10° ROW – Online, June 8th, 2021

Integrating .it RDAP Server with OpenID Connect through Keycloak: experiences and expectations

Francesco Donini, Mario Loffredo, Maurizio Martinelli
IIT-CNR/Registro.it

Keycloak

- Is an open source identity and access manager
- Makes it easy to secure applications and services with little to no code
- Enables SSO
- Supports login with social networks (e.g. Google, Twitter, Facebook)
- Can authenticate users with existing OpenID Connect or SAML 2.0 IdPs
- Provides built-in support to sync to existing LDAP or Active Directory servers but you can create custom extensions for any user database (e.g. a relational db)

Keycloak – key concepts

- **Realm**: a Keycloak space where you manage objects
- **User**: a user for the service to secure
- **Role**: a type or category of user
- **Client**: the service to secure
- A **Realm** can include more **Clients** and all the **Users** having the same **Roles** with respect to the **Clients**

Keycloak – access control

- Keycloak supports fine-grained authorization policies by combining different access control mechanisms (**ACM**):
  - **Role-based:** defines conditions for permissions where one or multiple roles are permitted to access an object;
  - **User-based:** defines conditions for permissions where one or multiple users are permitted to access an object;
  - **Attribute-based:** defines conditions for permissions based on an attribute obtained from the execution context or the current identity (through `policy-enforcement`);
  - **Time-based:** defines time restrictions on permissions.

Keycloak - adapters

- When securing clients and services, you need to specify:
  - the protocol (OpenID Connect – SAML)
  - the software platform (Java or other)

OpenID Connect – Java	SAML - Java
• Jboss EAP	• Jboss EAP
• WildFly	• WildFly
• Fuse	• Tomcat
• Tomcat	• Jetty
• Jetty 9	
• Servlet Filter	
• Spring Boot	
• Spring Security	

Keycloak – realm configuration

- **realm**: `name` of the realm;
- **resource**: the `client-id` of the application;
- **auth-server-url**: the base `URL` of the Keycloak server;
- **ssl-required:** ensures that all communication to and from the Keycloak server is over HTTPS (allowed values: `all`, `none`, (default) `external`);
- **use-resource-role-mappings:** if set to `true`, the adapter will look inside the token for application level role mappings for the user. If `false`, it will look at the realm level for user role mappings (default: `false`);
- **bearer-only:** if set to `true`, the adapter will not attempt to authenticate users, but only verify bearer tokens (default: `false`);
- **verify-token-audience**: if set to `true`, then during authentication with the bearer token, the adapter will verify whether the token contains this client name (resource) as an audience (default: `false`).

Why Keycloak? (1)

- Supports `OpenID`
- Free
- Followed by a big community of developers
- Allows for delegating all the authentication and authorization aspects (e.g. forgotten password handling, tokens management)
- Supports multiple IdPs

Why Keycloak? (2)

- Offers a comprehensive web-based GUI to set up configurations
- Provides Admin REST API
- Easily customizable and extensible
- Provides easy integration with WildFly and SpringBoot based applications
- Supports many ACMs

RDAP-OpenID at .it

- Keycloak (acting as an **OpenID Provider**)
- The .it RDAP *server* (acting as an **OpenID Relying Party**)
- The .it RDAP *client* (acting as an **OpenID End-User**)

.it RDAP OpenID – implementation constraints

- **General**:
  - same endpoints for **protected and unprotected** resources
  - need for an **ad-hoc web client** to improve the user interaction with the server
- **.it specific**:
  - **Java-Wildfly** based implementation;
  - different server platforms managed through **Docker** (i.e. *devel*, *public test*, *live*)
  - **different request** and **response** features according to the user profiles:
    - anonymous
    - authenticated: Registrar, Registry, other (e.g. authority)

