Keycloak Jump Start

Javaland 2025

Thomas Darimont | @thomasdarimont

Thomas Darimont

- **DI** Consultant | Identity Tailor GmbH
- **D**igital **I**dentities ❤️ Standards
- Open Source Enthusiast
- Spring Team Alumni
- Official Keycloak Maintainer
- OpenID Foundation Certification Team
- IDPro Certified
- Java User Group Saarland Organizer

@thomasdarimont
thomas@identity-tailor.de

Bewertung des Schulungstages 2025

Breaks

- 10:15-10:45 Morning Coffee-break
- 12:45-13:30 Lunch break
- 14:30-15:00 Afternoon Coffee-break

Agenda

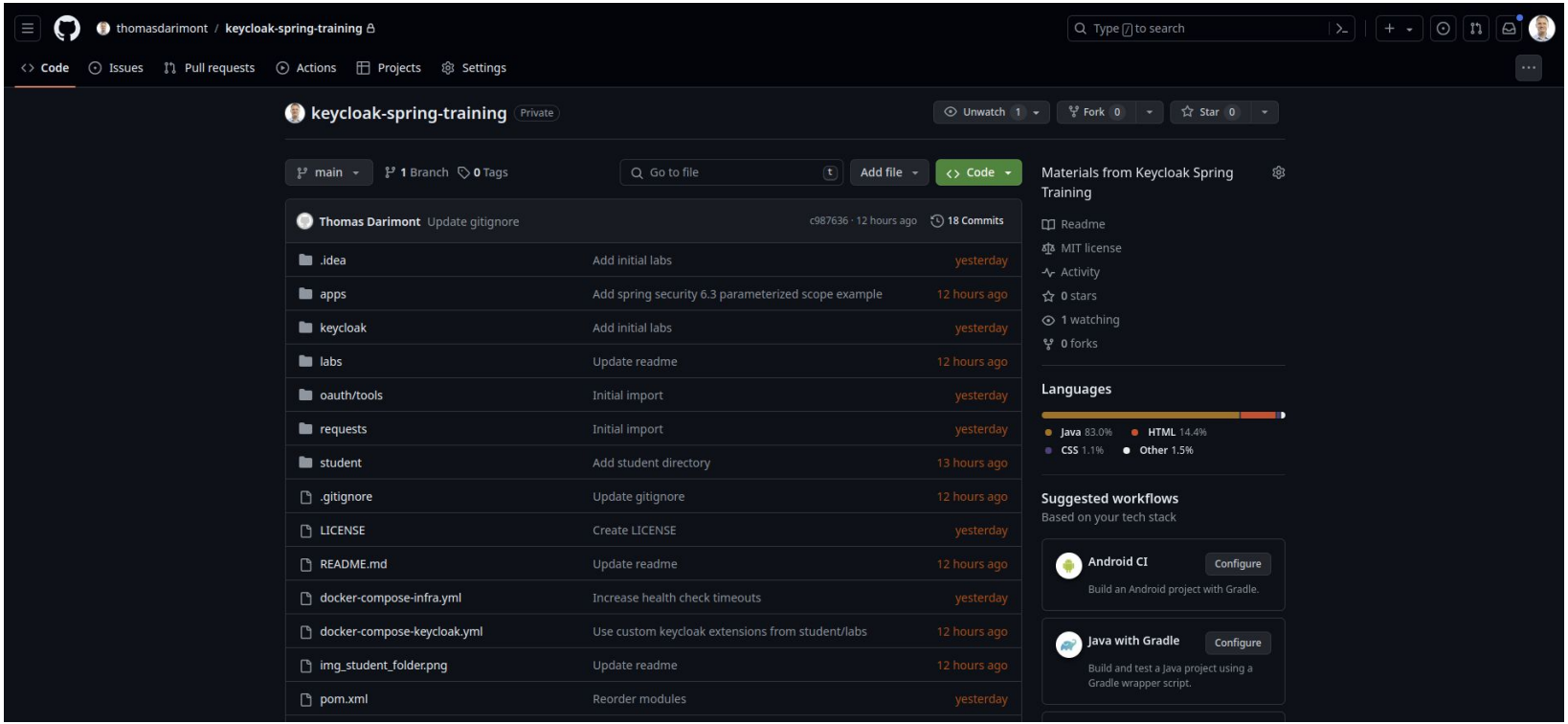
Part 1: Keycloak Overview

Part 2: OAuth / OpenID Connect

Part 3: Securing Spring Apps

Part 4: Extending Keycloak

Workshop Repository



Workshop Repository

github.com/thomasdarimont/keycloak-spring-training

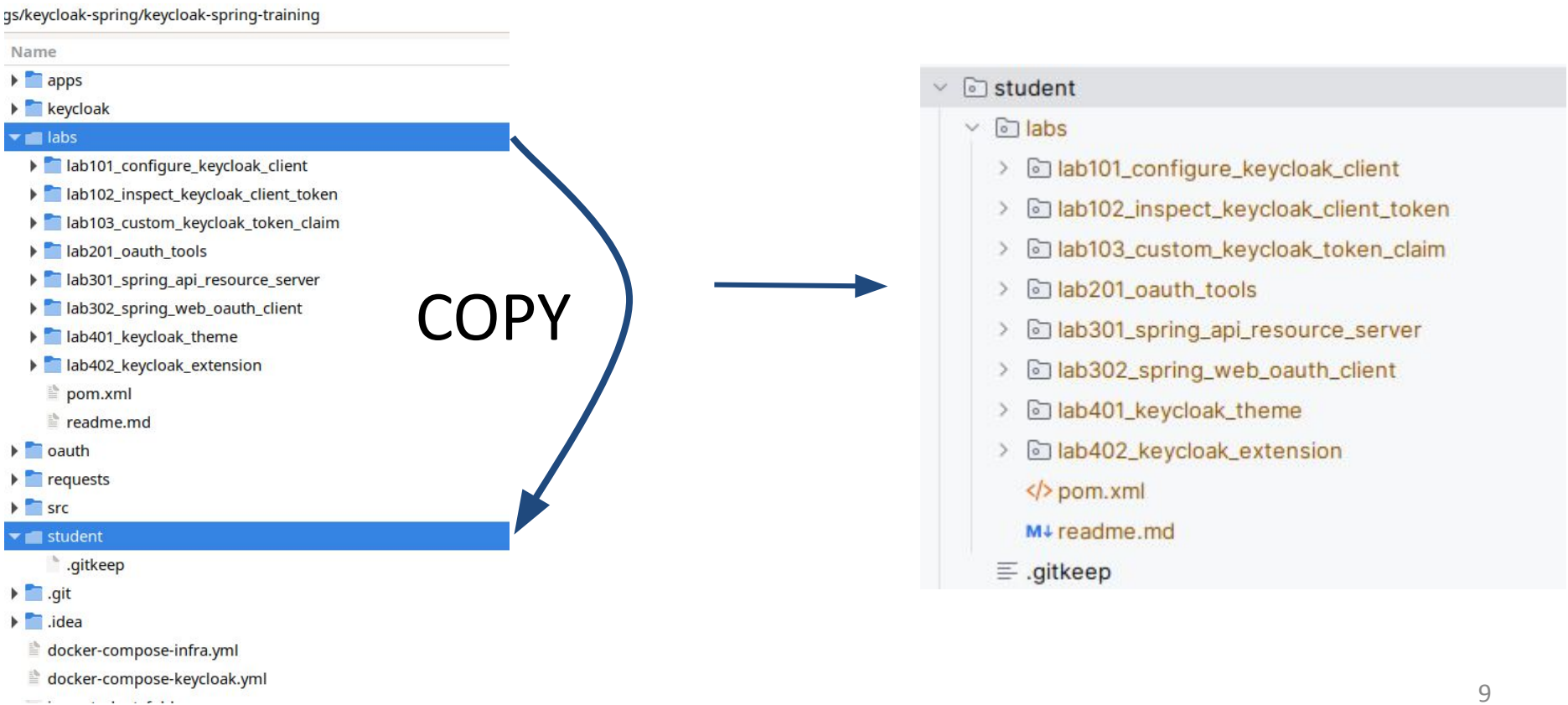
Run the following in a terminal

1) `git clone https://github.com/thomasdarimont/keycloak-spring-training.git`

2) `mvn verify`

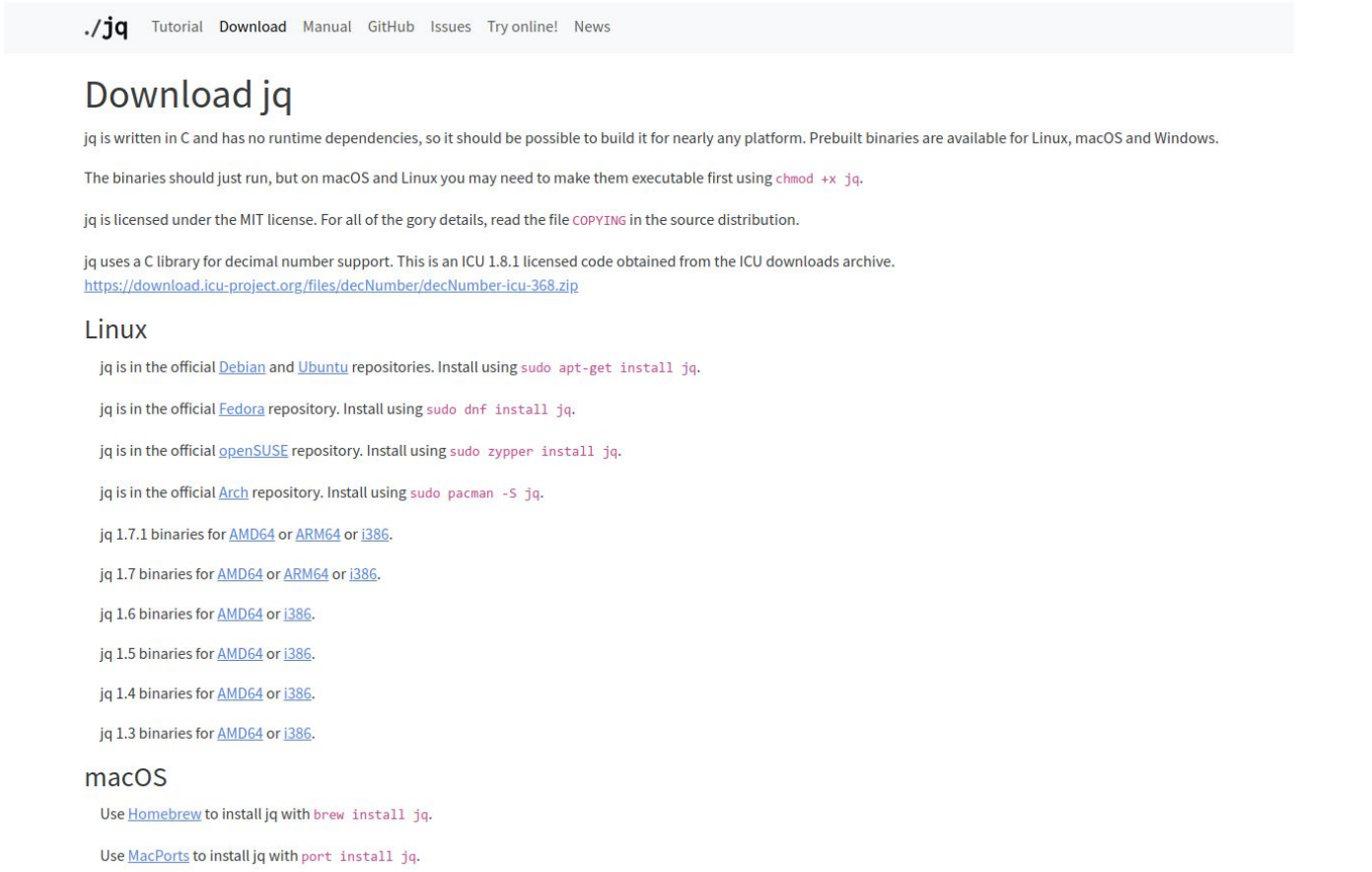
```
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO]
[INFO] keycloak-spring-training 1.0-SNAPSHOT ..... SUCCESS [  0.001 s]
[INFO] extensions 1.0-SNAPSHOT ..................... SUCCESS [  0.623 s]
[INFO] cli 0.0.1-SNAPSHOT .......................... SUCCESS [  1.425 s]
[INFO] api 0.0.1-SNAPSHOT .......................... SUCCESS [  1.841 s]
[INFO] web 0.0.1-SNAPSHOT .......................... SUCCESS [  0.033 s]
[INFO] bff 0.0.1-SNAPSHOT .......................... SUCCESS [  0.031 s]
[INFO] gateway 0.0.1-SNAPSHOT ...................... SUCCESS [  0.041 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.151 s
[INFO] Finished at: 2025-04-02T20:18:16+02:00
[INFO] ------------------------------------------------------------------------
```

Copy `labs` Folder to `student` Folder



Install `jq` Command-Line Tool

https://jqlang.github.io/jq/



./jq Tutorial Download Manual GitHub Issues Try online! News

Download jq

jq is written in C and has no runtime dependencies, so it should be possible to build it for nearly any platform. Prebuilt binaries are available for Linux, macOS and Windows.

The binaries should just run, but on macOS and Linux you may need to make them executable first using `chmod +x jq`.

jq is licensed under the MIT license. For all of the gory details, read the file `COPYING` in the source distribution.

jq uses a C library for decimal number support. This is an ICU 1.8.1 licensed code obtained from the ICU downloads archive.
https://download.icu-project.org/files/decNumber/decNumber-icu-368.zip

Linux

jq is in the official Debian and Ubuntu repositories. Install using `sudo apt-get install jq`.

jq is in the official Fedora repository. Install using `sudo dnf install jq`.

jq is in the official openSUSE repository. Install using `sudo zypper install jq`.

jq is in the official Arch repository. Install using `sudo pacman -S jq`.

jq 1.7.1 binaries for AMD64 or ARM64 or i386.

jq 1.7 binaries for AMD64 or ARM64 or i386.

jq 1.6 binaries for AMD64 or i386.

jq 1.5 binaries for AMD64 or i386.

jq 1.4 binaries for AMD64 or i386.

jq 1.3 binaries for AMD64 or i386.

macOS

Use Homebrew to install jq with `brew install jq`.

Use MacPorts to install jq with `port install jq`.

Part 1: Keycloak Overview

The Journey


Keycloak


Single Sign-on


Securing Applications


Selected Topics

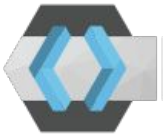
Identity & Access Management

Identity Management (IdM)

- User Management
- Lifecycle Management
- Credential Management
- Account Recovery
- User Federation

Access Management (AM)

- Policy Administration
- Entitlements Management
- Provisioning / Auditing
- Single Sign-on (SSO)
- Authentication (AuthN)
- Authorization (AuthZ)



Keycloak Overview

KEYCLOAK GitHub Stars 26k Guides Docs Downloads Community Blog

https://www.keycloak.org

Open Source Identity and Access Management

Add authentication to applications and secure services with minimum effort.
No need to deal with storing users or authenticating users.

Keycloak provides user federation, strong authentication, user management, fine-grained authorization, and more.

Get Started Download

Latest release 26.1.4

News

27 Mar Register now for KubeCon Japan in June

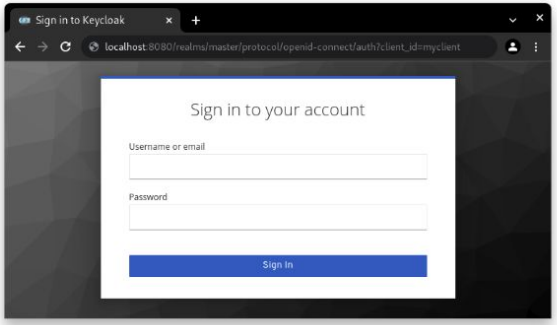
17 Mar Submit to KeyConf25 Japan Call-for-Papers!

14 Mar Keycloak 26.1.4 released

Single-Sign On

Users authenticate with Keycloak rather than individual applications. This means that your applications don't have to deal with login forms, authenticating users, and storing users. Once logged-in to Keycloak, users don't have to login again to access a different application.

This also applies to logout. Keycloak provides single-sign out, which means users only have to logout once to be logged-out of all applications that use Keycloak.



Project

- Java based **Authentication** & **Authorization** Server
- Started in 2013, broad adoption since 2015
- Apache License, **Red Hat** Developers
- Vital **Community** with **~1000+** Contributors
- CNCF incubating project since April 2023
- Free Version **Keycloak "Community"** (current 26.1.4)
- Commercial Offering **Red Hat build of Keycloak** (26.x)
- Very **robust**, good **Documentation**, many **Examples**

Features

- **Single Sign-on** and Single Sign-out
- **Standard Protocols** OAuth 2.0, OIDC 1.0, SAML 2.0, Kerberos
- Flexible **Authentication** and **Authorization**
- **Multi-Factor Authentication** OTP, WebAuthN, Passkeys
- **Social Login** Google, Facebook, Apple,...
- **Identity Federation** EntraID, ADFS, Auth0, Okta,...
- Provides centralized **User Management**
- Supports **Directory Services**
- **Customizable** and **Extensible**
- **Easy** Setup and Integration

Main Concepts

KEYCLOAK

Realm

Container - Central Structuring Element

Users / Groups

Accounts / Credentials / User Profile / Organizational Structures

Clients

Web App, SPAs, Web API, Services, CLI Apps, Mobile App, Identity Provider Clients

General Settings

Login Options, User Handling, Password Policies, Protocols, Sessions, Timeouts, Email, Realm

UI Customizations

Themes (Login, Account, Email, Admin), Localization

Authentication

Authentication Flows, Registration Flow, Reset Credentials Flow, Flow Bindings, Policies, MFA, Required Actions

Authorization

Roles, Role-Hierarchies, Scopes, Policies, Permissions

User Storage

Local, Federated (LDAP / AD), Kerberos, Custom, Identity Provider, Social Login

Security

Web Security, Brute-force protection, Public / Private Keys, Secret Keys, Signing & Encryption

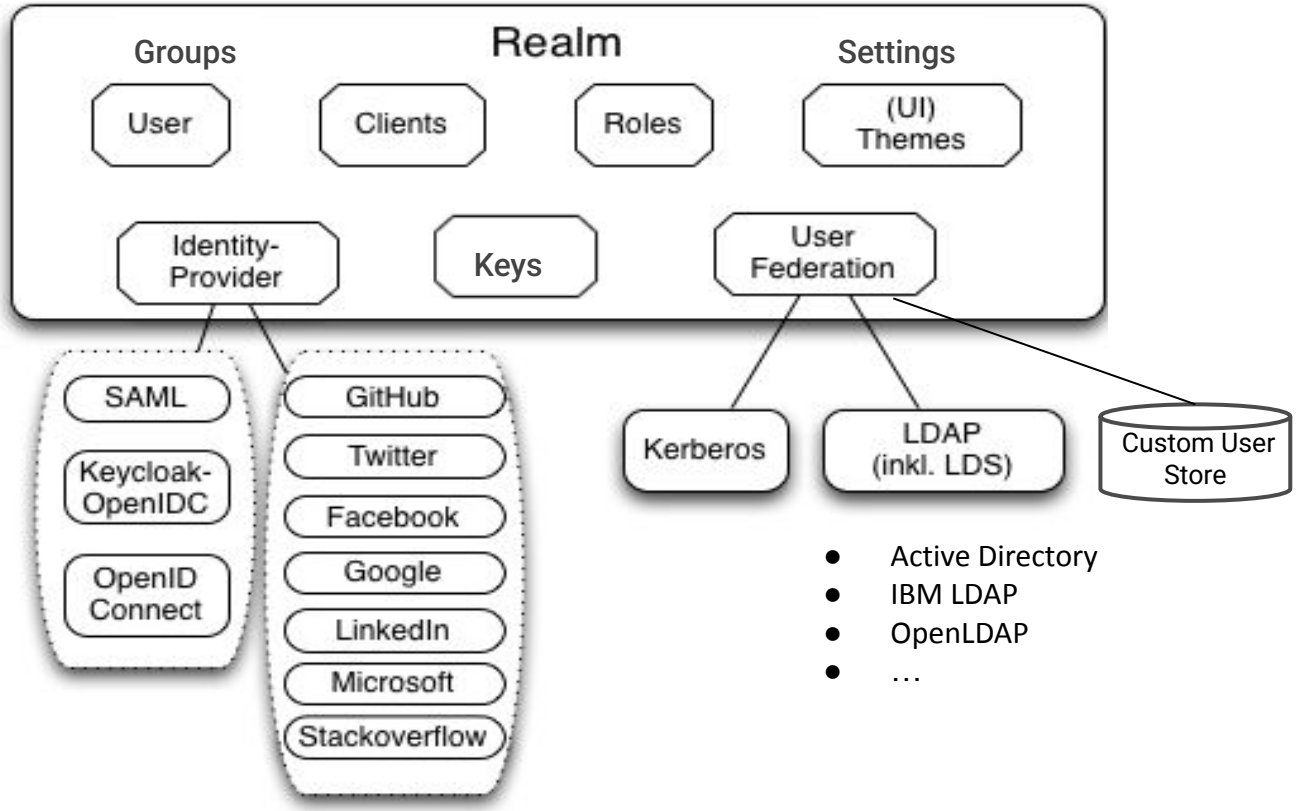
Keycloak Clients

- Client = Digital Product
- Client Examples
  - Classical server-side Web App (e.g. Spring Web MVC)
  - Single Page App (SPA)
  - Mobile App
  - Desktop / Native App
  - CLI App
  - Web API
  - Web API with Service Account
  - IoT Device, Smart TV, Infodisplay
  - Identity Provider Integration

Storage Federation vs. Identity Brokering

- Storage Federation
  - “User Federation” via Store
  - Access identities from other user stores through Keycloak
  - User stores: LDAP, Active Directory, Databases, REST
  - Allows to validate credentials against other user store
  - → Login via Keycloak Login Forms
- Identity Brokering
  - “User Federation” via Identity Provider
  - Delegate authentication to other Auth Service
  - via OIDC, SAML or built-in Social Login Providers
  - → Login via other Identity Provider

Main Concepts

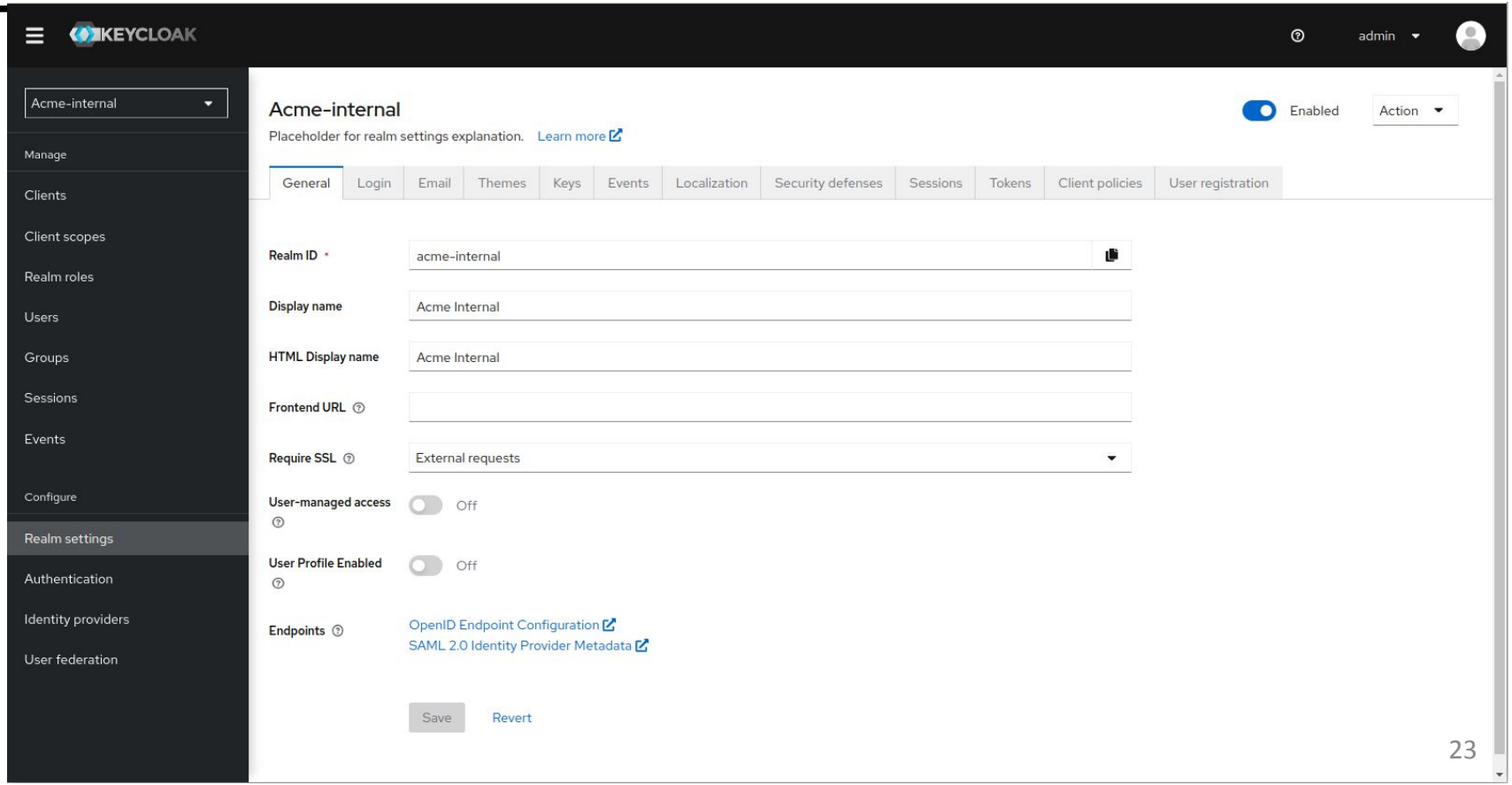


- ADFS
- Azure AD
- Google Workspace
- PingFederate
- Auth0
- Okta
- …

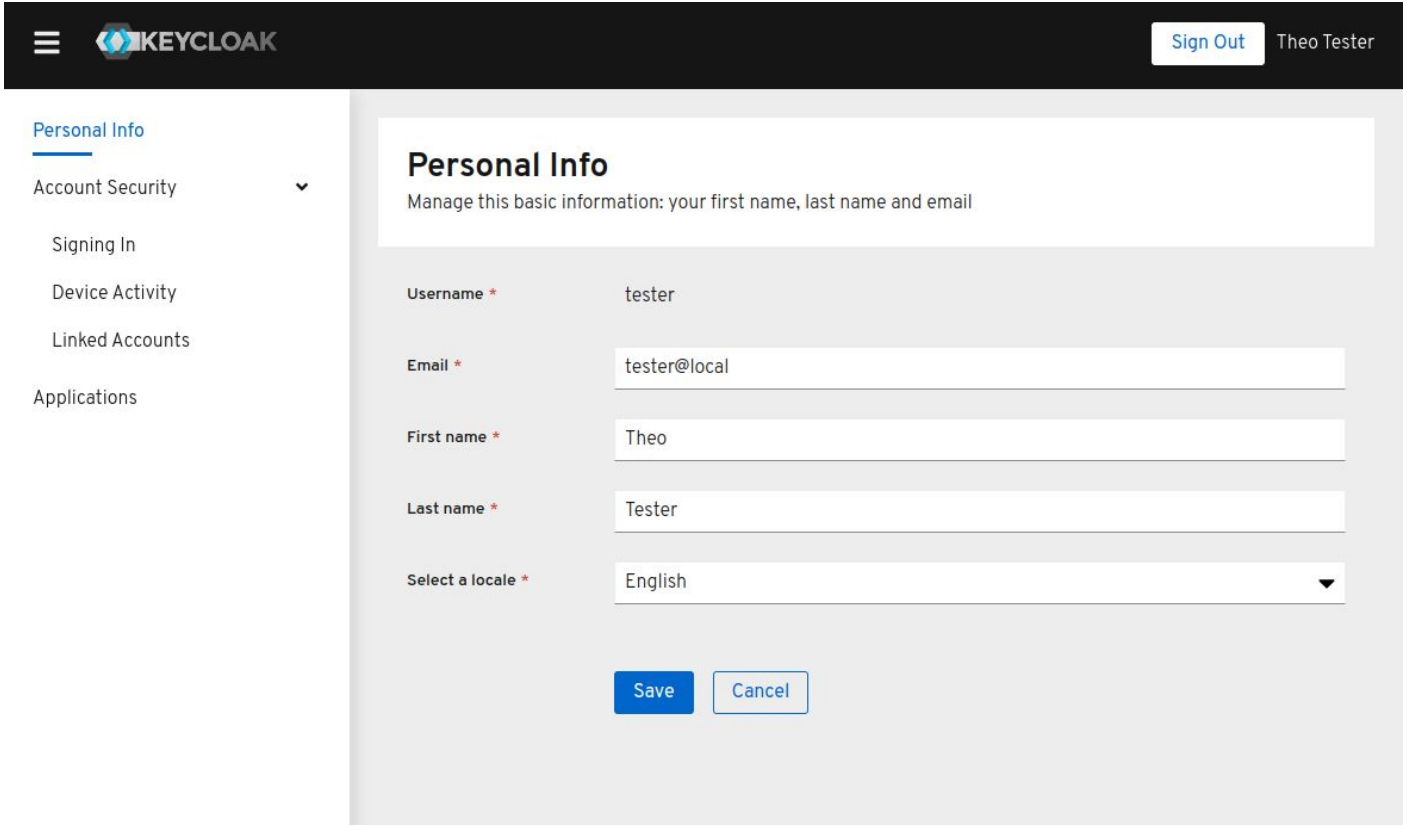
- Active Directory
- IBM LDAP
- OpenLDAP
- …

