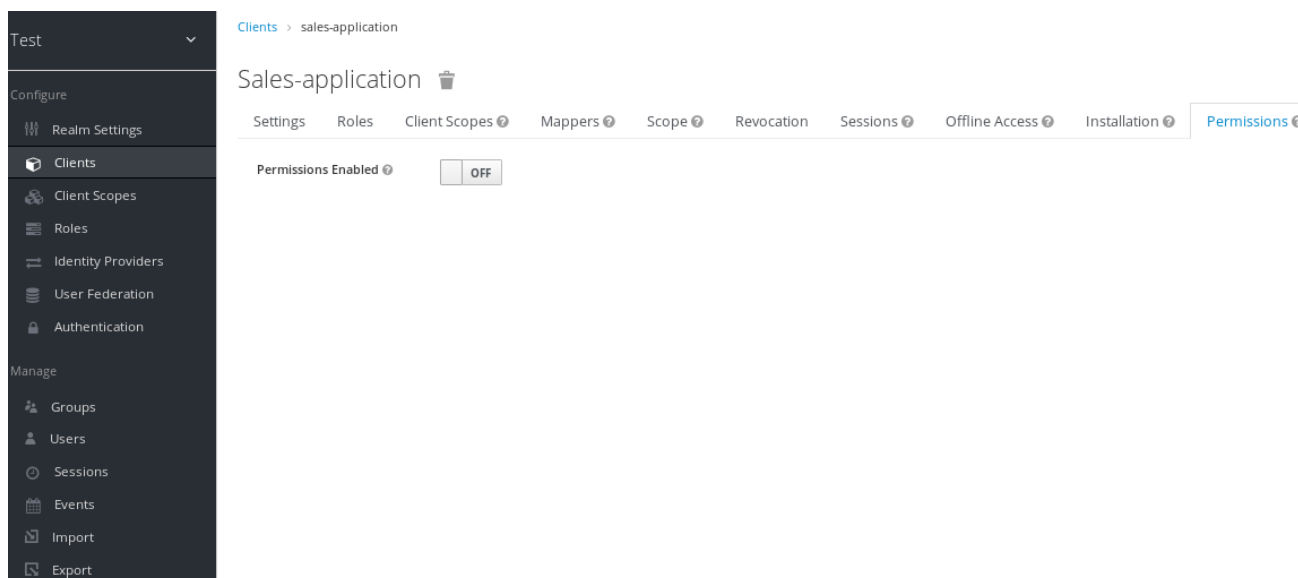
Access Type public

Standard Flow Enabled ON

Implicit Flow Enabled OFF

You should see a tab menu item called **Permissions**. Click on that tab.

Client permissions tab



Test

Configure

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

Manage

- Groups
- Users
- Sessions
- Events
- Import
- Export

Clients > sales-application

Sales-application

Settings Roles Client Scopes Mappers Scope Revocation Sessions Offline Access Installation Permissions

Permissions Enabled OFF

By default, each client is not enabled to do fine grain permissions. So turn the **Permissions Enabled** switch to on to initialize permissions.



IMPORTANT

If you turn the **Permissions Enabled** switch to off, it will delete any and all permissions you have defined for this client.

Client permissions tab

Clients > sales-application

Sales-application

Settings Roles Client Scopes Mappers Scope Revocation Sessions Offline Access Installation **Permissions**

Permissions Enabled ON

scope-name	Description	Actions
view	Policies that decide if an administrator can view this client	Edit
manage	Policies that decide if an administrator can manage this client	Edit
configure	Reduced management permissions for administrator. Cannot set scope, template, or protocol mappers.	Edit
map-roles	Policies that decide if an administrator can map roles defined by this client	Edit
map-roles-client-scope	Policies that decide if an administrator can apply roles defined by this client to the client scope of another client	Edit
map-roles-composite	Policies that decide if an administrator can apply roles defined by this client as a composite to another role	Edit
token-exchange	Policies that decide which clients are allowed exchange tokens for a token that is targeted to this client.	Edit

When you switch **Permissions Enabled** to on, it initializes various permission objects behind the scenes using [Authorization Services](#). For this example, we're interested in the **manage** permission for the client. Clicking on that will redirect you to the permission that handles the **manage** permission for the client. All authorization objects are contained in the **realm-management** client's **Authorization** tab.

Client manage permission

Clients > realm-management > Authorization > Permissions > manage.permission.client.9b183749-2a87-4591-88f2-549bc4a352f3

Manage.permission.client.9b183749-2a87-4591-88f2-549bc4a352f3

Name

Description

Resource

Scopes

Apply Policy

No policies assigned.

Decision Strategy

When first initialized the **manage** permission does not have any policies associated with it. You will need to create one by going to the policy tab. To get there fast, click on the **Authorization** link shown in the above image. Then click on the policies tab.

There's a pull down menu on this page called **Create policy**. There's a multitude of policies you can define. You can define a policy that is associated with a role or a group or even define rules in JavaScript. For this simple example, we're going to create a **User Policy**.

User policy

Test ▾

Configure

- ⚙️ Realm Settings
- 📁 Clients
- 🔗 Client Scopes
- 📄 Roles
- 🔗 Identity Providers
- 📄 User Federation
- 🔒 Authentication

Manage

- 👤 Groups

Clients > realm-management > Authorization > Policies > Add User Policy

Add User Policy

Name * ⓘ

Description ⓘ

Users * ⓘ

Username	Actions
sales-admin	Remove

Logic ⓘ

This policy will match a hard-coded user in the user database. In this case, it is the **sales-admin** user. We must then go back to the **sales-application** client's **manage** permission page and assign the policy to the permission object.

Assign user policy

Test ▾

Configure

- ⚙️ Realm Settings
- 📁 Clients
- 🔗 Client Scopes
- 📄 Roles
- 🔗 Identity Providers
- 📄 User Federation
- 🔒 Authentication

Manage

- 👤 Groups
- 👤 Users
- 🕒 Sessions

Clients > realm-management > Authorization > Permissions > manage.permission.client.9b183749-2a87-4591-88f2-549bc4a352f3

Manage.permission.client.9b183749-2a87-4591-88f2-549bc4a352f3 🗑️

Name * ⓘ

Description ⓘ

Resource ⓘ

Scopes * ⓘ

Apply Policy ⓘ

Name	Description	Actions
sales-app-admin		Remove

Decision Strategy ⓘ

The **sales-admin** user can now has permission to manage the **sales-application** client.

There's one more thing we have to do. Go to the **Role Mappings** tab and assign the **query-clients** role to the **sales-admin**.

Assign query-clients

Why do you have to do this? This role tells the Admin Console what menu items to render when the **sales-admin** visits the Admin Console. The **query-clients** role tells the Admin Console that it should render client menus for the **sales-admin** user.

IMPORTANT If you do not set the **query-clients** role, restricted admins like **sales-admin** will not see any menu options when they log into the Admin Console

11.3.1.2. Testing it out

Next, we log out of the master realm and re-login to the [dedicated admin console](#) for the **test** realm using the **sales-admin** as a username. This is located under `/auth/admin/test/console`.

Sales admin login

Client ID	Enabled	Base URL	Actions		
sales-application	True	Not defined	Edit	Export	Delete

This admin is now able to manage this one client.

11.3.2. Restrict user role mapping

Another thing you might want to do is to restrict the set of roles an admin is allowed to assign to a user. Continuing our last example, let's expand the permission set of the 'sales-admin' user so that he can also control which users are allowed to access this application. Through fine grain permissions, we can enable it so that the **sales-admin** can only assign roles that grant specific access to the **sales-application**. We can also restrict it so that the admin can only map roles and not perform any other types of user administration.

The **sales-application** has defined three different client roles.

Sales application roles

Clients > sales-application

Sales-application

Settings Roles Client Scopes Mappers Scope Revocation Sessions Offline Access Installation

Permissions

Role Name	Composite	Description	Actions	
viewLeads	False		Edit	Delete
leader-creator	False		Edit	Delete
admin	False		Edit	Delete

We want the **sales-admin** user to be able to map these roles to any user in the system. The first step to do this is to allow the role to be mapped by the admin. If we click on the **viewLeads** role, you'll see that there is a **Permissions** tab for this role.

View leads role permission tab

Clients > sales-application > Roles > viewLeads

ViewLeads

Details Attributes Permissions Users in Role

Role Name viewLeads

Description

Composite Roles OFF

If we click on that tab and turn the **Permissions Enabled** on, you'll see that there are a number of actions we can apply policies to.

View leads permissions

Clients > sales-application > viewLeads

ViewLeads

Details Attributes Permissions Users in Role

Permissions Enabled

scope-name	Description	Actions
map-role	Policies that decide if an administrator can map this role to a user or group	Edit
map-role-client-scope	Policies that decide if an administrator can apply this role to the client scope of a client	Edit
map-role-composite	Policies that decide if an administrator can apply this role as a composite to another role	Edit

The one we are interested in is **map-role**. Click on this permission and add the same User Policy that was created in the earlier example.

Map-roles permission

Test

Configure

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

Manage

- Groups
- Users
- Sessions

Clients > realm-management > Authorization > Permissions > map-role.permission.5a39b51f-8eff-4ce7-8e13-c49511067192

Map-role.permission.5a39b51f-8eff-4ce7-8e13-c49511067192

Name *

Description

Resource

Scopes *

Apply Policy

Name	Description	Actions
sales-app-admin		Remove

Decision Strategy

What we've done is say that the **sales-admin** can map the **viewLeads** role. What we have not done is specify which users the admin is allowed to map this role too. To do that we must go to the **Users** section of the admin console for this realm. Clicking on the **Users** left menu item brings us to the users interface of the realm. You should see a **Permissions** tab. Click on that and enable it.

Users permissions

Test

Configure

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

Manage

- Groups
- Users

Users

Lookup

Permissions Enabled

scope-name	Description	Actions
view	Policies that decide if an administrator can view all users in realm	Edit
manage	Policies that decide if an administrator can manage all users in the realm	Edit
map-roles	Policies that decide if administrator can map roles for all users	Edit
manage-group-membership	Policies that decide if an administrator can manage group membership for all users in the realm. This is used in conjunction with specific group policy	Edit
impersonate	Policies that decide if administrator can impersonate other users	Edit
user-impersonated	Policies that decide which users can be impersonated. These policies are applied to the user being impersonated.	Edit

The permission we are interested in is **map-roles**. This is a restrictive policy in that it only allows admins the ability to map roles to a user. If we click on the **map-roles** permission and again add the User Policy we created for this, our **sales-admin** will be able to map roles to any user.

The last thing we have to do is add the **view-users** role to the **sales-admin**. This will allow the admin to view users in the realm he wants to add the **sales-application** roles to.

Add view-users

The screenshot shows the Keycloak admin console interface. On the left is a dark sidebar with a 'Test' dropdown and a menu containing 'Configure', 'Realm', 'Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity', 'Providers', 'User', 'Federation', and 'Authentication'. The main content area is titled 'Sales-admin' and has tabs for 'Details', 'Attributes', 'Credentials', 'Role Mappings' (selected), 'Groups', 'Consents', and 'Sessions'. Below the tabs are two sections: 'Realm Roles' and 'Client Roles'. The 'Client Roles' section is expanded to show 'realm-managemen...'. Each section contains four columns: 'Available Roles', 'Assigned Roles', and 'Effective Roles'. In the 'Client Roles' section, the 'Available Roles' list includes 'view-clients', 'view-events', 'view-identity-providers', 'view-realm', and 'view-users' (which is highlighted in blue). Below the 'Available Roles' list is an 'Add selected >' button. The 'Assigned Roles' column has a '<< Remove selected' button. The 'Effective Roles' column shows 'offline_access' and 'uma_authorization'.

11.3.2.1. Testing it out

Next, we log out of the master realm and re-login to the [dedicated admin console](#) for the **test** realm using the **sales-admin** as a username. This is located under `/auth/admin/test/console`.

You will see that now the **sales-admin** can view users in the system. If you select one of the users you'll see that each user detail page is read only, except for the **Role Mappings** tab. Going to this tab you'll find that there are no **Available** roles for the admin to map to the user except when we browse the **sales-application** roles.

Add viewleads

The screenshot shows the Keycloak admin console interface for the 'Test' realm. The sidebar menu is different, showing 'Configure', 'Clients', 'Manage', 'Groups', and 'Users'. The main content area is titled 'Salesman' and has tabs for 'Details', 'Attributes', 'Role Mappings' (selected), 'Groups', 'Consents', and 'Sessions'. Below the tabs are two sections: 'Realm Roles' and 'Client Roles'. The 'Client Roles' section is expanded to show 'sales-application'. Each section contains four columns: 'Available Roles', 'Assigned Roles', and 'Effective Roles'. In the 'Client Roles' section, the 'Available Roles' list includes 'viewLeads' (which is highlighted in blue). Below the 'Available Roles' list is an 'Add selected >' button. The 'Assigned Roles' column has a '<< Remove selected' button. The 'Effective Roles' column is empty.

We've only specified that the **sales-admin** can map the **viewLeads** role.

11.3.2.2. Per client map-roles shortcut

It would be tedious if we had to do this for every client role that the **sales-application** published. To make things easier, there's a way to specify that an admin can map any role defined by a client. If we log

back into the admin console to our master realm admin and go back to the **sales-application** permissions page, you'll see the **map-roles** permission.

Client map-roles permission

The screenshot shows the admin console interface for the 'Sales-application' client. The breadcrumb navigation is 'Clients > sales-application'. The page title is 'Sales-application' with a trash icon. Below the title are tabs for 'Settings', 'Roles', 'Client Scopes', 'Mappers', 'Scope', 'Revocation', 'Sessions', 'Offline Access', 'Installation', and 'Permissions'. The 'Permissions' tab is active. A toggle for 'Permissions Enabled' is set to 'ON'. Below this is a table with columns 'scope-name', 'Description', and 'Actions'.

scope-name	Description	Actions
view	Policies that decide if an administrator can view this client	Edit
manage	Policies that decide if an administrator can manage this client	Edit
configure	Reduced management permissions for administrator. Cannot set scope, template, or protocol mappers.	Edit
map-roles	Policies that decide if an administrator can map roles defined by this client	Edit
map-roles-client-scope	Policies that decide if an administrator can apply roles defined by this client to the client scope of another client	Edit
map-roles-composite	Policies that decide if an administrator can apply roles defined by this client as a composite to another role	Edit
token-exchange	Policies that decide which clients are allowed exchange tokens for a token that is targeted to this client.	Edit

If you grant access to this particular permission to an admin, that admin will be able map any role defined by the client.

11.3.3. Full list of permissions

You can do a lot more with fine grain permissions beyond managing a specific client or the specific roles of a client. This chapter defines the whole list of permission types that can be described for a realm.

11.3.3.1. Role

When going to the **Permissions** tab for a specific role, you will see these permission types listed.

map-role

Policies that decide if an admin can map this role to a user. These policies only specify that the role can be mapped to a user, not that the admin is allowed to perform user role mapping tasks. The admin will also have to have manage or role mapping permissions. See [Users Permissions](#) for more information.

map-role-composite

Policies that decide if an admin can map this role as a composite to another role. An admin can define roles for a client if he has to manage permissions for that client but he will not be able to add composites to those roles unless he has the **map-role-composite** privileges for the role he wants to add as a composite.

map-role-client-scope

Policies that decide if an admin can apply this role to the scope of a client. Even if the admin can manage the client, he will not have permission to create tokens for that client that contain this role unless this privilege is granted.

11.3.3.2. Client

When going to the **Permissions** tab for a specific client, you will see these permission types listed.

view

Policies that decide if an admin can view the client's configuration.

manage

Policies that decide if an admin can view and manage the client's configuration. There are some

issues with this in that privileges could be leaked unintentionally. For example, the admin could define a protocol mapper that hardcoded a role even if the admin does not have privileges to map the role to the client's scope. This is currently the limitation of protocol mappers as they don't have a way to assign individual permissions to them like roles do.

configure

Reduced set of privileges to manage the client. It is like the **manage** scope except the admin is not allowed to define protocol mappers, change the client template, or the client's scope.

map-roles

Policies that decide if an admin can map any role defined by the client to a user. This is a shortcut, easy-of-use feature to avoid having to define policies for each and every role defined by the client.

map-roles-composite

Policies that decide if an admin can map any role defined by the client as a composite to another role. This is a shortcut, easy-of-use feature to avoid having to define policies for each and every role defined by the client.

map-roles-client-scope

Policies that decide if an admin can map any role defined by the client to the scope of another client. This is a shortcut, easy-of-use feature to avoid having to define policies for each and every role defined by the client.

11.3.3.3. Users

When going to the **Permissions** tab for all users, you will see these permission types listed.

view

Policies that decide if an admin can view all users in the realm.

manage

Policies that decide if an admin can manage all users in the realm. This permission grants the admin the privilege to perform user role mappings, but it does not specify which roles the admin is allowed to map. You'll need to define the privilege for each role you want the admin to be able to map.

map-roles

This is a subset of the privileges granted by the **manage** scope. In this case the admin is only allowed to map roles. The admin is not allowed to perform any other user management operation. Also, like **manage**, the roles that the admin is allowed to apply must be specified per role or per set of roles if dealing with client roles.

manage-group-membership

Similar to **map-roles** except that it pertains to group membership: which groups a user can be added or removed from. These policies just grant the admin permission to manage group membership, not which groups the admin is allowed to manage membership for. You'll have to specify policies for each group's **manage-members** permission.

impersonate

Policies that decide if the admin is allowed to impersonate other users. These policies are applied to the admin's attributes and role mappings.

user-impersonated

Policies that decide which users can be impersonated. These policies will be applied to the user being impersonated. For example, you might want to define a policy that will forbid anybody from impersonating a user that has admin privileges.

11.3.3.4. Group

When going to the **Permissions** tab for a specific group, you will see these permission types listed.

view

Policies that decide if the admin can view information about the group.

manage

Policies that decide if the admin can manage the configuration of the group.

view-members

Policies that decide if the admin can view the user details of members of the group.

manage-members

Policies that decide if the admin can manage the users that belong to this group.

manage-membership

Policies that decide if an admin can change the membership of the group. Add or remove members from the group.

CHAPTER 12. MANAGING OPENID CONNECT AND SAML CLIENTS

Clients are entities that can request authentication of a user. Clients come in two forms. The first type of client is an application that wants to participate in single-sign-on. These clients just want Red Hat Single Sign-On to provide security for them. The other type of client is one that is requesting an access token so that it can invoke other services on behalf of the authenticated user. This section discusses various aspects around configuring clients and various ways to do it.

12.1. OIDC CLIENTS

[OpenID Connect](#) is the recommended protocol to secure applications. It was designed from the ground up to be web friendly and it works best with HTML5/JavaScript applications.

12.1.1. Creating an OpenID Connect Client

To protect an application that uses the OpenID connect protocol, you create a client.

Procedure

1. Click **Clients** in the menu.
2. Click **Create** to go to the **Add Client** page.

Add client

The screenshot shows the 'Add Client' page. On the left is a dark sidebar menu with 'Clients' highlighted. The main area has a breadcrumb 'Clients > Add Client' and the title 'Add Client'. Below the title are four fields: 'Import' with a 'Select file' button, 'Client ID *' with a text input containing 'myapp', 'Client Protocol' with a dropdown menu showing 'openid-connect', and 'Root URL' with a text input containing 'http://localhost:808/myapp'. At the bottom are 'Save' and 'Cancel' buttons.

3. Enter any name for **Client ID**.
4. Select **openid-connect** in the **Client Protocol** drop down box.
5. Enter the base URL of your application in the **Root URL** field.
6. Click **Save**.

This action creates the client and bring you to the **Settings** tab.

Client settings

The screenshot shows the 'Settings' tab for an OAuth client named 'myapp'. The left sidebar contains navigation options: Master, Configure (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication), and Manage (Groups, Users, Sessions, Events, Import, Export). The main content area has tabs for Settings, Roles, Client Scopes, Mappers, Scope, Revocation, Sessions, Offline Access, and Installation. The 'Settings' tab is active, displaying various configuration fields:

- Client ID: myapp
- Name: (empty)
- Description: (empty)
- Enabled: ON
- Consent Required: OFF
- Login Theme: (dropdown)
- Client Protocol: openid-connect
- Access Type: public
- Standard Flow Enabled: ON
- Implicit Flow Enabled: OFF
- Direct Access Grants Enabled: ON
- Root URL: http://localhost:808/myapp
- * Valid Redirect URIs: http://localhost:808/myapp/* (with add and remove buttons)
- Base URL: (empty)
- Admin URL: http://localhost:808/myapp
- Web Origins: http://localhost:808 (with add and remove buttons)

Additional resources

- For more information about the OIDC protocol, see [OpenID Connect](#).

12.1.2. Basic settings

When you create an OIDC client, you see the following fields on the **Settings** tab.

Client ID

The alpha-numeric ID string that is used in OIDC requests and in the Red Hat Single Sign-On database to identify the client.

Name

The name for the client in Red Hat Single Sign-On UI screen. To localize the name, set up a replacement string value. For example, a string value such as `${myapp}`. See the [Server Developer Guide](#) for more information.

Description

The description of the client. This setting can also be localized.

Enabled

When turned off, the client cannot request authentication.

Consent Required

When turned on, users see a consent page that they can use to grant access to that application. It will also display metadata so the user knows the exact information that the client can access.

Access Type

The type of OIDC client.

Confidential

For server-side clients that perform browser logins and require client secrets when making an Access Token Request. This setting should be used for server-side applications.

Public

For client-side clients that perform browser logins. As it is not possible to ensure that secrets can be kept safe with client-side clients, it is important to restrict access by configuring correct redirect URIs.

Bearer-only

The application allows only bearer token requests. When turned on, this application cannot participate in browser logins.

Standard Flow Enabled

When enabled, clients can use the OIDC [Authorization Code Flow](#).

Implicit Flow Enabled

When enabled, clients can use the OIDC [Implicit Flow](#).

Direct Access Grants Enabled

When enabled, clients can use the OIDC [Direct Access Grants](#).

OAuth 2.0 Device Authorization Grant Enabled

If this is on, clients are allowed to use the OIDC [Device Authorization Grant](#).

OpenID Connect Client Initiated Backchannel Authentication Grant Enabled

If this is on, clients are allowed to use the OIDC [Client Initiated Backchannel Authentication Grant](#).

Root URL

If Red Hat Single Sign-On uses any configured relative URLs, this value is prepended to them.

Valid Redirect URIs

Required field. Enter a URL pattern and click + to add and - to remove existing URLs and click **Save**. Exact (case sensitive) string matching is used to compare valid redirect URIs.

You can use wildcards at the end of the URL pattern. For example **http://host.com/path/***. To avoid security issues, if the passed redirect URI contains the **userinfo** part or its **path** manages access to parent directory (**/./**) no wildcard comparison is performed but the standard and secure exact string matching.

The full wildcard * valid redirect URI can also be configured to allow any **http** or **https** redirect URI. Please do not use it in production environments.

Exclusive redirect URI patterns are typically more secure. See [Unspecific Redirect URIs](#) for more information.

Base URL

This URL is used when Red Hat Single Sign-On needs to link to the client.

Admin URL

Callback endpoint for a client. The server uses this URL to make callbacks like pushing revocation policies, performing backchannel logout, and other administrative operations. For Red Hat Single Sign-On servlet adapters, this URL can be the root URL of the servlet application. For more information, see [Securing Applications and Services Guide](#).

Logo URL

URL that references a logo for the Client application.

Policy URL

URL that the Relying Party Client provides to the End-User to read about how the profile data will be used.

Terms of Service URL

URL that the Relying Party Client provides to the End-User to read about the Relying Party's terms of service.

Web Origins

Enter a URL pattern and click + to add and - to remove existing URLs. Click **Save**.

This option handles [Cross-Origin Resource Sharing \(CORS\)](#). If browser JavaScript attempts an AJAX HTTP request to a server whose domain is different from the one that the JavaScript code came from, the request must use CORS. The server must handle CORS requests, otherwise the browser will not display or allow the request to be processed. This protocol protects against XSS, CSRF, and other JavaScript-based attacks.

Domain URLs listed here are embedded within the access token sent to the client application. The client application uses this information to decide whether to allow a CORS request to be invoked on it. Only Red Hat Single Sign-On client adapters support this feature. See [Securing Applications and Services Guide](#) for more information.

Front Channel Logout

If **Front Channel Logout** is enabled, the application should be able to log out users through the front channel as per [OpenID Connect Front-Channel Logout](#) specification. If enabled, you should also provide the **Front-Channel Logout URL**.

Front-Channel Logout URL

URL that will be used by Red Hat Single Sign-On to send logout requests to clients through the front-channel.

Backchannel Logout URL

URL that will cause the client to log itself out when a logout request is sent to this realm (via `end_session_endpoint`). If omitted, no logout requests are sent to the client.

12.1.3. Advanced settings

When you click *Advanced Settings*, additional fields are displayed.

OAuth 2.0 Mutual TLS Certificate Bound Access Tokens Enabled



NOTE

To enable mutual TLS in Red Hat Single Sign-On, see [Enable mutual SSL in WildFly](#).

Mutual TLS binds an access token and a refresh token together with a client certificate, which is exchanged during a TLS handshake. This binding prevents an attacker from using stolen tokens.

This type of token is a holder-of-key token. Unlike bearer tokens, the recipient of a holder-of-key token can verify if the sender of the token is legitimate.

If this setting is on, the workflow is:

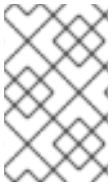
1. A token request is sent to the token endpoint in an authorization code flow or hybrid flow.
2. Red Hat Single Sign-On requests a client certificate.
3. Red Hat Single Sign-On receives the client certificate.
4. Red Hat Single Sign-On successfully verifies the client certificate.

If verification fails, Red Hat Single Sign-On rejects the token.

In the following cases, Red Hat Single Sign-On will verify the client sending the access token or the refresh token:

- A token refresh request is sent to the token endpoint with a holder-of-key refresh token.
- A UserInfo request is sent to UserInfo endpoint with a holder-of-key access token.
- A logout request is sent to Logout endpoint with a holder-of-key refresh token.

See [Mutual TLS Client Certificate Bound Access Tokens](#) in the OAuth 2.0 Mutual TLS Client Authentication and Certificate Bound Access Tokens for more details.



NOTE

Currently, Red Hat Single Sign-On client adapters do not support holder-of-key token verification. Red Hat Single Sign-On adapters treat access and refresh tokens as bearer tokens.

Advanced Settings for OIDC

The Advanced Settings for OpenID Connect allows you to configure overrides at the client level for [session and token timeouts](#).

Advanced Settings ?