.it RDAP OpenID – Keycloak Java adapters

- **WildFly Adapter**
  - Realm configuration is included in `standalone.xml`
  - Roles-Resources mapping is defined in `web.xml`
  - No `policy-enforcement`
  - No multi-realm
  - Jar dependencies to access some security objects
- **Servlet Filter Adapter**
  - Realm configuration and `policy-enforcement` is included in `keycloak.json`
  - Roles-Resources mapping is defined in `web.xml`
  - Multi-realm allowed through different `keycloak.json` files
  - Jar dependencies
- **SpringBoot Adapter**
  - Configuration is all included in a normal SpringBoot configuration file:
    - Realm configuration
    - Roles-Resources mapping
    - `policy-enforcement`
    - Multi-realm

.it RDAP OpenID – adapter selection

- **WildFly Adapter**:
  - lowest implementation effort to integrate with Keycloak;
  - installation made by a Dockerfile inside the server project;
  - minimal configuration;
  - set up of WildFly `standalone.xml` guided by platform-related `jboss-cli` scripts.

.it RDAP OpenID – realm configuration example

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
   <secure-deployment name="rdap-server.war">
      <realm>rdap</realm>
      <resource>rdap-server</resource>
      <auth-server-url>http://auth.pubtest.nic.it/auth/</auth-server-url>
      <use-resource-role-mappings>true</use-resource-role-mappings>
      <bearer-only>true</bearer-only>
      <ssl-required>none</ssl-required>
      <verify-token-audience>true</verify-token-audience>
   </secure-deployment>
</subsystem>
```

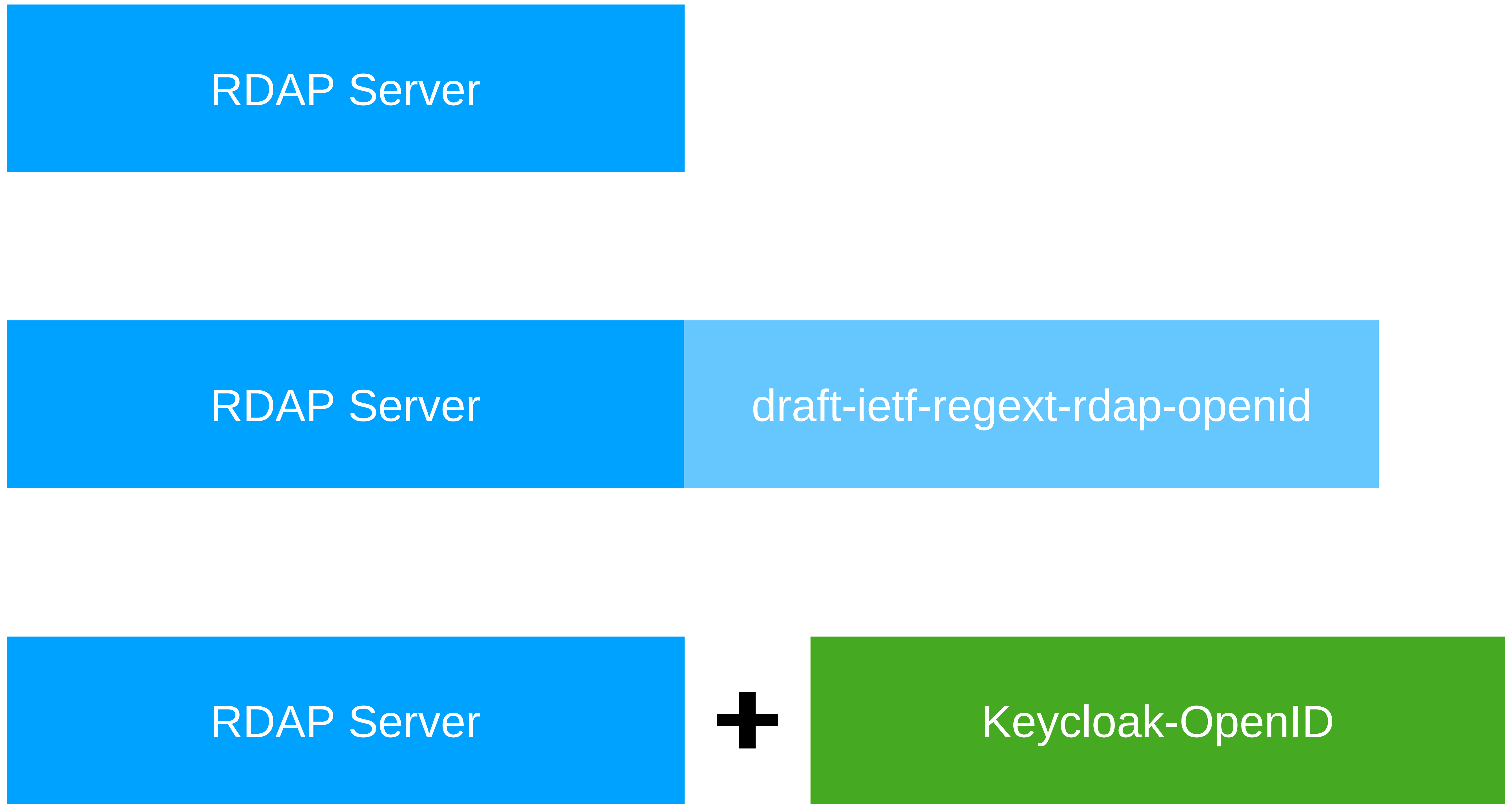
.it RDAP OpenID – users vs. roles mapping

- Created one ad-hoc realm: `rdap`
  - Unable to use existing .it realms having different categories of users
- Request and response features based only on roles
- Four roles defined: `ANONYMOUS`, `AUTH_REGISTRAR`, `AUTH_REGISTRY`, `AUTH_USER`
- All endpoints are considered protected (every access is mediated by Keycloak);
- An anonymous user is authenticated through publicly known credentials.