Admin Console

Admin UI


KEYCLOAK
admin
Acme-internal
Manage
Clients
Client scopes
Realm roles
Users
Groups
Sessions
Events
Configure
Realm settings
Authentication
Identity providers
User federation
Acme-internal
Placeholder for realm settings explanation.
Learn more
Enabled
Action
General
Login
Email
Themes
Keys
Events
Localization
Security defenses
Sessions
Tokens
Client policies
User registration
Realm ID *
acme-internal
Display name
Acme Internal
HTML Display name
Acme Internal
Frontend URL
Require SSL
External requests
User-managed access
Off
User Profile Enabled
Off
Endpoints
OpenID Endpoint Configuration
SAML 2.0 Identity Provider Metadata
Save
Revert

Account Console



Technology Stack

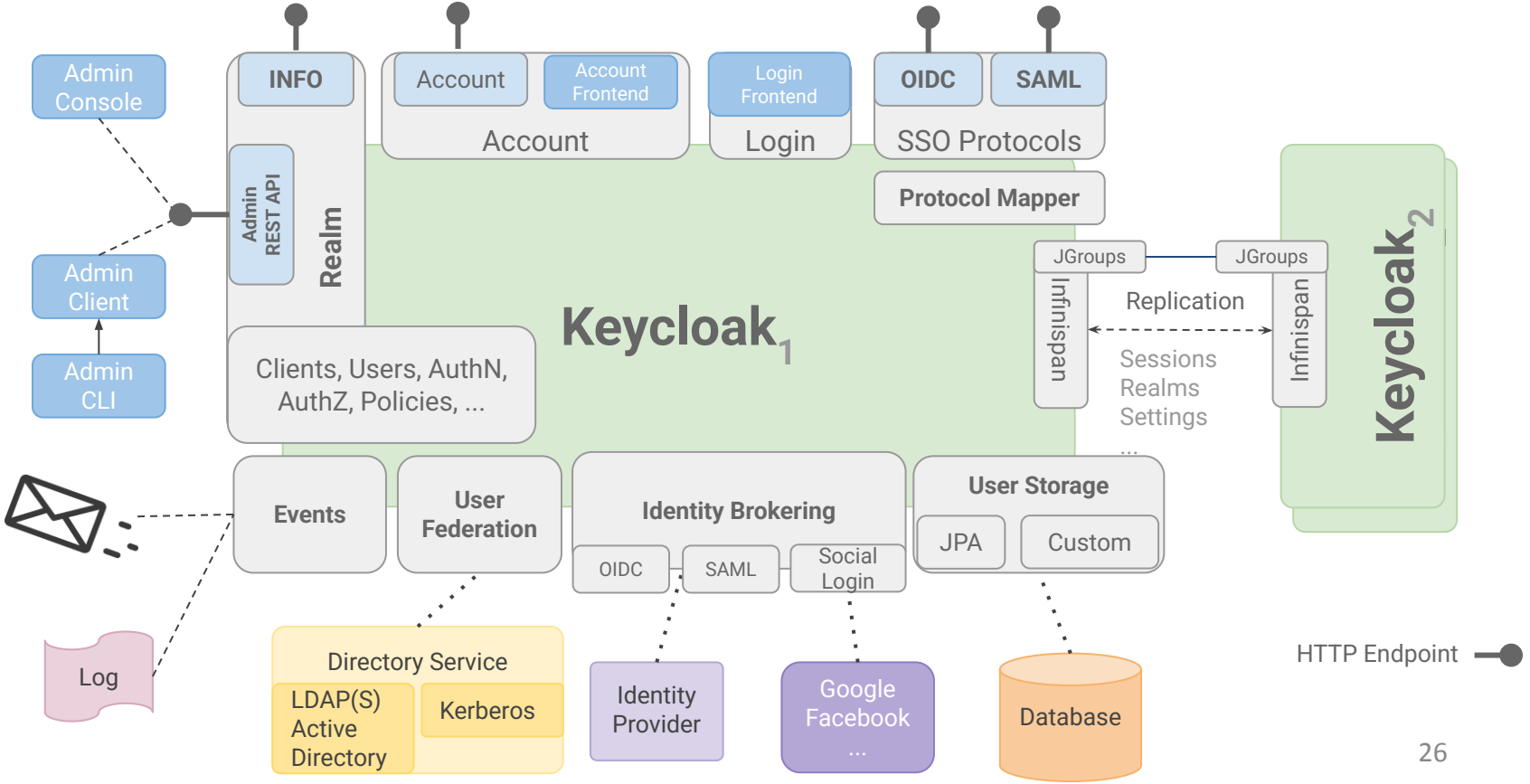
Keycloak Server

- **Quarkus / Vert.x**
- JAX-RS (Resteasy)
- JPA (Hibernate)
- Infinispan (JGroups)
- Freemarker
- Jackson 2.x
- Liquibase
- JBoss Logging
- Apache Directory API
- Commons HTTP Client

Admin / Account UI

- React
- PatternFly
- Bootstrap

Server Components



Keycloak Docker Quick Start

```
docker run -it \
  -p 8080:8080 \
  -p 9000:9000 \
  -e KC_BOOTSTRAP_ADMIN_USERNAME=admin \
  -e KC_BOOTSTRAP_ADMIN_PASSWORD=admin \
  quay.io/keycloak/keycloak:26.1.4 \
  start-dev
```

Browse to: http://localhost:8080/admin

More Infos: https://www.keycloak.org/server/containers

Single Sign-on with Keycloak

How it works

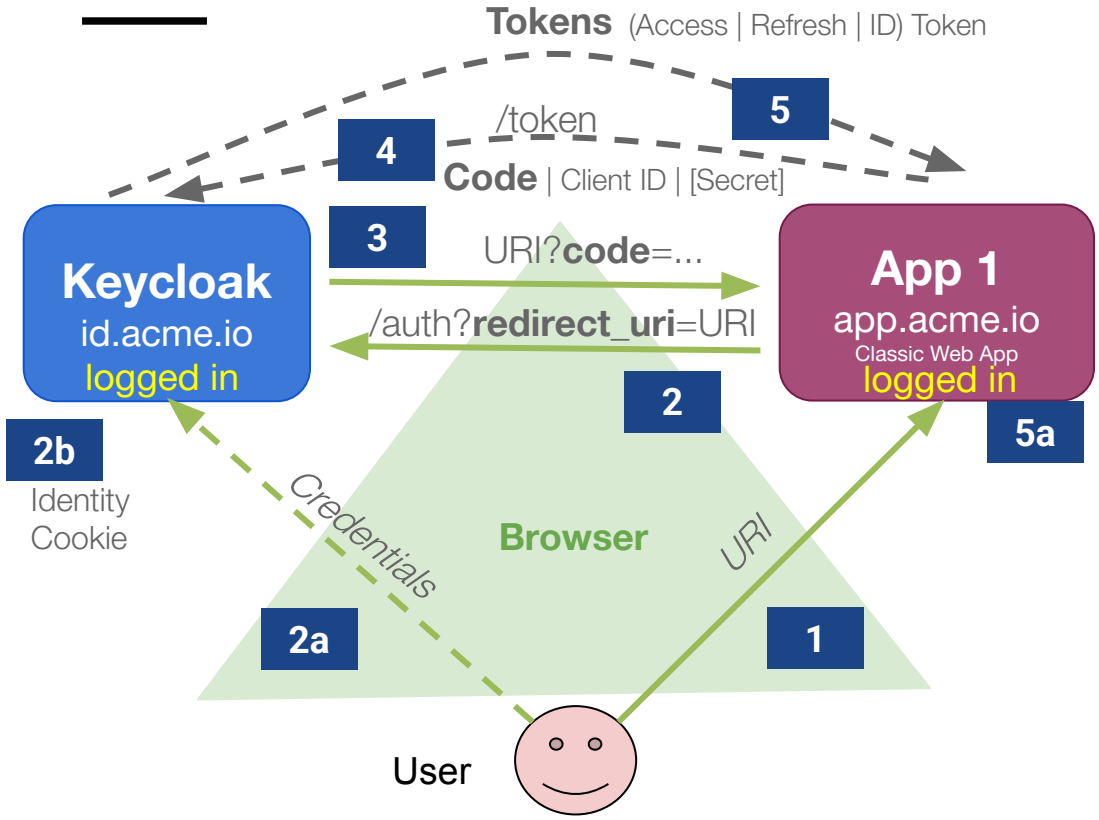
Single Sign-on

- **SSO** ⇒ Login **once** to access all applications
- **Standardized Protocols**
  - OpenID Connect 1.0 (OIDC)
  - Security Assertion Markup Language 2.0 (SAML)
  - Kerberos
- **Browser based “Web SSO”**
  - Web, Mobile and Desktop Apps
- Support for **Single Logout**
  - Logouts can be propagated to applications
  - Applications can opt-in

Supported Authentication Protocols

- **OIDC OpenID Connect 1.0**
  - Protocol based on OAuth 2.0
  - Leverages OAuth 2.0 tokens + *IDToken* to encode *Identity*
  - Tokens are encoded as JSON Web Tokens (JWT)
  - Requires secure channel HTTPS/TLS
- SAML 2.0 Security Assertion Markup Language
  - Mature standard & common in enterprise environments
  - XML based protocol
  - Uses XML signature and encryption
- Kerberos
  - Classic Windows Authentication

Web SSO with OIDC*: Unauthenticated User



1 Unauthenticated User **accesses** App

2 App **redirects** to Keycloak for Login

2a User **submits** *Credentials* to Keycloak

2b Credentials OK? → Keycloak **creates** *SSO Session and emits Cookies*

3 **Generates** *Code* and **redirects** User back to App

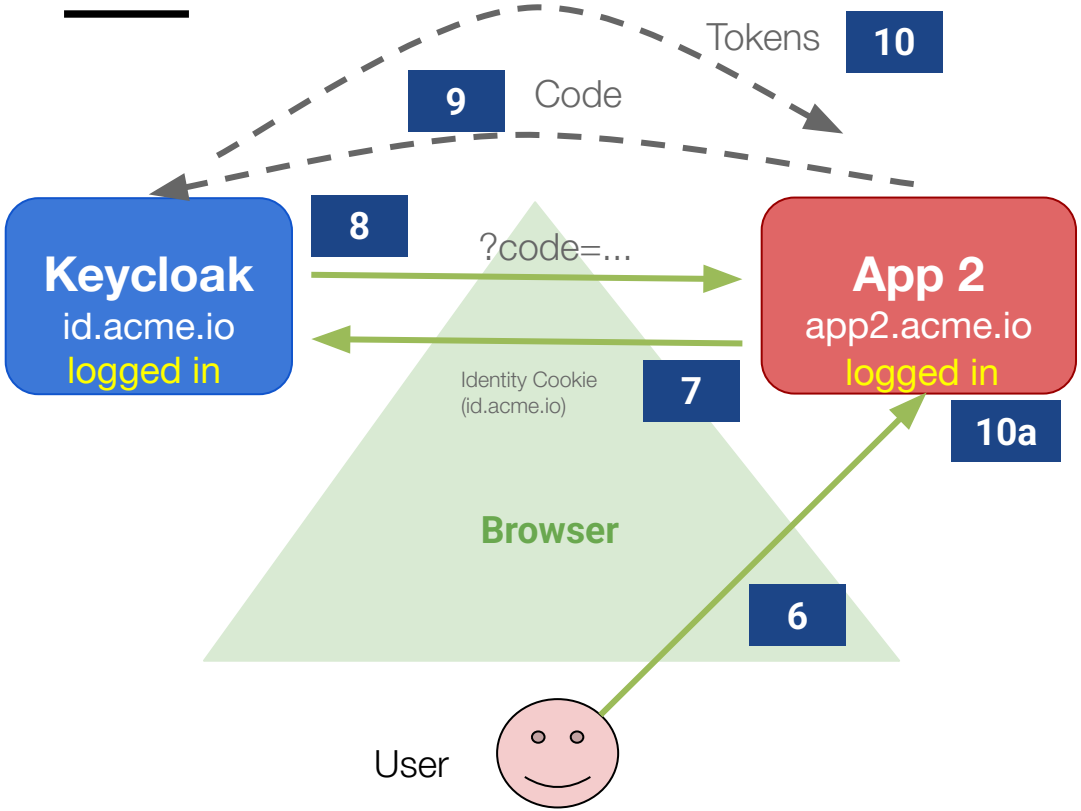
4 App **exchanges** *Code* to *Tokens* with Keycloak via separate Channel

5 App **verifies** received *Tokens* and associates it with a *session*

5a User is now *logged-in* to App

*) Authorization Code Grant Flow with OpenID Connect based on OAuth 2.0 without PKCE

Web SSO with OIDC: Authenticated User



...

6 Authenticated user **accesses** App 2

7 App 2 **redirects** user to Keycloak for login with Cookie

8 Keycloak **detects** *SSO Session*, **generates** *code*, **redirects** to App 2

9 App 2 **exchanges** *code* for *tokens* with Keycloak via separate channel

10 App 2 **verifies** *received tokens* and associates it with a *session*

10a User is now *logged-in* to App 2

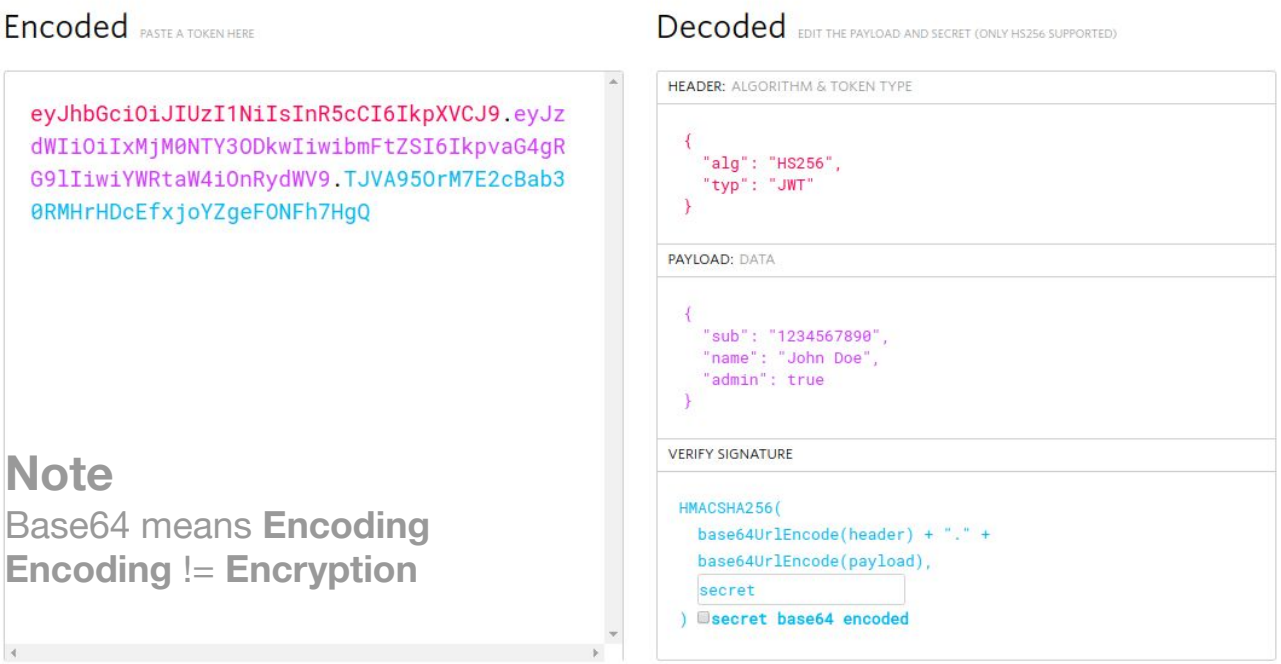
Keycloak Tokens

- OAuth / OpenID Connect
  - Signed self-contained **JSON Web Token**
  - **Claims**: KV-Pairs with User information + Metadata
  - Issued by Keycloak, **signed** with active Realm **Private Key**
  - **Verified** by Client with Realm **Public Key**
  - Limited lifespan, can be revoked

- Essential Token Types
  - **Access-Token** short-lived (Minutes+) → used for **accessing Resources**
  - **Refresh-Token** longer-lived (Hours+) → used for **requesting new Tokens**
    - **Offline-Token** long-lived (Days++) special "*Refresh-Token*" that "never" expires → Mobile Apps
  - **IDToken** → contains **User information** (OIDC)

JSON Web Tokens

`<header-base64Url>.<payload-base64Url>.<signature-base64Url>`



https://jwt.io

JSON Web Token Components

`<header-base64Url>.<payload-base64Url>.<signature-base64Url>`

- Header
  - Includes algorithm used to sign/encrypt the JWT
  - usually it is HMAC SHA-256 or HS256
- Payload
  - Contains claims which are key, value pairs of important details about the user
  - including information like token issuer, audience, subject, issued at and others
- Signature
  - Encoding of header and payload hashed with the algorithm referenced in header

Keycloak JSON Web Token Example

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2
lkIiA6ICJMT0Rxc1Q3NFRwMFJRcjlHSmVpSXJRVnNV
blZZQzk3eF9fZ0ttc0k1TE93In0.eyJqdGkiOiJiMG
IyMGRjYy0wNmRkLTRiMzgtYTUyOS00ZDhiODg2Njdh
YjIiLCJleHAiOjE0OTA2NTM3NDIsIm5iZiI6MCwiaW
F0IjoxNDkwNjUzNDQyLCJpc3MiOiJodHRwOi8vc3Nv
LnRkbGFicy5sb2NhbDo4ODk5L3UvYXV0aC9yZWFsbX
MvamF2YWxhbmQiLCJhdWQiOiJpZG0tY2xpZW50Iiwi
c3ViIjoiMjI0YjgzYWQtY2RkMi00NjY3LWFlODUtZW
EzZDhmZDNhNmFjIiwidHlwIjoiQmVhcmVyIiwiYXpw
IjoiaWRtLWNsaWVudCIsImF1dGhfdGltZSI6MCwic2
Vzc2lvbl9zdGF0ZSI6IjZmZDQ3MjNkLTQwYjItNGM4
Ny1iMzliLTk4YTA3N2ZmM2FkNCIsImFjciI6IjEiLC
JjbGllbnRfc2Vzc2lvbiI6IjM4Nzk5ZjgyLTBkNmMt
NDAyYy1hYmEwLTY3ZDI3NGVjZWIzMCIsImFsbG93ZW
Qtb3JpZ2lucyI6W10sInJlYWxtX2FjY2VzcyI6eyJy
b2xlcyI6WyJ1bWFfYXV0aG9yaXphdGlvbiIsInVzZX
IiXX0sInJlc291cmNlX2FjY2VzcyI6eyJhcHAtZ3Jl
ZXRpbmctc2VydmljZSI6eyJyb2xlcyI6WyJ1c2VyIl
19LCJkZW1vLXNlcnZpY2UiOnsicm9sZXMiOlsidXNl
ciJdfSwiYXBwLWphdmFlZS1wZXRjbGluaWMiOnsicm
9sZXMiOlsidXNlciJdfSwiYWNjb3VudCI6eyJyb2xl
cyI6WyJtYW5hZ2UtYWNjb3VudCIsInZpZXctcHJvZm
lsZSJdfSwiYXBwLWRlc2t0b3AiOnsicm9sZXMiOlsi
dXNlciJdfX0sIm5hbWUiOiJUaGVvIFRlc3RlciIsIn
ByZWZlcnJlZF91c2VybmFtZSI6InRlc3RlciIsImdp
```


Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE


```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "LODqsT74Tp0RQr9GJeiIrQVsUnVYC97x__gKmsI5LOw"
}
```


PAYLOAD: DATA