Access Token Lifespan ?	<input type="text"/>	Minutes ▼
Client Session Idle ?	<input type="text"/>	Minutes ▼
Client Session Max ?	<input type="text"/>	Minutes ▼
Client Offline Session Idle ?	<input type="text"/>	Minutes ▼
Client Offline Session Max ?	<input type="text"/>	Minutes ▼

Configuration	Description
Access Token Lifespan	The value overrides the realm option with same name.
Client Session Idle	The value overrides the realm option with the same name. The value should be shorter than the global SSO Session Idle .
Client Session Max	The value overrides the realm option with the same name. The value should be shorter than the global SSO Session Max .
Client Offline Session Idle	This setting allows you to configure a shorter offline session idle timeout for the client. The timeout is amount of time the session remains idle before Red Hat Single Sign-On revokes its offline token. If not set, realm Offline Session Idle is used.
Client Offline Session Max	This setting allows you to configure a shorter offline session max lifespan for the client. The lifespan is the maximum time before Red Hat Single Sign-On revokes the corresponding offline token. This option needs Offline Session Max Limited enabled globally in the realm, and defaults to Offline Session Max .

Proof Key for Code Exchange Code Challenge Method

If an attacker steals an authorization code of a legitimate client, Proof Key for Code Exchange (PKCE) prevents the attacker from receiving the tokens that apply to the code.

An administrator can select one of these options:

(blank)

Red Hat Single Sign-On does not apply PKCE unless the client sends appropriate PKCE parameters to Red Hat Single Sign-Ons authorization endpoint.

S256

Red Hat Single Sign-On applies to the client PKCE whose code challenge method is S256.

plain

Red Hat Single Sign-On applies to the client PKCE whose code challenge method is plain.

See [RFC 7636 Proof Key for Code Exchange by OAuth Public Clients](#) for more details.

Signed and Encrypted ID Token Support

Red Hat Single Sign-On can encrypt ID tokens according to the [Json Web Encryption \(JWE\)](#) specification. The administrator determines if ID tokens are encrypted for each client.

The key used for encrypting the ID token is the Content Encryption Key (CEK). Red Hat Single Sign-On and a client must negotiate which CEK is used and how it is delivered. The method used to determine the CEK is the Key Management Mode. The Key Management Mode that Red Hat Single Sign-On supports is Key Encryption.

In Key Encryption:

1. The client generates an asymmetric cryptographic key pair.
2. The public key is used to encrypt the CEK.
3. Red Hat Single Sign-On generates a CEK per ID token
4. Red Hat Single Sign-On encrypts the ID token using this generated CEK
5. Red Hat Single Sign-On encrypts the CEK using the client's public key.
6. The client decrypts this encrypted CEK using their private key
7. The client decrypts the ID token using the decrypted CEK.

No party, other than the client, can decrypt the ID token.

The client must pass its public key for encrypting CEK to Red Hat Single Sign-On. Red Hat Single Sign-On supports downloading public keys from a URL provided by the client. The client must provide public keys according to the [Json Web Keys \(JWK\)](#) specification.

The procedure is:

1. Open the client's **Keys** tab.
2. Toggle **JWKS URL** to ON.
3. Input the client's public key URL in the **JWKS URL** textbox.

Key Encryption's algorithms are defined in the [Json Web Algorithm \(JWA\)](#) specification. Red Hat Single Sign-On supports:

- RSAES-PKCS1-v1_5(RSA1_5)

- RSAES OAEP using default parameters (RSA-OAEP)
- RSAES OAEP 256 using SHA-256 and MFG1 (RSA-OAEP-256)

The procedure to select the algorithm is:

1. Open the client's **Settings** tab.
2. Open **Fine Grain OpenID Connect Configuration**
3. Select the algorithm from **ID Token Encryption Content Encryption Algorithm** pull-down menu.

ACR to Level of Authentication (LoA) Mapping

In the advanced settings of a client, you can define which **Authentication Context Class Reference (ACR)** value is mapped to which **Level of Authentication (LoA)**. This mapping can be specified also at the realm as mentioned in the [ACR to LoA Mapping](#). A best practice is to configure this mapping at the realm level, which allows to share the same settings across multiple clients.

The **Default ACR Values** can be used to specify the default values when the login request is sent from this client to Red Hat Single Sign-On without **acr_values** parameter and without a **claims** parameter that has an **acr** claim attached. See [official OIDC dynamic client registration specification](#).



WARNING

Note that default ACR values are used as the default level, however it cannot be reliably used to enforce login with the particular level. For example, assume that you configure the **Default ACR Values** to level 2. Then by default, users will be required to authenticate with level 2. However when the user explicitly attaches the parameter into login request such as **acr_values=1**, then the level 1 will be used. As a result, if the client really requires level 2, the client is encouraged to check the presence of the **acr** claim inside ID Token and doublecheck that it contains the requested level 2.

ACR to LoA Mapping ?	ACR	LOA	+
Default ACR Values ?	silver		-
			+

> Authentication Flow Overrides ?

Save Cancel

For further details see [Step-up Authentication](#) and [the official OIDC specification](#).

12.1.4. Confidential client credentials

If the [access type](#) of the client is set to **confidential**, the credentials of the client must be configured under the **Credentials** tab.

Credentials tab

The screenshot shows the 'Credentials' tab for a client named 'Myapp'. The left sidebar contains navigation options: Master, Configure (Realm Settings, Clients, Client Scopes, Roles, Identity, Providers), and Manage (User Federation, Authentication, Groups). The main content area shows the 'Myapp' client configuration. The 'Client Authenticator' is set to 'Client Id and Secret'. The 'Secret' field contains the value '37e3055c-2741-484a-8f7e-7849l' and has a 'Regenerate Secret' button. Below this, the 'Registration access token' field is empty and has a 'Regenerate registration access token' button. The top navigation includes 'Settings', 'Credentials', 'Roles', 'Client Scopes', 'Mappers', 'Scope', 'Revocation', and 'Sessions'. Sub-tabs include 'Offline Access', 'Clustering', and 'Installation'.

The **Client Authenticator** drop-down list specifies the type of credential to use for your client.

Client ID and Secret

This choice is the default setting. The secret is automatically generated for you and the clicking **Regenerate Secret** recreates the secret if necessary.

Signed JWT

The screenshot shows the 'Credentials' tab for a client named 'Myapp' with the 'Client Authenticator' set to 'Signed JWT'. The 'Use JWKS URL' toggle is turned 'OFF'. Below this, it states 'No client certificate configured' and provides buttons for 'Generate new keys and certificate', 'Import Certificate', 'Save', and 'Cancel'. The 'Registration access token' field is empty and has a 'Regenerate registration access token' button. The interface is otherwise identical to the previous screenshot, showing the same sidebar and top navigation.

Signed JWT is "Signed Json Web Token".

When choosing this credential type you will have to also generate a private key and certificate for the client in the tab **Keys**. The private key will be used to sign the JWT, while the certificate is used by the server to verify the signature.

Keys tab

The screenshot shows the 'Keys' configuration page for a client named 'myapp'. The left sidebar contains navigation options: Master, Configure, Realm Settings, Clients, Client Scopes, Roles, Identity, and Providers. The main content area has a breadcrumb 'Clients > myapp' and a title 'Myapp'. Below the title are tabs for Settings, Credentials, Keys (selected), Roles, Client Scopes, Mappers, Scope, Revocation, and Sessions. Under the 'Keys' tab, there are sub-tabs for Offline Access, Clustering, and Installation. A toggle for 'Use JWKS URL' is set to 'OFF'. Below this, it states 'No client certificate configured'. At the bottom, there are four buttons: 'Generate new keys and certificate', 'Import Certificate', 'Save', and 'Cancel'.

Click on the **Generate new keys and certificate** button to start this process.

Generate keys

The screenshot shows the 'Generate Private Key' dialog. The breadcrumb is 'Clients > myapp > Credentials > Generate Client Private Key'. The title is 'Generate Private Key'. The dialog contains four input fields: 'Archive Format' (a dropdown menu with 'JKS' selected), 'Key Alias' (a text input field with 'myapp'), 'Key Password' (a password input field with a visibility toggle), and 'Store Password' (a password input field with a visibility toggle). At the bottom, there are two buttons: 'Generate and Download' (highlighted in blue) and 'Cancel'.

1. Select the archive format you want to use.
2. Enter a **key password**.
3. Enter a **store password**.
4. Click **Generate and Download**.

When you generate the keys, Red Hat Single Sign-On will store the certificate and you download the private key and certificate for your client.

You can also generate keys using an external tool and then import the client's certificate by clicking **Import Certificate**.

Import certificate

1. Select the archive format of the certificate.
2. Enter the store password.
3. Select the certificate file by clicking **Import File**.
4. Click **Import**.

Importing a certificate is unnecessary if you click **Use JWKS URL**. In this case, you can provide the URL where the public key is published in **JWK** format. With this option, if the key is ever changed, Red Hat Single Sign-On reimports the key.

If you are using a client secured by Red Hat Single Sign-On adapter, you can configure the JWKS URL in this format, assuming that <https://myhost.com/myapp> is the root URL of your client application:

```
https://myhost.com/myapp/k_jwks
```

See [Server Developer Guide](#) for more details.



WARNING

Red Hat Single Sign-On caches public keys of OIDC clients. If the private key of your client is compromised, update your keys and clear the key cache. See [Clearing the cache](#) section for more details.

Signed JWT with Client Secret

If you select this option, you can use a JWT signed by client secret instead of the private key.

The client secret will be used to sign the JWT by the client.

X509 Certificate

Red Hat Single Sign-On will validate if the client uses proper X509 certificate during the TLS Handshake.

**NOTE**

This option requires mutual TLS in Red Hat Single Sign-On. See [Enable mutual SSL in WildFly](#).

X509 certificate

The screenshot shows the administration console for a client named 'Myapp'. The 'Credentials' tab is active, showing the configuration for an 'X509 Certificate' client authenticator. The 'Subject DN' field is set to ':N=loalhost, OU=Keycloak, O=JBo'. There are 'Save' and 'Cancel' buttons. Below the main configuration, there is a 'Registration access token' field and a 'Regenerate registration access token' button.

The validator also checks the Subject DN field of the certificate with a configured regexp validation expression. For some use cases, it is sufficient to accept all certificates. In that case, you can use **(.*?)(?:\$)** expression.

Two ways exist for Red Hat Single Sign-On to obtain the Client ID from the request:

- The **client_id** parameter in the query (described in Section 2.2 of the [OAuth 2.0 Specification](#)).
- Supply **client_id** as a form parameter.

12.1.5. Client Secret Rotation**NOTE**

Client Secret Rotation is **Technology Preview** and is not fully supported. This feature is disabled by default.

To enable start the server with **-Dkeycloak.profile=preview** or **-Dkeycloak.profile.feature.client_secret_rotation=enabled**. For more details see [Profiles](#).

For a client with [Confidential Access Type](#) Red Hat Single Sign-On supports the functionality of rotating client secrets through [Client Policies](#).

The client secrets rotation policy provides greater security in order to alleviate problems such as secret leakage. Once enabled, Red Hat Single Sign-On supports up to two concurrently active secrets for each client. The policy manages rotations according to the following settings:

- **Secret expiration:** [seconds] - When the secret is rotated, this is the expiration of time of the new secret. The amount, *in seconds*, added to the secret creation date. Calculated at policy execution time.

- **Rotated secret expiration:**[seconds] - When the secret is rotated, this value is the remaining expiration time for the old secret. This value should be always smaller than Secret expiration. When the value is 0, the old secret will be immediately removed during client rotation. The amount, *in seconds*, added to the secret rotation date. Calculated at policy execution time.
- **Remaining expiration time for rotation during update**[seconds] - Time period when an update to a dynamic client should perform client secret rotation. Calculated at policy execution time.

When a client secret rotation occurs, a new main secret is generated and the old client main secret becomes the secondary secret with a new expiration date.

12.1.5.1. Rules for client secret rotation

Rotations do not occur automatically or through a background process. In order to perform the rotation, an update action is required on the client, either through the Red Hat Single Sign-On Admin Console through the function of **Regenerate Secret**, in the client's credentials tab or Admin REST API. When invoking a client update action, secret rotation occurs according to the rules:

- When the value of **Secret expiration** is less than the current date.
- During dynamic client registration client-update request, the client secret will be automatically rotated if the value of **Remaining expiration time for rotation during update** match the period between the current date and the **Secret expiration**.

Additionally it is possible through Admin REST API to force a client secret rotation at any time.



NOTE

During the creation of new clients, if the client secret rotation policy is active, the behavior will be applied automatically.



WARNING

To apply the secret rotation behavior to an existing client, update that client after you define the policy so that the behavior is applied.

12.1.6. Creating an OIDC Client Secret Rotation Policy

The following is an example of defining a secret rotation policy:

Procedure

1. Click **Realm Settings** in the left menu.
2. Click **Client Policies** tab.
3. On Profiles Page, Click **Create**

Create a profile

4. Enter any name for **Name**.
5. Enter a description that helps you identify the purpose of the profile for **Description**.
6. Click **Save**.
This action creates the profile and enables you to configure executors.
7. Click **Create** to configure an executor for this profile.

Create a profile executors

8. Select *secret-rotation* for **Executor Type**.
9. Enter the maximum duration time of each secret, in seconds, for **Secret Expiration**.
10. Enter the maximum duration time of each rotated secret, in seconds, for **Rotated Secret Expiration**.



WARNING

Remember that the **Rotated Secret Expiration** value must always be less than **Secret Expiration**.

11. Enter the amount of time, in seconds, after which any update action will update the client for **Remain Expiration Time**.
12. Click **Save**.



NOTE

In the example above:

- Each secret is valid for one week.
- The rotated secret expires after two days.
- The window for updating dynamic clients starts one day before the secret expires.

13. Return to the **Client Policies** tab.

14. Click **Policies**.

15. Click **Create**.

Create the Client Secret Rotation Policy

16. Enter any name for **Name**.

17. Enter a description that helps you identify the purpose of the policy for **Description**.

18. Click **Save**.

This action creates the policy and enables you to associate policies with profiles. It also allows you to configure the conditions for policy execution.

19. Under Conditions, click **Create**.

Create the Client Secret Rotation Policy Condition

20. To apply the behavior to all confidential clients select *client-access-type* in the **Condition Type** field



NOTE

To apply to a specific group of clients, another approach would be to select the *client-roles* type in the **Condition Type** field. In this way, you could create specific roles and assign a custom rotation configuration to each role.

21. Add *confidential* to the field **Client Access Type**.
22. Click **Save**.
23. Back in the policy setting, under *Client Profiles*, in the **Add client profile** selection menu, select the profile **Weekly Client Secret Rotation Profile** created earlier.

Client Secret Rotation Policy

Client Policies > Weekly Client Secret Rotation Policy

Weekly Client Secret Rotation Policy

Name *

Description

Enabled ON

Conditions ?

Type	Actions
client-access-type	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Client Profiles ?

Name	Actions
Weekly Client Secret Rotation Profile	<input type="button" value="Delete"/>



NOTE

To apply the secret rotation behavior to an existing client, follow the following steps:

Using the Admin Console

1. Go to some client.
2. Go to tab **Credentials**.
3. Click **Re-generate secret**.

Using client REST services it can be executed in two ways:

- Through an update operation on a client
- Through the regenerate client secret endpoint

12.1.7. Using a service account

Each OIDC client has a built-in *service account*. Use this *service account* to obtain an access token.

Procedure

1. Click **Clients** in the menu.
2. Select your client.
3. Click the **Settings** tab.
4. Set the [Access Type](#) of your client to **confidential**.
5. Toggle **Service Accounts Enabled** to **ON**.
6. Click **Save**.
7. Configure your [client credentials](#).
8. Click the **Scope** tab.
9. Verify that you have roles or toggle **Full Scope Allowed** to **ON**.
10. Click the **Service Account Roles** tab
11. Configure the roles available to this service account for your client.

Roles from access tokens are the intersection of:

- Role scope mappings of a client combined with the role scope mappings inherited from linked client scopes.
- Service account roles.

The REST URL to invoke is `/auth/realms/{realm-name}/protocol/openid-connect/token`. This URL must be invoked as a POST request and requires that you post the client credentials with the request.

By default, client credentials are represented by the `clientId` and `clientSecret` of the client in the **Authorization: Basic** header but you can also authenticate the client with a signed JWT assertion or any other custom mechanism for client authentication.

You also need to set the `grant_type` parameter to "client_credentials" as per the OAuth2 specification.

For example, the POST invocation to retrieve a service account can look like this:

```
POST /auth/realms/demo/protocol/openid-connect/token
Authorization: Basic cHJvZHVjdC1zYS1jbGllbnQ6cGFzc3dvcmQ=
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
```

The response would be similar to this [Access Token Response](#) from the OAuth 2.0 specification.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "access_token":"2YotnFZFEjr1zCsicMWpAA",
```

```
"token_type":"bearer",  
"expires_in":60  
}
```

Only the access token is returned by default. No refresh token is returned and no user session is created on the Red Hat Single Sign-On side upon successful authentication by default. Due to the lack of refresh token, re-authentication is required when the access token expires. However, this situation does not mean any additional overhead for the Red Hat Single Sign-On server because sessions are not created by default.

In this situation, logout is unnecessary. However, issued access tokens can be revoked by sending requests to the OAuth2 Revocation Endpoint as described in the [OpenID Connect Endpoints](#) section.

Additional resources

For more details, see [Client Credentials Grant](#).

12.1.8. Audience support

Typically, the environment where Red Hat Single Sign-On is deployed consists of a set of *confidential* or *public* client applications that use Red Hat Single Sign-On for authentication.

Services (*Resource Servers* in the [OAuth 2 specification](#)) are also available that serve requests from client applications and provide resources to these applications. These services require an *Access token* (Bearer token) to be sent to them to authenticate a request. This token is obtained by the frontend application upon login to Red Hat Single Sign-On.

In the environment where trust among services is low, you may encounter this scenario:

1. A frontend client application requires authentication against Red Hat Single Sign-On.
2. Red Hat Single Sign-On authenticates a user.
3. Red Hat Single Sign-On issues a token to the application.
4. The application uses the token to invoke an untrusted service.
5. The untrusted service returns the response to the application. However, it keeps the applications token.
6. The untrusted service then invokes a trusted service using the applications token. This results in broken security as the untrusted service misuses the token to access other services on behalf of the client application.

This scenario is unlikely in environments with a high level of trust between services but not in environments where trust is low. In some environments, this workflow may be correct as the untrusted service may have to retrieve data from a trusted service to return data to the original client application.

An unlimited audience is useful when a high level of trust exists between services. Otherwise, the audience should be limited. You can limit the audience and, at the same time, allow untrusted services to retrieve data from trusted services. In this case, ensure that the untrusted service and the trusted service are added as audiences to the token.

To prevent any misuse of the access token, limit the audience on the token and configure your services to verify the audience on the token. The flow will change as follows:

1. A frontend application authenticates against Red Hat Single Sign-On.
2. Red Hat Single Sign-On authenticates a user.
3. Red Hat Single Sign-On issues a token to the application. The application knows that it will need to invoke an untrusted service so it places **scope=<untrusted service>** in the authentication request sent to Red Hat Single Sign-On (see [Client Scopes section](#) for more details about the *scope* parameter).
The token issued to the application contains a reference to the untrusted service in its audience (**"audience": ["<untrusted service>"]**) which declares that the client uses this access token to invoke the untrusted service.
4. The untrusted service invokes a trusted service with the token. Invocation is not successful because the trusted service checks the audience on the token and find that its audience is only for the untrusted service. This behavior is expected and security is not broken.

If the client wants to invoke the trusted service later, it must obtain another token by reissuing the SSO login with **scope=<trusted service>**. The returned token will then contain the trusted service as an audience:

```
"audience": [ "<trusted service>" ]
```

Use this value to invoke the **<trusted service>**.

12.1.8.1. Setup

When setting up audience checking:

- Ensure that services are configured to check audience on the access token sent to them by adding the flag **verify-token-audience** in the adapter configuration. See [Adapter configuration](#) for details.
- Ensure that access tokens issued by Red Hat Single Sign-On contain all necessary audiences. Audiences can be added using the client roles as described in the [next section](#) or hardcoded. See [Hardcoded audience](#).

12.1.8.2. Automatically add audience

An *Audience Resolve* protocol mapper is defined in the default client scope *roles*. The mapper checks for clients that have at least one client role available for the current token. The client ID of each client is then added as an audience, which is useful if your service (usually bearer-only) clients rely on client roles.

For example, for a bearer-only client and a confidential client, you can use the access token issued for the confidential client to invoke the bearer-only client REST service. The bearer-only client will be automatically added as an audience to the access token issued for the confidential client if the following are true:

- The bearer-only client has any client roles defined on itself.
- Target user has at least one of those client roles assigned.
- Confidential client has the role scope mappings for the assigned role.

**NOTE**

If you want to ensure that the audience is not added automatically, do not configure role scope mappings directly on the confidential client. Instead, you can create a dedicated client scope that contains the role scope mappings for the client roles of your dedicated client scope.

Assuming that the client scope is added as an optional client scope to the confidential client, the client roles and the audience will be added to the token if explicitly requested by the `scope=<trusted service>` parameter.

**NOTE**

The frontend client itself is not automatically added to the access token audience, therefore allowing easy differentiation between the access token and the ID token, since the access token will not contain the client for which the token is issued as an audience.

If you need the client itself as an audience, see the [hardcoded audience](#) option. However, using the same client as both frontend and REST service is not recommended.

12.1.8.3. Hardcoded audience

When your service relies on realm roles or does not rely on the roles in the token at all, it can be useful to use a hardcoded audience. A hardcoded audience is a protocol mapper, that will add the client ID of the specified service client as an audience to the token. You can use any custom value, for example a URL, if you want to use a different audience than the client ID.

You can add the protocol mapper directly to the frontend client. If the protocol mapper is added directly, the audience will be always added as well.

For more control over the protocol mapper, you can create the protocol mapper on the dedicated client scope, which will be called for example **good-service**.

Audience protocol mapper

The screenshot shows the administration console interface. On the left is a navigation sidebar with a 'Master' dropdown and sections for 'Configure' (Realm Settings, Clients, Client Scopes, Roles, Identity Providers, User Federation, Authentication) and 'Manage' (Groups, Users). The main content area shows the breadcrumb 'Client Scopes > good-client > Mappers > Audience for good-client' and the title 'Audience For Good-client'. The configuration form includes:

- Protocol**: openid-connect
- ID**: 6821529e-e9fd-42c5-8933-0833d9cc60b2
- Name**: Audience for good-client
- Mapper Type**: Audience
- Included Client Audience**: good-client (dropdown)
- Included Custom Audience**: (empty text field)
- Add to ID token**: OFF (toggle)
- Add to access token**: ON (toggle)

- From the [Installation tab](#) of the **good-service** client, you can generate the adapter configuration and you can confirm that `verify-token-audience` option will be set to **true**. This forces the adapter to verify the audience if you use this configuration.
- You need to ensure that the confidential client is able to request **good-service** as an audience in its tokens.

On the confidential client:

1. Click the *Client Scopes* tab.
 2. Assign **good-service** as an optional (or default) client scope.
See [Client Scopes Linking section](#) for more details.
- You can optionally [Evaluate Client Scopes](#) and generate an example access token. **good-service** will be added to the audience of the generated access token if **good-service** is included in the *scope* parameter, when you assigned it as an optional client scope.
 - In your confidential client application, ensure that the *scope* parameter is used. The value **good-service** must be included when you want to issue the token for accessing **good-service**. See:
 - [parameters forwarding section](#) if your application uses the servlet adapter.
 - [javascript adapter section](#) if your application uses the javascript adapter.



NOTE

Both the *Audience* and *Audience Resolve* protocol mappers add the audiences to the access token only, by default. The ID Token typically contains only a single audience, the client ID for which the token was issued, a requirement of the OpenID Connect specification. However, the access token does not necessarily have the client ID, which was the token issued for, unless the audience mappers added it.

12.2. CREATING A SAML CLIENT

Red Hat Single Sign-On supports [SAML 2.0](#) for registered applications. POST and Redirect bindings are supported. You can choose to require client signature validation. You can have the server sign and/or encrypt responses as well.

Procedure

1. Click **Clients** in the menu.
2. Click **Create** to go to the **Add Client** page.

Add client

Master ▾

Configure

- ⚙️ Realm Settings
- 📁 Clients**
- 🔗 Client Scopes
- 📄 Roles
- 🔄 Identity
- Providers
- 👤 User Federation
- 🔒 Authentication

Clients > Add Client

Add Client

Import

Client ID * ⓘ

Client Protocol ⓘ

Client SAML Endpoint ⓘ

3. Enter the **Client ID** of the client. This is often a URL and is the expected **issuer** value in SAML requests sent by the application.
4. Select **saml** in the **Client Protocol** drop down box.
5. Enter the **Client SAML Endpoint** URL. This URL is where you want the Red Hat Single Sign-On server to send SAML requests and responses. Generally, applications have one URL for processing SAML requests. Multiple URLs can be set in the **Settings** tab of the client.
6. Click **Save**. This action creates the client and brings you to the **Settings** tab.

Client settings

The screenshot shows the 'Settings' tab for a client named 'Mysamlapp'. The left sidebar contains navigation options: Master, Configure (Realm, Settings), Clients, Client Scopes, Roles, Identity, Providers (User, Federation), Authentication, and Manage (Groups, Users, Sessions, Events, Import, Export). The main content area shows the 'Settings' tab selected, with sub-tabs for SAML Keys, Roles, Client Scopes, Mappers, Scope, Sessions, and Offline Access. Below these are 'Clustering' and 'Installation' sub-sections. The settings are as follows:

Client ID	mysamlapp
Name	
Description	
Enabled	<input checked="" type="checkbox"/>
Consent Required	<input type="checkbox"/>
Login Theme	
Client Protocol	saml
Include AuthnStatement	<input checked="" type="checkbox"/>
Include OneTimeUse Condition	<input type="checkbox"/>
Sign Documents	<input checked="" type="checkbox"/>

The following list describes each setting:

Client ID

The alpha-numeric ID string that is used in OIDC requests and in the Red Hat Single Sign-On database to identify the client. This value must match the issuer value sent with AuthNRequests. Red Hat Single Sign-On pulls the issuer from the Authn SAML request and match it to a client by this value.

Name

The name for the client in a Red Hat Single Sign-On UI screen. To localize the name, set up a replacement string value. For example, a string value such as `${myapp}`. See the [Server Developer Guide](#) for more information.

Description

The description of the client. This setting can also be localized.

Enabled

When set to OFF, the client cannot request authentication.

Consent Required

When set to ON, users see a consent page that grants access to that application. The page also displays the metadata of the information that the client can access. If you have ever done a social login to Facebook, you often see a similar page. Red Hat Single Sign-On provides the same functionality.

Include AuthnStatement

SAML login responses may specify the authentication method used, such as password, as well as timestamps of the login and the session expiration. **Include AuthnStatement** is enabled by default, so that the **AuthnStatement** element will be included in login responses. Setting this to OFF prevents clients from determining the maximum session length, which can create client sessions that do not expire.

Sign Documents

When set to ON, Red Hat Single Sign-On signs the document using the realms private key.

Optimize REDIRECT signing key lookup

When set to ON, the SAML protocol messages include the Red Hat Single Sign-On native extension. This extension contains a hint with the signing key ID. The SP uses the extension for signature validation instead of attempting to validate the signature using keys. This option applies to REDIRECT bindings where the signature is transferred in query parameters and this information is not found in the signature information. This is contrary to POST binding messages where key ID is always included in document signature.

This option is used when Red Hat Single Sign-On server and adapter provide the IDP and SP. This option is only relevant when **Sign Documents** is set to ON.

Sign Assertions

The assertion is signed and embedded in the SAML XML Auth response.

Signature Algorithm

The algorithm used in signing SAML documents.

SAML Signature Key Name

Signed SAML documents sent using POST binding contain the identification of the signing key in the **KeyName** element. This action can be controlled by the **SAML Signature Key Name** option. This option controls the contents of the **KeyName**.

- **KEY_ID** The **KeyName** contains the key ID. This option is the default option.
- **CERT_SUBJECT** The **KeyName** contains the subject from the certificate corresponding to the realm key. This option is expected by Microsoft Active Directory Federation Services.
- **NONE** The **KeyName** hint is completely omitted from the SAML message.

Canonicalization Method

The canonicalization method for XML signatures.

Encrypt Assertions

Encrypts the assertions in SAML documents with the realms private key. The AES algorithm uses a key size of 128 bits.

Client Signature Required

If **Client Signature Required** is enabled, documents coming from a client are expected to be signed. Red Hat Single Sign-On will validate this signature using the client public key or cert set up in the **Keys** tab.

Force POST Binding

By default, Red Hat Single Sign-On responds using the initial SAML binding of the original request. By enabling **Force POST Binding** Red Hat Single Sign-On responds using the SAML POST binding even if the original request used the redirect binding.

Front Channel Logout

If **Front Channel Logout** is enabled, the application requires a browser redirect to perform a logout. For example, the application may require a cookie to be reset which could only be done via a redirect. If **Front Channel Logout** is disabled, Red Hat Single Sign-On invokes a background SAML request to log out of the application.

Force Name ID Format

If a request has a name ID policy, ignore it and use the value configured in the Admin Console under **Name ID Format**

Allow ECP Flow

If true, this application is allowed to use SAML ECP profile for authentication.

Name ID Format

The Name ID Format for the subject. This format is used if no name ID policy is specified in a request, or if the Force Name ID Format attribute is set to ON.

Root URL

When Red Hat Single Sign-On uses a configured relative URL, this value is prepended to the URL.

Valid Redirect URIs

Enter a URL pattern and click the + sign to add. Click the - sign to remove. Click **Save** to save these changes. Wildcards values are allowed only at the end of a URL. For example, [http://host.com/*\\$\\$](http://host.com/*$$). This field is used when the exact SAML endpoints are not registered and Red Hat Single Sign-On pulls the Assertion Consumer URL from a request.

Base URL

If Red Hat Single Sign-On needs to link to a client, this URL is used.

Logo URL

URL that references a logo for the Client application.

Policy URL

URL that the Relying Party Client provides to the End-User to read about how the profile data will be used.

Terms of Service URL

URL that the Relying Party Client provides to the End-User to read about the Relying Party's terms of service.

Master SAML Processing URL

This URL is used for all SAML requests and the response is directed to the SP. It is used as the Assertion Consumer Service URL and the Single Logout Service URL.

If login requests contain the Assertion Consumer Service URL then those login requests will take precedence. This URL must be validated by a registered Valid Redirect URI pattern.

Assertion Consumer Service POST Binding URL

POST Binding URL for the Assertion Consumer Service.

Assertion Consumer Service Redirect Binding URL

Redirect Binding URL for the Assertion Consumer Service.

Logout Service POST Binding URL

POST Binding URL for the Logout Service.

Logout Service Redirect Binding URL

Redirect Binding URL for the Logout Service.

Logout Service Artifact Binding URL

Artifact Binding URL for the Logout Service. When set together with the **Force Artifact Binding** option, *Artifact* binding is forced for both login and logout flows. *Artifact* binding is not used for logout unless this property is set.

Artifact Binding URL

URL to send the HTTP artifact messages to.

Artifact Resolution Service

URL of the client SOAP endpoint where to send the **ArtifactResolve** messages to.

12.2.1. IDP Initiated login

IDP Initiated Login is a feature that allows you to set up an endpoint on the Red Hat Single Sign-On server that will log you into a specific application/client. In the **Settings** tab for your client, you need to specify the **IDP Initiated SSO URL Name**. This is a simple string with no whitespace in it. After this you can reference your client at the following URL: **root/auth/realms/{realm}/protocol/saml/clients/{url-name}**

The IDP initiated login implementation prefers *POST* over *REDIRECT* binding (check [saml bindings](#) for more information). Therefore the final binding and SP URL are selected in the following way:

1. If the specific **Assertion Consumer Service POST Binding URL**s defined (inside **Fine Grain SAML Endpoint Configuration** section of the client settings) *POST* binding is used through that URL.
2. If the general **Master SAML Processing URL** is specified then *POST* binding is used again through this general URL.
3. As the last resort, if the **Assertion Consumer Service Redirect Binding URL**s configured (inside **Fine Grain SAML Endpoint Configuration**) *REDIRECT* binding is used with this URL.

If your client requires a special relay state, you can also configure this on the **Settings** tab in the **IDP Initiated SSO Relay State** field. Alternatively, browsers can specify the relay state in a **RelayState** query parameter, i.e. **root/auth/realms/{realm}/protocol/saml/clients/{url-name}?RelayState=thestate**.

When using [identity brokering](#), it is possible to set up an IDP Initiated Login for a client from an external IDP. The actual client is set up for IDP Initiated Login at broker IDP as described above. The external IDP has to set up the client for application IDP Initiated Login that will point to a special URL pointing to the broker and representing IDP Initiated Login endpoint for a selected client at the brokering IDP. This means that in client settings at the external IDP:

- **IDP Initiated SSO URL Name** is set to a name that will be published as IDP Initiated Login initial point,
- **Assertion Consumer Service POST Binding URL** in the **Fine Grain SAML Endpoint Configuration** section has to be set to the following URL: **broker-root/auth/realms/{broker-realm}/broker/{idp-name}/endpoint/clients/{client-id}**, where:

• *broker-root* is base broker URL

- `broker-url` is base broker URL
- `broker-realm` is name of the realm at broker where external IDP is declared
- `idp-name` is name of the external IDP at broker
- `client-id` is the value of **IDP Initiated SSO URL Name** attribute of the SAML client defined at broker. It is this client, which will be made available for IDP Initiated Login from the external IDP.

Please note that you can import basic client settings from the brokering IDP into client settings of the external IDP - just use [SP Descriptor](#) available from the settings of the identity provider in the brokering IDP, and add **clients/client-id** to the endpoint URL.

12.2.2. Using an entity descriptor to create a client

Instead of registering a SAML 2.0 client manually, you can import the client using a standard SAML Entity Descriptor XML file.

The Add Client page includes an **Import** option.

Add client

Procedure

1. Click **Select File**.
2. Load the file that contains the XML entity descriptor information.
3. Review the information to ensure everything is set up correctly.

Some SAML client adapters, such as `mod-auth-mellon`, need the XML Entity Descriptor for the IDP. You can find this descriptor by going to this URL:

```
root/auth/realms/{realm}/protocol/saml/descriptor
```

where `realm` is the realm of your client.

12.3. CLIENT LINKS

To link from one client to another, Red Hat Single Sign-On provides a redirect endpoint: **/realms/realms_name/clients/{client-id}/redirect**.

If a client accesses this endpoint using a **HTTP GET** request, Red Hat Single Sign-On returns the configured base URL for the provided Client and Realm in the form of an **HTTP 307** (Temporary Redirect) in the response's **Location** header. As a result of this, a client needs only to know the Realm name and the Client ID to link to them. This indirection avoids hard-coding client base URLs.

As an example, given the realm **master** and the client-id **account**:

```
http://host:port/auth/realms/master/clients/account/redirect
```

This URL temporarily redirects to: <http://host:port/auth/realms/master/account>

12.4. OIDC TOKEN AND SAML ASSERTION MAPPINGS

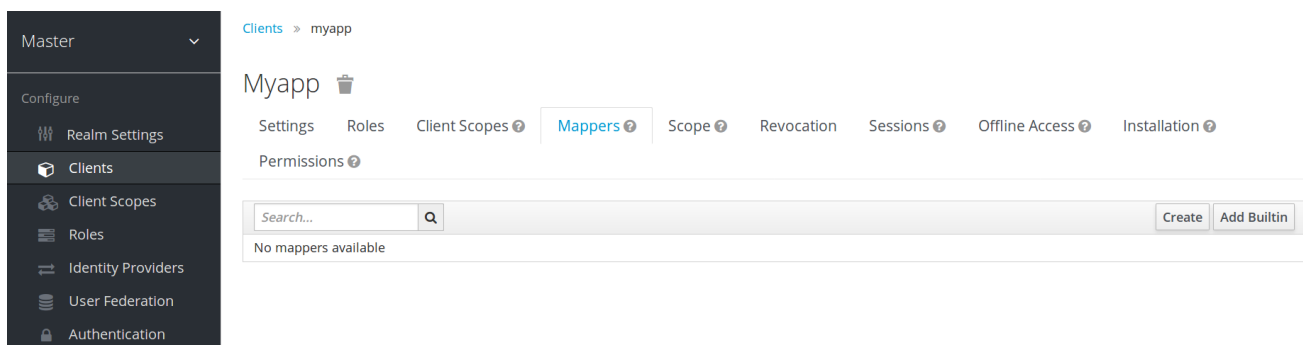
Applications receiving ID tokens, access tokens, or SAML assertions may require different roles and user metadata.

You can use Red Hat Single Sign-On to:

- Hardcode roles, claims and custom attributes.
- Pull user metadata into a token or assertion.
- Rename roles.

You perform these actions in the **Mappers** tab in the Admin Console.

Mappers tab



New clients do not have built-in mappers but they can inherit some mappers from client scopes. See the [client scopes section](#) for more details.

Protocol mappers map items (such as an email address, for example) to a specific claim in the identity and access token. The function of a mapper should be self explanatory from its name. You add pre-configured mappers by clicking **Add Builtin**.

Each mapper has a set of common settings. Additional settings are available, depending on the mapper type. Click **Edit** next to a mapper to access the configuration screen to adjust these settings.

Mapper config

Master ▾

Configure

- Realm Settings
- Clients**
- Client Scopes
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Groups
- Users
- Sessions
- Events
- Import
- Export

Clients > myapp > Mappers > Create Protocol Mappers

Create Protocol Mapper

Protocol

Name

Mapper Type

Realm Role prefix

Multivalued

Token Claim Name

Claim JSON Type

Add to ID token

Add to access token

Add to userinfo

Details on each option can be viewed by hovering over its tooltip.

You can use most OIDC mappers to control where the claim gets placed. You opt to include or exclude the claim from the *id* and access tokens by adjusting the **Add to ID token** and **Add to access token** switches.

You can add mapper types as follows:

Procedure

1. Go to the **Mappers** tab.
2. Click **Create**.

Add mapper

Master

Configure

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

Manage

- Groups
- Users
- Sessions
- Events
- Import

Clients > myapp > Mappers > Create Protocol Mappers

Create Protocol Mapper

Protocol

Name

Consent Required OFF

Mapper Type

Token Claim Name

Full group path OFF

Add to ID token OFF

Add to access token OFF

Add to userinfo OFF

3. Select a **Mapper Type** from the list box.

12.4.1. Priority order

Mapper implementations have *priority order*. *Priority order* is not the configuration property of the mapper. It is the property of the concrete implementation of the mapper.

Mappers are sorted by the order in the list of mappers. The changes in the token or assertion are applied in that order with the lowest applying first. Therefore, the implementations that are dependent on other implementations are processed in the necessary order.

For example, to compute the roles which will be included with a token:

1. Resolve audiences based on those roles.
2. Process a JavaScript script that uses the roles and audiences already available in the token.

12.4.2. OIDC user session note mappers

User session details are defined using mappers and are automatically included when you use or enable a feature on a client. Click **Add builtin** to include session details.

Impersonated user sessions provide the following details:

- **IMPERSONATOR_ID**: The ID of an impersonating user.
- **IMPERSONATOR_USERNAME**: The username of an impersonating user.

Service account sessions provide the following details:

- **clientId**: The client ID of the service account.
- **clientAddress**: The remote host IP of the service account's authenticated device.
- **clientHost**: The remote host name of the service account's authenticated device.

12.4.3. Script mapper

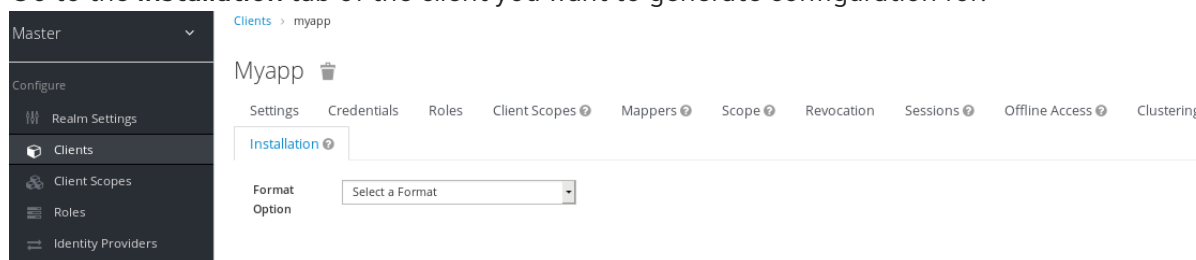
Use the **Script Mapper** to map claims to tokens by running user-defined JavaScript code. For more details about deploying scripts to the server, see [JavaScript Providers](#).

When scripts deploy, you should be able to select the deployed scripts from the list of available mappers.

12.5. GENERATING CLIENT ADAPTER CONFIG

Red Hat Single Sign-On can generate configuration files that you can use to install a client adapter in your application's deployment environment. A number of adapter types are supported for OIDC and SAML.

1. Go to the **Installation** tab of the client you want to generate configuration for.



2. Select the **Format Option** you want configuration generated for.

All Red Hat Single Sign-On client adapters for OIDC and SAML are supported. The mod-auth-mellon Apache HTTPD adapter for SAML is supported as well as standard SAML entity descriptor files.

12.6. CLIENT SCOPES

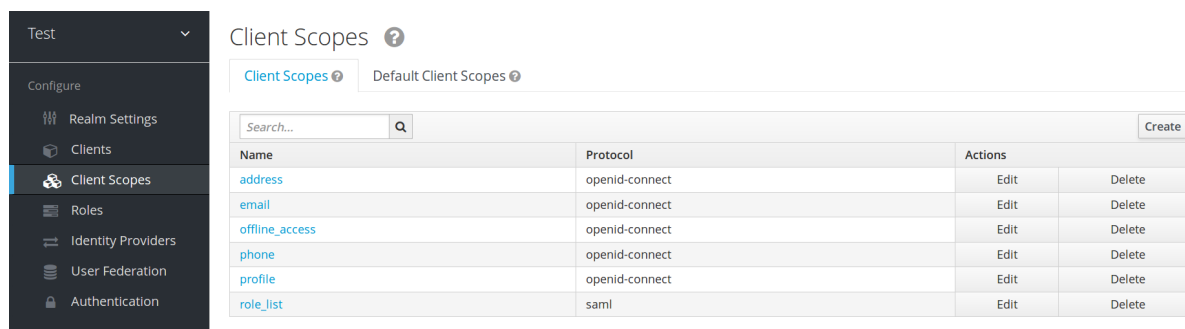
Use Red Hat Single Sign-On to define a shared client configuration in an entity called a *client scope*. A *client scope* configures [protocol mappers](#) and [role scope mappings](#) for multiple clients.

Client scopes also support the OAuth 2 **scope** parameter. Client applications use this parameter to request claims or roles in the access token, depending on the requirement of the application.

To create a client scope, follow these steps:

1. Click **Client Scopes** in the menu.

Client scopes list



2. Click **Create**.
3. Name your client scope.
4. Click **Save**.

A *client scope* has similar tabs to regular clients. You can define [protocol mappers](#) and [role scope mappings](#). These mappings can be inherited by other clients and are configured to inherit from this client scope.

12.6.1. Protocol

When you create a client scope, choose the **Protocol**. Clients linked in the same scope must have the same protocol.

Each realm has a set of pre-defined built-in client scopes in the menu.

- SAML protocol: The **role_list**. This scope contains one protocol mapper for the roles list in the SAML assertion.
- OpenID Connect protocol: Several client scopes are available:
 - **roles**
This scope is not defined in the OpenID Connect specification and is not added automatically to the **scope** claim in the access token. This scope has mappers, which are used to add the roles of the user to the access token and add audiences for clients that have at least one client role. These mappers are described in more detail in the [Audience section](#).
 - **web-origins**
This scope is also not defined in the OpenID Connect specification and not added to the **scope** claiming the access token. This scope is used to add allowed web origins to the access token **allowed-origins** claim.
 - **microprofile-jwt**
This scope handles claims defined in the [MicroProfile/JWT Auth Specification](#). This scope defines a user property mapper for the **upn** claim and a realm role mapper for the **groups** claim. These mappers can be changed so different properties can be used to create the MicroProfile/JWT specific claims.
 - **offline_access**
This scope is used in cases when clients need to obtain offline tokens. More details on offline tokens is available in the [Offline Access section](#) and in the [OpenID Connect specification](#).
 - **profile**
 - **email**
 - **address**
 - **phone**

The client scopes **profile**, **email**, **address** and **phone** are defined in the [OpenID Connect specification](#). These scopes do not have any role scope mappings defined but they do have protocol mappers defined. These mappers correspond to the claims defined in the OpenID Connect specification.

For example, when you open the **phone** client scope and open the **Mappers** tab, you will see the protocol mappers which correspond to the claims defined in the specification for the scope **phone**.

Client scope mappers

Client Scopes > phone

Phone

Settings [Mappers](#) [Scope](#)

Name	Category	Type	Actions	
phone number verified	Token mapper	User Attribute	Edit	Delete
phone number	Token mapper	User Attribute	Edit	Delete

When the **phone** client scope is linked to a client, the client automatically inherits all the protocol mappers defined in the **phone** client scope. Access tokens issued for this client contain the phone number information about the user, assuming that the user has a defined phone number.

Built-in client scopes contain the protocol mappers as defined in the specification. You are free to edit client scopes and create, update, or remove any protocol mappers or role scope mappings.

12.6.2. Consent related settings

Client scopes contain options related to the consent screen. Those options are useful if the linked client if **Consent Required** is enabled on the client.

Display On Consent Screen

If **Display On Consent Screen** is enabled, and the scope is added to a client that requires consent, the text specified in **Consent Screen Text** will be displayed on the consent screen. This text is shown when the user is authenticated and before the user is redirected from Red Hat Single Sign-On to the client. If **Display On Consent Screen** is disabled, this client scope will not be displayed on the consent screen.

Consent Screen Text

The text displayed on the consent screen when this client scope is added to a client when consent required defaults to the name of client scope. The value for this text can be customised by specifying a substitution variable with `#{var-name}` strings. The customised value is configured within the property files in your theme. See the [Server Developer Guide](#) for more information on customisation.

12.6.3. Link client scope with the client

Linking between a client scope and a client is configured in the **Client Scopes** tab of the client. Two ways of linking between client scope and client are available.

Default Client Scopes

This setting is applicable to the OpenID Connect and SAML clients. Default client scopes are applied when issuing OpenID Connect tokens or SAML assertions for a client. The client will inherit Protocol Mappers and Role Scope Mappings that are defined on the client scope. For the OpenID Connect Protocol, the Mappers and Role Scope Mappings are always applied, regardless of the value used for the scope parameter in the OpenID Connect authorization request.

Optional Client Scopes

This setting is applicable only for OpenID Connect clients. Optional client scopes are applied when issuing tokens for this client but only when requested by the **scope** parameter in the OpenID Connect authorization request.

12.6.3.1. Example

For this example, assume the client has **profile** and **email** linked as default client scopes, and **phone** and **address** linked as optional client scopes. The client uses the value of the scope parameter when sending a request to the OpenID Connect authorization endpoint.

`scope=openid phone`

The scope parameter contains the string, with the scope values divided by spaces. The value **openid** is the meta-value used for all OpenID Connect requests. The token will contain mappers and role scope mappings from the default client scopes **profile** and **email** as well as **phone**, an optional client scope requested by the scope parameter.

12.6.4. Evaluating Client Scopes

The **Mappers** tab contains the protocol mappers and the **Scope** tab contains the role scope mappings declared for this client. They do not contain the mappers and scope mappings inherited from client scopes. It is possible to see the effective protocol mappers (that is the protocol mappers defined on the client itself as well as inherited from the linked client scopes) and the effective role scope mappings used when generating a token for a client.

Procedure

1. Click the **Client Scopes** tab for the client.
2. Open the sub-tab **Evaluate**.
3. Select the optional client scopes that you want to apply.

This will also show you the value of the **scope** parameter. This parameter needs to be sent from the application to the Red Hat Single Sign-On OpenID Connect authorization endpoint.

Evaluating client scopes

The screenshot displays the 'Account' configuration page in the Admin Console, specifically the 'Client Scopes' tab. The 'Evaluate' sub-tab is active, showing the 'Scope Parameter' set to 'openid phone'. Below this, there are four panels: 'Client Scopes' (empty), 'Available Optional Client Scopes' (listing 'address' and 'offline_access'), 'Selected Optional Client Scopes' (listing 'phone'), and 'Effective Client Scopes' (listing 'email', 'phone', and 'profile'). The 'User' field is set to 'john'. A blue 'Evaluate' button is visible. At the bottom, the 'Generated Access Token' tab is selected, displaying a JSON token:

```
{
  "exp": 1615911782,
  "iat": 1615911482,
  "jti": "c13a3d42-16af-48ec-b4ae-01e762594877",
  "iss": "http://localhost:8180/auth/realms/test",
  "sub": "6f62dc96-2695-4509-aa92-868da4536638",
  "typ": "Bearer",
  "azp": "account",
```



NOTE

To send a custom value for a **scope** parameter from your application, see the [parameters forwarding section](#), for servlet adapters or the [javascript adapter section](#), for javascript adapters.

All examples are generated for the particular user and issued for the particular client, with the specified value of the **scope** parameter. The examples include all of the claims and role mappings used.

12.6.5. Client scopes permissions

When issuing tokens to a user, the client scope applies only if the user is permitted to use it.

When a client scope does not have any role scope mappings defined, each user is permitted to use this client scope. However, when a client scope has role scope mappings defined, the user must be a member of at least one of the roles. There must be an intersection between the user roles and the roles of the client scope. Composite roles are factored into evaluating this intersection.

If a user is not permitted to use the client scope, no protocol mappers or role scope mappings will be used when generating tokens. The client scope will not appear in the **scope** value in the token.

12.6.6. Realm default client scopes

Use **Realm Default Client Scopes** to define sets of client scopes that are automatically linked to newly created clients.

Procedure

1. Click the **Client Scopes** tab for the client.
2. Click **Default Client Scopes**.

From here, select the client scopes that you want to add as **Default Client Scopes** to newly created clients and **Optional Client Scopes**.

Default client scopes

The screenshot shows the 'Default Client Scopes' configuration page for a client named 'Test'. The page is divided into two main sections: 'Default Client Scopes' and 'Optional Client Scopes'. Each section has an 'Available Client Scopes' list and an 'Assigned' list. The 'Default Client Scopes' section shows 'role_list', 'profile', and 'email' in the assigned list. The 'Optional Client Scopes' section shows 'offline_access', 'address', and 'phone' in the assigned list. A sidebar on the left contains navigation options under 'Configure' and 'Manage'.

Section	Available Client Scopes	Assigned
Default Client Scopes	(Empty)	role_list profile email
Optional Client Scopes	(Empty)	offline_access address phone

When a client is created, you can unlink the default client scopes, if needed. This is similar to removing [Default Roles](#).

12.6.7. Scopes explained

Client scope

Client scopes are entities in Red Hat Single Sign-On that are configured at the realm level and can be linked to clients. Client scopes are referenced by their name when a request is sent to the Red Hat Single Sign-On authorization endpoint with a corresponding value of the **scope** parameter. See the [client scopes linking](#) section for more details.

Role scope mapping

This is available under the **Scope** tab of a client or client scope. Use **Role scope mapping** to limit the roles that can be used in the access tokens. See the [Role Scope Mappings section](#) for more details.

12.7. CLIENT POLICIES

To make it easy to secure client applications, it is beneficial to realize the following points in a unified way.

- Setting policies on what configuration a client can have
- Validation of client configurations
- Conformance to a required security standards and profiles such as Financial-grade API (FAPI)

To realize these points in a unified way, *Client Policies* concept is introduced.

12.7.1. Use-cases

Client Policies realize the following points mentioned as follows.

Setting policies on what configuration a client can have

Configuration settings on the client can be enforced by client policies during client creation/update, but also during OpenID Connect requests to Red Hat Single Sign-On server, which are related to particular client. Red Hat Single Sign-On supports similar thing also through the Client Registration Policies described in the [Securing Applications and Services Guide](#). However, Client Registration Policies can only cover OIDC Dynamic Client Registration. Client Policies cover not only what Client Registration Policies can do, but other client registration and configuration ways. The current plans are for Client Registration to be replaced by Client Policies.

Validation of client configurations

Red Hat Single Sign-On supports validation whether the client follows settings like Proof Key for Code Exchange, Request Object Signing Algorithm, Holder-of-Key Token, and so on on some endpoints like Authorization Endpoint, Token Endpoint, and so on. These can be specified by each setting item (on Admin Console, switch, pulldown menu and so on). To make the client application secure, the administrator needs to set many settings in the appropriate way, which makes it difficult for the administrator to secure the client application. Client Policies can do these validation of client configurations mentioned just above and they can also be used to auto-configure some client configuration switches to meet the advanced security requirements. In the future, individual client configuration settings may be replaced by Client Policies directly performing required validations.

Conformance to a required security standards and profiles such as FAPI

The *Global client profiles* are client profiles pre-configured in Red Hat Single Sign-On by default. They are pre-configured to be compliant with standard security profiles like [FAPI](#), which makes it easy for the administrator to secure their client application to be compliant with the particular security

profile. At this moment, Red Hat Single Sign-On has global profiles for the support of FAPI 1 specification. The administrator will just need to configure the client policies to specify which clients should be compliant with the FAPI. The administrator can configure client profiles and client policies, so that Red Hat Single Sign-On clients can be easily made compliant with various other security profiles like SPA, Native App, Open Banking and so on.

12.7.2. Protocol

The client policy concept is independent of any specific protocol. However, Red Hat Single Sign-On currently supports it only just for the [OpenID Connect \(OIDC\) protocol](#).

12.7.3. Architecture

Client Policies consists of the four building blocks: Condition, Executor, Profile and Policy.

12.7.3.1. Condition

A condition determines to which client a policy is adopted and when it is adopted. Some conditions are checked at the time of client create/update when some other conditions are checked during client requests (OIDC Authorization request, Token endpoint request and so on). The condition checks whether one specified criteria is satisfied. For example, some condition checks whether the access type of the client is confidential.

The condition can not be used solely by itself. It can be used in a [policy](#) that is described afterwards.

A condition can be configurable the same as other configurable providers. What can be configured depends on each condition's nature.

The following conditions are provided:

The way of creating/updating a client

- Dynamic Client Registration (Anonymous or Authenticated with Initial access token or Registration access token)
- Admin REST API (Admin Console and so on)

So for example when creating a client, a condition can be configured to evaluate to true when this client is created by OIDC Dynamic Client Registration without initial access token (Anonymous Dynamic Client Registration). So this condition can be used for example to ensure that all clients registered through OIDC Dynamic Client Registration are FAPI compliant.

Author of a client (Checked by presence to the particular role or group)

On OpenID Connect dynamic client registration, an author of a client is the end user who was authenticated to get an access token for generating a new client, not Service Account of the existing client that actually accesses the registration endpoint with the access token. On registration by Admin REST API, an author of a client is the end user like the administrator of the Red Hat Single Sign-On.

Client Access Type (confidential, public, bearer-only)

For example when a client sends an authorization request, a policy is adopted if this client is confidential.

Client Scope

Evaluates to true if the client has a particular client scope (either as default or as an optional scope used in current request). This can be used for example to ensure that OIDC authorization requests with scope **fapi-example-scope** need to be FAPI compliant.

Client Role

Applies for clients with the client role of the specified name

Client Domain Name, Host or IP Address

Applied for specific domain names of client. Or for the cases when the administrator registers/updates client from particular Host or IP Address.

Any Client

This condition always evaluates to true. It can be used for example to ensure that all clients in the particular realm are FAPI compliant.

12.7.3.2. Executor

An executor specifies what action is executed on a client to which a policy is adopted. The executor executes one or several specified actions. For example, some executor checks whether the value of the parameter **redirect_uri** in the authorization request matches exactly with one of the pre-registered redirect URIs on Authorization Endpoint and rejects this request if not.

The executor can not be used solely by itself. It can be used in a [profile](#) that is described afterwards.

An executor can be configurable the same as other configurable providers. What can be configured depends on the nature of each executor.

An executor acts on various events. An executor implementation can ignore certain types of events (For example, executor for checking OIDC **request** object acts just on the OIDC authorization request). Events are:

- Creating a client (including creation through dynamic client registration)
- Updating a client
- Sending an authorization request
- Sending a token request
- Sending a token refresh request
- Sending a token revocation request
- Sending a token introspection request
- Sending a userinfo request
- Sending a logout request with a refresh token

On each event, an executor can work in multiple phases. For example, on creating/updating a client, the executor can modify the client configuration by auto-configure specific client settings. After that, the executor validates this configuration in validation phase.

One of several purposes for this executor is to realize the security requirements of client conformance profiles like FAPI. To do so, the following executors are needed:

- Enforce secure [Client Authentication method](#) is used for the client

- Enforce [Holder-of-key tokens](#) are used
- Enforce [Proof Key for Code Exchange \(PKCE\)](#) is used
- Enforce secure signature algorithm for [Signed JWT client authentication \(private-key-jwt\)](#) is used
- Enforce HTTPS redirect URI and make sure that configured redirect URI does not contain wildcards
- Enforce OIDC **request** object satisfying high security level
- Enforce Response Type of OIDC Hybrid Flow including ID Token used as *detached signature* as described in the FAPI 1 specification, which means that ID Token returned from Authorization response won't contain user profile data
- Enforce more secure **state** and **nonce** parameters treatment for preventing CSRF
- Enforce more secure signature algorithm when client registration
- Enforce **binding_message** parameter is used for CIBA requests
- Enforce [Client Secret Rotation](#)

12.7.3.3. Profile

A profile consists of several executors, which can realize a security profile like FAPI. Profile can be configured by the Admin REST API (Admin Console) together with its executors. Three *global profiles* exist and they are configured in Red Hat Single Sign-On by default with pre-configured executors compliant with the FAPI Baseline, FAPI Advanced and FAPI CIBA specifications. More details exist in the FAPI section of the [Securing Applications and Services Guide](#).

12.7.3.4. Policy

A policy consists of several conditions and profiles. The policy can be adopted to clients satisfying all conditions of this policy. The policy refers several profiles and all executors of these profiles execute their task against the client that this policy is adopted to.

12.7.4. Configuration

Policies, profiles, conditions, executors can be configured by Admin REST API, which means also the Admin Console. To do so, there is a tab *Realm* → *Realm Settings* → *Client Policies*, which means the administrator can have client policies per realm.

The *Global Client Profiles* are automatically available in each realm. However there are no client policies configured by default. This means that the administrator is always required to create any client policy if they want for example the clients of his realm to be FAPI compliant. Global profiles cannot be updated, but the administrator can easily use them as a template and create their own profile if they want to do some slight changes in the global profile configurations. There is JSON Editor available in the Admin Console, which simplifies the creation of new profile based on some global profile.

12.7.5. Backward Compatibility

Client Policies can replace Client Registration Policies described in the [Securing Applications and Services Guide](#). However, Client Registration Policies also still co-exist. This means that for example during a Dynamic Client Registration request to create/update a client, both client policies and client

registration policies are applied.

The current plans are for the Client Registration Policies feature to be removed and the existing client registration policies will be migrated into new client policies automatically.

12.7.6. Client Secret Rotation Example

See an example configuration for [client secret rotation](#).

CHAPTER 13. USING A VAULT TO OBTAIN SECRETS

To obtain a secret from a vault rather than entering it directly, enter the following specially crafted string into the appropriate field:

```
**${vault.**_key_**}
```

where the **key** is the name of the secret recognized by the vault.

To prevent secrets from leaking across realms, Red Hat Single Sign-On combines the realm name with the **key** obtained from the vault expression. This method means that the **key** does not directly map to an entry in the vault but creates the final entry name according to the algorithm used to combine the **key** with the realm name.

You can obtain the secret from the vault in the following fields:

SMTP password

In the realm [SMTP settings](#)

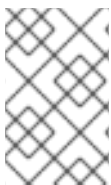
LDAP bind credential

In the [LDAP settings](#) of LDAP-based user federation.

OIDC identity provider secret

In the *Client Secret* inside identity provider [OpenID Connect Config](#)

To use a vault, register a vault provider in Red Hat Single Sign-On. You can use the providers described [here](#) or implement your provider. See the [Server Developer Guide](#) for more information.



NOTE

Red Hat Single Sign-On permits a maximum of one active vault provider per Red Hat Single Sign-On instance at a time. Configure the vault provider in each instance within the cluster consistently.

13.1. KUBERNETES / OPENSIFT FILES PLAIN-TEXT VAULT PROVIDER

Red Hat Single Sign-On supports vault implementation for [Kubernetes secrets](#). You can mount Kubernetes secrets as data volumes, and they appear as a directory with a flat-file structure. Red Hat Single Sign-On represents each secret as a file with the file's name as the secret name and the file's contents as the secret value.

You must name the files within this directory as the secret name prefixed by the realm name and an underscore. Double all underscores within the secret name or the realm name in the file name. For example, for a field within a realm named **sso_realm**, a reference to a secret with the name **secret-name** would be written as **\${vault.secret-name}**, and the file name looked up would be **sso__realm_secret-name**. Note the underscore doubled in realm name.

To use this type of secret store, you must declare the **files-plaintext** vault provider in the `standalone.xml` file and set its parameter for the directory containing the mounted volume. This example shows the **files-plaintext** provider with the directory where vault files are searched set to **standalone/configuration/vault** relative to the Red Hat Single Sign-On base directory:

```
<spi name="vault">
  <default-provider>files-plaintext</default-provider>
  <provider name="files-plaintext" enabled="true">
```

```

<properties>
  <property name="dir" value="{jboss.home.dir}/standalone/configuration/vault/" />
</properties>
</provider>
</spi>

```

Here is the equivalent configuration using CLI commands:

```

/subsystem=keycloak-server/spi=vault/:add
/subsystem=keycloak-server/spi=vault/provider=files-plaintext/:add(enabled=true,properties={dir =>
"{jboss.home.dir}/standalone/configuration/vault"})
/subsystem=keycloak-server/spi=vault:write-attribute(name=default-provider,value=files-plaintext)

```

13.2. ELYTRON CREDENTIAL STORE VAULT PROVIDER

Red Hat Single Sign-On also provides support for reading secrets stored in an Elytron credential store. The **elytron-cs-keystore** vault provider can retrieve secrets from the credential store's keystore based implementation, which is also the default implementation Elytron provides.

A keystore backs this credential store. **JCEKS** is the default format, but you can use other formats such as **PKCS12**. Users can create and manage the store contents using the **elytron** subsystem in WildFly/JBoss EAP, or the **elytron-tool.sh** script.

To use this provider, you must declare the **elytron-cs-keystore** in the **keycloak-server** subsystem and set the location and master secret of the keystore created by Elytron. An example of the minimal configuration for the provider follows:

```

<spi name="vault">
  <default-provider>elytron-cs-keystore</default-provider>
  <provider name="elytron-cs-keystore" enabled="true">
    <properties>
      <property name="location" value="{jboss.home.dir}/standalone/configuration/vault/credential-
store.jceks" />
      <property name="secret" value="secretpw1!" />
    </properties>
  </provider>
</spi>

```

If the underlying keystore has a format different from **JCEKS**, you must specify this format by using the **keyStoreType**:

```

<spi name="vault">
  <default-provider>elytron-cs-keystore</default-provider>
  <provider name="elytron-cs-keystore" enabled="true">
    <properties>
      <property name="location" value="{jboss.home.dir}/standalone/configuration/vault/credential-
store.p12" />
      <property name="secret" value="secretpw1!" />
      <property name="keyStoreType" value="PKCS12" />
    </properties>
  </provider>
</spi>

```

For the secret, the **elytron-cs-keystore** provider supports clear-text values and masked values by using the **elytron-tool.sh** script:

```
<spi name="vault">
  ...
  <property name="secret" value="MASK-3u2HNQaMogJJ8VP7J6gRII;12345678;321"/>
  ...
</spi>
```

For more information about creating and managing elytron credential stores and masking keystore secrets, see the Elytron documentation.



NOTE

Red Hat Single Sign-On implements the **elytron-cs-keystore** vault provider as a WildFly extension and is available if the Red Hat Single Sign-On server runs on WildFly/JBoss EAP only.

13.3. KEY RESOLVERS

All built-in providers support the configuration of key resolvers. A key resolver implements the algorithm or strategy for combining the realm name with the key, obtained from the **`\${vault.key}** expression, into the final entry name used to retrieve the secret from the vault. Red Hat Single Sign-On uses the **keyResolvers** property to configure the resolvers that the provider uses. The value is a comma-separated list of resolver names. An example of the configuration for the **files-plaintext** provider follows:

```
<spi name="vault">
  <default-provider>files-plaintext</default-provider>
  <provider name="files-plaintext" enabled="true">
    <properties>
      <property name="dir" value="${jboss.home.dir}/standalone/configuration/vault" />
      <property name="keyResolvers" value="REALM_UNDERSCORE_KEY, KEY_ONLY"/>
    </properties>
  </provider>
</spi>
```

The resolvers run in the same order you declare them in the configuration. For each resolver, Red Hat Single Sign-On uses the last entry name the resolver produces, which combines the realm with the vault key to search for the vault's secret. If Red Hat Single Sign-On finds a secret, it returns the secret. If not, Red Hat Single Sign-On uses the next resolver. This search continues until Red Hat Single Sign-On finds a non-empty secret or runs out of resolvers. If Red Hat Single Sign-On finds no secret, Red Hat Single Sign-On returns an empty secret.

In the previous example, Red Hat Single Sign-On uses the **REALM_UNDERSCORE_KEY** resolver first. If Red Hat Single Sign-On finds an entry in the vault that using that resolver, Red Hat Single Sign-On returns that entry. If not, Red Hat Single Sign-On searches again using the **KEY_ONLY** resolver. If Red Hat Single Sign-On finds an entry by using the **KEY_ONLY** resolver, Red Hat Single Sign-On returns that entry. If Red Hat Single Sign-On uses all resolvers, Red Hat Single Sign-On returns an empty secret.

A list of the currently available resolvers follows:

Name	Description
KEY_ONLY	Red Hat Single Sign-On ignores the realm name and uses the key from the vault expression.
REALM_UNDERSCORE_KEY	Red Hat Single Sign-On combines the realm and key by using an underscore character. Red Hat Single Sign-On escapes occurrences of underscores in the realm or key with another underscore character. For example, if the realm is called master_realm and the key is smtp_key , the combined key is master__realm_smtp__key .
REALM_FILESEPARATOR_KEY	Red Hat Single Sign-On combines the realm and key by using the platform file separator character.

If you have not configured a resolver for the built-in providers, Red Hat Single Sign-On selects the **REALM_UNDERSCORE_KEY**.

13.4. SAMPLE CONFIGURATION

The following is an example of configuring a vault and credential store. The procedure involves two parts:

- Creating the credential store and a vault, where the credential store and vault passwords are in plain text.
- Updating the credential store and vault to have the password use a mask provided by **elytron-tool.sh**.

In this example, the test target used is an LDAP instance with **BIND DN credential: secret12**. The target is mapped using user federation in the realm **ldaptest**.

13.4.1. Configuring the credential store and vault without a mask

You create the credential store and a vault where the credential store and vault passwords are in plain text.

Prerequisites

- A running LDAP instance has **BIND DN credential: secret12**.
- The alias uses the format `<realm-name>_<key-value>` when using the default key resolver. In this case, the instance is running in the realm **ldaptest** and **ldaptest_ldap_secret** is the alias that corresponds to the value **ldap_secret** in that realm.



NOTE

The resolver replaces underscore characters with double underscore characters in the realm and key names. For example, for the key **ldaptest_ldap_secret**, the final key will be **ldaptest_ldap__secret**.

Procedure

1. Create the Elytron credential store.

```
[standalone@localhost:9990 /] /subsystem=elytron/credential-store=test-store:add(create=true, location=/home/test/test-store.p12, credential-reference={clear-text=testpwd1!},implementation-properties={keyStoreType=PKCS12})
```

2. Add an alias to the credential store.

```
/subsystem=elytron/credential-store=test-store:add-alias(alias=ldaptest_ldap__secret,secret-value=secret12)
```

Notice how the resolver causes the key **ldaptest_ldap__secret** to use double underscores.

3. List the aliases from the credential store to inspect the contents of the keystore that is produced by Elytron.

```
keytool -list -keystore /home/test/test-store.p12 -storetype PKCS12 -storepass testpwd1!
Keystore type: PKCS12
Keystore provider: SUN
```

Your keystore contains 1 entries

```
ldaptest_ldap__secret/passwordcredential/clear/, Oct 12, 2020, SecretKeyEntry,
```

4. Configure the vault SPI in Red Hat Single Sign-On.

```
/subsystem=keycloak-server/spi=vault:add(default-provider=elytron-cs-keystore)
```

```
/subsystem=keycloak-server/spi=vault/provider=elytron-cs-keystore:add(enabled=true,
properties={location=>/home/test/test-store.p12, secret=>testpwd1!,
keyStoreType=>PKCS12})
```

At this point, the vault and credentials store passwords are not masked.

```
<spi name="vault">
  <default-provider>elytron-cs-keystore</default-provider>
  <provider name="elytron-cs-keystore" enabled="true">
    <properties>
      <property name="location" value="/home/test/test-store.p12"/>
      <property name="secret" value="testpwd1!"/>
      <property name="keyStoreType" value="PKCS12"/>
    </properties>
  </provider>
</spi>

<credential-stores>
  <credential-store name="test-store" location="/home/test/test-store.p12"
create="true">
    <implementation-properties>
      <property name="keyStoreType" value="PKCS12"/>
    </implementation-properties>
```



```
<credential-reference clear-text="testpwd1!"/>
</credential-store>
</credential-stores>
```

5. In the LDAP provider, replace **binDN credential** with **`\${vault.Idap_secret}`**.
6. Test your LDAP connection.

LDAP Vault

13.4.2. Masking the password in the credential store and vault

You can now update the credential store and vault to have passwords that use a mask provided by **elytron-tool.sh**.

1. Create a masked password using values for the **salt** and the **iteration** parameters:

```
$ EAP_HOME/bin/elytron-tool.sh mask --salt SALT --iteration ITERATION_COUNT --secret
PASSWORD
```

For example:

```
elytron-tool.sh mask --salt 12345678 --iteration 123 --secret testpwd1!
MASK-3BUbFEyWu0IRAu8.fCqyUk;12345678;123
```

2. Update the Elytron credential store configuration to use the masked password.

```
/subsystem=elytron/credential-store=cs-store:write-attribute(name=credential-
reference.clear-text,value="MASK-3BUbFEyWu0IRAu8.fCqyUk;12345678;123")
```

3. Update the Red Hat Single Sign-On vault configuration to use the masked password.

```
/subsystem=keycloak-server/spi=vault/provider=elytron-cs-keystore:remove()
/subsystem=keycloak-server/spi=vault/provider=elytron-cs-keystore:add(enabled=true,
properties={location=>/home/test/test-store.p12, secret=>"MASK-
3BUbFEyWu0IRAu8.fCqyUk;12345678;123", keyStoreType=>PKCS12})
```

The vault and credential store are now masked:

```
<spi name="vault">
  <default-provider>elytron-cs-keystore</default-provider>
  <provider name="elytron-cs-keystore" enabled="true">
    <properties>
      <property name="location" value="/home/test/test-store.p12"/>
      <property name="secret" value="MASK-
3BUbFEyWu0IRAu8.fCqyUk;12345678;123"/>
      <property name="keyStoreType" value="PKCS12"/>
    </properties>
  </provider>
</spi>
....
```

```
.....
<credential-stores>
  <credential-store name="test-store" location="/home/test/test-store.p12"
create="true">
    <implementation-properties>
      <property name="keyStoreType" value="PKCS12"/>
    </implementation-properties>
    <credential-reference clear-text="MASK-
3BUbFEyWu0IRAu8.fCqyUk;12345678;123"/>
  </credential-store>
</credential-stores>
```

4. You can now test the connection to the LDAP using `${vault.ldap_secret}`.

Additional resources

For more information about the Elytron tool, see [Using Credential Stores with Elytron Client](#).

CHAPTER 14. CONFIGURING AUDITING TO TRACK EVENTS

Red Hat Single Sign-On includes a suite of auditing capabilities. You can record every login and administrator action and review those actions in the Admin Console. Red Hat Single Sign-On also includes a Listener SPI that listens for events and can trigger actions. Examples of built-in listeners include log files and sending emails if an event occurs.

14.1. LOGIN EVENTS

You can record and view every event that affects users. Red Hat Single Sign-On triggers login events for actions such as successful user login, a user entering an incorrect password, or a user account updating. By default, Red Hat Single Sign-On does not store or display events in the Admin Console. Only the error events are logged to the Admin Console and the server's log file.

To start saving events, enable storage.

Procedure

1. Click **Events** in the menu.
2. Click the **Config** tab.

Event Configuration

The screenshot displays the 'Events Config' page in the Admin Console. On the left, a dark sidebar menu is open, showing 'Master' at the top and 'Configure' below it. Under 'Configure', several options are listed: 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', 'Authentication', 'Groups', and 'Users'. The 'Events Config' page has three tabs: 'Login Events', 'Admin Events', and 'Config', with 'Config' selected. Below the tabs, there is a section for 'Event Listeners' with a text input field containing 'jboss-logging'. Further down, there are two sections: 'Login Events Settings' and 'Admin Events Settings'. Each section has a 'Save Events' toggle switch, both of which are currently set to 'OFF'. At the bottom of the 'Admin Events Settings' section, there are 'Clear changes' and 'Save' buttons.

3. Toggle **Save Events** to **ON**.

Save Events

The screenshot shows the 'Events Config' page in Red Hat Single Sign-On 7.6. The left sidebar contains navigation options under 'Master', 'Configure', and 'Manage'. The main area is titled 'Events Config' and has tabs for 'Login Events', 'Admin Events', and 'Config'. The 'Config' tab is selected. In the 'Event Listeners' section, 'jboss-logging' is entered. The 'Login Events Settings' section has 'Save Events' turned 'ON'. A large list of event types is shown under 'Saved Types', each with a checkbox. Below this list is a 'Clear events' button. The 'Expiration' field is set to 'Hours'. At the bottom, the 'Admin Events Settings' section has 'Save Events' turned 'OFF', a 'Clear changes' button, and a 'Save' button.

4. Specify the events to store in the **Saved Types** field.

You can click the **Clear events** button to delete all events.

Specify the length of time to store events in the **Expiration** field. When you enable login event storage and enable your settings, click the **Save** button.

Click the **Login Events** tab to view the events.

Login Events

The screenshot shows the 'Events' page with a sidebar on the left containing 'Configure' and 'Manage' sections. The main content area has tabs for 'Login Events', 'Admin Events', and 'Config'. A table displays three events:

Time	Event Type	Details								
1/17/17 4:36:47 PM	CODE_TO_TOKEN	<table border="1"> <tr><td>Client</td><td>security-admin-console</td></tr> <tr><td>User</td><td>780d038c-b0a4-4268-8492-36ba96573a9e</td></tr> <tr><td>IP Address</td><td>127.0.0.1</td></tr> <tr><td>Details</td><td>+</td></tr> </table>	Client	security-admin-console	User	780d038c-b0a4-4268-8492-36ba96573a9e	IP Address	127.0.0.1	Details	+
Client	security-admin-console									
User	780d038c-b0a4-4268-8492-36ba96573a9e									
IP Address	127.0.0.1									
Details	+									
1/17/17 4:36:46 PM	LOGIN	<table border="1"> <tr><td>Client</td><td>security-admin-console</td></tr> <tr><td>User</td><td>780d038c-b0a4-4268-8492-36ba96573a9e</td></tr> <tr><td>IP Address</td><td>127.0.0.1</td></tr> <tr><td>Details</td><td>+</td></tr> </table>	Client	security-admin-console	User	780d038c-b0a4-4268-8492-36ba96573a9e	IP Address	127.0.0.1	Details	+
Client	security-admin-console									
User	780d038c-b0a4-4268-8492-36ba96573a9e									
IP Address	127.0.0.1									
Details	+									
1/17/17 4:36:42 PM	LOGOUT	<table border="1"> <tr><td>Client</td><td></td></tr> <tr><td>User</td><td>780d038c-b0a4-4268-8492-36ba96573a9e</td></tr> <tr><td>IP Address</td><td>127.0.0.1</td></tr> <tr><td>Details</td><td>+</td></tr> </table>	Client		User	780d038c-b0a4-4268-8492-36ba96573a9e	IP Address	127.0.0.1	Details	+
Client										
User	780d038c-b0a4-4268-8492-36ba96573a9e									
IP Address	127.0.0.1									
Details	+									

You can filter events using the **Filter** button.

Login Events Filter

The screenshot shows the 'Events' page with the 'Login Events' filter applied. The filter interface includes fields for 'Event Type' (set to 'LOGIN'), 'Client', 'User', 'Date (From)', and 'Date (To)'. The 'Update' button is visible. Below the filter, a table shows the filtered event:

Time	Event Type	Details								
1/17/17 4:36:46 PM	LOGIN	<table border="1"> <tr><td>Client</td><td>security-admin-console</td></tr> <tr><td>User</td><td>780d038c-b0a4-4268-8492-36ba96573a9e</td></tr> <tr><td>IP Address</td><td>127.0.0.1</td></tr> <tr><td>Details</td><td>+</td></tr> </table>	Client	security-admin-console	User	780d038c-b0a4-4268-8492-36ba96573a9e	IP Address	127.0.0.1	Details	+
Client	security-admin-console									
User	780d038c-b0a4-4268-8492-36ba96573a9e									
IP Address	127.0.0.1									
Details	+									

In this example, we filter only **Login** events. Click **Update** to run the filter.

14.1.1. Event types

Login events:

Event	Description
Login	A user logs in.
Register	A user registers.

Event	Description
Logout	A user logs out.
Code to Token	An application, or client, exchanges a code for a token.
Refresh Token	An application, or client, refreshes a token.

Account events:

Event	Description
Social Link	A user account links to a social media provider.
Remove Social Link	The link from a social media account to a user account severs.
Update Email	An email address for an account changes.
Update Profile	A profile for an account changes.
Send Password Reset	Red Hat Single Sign-On sends a password reset email.
Update Password	The password for an account changes.
Update TOTP	The Time-based One-time Password (TOTP) settings for an account changes.
Remove TOTP	Red Hat Single Sign-On removes TOTP from an account.
Send Verify Email	Red Hat Single Sign-On sends an email verification email.
Verify Email	Red Hat Single Sign-On verifies the email address for an account.

Each event has a corresponding error event.

14.1.2. Event listener

Event listeners listen for events and perform actions based on that event. Red Hat Single Sign-On includes two built-in listeners, the Logging Event Listener and Email Event Listener.

14.1.2.1. The logging event listener

When the Logging Event Listener is enabled, this listener writes to a log file when an error event occurs.

An example log message from a Logging Event Listener:

```
11:36:09,965 WARN [org.keycloak.events] (default task-51) type=LOGIN_ERROR, realmId=master,
  clientId=myapp,
  userId=19aeb848-96fc-44f6-b0a3-59a17570d374, ipAddress=127.0.0.1,
  error=invalid_user_credentials, auth_method=openid-connect, auth_type=code,
  redirect_uri=http://localhost:8180/myapp,
  code_id=b669da14-cdbb-41d0-b055-0810a0334607, username=admin
```

You can use the Logging Event Listener to protect against hacker bot attacks:

1. Parse the log file for the **LOGIN_ERROR** event.
2. Extract the IP Address of the failed login event.
3. Send the IP address to an intrusion prevention software framework tool.

The Logging Event Listener logs events to the **org.keycloak.events** log category. Red Hat Single Sign-On does not include debug log events in server logs, by default.

To include debug log events in server logs:

1. Edit the **standalone.xml** file.
2. Change the log level used by the Logging Event listener.

Alternately, you can configure the log level for **org.keycloak.events**.

For example, to change the log level add the following:

```
<subsystem xmlns="urn:jboss:domain:logging:...">
  ...
  <logger category="org.keycloak.events">
    <level name="DEBUG"/>
  </logger>
</subsystem>
```

To change the log level used by the Logging Event listener, add the following:

```
<subsystem xmlns="urn:jboss:domain:keycloak-server:...">
  ...
  <spi name="eventsListener">
    <provider name="jboss-logging" enabled="true">
      <properties>
        <property name="success-level" value="info"/>
        <property name="error-level" value="error"/>
      </properties>
    </provider>
  </spi>
</subsystem>
```

The valid values for log levels are **debug**, **info**, **warn**, **error**, and **fatal**.

14.1.2.2. The Email Event Listener

The Email Event Listener sends an email to the user's account when an event occurs and supports the following events:

- Login Error.
- Update Password.
- Update Time-based One-time Password (TOTP).
- Remove Time-based One-time Password (TOTP).

Procedure

To enable the Email Listener:

1. Click **Events** from the menu.
2. Click the **Config** tab.
3. Click the **Event Listeners** field.
4. Select **email**.

You can exclude events by editing the **standalone.xml**, **standalone-ha.xml**, or **domain.xml** configuration files included in your distribution. For example:

```
<spi name="eventsListener">
  <provider name="email" enabled="true">
    <properties>
      <property name="exclude-events" value="
[&quot;UPDATE_TOTP&quot;,&quot;REMOVE_TOTP&quot;]"/>
    </properties>
  </provider>
</spi>
```

You can set a maximum length of the Event detail in the database by editing the **standalone.xml**, **standalone-ha.xml**, or **domain.xml** configuration files. This setting is useful if a field (for example, `redirect_uri`) is long. For example:

```
<spi name="eventsStore">
  <provider name="jpa" enabled="true">
    <properties>
      <property name="max-detail-length" value="1000"/>
    </properties>
  </provider>
</spi>
```

See the [Server Installation and Configuration Guide](#) for more details on the location of the **standalone.xml**, **standalone-ha.xml**, or **domain.xml** files.

14.2. ADMIN EVENTS

You can record all actions that are performed by an administrator in the Admin Console. The Admin Console performs administrative actions by invoking the Red Hat Single Sign-On REST interface and

Red Hat Single Sign-On audits these REST invocations. You can view the resulting events in the Admin Console.

To enable auditing of Admin actions:

Procedure

1. Click **Events** in the menu.
2. Click the **Config** tab.

Event configuration

The screenshot displays the 'Events Config' page in the Admin Console. On the left, a dark sidebar menu is open, showing 'Master' at the top and 'Configure' expanded below it. Under 'Configure', options include 'Realm Settings', 'Clients', 'Client Scopes', 'Roles', 'Identity Providers', 'User Federation', and 'Authentication'. Under 'Manage', options include 'Groups' and 'Users'. The main content area has a header 'Events Config' with a help icon and three tabs: 'Login Events', 'Admin Events', and 'Config'. Below the tabs, there is a section for 'Event Listeners' with a search box containing 'jboss-logging'. This is followed by 'Login Events Settings' and 'Admin Events Settings' sections, each containing a 'Save Events' toggle switch currently set to 'OFF'. At the bottom of the 'Admin Events Settings' section, there are 'Clear changes' and 'Save' buttons.

3. Toggle **Save Events** to **ON** in the **Admin Events Settings** section. Red Hat Single Sign-On displays the **Include Representation** switch.

Admin event configuration

4. Toggle **Include Representation** to **ON**.

The **Include Representation** switch includes JSON documents sent through the admin REST API so you can view the administrators actions. To clear the database of stored actions, click **Clear admin events**.

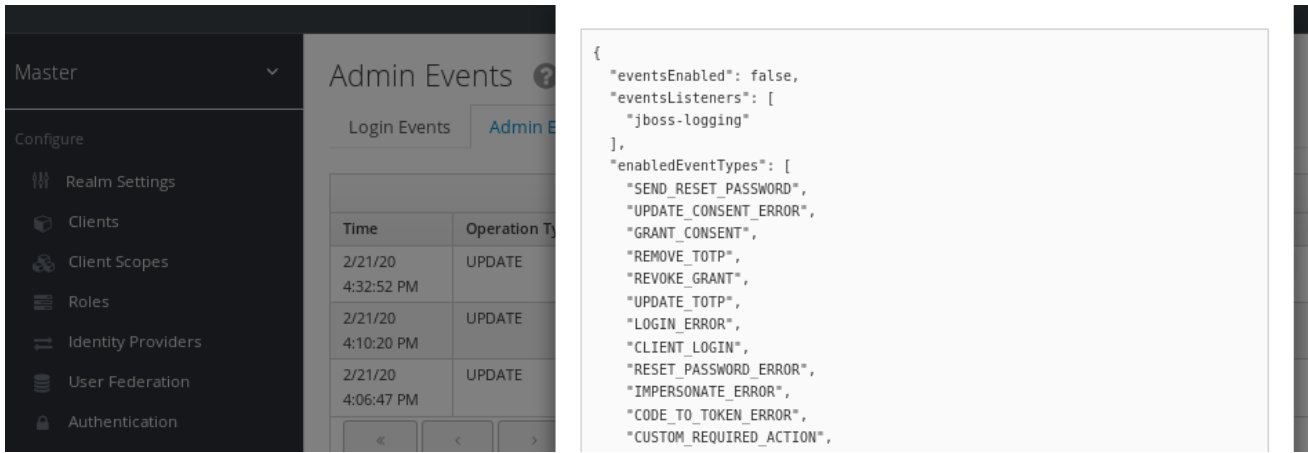
To view the admin events, click the **Admin Events** tab.

Admin events

Time	Operation Type	Resource Type	Resource Path	Details
2/21/20 4:32:52 PM	UPDATE	REALM	events/config	Auth Representation
2/21/20 4:10:20 PM	UPDATE	REALM	events/config	Auth
2/21/20 4:06:47 PM	UPDATE	REALM	events/config	Auth

If the **Details** column has a **Representation** button, click the **Representation** button to view the JSON Red Hat Single Sign-On sent with the operation.

Admin representation



The screenshot shows the 'Admin Events' configuration page. On the left is a navigation menu with options like 'Realm Settings', 'Clients', 'Roles', etc. The main area displays a table of events:

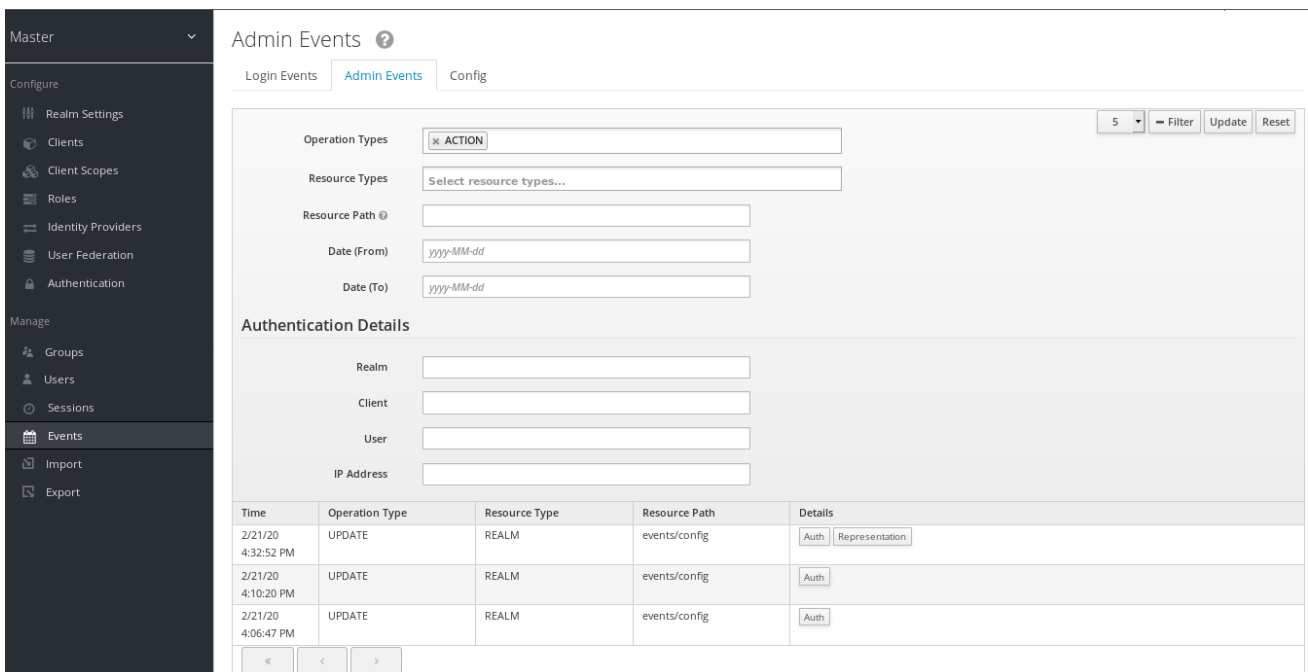
Time	Operation Type
2/21/20 4:32:52 PM	UPDATE
2/21/20 4:10:20 PM	UPDATE
2/21/20 4:06:47 PM	UPDATE

To the right, a JSON configuration snippet is shown:

```
{
  "eventsEnabled": false,
  "eventsListeners": [
    "jboss-logging"
  ],
  "enabledEventTypes": [
    "SEND_RESET_PASSWORD",
    "UPDATE_CONSENT_ERROR",
    "GRANT_CONSENT",
    "REMOVE_TOTP",
    "REVOKE_GRANT",
    "UPDATE_TOTP",
    "LOGIN_ERROR",
    "CLIENT_LOGIN",
    "RESET_PASSWORD_ERROR",
    "IMPERSONATE_ERROR",
    "CODE_TO_TOKEN_ERROR",
    "CUSTOM_REQUIRED_ACTION",
  ]
}
```

Click **Filter** to view specific events.

Admin event filter



The screenshot shows the 'Admin Events' configuration page with the 'Filter' section expanded. The configuration fields are:

- Operation Types:
- Resource Types:
- Resource Path:
- Date (From):
- Date (To):

The 'Authentication Details' section contains:

- Realm:
- Client:
- User:
- IP Address:

At the top right of the filter section are buttons for '5', 'Filter', 'Update', and 'Reset'. Below the configuration fields is a table of events:

Time	Operation Type	Resource Type	Resource Path	Details
2/21/20 4:32:52 PM	UPDATE	REALM	events/config	Auth Representation
2/21/20 4:10:20 PM	UPDATE	REALM	events/config	Auth
2/21/20 4:06:47 PM	UPDATE	REALM	events/config	Auth

CHAPTER 15. IMPORTING AND EXPORTING THE DATABASE

Red Hat Single Sign-On includes the ability to export and import its entire database.

You can migrate the whole Red Hat Single Sign-On database from one environment to another or migrate to another database. The export/import triggers at server boot time, and its parameters pass through Java properties.



NOTE

Because import and export trigger at server startup, take no actions on the server or the database during export/import.

You can export/import your database to:

- A directory on the filesystem.
- A single JSON file on your filesystem.

When importing from a directory, the filenames must follow this naming convention:

- `<REALM_NAME>-realm.json`. For example, "acme-roadrunner-affairs-realm.json" for the realm named "acme-roadrunner-affairs".
- `<REALM_NAME>-users-<INDEX>.json`. For example, "acme-roadrunner-affairs-users-0.json" for the first user's file of the realm named "acme-roadrunner-affairs"

If you export to a directory, you can specify the number of users stored in each JSON file.



NOTE

Exporting into single files can produce large files, so if your database contains more than 500 users, export to a directory and not a single file. Exporting many users into a directory performs optimally as the directory provider uses a separate transaction for each "page" (a file of users).

The default count of users per file and per transaction is fifty, but you can override this number. See [keycloak.migration.usersPerFile](#) for more information.

Exporting to or importing from a single file uses one transaction, which can impair performance if the database contains many users.

To export into an unencrypted directory:

```
bin/standalone.sh -Dkeycloak.migration.action=export
-Dkeycloak.migration.provider=dir -Dkeycloak.migration.dir=<DIR TO EXPORT TO>
```

To export into single JSON file:

```
bin/standalone.sh -Dkeycloak.migration.action=export
-Dkeycloak.migration.provider=singleFile -Dkeycloak.migration.file=<FILE TO EXPORT TO>
```

Similarly, for importing, use **-Dkeycloak.migration.action=import** rather than **export**. For example:

```
bin/standalone.sh -Dkeycloak.migration.action=import
-Dkeycloak.migration.provider=singleFile -Dkeycloak.migration.file=<FILE TO IMPORT>
-Dkeycloak.migration.strategy=OVERWRITE_EXISTING
```

Other command line options include:

-Dkeycloak.migration.realmName

Use this property to export one specifically named realm. If this parameter is not specified, all realms export.

-Dkeycloak.migration.usersExportStrategy

This property specifies where users export to. Possible values include:

- **DIFFERENT_FILES** - Users export into different files subject to the maximum [number of users per file](#). **DIFFERENT_FILES** is the default value for this property.
- **SKIP** - Red Hat Single Sign-On skips exporting users.
- **REALM_FILE** - Users export to the same file with the realm settings. The file is similar to "foo-realm.json" with realm data and users.
- **SAME_FILE** - Users export to the same file but different from the realm file. The result is similar to "foo-realm.json" with realm data and "foo-users.json" with users.

-Dkeycloak.migration.usersPerFile

This property specifies the number of users per file and database transaction. By default, its value is fifty. Red Hat Single Sign-On uses this property if `keycloak.migration.usersExportStrategy` is **DIFFERENT_FILES**.

-Dkeycloak.migration.strategy

Red Hat Single Sign-On uses this property when importing. It specifies how to proceed when a realm with the same name already exists in the database.

Possible values are:

- **IGNORE_EXISTING** - Do not import a realm if a realm with the same name already exists.
- **OVERWRITE_EXISTING** - Remove the existing realm and import the realm again with new data from the JSON file. Use this value to migrate from one environment to another fully.

If you are importing files that are not from a Red Hat Single Sign-On export, use the **keycloak.import** option. If you are importing more than one realm file, specify a comma-separated list of filenames. A list of filenames is more suitable than the previous cases because this happens after Red Hat Single Sign-On initializes the master realm.

Examples:

- `-Dkeycloak.import=/tmp/realm1.json`
- `-Dkeycloak.import=/tmp/realm1.json,/tmp/realm2.json`

**NOTE**

You cannot use the **keycloak.import** parameter with **keycloak.migration.X** parameters. If you use these parameters together, Red Hat Single Sign-On ignores the **keycloak.import** parameter. The **keycloak.import** mechanism ignores the realms which already exist in Red Hat Single Sign-On. The **keycloak.import** mechanism is convenient for development purposes, but if more flexibility is needed, use the **keycloak.migration.X** parameters.

15.1. ADMIN CONSOLE EXPORT/IMPORT

Red Hat Single Sign-On imports most resources from the Admin Console as well as exporting most resources. Red Hat Single Sign-On does not support the export of users.

**NOTE**

Red Hat Single Sign-On masks attributes containing secrets or private information in the export file. Export files from the Admin Console are not suitable for backups or data transfer between servers. Only boot-time exports are suitable for backups or data transfer between servers.

You can use the files created during an export to import from the Admin Console. You can export from one realm and import to another realm, or you can export from one server and import to another.

**NOTE**

The admin console export/import permits one realm per file only.

**WARNING**

The Admin Console import can overwrite resources. Use this feature with caution, especially on a production server. JSON files from the Admin Console Export operation are not appropriate for data import because they contain invalid values for secrets.

**WARNING**

You can use the Admin Console to export clients, groups, and roles. If the database in your realm contains many clients, groups, and roles, the export may take a long time to conclude, and the Red Hat Single Sign-On server may not respond to user requests. Use this feature with caution, especially on a production server.

CHAPTER 16. MITIGATING SECURITY THREATS

Security vulnerabilities exist in any authentication server. See the Internet Engineering Task Force's (IETF) [OAuth 2.0 Threat Model](#) and the [OAuth 2.0 Security Best Current Practice](#) for more information.

16.1. HOST

Red Hat Single Sign-On uses the public hostname in several ways, such as within token issuer fields and URLs in password reset emails.

By default, the hostname derives from request headers. No validation exists to ensure a hostname is valid. If you are not using a load balancer, or proxy, with Red Hat Single Sign-On to prevent invalid host headers, configure the acceptable hostnames.

The hostname's Service Provider Interface (SPI) provides a way to configure the hostname for requests. You can use this built-in provider to set a fixed URL for frontend requests while allowing backend requests based on the request URI. If the built-in provider does not have the required capability, you can develop a customized provider.

16.2. ADMIN ENDPOINTS AND ADMIN CONSOLE

Red Hat Single Sign-On exposes the administrative REST API and the web console on the same port as non-administrative usage by default. Do not expose administrative endpoints externally if external access is not necessary. If you need to expose administrative endpoints externally, you can expose them directly in Red Hat Single Sign-On or use a proxy.

To expose endpoints by using a proxy, consult the documentation for the proxy. You need to control access to requests to the **/auth/admin** endpoint.

Two options are available in Red Hat Single Sign-On to expose endpoints directly, IP restriction and separate ports.

When the Admin Console becomes inaccessible on the frontend URL of Red Hat Single Sign-On, configure a fixed admin URL in the default hostname provider.

16.2.1. IP restriction

You can restrict access to **/auth/admin** to only specific IP addresses. For example, restrict access to **/auth/admin** to IP addresses in the range **10.0.0.1** to **10.0.0.255**.

```
<subsystem xmlns="urn:jboss:domain:undertow:12.0">
  ...
  <server name="default-server">
    ...
    <host name="default-host" alias="localhost">
      ...
      <filter-ref name="ipAccess"/>
    </host>
  </server>
  <filters>
    <expression-filter name="ipAccess" expression="path-prefix('/auth/admin') -> ip-access-control(acl={'10.0.0.0/24 allow'})"/>
  </filters>
</subsystem>
```

```

</filters>
...
</subsystem>

```

You can also restrict access to specific IP addresses by using these CLI commands:

```

/subsystem=undertow/configuration=filter/expression-filter=ipAccess:add(,expression="path-
prefix[/auth/admin] -> ip-access-control(acl={'10.0.0.0/24 allow'})")
/subsystem=undertow/server=default-server/host=default-host/filter-ref=ipAccess:add()

```



NOTE

For IP restriction using a proxy, configure the proxy to ensure Red Hat Single Sign-On receives the client IP address and not the proxy IP address.

16.2.2. Port restriction

You can expose **/auth/admin** to a different unexposed port. For example, expose **/auth/admin** on port **8444** and prevent access to the default port **8443**.

```

<subsystem xmlns="urn:jboss:domain:undertow:12.0">
...
  <server name="default-server">
...
    <https-listener name="https" socket-binding="https" security-realm="ApplicationRealm" enable-
http2="true"/>
    <https-listener name="https-admin" socket-binding="https-admin" security-
realm="ApplicationRealm" enable-http2="true"/>
    <host name="default-host" alias="localhost">
...
      <filter-ref name="portAccess"/>
    </host>
  </server>
  <filters>
    <expression-filter name="portAccess" expression="path-prefix('/auth/admin') and not equals(%p,
8444) -> response-code(403)"/>
  </filters>
...
</subsystem>
...

<socket-binding-group name="standard-sockets" default-interface="public" port-
offset="${jboss.socket.binding.port-offset:0}">
...
  <socket-binding name="https" port="${jboss.https.port:8443}"/>
  <socket-binding name="https-admin" port="${jboss.https.port:8444}"/>
...
</socket-binding-group>

```

You can expose **/auth/admin** on port **8444** and prevent access to the default port **8443** by using CLI commands:

```

/socket-binding-group=standard-sockets/socket-binding=https-admin/:add(port=8444)

```



```
/subsystem=undertow/server=default-server/https-listener=https-admin:add(socket-binding=https-admin, security-realm=ApplicationRealm, enable-http2=true)
```

```
/subsystem=undertow/configuration=filter/expression-filter=portAccess:add(,expression="path-prefix('/auth/admin') and not equals(%p, 8444) -> response-code(403)")
```

```
/subsystem=undertow/server=default-server/host=default-host/filter-ref=portAccess:add()
```

16.3. BRUTE FORCE ATTACKS

A brute force attack attempts to guess a user's password by trying to log in multiple times. Red Hat Single Sign-On has brute force detection capabilities and can temporarily disable a user account if the number of login failures exceeds a specified threshold.



NOTE

Red Hat Single Sign-On disables brute force detection by default. Enable this feature to protect against brute force attacks.

Procedure

To enable this protection:

1. Click **Realm Settings** in the menu
2. Click the **Security Defenses** tab.
3. Click the **Brute Force Detection** tab.

Brute force detection

The screenshot shows the Red Hat Single Sign-On administration console. On the left is a dark sidebar menu with 'Master' at the top, followed by 'Configure' and 'Manage' sections. Under 'Configure', 'Realm Settings' is selected. The main content area shows the 'Master' configuration page with tabs for 'General', 'Login', 'Keys', 'Email', 'Themes', 'Cache', 'Tokens', 'Client Registration', and 'Security Defenses'. The 'Security Defenses' tab is active, and the 'Brute Force Detection' sub-tab is selected. The settings are as follows:

- Enabled:** ON (toggle)
- Permanent Lockout:** OFF (toggle)
- Max Login Failures:** 30 (input field)
- Wait Increment:** 1 (input field) / Minutes (dropdown)
- Quick Login Check Milli Seconds:** 1000 (input field)
- Minimum Quick Login Wait:** 1 (input field) / Minutes (dropdown)
- Max Wait:** 15 (input field) / Minutes (dropdown)
- Failure Reset Time:** 12 (input field) / Hours (dropdown)

At the bottom of the settings are 'Save' and 'Cancel' buttons.

Red Hat Single Sign-On can deploy permanent lockout and temporary lockout actions when it detects an attack. Permanent lockout disables a user account until an administrator re-enables it. Temporary lockout disables a user account for a specific period of time. The time period that the account is disabled increases as the attack continues.

**NOTE**

When a user is temporarily locked and attempts to log in, Red Hat Single Sign-On displays the default **Invalid username or password** error message. This message is the same error message as the message displayed for an invalid username or invalid password to ensure the attacker is unaware the account is disabled.

Common Parameters

Name	Description	Default
Max Login Failures	The maximum number of login failures.	30 failures.
Quick Login Check Milliseconds	The minimum time between login attempts.	1000 milliseconds.
Minimum Quick Login Wait	The minimum time the user is disabled when login attempts are quicker than <i>Quick Login Check Milliseconds</i> .	1 minute.

Permanent Lockout Flow

1. On successful login
 - a. Reset **count**
2. On failed login
 - a. Increment **count**
 - b. If **count** greater than *Max Login Failures*
 - i. Permanently disable user
 - c. Else if the time between this failure and the last failure is less than *Quick Login Check Milliseconds*
 - i. Temporarily disable user for *Minimum Quick Login Wait*

When Red Hat Single Sign-On disables a user, the user cannot log in until an administrator enables the user. Enabling an account resets the **count**.

Temporary Lockout Parameters

Name	Description	Default
Wait Increment	The time added to the time a user is temporarily disabled when the user's login attempts exceed <i>Max Login Failures</i> .	1 minute.

Name	Description	Default
Max Wait	The maximum time a user is temporarily disabled.	15 minutes.
Failure Reset Time	The time when the failure count resets. The timer runs from the last failed login.	12 hours.

Temporary Lockout Algorithm

1. On successful login
 - a. Reset **count**
2. On failed login
 - a. If the time between this failure and the last failure is greater than *Failure Reset Time*
 - i. Reset **count**
 - b. Increment **count**
 - c. Calculate **wait** using $Wait\ Increment * (count / Max\ Login\ Failures)$. The division is an integer division rounded down to a whole number
 - d. If **wait** equals 0 and the time between this failure and the last failure is less than *Quick Login Check Milliseconds*, set **wait** to *Minimum Quick Login Wait*.
 - i. Temporarily disable the user for the smaller of **wait** and *Max Wait* seconds

'count' does not increment when a temporarily disabled account commits a login failure.

The downside of Red Hat Single Sign-On brute force detection is that the server becomes vulnerable to denial of service attacks. When implementing a denial of service attack, an attacker can attempt to log in by guessing passwords for any accounts it knows and eventually causing Red Hat Single Sign-On to disable the accounts.

Consider using intrusion prevention software (IPS). Red Hat Single Sign-On logs every login failure and client IP address failure. You can point the IPS to the Red Hat Single Sign-On server's log file, and the IPS can modify firewalls to block connections from these IP addresses.

16.3.1. Password policies

Ensure you have a complex password policy to force users to choose complex passwords. See the [Password Policies](#) chapter for more information. Prevent password guessing by setting up the Red Hat Single Sign-On server to use one-time-passwords.

16.4. READ-ONLY USER ATTRIBUTES

Typical users who are stored in Red Hat Single Sign-On have various attributes related to their user profiles. Such attributes include email, firstName or lastName. However users may also have attributes, which are not typical profile data, but rather metadata. The metadata attributes usually should be read-

only for the users and the typical users never should have a way to update those attributes from the Red Hat Single Sign-On user interface or Account REST API. Some of the attributes should be even read-only for the administrators when creating or updating user with the Admin REST API.

The metadata attributes are usually attributes from those groups:

- Various links or metadata related to the user storage providers. For example in case of the LDAP integration, the **LDAP_ID** attribute contains the ID of the user in the LDAP server.
- Metadata provisioned by User Storage. For example **createdTimestamp** provisioned from the LDAP should be always read-only by user or administrator.
- Metadata related to various authenticators. For example **KERBEROS_PRINCIPAL** attribute can contain the kerberos principal name of the particular user. Similarly attribute **usercertificate** can contain metadata related to binding the user with the data from the X.509 certificate, which is used typically when X.509 certificate authentication is enabled.
- Metadata related to the identifier of users by the applications/clients. For example **saml.persistent.name.id.for.my_app** can contain SAML NameID, which will be used by the client application **my_app** as the identifier of the user.
- Metadata related to the authorization policies, which are used for the attribute based access control (ABAC). Values of those attributes may be used for the authorization decisions. Hence it is important that those attributes cannot be updated by the users.

From the long term perspective, Red Hat Single Sign-On will have a proper User Profile SPI, which will allow fine-grained configuration of every user attribute. Currently this capability is not fully available yet. So Red Hat Single Sign-On has the internal list of user attributes, which are read-only for the users and read-only for the administrators configured at the server level.

This is the list of the read-only attributes, which are used internally by the Red Hat Single Sign-On default providers and functionalities and hence are always read-only:

- For users: **KERBEROS_PRINCIPAL, LDAP_ID, LDAP_ENTRY_DN, CREATED_TIMESTAMP, createTimestamp, modifyTimestamp, userCertificate, saml.persistent.name.id.for.*, ENABLED, EMAIL_VERIFIED**
- For administrators: **KERBEROS_PRINCIPAL, LDAP_ID, LDAP_ENTRY_DN, CREATED_TIMESTAMP, createTimestamp, modifyTimestamp**

System administrators have a way to add additional attributes to this list. The configuration is currently available at the server level.

You can add this configuration to your **standalone(-*).xml** files to the configuration of the Red Hat Single Sign-On server subsystem:

```
<spi name="userProfile">
  <provider name="legacy-user-profile" enabled="true">
    <properties>
      <property name="read-only-attributes" value="[&quot;foo&quot;,&quot;bar*&quot;]"/>
      <property name="admin-read-only-attributes" value="[&quot;foo&quot;]"/>
    </properties>
  </provider>
</spi>
```

The same can be configured with the usage of the JBoss CLI with the commands:

-

```

/subsystem=keycloak-server/spi=userProfile/:add
/subsystem=keycloak-server/spi=userProfile/provider=legacy-user-profile/:add(properties=
 {},enabled=true)
/subsystem=keycloak-server/spi=userProfile/provider=legacy-user-profile/:map-
put(name=properties,key=read-only-attributes,value=[foo,bar*])
/subsystem=keycloak-server/spi=userProfile/provider=legacy-user-profile/:map-
put(name=properties,key=admin-read-only-attributes,value=[foo])

```

For this example, users and administrators would not be able to update attribute **foo**. Users would not be able to edit any attributes starting with the **bar**. So for example **bar** or **barrier**. Configuration is case insensitive, so attributes like **FOO** or **BarRier** will be denied as well for this example. The wildcard character ***** is supported only at the end of the attribute name, so the administrator can effectively deny all the attributes starting with the specified character. The ***** in the middle of the attribute is considered as a normal character.

16.5. CLICKJACKING

Clickjacking is a technique of tricking users into clicking on a user interface element different from what users perceive. A malicious site loads the target site in a transparent iFrame, overlaid on top of a set of dummy buttons placed directly under important buttons on the target site. When a user clicks a visible button, they are clicking a button on the hidden page. An attacker can steal a user's authentication credentials and access their resources by using this method.

By default, every response by Red Hat Single Sign-On sets some specific HTTP headers that can prevent this from happening. Specifically, it sets [X-Frame-Options](#) and [Content-Security-Policy](#). You should take a look at the definition of both of these headers as there is a lot of fine-grain browser access you can control.

Procedure

In the Admin Console, you can specify the values of the X-Frame-Options and Content-Security-Policy headers.

1. Click the **Realm Settings** menu item.
2. Click the **Security Defenses** tab.

Security Defenses

The screenshot shows the Keycloak Admin Console interface. On the left is a dark sidebar with a 'Master' dropdown and a 'Configure' section containing 'Realm Settings' (highlighted) and 'Manage' (Groups, Users, Sessions, Events, Import, Export). The main content area is titled 'Master' and has tabs for 'General', 'Login', 'Keys', 'Email', 'Themes', 'Cache', 'Tokens', 'Client Registration', and 'Security Defenses' (selected). Under 'Security Defenses', there are two sub-tabs: 'Headers' (selected) and 'Brute Force Detection'. The 'Headers' tab contains several configuration fields:

- X-Frame-Options**: ALLOW-FROM https://www.google.com
- Content-Security-Policy**: frame-src 'self'; https://www.google.com
- Content-Security-Policy-Report-Only**: (empty field)
- X-Content-Type-Options**: nosniff
- X-Robots-Tag**: none
- X-XSS-Protection**: 1; mode=block
- HTTP Strict Transport Security (HSTS)**: max-age=31536000; includeSubDomains

At the bottom of the configuration area are 'Save' and 'Cancel' buttons.

By default, Red Hat Single Sign-On only sets up a *same-origin* policy for iframes.

16.6. SSL/HTTPS REQUIREMENT

OAuth 2.0/OpenID Connect uses access tokens for security. Attackers can scan your network for access tokens and use them to perform malicious operations for which the token has permission. This attack is known as a man-in-the-middle attack. Use SSL/HTTPS for communication between the Red Hat Single Sign-On auth server and the clients Red Hat Single Sign-On secures to prevent man-in-the-middle attacks.

Red Hat Single Sign-On has [three modes for SSL/HTTPS](#). SSL is complex to set up, so Red Hat Single Sign-On allows non-HTTPS communication over private IP addresses such as localhost, 192.168.x.x, and other private IP addresses. In production, ensure you enable SSL and SSL is compulsory for all operations.

On the adapter/client-side, you can disable the SSL trust manager. The trust manager ensures the client's identity that Red Hat Single Sign-On communicates with is valid and ensures the DNS domain name against the server's certificate. In production, ensure that each of your client adapters uses a truststore to prevent DNS man-in-the-middle attacks.

16.7. CSRF ATTACKS

A Cross-site request forgery (CSRF) attack uses HTTP requests from users that websites have already authenticated. Any site using cookie-based authentication is vulnerable to CSRF attacks. You can mitigate these attacks by matching a state cookie against a posted form or query parameter.

The OAuth 2.0 login specification requires that a state cookie matches against a transmitted state parameter. Red Hat Single Sign-On fully implements this part of the specification, so all logins are protected.

The Red Hat Single Sign-On Admin Console is a JavaScript/HTML5 application that makes REST calls to the backend Red Hat Single Sign-On admin REST API. These calls all require bearer token authentication and consist of JavaScript Ajax calls, so CSRF is impossible. You can configure the admin REST API to validate the CORS origins.

The user account management section in Red Hat Single Sign-On can be vulnerable to CSRF. To prevent CSRF attacks, Red Hat Single Sign-On sets a state cookie and embeds the value of this cookie in hidden form fields or query parameters within action links. Red Hat Single Sign-On checks the query/form parameter against the state cookie to verify that the user makes the call.

16.8. UNSPECIFIC REDIRECT URIS

Make your registered redirect URIs as specific as feasible. Registering vague redirect URIs for [Authorization Code Flows](#) can allow malicious clients to impersonate another client with broader access. Impersonation can happen if two clients live under the same domain, for example.

16.9. FAPI COMPLIANCE

To make sure that Red Hat Single Sign-On server will validate your client to be more secure and FAPI compliant, you can configure client policies for the FAPI support. Details are described in the FAPI section of [Securing Applications and Services Guide](#). Among other things, this ensures some security best practices described above like SSL required for clients, secure redirect URI used and more of similar best practices.

16.10. COMPROMISED ACCESS AND REFRESH TOKENS

Red Hat Single Sign-On includes several actions to prevent malicious actors from stealing access tokens and refresh tokens. The crucial action is to enforce SSL/HTTPS communication between Red Hat Single Sign-On and its clients and applications. Red Hat Single Sign-On does not enable SSL by default.

Another action to mitigate damage from leaked access tokens is to shorten the token's lifespans. You can specify token lifespans within the [timeouts page](#). Short lifespans for access tokens force clients and applications to refresh their access tokens after a short time. If an admin detects a leak, the admin can log out all user sessions to invalidate these refresh tokens or set up a revocation policy.

Ensure refresh tokens always stay private to the client and are never transmitted.

You can mitigate damage from leaked access tokens and refresh tokens by issuing these tokens as holder-of-key tokens. See [OAuth 2.0 Mutual TLS Client Certificate Bound Access Token](#) for more information.

If an access token or refresh token is compromised, access the Admin Console and push a not-before revocation policy to all applications. Pushing a not-before policy ensures that any tokens issued before that time become invalid. Pushing a new not-before policy ensures that applications must download new public keys from Red Hat Single Sign-On and mitigate damage from a compromised realm signing key. See the [keys chapter](#) for more information.

You can disable specific applications, clients, or users if they are compromised.

16.11. COMPROMISED AUTHORIZATION CODE

For the [OIDC Auth Code Flow](#), Red Hat Single Sign-On generates a cryptographically strong random value for its authorization codes. An authorization code is used only once to obtain an access token.

On the [timeouts page](#) in the Admin Console, you can specify the length of time an authorization code is valid. Ensure that the length of time is less than 10 seconds, which is long enough for a client to request a token from the code.

You can also defend against leaked authorization codes by applying [Proof Key for Code Exchange \(PKCE\)](#) to clients.

16.12. OPEN REDIRECTORS

An open redirector is an endpoint using a parameter to automatically redirect a user agent to the location specified by the parameter value without validation. An attacker can use the end-user authorization endpoint and the redirect URI parameter to use the authorization server as an open redirector, using a user's trust in an authorization server to launch a phishing attack.

Red Hat Single Sign-On requires that all registered applications and clients register at least one redirection URI pattern. When a client requests that Red Hat Single Sign-On performs a redirect, Red Hat Single Sign-On checks the redirect URI against the list of valid registered URI patterns. Clients and applications must register as specific a URI pattern as possible to mitigate open redirector attacks.

If an application requires a non http(s) custom scheme, it should be an explicit part of the validation pattern (for example `custom:/app/*`). For security reasons a general pattern like `*` does not cover non http(s) schemes.

16.13. PASSWORD DATABASE COMPROMISED

Red Hat Single Sign-On does not store passwords in raw text but as hashed text, using the PBKDF2 hashing algorithm. Red Hat Single Sign-On performs 27,500 hashing iterations, the number of iterations recommended by the security community. This number of hashing iterations can adversely affect performance as PBKDF2 hashing uses a significant amount of CPU resources.

16.14. LIMITING SCOPE

By default, new client applications have unlimited **role scope mappings**. Every access token for that client contains all permissions that the user has. If an attacker compromises the client and obtains the client's access tokens, each system that the user can access is compromised.

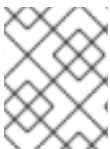
Limit the roles of an access token by using the [Scope menu](#) for each client. Alternatively, you can set role scope mappings at the Client Scope level and assign Client Scopes to your client by using the [Client Scope menu](#).

16.15. LIMIT TOKEN AUDIENCE

In environments with low levels of trust among services, limit the audiences on the token. See the [OAuth2 Threat Model](#) and the [Audience Support](#) section for more information.

16.16. LIMIT AUTHENTICATION SESSIONS

When a login page is opened for the first time in a web browser, Red Hat Single Sign-On creates an object called authentication session that stores some useful information about the request. Whenever a new login page is opened from a different tab in the same browser, Red Hat Single Sign-On creates a new record called authentication sub-session that is stored within the authentication session. Authentication requests can come from any type of clients such as the Admin CLI. In that case, a new authentication session is also created with one authentication sub-session. Please note that authentication sessions can be created also in other ways than using a browser flow. The text below is applicable regardless of the source flow.



NOTE

This section describes deployments that use the RHDG provider for authentication sessions.

Authentication session is internally stored as **RootAuthenticationSessionEntity**. Each **RootAuthenticationSessionEntity** can have multiple authentication sub-sessions stored within the **RootAuthenticationSessionEntity** as a collection of **AuthenticationSessionEntity** objects. Red Hat Single Sign-On stores authentication sessions in a dedicated RHDG cache. The number of **AuthenticationSessionEntity** per **RootAuthenticationSessionEntity** contributes to the size of each cache entry. Total memory footprint of authentication session cache is determined by the number of stored **RootAuthenticationSessionEntity** and by the number of **AuthenticationSessionEntity** within each **RootAuthenticationSessionEntity**.

The number of maintained **RootAuthenticationSessionEntity** objects corresponds to the number of unfinished login flows from the browser. To keep the number of **RootAuthenticationSessionEntity** under control, using an advanced firewall control to limit ingress network traffic is recommended.

Higher memory usage may occur for deployments where there are many active **RootAuthenticationSessionEntity** with a lot of **AuthenticationSessionEntity**. If the load balancer does not support or is not configured for [session stickiness](#), the load over network in a cluster can increase significantly. The reason for this load is that each request that lands on a node that does not

own the appropriate authentication session needs to retrieve and update the authentication session record in the owner node which involves a separate network transmission for both the retrieval and the storage.

The maximum number of **AuthenticationSessionEntity** per **RootAuthenticationSessionEntity** can be configured in **authenticationSessions** SPI by setting property **authSessionsLimit**. The default value is set to 300 **AuthenticationSessionEntity** per a **RootAuthenticationSessionEntity**. When this limit is reached, the oldest authentication sub-session will be removed after a new authentication session request.

The following example shows how to limit the number of active **AuthenticationSessionEntity** per a **RootAuthenticationSessionEntity** to 100.

```
<subsystem xmlns="urn:jboss:domain:keycloak-server:1.2">
  ...
  <spi name="authenticationSessions">
    <default-provider>infinispan</default-provider>
    <provider name="infinispan" enabled="true">
      <properties>
        <property name="authSessionsLimit" value="100"/>
      </properties>
    </provider>
  </spi>
  ...
</subsystem>
```

Equivalent configuration using CLI commands:

```
/subsystem=keycloak-server/spi=authenticationSessions:add(default-provider=infinispan)
/subsystem=keycloak-server/spi=authenticationSessions/provider=infinispan:add(properties=
{authSessionsLimit => "100"},enabled=true)
```

16.17. SQL INJECTION ATTACKS

Currently, Red Hat Single Sign-On has no known SQL injection vulnerabilities.

CHAPTER 17. ACCOUNT CONSOLE

Red Hat Single Sign-On users can manage their accounts through the Account Console. Users can configure their profiles, add two-factor authentication, include identity provider accounts, and oversee device activity.

Additional resources

- The Account Console can be configured in terms of appearance and language preferences. An example is adding attributes to the **Personal info** page by clicking **Personal info** link and completing and saving details. For more information, see reference: [https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.6/html-single/server_developer_guide/\[Server Developer Guide\]](https://access.redhat.com/documentation/en-us/red_hat_single_sign-on/7.6/html-single/server_developer_guide/[Server Developer Guide]).

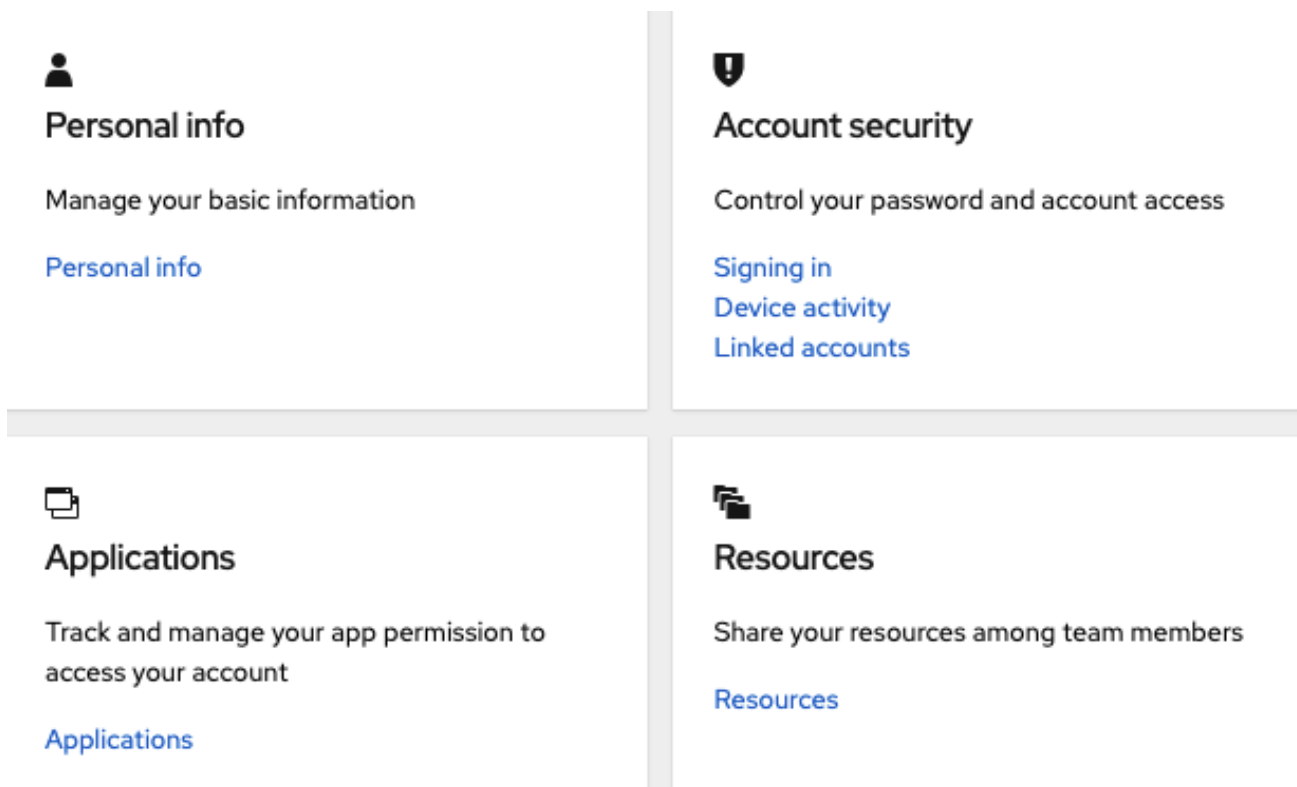
17.1. ACCESSING THE ACCOUNT CONSOLE

Any user can access the Account Console.

Procedure

1. Make note of the realm name and IP address for the Red Hat Single Sign-On server where your account exists.
2. In a web browser, enter a URL in this format: **<server-root>/auth/realms/{realm-name}/account**.
3. Enter your login name and password.

Account Console



17.2. CONFIGURING WAYS TO SIGN IN

You can sign in to this console using basic authentication (a login name and password) or two-factor authentication. For two-factor authentication, use one of the following procedures.

17.2.1. Two-factor authentication with OTP

Prerequisites

- OTP is a valid authentication mechanism for your realm.

Procedure

1. Click **Account security** in the menu.
2. Click **Signing in**.
3. Click **Set up authenticator application**.

Signing in

Signing in

Configure ways to sign in.

Basic authentication

Password

Sign in by entering your password.

My password	Created May 3, 2022, 11:56 AM	Update
-------------	---	------------------------

Two-factor authentication

Authenticator application



Enter a verification code from authenticator application.

Authenticator application is not set up.

4. Follow the directions that appear on the screen to use either [FreeOTP](#) or [Google Authenticator](#) on your mobile device as your OTP generator.
5. Scan the QR code in the screen shot into the OTP generator on your mobile device.
6. Log out and log in again.
7. Respond to the prompt by entering an OTP that is provided on your mobile device.

17.2.2. Two-factor authentication with WebAuthn

Prerequisites

- WebAuthn is a valid two-factor authentication mechanism for your realm. Please follow the [WebAuthn](#) section for more details.

Procedure

1. Click **Account Security** in the menu.
2. Click **Signing In**.
3. Click **Set up Security Key**.

Signing In

Basic Authentication

Password

Log in by entering your password.

My Password
Created: August 19, 2021,
11:26 AM
Update

Two-Factor Authentication

Authenticator Application [Set up Authenticator Application](#)

Enter a verification code from authenticator application.

Authenticator Application is not set up.

Security Key [Set up Security Key](#)

Use your security key to log in.

Security Key is not set up.

4. Prepare your WebAuthn Security Key. How you prepare this key depends on the type of WebAuthn security key you use. For example, for a USB based Yubikey, you may need to put your key into the USB port on your laptop.
5. Click **Register** to register your security key.
6. Log out and log in again.
7. Assuming authentication flow was correctly set, a message appears asking you to authenticate with your Security Key as second factor.

17.2.3. Passwordless authentication with WebAuthn

Prerequisites

- WebAuthn is a valid passwordless authentication mechanism for your realm. Please follow the [Passwordless WebAuthn section](#) for more details.

Procedure

1. Click **Account Security** in the menu.
2. Click **Signing In**.
3. Click **Set up Security Key** in the **Passwordless** section.

Signing In

Basic Authentication

Password

Log in by entering your password.

My Password
Created: August 19, 2021,
11:26 AM
Update

Two-Factor Authentication

Authenticator Application [Set up Authenticator Application](#)

Enter a verification code from authenticator application.

Authenticator Application is not set up.

Passwordless

Security Key [Set up Security Key](#)

Use your security key for passwordless log in.

Security Key is not set up.

4. Prepare your WebAuthn Security Key. How you prepare this key depends on the type of WebAuthn security key you use. For example, for a USB based Yubikey, you may need to put your key into the USB port on your laptop.
5. Click **Register** to register your security key.
6. Log out and log in again.
7. Assuming authentication flow was correctly set, a message appears asking you to authenticate with your Security Key as second factor. You no longer need to provide your password to log in.

17.3. VIEWING DEVICE ACTIVITY

You can view the devices that are logged in to your account.

Procedure

1. Click **Account security** in the menu.
2. Click **Device activity**.
3. Log out a device if it looks suspicious.

Devices

Device activity

Sign out of any unfamiliar devices.

Signed in devices

 Refresh

IP address	Last accessed	Clients	Started	Expires
127.0.0.1	June 1, 2022, 11:36 AM	security-admin-console-v2, Account, Account Console	June 1, 2022, 10:54 AM	June 1, 2022, 8:54 PM

17.4. ADDING AN IDENTITY PROVIDER ACCOUNT

You can link your account with an [identity broker](#). This option is often used to link social provider accounts.

Procedure

1. Log into the Admin Console.
2. Click **Identity providers** in the menu.
3. Select a provider and complete the fields.
4. Return to the Account Console.
5. Click **Account security** in the menu.
6. Click **Linked accounts**.

The identity provider you added appears in this page.

Linked Accounts

Linked accounts

Manage logins through third-party accounts.

Linked login providers

No linked providers

Unlinked login providers



GitHub

Social login

[Link account](#)

17.5. ACCESSING OTHER APPLICATIONS

The **Applications** menu item shows users which applications you can access. In this case, only the Account Console is available.

Applications

Applications

Manage your application permissions.

	Name	Application type	Status
>	Security-admin-console-v2	Internal	In use
>	Account	Internal	In use
>	Account Console	Internal	In use

CHAPTER 18. ADMIN CLI

With Red Hat Single Sign-On, you can perform administration tasks from the command-line interface (CLI) by using the Admin CLI command-line tool.

18.1. INSTALLING THE ADMIN CLI

Red Hat Single Sign-On packages the Admin CLI server distribution with the execution scripts in the **bin** directory.

The Linux script is called **kcadm.sh**, and the script for Windows is called **kcadm.bat**. Add the Red Hat Single Sign-On server directory to your **PATH** to use the client from any location on your file system.

For example:

- Linux:

```
$ export PATH=$PATH:$KEYCLOAK_HOME/bin
$ kcadm.sh
```

- Windows:

```
c:\> set PATH=%PATH%;%KEYCLOAK_HOME%\bin
c:\> kcadm
```



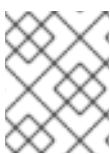
NOTE

You must set the **KEYCLOAK_HOME** environment variable to the path where you extracted the Red Hat Single Sign-On Server distribution.

To avoid repetition, the rest of this document only uses Windows examples in places where the CLI differences are more than just in the **kcadm** command name.

18.2. USING THE ADMIN CLI

The Admin CLI makes HTTP requests to Admin REST endpoints. Access to the Admin REST endpoints requires authentication.



NOTE

Consult the Admin REST API documentation for details about JSON attributes for specific endpoints.

1. Start an authenticated session by logging in. You can now perform create, read, update, and delete (CRUD) operations.

For example:

- Linux:

```
$ kcadm.sh config credentials --server http://localhost:8080/auth --realm demo --user
admin --client admin
$ kcadm.sh create realms -s realm=demorealm -s enabled=true -o
```



```
$ CID=$(kcadm.sh create clients -r demorealm -s clientId=my_client -s 'redirectUri=
["http://localhost:8980/myapp/*"]' -i)
$ kcadm.sh get clients/$CID/installation/providers/keycloak-oidc-keycloak-json
```

- Windows:

```
c:\> kcadm config credentials --server http://localhost:8080/auth --realm demo --user
admin --client admin
c:\> kcadm create realms -s realm=demorealm -s enabled=true -o
c:\> kcadm create clients -r demorealm -s clientId=my_client -s "redirectUri=
["http://localhost:8980/myapp/*\"]" -i > clientid.txt
c:\> set /p CID=<clientid.txt
c:\> kcadm get clients/%CID%/installation/providers/keycloak-oidc-keycloak-json
```

2. In a production environment, access Red Hat Single Sign-On by using **https:** to avoid exposing tokens. If a trusted certificate authority, included in Java's default certificate truststore, has not issued a server's certificate, prepare a **truststore.jks** file and instruct the Admin CLI to use it. For example:

- Linux:

```
$ kcadm.sh config truststore --trustpass $PASSWORD ~/.keycloak/truststore.jks
```

- Windows:

```
c:\> kcadm config truststore --trustpass %PASSWORD%
%HOMEPATH%\keycloak\truststore.jks
```

18.3. AUTHENTICATING

When you log in with the Admin CLI, you specify:

- A server endpoint URL
- A realm
- A user name

Another option is to specify a `clientId` only, which creates a unique service account for you to use.

When you log in using a user name, use a password for the specified user. When you log in using a `clientId`, you need the client secret only, not the user password. You can also use the **Signed JWT** rather than the client secret.

Ensure the account used for the session has the proper permissions to invoke Admin REST API operations. For example, the **realm-admin** role of the **realm-management** client can administer the realm of the user.

Two primary mechanisms are available for authentication. One mechanism uses **kcadm config credentials** to start an authenticated session.

```
$ kcadm.sh config credentials --server http://localhost:8080/auth --realm master --user admin --
password admin
```

This mechanism maintains an authenticated session between the **kcadm** command invocations by saving the obtained access token and its associated refresh token. It can maintain other secrets in a private configuration file. See the [next chapter](#) for more information.

The second mechanism authenticates each command invocation for the duration of the invocation. This mechanism increases the load on the server and the time spent on round trips obtaining tokens. The benefit of this approach is that it is unnecessary to save tokens between invocations, so nothing is saved to disk. Red Hat Single Sign-On uses this mode when the **--no-config** argument is specified.

For example, when performing an operation, specify all the information required for authentication.

```
$ kcadm.sh get realms --no-config --server http://localhost:8080/auth --realm master --user admin --password admin
```

Run the **kcadm.sh help** command for more information on using the Admin CLI.

Run the **kcadm.sh config credentials --help** command for more information about starting an authenticated session.

18.4. WORKING WITH ALTERNATIVE CONFIGURATIONS

By default, the Admin CLI maintains a configuration file named **kcadm.config**. Red Hat Single Sign-On places this file in the user's home directory. In Linux-based systems, the full pathname is **\$HOME/.keycloak/kcadm.config**. In Windows, the full pathname is **%HOMEPATH%\keycloak\kcadm.config**.

You can use the **--config** option to point to a different file or location so you can maintain multiple authenticated sessions in parallel.



NOTE

Perform operations tied to a single configuration file from a single thread.

Ensure the configuration file is invisible to other users on the system. It contains access tokens and secrets that must be private. Red Hat Single Sign-On creates the **~/.keycloak** directory and its contents automatically with proper access limits. If the directory already exists, Red Hat Single Sign-On does not update the directory's permissions.

It is possible to avoid storing secrets inside a configuration file, but doing so is inconvenient and increases the number of token requests. Use the **--no-config** option with all commands and specify the authentication information the **config credentials** command requires with each invocation of **kcadm**.

18.5. BASIC OPERATIONS AND RESOURCE URIS

The Admin CLI can generically perform CRUD operations against Admin REST API endpoints with additional commands that simplify particular tasks.

The main usage pattern is listed here:

```
$ kcadm.sh create ENDPOINT [ARGUMENTS]
$ kcadm.sh get ENDPOINT [ARGUMENTS]
$ kcadm.sh update ENDPOINT [ARGUMENTS]
$ kcadm.sh delete ENDPOINT [ARGUMENTS]
```

The **create**, **get**, **update**, and **delete** commands map to the HTTP verbs **POST**, **GET**, **PUT**, and **DELETE**, respectively. ENDPOINT is a target resource URI and can be absolute (starting with **http:** or **https:**) or relative, that Red Hat Single Sign-On uses to compose absolute URLs in the following format:

```
SERVER_URI/admin/realms/REALM/ENDPOINT
```

For example, if you authenticate against the server <http://localhost:8080/auth> and realm is **master**, using **users** as ENDPOINT creates the <http://localhost:8080/auth/admin/realms/master/users> resource URL.

If you set ENDPOINT to **clients**, the effective resource URI is <http://localhost:8080/auth/admin/realms/master/clients>.

Red Hat Single Sign-On has a **realms** endpoint that is the container for realms. It resolves to:

```
SERVER_URI/admin/realms
```

Red Hat Single Sign-On has a **serverinfo** endpoint. This endpoint is independent of realms.

When you authenticate as a user with realm-admin powers, you may need to perform commands on multiple realms. If so, specify the **-r** option to tell the CLI which realm the command is to execute against explicitly. Instead of using **REALM** as specified by the **--realm** option of **kcadm.sh config credentials**, the command uses **TARGET_REALM**.