.it RDAP OpenID – roles vs. resources mapping

Roles	Resources
`ANONYMOUS`	Allowed to access `/domain` and `/help` `/domain`: unpublic data are either redacted or not returned (WHOIS-like response) `/help`: provides information about the allowed features
`AUTH_REGISTRAR`	Allowed to access every endpoint and every data about the sponsored objects are returned (inner filter)
`AUTH_REGISTRY`	Allowed to access every endpoint and every data are returned
`AUTH_USER`	The same as AUTH_REGISTRY but for a limited time and able to submit a fixed request

draft-ietf-regext-rdap-openid vs. Keycloak-RDAP-OpenID



draft-ietf-regext-rdap-openid vs. Keycloak-RDAP-OpenID

draft-ietf-regext-rdap-openid	Keycloak-RDAP-OpenID
• OpenID compliant • Requires the RDAP server to manage: • IdP discovery • end user authorization • tokens • Requires `rdap_openid_level_0` conformance: • new requests and responses implementation • handling specialized claims for RDAP: • `purpose` • `dnt` • Requires additional effort to support clients with limited user interfaces	• OpenID compliant • Delegates Keycloak to manage: • IdP discovery (as IdP itself or as a bridge to IdPs) • end user authorization • tokens • No `rdap_openid_level_0` conformance • no futher requests and responses to implement • specialized claims for RDAP: • `purpose:` redundant because role-dependent • `dnt`: not compliant with EU NIS (logging) • Leverages Keycloak to support OTP Access Token can be managed manually

Future activities

- Currently, `ANONYMOUS`, `AUTH_REGISTRY` and `AUTH_REGISTRAR` roles are supported
- To support `AUTH_USER` we need to use `policy-enforcement` to deal with more fine-grained ACMs:
  - ➢ changing the adapter and possible migration of the RDAP server from WildFly to SpringBoot;
  - ➢ implementing an additional service, interacting with Keycloak via Admin REST API, to provide temporary and query-based credentials:
    - ➢ providing specific OpenID claims

Thanks for your attention!

Q&A

Demo time