```
{
  "jti": "b0b20dcc-06dd-4b38-a529-4d8b88667ab2",
  "exp": 1490653742,
  "nbf": 0,
  "iat": 1490653442,
  "iss":
"http://sso.tdlabs.local:8899/u/auth/realms/javaland",
  "aud": "idm-client",
  "sub": "224b87ad-cdd2-4667-ae85-ea3d8fd3a6ac",
  "typ": "Bearer",
  "azp": "idm-client",
  "auth_time": 0,
  "session_state": "6fd4723d-40b2-4c87-b39b-98a077ff3ad4",
  "acr": "1",
  "client_session": "38799f82-0d6c-402c-aba0-67d274eceb30",
  "allowed-origins": [],
  "realm_access": {
    "roles": [
      "uma_authorization",
      "user"
    ]
  },
  "resource_access": {
    "app-greeting-service": {
      "roles": [
        "user"
      ]
```


Header

Claims

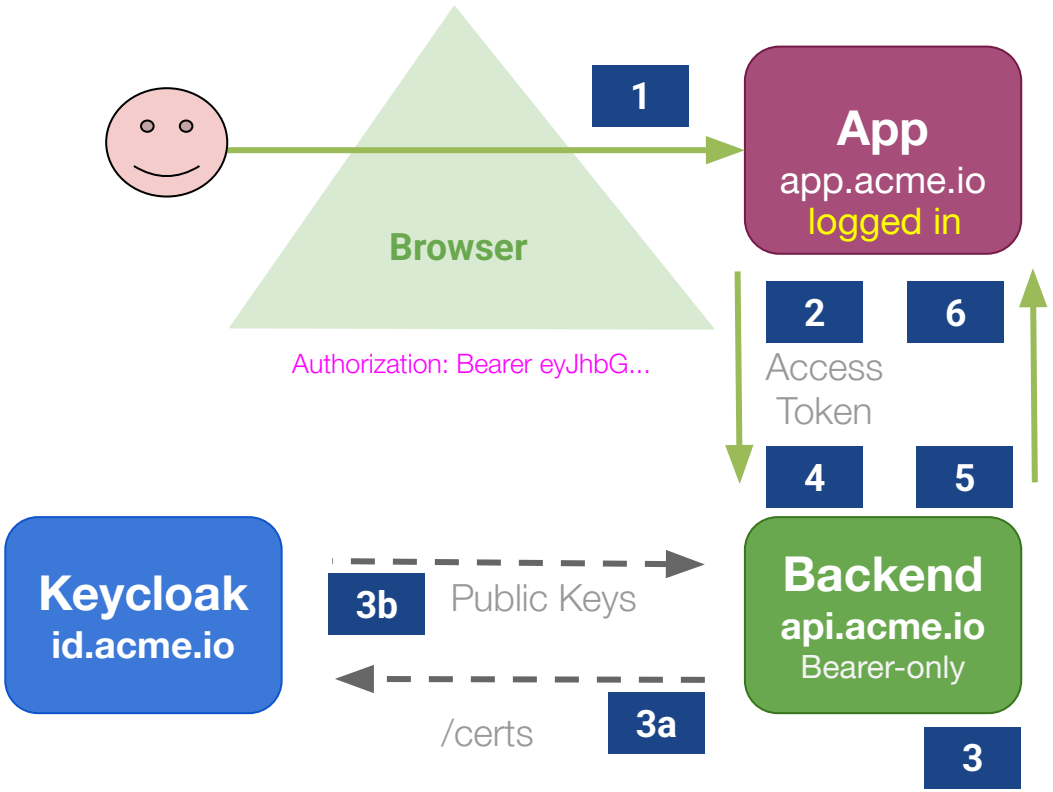
https://jwt.io

OpenID Connect Standard Claims

https://openid.net/specs/openid-connect-core-1_0.html#StandardClaims

Member	Type	Description
sub	string	Subject - Identifier for the End-User at the Issuer.
name	string	End-User's full name in displayable form including all name parts, possibly including titles and suffixes, ordered according to the End-User's locale and preferences.
given_name	string	Given name(s) or first name(s) of the End-User. Note that in some cultures, people can have multiple given names; all can be present, with the names being separated by space characters.
family_name	string	Surname(s) or last name(s) of the End-User. Note that in some cultures, people can have multiple family names or no family name; all can be present, with the names being separated by space characters.
middle_name	string	Middle name(s) of the End-User. Note that in some cultures, people can have multiple middle names; all can be present, with the names being separated by space characters. Also note that in some cultures, middle names are not used.
nickname	string	Casual name of the End-User that may or may not be the same as the `given_name`. For instance, a `nickname` value of `Mike` might be returned alongside a `given_name` value of `Michael`.
preferred_username	string	Shorthand name by which the End-User wishes to be referred to at the RP, such as `janedoe` or `j.doe`. This value MAY be any valid JSON string including special characters such as `@`, `/`, or whitespace. The RP MUST NOT rely upon this value being unique, as discussed in **Section 5.7**.
profile	string	URL of the End-User's profile page. The contents of this Web page SHOULD be about the End-User.
picture	string	URL of the End-User's profile picture. This URL MUST refer to an image file (for example, a PNG, JPEG, or GIF image file), rather than to a Web page containing an image. Note that this URL SHOULD specifically reference a profile photo of the End-User suitable for displaying when describing the End-User, rather than an arbitrary photo taken by the End-User.
website	string	URL of the End-User's Web page or blog. This Web page SHOULD contain information published by the End-User or an organization that the End-User is affiliated with.
email	string	End-User's preferred e-mail address. Its value MUST conform to the **RFC 5322** [RFC5322] addr-spec syntax. The RP MUST NOT rely upon this value being unique, as discussed in **Section 5.7**.
email_verified	boolean	True if the End-User's e-mail address has been verified; otherwise false. When this Claim Value is `true`, this means that the OP took affirmative steps to ensure that this e-mail address was controlled by the End-User at the time the verification was performed. The means by which an e-mail address is verified is context-specific, and dependent upon the trust framework or contractual agreements within which the parties are operating.
gender	string	End-User's gender. Values defined by this specification are `female` and `male`. Other values MAY be used when neither of the defined values are applicable.
birthdate	string	End-User's birthday, represented as an **ISO 8601:2004** [ISO8601-2004] `YYYY-MM-DD` format. The year MAY be `0000`, indicating that it is omitted. To represent only the year, `YYYY` format is allowed. Note that depending on the underlying platform's date related function, providing just year can result in varying month and day, so the implementers need to take this factor into account to correctly process the dates.
zoneinfo	string	String from zoneinfo **[zoneinfo]** time zone database representing the End-User's time zone. For example, `Europe/Paris` or `America/Los_Angeles`.
locale	string	End-User's locale, represented as a **BCP47** [RFC5646] language tag. This is typically an **ISO 639-1 Alpha-2** [ISO639-1] language code in lowercase and an **ISO 3166-1 Alpha-2** [ISO3166-1] country code in uppercase, separated by a dash. For example, `en-US` or `fr-CA`. As a compatibility note, some implementations have used an underscore as the separator rather than a dash, for example, `en_US`; Relying Parties MAY choose to accept this locale syntax as well.
phone_number	string	End-User's preferred telephone number. **E.164** [E.164] is RECOMMENDED as the format of this Claim, for example, `+1 (425) 555-1212` or `+56 (2) 687 2400`. If the phone number contains an extension, it is RECOMMENDED that the extension be represented using the **RFC 3966** [RFC3966] extension syntax, for example, `+1 (604) 555-1234;ext=5678`.
phone_number_verified	boolean	True if the End-User's phone number has been verified; otherwise false. When this Claim Value is `true`, this means that the OP took affirmative steps to ensure that this phone number was controlled by the End-User at the time the verification was performed. The means by which a phone number is verified is context-specific, and dependent upon the trust framework or contractual agreements within which the parties are operating. When true, the `phone_number` Claim MUST be in E.164 format and any extensions MUST be represented in RFC 3966 format.
address	JSON object	End-User's preferred postal address. The value of the `address` member is a JSON **[RFC4627]** structure containing some or all of the members defined in **Section 5.1.1**.
updated_at	number	Time the End-User's information was last updated. Its value is a JSON number representing the number of seconds from 1970-01-01T0:0:0Z as measured in UTC until the date/time.

Calling Backend Services with Access-Token



1. Authenticated User **accesses** App
2. App **uses** *Access-Token* in *HTTP Header* to access backend
3. Backend **looks-up** *Realm Public Key* in cache with in *Kid* from *JWT*
3a. *If not found*, **fetch** *Public Keys* from Keycloak's JWKS endpoint
3b. Keycloak **returns** *Realm Public Keys*
4. Backend **verifies** signature of *Access-Token* with *Realm Public Key*
5. Backend Service **grants** *access* and **returns** *user data*
6. App can now display user data

Keycloak Client Integrations

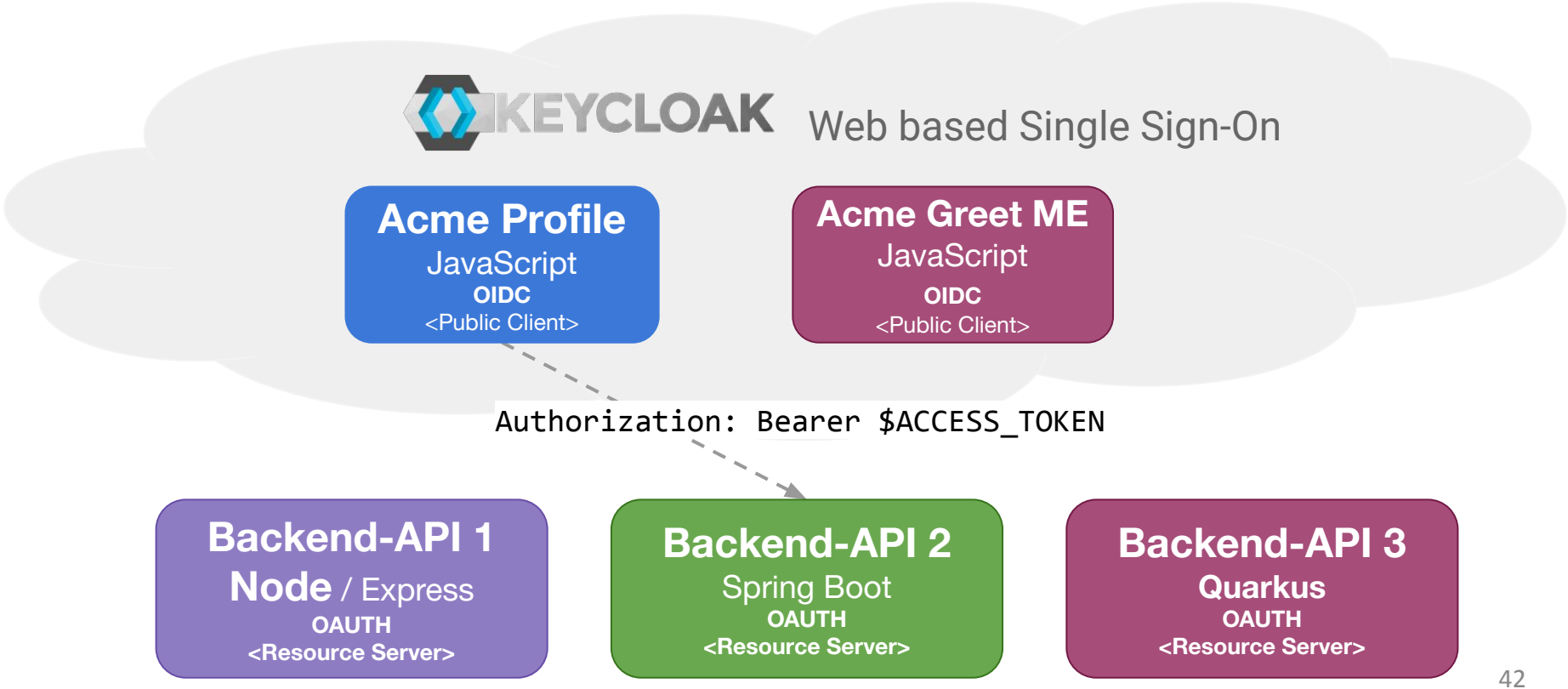
Keycloak Integration Options

- Keycloak Adapters (Deprecated! → Blog Post)
- Use generic library integrations! (Java, .Net (Core), Python, Node,...)
  - Spring Security 5+ JWT / OAuth / OIDC support is sufficient for your use-cases
  - see OIDC and SAML
- Reverse Proxies
  - Pomerium - Identity Aware Proxy with IdP support
  - oauth2-proxy - Auth-Proxy written in Go
  - OpenResty - Nginx with Lua plugins, e.g. lua-resty-oidc
  - Nginx with auth_request module
  - Apache httpd with mod_auth_openidc
  - Vouch Proxy

Keycloak Demo

Securing Apps

Demo Environment



Keycloak Example Collection

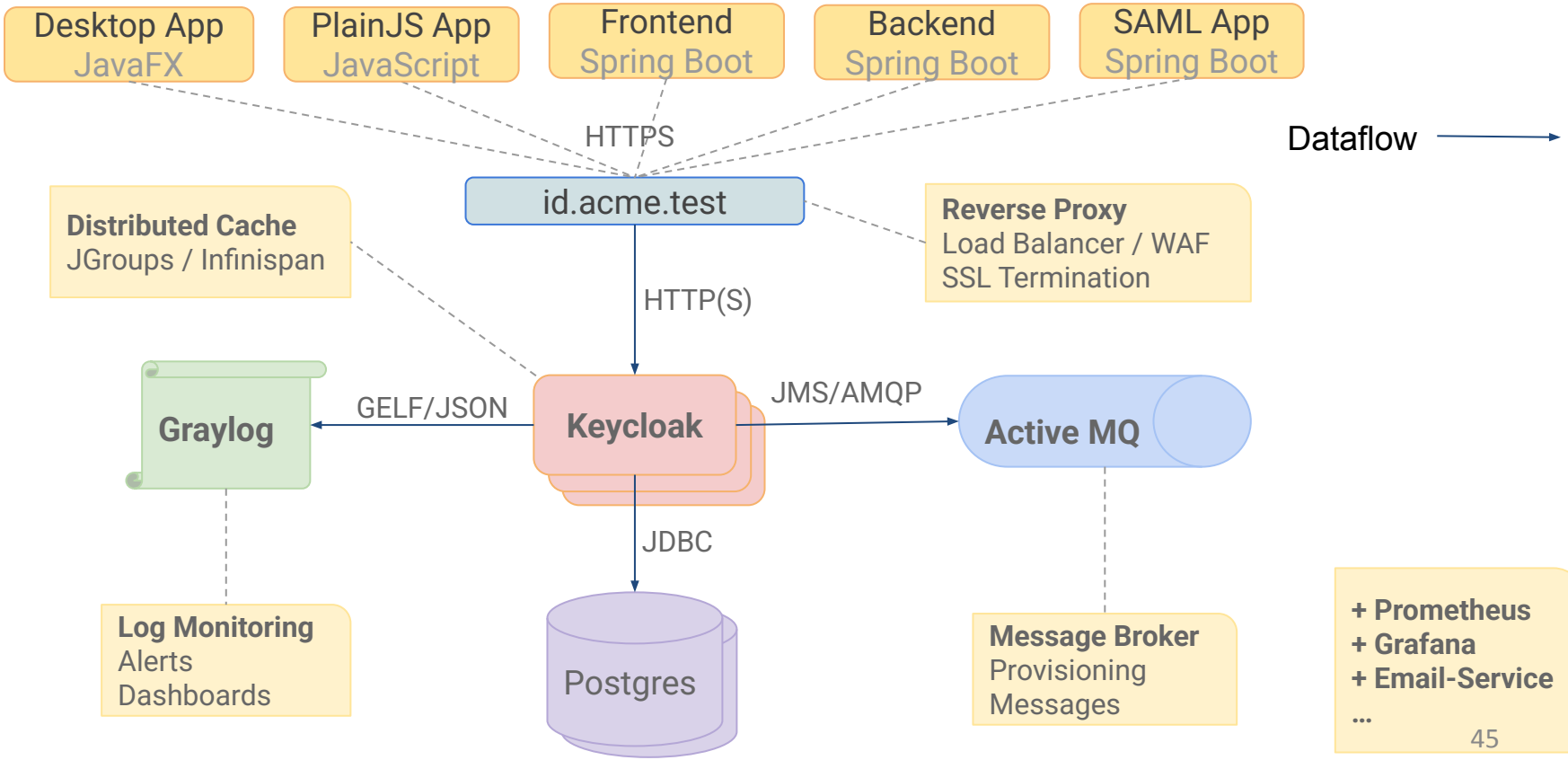
- Securing Apps
- Example Extensions
- Realm Config as Code
- Example Applications

thomasdarimont/keycloak-project-example

Keycloak in the Field

How can a Keycloak environment look like?

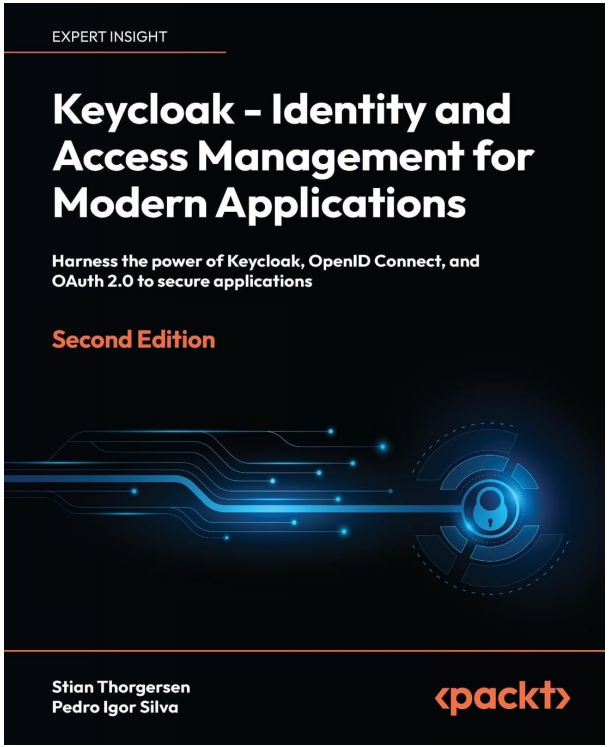
Example Environment



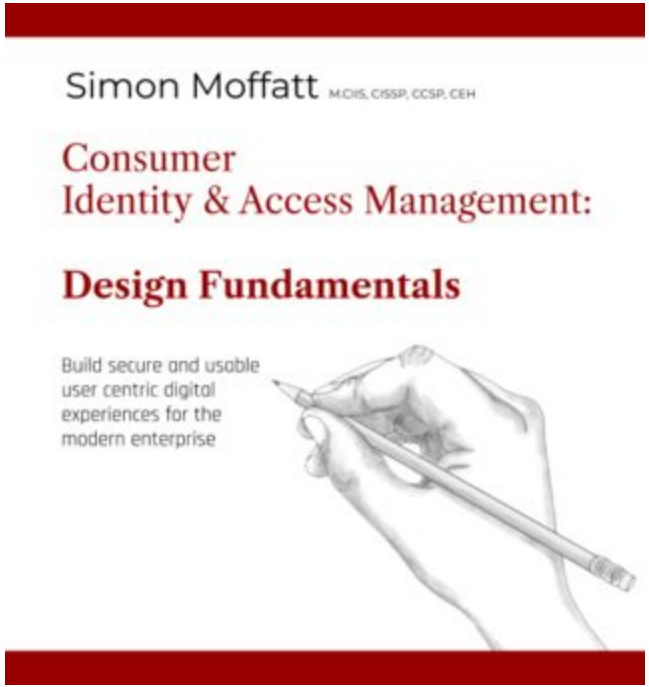
KEYCLOAK Summary

- Easy to get started
  - unzip & run, Keycloak Docker Images
- Provides many features out of the box
  - SSO, Social Login, Federation, User Management,...
- Builds on proven and robust standards
  - OAuth 2.0, OpenID Connect 1.0, SAML 2.0
- Very extensible and easy to integrate
  - Many customization options & extension points
- A pivotal part of a modern Identity Management

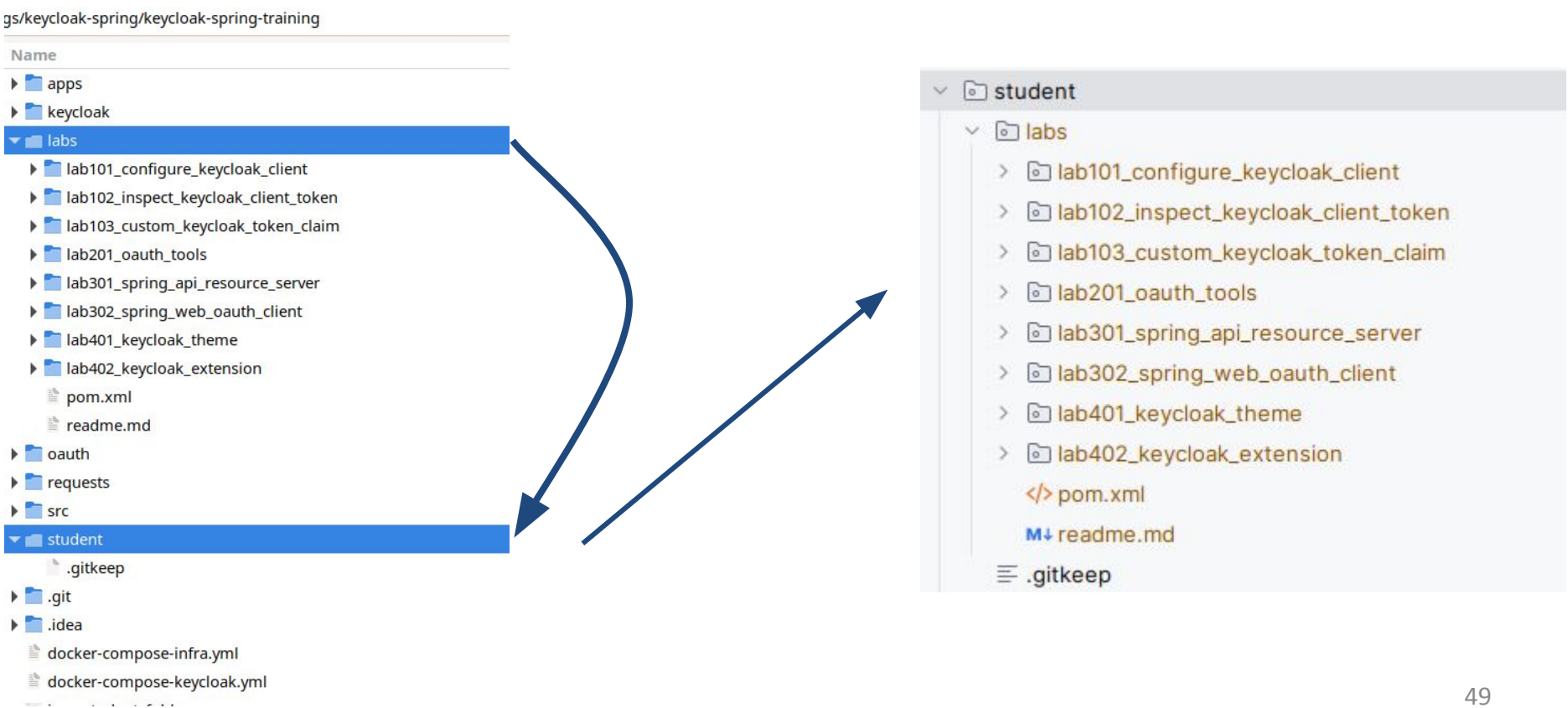
Keycloak Book



CIAM Book


Simon Moffatt M.CIIS, CISSP, CCSP, CEH
Consumer
Identity & Access Management:
Design Fundamentals
Build secure and usable
user centric digital
experiences for the
modern enterprise

Reminder: Copy `labs` Folder to `student` Folder



Keycloak Labs

- The **student folder** contains the following **labs**
  - **lab101**_configure_keycloak_client
  - **lab102**_inspect_keycloak_client_token
  - **lab103**_custom_keycloak_token_claim
- Follow the **instructions** in the **readme** files
- We'll will do the first lab together

Part 2: OAuth / OpenID Connect

Authentication & Authorization

- Authentication (AuthN)
  - Determines ***who*** *the user* ***is***

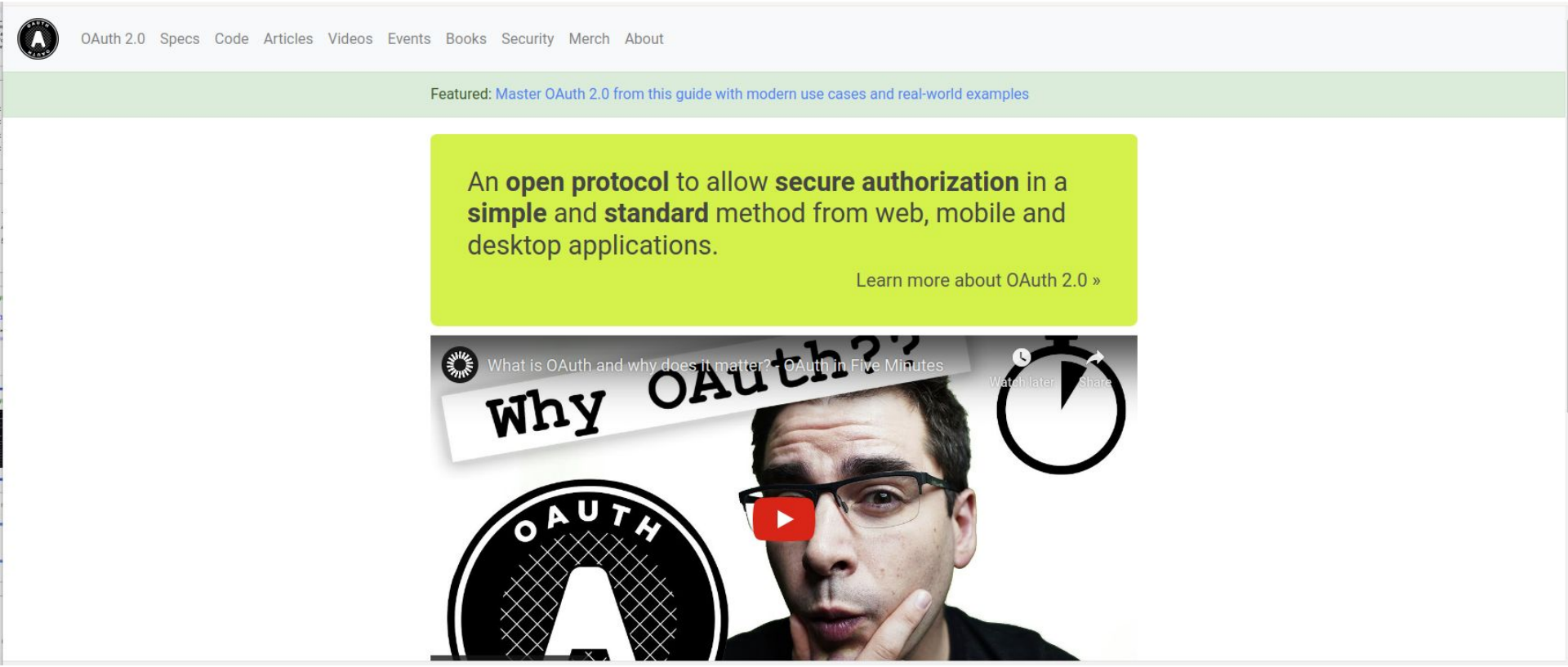
- Authorization (AuthZ)
  - Determines ***what*** *the user* ***is allowed*** *to do*

OAuth vs OpenID Connect

- **OAuth is an *Authorization* Protocol *Framework***
  - Many protocols build on top of OAuth
  - Authorization delegation Protocol
  - API access (on behalf of the user) is the main function

- **OpenID Connect is an *Identity Layer* on top of OAuth**
  - Defines Authentication metadata
  - Session handling
  - Controls authentication
  - Supports federation

OAuth 2.0


OAuth 2.0 Specs Code Articles Videos Events Books Security Merch About
Featured: Master OAuth 2.0 from this guide with modern use cases and real-world examples
An open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications.
Learn more about OAuth 2.0 »
What is OAuth and why does it matter? - OAuth in Five Minutes
Watch later
Share
Why OAuth??
OAUTH

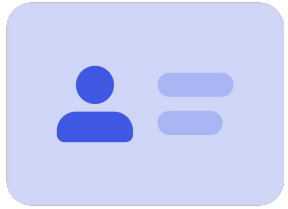
OAuth Overview

- OAuth 2 IETF Standard since 2012 (RFC 6749)
- Authorization Protocol “Framework”
  - Security Protocol, for Access Delegation
  - Defines *Grant Types* → Methods to obtain an Access Token → “*Flows*”
  - Many protocols built on top of OAuth
- Focus on Authorization
  - Authentication left as implementation detail to Authorization Server
  - Uses token based authentication
  - Requires HTTPS (TLS)
- Current Version OAuth 2.1 draft
  - Consolidates many OAuth RFCs with collective enhancements
  - Based on Best Current Practices

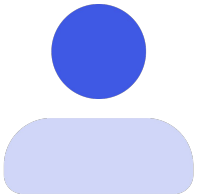
The Problem

- 3rd Party App needs access to API on Users behalf
- App should not “see” the User’s Credentials
- User needs to be aware that it has allowed App access

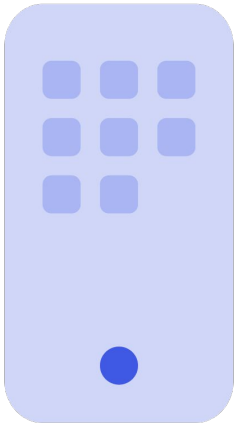
Problem Space



User
Account

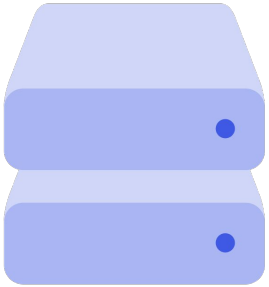


User



App
(3rd Party)

Access Data

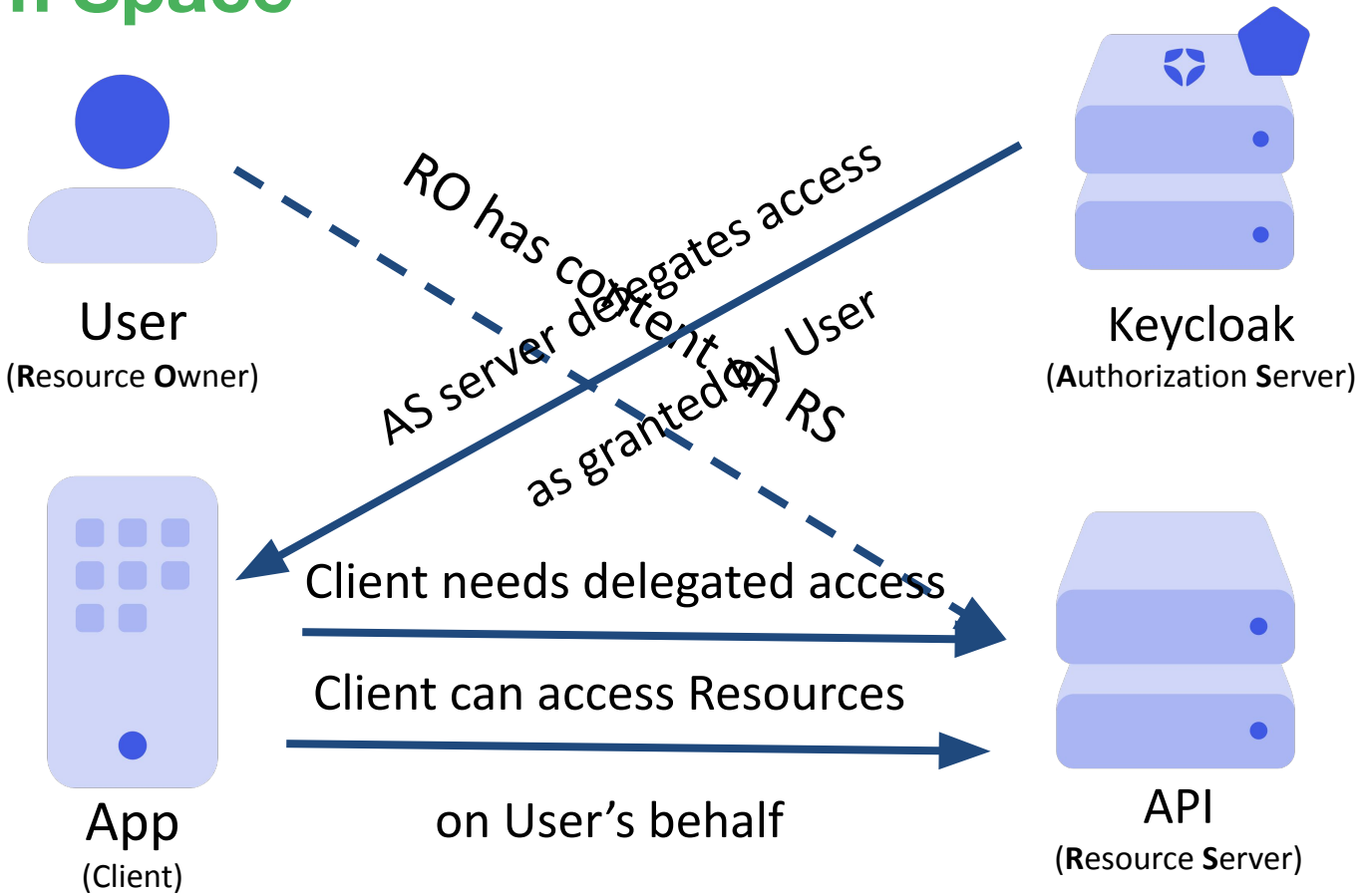


API
User Data

OAuth Actors

- **Resource Owner** (RO)
  - User that can claim ownership over some resources
- **Resource Server** (RS)
  - Service / Web Application which hosts resources related to users
- **Client**
  - Application that wants to access data from a RS on behalf of a user
- **Authorization Server** (AS)
  - Server which issues tokens for accessing a RS

Problem Space



OAuth Client Types

- **Confidential** (Authenticated Clients)
  - Can securely manage a secret → client can authenticate against AS
  - Examples: Server-side Web App, SPA Hosting Backend, Identity Provider
  - Multiple client authentication methods
    i. Client ID / Secret, Private Key JWT, X.509 Certificate
    ii. Most common Client ID / Client Secret / [Redirect URI]

- **Public** (Unauthenticated Clients)
  - can **NOT** securely manage a secret
  - Examples: JavaScript SPA, CLI App, Desktop App, Mobile App
  - Client ID / [Redirect URI]

OAuth Grants = Interaction Patterns = Flows

All OAuth Grants

- **Authorization Code Grant**
  - "Standard Flow" for interactive Browser logins
- **Refresh Token Grant**
  - Obtain new tokens
- **Client Credentials Grant**
  - Machine to Machine / Service to Service communication
- **Device Code Grant**
  - Authenticate through different device (CLI Apps, constrained IoT devices)
- **Legacy Flows** Not recommended, don't use for new apps!
  - Resource Owner Password Credentials Grant / "Password Flow"
  - Implicit Grant / "Implicit Flow"

OAuth 2 Endpoints

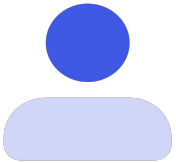
- **Authorization Endpoint**
  - Defined in RFC 6749 (The OAuth 2.0 Authorization Framework)
  - Used to interact with the resource owner for obtaining an authorization grant
  - For obtaining user consent and authorization codes
  - API requires `response_type` as a mandatory request parameter
  - May issue tokens, depending on the `response_type`
- **Token Endpoint**
  - Also defined in RFC 6749
  - For exchanging an authorization code, refresh token, or credentials for tokens
    - Access-Token
    - Refresh-Token

OAuth 2 Endpoints cont.

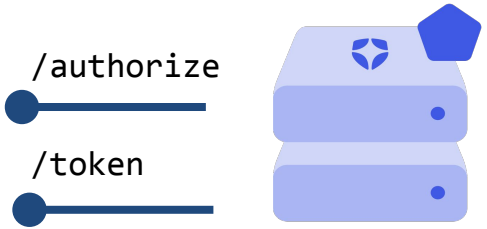
- **Revocation Endpoint**
  - Defined in RFC 7009, OAuth 2.0 Token Revocation
  - Used by clients to revoke access tokens or refresh tokens
- **Introspection Endpoint**
  - Defined in RFC 7662, OAuth 2.0 Token Introspection
  - For resource servers to obtain information about access token validity and metadata

Code Flow

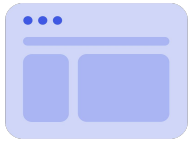
Code Flow



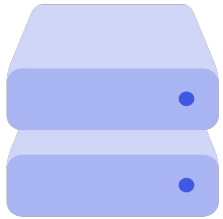
Resource **O**wner



Authorization **S**erver

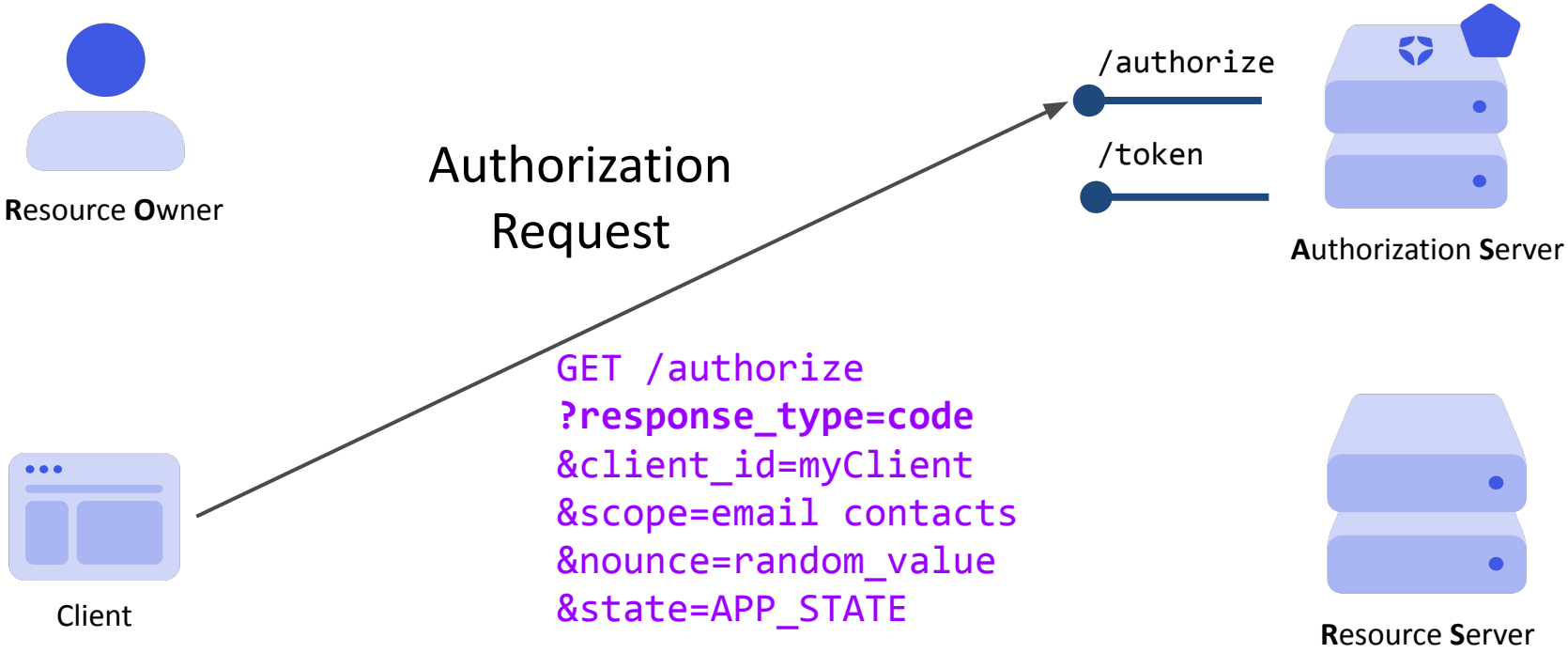


Client

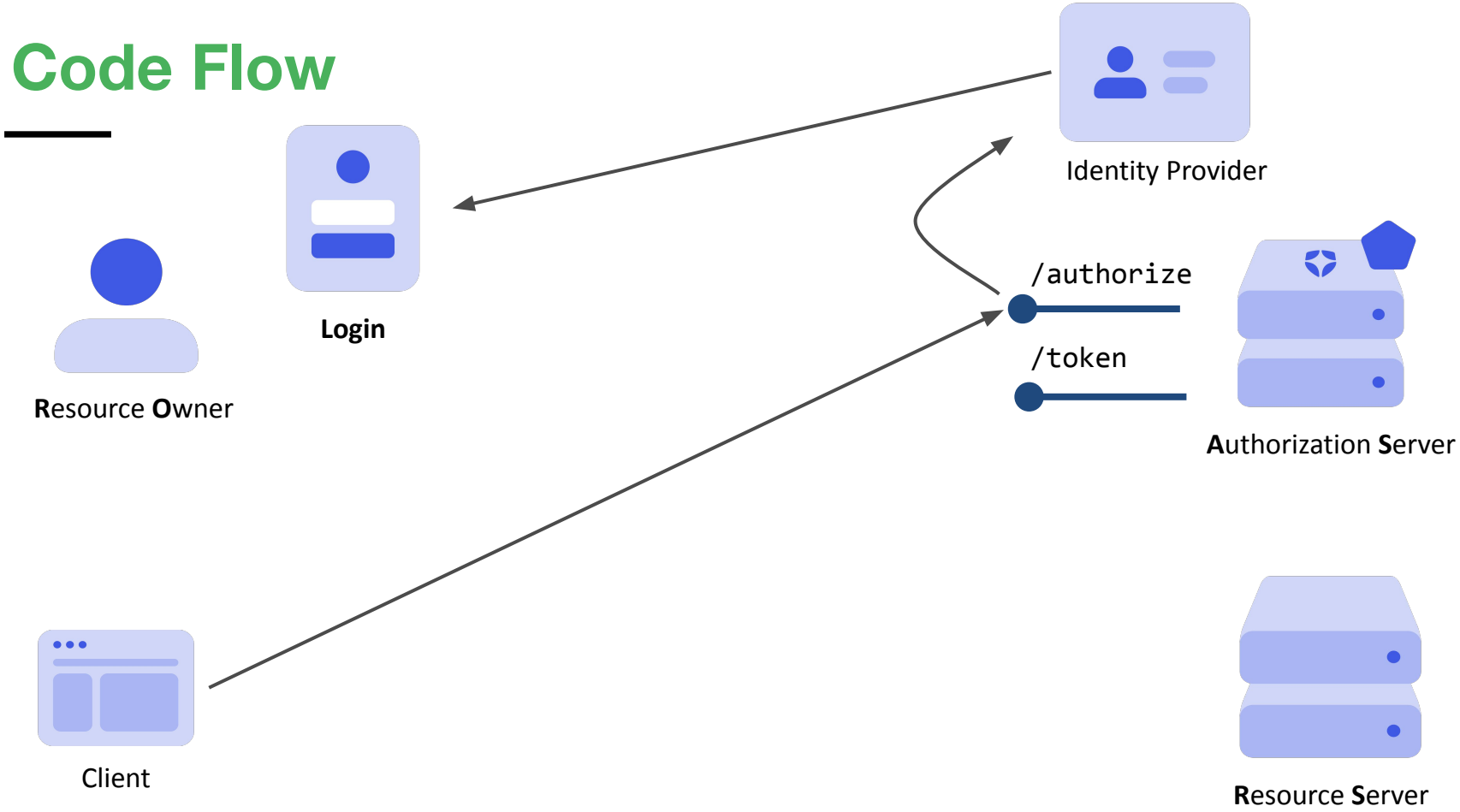


Resource **S**erver

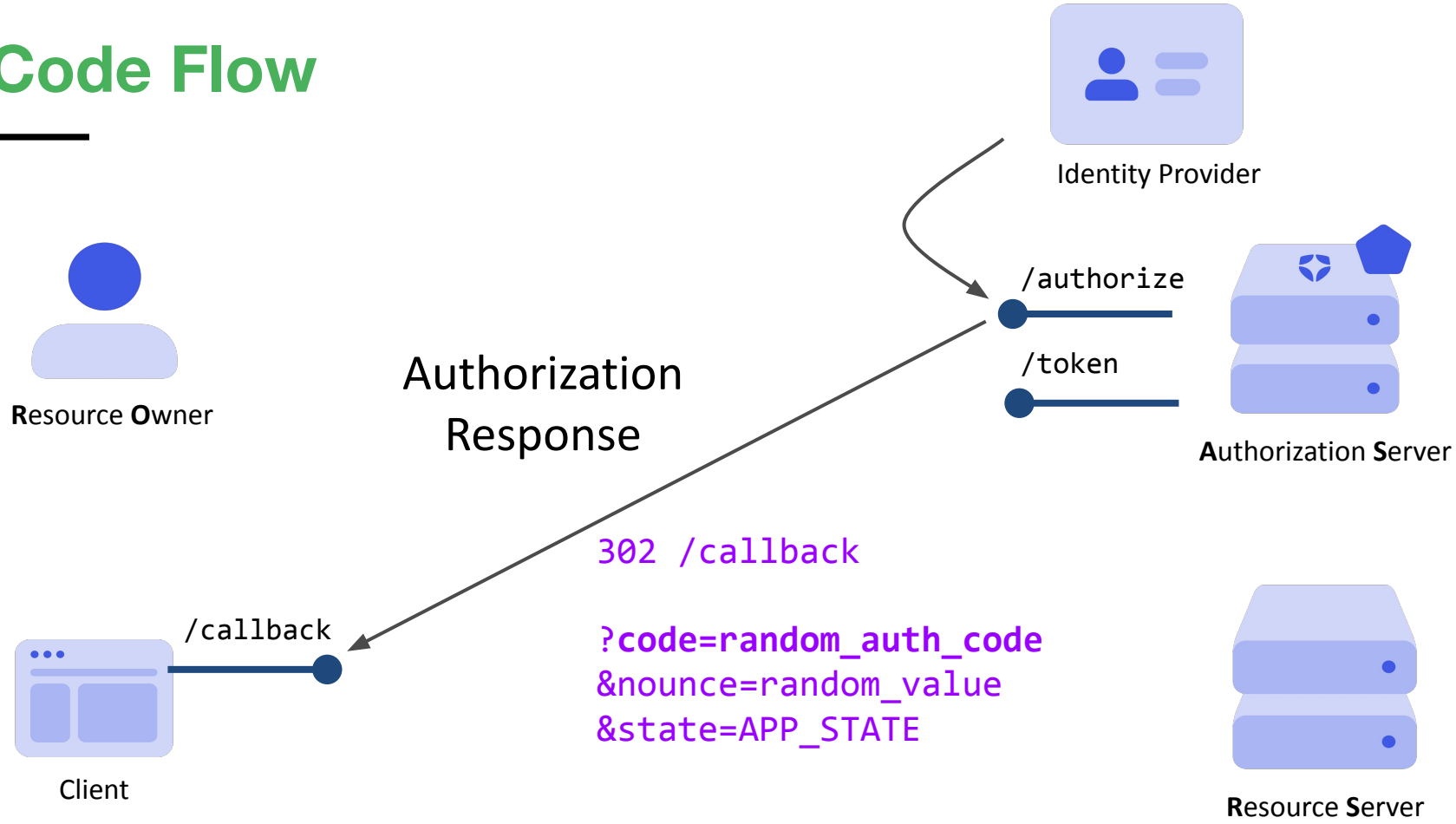
Code Flow


Resource Owner
Authorization Request
/authorize
/token
Authorization Server
GET /authorize
?response_type=code
&client_id=myClient
&scope=email contacts
&nounce=random_value
&state=APP_STATE
Client
Resource Server

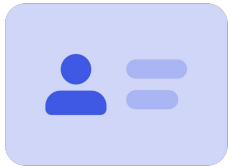
Code Flow


Identity Provider
Login
Resource Owner
/authorize
/token
Authorization Server
Client
Resource Server

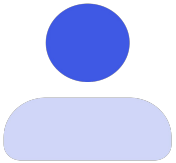
Code Flow


Identity Provider
/authorize
/token
Authorization Server
Resource Owner
Authorization Response
302 /callback
?code=random_auth_code
&nounce=random_value
&state=APP_STATE
/callback
Client
Resource Server

Code Flow



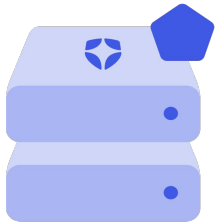
Identity Provider



Resource **O**wner

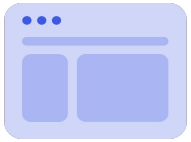
/authorize

/token



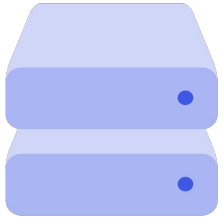
Authorization **S**erver

Token
Request



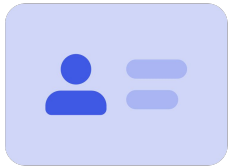
Client