```
SERVER_URI/admin/realms/TARGET_REALM/ENDPOINT
```

For example:

```
$ kcadm.sh config credentials --server http://localhost:8080/auth --realm master --user admin --
password admin
$ kcadm.sh create users -s username=testuser -s enabled=true -r demorealm
```

In this example, you start a session authenticated as the **admin** user in the **master** realm. You then perform a POST call against the resource URL <http://localhost:8080/auth/admin/realms/demorealm/users>.

The **create** and **update** commands send a JSON body to the server. You can use **-f FILENAME** to read a pre-made document from a file. When you can use the **-f -** option, Red Hat Single Sign-On reads the message body from the standard input. You can specify individual attributes and their values, as seen in the **create users** example. Red Hat Single Sign-On composes the attributes into a JSON body and sends them to the server.

Several methods are available in Red Hat Single Sign-On to update a resource using the **update** command. You can determine the current state of a resource and save it to a file, edit that file, and send it to the server for an update.

For example:

```
$ kcadm.sh get realms/demorealm > demorealm.json
$ vi demorealm.json
$ kcadm.sh update realms/demorealm -f demorealm.json
```

This method updates the resource on the server with the attributes in the sent JSON document.

Another method is to perform an on-the-fly update by using the **-s**, **--set** options to set new values.

For example:

```
$ kcadm.sh update realms/demorealm -s enabled=false
```

This method sets the **enabled** attribute to **false**.

By default, the **update** command performs a **get** and then merges the new attribute values with existing values. In some cases, the endpoint may support the **put** command but not the **get** command. You can use the **-n** option to perform a no-merge update, which performs a **put** command without first running a **get** command.

18.6. REALM OPERATIONS

Creating a new realm

Use the **create** command on the **realms** endpoint to create a new enabled realm. Set the attributes to **realm** and **enabled**.

```
$ kcadm.sh create realms -s realm=demorealm -s enabled=true
```

Red Hat Single Sign-On disables realms by default. You can use a realm immediately for authentication by enabling it.

A description for a new object can also be in JSON format.

```
$ kcadm.sh create realms -f demorealm.json
```

You can send a JSON document with realm attributes directly from a file or pipe the document to standard input.

For example:

- Linux:

```
$ kcadm.sh create realms -f - << EOF
{ "realm": "demorealm", "enabled": true }
EOF
```

- Windows:

```
c:\> echo { "realm": "demorealm", "enabled": true } | kcadm create realms -f -
```

Listing existing realms

This command returns a list of all realms.

```
$ kcadm.sh get realms
```



NOTE

Red Hat Single Sign-On filters the list of realms on the server to return realms a user can see only.

The list of all realm attributes can be verbose, and most users are interested in a subset of attributes, such as the realm name and the enabled status of the realm. You can specify the attributes to return by using the **--fields** option.

```
$ kcadm.sh get realms --fields realm,enabled
```

You can display the result as comma-separated values.

```
$ kcadm.sh get realms --fields realm --format csv --noquotes
```

Getting a specific realm

Append a realm name to a collection URI to get an individual realm.

```
$ kcadm.sh get realms/master
```

Updating a realm

1. Use the **-s** option to set new values for the attributes when you do not want to change all of the realm's attributes.

For example:

```
$ kcadm.sh update realms/demorealm -s enabled=false
```

2. If you want to set all writable attributes to new values:

- a. Run a **get** command.
- b. Edit the current values in the JSON file.
- c. Resubmit.
For example:

```
$ kcadm.sh get realms/demorealm > demorealm.json
$ vi demorealm.json
$ kcadm.sh update realms/demorealm -f demorealm.json
```

Deleting a realm

Run the following command to delete a realm:

```
$ kcadm.sh delete realms/demorealm
```

Turning on all login page options for the realm

Set the attributes that control specific capabilities to **true**.

For example:

```
$ kcadm.sh update realms/demorealm -s registrationAllowed=true -s
registrationEmailAsUsername=true -s rememberMe=true -s verifyEmail=true -s
resetPasswordAllowed=true -s editUsernameAllowed=true
```

Listing the realm keys

Use the **get** operation on the **keys** endpoint of the target realm.

```
$ kcadm.sh get keys -r demorealm
```

Generating new realm keys

1. Get the ID of the target realm before adding a new RSA-generated key pair.

For example:

```
$ kcadm.sh get realms/demorealm --fields id --format csv --noquotes
```

2. Add a new key provider with a higher priority than the existing providers as revealed by **kcadm.sh get keys -r demorealm**.

For example:

- Linux:

```
$ kcadm.sh create components -r demorealm -s name=rsa-generated -s providerId=rsa-generated -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s 'config.priority=["101"]' -s 'config.enabled=["true"]' -s 'config.active=["true"]' -s 'config.keySize=["2048"]'
```

- Windows:

```
c:\> kcadm create components -r demorealm -s name=rsa-generated -s providerId=rsa-generated -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s "config.priority=["101"]" -s "config.enabled=["true"]" -s "config.active=["true"]" -s "config.keySize=["2048"]"
```

3. Set the **parentId** attribute to the value of the target realm's ID.

The newly added key is now the active key, as revealed by **kcadm.sh get keys -r demorealm**.

Adding new realm keys from a Java Key Store file

1. Add a new key provider to add a new key pair pre-prepared as a JKS file.

For example, on:

- Linux:

```
$ kcadm.sh create components -r demorealm -s name=java-keystore -s providerId=java-keystore -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s 'config.priority=["101"]' -s 'config.enabled=["true"]' -s 'config.active=["true"]' -s 'config.keystore=["/opt/keycloak/keystore.jks"]' -s 'config.keystorePassword=["secret"]' -s 'config.keyPassword=["secret"]' -s 'config.keyAlias=["localhost"]'
```

- Windows:

```
c:\> kcadm create components -r demorealm -s name=java-keystore -s providerId=java-keystore -s providerType=org.keycloak.keys.KeyProvider -s parentId=959844c1-d149-41d7-8359-6aa527fca0b0 -s "config.priority=["101"]" -s "config.enabled=["true"]" -s "config.active=["true"]" -s "config.keystore=["/opt/keycloak/keystore.jks"]" -s "config.keystorePassword=["secret"]" -s "config.keyPassword=["secret"]" -s "config.keyAlias=["localhost"]"
```

2. Ensure you change the attribute values for **keystore**, **keystorePassword**, **keyPassword**, and **alias** to match your specific keystore.
3. Set the **parentId** attribute to the value of the target realm's ID.

Making the key passive or disabling the key

1. Identify the key you want to make passive.

```
$ kcadm.sh get keys -r demorealm
```

2. Use the key's **providerId** attribute to construct an endpoint URI, such as **components/PROVIDER_ID**.

3. Perform an **update**.

For example:

- Linux:

```
$ kcadm.sh update components/PROVIDER_ID -r demorealm -s 'config.active=["false"]'
```

- Windows:

```
c:\> kcadm update components/PROVIDER_ID -r demorealm -s "config.active=["false"]"
```

You can update other key attributes:

4. Set a new **enabled** value to disable the key, for example, **config.enabled=["false"]**.
5. Set a new **priority** value to change the key's priority, for example, **config.priority=["110"]**.

Deleting an old key

1. Ensure the key you are deleting is inactive and you have disabled it. This action is to prevent existing tokens held by applications and users from failing.
2. Identify the key to delete.

```
$ kcadm.sh get keys -r demorealm
```

3. Use the **providerId** of the key to perform the delete.

```
$ kcadm.sh delete components/PROVIDER_ID -r demorealm
```

Configuring event logging for a realm

Use the **update** command on the **events/config** endpoint.

The **eventsListeners** attribute contains a list of EventListenerProviderFactory IDs, specifying all event listeners that receive events. Attributes are available that control built-in event storage, so you can query past events using the Admin REST API. Red Hat Single Sign-On has separate control over the logging of service calls (**eventsEnabled**) and the auditing events triggered by the Admin Console or Admin REST API (**adminEventsEnabled**). You can set up the **eventsExpiration** event to expire to prevent your database from filling. Red Hat Single Sign-On sets **eventsExpiration** to time-to-live expressed in seconds.

You can set up a built-in event listener that receives all events and logs the events through JBoss-logging. Using the **org.keycloak.events** logger, Red Hat Single Sign-On logs error events as **WARN** and other events as **DEBUG**.

For example:

- Linux:

```
$ kcadm.sh update events/config -r demorealm -s 'eventsListeners=["jboss-logging"]'
```

- Windows:

```
c:\> kcadm update events/config -r demorealm -s "eventsListeners=["jboss-logging"]"
```

For example:

You can turn on storage for all available ERROR events, not including auditing events, for two days so you can retrieve the events through Admin REST.

- Linux:

```
$ kcadm.sh update events/config -r demorealm -s eventsEnabled=true -s 'enabledEventTypes=["LOGIN_ERROR","REGISTER_ERROR","LOGOUT_ERROR","CODE_TO_TOKEN_ERROR","CLIENT_LOGIN_ERROR","FEDERATED_IDENTITY_LINK_ERROR","REMOVE_FEDERATED_IDENTITY_ERROR","UPDATE_EMAIL_ERROR","UPDATE_PROFILE_ERROR","UPDATE_PASSWORD_ERROR","UPDATE_TOTP_ERROR","VERIFY_EMAIL_ERROR","REMOVE_TOTP_ERROR","SEND_VERIFY_EMAIL_ERROR","SEND_RESET_PASSWORD_ERROR","SEND_IDENTITY_PROVIDER_LINK_ERROR","RESET_PASSWORD_ERROR","IDENTITY_PROVIDER_FIRST_LOGIN_ERROR","IDENTITY_PROVIDER_POST_LOGIN_ERROR","CUSTOM_REQUIRED_ACTION_ERROR","EXECUTE_ACTIONS_ERROR","CLIENT_REGISTER_ERROR","CLIENT_UPDATE_ERROR","CLIENT_DELETE_ERROR"]' -s eventsExpiration=172800
```

- Windows:

```
c:\> kcadm update events/config -r demorealm -s eventsEnabled=true -s "enabledEventTypes=["LOGIN_ERROR","REGISTER_ERROR","LOGOUT_ERROR","CODE_TO_TOKEN_ERROR","CLIENT_LOGIN_ERROR","FEDERATED_IDENTITY_LINK_ERROR","REMOVE_FEDERATED_IDENTITY_ERROR","UPDATE_EMAIL_ERROR","UPDATE_PROFILE_ERROR","UPDATE_PASSWORD_ERROR","UPDATE_TOTP_ERROR","VERIFY_EMAIL_ERROR","REMOVE_TOTP_ERROR","SEND_VERIFY_EMAIL_ERROR","SEND_RESET_PASSWORD_ERROR","SEND_IDENTITY_PROVIDER_LINK_ERROR","RESET_PASSWORD_ERROR","IDENTITY_PROVIDER_FIRST_LOGIN_ERROR","IDENTITY_PROVIDER_POST_LOGIN_ERROR","CUSTOM_REQUIRED_ACTION_ERROR","EXECUTE_ACTIONS_ERROR","CLIENT_REGISTER_ERROR","CLIENT_UPDATE_ERROR","CLIENT_DELETE_ERROR"]" -s eventsExpiration=172800
```

You can reset stored event types to **all available event types**. Setting the value to an empty list is the same as enumerating all.

```
$ kcadm.sh update events/config -r demorealm -s enabledEventTypes=[]
```

You can enable storage of auditing events.

```
$ kcadm.sh update events/config -r demorealm -s adminEventsEnabled=true -s adminEventsDetailsEnabled=true
```


You can get the last 100 events. The events are ordered from newest to oldest.

```
$ kcadm.sh get events --offset 0 --limit 100
```

You can delete all saved events.

```
$ kcadm delete events
```

Flushing the caches

1. Use the **create** command with one of these endpoints to clear caches:

- **clear-realm-cache**
- **clear-user-cache**
- **clear-keys-cache**

2. Set **realm** to the same value as the target realm.

For example:

```
$ kcadm.sh create clear-realm-cache -r demorealm -s realm=demorealm
$ kcadm.sh create clear-user-cache -r demorealm -s realm=demorealm
$ kcadm.sh create clear-keys-cache -r demorealm -s realm=demorealm
```

Importing a realm from exported .json file

1. Use the **create** command on the **partialImport** endpoint.

2. Set **ifResourceExists** to **FAIL**, **SKIP**, or **OVERWRITE**.

3. Use **-f** to submit the exported realm **.json** file.

For example:

```
$ kcadm.sh create partialImport -r demorealm2 -s ifResourceExists=FAIL -o -f
demorealm.json
```

If the realm does not yet exist, create it first.

For example:

```
$ kcadm.sh create realms -s realm=demorealm2 -s enabled=true
```

18.7. ROLE OPERATIONS

Creating a realm role

Use the **roles** endpoint to create a realm role.

```
$ kcadm.sh create roles -r demorealm -s name=user -s 'description=Regular user with a limited set of
permissions'
```

Creating a client role

1. Identify the client.
2. Use the **get** command to list the available clients.

```
$ kcadm.sh get clients -r demorealm --fields id,clientId
```

3. Create a new role by using the **clientId** attribute to construct an endpoint URI, such as **clients/ID/roles**.

For example:

```
$ kcadm.sh create clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles -r demorealm -s name=editor -s 'description=Editor can edit, and publish any article'
```

Listing realm roles

Use the **get** command on the **roles** endpoint to list existing realm roles.

```
$ kcadm.sh get roles -r demorealm
```

You can use the **get-roles** command also.

```
$ kcadm.sh get-roles -r demorealm
```

Listing client roles

Red Hat Single Sign-On has a dedicated **get-roles** command to simplify the listing of realm and client roles. The command is an extension of the **get** command and behaves the same as the **get** command but with additional semantics for listing roles.

Use the **get-roles** command by passing it the **clientId** (**--cclientid**) option or the **id** (**--cid**) option to identify the client to list client roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --cclientid realm-management
```

Getting a specific realm role

Use the **get** command and the role **name** to construct an endpoint URI for a specific realm role, **roles/ROLE_NAME**, where **user** is the existing role's name.

For example:

```
$ kcadm.sh get roles/user -r demorealm
```

You can use the **get-roles** command, passing it a role name (**--rolename** option) or ID (**--roleid** option).

For example:

```
$ kcadm.sh get-roles -r demorealm --rolename user
```

Getting a specific client role

Use the **get-roles** command, passing it the **clientId** attribute (**--cclientid** option) or ID attribute (**--cid** option) to identify the client, and pass the role name (**--rolename** option) or the role ID attribute (**--roleid**) to identify a specific client role.

For example:

```
$ kcadm.sh get-roles -r demorealm --cclientid realm-management --rolename manage-clients
```

Updating a realm role

Use the **update** command with the endpoint URI you used to get a specific realm role.

For example:

```
$ kcadm.sh update roles/user -r demorealm -s 'description=Role representing a regular user'
```

Updating a client role

Use the **update** command with the endpoint URI that you used to get a specific client role.

For example:

```
$ kcadm.sh update clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles/editor -r demorealm -s 'description=User that can edit, and publish articles'
```

Deleting a realm role

Use the **delete** command with the endpoint URI that you used to get a specific realm role.

For example:

```
$ kcadm.sh delete roles/user -r demorealm
```

Deleting a client role

Use the **delete** command with the endpoint URI that you used to get a specific client role.

For example:

```
$ kcadm.sh delete clients/a95b6af3-0bdc-4878-ae2e-6d61a4eca9a0/roles/editor -r demorealm
```

Listing assigned, available, and effective realm roles for a composite role

Use the **get-roles** command to list assigned, available, and effective realm roles for a composite role.

1. To list **assigned** realm roles for the composite role, specify the target composite role by name (**-rname** option) or ID (**--rid** option).

For example:

```
$ kcadm.sh get-roles -r demorealm --rname testrole
```

2. Use the **--effective** option to list **effective** realm roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --rname testrole --effective
```

3. Use the **--available** option to list realm roles that you can add to the composite role.

For example:

```
$ kcadm.sh get-roles -r demorealm --rname testrole --available
```

Listing assigned, available, and effective client roles for a composite role

Use the **get-roles** command to list assigned, available, and effective client roles for a composite role.

1. To list **assigned** client roles for the composite role, you can specify the target composite role by name (**--rname** option) or ID (**--rid** option) and client by the clientId attribute (**--cclientid** option) or ID (**--cid** option).

For example:

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management
```

2. Use the **--effective** option to list **effective** realm roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management --effective
```

3. Use the **--available** option to list realm roles that you can add to the target composite role.

For example:

```
$ kcadm.sh get-roles -r demorealm --rname testrole --cclientid realm-management --available
```

Adding realm roles to a composite role

Red Hat Single Sign-On provides an **add-roles** command for adding realm roles and client roles.

This example adds the **user** role to the composite role **testrole**.

```
$ kcadm.sh add-roles --rname testrole --rolename user -r demorealm
```

Removing realm roles from a composite role

Red Hat Single Sign-On provides a **remove-roles** command for removing realm roles and client roles.

The following example removes the **user** role from the target composite role **testrole**.

```
$ kcadm.sh remove-roles --rname testrole --rolename user -r demorealm
```

Adding client roles to a realm role

Red Hat Single Sign-On provides an **add-roles** command for adding realm roles and client roles.

The following example adds the roles defined on the client **realm-management**, **create-client**, and **view-users**, to the **testrole** composite role.

```
$ kcadm.sh add-roles -r demorealm --rname testrole --cclientid realm-management --rolename create-client --rolename view-users
```

Adding client roles to a client role

1. Determine the ID of the composite client role by using the **get-roles** command.

For example:

```
$ kcadm.sh get-roles -r demorealm --cclientid test-client --rolename operations
```

2. Assume that a client exists with a `clientId` attribute named **test-client**, a client role named **support**, and a client role named **operations** which becomes a composite role that has an ID of "fc400897-ef6a-4e8c-872b-1581b7fa8a71".
3. Use the following example to add another role to the composite role.

```
$ kcadm.sh add-roles -r demorealm --cclientid test-client --rid fc400897-ef6a-4e8c-872b-1581b7fa8a71 --rolename support
```

4. List the roles of a composite role by using the **get-roles --all** command.
For example:

```
$ kcadm.sh get-roles --rid fc400897-ef6a-4e8c-872b-1581b7fa8a71 --all
```

Removing client roles from a composite role

Use the **remove-roles** command to remove client roles from a composite role.

Use the following example to remove two roles defined on the client **realm-management**, the **create-client** role and the **view-users** role, from the **testrole** composite role.

```
$ kcadm.sh remove-roles -r demorealm --rname testrole --cclientid realm-management --rolename create-client --rolename view-users
```

Adding client roles to a group

Use the **add-roles** command to add realm roles and client roles.

The following example adds the roles defined on the client **realm-management**, **create-client** and **view-users**, to the **Group** group (**--gname** option). Alternatively, you can specify the group by ID (**--gid** option).

See [Group operations](#) for more information.

```
$ kcadm.sh add-roles -r demorealm --gname Group --cclientid realm-management --rolename create-client --rolename view-users
```

Removing client roles from a group

Use the **remove-roles** command to remove client roles from a group.

The following example removes two roles defined on the client **realm management**, **create-client** and **view-users**, from the **Group** group.

See [Group operations](#) for more information.

```
$ kcadm.sh remove-roles -r demorealm --gname Group --cclientid realm-management --rolename create-client --rolename view-users
```

18.8. CLIENT OPERATIONS

Creating a client

1. Run the **create** command on a **clients** endpoint to create a new client.
For example:

```
$ kcadm.sh create clients -r demorealm -s clientId=myapp -s enabled=true
```

2. Specify a secret if to set a secret for adapters to authenticate.
For example:

```
$ kcadm.sh create clients -r demorealm -s clientId=myapp -s enabled=true -s  
clientAuthenticatorType=client-secret -s secret=d0b8122f-8dfb-46b7-b68a-f5cc4e25d000
```

Listing clients

Use the **get** command on the **clients** endpoint to list clients.

This example filters the output to list only the **id** and **clientId** attributes:

```
$ kcadm.sh get clients -r demorealm --fields id,clientId
```

Getting a specific client

Use the client ID to construct an endpoint URI that targets a specific client, such as **clients/ID**.

For example:

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm
```

Getting the current secret for a specific client

Use the client ID to construct an endpoint URI, such as **clients/ID/client-secret**.

For example:

```
$ kcadm.sh get clients/$CID/client-secret
```

Generate a new secret for a specific client

Use the client ID to construct an endpoint URI, such as **clients/ID/client-secret**.

For example:

```
$ kcadm.sh create clients/$CID/client-secret
```

Updating the current secret for a specific client

Use the client ID to construct an endpoint URI, such as **clients/ID**.

For example:

```
$ kcadm.sh update clients/$CID -s "secret=newSecret"
```

Getting an adapter configuration file (keycloak.json) for a specific client

Use the client ID to construct an endpoint URI that targets a specific client, such as **clients/ID/installation/providers/keycloak-oidc-keycloak-json**.

For example:

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3/installation/providers/keycloak-oidc-  
keycloak-json -r demorealm
```

Getting a WildFly subsystem adapter configuration for a specific client

Use the client ID to construct an endpoint URI that targets a specific client, such as **clients/ID/installation/providers/keycloak-oidc-jboss-subsystem**.

For example:

```
$ kcadm.sh get clients/c7b8547f-e748-4333-95d0-410b76b3f4a3/installation/providers/keycloak-oidc-jboss-subsystem -r demorealm
```

Getting a Docker-v2 example configuration for a specific client

Use the client ID to construct an endpoint URI that targets a specific client, such as **clients/ID/installation/providers/docker-v2-compose-yaml**.

The response is in **.zip** format.

For example:

```
$ kcadm.sh get http://localhost:8080/auth/admin/realms/demorealm/clients/8f271c35-44e3-446f-8953-b0893810ebe7/installation/providers/docker-v2-compose-yaml -r demorealm > keycloak-docker-compose-yaml.zip
```

Updating a client

Use the **update** command with the same endpoint URI that you use to get a specific client.

For example:

- Linux:

```
$ kcadm.sh update clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm -s enabled=false -s publicClient=true -s 'redirectUri=["http://localhost:8080/myapp/*"]' -s baseUrl=http://localhost:8080/myapp -s adminUrl=http://localhost:8080/myapp
```

- Windows:

```
c:\> kcadm update clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm -s enabled=false -s publicClient=true -s "redirectUri=["http://localhost:8080/myapp/*"]" -s baseUrl=http://localhost:8080/myapp -s adminUrl=http://localhost:8080/myapp
```

Deleting a client

Use the **delete** command with the same endpoint URI that you use to get a specific client.

For example:

```
$ kcadm.sh delete clients/c7b8547f-e748-4333-95d0-410b76b3f4a3 -r demorealm
```

Adding or removing roles for client's service account

A client's service account is a user account with username **service-account-CLIENT_ID**. You can perform the same user operations on this account as a regular account.

18.9. USER OPERATIONS

Creating a user

Run the **create** command on the **users** endpoint to create a new user.

For example:

```
$ kcadm.sh create users -r demorealm -s username=testuser -s enabled=true
```

Listing users

Use the **users** endpoint to list users. The target user must change their password the next time they log in.

For example:

```
$ kcadm.sh get users -r demorealm --offset 0 --limit 1000
```

You can filter users by **username**, **firstName**, **lastName**, or **email**.

For example:

```
$ kcadm.sh get users -r demorealm -q email=google.com
$ kcadm.sh get users -r demorealm -q username=testuser
```



NOTE

Filtering does not use exact matching. This example matches the value of the **username** attribute against the ***testuser*** pattern.

You can filter across multiple attributes by specifying multiple **-q** options. Red Hat Single Sign-On returns users that match the condition for all the attributes only.

Getting a specific user

Use the user ID to compose an endpoint URI, such as **users/USER_ID**.

For example:

```
$ kcadm.sh get users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm
```

Updating a user

Use the **update** command with the same endpoint URI that you use to get a specific user.

For example:

- Linux:

```
$ kcadm.sh update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm -s
'requiredActions=
["VERIFY_EMAIL","UPDATE_PROFILE","CONFIGURE_TOTP","UPDATE_PASSWORD"]'
```

- Windows:

```
c:\> kcadm update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm -s "requiredActions=
["VERIFY_EMAIL","\UPDATE_PROFILE","\CONFIGURE_TOTP","\UPDATE_PASSWORD"]"
```

Deleting a user

Use the **delete** command with the same endpoint URI that you use to get a specific user.

For example:

```
$ kcadm.sh delete users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2 -r demorealm
```

Resetting a user's password

Use the dedicated **set-password** command to reset a user's password.

For example:

```
$ kcadm.sh set-password -r demorealm --username testuser --new-password NEWPASSWORD --temporary
```

This command sets a temporary password for the user. The target user must change the password the next time they log in.

You can use **--userid** to specify the user by using the **id** attribute.

You can achieve the same result using the **update** command on an endpoint constructed from the one you used to get a specific user, such as **users/USER_ID/reset-password**.

For example:

```
$ kcadm.sh update users/0ba7a3fd-6fd8-48cd-a60b-2e8fd82d56e2/reset-password -r demorealm -s type=password -s value=NEWPASSWORD -s temporary=true -n
```

The **-n** parameter ensures that Red Hat Single Sign-On performs the **PUT** command without performing a **GET** command before the **PUT** command. This is necessary because the **reset-password** endpoint does not support **GET**.

Listing assigned, available, and effective realm roles for a user

You can use a **get-roles** command to list assigned, available, and effective realm roles for a user.

1. Specify the target user by user name or ID to list the user's **assigned** realm roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --username testuser
```

2. Use the **--effective** option to list **effective** realm roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --username testuser --effective
```

3. Use the **--available** option to list realm roles that you can add to a user.

For example:

```
$ kcadm.sh get-roles -r demorealm --username testuser --available
```

Listing assigned, available, and effective client roles for a user

Use a **get-roles** command to list assigned, available, and effective client roles for a user.

1. Specify the target user by user name (**--username** option) or ID (**--uid** option) and client by a clientId attribute (**--clientid** option) or an ID (**--cid** option) to list **assigned** client roles for the user.

For example:

```
$ kcadm.sh get-roles -r demorealm --username testuser --cclientid realm-management
```

2. Use the **--effective** option to list **effective** realm roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --username testuser --cclientid realm-management --effective
```

3. Use the **--available** option to list realm roles that you can add to a user.

For example:

```
$ kcadm.sh get-roles -r demorealm --username testuser --cclientid realm-management --available
```

Adding realm roles to a user

Use an **add-roles** command to add realm roles to a user.