```
POST /token
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&client_id=myClient
&client_secret=geheim
&code=random_auth_code
```

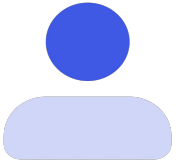


Resource **S**erver

Code Flow



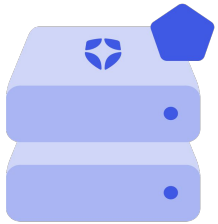
Identity Provider



Resource **O**wner

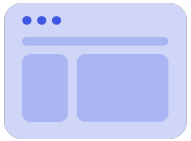
/authorize

/token



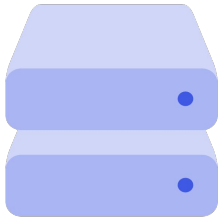
Authorization **S**erver

Token
Response



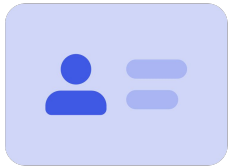
Client

```
{
  “access_token”: “eyAaaaaaaaa…”,
  “scope”: “email contacts”,
  “token_type”: “bearer”,
  “refresh_token”: “eyAaaaaaaa…”,
  …
}
```

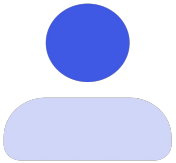


Resource **S**erver

Code Flow



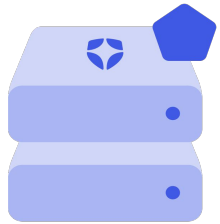
Identity Provider



Resource **O**wner

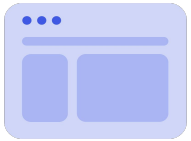
/authorize

/token



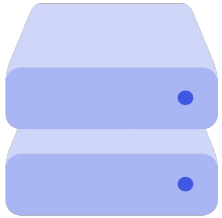
Authorization **S**erver

Resource Access



Client

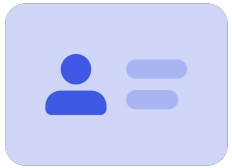
Access Token



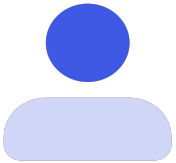
Resource **S**erver

Refresh Flow

Refresh Flow



Identity Provider

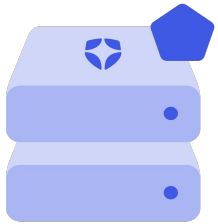


Resource **O**wner

Token
Request

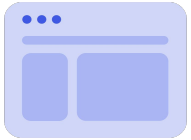
/authorize

/token



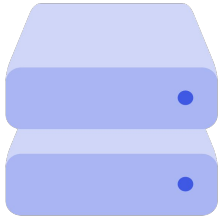
Authorization **S**erver

Refresh Token



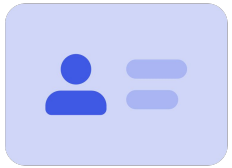
Client

```
POST /token
Content-Type: application/x-www-form-urlencoded

grant_type=refresh_token
&client_id=myClient
&client_secret=geheim
&refresh_token=eyAaaaaaaa
```

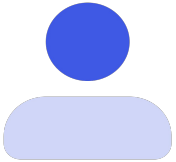


Resource **S**erver

Refresh Flow



Identity Provider

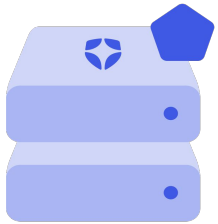


Resource **O**wner

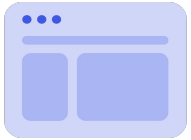
Token
Request

/authorize

/token

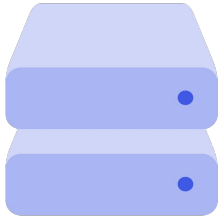


Authorization **S**erver



Client

```
{
  “access_token”: “eyAaaaabbbbb…”,
  “scope”: “email contacts”,
  “token_type”: “bearer”,
  “refresh_token”: “eyAaaabbbbb…”,
  …
}
```



Resource **S**erver

Code Grant Flow + PKCE

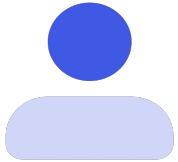
Proof Key for Code Exchange (PKCE)

- Problem
  - Authorization code can be intercepted and used by different client instance
  - Public clients have no secure way to store *client_secret*

- Solution
  - PKCE (RFC7636) adds a “code verifier” to the protocol
  - “Code verifier” generated on the Client; “Challenge” hash value sent in Auth request
  - → Idea: Only the client with the original “code verifier” can obtain the token
  - → Mitigates Authorization Code interception attack

Note: OAuth 2.1 recommends to use PKCE for every Auth Code Grant Flow

Code Flow + PKCE



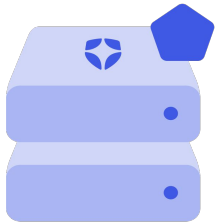
Resource **O**wner

Authorization Request

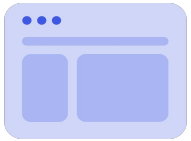
+ code_challenge

/authorize

/token



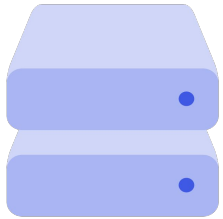
Authorization **S**erver



Client

verifier → sha256(verifier) → **challenge**

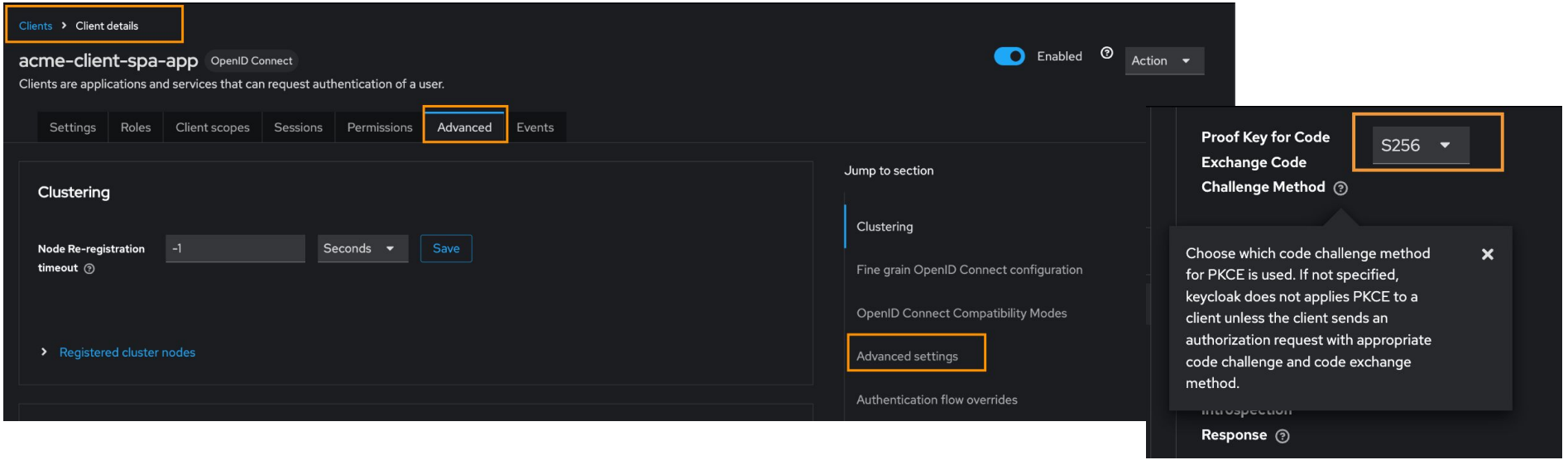
```
GET /authorize
?response_type=code
&client_id=myClient
&scope=email contacts
&nounce=random_value
&state=APP_STATE
&code_challenge=challenge
&code_challenge_method=S256
```



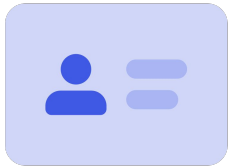
Resource **S**erver

How to enforce PKCE in Keycloak?

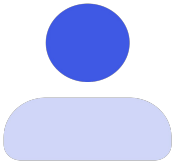
Client Details → Advanced → Advanced Settings → **Proof Key Code ... = S256**



Code Flow + PKCE



Identity Provider

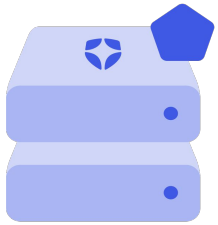


Resource **O**wner

Token
Request
\+ verifier

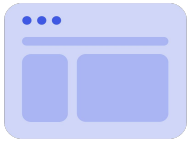
/authorize

/token



Authorization **S**erver

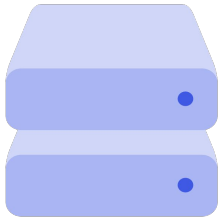
```
code_challenge=challenge
code_challenge_method=S256
```



Client

verifier ⟶ sha256(verifier) ⟶ challenge

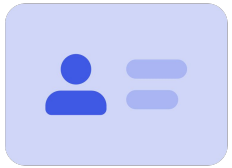
```
POST /token
Content-Type: application/x-www-form-urlencoded

grant_type=authorization_code
&client_id=myClient
&client_secret=geheim
&code=random_auth_code
&code_verifier=verifier
```



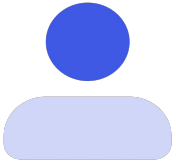
Resource **S**erver

Client Credentials Flow

Client Credentials Flow



Identity Provider

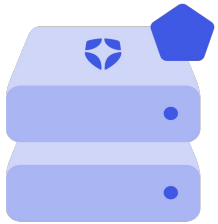


Resource **O**wner

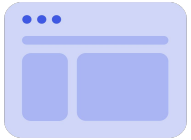
Token
Request

/authorize

/token

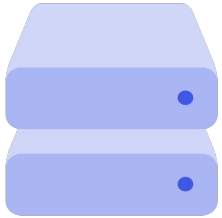


Authorization **S**erver



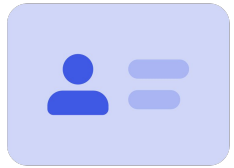
Client