Use the following example to add the **user** role to user **testuser**:

```
$ kcadm.sh add-roles --username testuser --rolename user -r demorealm
```

Removing realm roles from a user

Use a **remove-roles** command to remove realm roles from a user.

Use the following example to remove the **user** role from the user **testuser**:

```
$ kcadm.sh remove-roles --username testuser --rolename user -r demorealm
```

Adding client roles to a user

Use an **add-roles** command to add client roles to a user.

Use the following example to add two roles defined on the client **realm management**, the **create-client** role and the **view-users** role, to the user **testuser**.

```
$ kcadm.sh add-roles -r demorealm --username testuser --cclientid realm-management --rolename create-client --rolename view-users
```

Removing client roles from a user

Use a **remove-roles** command to remove client roles from a user.

Use the following example to remove two roles defined on the realm management client:

```
$ kcadm.sh remove-roles -r demorealm --username testuser --cclientid realm-management --rolename create-client --rolename view-users
```

Listing a user's sessions

1. Identify the user's ID,
2. Use the ID to compose an endpoint URI, such as **users/ID/sessions**.

- Use the **get** command to retrieve a list of the user's sessions.
For example:

```
$kcadm get users/6da5ab89-3397-4205-afaa-e201ff638f9e/sessions
```

Logging out a user from a specific session

- Determine the session's ID as described earlier.
- Use the session's ID to compose an endpoint URI, such as **sessions/ID**.
- Use the **delete** command to invalidate the session.
For example:

```
$ kcadm.sh delete sessions/d0eaa7cc-8c5d-489d-811a-69d3c4ec84d1
```

Logging out a user from all sessions

Use the user's ID to construct an endpoint URI, such as **users/ID/logout**.

Use the **create** command to perform **POST** on that endpoint URI.

For example:

```
$ kcadm.sh create users/6da5ab89-3397-4205-afaa-e201ff638f9e/logout -r demorealm -s realm=demorealm -s user=6da5ab89-3397-4205-afaa-e201ff638f9e
```

18.10. GROUP OPERATIONS

Creating a group

Use the **create** command on the **groups** endpoint to create a new group.

For example:

```
$ kcadm.sh create groups -r demorealm -s name=Group
```

Listing groups

Use the **get** command on the **groups** endpoint to list groups.

For example:

```
$ kcadm.sh get groups -r demorealm
```

Getting a specific group

Use the group's ID to construct an endpoint URI, such as **groups/GROUP_ID**.

For example:

```
$ kcadm.sh get groups/51204821-0580-46db-8f2d-27106c6b5ded -r demorealm
```

Updating a group

Use the **update** command with the same endpoint URI that you use to get a specific group.

For example:

```
$ kcadm.sh update groups/51204821-0580-46db-8f2d-27106c6b5ded -s 'attributes.email=
["group@example.com"]' -r demorealm
```

Deleting a group

Use the **delete** command with the same endpoint URI that you use to get a specific group.

For example:

```
$ kcadm.sh delete groups/51204821-0580-46db-8f2d-27106c6b5ded -r demorealm
```

Creating a subgroup

Find the ID of the parent group by listing groups. Use that ID to construct an endpoint URI, such as **groups/GROUP_ID/children**.

For example:

```
$ kcadm.sh create groups/51204821-0580-46db-8f2d-27106c6b5ded/children -r demorealm -s
name=SubGroup
```

Moving a group under another group

1. Find the ID of an existing parent group and the ID of an existing child group.
2. Use the parent group's ID to construct an endpoint URI, such as **groups/PARENT_GROUP_ID/children**.
3. Run the **create** command on this endpoint and pass the child group's ID as a JSON body.

For example:

```
$ kcadm.sh create groups/51204821-0580-46db-8f2d-27106c6b5ded/children -r demorealm -s
id=08d410c6-d585-4059-bb07-54dcb92c5094 -s name=SubGroup
```

Get groups for a specific user

Use a user's ID to determine a user's membership in groups to compose an endpoint URI, such as **users/USER_ID/groups**.

For example:

```
$ kcadm.sh get users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups -r demorealm
```

Adding a user to a group

Use the **update** command with an endpoint URI composed of a user's ID and a group's ID, such as **users/USER_ID/groups/GROUP_ID**, to add a user to a group.

For example:

```
$ kcadm.sh update users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups/ce01117a-7426-4670-
a29a-5c118056fe20 -r demorealm -s realm=demorealm -s userId=b544f379-5fc4-49e5-8a8d-
5cfb71f46f53 -s groupId=ce01117a-7426-4670-a29a-5c118056fe20 -n
```

Removing a user from a group

Use the **delete** command on the same endpoint URI you use for adding a user to a group, such as **users/USER_ID/groups/GROUP_ID**, to remove a user from a group.

For example:

```
$ kcadm.sh delete users/b544f379-5fc4-49e5-8a8d-5cfb71f46f53/groups/ce01117a-7426-4670-a29a-5c118056fe20 -r demorealm
```

Listing assigned, available, and effective realm roles for a group

Use a dedicated **get-roles** command to list assigned, available, and effective realm roles for a group.

1. Specify the target group by name (**--gname** option), path (**--gpath** option), or ID (**--gid** option) to list **assigned** realm roles for the group.

For example:

```
$ kcadm.sh get-roles -r demorealm --gname Group
```

2. Use the **--effective** option to list **effective** realm roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --gname Group --effective
```

3. Use the **--available** option to list realm roles that you can add to the group.

For example:

```
$ kcadm.sh get-roles -r demorealm --gname Group --available
```

Listing assigned, available, and effective client roles for a group

Use the **get-roles** command to list assigned, available, and effective client roles for a group.

1. Specify the target group by name (**--gname** option) or ID (**--gid** option),
2. Specify the client by the clientId attribute (**--cclientid** option) or ID (**--id** option) to list **assigned** client roles for the user.

For example:

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management
```

3. Use the **--effective** option to list **effective** realm roles.

For example:

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management --effective
```

4. Use the **--available** option to list realm roles that you can still add to the group.

For example:

```
$ kcadm.sh get-roles -r demorealm --gname Group --cclientid realm-management --available
```

18.11. IDENTITY PROVIDER OPERATIONS

Listing available identity providers

Use the **serverinfo** endpoint to list available identity providers.

For example:

```
$ kcadm.sh get serverinfo -r demorealm --fields 'identityProviders(*)'
```



NOTE

Red Hat Single Sign-On processes the **serverinfo** endpoint similarly to the **realms** endpoint. Red Hat Single Sign-On does not resolve the endpoint relative to a target realm because it exists outside any specific realm.

Listing configured identity providers

Use the **identity-provider/instances** endpoint.

For example:

```
$ kcadm.sh get identity-provider/instances -r demorealm --fields alias,providerId,enabled
```

Getting a specific configured identity provider

Use the identity provider's **alias** attribute to construct an endpoint URI, such as **identity-provider/instances/ALIAS**, to get a specific identity provider.

For example:

```
$ kcadm.sh get identity-provider/instances/facebook -r demorealm
```

Removing a specific configured identity provider

Use the **delete** command with the same endpoint URI that you use to get a specific configured identity provider to remove a specific configured identity provider.

For example:

```
$ kcadm.sh delete identity-provider/instances/facebook -r demorealm
```

Configuring a Keycloak OpenID Connect identity provider

1. Use **keycloak-oidc** as the **providerId** when you create a new identity provider instance.
2. Provide the **config** attributes: **authorizationUrl**, **tokenUrl**, **clientId**, and **clientSecret**.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=keycloak-oidc -s
providerId=keycloak-oidc -s enabled=true -s 'config.useJwksUrl="true"' -s
config.authorizationUrl=http://localhost:8180/auth/realms/demorealm/protocol/openid-
connect/auth -s
config.tokenUrl=http://localhost:8180/auth/realms/demorealm/protocol/openid-connect/token -
s config.clientId=demo-oidc-provider -s config.clientSecret=secret
```

Configuring an OpenID Connect identity provider

Configure the generic OpenID Connect provider the same way you configure the Keycloak OpenID Connect provider, except you set the **providerId** attribute value to **oidc**.

Configuring a SAML 2 identity provider

1. Use **saml** as the **providerId**.

2. Provide the **config** attributes: **singleSignOnServiceUrl**, **nameIDPolicyFormat**, and **signatureAlgorithm**.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=saml -s providerId=saml -s
enabled=true -s 'config.useJwksUrl="true"' -s
config.singleSignOnServiceUrl=http://localhost:8180/auth/realms/saml-broker-realm/protocol/saml -s
config.nameIDPolicyFormat=urn:oasis:names:tc:SAML:2.0:nameid-format:persistent -s
config.signatureAlgorithm=RSA_SHA256
```

Configuring a Facebook identity provider

1. Use **facebook** as the **providerId**.
2. Provide the **config** attributes: **clientId** and **clientSecret**. You can find these attributes in the Facebook Developers application configuration page for your application. See the [facebook identity broker](#) page for more information.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=facebook -s
providerId=facebook -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=FACEBOOK_CLIENT_ID -s
config.clientSecret=FACEBOOK_CLIENT_SECRET
```

Configuring a Google identity provider

1. Use **google** as the **providerId**.
2. Provide the **config** attributes: **clientId** and **clientSecret**. You can find these attributes in the Google Developers application configuration page for your application. See the [Google identity broker](#) page for more information.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=google -s
providerId=google -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=GOOGLE_CLIENT_ID -s config.clientSecret=GOOGLE_CLIENT_SECRET
```

Configuring a Twitter identity provider

1. Use **twitter** as the **providerId**.
2. Provide the **config** attributes **clientId** and **clientSecret**. You can find these attributes in the Twitter Application Management application configuration page for your application. See the [Twitter identity broker](#) page for more information.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=google -s
providerId=google -s enabled=true -s 'config.useJwksUrl="true"' -s
config.clientId=TWITTER_API_KEY -s config.clientSecret=TWITTER_API_SECRET
```

Configuring a GitHub identity provider

1. Use **github** as the **providerId**.

2. Provide the **config** attributes **clientId** and **clientSecret**. You can find these attributes in the GitHub Developer Application Settings page for your application. See the [Github identity broker](#) page for more information.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=github -s  
providerId=github -s enabled=true -s 'config.useJwksUrl="true"' -s  
config.clientId=GITHUB_CLIENT_ID -s config.clientSecret=GITHUB_CLIENT_SECRET
```

Configuring a LinkedIn identity provider

1. Use **linkedin** as the **providerId**.
2. Provide the **config** attributes **clientId** and **clientSecret**. You can find these attributes in the LinkedIn Developer Console application page for your application. See the [LinkedIn identity broker](#) page for more information.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=linkedin -s  
providerId=linkedin -s enabled=true -s 'config.useJwksUrl="true"' -s  
config.clientId=LINKEDIN_CLIENT_ID -s config.clientSecret=LINKEDIN_CLIENT_SECRET
```

Configuring a Microsoft Live identity provider

1. Use **microsoft** as the **providerId**.
2. Provide the **config** attributes **clientId** and **clientSecret**. You can find these attributes in the Microsoft Application Registration Portal page for your application. See the [Microsoft identity broker](#) page for more information.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=microsoft -s  
providerId=microsoft -s enabled=true -s 'config.useJwksUrl="true"' -s  
config.clientId=MICROSOFT_APP_ID -s config.clientSecret=MICROSOFT_PASSWORD
```

Configuring a Stack Overflow identity provider

1. Use **stackoverflow** command as the **providerId**.
2. Provide the **config** attributes **clientId**, **clientSecret**, and **key**. You can find these attributes in the Stack Apps OAuth page for your application. See the [Stack Overflow identity broker](#) page for more information.

For example:

```
$ kcadm.sh create identity-provider/instances -r demorealm -s alias=stackoverflow -s  
providerId=stackoverflow -s enabled=true -s 'config.useJwksUrl="true"' -s  
config.clientId=STACKAPPS_CLIENT_ID -s  
config.clientSecret=STACKAPPS_CLIENT_SECRET -s config.key=STACKAPPS_KEY
```

18.12. STORAGE PROVIDER OPERATIONS

Configuring a Kerberos storage provider

1. Use the **create** command against the **components** endpoint.

2. Specify the realm id as a value of the **parentId** attribute.
3. Specify **kerberos** as the value of the **providerId** attribute, and **org.keycloak.storage.UserStorageProvider** as the value of the **providerType** attribute.
4. For example:

```
$ kcadm.sh create components -r demorealm -s parentId=demorealmId -s id=demokerberos
-s name=demokerberos -s providerId=kerberos -s
providerType=org.keycloak.storage.UserStorageProvider -s 'config.priority=["0"]' -s
'config.debug=["false"]' -s 'config.allowPasswordAuthentication=["true"]' -s 'config.editMode=
["UNSYNCED"]' -s 'config.updateProfileFirstLogin=["true"]' -s
'config.allowKerberosAuthentication=["true"]' -s 'config.kerberosRealm=["KEYCLOAK.ORG"]'
-s 'config.keyTab=["http.keytab"]' -s 'config.serverPrincipal=
["HTTP/localhost@KEYCLOAK.ORG"]' -s 'config.cachePolicy=["DEFAULT"]'
```

Configuring an LDAP user storage provider

1. Use the **create** command against the **components** endpoint.
2. Specify **ldap** as the value of the **providerId** attribute, and **org.keycloak.storage.UserStorageProvider** as the value of the **providerType** attribute.
3. Provide the realm ID as the value of the **parentId** attribute.
4. Use the following example to create a Kerberos-integrated LDAP provider.

```
$ kcadm.sh create components -r demorealm -s name=kerberos-ldap-provider -s
providerId=ldap -s providerType=org.keycloak.storage.UserStorageProvider -s
parentId=3d9c572b-8f33-483f-98a6-8bb421667867 -s 'config.priority=["1"]' -s
'config.fullSyncPeriod=["-1"]' -s 'config.changedSyncPeriod=["-1"]' -s 'config.cachePolicy=
["DEFAULT"]' -s 'config.evictionDay=[]' -s 'config.evictionHour=[]' -s 'config.evictionMinute=[]' -s
'config.maxLifespan=[]' -s 'config.batchSizeForSync=["1000"]' -s 'config.editMode=
["WRITABLE"]' -s 'config.syncRegistrations=["false"]' -s 'config.vendor=["other"]' -s
'config.usernameLDAPAttribute=["uid"]' -s 'config.rdnLDAPAttribute=["uid"]' -s
'config.uuidLDAPAttribute=["entryUUID"]' -s 'config.userObjectClasses=["inetOrgPerson,
organizationalPerson"]' -s 'config.connectionUrl=["ldap://localhost:10389"]' -s
'config.usersDn=["ou=People,dc=keycloak,dc=org"]' -s 'config.authType=["simple"]' -s
'config.bindDn=["uid=admin,ou=system"]' -s 'config.bindCredential=["secret"]' -s
'config.searchScope=["1"]' -s 'config.useTruststoreSpi=["ldapsOnly"]' -s
'config.connectionPooling=["true"]' -s 'config.pagination=["true"]' -s
'config.allowKerberosAuthentication=["true"]' -s 'config.serverPrincipal=
["HTTP/localhost@KEYCLOAK.ORG"]' -s 'config.keyTab=["http.keytab"]' -s
'config.kerberosRealm=["KEYCLOAK.ORG"]' -s 'config.debug=["true"]' -s
'config.useKerberosForPasswordAuthentication=["true"]'
```

Removing a user storage provider instance

1. Use the storage provider instance's **id** attribute to compose an endpoint URI, such as **components/ID**.
2. Run the **delete** command against this endpoint.
For example:

```
$ kcadm.sh delete components/3d9c572b-8f33-483f-98a6-8bb421667867 -r demorealm
```

Triggering synchronization of all users for a specific user storage provider

1. Use the storage provider's **id** attribute to compose an endpoint URI, such as **user-storage/ID_OF_USER_STORAGE_INSTANCE/sync**.
2. Add the **action=triggerFullSync** query parameter.
3. Run the **create** command.
For example:

```
$ kcadm.sh create user-storage/b7c63d02-b62a-4fc1-977c-947d6a09e1ea/sync?
action=triggerFullSync
```

Triggering synchronization of changed users for a specific user storage provider

1. Use the storage provider's **id** attribute to compose an endpoint URI, such as **user-storage/ID_OF_USER_STORAGE_INSTANCE/sync**.
2. Add the **action=triggerChangedUsersSync** query parameter.
3. Run the **create** command.
For example:

```
$ kcadm.sh create user-storage/b7c63d02-b62a-4fc1-977c-947d6a09e1ea/sync?
action=triggerChangedUsersSync
```

Test LDAP user storage connectivity

1. Run the **get** command on the **testLDAPConnection** endpoint.
2. Provide query parameters **bindCredential**, **bindDn**, **connectionUrl**, and **useTruststoreSpi**.
3. Set the **action** query parameter to **testConnection**.
For example:

```
$ kcadm.sh create testLDAPConnection -s action=testConnection -s bindCredential=secret -s
bindDn=uid=admin,ou=system -s connectionUrl=ldap://localhost:10389 -s
useTruststoreSpi=ldapsOnly
```

Test LDAP user storage authentication

1. Run the **get** command on the **testLDAPConnection** endpoint.
2. Provide the query parameters **bindCredential**, **bindDn**, **connectionUrl**, and **useTruststoreSpi**.
3. Set the **action** query parameter to **testAuthentication**.
For example:

```
$ kcadm.sh create testLDAPConnection -s action=testAuthentication -s
bindCredential=secret -s bindDn=uid=admin,ou=system -s
connectionUrl=ldap://localhost:10389 -s useTruststoreSpi=ldapsOnly
```

18.13. ADDING MAPPERS

Adding a hard-coded role LDAP mapper

1. Run the **create** command on the **components** endpoint.
2. Set the **providerType** attribute to **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**.
3. Set the **parentId** attribute to the ID of the LDAP provider instance.
4. Set the **providerId** attribute to **hardcoded-ldap-role-mapper**. Ensure you provide a value of **role** configuration parameter.

For example:

```
$ kcadm.sh create components -r demorealm -s name=hardcoded-ldap-role-mapper -s
providerId=hardcoded-ldap-role-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config.role=["realm-
management.create-client"]'
```

Adding an MS Active Directory mapper

1. Run the **create** command on the **components** endpoint.
2. Set the **providerType** attribute to **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**.
3. Set the **parentId** attribute to the ID of the LDAP provider instance.
4. Set the **providerId** attribute to **msad-user-account-control-mapper**.

For example:

```
$ kcadm.sh create components -r demorealm -s name=msad-user-account-control-mapper -
s providerId=msad-user-account-control-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea
```

Adding a user attribute LDAP mapper

1. Run the **create** command on the **components** endpoint.
2. Set the **providerType** attribute to **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**.
3. Set the **parentId** attribute to the ID of the LDAP provider instance.
4. Set the **providerId** attribute to **user-attribute-ldap-mapper**.

For example:

```
$ kcadm.sh create components -r demorealm -s name=user-attribute-ldap-mapper -s
providerId=user-attribute-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."user.model.attribute"=
["email"]' -s 'config."ldap.attribute"=["mail"]' -s 'config."read.only"=["false"]' -s
'config."always.read.value.from.ldap"=["false"]' -s 'config."is.mandatory.in.ldap"=["false"]'
```

Adding a group LDAP mapper

1. Run the **create** command on the **components** endpoint.
2. Set the **providerType** attribute to **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**.
3. Set the **parentId** attribute to the ID of the LDAP provider instance.
4. Set the **providerId** attribute to **group-ldap-mapper**.
For example:

```
$ kcadm.sh create components -r demorealm -s name=group-ldap-mapper -s
providerId=group-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."groups.dn"=[]' -s
'config."group.name.ldap.attribute"=["cn"]' -s 'config."group.object.classes"=
["groupOfNames"]' -s 'config."preserve.group.inheritance"=["true"]' -s
'config."membership.ldap.attribute"=["member"]' -s 'config."membership.attribute.type"=
["DN"]' -s 'config."groups.ldap.filter"=[]' -s 'config.mode=["LDAP_ONLY"]' -s
'config."user.roles.retrieve.strategy"=["LOAD_GROUPS_BY_MEMBER_ATTRIBUTE"]' -s
'config."mapped.group.attributes"=["admins-group"]' -s
'config."drop.non.existing.groups.during.sync"=["false"]' -s 'config.roles=["admins"]' -s
'config.groups=["admins-group"]' -s 'config.group=[]' -s 'config.preserve=["true"]' -s
'config.membership=["member"]'
```

Adding a full name LDAP mapper

1. Run the **create** command on the **components** endpoint.
2. Set the **providerType** attribute to **org.keycloak.storage.ldap.mappers.LDAPStorageMapper**.
3. Set the **parentId** attribute to the ID of the LDAP provider instance.
4. Set the **providerId** attribute to **full-name-ldap-mapper**.
For example:

```
$ kcadm.sh create components -r demorealm -s name=full-name-ldap-mapper -s
providerId=full-name-ldap-mapper -s
providerType=org.keycloak.storage.ldap.mappers.LDAPStorageMapper -s
parentId=b7c63d02-b62a-4fc1-977c-947d6a09e1ea -s 'config."ldap.full.name.attribute"=
["cn"]' -s 'config."read.only"=["false"]' -s 'config."write.only"=["true"]'
```

18.14. AUTHENTICATION OPERATIONS

Setting a password policy

1. Set the realm's **passwordPolicy** attribute to an enumeration expression that includes the specific policy provider ID and optional configuration.
2. Use the following example to set a password policy to default values. The default values include:
 - 27,500 hashing iterations

- at least one special character
- at least one uppercase character
- at least one digit character
- not be equal to a user's **username**
- be at least eight characters long

```
$ kcadm.sh update realms/demorealm -s 'passwordPolicy="hashIterations and specialChars and upperCase and digits and notUsername and length"'
```

3. To use values different from defaults, pass the configuration in brackets.
4. Use the following example to set a password policy to:

- 25,000 hash iterations
- at least two special characters
- at least two uppercase characters
- at least two lowercase characters
- at least two digits
- be at least nine characters long
- not be equal to a user's **username**
- not repeat for at least four changes back

```
$ kcadm.sh update realms/demorealm -s 'passwordPolicy="hashIterations(25000) and specialChars(2) and upperCase(2) and lowerCase(2) and digits(2) and length(9) and notUsername and passwordHistory(4)'"
```

Obtaining the current password policy

You can get the current realm configuration by filtering all output except for the **passwordPolicy** attribute.

For example, display **passwordPolicy** for **demorealm**.

```
$ kcadm.sh get realms/demorealm --fields passwordPolicy
```

Listing authentication flows

Run the **get** command on the **authentication/flows** endpoint.

For example:

```
$ kcadm.sh get authentication/flows -r demorealm
```

Getting a specific authentication flow

Run the **get** command on the **authentication/flows/FLOW_ID** endpoint.

For example:

```
$ kcadm.sh get authentication/flows/febfd772-e1a1-42fb-b8ae-00c0566fafb8 -r demorealm
```

Listing executions for a flow

Run the **get** command on the **authentication/flows/FLOW_ALIAS/executions** endpoint.

For example:

```
$ kcadm.sh get authentication/flows/Copy%20of%20browser/executions -r demorealm
```

Adding configuration to an execution

1. Get execution for a flow.
2. Note the ID of the flow.
3. Run the **create** command on the **authentication/executions/{executionId}/config** endpoint.

For example:

```
$ kcadm create "authentication/executions/a3147129-c402-4760-86d9-3f2345e401c7/config" -r  
examplerealm -b '{"config":{"x509-cert-auth.mapping-source-selection":"Match SubjectDN using  
regular expression","x509-cert-auth.regular-expression":"(?:.*?)(?:$)","x509-cert-auth.mapper-  
selection":"Custom Attribute Mapper","x509-cert-auth.mapper-selection.user-attribute-  
name":"usercertificate","x509-cert-auth.crl-checking-enabled":"","x509-cert-auth.crl-dp-checking-  
enabled":false,"x509-cert-auth.crl-relative-path":"crl.pem","x509-cert-auth.ocsp-checking-  
enabled":"","x509-cert-auth.ocsp-responder-uri":"","x509-cert-auth.keyusage":"","x509-cert-  
auth.extendedkeyusage":"","x509-cert-auth.confirmation-page-  
disallowed":"","alias":"my_otp_config"}'
```

Getting configuration for an execution

1. Get execution for a flow.
2. Note its **authenticationConfig** attribute, which contains the config ID.
3. Run the **get** command on the **authentication/config/ID** endpoint.

For example:

```
$ kcadm get "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r examplerealm
```

Updating configuration for an execution

1. Get the execution for the flow.
2. Get the flow's **authenticationConfig** attribute.
3. Note the config ID from the attribute.
4. Run the **update** command on the **authentication/config/ID** endpoint.

For example:

```
$ kcadm update "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r exemplerealm -
b '{"id":"dd91611a-d25c-421a-87e2-227c18421833","alias":"my_otp_config","config":{"x509-cert-
auth.extendedkeyusage":"","x509-cert-auth.mapper-selection.user-attribute-
name":"usercertificate","x509-cert-auth.ocsp-responder-uri":"","x509-cert-auth.regular-expression":"
(. *?)(?:$)","x509-cert-auth.crl-checking-enabled":"true","x509-cert-auth.confirmation-page-
disallowed":"","x509-cert-auth.keyusage":"","x509-cert-auth.mapper-selection":"Custom Attribute
Mapper","x509-cert-auth.crl-relative-path":"crl.pem","x509-cert-auth.crl-dp-checking-
enabled":"false","x509-cert-auth.mapping-source-selection":"Match SubjectDN using regular
expression","x509-cert-auth.ocsp-checking-enabled":""}}'
```

Deleting configuration for an execution

1. Get execution for a flow.
2. Get the flows **authenticationConfig** attribute.
3. Note the config ID from the attribute.
4. Run the **delete** command on the **authentication/config/ID** endpoint.

For example:

```
$ kcadm delete "authentication/config/dd91611a-d25c-421a-87e2-227c18421833" -r exemplerealm
```