```
POST /token
Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials
&client_id=myClient
&client_secret=geheim
&scope=contacts api
```

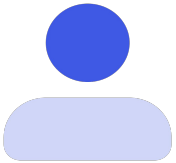


Resource **S**erver

Client Credentials Flow



Identity Provider

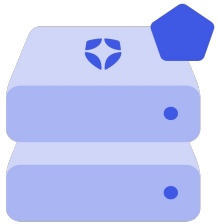


Resource **O**wner

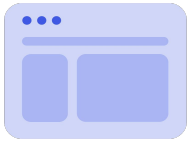
Token
Request

/authorize

/token

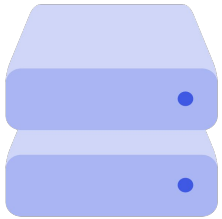


Authorization **S**erver



Client

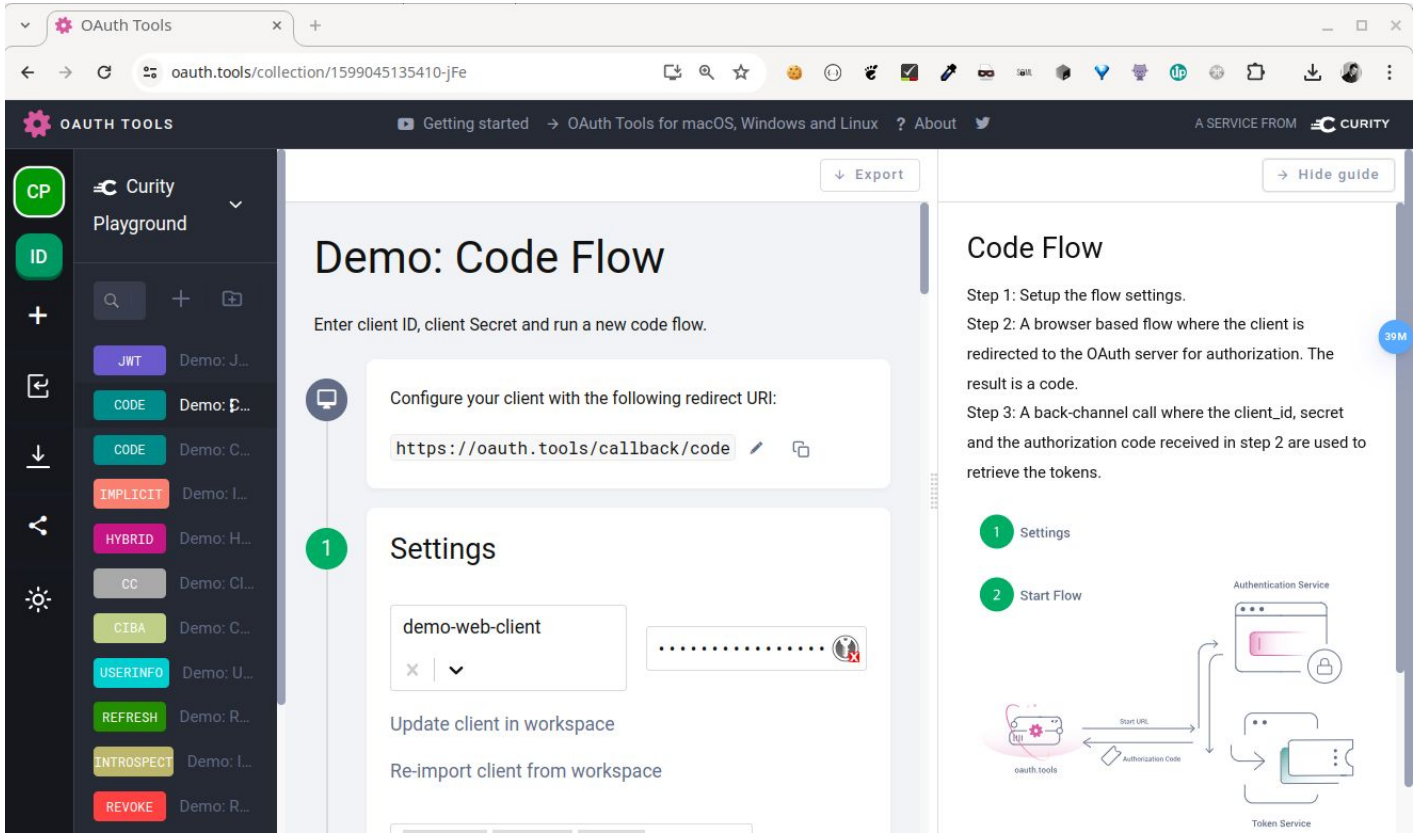
```
{
  “access_token”: “eyAaaaabbbbb…”,
  “scope”: “api contacts”,
  “token_type”: “bearer”,
  …
}
```



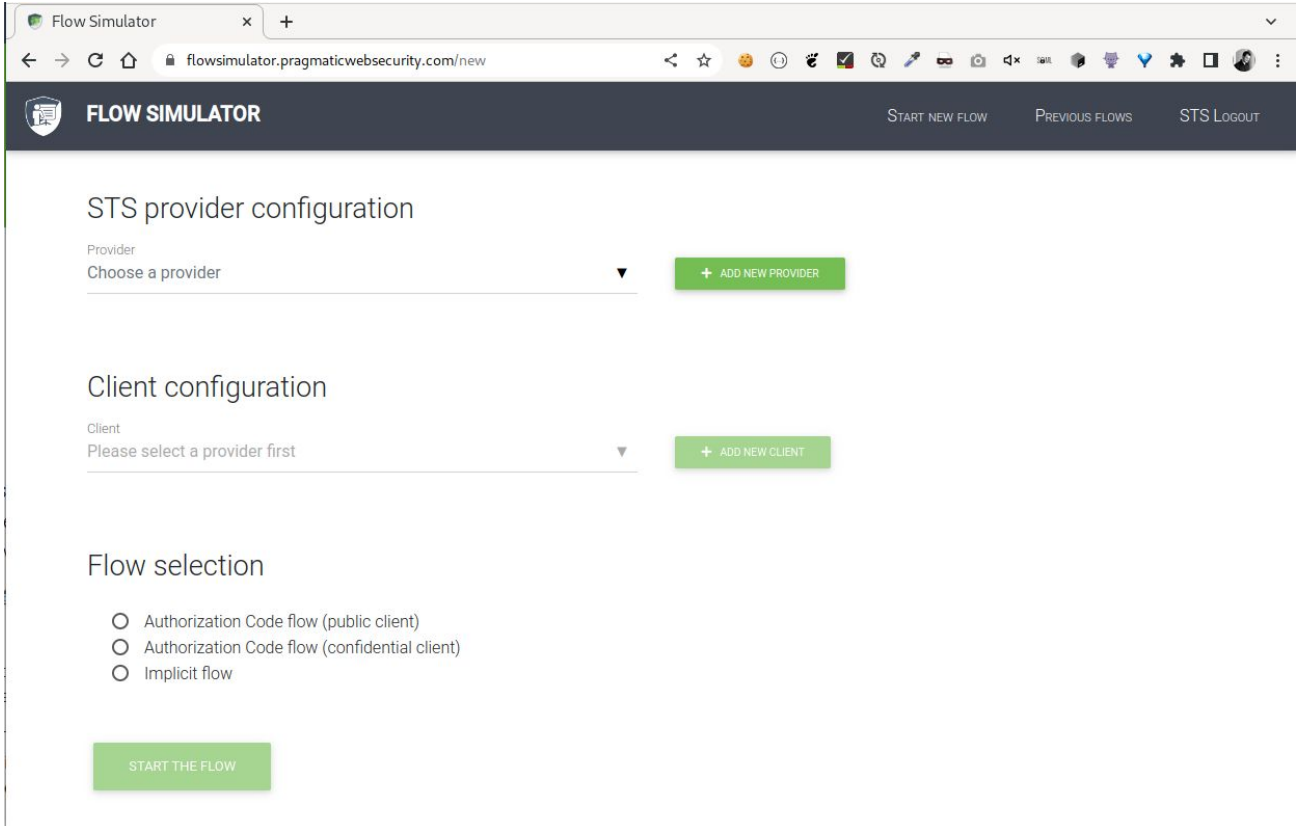
Resource **S**erver

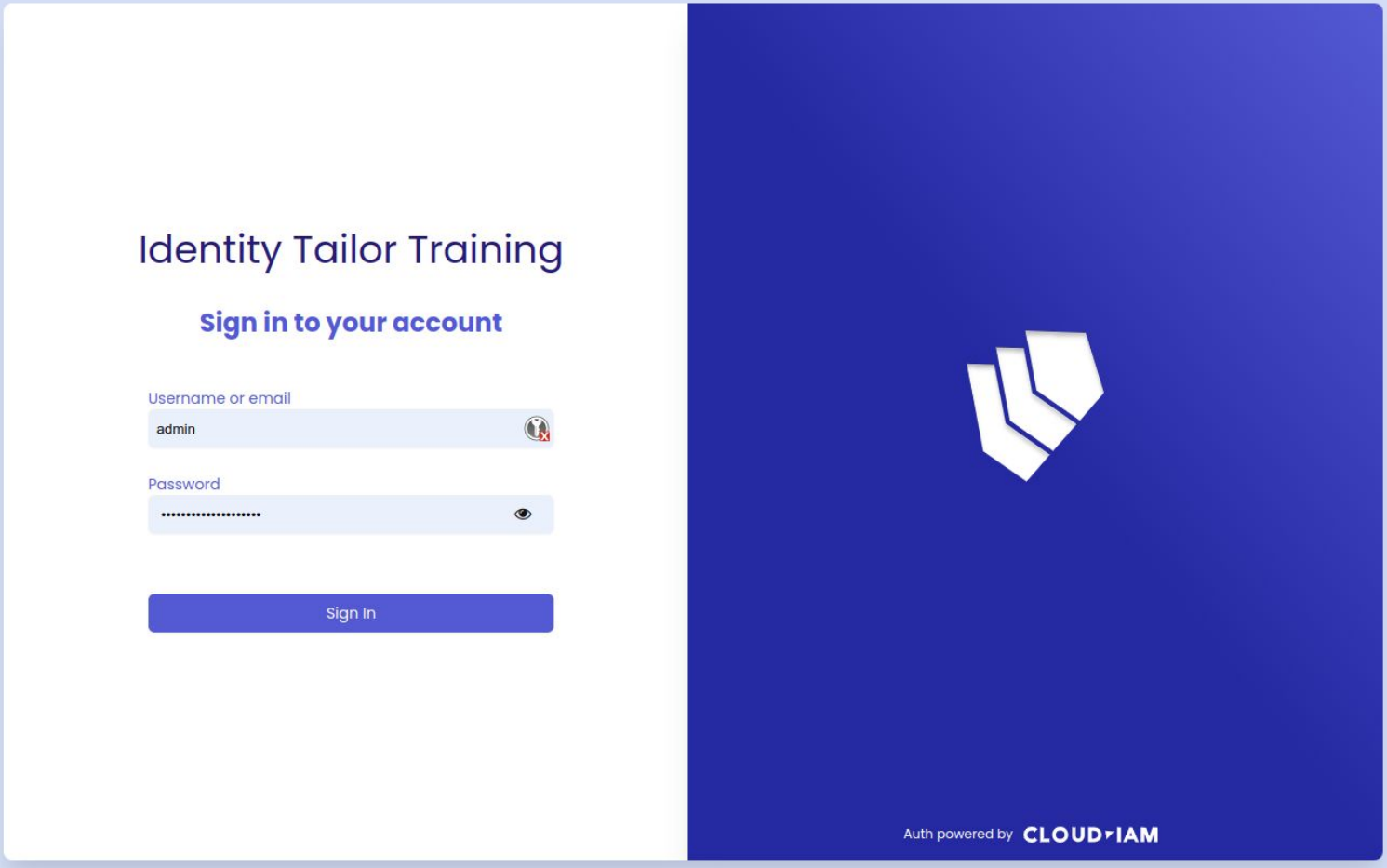
OAuth Flow Demos

OAuth.Tools


OAuth Tools
oauth.tools/collection/1599045135410-jFe
OAUTH TOOLS
Getting started
OAuth Tools for macOS, Windows and Linux
About
A SERVICE FROM CURITY
CP
ID
Curity Playground
JWT
CODE
CODE
IMPLICIT
HYBRID
CC
CIBA
USERINFO
REFRESH
INTROSPECT
REVOKE
Export
Hide guide
Demo: Code Flow
Enter client ID, client Secret and run a new code flow.
Configure your client with the following redirect URI:
https://oauth.tools/callback/code
1
Settings
demo-web-client
Update client in workspace
Re-import client from workspace
Code Flow
Step 1: Setup the flow settings.
Step 2: A browser based flow where the client is redirected to the OAuth server for authorization. The result is a code.
Step 3: A back-channel call where the client_id, secret and the authorization code received in step 2 are used to retrieve the tokens.
1 Settings
2 Start Flow
Authentication Service
Token Service

Pragmatic Web Security Flow Simulator



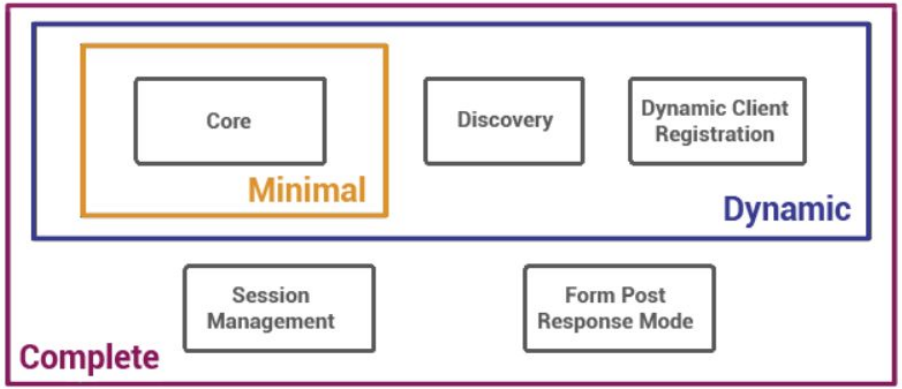

Identity Tailor Training
Sign in to your account
Username or email
admin
Password
Sign In
Auth powered by CLOUDIAM

OAuth Labs

- We use oauth.tools and a Keycloak cloud instance
- The **student folder** contains the following **labs**
    - **lab201**_oauth_tools (multiple flows)
- Follow the **instructions** in the **readme** files
- We'll will do this lab together

OpenID Connect with Keycloak

OpenID Connect Protocol Suite



Underpinnings

OpenID Connect

- Authentication Protocol
  - Extends OAuth 2 with “Identity Layer”
  - Extensive Specification
- Simple, widespread, flexible
- Recommended for Web- and Mobile-Applications
- Additional protocol scope “*openid*”
- User Information exposed via Token or Endpoint
  - Identity Token (IDToken as JWT)
  - UserInfo Endpoint

OpenID Connect Features

- Session Management for SSO + SLO (Logout)
- Identity Token - claims with User-Information
- User Info Endpoint for accessing user information
- Discovery - Self-Describing Endpoints
- Client Self-Registration
- Front- / Backchannel Logout

OpenID Connect Terms

- **End-User (User)**
  - User (resource owner) that claims ownership over some resources
- **Relying Party (RP)**
  - Client, which hosts the resources or provides a service
  - “OAuth Client” / Service Provider / Resource Server / Web Application
- **OpenID Provider (OP)**
  - “Authorization Server” which issues tokens for accessing a RP
  - Manages User Information / Credentials
  - Supports “Direct Authentication” and “Identity Brokering”
  - Issues / Validates Tokens

[OpenID Connect Basic Client Implementer's Guide 1.0]

OpenID Connect Endpoints (excerpt)

- **Discovery Endpoint**
  - Exposes all supported Endpoints as URLs & Provider Metadata
  - Allows for automatic IdP Configuration
- **JWKS Endpoint**
  - Exposes public parts of cryptographic keys used for signing and encryption
- **User-Info Endpoint**
  - Access user information ad-hoc with a valid Access-Token
  - “No size limits” in response body
- **Token Introspection Endpoint**
  - Can be used to validate & decode an Access-Token
  - Can be used to check for “active session”
- **End Session Endpoint**
  - Can be used to logout a User

OpenID Connect Discovery Endpoint

```
GET http://localhost:9090/auth/realms/training/.well-known/openid-configuration HTTP/1.1
```

```
{
"issuer": "$IDP/auth/realms/training",
"authorization_endpoint": "$IDP/auth/realms/training/protocol/openid-connect/auth",
"token_endpoint": "$IDP/auth/realms/training/protocol/openid-connect/token",
"token_introspection_endpoint": "$IDP/auth/realms/training/protocol/openid-connect/token/introspect",
"userinfo_endpoint": "$IDP/auth/realms/training/protocol/openid-connect/userinfo",
"end_session_endpoint": "$IDP/auth/realms/training/protocol/openid-connect/logout",
"jwks_uri": "$IDP/auth/realms/training/protocol/openid-connect/certs",
"check_session_iframe": "$IDP/auth/realms/training/protocol/openid-connect/login-status-iframe.html",
"grant_types_supported": ["authorization_code", "implicit", "refresh_token", "password", "client_credentials"],
...
```

OpenID Connect Introspection Endpoint

```
GET http://localhost:9090/auth/realms/training/protocol/openid-connect/introspect HTTP/1.1
Authorization: Bearer $ACCESS_TOKEN

{
      "exp": 1695199604,
      "iat": 1695199304,
      "auth_time": 1695199301,
      "jti": "afd0be34-a2a5-4f70-8232-c0de16285231",
      "iss": "http://localhost:9090/auth/realms/training",
      "sub": "61445020-4503-4a1d-a807-194c4944f559",
…
      "username": "tester",
      "active": true
}
```

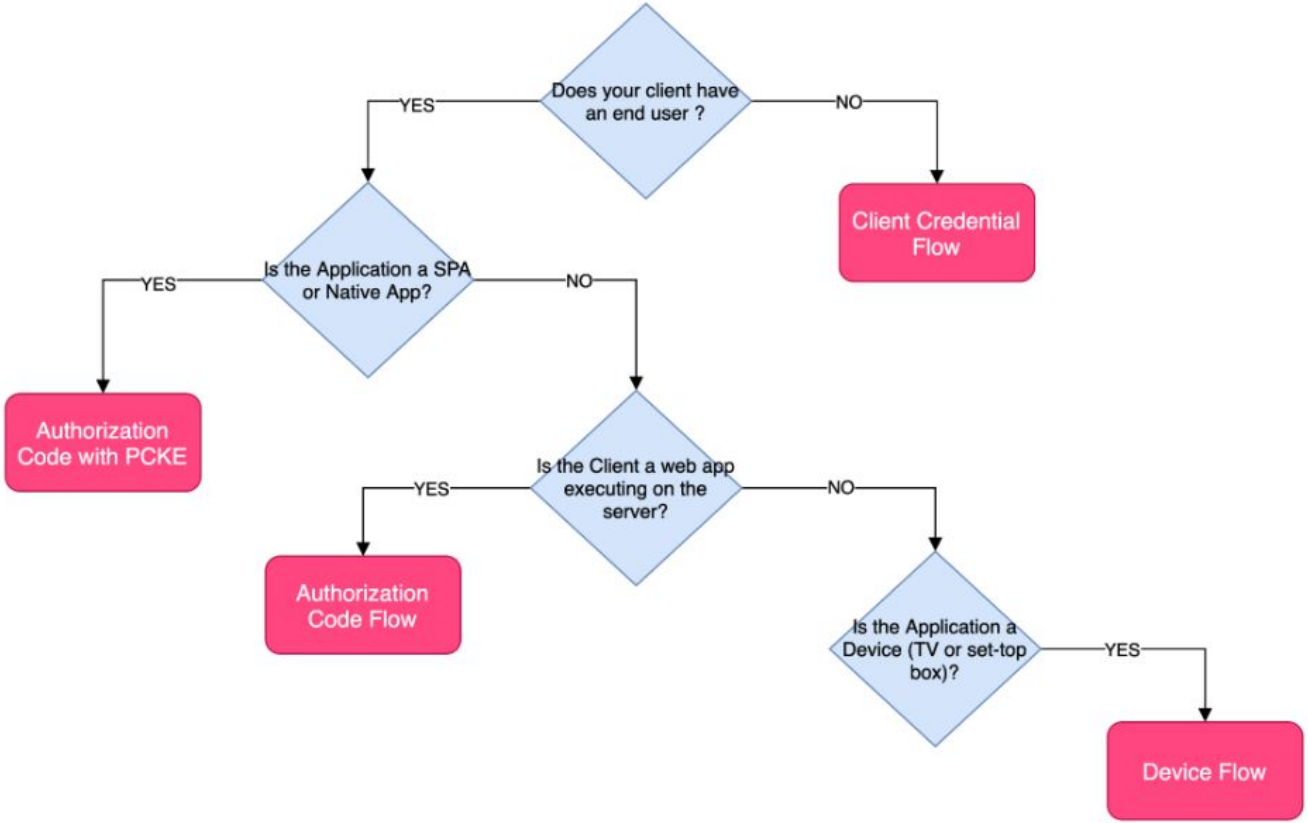
OpenID Connect Userinfo Endpoint

```
GET http://localhost:9090/auth/realms/training/protocol/openid-connect/userinfo HTTP/1.1
Authorization: Bearer $ACCESS_TOKEN

{
  "sub": "af86fe6e-6558-4872-88ee-0e9448e5ae91",
  "email_verified": false,
  "favorite-color": "blue",
  "name": "Theodore Tester",
  "preferred_username": "tester",
  "locale": "en",
  "given_name": "Theodore",
  "family_name": "Tester",
  "email": "tom+tester@localhost"
}
```

Application Types and Authentication Flows Revisited

Application Type	Flow
“Interactive” Application with Browser based Login (Desktop, Mobile, Browser)	Authorization Code Flow + PKCE
Machine-to-Machine (Service-2-Service)	Client Credentials Flow
Constraint Device, CLI (Smart TV, Infodisplay, etc.)	Device Code Flow (+PKCE if supported)

Authentication Flow Decision Tree



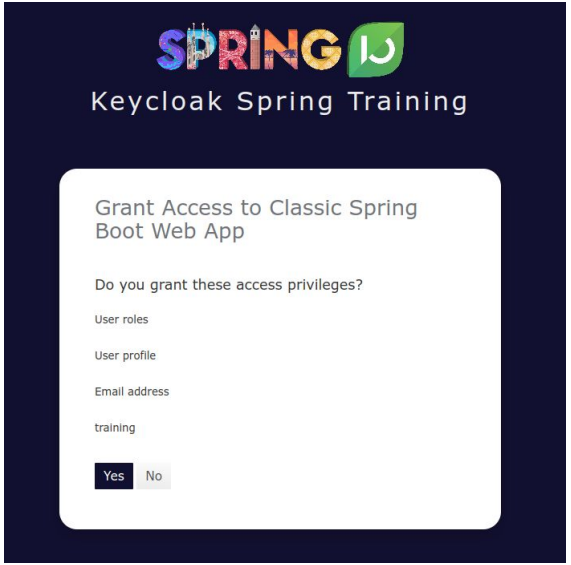
Scope and Claims

OAuth Scopes

- Denotes “Scope” of Access
  - Which data contexts can be accessed
  - What capabilities does the (OAuth) client have?
  - → “client” privileges NOT “user” privileges
- “Keys, not values”
  - Requested by Client
  - Consented by the User
  - Authorized by the Authorization Server
- Example scopes
  - openid
  - contacts
  - email
  - customer_update

User Consent

- Scopes can be consented by the User
- User can optionally consent accesses to client
- Often used with 3rd party client integrations



Claims

- Key / Value items
  - Contained inside the token / userinfo (OIDC)
  - Asserted by the issuer
  - Claim truth about the subject
- Can be used for Fine-Grained Access Control
- Example claims
  - email: “thomas@identity-tailor.de”
  - email_verified: true
  - acme_age_class: “over21”
  - preferred_username: “tester”
  - amr: [“pwd”,”mfa”]
  - subscription: “premium”


```
},
"resource_access" : {
  "account" : {
    "roles" : [ "manage-account", "manage-account-links", "view-profile" ]
  }
},
"scope" : "openid profile email training",
"sid" : "b2cd34ae-ba44-4c3d-b74f-c09582176f51",
"email_verified" : true,
"name" : "Theo Tester",
"preferred_username" : "tester",
"given_name" : "Theo",
"family_name" : "Tester",
"email" : "tester@local"
```

Scopes and Claim Hierarchies

- OpenID Connect defines scope claim mappings
  - Scope “email” → email + email_verified
  - Scope “address” → formatted, street_address, locality, region, postal_code,country
- Custom Scope / Claim hierarchies
  - Scope “contacts_read”
    i. sales_team
  - Scope “media_stream”
    i. age_group
    ii. subscription_plan

Scopes and Claim Summary

- Scopes define “scope of access” for a Token
  - Application level permissions
- Claims provide User Details
  - Trusted identity information about the user
  - Context information (auth level, etc.)
  - User “level permissions”
- Scopes are coarse-grained
- Claims are fine-grained

Part 3: Securing Spring Apps

Spring OAuth2 / OpenID Connect Support

- Spring OAuth2 Resource Server
  - Protect access to an API with an Access Token
- Spring OAuth2 Client
  - Application can use SSO and act as an OIDC Relying Party
- Spring OAuth2 Authorization Server
  - Custom Authorization Server / OpenID Provider
- Spring Cloud Gateway
  - API Gateway with Resource Server / OAuth2 Client support

Spring OAuth2 Resource Server

- Expose a API as an OAuth 2 Resource Server
- Perform authorization based on JWT Access Tokens
- Allow legitimate consumers to call API

Spring OAuth2 Resource Server Config

Example: apps/api

pom.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-resource-server</artifactId>
</dependency>
```

application.yml

```yaml
spring:
  application:
    name: api
  security:
    oauth2:
      resourceserver:
        jwt:
          issuer-uri: "http://localhost:9090/auth/realms/training"
          jwk-set-uri: "${spring.security.oauth2.resourceserver.jwt.issuer-uri}/protocol/openid-connect/certs"
          principal-claim-name: preferred_username
          jws-algorithms: RS256,ES256
```

Allows only this issuer!

Fetch the JWKS here
(public key material)

Allows only those signature algos

Spring OAuth2 Resource Server Filter Chain

```
Thomas Darimont
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

    http.sessionManagement(sessions -> {
        sessions.sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    });
    http.authorizeHttpRequests(requests -> {
        // declarative route configuration
        requests.requestMatchers("/api/**").authenticated();
        // add additional routes
        requests.anyRequest().denyAll(); //
    });
    http.oauth2ResourceServer(resourceServer -> {
        resourceServer.jwt(Customizer.withDefaults());
    });

    return http.build();
}
```

We want stateless services!

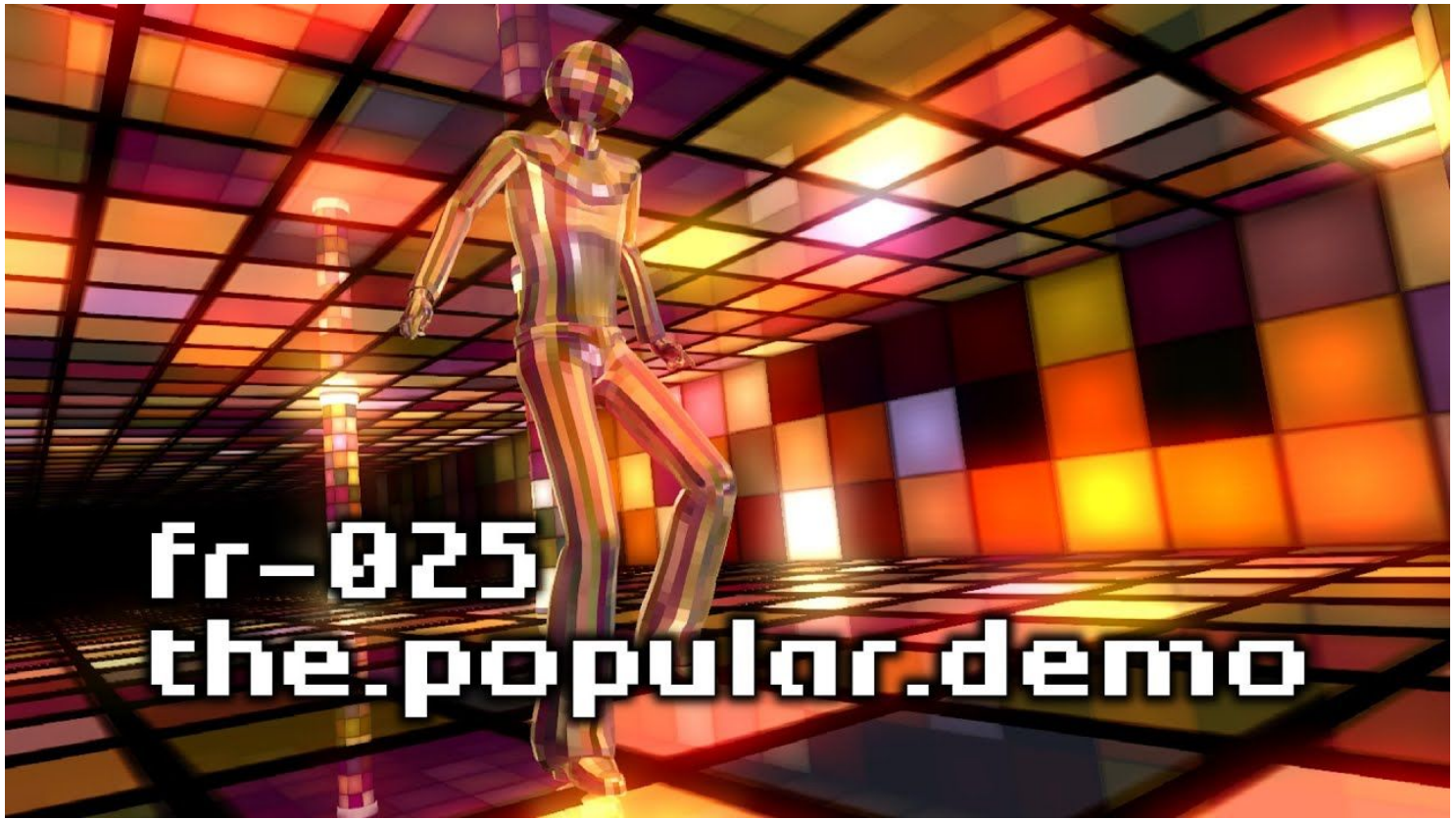
Every API access must be authenticated

Enables ResourceServer configuration

Spring OAuth2 Resource Server Customization

- Keycloak uses custom role claims
  - `realm_resource.roles` → Realm Roles
  - `resource_access.$client_id.roles` → Client specific roles
- `KeycloakJwtAuthenticationConverter`
  - extends `JwtAuthenticationConverter`
  - Custom `GrantedAuthorities` resolution
- Produces a `JwtAuthenticationToken`

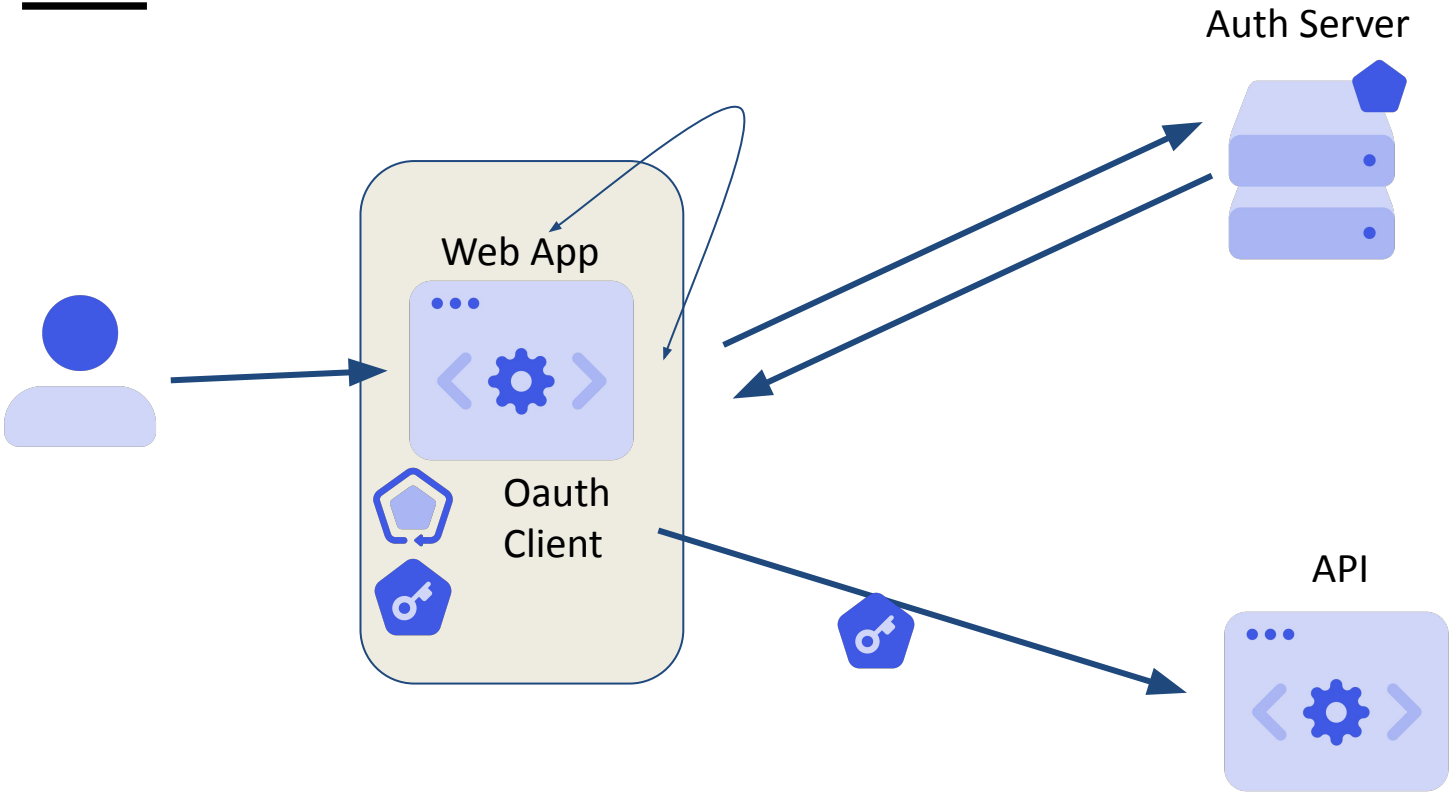
Spring OAuth2 Resource Server Demo: API



Spring OAuth2 Client

- Authenticate Web App via OpenID Connect
- Leverage Single Sign-On Infrastructure
- Integrate with existing Identity Providers
- Use user Access Token to call down-stream APIs

Spring OAuth2 Client



Spring OAuth2 Client Config

Example: apps/web

pom.xml

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

application.yaml

```yaml
spring:
  application:
    name: web
  security:
    oauth2:
      client:
        provider:
          keycloak:
            issuerUri: "http://localhost:9090/auth/realms/training"
            user-name-attribute: "preferred_username"
        registration:
          keycloak:
            client-name: "Keycloak"
            client-id: "training-web"
            client-secret: "secret"
            client-authentication-method: "client_secret_post"
            authorizationGrantType: "authorization_code"
            redirect-uri: "{baseUrl}/login/oauth2/code/{registrationId}"
            scope: "openid,profile,email,training"
```

Declare Keycloak as Identity Provider

Client configuration for this App in Keycloak

We want to use Auth Code Flow

Keycloak will use this callback URI

Spring OAuth2 Client Filter Chain

OAuth Client

Auth Code
Flow + PKCE

```
// Configure this app as an OAuth2 Client
http.oauth2Client(o2cc -> {
    var oauth2AuthRequestResolver = new DefaultOAuth2AuthorizationRequestResolver( //
            clientRegistrationRepository, //
            OAuth2AuthorizationRequestRedirectFilter.DEFAULT_AUTHORIZATION_REQUEST_BASE_URI //
    );
    // We follow the recommendations of OAuth 2.1 and use PKCE also for confidential clients with auth code grant flow
    oauth2AuthRequestResolver.setAuthorizationRequestCustomizer(OAuth2AuthorizationRequestCustomizers.withPkce());
    o2cc.authorizationCodeGrant(custom -> {
        custom.authorizationRequestResolver(oauth2AuthRequestResolver);
    });
});
```

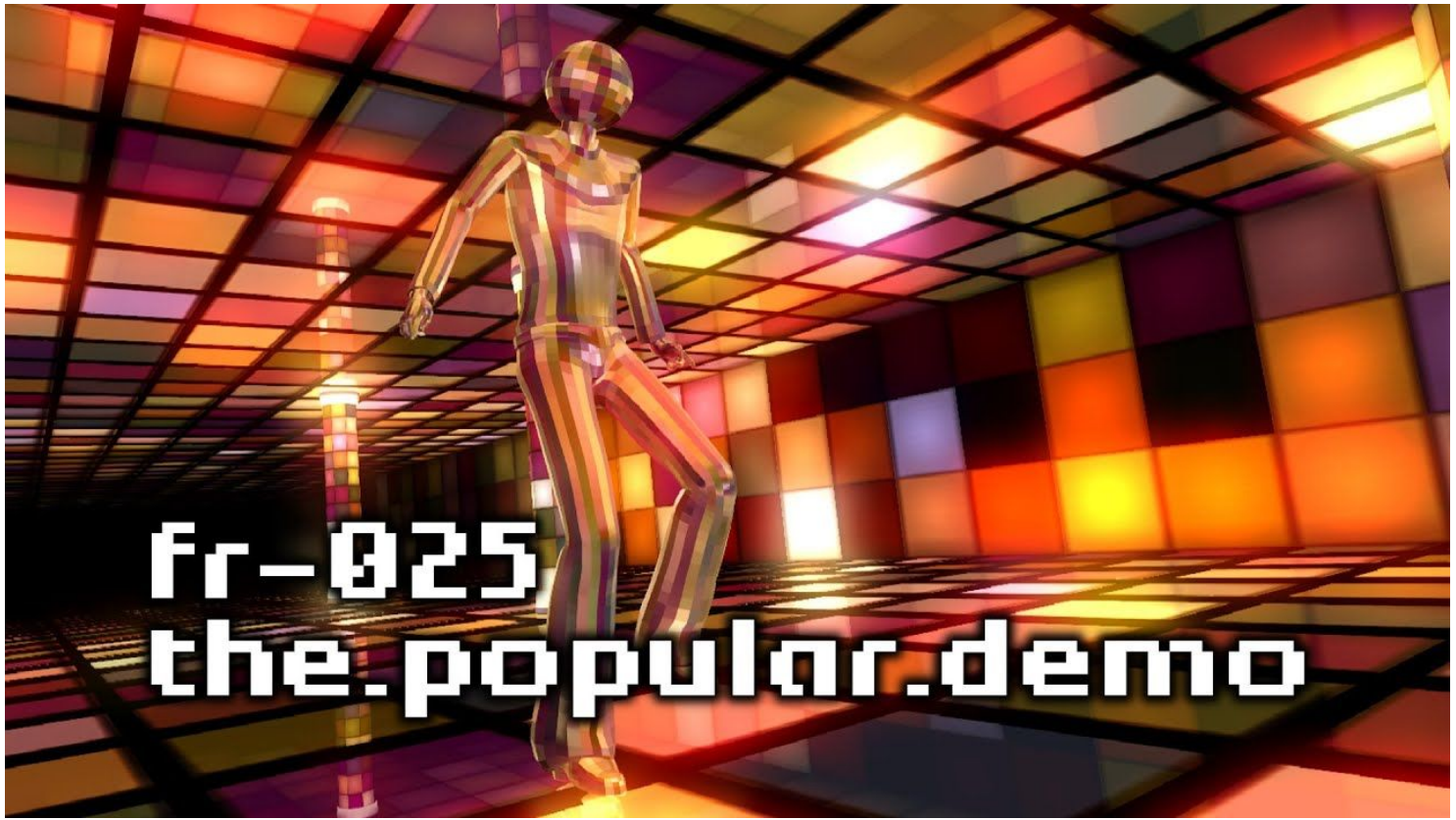
```
// Configure to use the UserInfo Endpoint after login
http.oauth2Login(o2login -> {
    o2login.userInfoEndpoint(customizer -> {
        // Add special handling to process Keycloak Role Information
        customizer.userAuthoritiesMapper(new KeycloakRolesMapper());
    });
});
```

Extracts Authorities
from UserInfo

```
// We use a custom logout-handler which generates the proper OIDC end_session URL for Keycloak
http.logout(logout -> {
    logout.addLogoutHandler(new KeycloakLogoutHandler());
});
```

Construct proper Logout URL

Spring OAuth2 Client Demo: Web



Spring Boot Backend with SPA Frontend

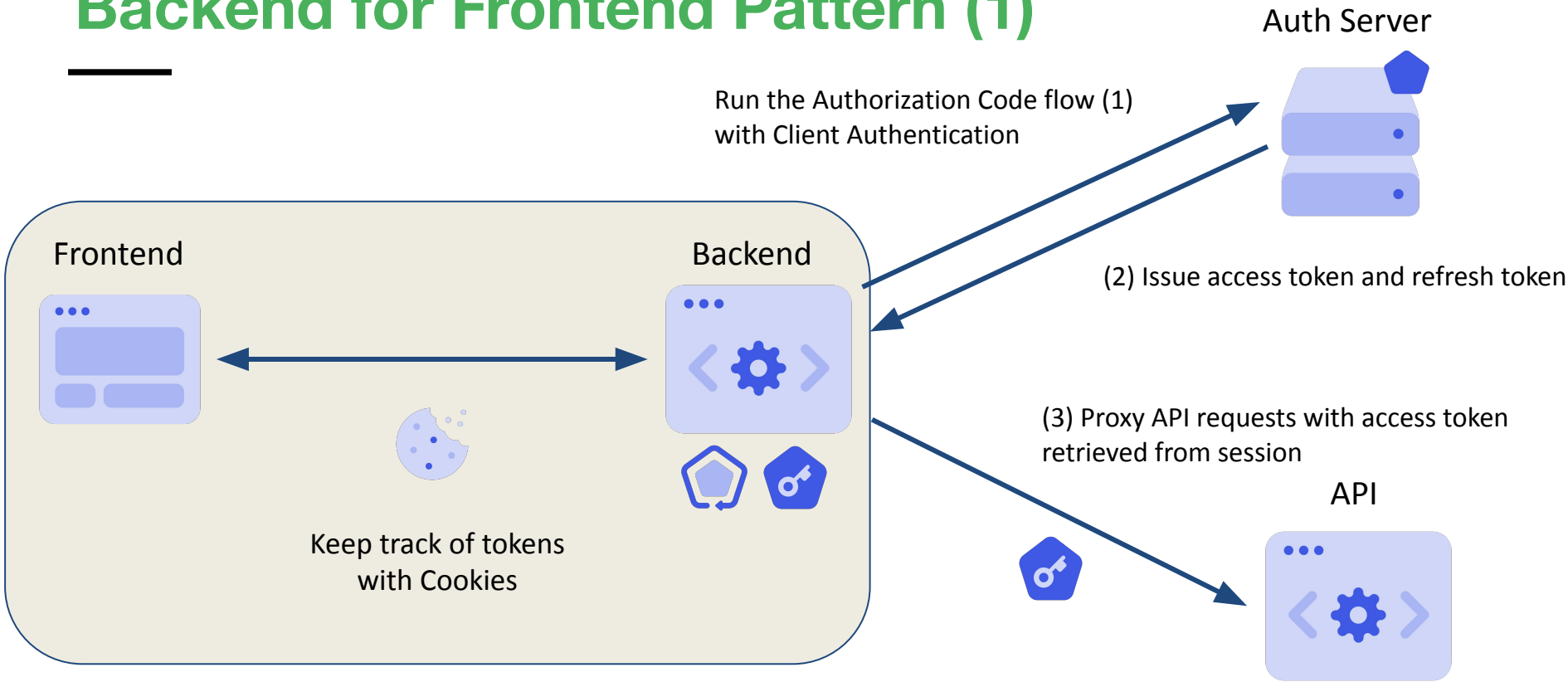
- Provide a Backend for a Browser based App (SPA)
- Authenticate Web App via OpenID Connect
- Leverage Single Sign-On Infrastructure
- Integrate with existing Identity Providers
- Use user Access Token to call down-stream APIs
- Don’t expose tokens to the Browser!

Backend for Frontend Motivation

- Browser based Apps (SPAs) in Combination with a Backend
- Tokens exposed in the browser cannot be secured properly
  - Hard to secure (XSS, Code hijacking)
  - Exposed token can be used to call arbitrary backend APIs
  - Tokens often too powerful
  - OAuth in Frontend increases complexity
- We want
  - Simple & Secure Frontends
  - To avoid exposing tokens to the Browser
  - Limit the API surface the frontend can use

Backend for Frontend (BFF) to the Rescue!

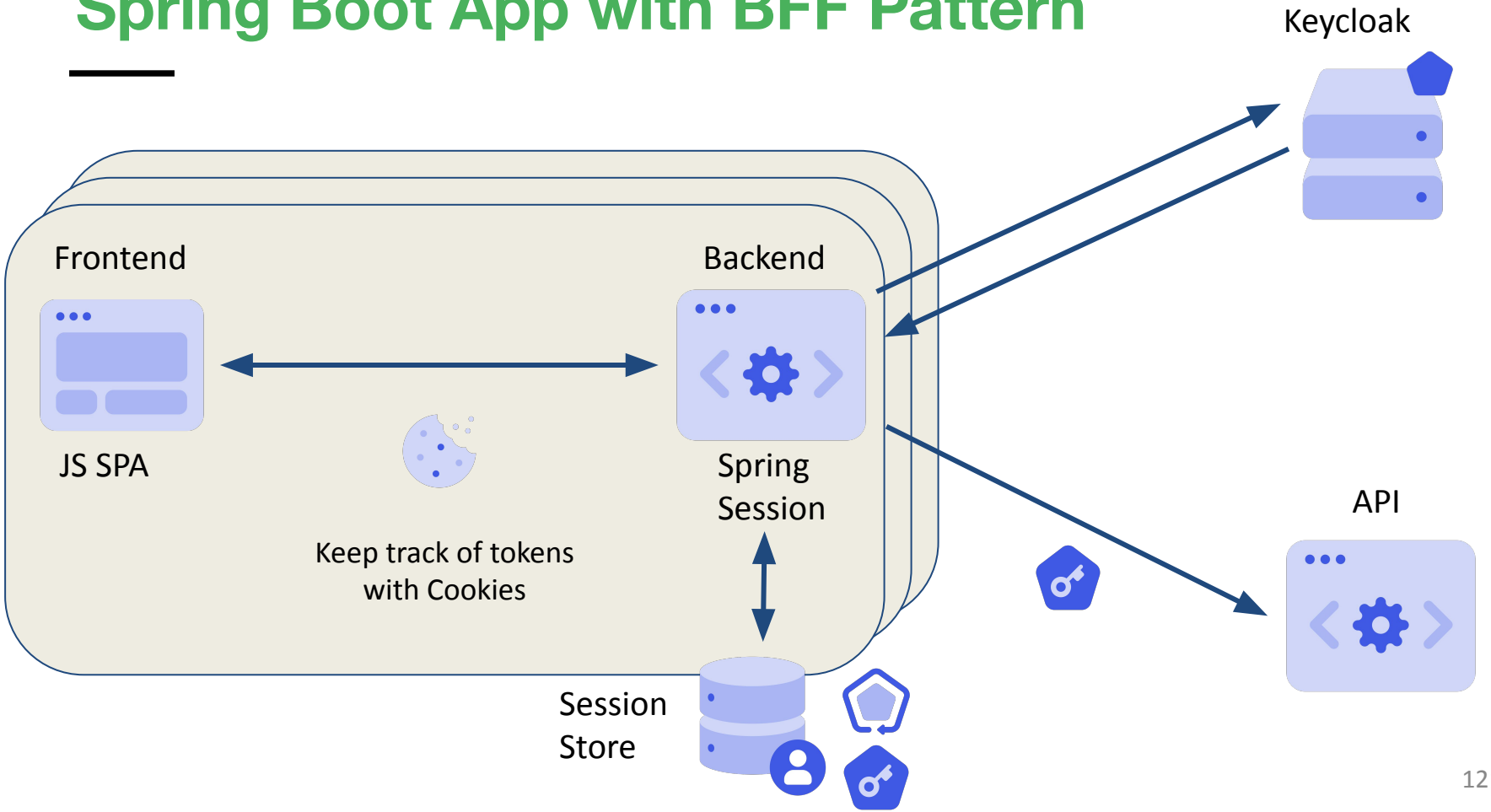
Backend for Frontend Pattern (1)



Backend for Frontend Pattern (2)

- Backend acts as Confidential Client
- OAuth Flows and Token Management handled by the Backend
- Cookie based Session tracking
- Increases Security of Browser based Apps
- State distribution required (Sessions, Tokens)
- Often used in banking and healthcare scenarios

Spring Boot App with BFF Pattern



Spring Boot BFF App Config

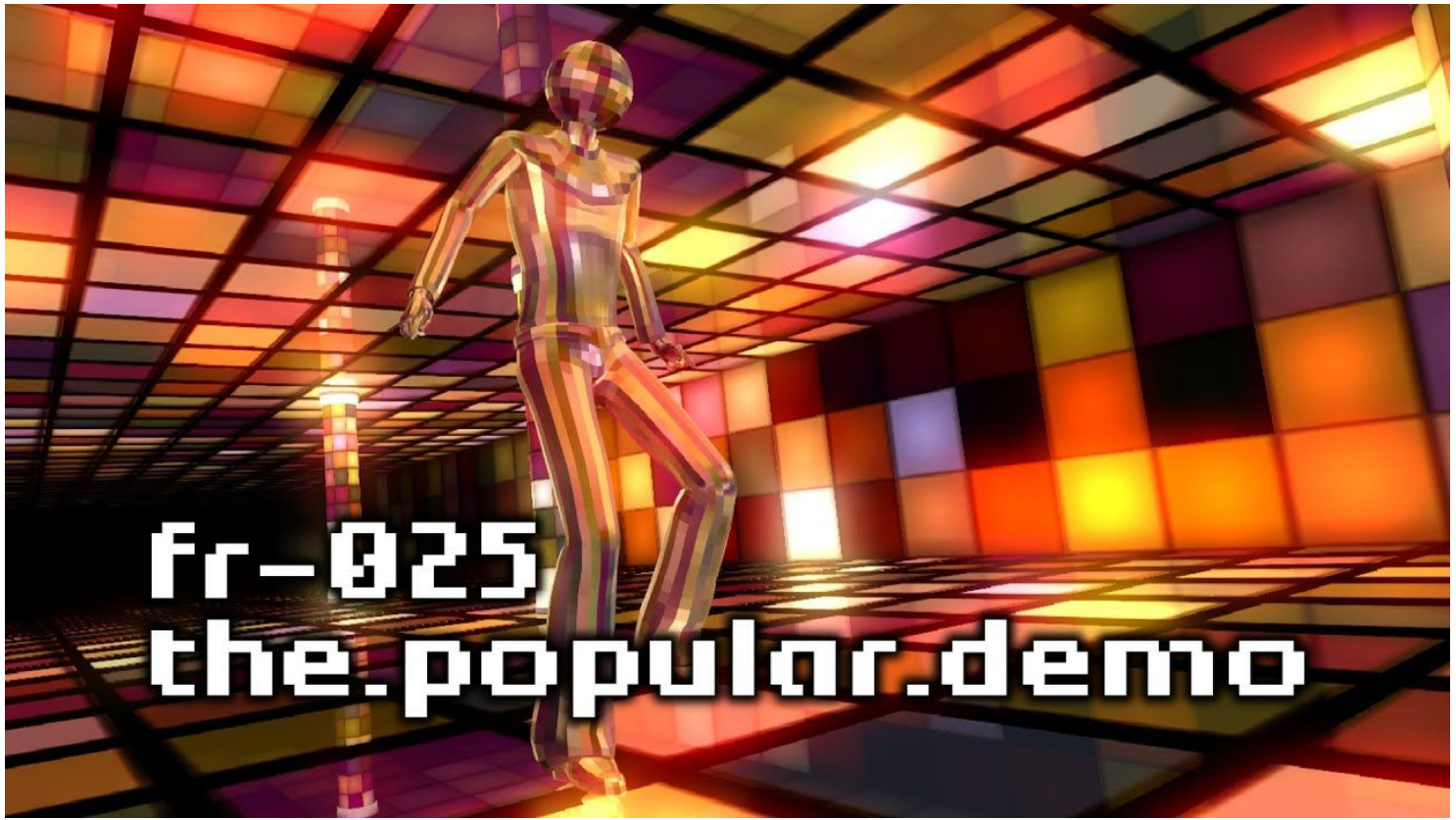
Example: apps/bff

```xml
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
```

```xml
<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>
```

```yaml
security:
  oauth2:
    client:
      provider:
        keycloak:
          issuerUri: http://localhost:9090/auth/realms/training
          user-name-attribute: preferred_username
      registration:
        keycloak:
          client-id: 'training-bff'
          client-secret: 'secret'
          client-authentication-method: client_secret_post
          authorizationGrantType: authorization_code
          redirect-uri: '{baseUrl}/login/oauth2/code/{registrationId}'
          scope: openid
```

Spring Boot BFF App Demo



Spring Cloud Gateway

- Hide your APIs behind a Gateway
- Use uniform authentication / access logic
- Can be used to implement BFF pattern
- Available Variants: Reactive and Classic MVC

Spring Cloud Gateway Config

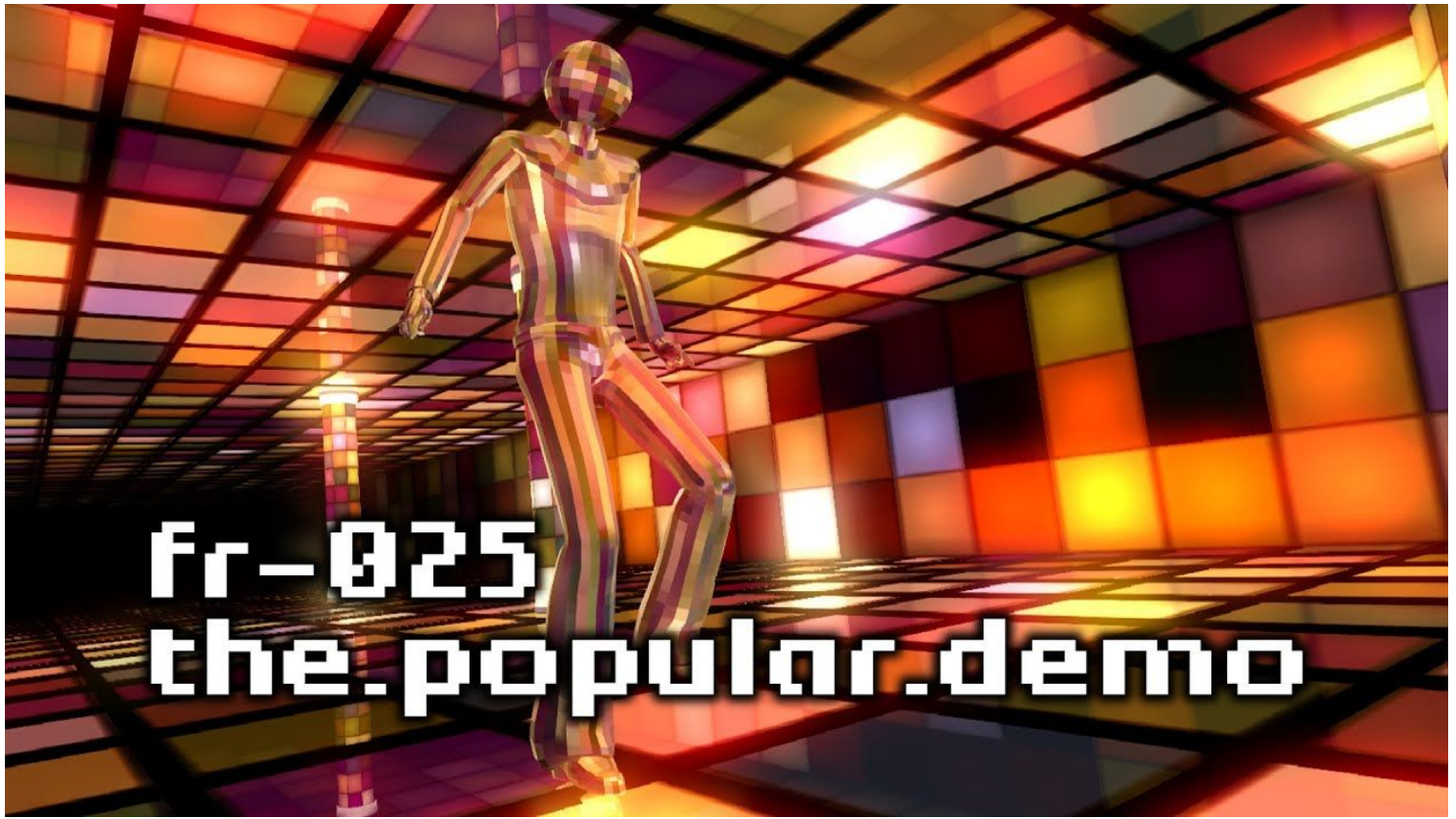
Example: apps/gateway

```xml
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-oauth2-resource-server</artifactId>
</dependency>
```

```yaml
spring:
  application:
    name: gateway
  cloud:
    gateway:
      default-filters:
        - TokenRelay=
      routes:
        - id: api
          uri: http://localhost:8090
          predicates:
            - Path=/api/**
```

```yaml
security:
  oauth2:
    resourceserver:
      jwt:
        jwk-set-uri: "${spring.security.oauth2.client.provider.keycloak.issuer-uri}/protocol/openid-connect/certs"
    client:
      provider:
        keycloak:
          issuer-uri: "http://localhost:9090/auth/realms/training"
      registration:
        keycloak:
          provider: keycloak
          client-id: training-gateway
          client-secret: "secret"
          authorization-grant-type: authorization_code
          scope: openid,training
```

Spring Cloud Gateway Demo

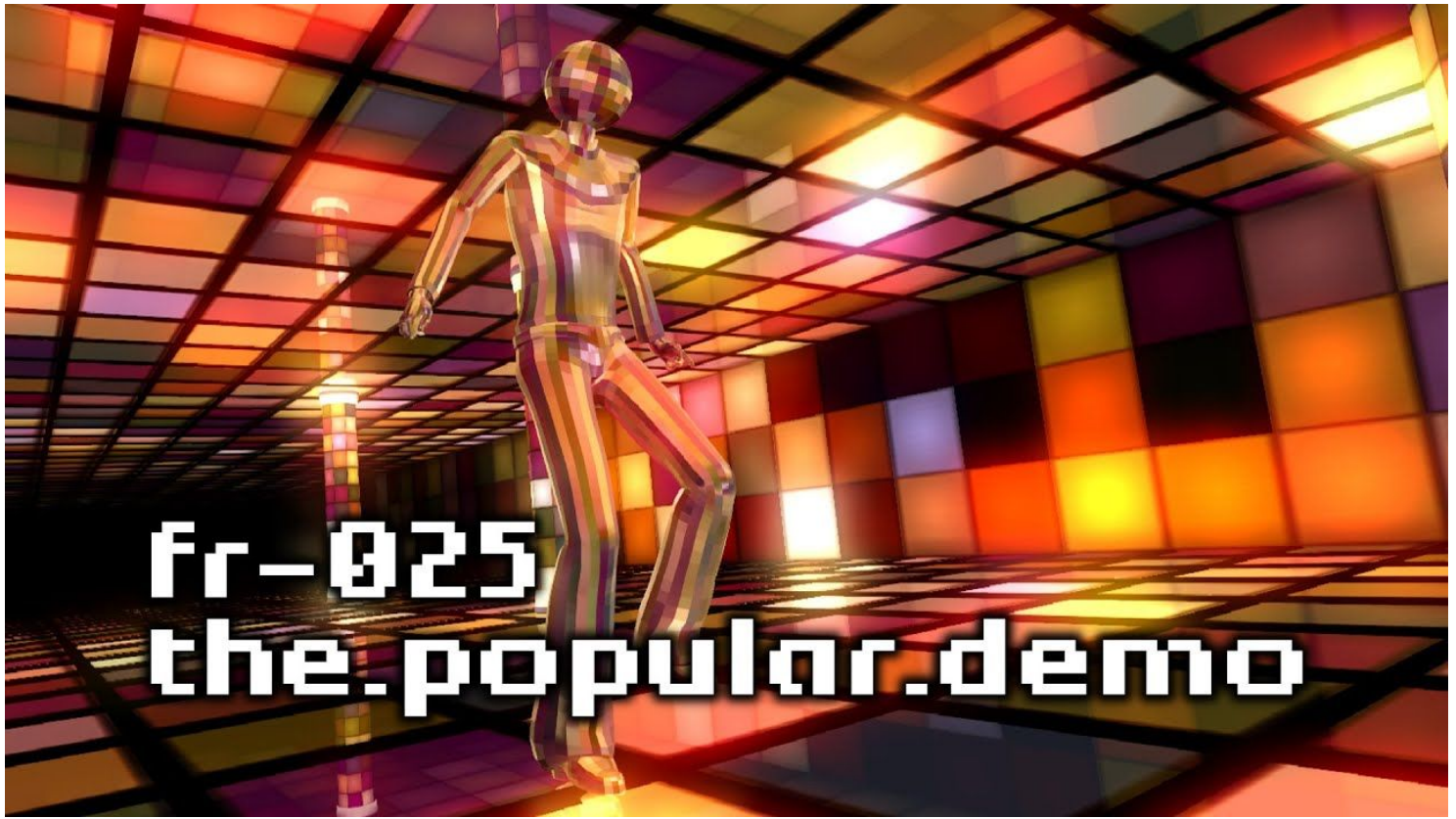


CLI Apps with Device Flow

Example: apps/cli

- Grant CLI app access to APIs on behalf of the user
- Can also be used by constraint IoT devices
- Common pattern for cloud tools (gcloud, az, etc.)
- CLI App / Device shows Code
- User enters code in special URL on a different device
- User authenticates on different device
- CLI App / Device, waits (polling) for auth completion
- CLI App / Device obtains an access token

Spring Boot Device Flow Demo



Spring Labs

- The **student folder** contains the following **labs**
  - **lab301**_spring_api_resource_server
  - **lab302**_spring_web_oauth_client
- Follow the **instructions** in the **readme** files
- We'll will do this lab together

Part 4: Customizing Keycloak

Keycloak Extensibility Options

- Configuration
  - Realm Configuration
  - Server Configuration
- Themes
  - Login / Registration, Account Console, Emails, Admin Console
  - Pre-build Themes
  - Custom Themes
- Extensions
  - Custom Authentication Mechanisms, Custom User Stores, etc.
  - Pre-build Extensions (Community Extensions)
  - Custom Extensions (Java / JavaScript)

Keycloak Config as Code

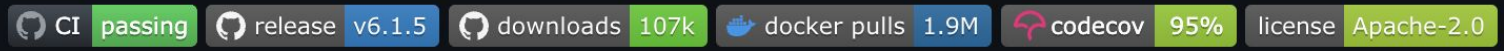
Keycloak Kubernetes Operator / Helm Charts

- Automatic Keycloak Deployment
- Keycloak Operator
  - Maintained by Keycloak Team (Guides)
  - Easy way to deploy Keycloak in Kubernetes
- Keycloak Helm Charts
  - codecentric Keycloak Helmchart
  - Bitnami Keycloak Helmchart

Keycloak Realm Config as Code

- Declarative Configuration enables GitOps
- Manage Realms, Clients, Identity Providers, etc.
- Many Tools available
  - JSON Export / Import
  - Keycloak Admin CLI (kcadm.sh)
  - Keycloak Admin REST API (OpenAPI Schema Support)
  - Keycloak Config CLI (YAML or JSON configuration)
  - Keycloak Terraform Provider (Terraform, also works with OpenTOFU)
  - Pulumi Keycloak Provider
- Some tools also support edits via Admin UI

README | Code of conduct | Apache-2.0 license



keycloak-config-cli

keycloak-config-cli is a Keycloak utility to ensure the desired configuration state for a realm based on a JSON/YAML file. The format of the JSON/YAML file based on the export realm format. Store and handle the configuration files inside git just like normal code. A Keycloak restart isn't required to apply the configuration.

Config files

The config files are based on the keycloak export files. You can use them to re-import your settings. But keep your files as small as possible. Remove all UUIDs and all stuff which is default set by keycloak.

moped.json is a full working example file you can consider. Other examples are located in the test resources.

Keycloak Config CLI

```yaml
realm: acme-client-examples
displayName: "Acme Client Examples ($(env:KC_STAGE:-default))"
enabled: true
clients:
  - clientId: acme-client-spa-app
    protocol: openid-connect
    name: Acme SPA Frontend App
    description: "JavaScript based Single-Page App as Public Client that uses Authorization Code Grant Flow"
    enabled: true
    publicClient: true
    standardFlowEnabled: true
    directAccessGrantsEnabled: false
    # Show client in account-console
    alwaysDisplayInConsole: true
    serviceAccountsEnabled: false
    fullScopeAllowed: false
    rootUrl: "https://www.keycloak.org/app"
    baseUrl: "/#url=https://id.acme.test:8443/auth&realm=acme-client-examples&client=acme-client-spa-app"
    adminUrl: ""
    redirectUris:
      - "/*"
      - "https://flowsimulator.pragmaticwebsecurity.com"
    webOrigins:
      - "+"
    defaultClientScopes:
      - "email"
      - "profile"
    optionalClientScopes:
      - "phone"
    attributes:
      "pkce.code.challenge.method": "S256"
      "post.logout.redirect.uris": "+"
```

Configuration (Yaml / JSON)

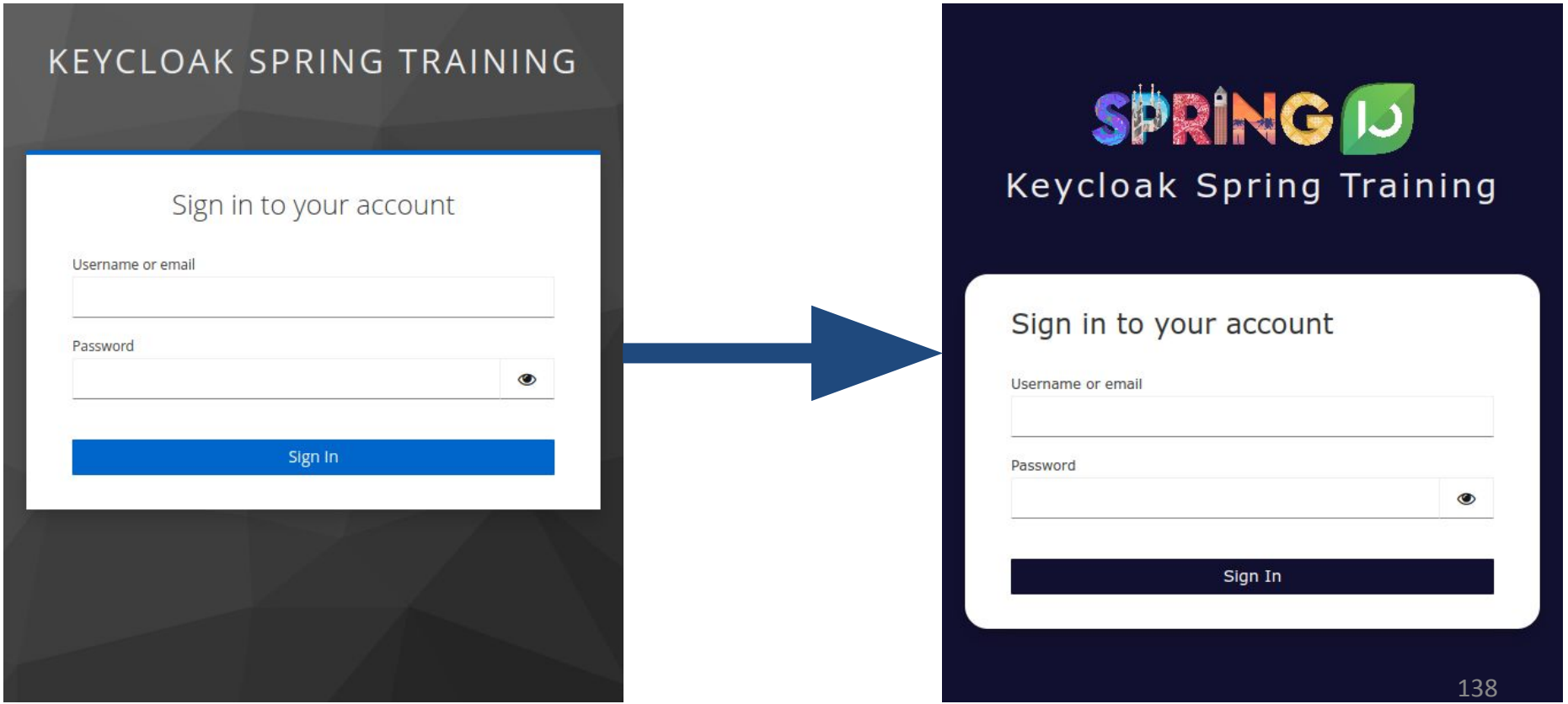
apply-config.sh

```bash
#!/usr/bin/env bash
set -e

script_dir=$(cd "$(dirname "$0")"; pwd)
kc_env=${KC_STAGE:-dev}
kc_cli_home=${kc_cli_home:-""}

#CLI_VERSION=v4.7.0-15.1.1
# Add for remote debugging
# -e JAVA_TOOL_OPTIONS="-agentlib:jdwp=transport=dt_socket,server=y,suspend=n,address=*:5005" \
# -p 5005:5005 \

# Source all variables from keycloak and secret files
set -o allexport
source $script_dir/stages/$kc_env/keycloak.env
source $script_dir/stages/$kc_env/secrets.env
set +o allexport

# resolve secrets

# Run Keycloak Config CLI
java -jar keycloak-config-cli-26.1.0.jar \
  --keycloak.url=$KEYCLOAK_URL \
  --keycloak.client-id=keycloak-config-cli \
  --keycloak.client-secret="$KEYCLOAK_CLIENT_SECRET" \
  --keycloak.grant-type=client_credentials \
  --keycloak.availability-check.enabled=true \
  --keycloak.availability-check.timeout=120s \
  --import.files.locations='./target/*' \
  --import.var-substitution.enabled=true \
  --import.validate=true \
  --import.cache.enabled=true
```

Run (Java / Docker / Native)

Keycloak Themes

Keycloak Theme Example



Keycloak Theming Mechanism

- Freemarker Templates
  - Built-in themes ($KEYCLOAK_HOME/lib/lib/main/org.keycloak.keycloak-themes-$KC_VERSION.jar)
  - Folder-based themes (discovered in $KEYCLOAK_HOME/themes)
  - .jar-based themes (discovered in $KEYCLOAK_HOME/providers)
- Slots for Themes
  - login → Login pages
  - account → Account Console
  - email → Emails
  - admin → Admin Console
- Two Deployment Variants
  - Standalone (Folder)
  - Packaged with extension in .jar

Theme Structure

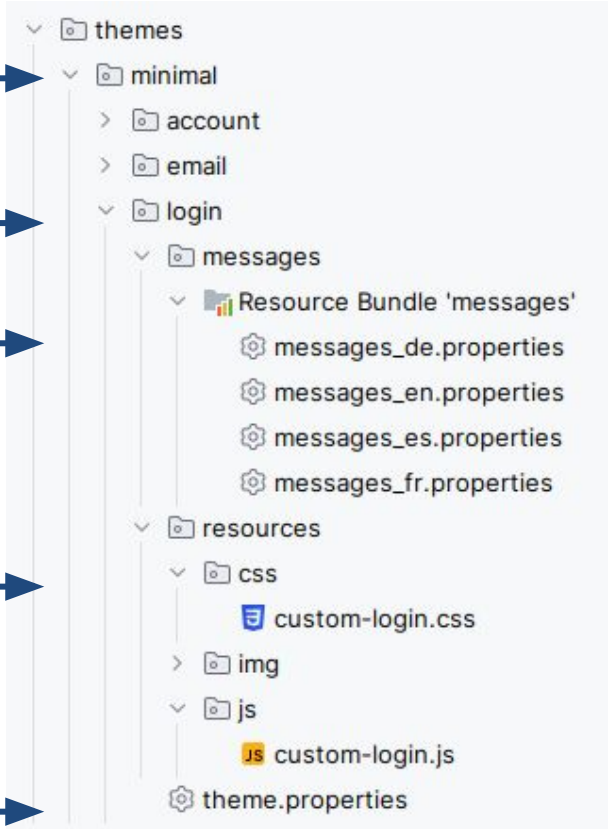
Theme

Type

Messages

Resources

Descriptor



Inherit templates from “keycloak” parent theme.

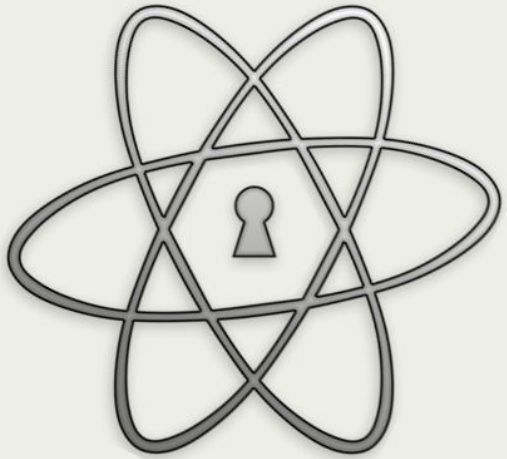
Use resources from “common” folder in “keycloak” theme.

```
parent=keycloak
import=common/keycloak
styles=css/login.css css/custom-login.css
scripts=js/custom-login.js
```

Keycloak theming

For the modern web.

Customize the look and feel of your login and registration pages without having to mess with FreeMarker.



Keycloak Extensions

Keycloak Extension Mechanism

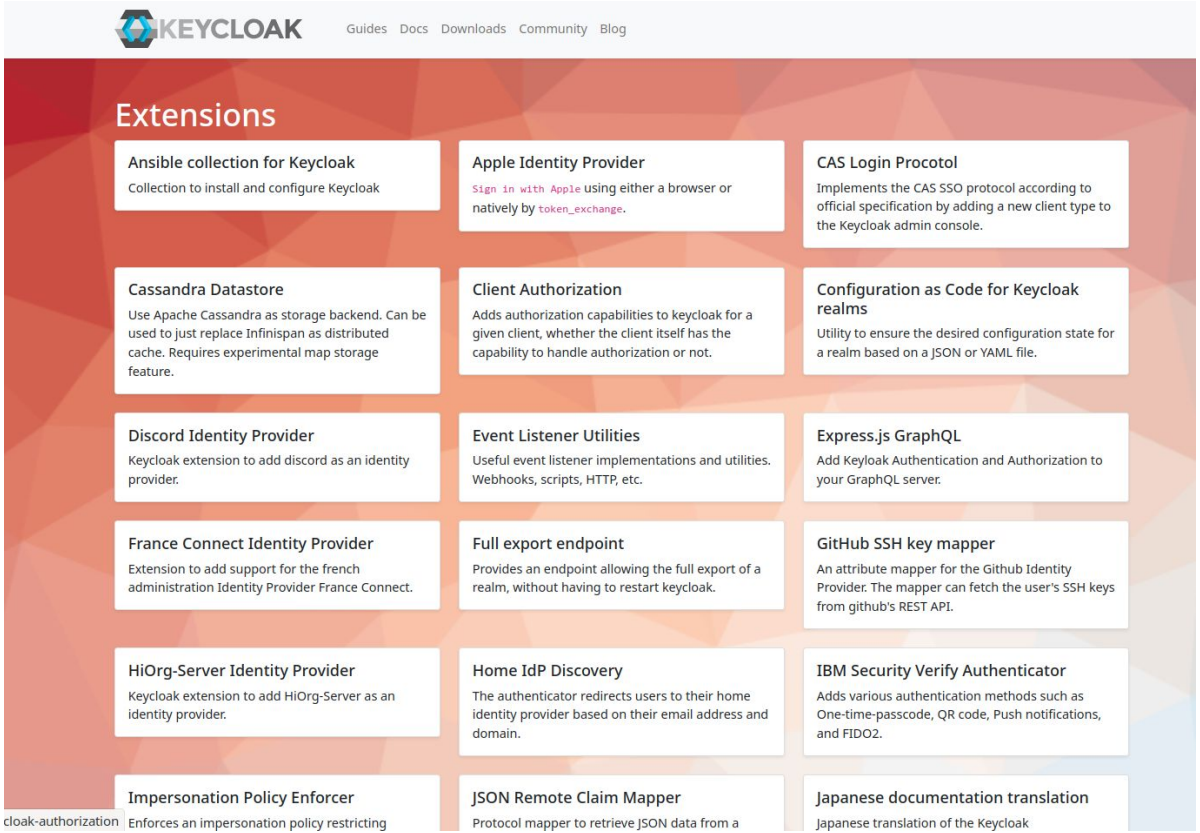
- Allows to extend Keycloak functionality with custom code
- Service Provider Interfaces (SPI)
  - SPI = “Extension Point”
  - **Extension** = **`Provider`** + **`ProviderFactory`** + **`Service Manifest`** + **`Packaging`**
  - Based on Java Service Loader Mechanism (since Java 6)
  - Each SPI has a unique “*SPI Name*” and a `ProviderFactory` has a “*Provider ID*”
  - Many SPIs built-in ~ 120 Service Provider Interfaces (SPIs) + ~ 300 Providers
- Keycloak Extensions Discovery
  - One or more extensions can be bundled in a **`.jar`**-File
  - Keycloak scans **`$KEYCLOAK_HOME/providers/`** at startup

Keycloak SPI Overview (Keycloak 26.1.4)

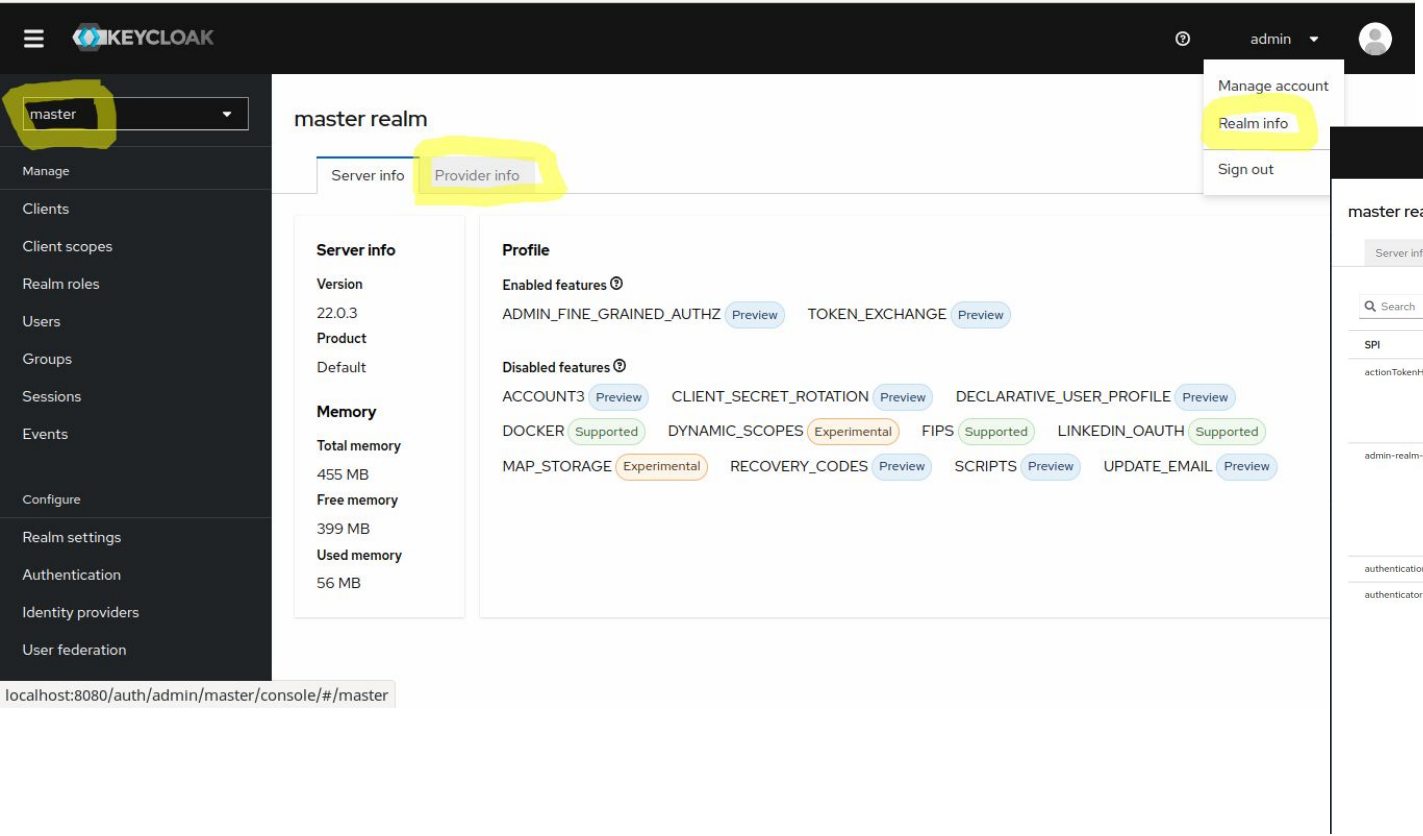
Keycloak Extension Highlights

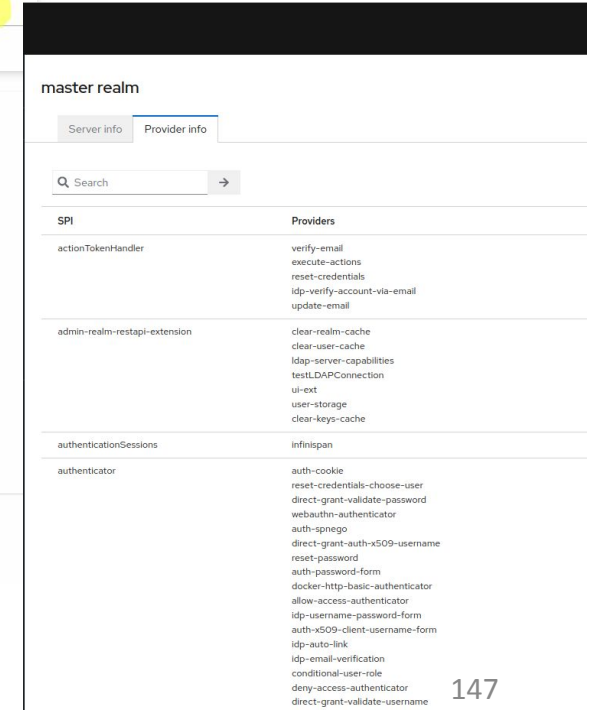
- Authenticators
  - Your custom Login and Registration logic
- Protocol Mappers for OIDC / SAML
  - Add custom information to OAuth Token claims / SAML Assertions
- Required Actions
  - Require users to complete a task after Login (Complete Profile, Terms & Conditions)
- Event Listener
  - Custom User Event Handling (Auditing, Provisioning)
- User Storage
  - Integrate User Accounts from existing User Stores (JDBC, REST, etc.)
- Credential Hashing Mechanisms
  - Leverage your existing Password Hashing Scheme to ease Migration from Legacy Systems

Keycloak Community Extensions



List Installed SPI Providers in Admin UI





SPI Example: AuthenticatorSpi

```java
package org.keycloak.authentication;

import org.keycloak.provider.Provider;
import org.keycloak.provider.ProviderFactory;
import org.keycloak.provider.Spi;

/** @author <a href="mailto:[illegible]@redhat.com">Stian Thorgersen</a> */
public class AuthenticatorSpi implements Spi {

    public static final String SPI_NAME = "authenticator";

    @Override
    public boolean isInternal() { return true; }

    @Override
    public String getName() { return SPI_NAME; }

    @Override
    public Class<? extends Provider> getProviderClass() { return Authenticator.class; }

    @Override
    public Class<? extends ProviderFactory> getProviderFactoryClass() { return AuthenticatorFactory.class; }

}
```

SPI class provided by Keycloak

SPI "name" used for lookups and configuration.

SPI Implementors need to implement this interface.
→ **Provider**

SPI Implementors need to implement this interface.
→ **Factory** to create **Provider** instances

Called whenever Provider is needed.

Provider Example: Authenticator

```java
@AutoService(AuthenticatorFactory.class)
class HelloAuthenticatorFactory implements AuthenticatorFactory {

    public String getId() { return "acme-auth-hello"; }

    public String getDisplayType() { return "Acme: Hello Auth"; }

    public String getHelpText() { return "Prints a hello message to the log"; }

    public Authenticator create(KeycloakSession session) { return new HelloAuthenticator(); }
```

Generates the **Service Manifest**

"**Provider ID**" used for lookups and configuration.

`create(..)` method called by Keycloak when Provider is needed.

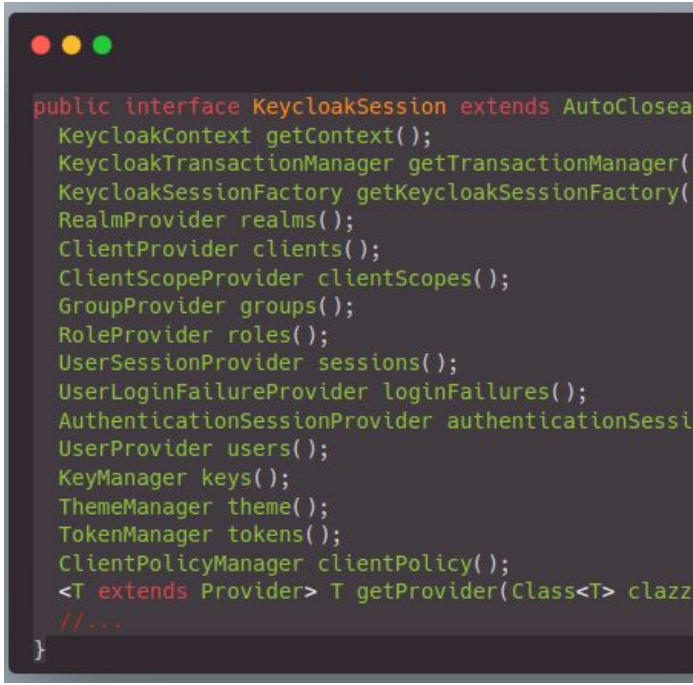
```java
class HelloAuthenticator implements Authenticator {

    public void authenticate(AuthenticationFlowContext context) {
        UserModel user = context.getUser();
        System.out.print("Hello " + user.getUsername());
        context.success();
    }

    public void action(AuthenticationFlowContext context) {}
```

`authenticate(..)` method handles authentication

`action(..)` method handles form submissions

KeycloakSession - Facade to Keycloak APIs

- Entrypoint to access Keycloak services
- Bound to current HTTP Request
- Provides access to KeycloakContext
  - Current Realm
  - Current Client
  - Current Request
  - Current AuthSession / User
- Usually passed as parameter
- Programmatic access:
  org.keycloak.utils.KeycloakSessionUtil#getKeycloakSession



```
public interface KeycloakSession extends AutoClosea
  KeycloakContext getContext();
  KeycloakTransactionManager getTransactionManager(
  KeycloakSessionFactory getKeycloakSessionFactory(
  RealmProvider realms();
  ClientProvider clients();
  ClientScopeProvider clientScopes();
  GroupProvider groups();
  RoleProvider roles();
  UserSessionProvider sessions();
  UserLoginFailureProvider loginFailures();
  AuthenticationSessionProvider authenticationSessi
  UserProvider users();
  KeyManager keys();
  ThemeManager theme();
  TokenManager tokens();
  ClientPolicyManager clientPolicy();
  <T extends Provider> T getProvider(Class<T> clazz
  //...
}
```

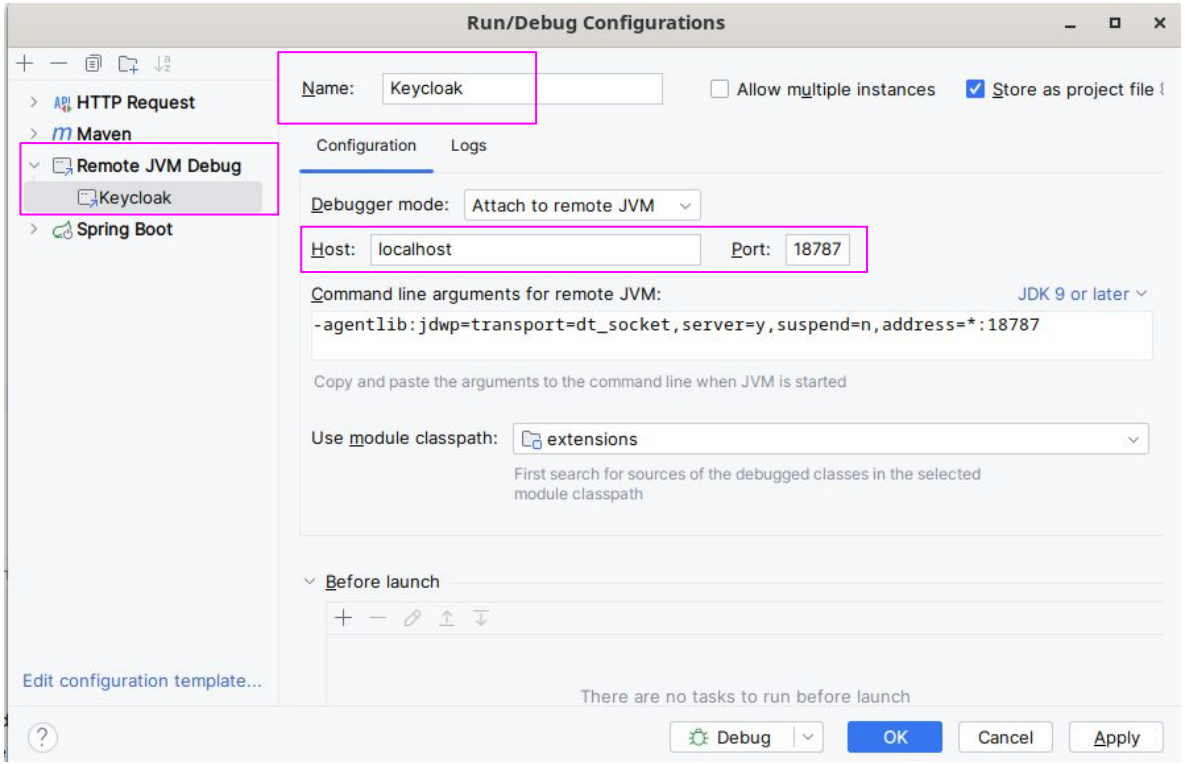
Keycloak Debugging

Enable Debugging

```
DEBUG: "true"
DEBUG_PORT: "*:18787"
```

or

start Keycloak
with **`--debug`**



How to create a new Extension?

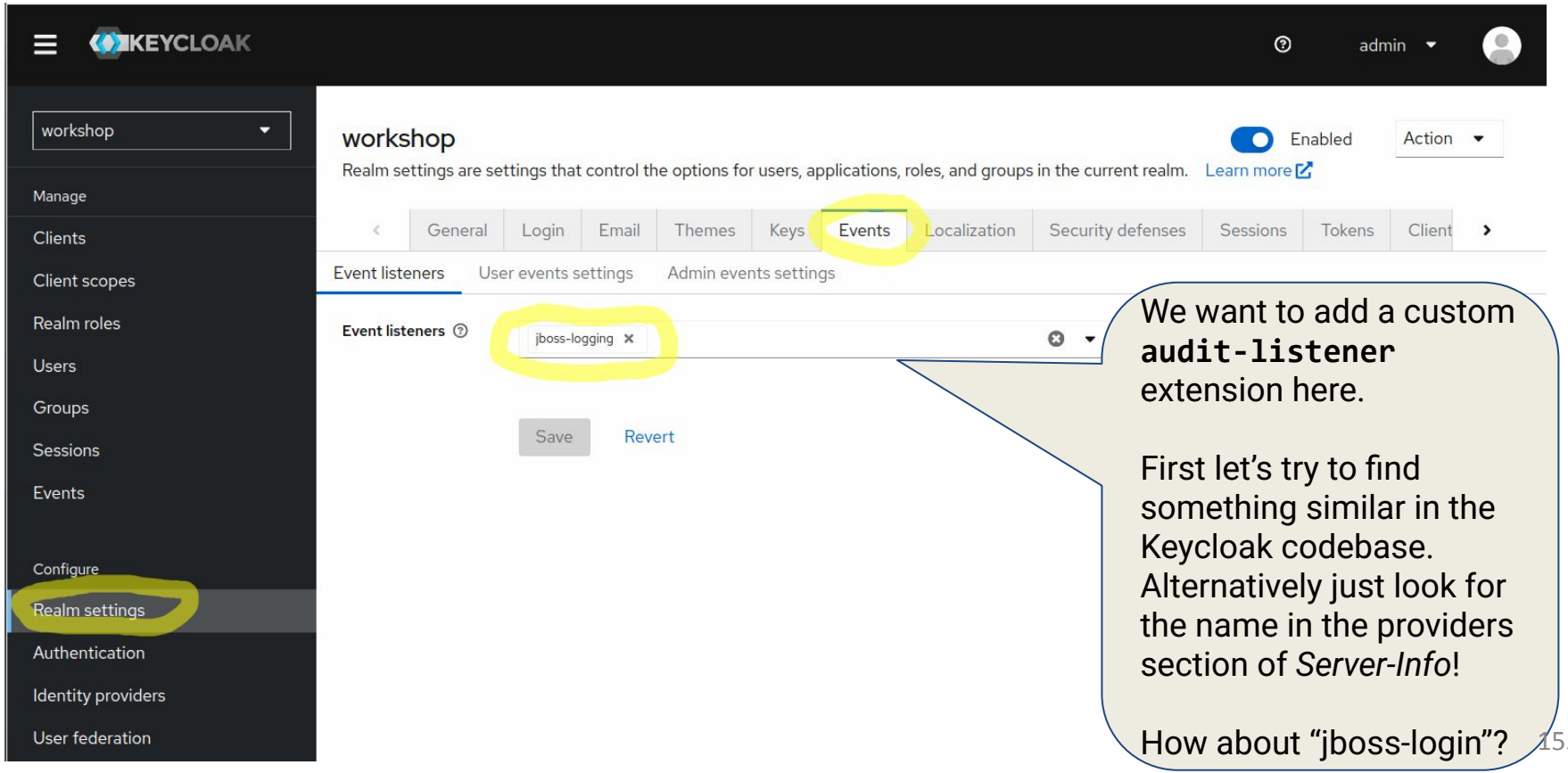
1. Take a look at the Keycloak Server Developer Documentation
2. Find proper SPI → `org.keycloak.provider.Spi`
3. Find out which Interfaces to implement
4. Implement Interfaces
5. Create or Generate a Service Manifest
6. Package Extension
7. Deploy Extension
8. Configure Extension

Example: AuditListener Extension

Use Case *“We want to forward relevant user events (e.g. login, logout, password changes) to an external audit-service.”*

AuditListener Extension Demo

How to start? Which SPI do we need?



```java
/.../

package org.keycloak.events.log;

import ...

/**
 * @author <a href="mailto:sthorger@redhat.com">Stian Thorgersen</a>
 */
public class JBossLoggingEventListenerProviderFactory implements EventListenerProviderFactory {

    public static final String ID = "jboss-logging";

    private static final Logger logger = Logger.getLogger("org.keycloak.events");

    private Logger.Level successLevel;
    private Logger.Level errorLevel;

    @Override
    public EventListenerProvider create(KeycloakSession session) {
        return new JBossLoggingEventListenerProvider(session, logger, successLevel, errorLevel);
    }

    @Override
    public void init(Config.Scope config) {
        successLevel = Logger.Level.valueOf(config.get("success-level", "debug").toUpperCase());
        errorLevel = Logger.Level.valueOf(config.get("error-level", "warn").toUpperCase());
    }
```

Custom Extension: AuditListener

- Spi org.keycloak.events.EventListenerSpi
- Implement Interfaces EventListenerProvider + EventListenerProviderFactory
- Create or generate Service Manifest File
  - File with FQCN* of SPI **`Factory` class** in **`src/main/resources/META-INF/services`**
  - This file contains the FQCN of the **`Factory`** implementation class
- Package and Deploy Extension
  - Build an extension **`.jar`**
  - Copy `extension.jar` to **`$KEYCLOAK_HOME/providers/`**
- Configure Extension
  - Configure `AuditListener` in `Admin UI -> Realm -> Events -> Config`

*) FQCN = Fully Qualified Class Name -> `com.company.domain.ClassName`

AuditListener Extension Dependencies

- Keycloak ships with many built-in libraries
  - $KEYCLOAK_HOME/lib
  - Sufficient for most extensions
- We use built-in Keycloak Dependencies
  - keycloak-server-spi
  - keycloak-server-spi-private
  - keycloak-services
  - → Scope “provided”

```
<dependencies>

    <dependency>
        <groupId>org.keycloak</groupId>
        <artifactId>keycloak-server-spi</artifactId>
        <version>${keycloak.version}</version>
        <scope>provided</scope>
    </dependency>

    <!-- Required for Event Listener SPI -->
    <dependency>
        <groupId>org.keycloak</groupId>
        <artifactId>keycloak-server-spi-private</artifactId>
        <version>${keycloak.version}</version>
        <scope>provided</scope>
    </dependency>

    <dependency>
        <groupId>org.keycloak</groupId>
        <artifactId>keycloak-services</artifactId>
        <version>${keycloak.version}</version>
        <scope>provided</scope>
    </dependency>
```

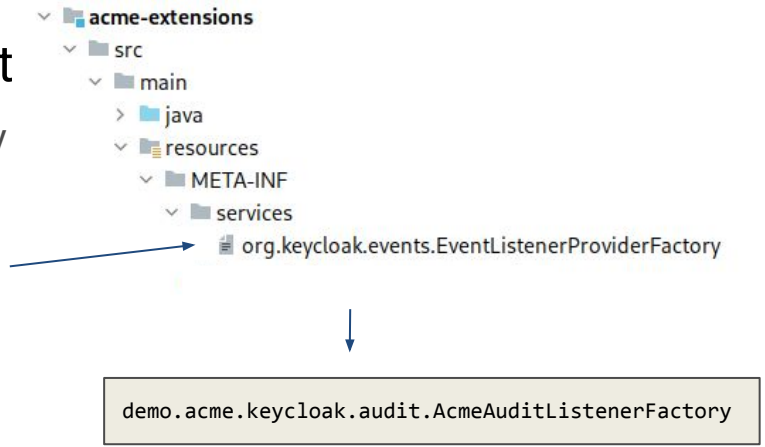
AuditListener Extension Implementation Sketch

```java
import org.keycloak.Config;
import org.keycloak.events.*;
import org.keycloak.models.*;

public class AcmeAuditListenerFactory implements EventListenerProviderFactory {

    public static final String ID = "acme-audit-listener";

    private static final AcmeAuditListener INSTANCE = new AcmeAuditListener();

    @Override
    public String getId() { return ID; }

    @Override // return singleton instance, or create new AcmeAuditListener(session)
    public EventListenerProvider create(KeycloakSession session) { return INSTANCE; }

    @Override // we could read settings from keycloak.conf / CLI / ENV
    public void init(Config.Scope config) { /* configure factory */ }

    @Override // we could init our provider with information from other providers
    public void postInit(KeycloakSessionFactory factory) { /* post-process factory */ }

    @Override // close resources if necessary
    public void close() { /* release resources if necessary */ }
}
```

```java
import org.keycloak.events.*;
import org.keycloak.events.admin.*;

public class AcmeAuditListener implements EventListenerProvider {

    @Override
    public void onEvent(Event event) {
        // called for each UserEvent's
    }

    @Override
    public void onEvent(AdminEvent event, boolean includeRep) {
        // called for each AdminEvent's
    }

    @Override
    public void close() {
        // called after component use
    }
}
```

AuditListener Extension Packaging Config

- Keycloak Extension Service Manifest
  - Needed by Keycloak for Extension Discovery
  - File on classpath in `META-INF/services`
  - Contains the FQCN of the Implementation



- Write by hand or generate
  - Annotation Processing Tools like [auto-service] can generate the service manifest file for you!

```
@AutoService(EventListenerProviderFactory.class)
public static class Factory implements EventListenerProviderFactory {
```

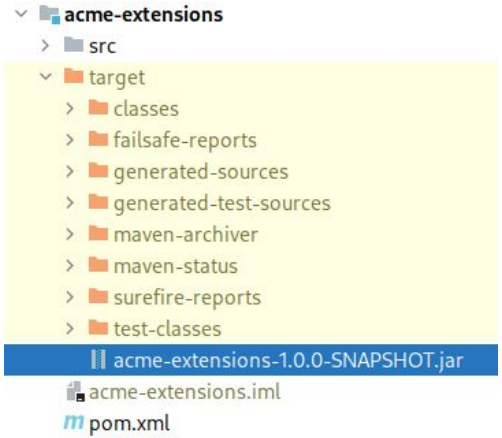
```
com.thomasdarimont.training.keycloak.event.AcmeAuditListener$Factory
```

AuditListener Extension Packaging

- Extensions can be packaged as `.jar`

- Custom libraries are possible with .jar, e.g. as...
  - Library repackaged in a “shaded” uber-jar
    or Maven assembly plugin “jar with dependencies”
  - … or copied to $KEYCLOAK_HOME/providers

- We use a combined acme-extensions.jar
  - We only use Keycloak libraries and deploy multiple extensions in one .jar

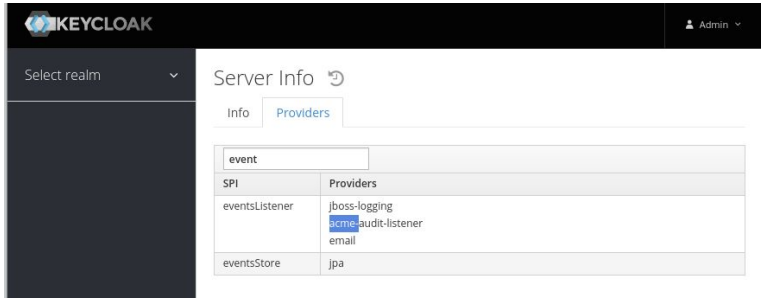


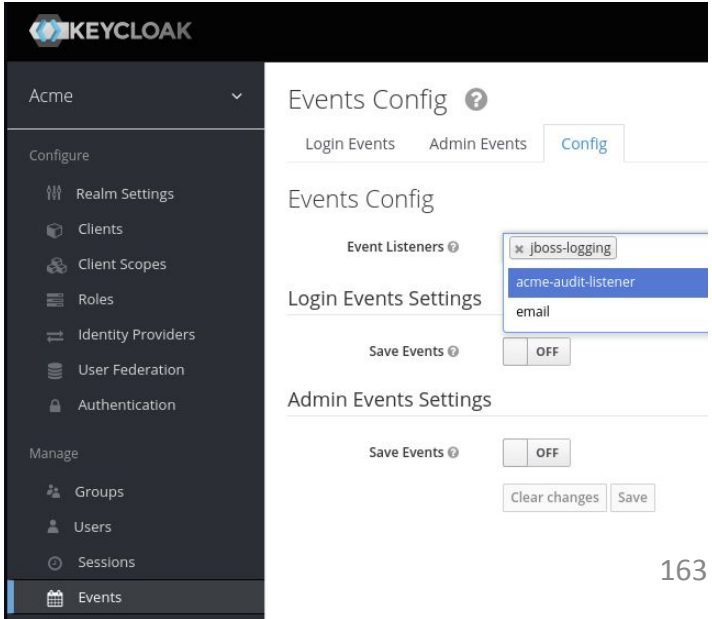
AuditListener Extension Deployment

- Simply copy the `.jar` to the Keycloak Providers folder
  - `cp acme-extensions.jar $KEYCLOAK_HOME/providers`
- Restart Keycloak Server

AuditListener Extension Configuration

- The `acme-audit-listener` Provider Id appears in the providers list
- We can select the `acme-audit-listener` in the Events Config



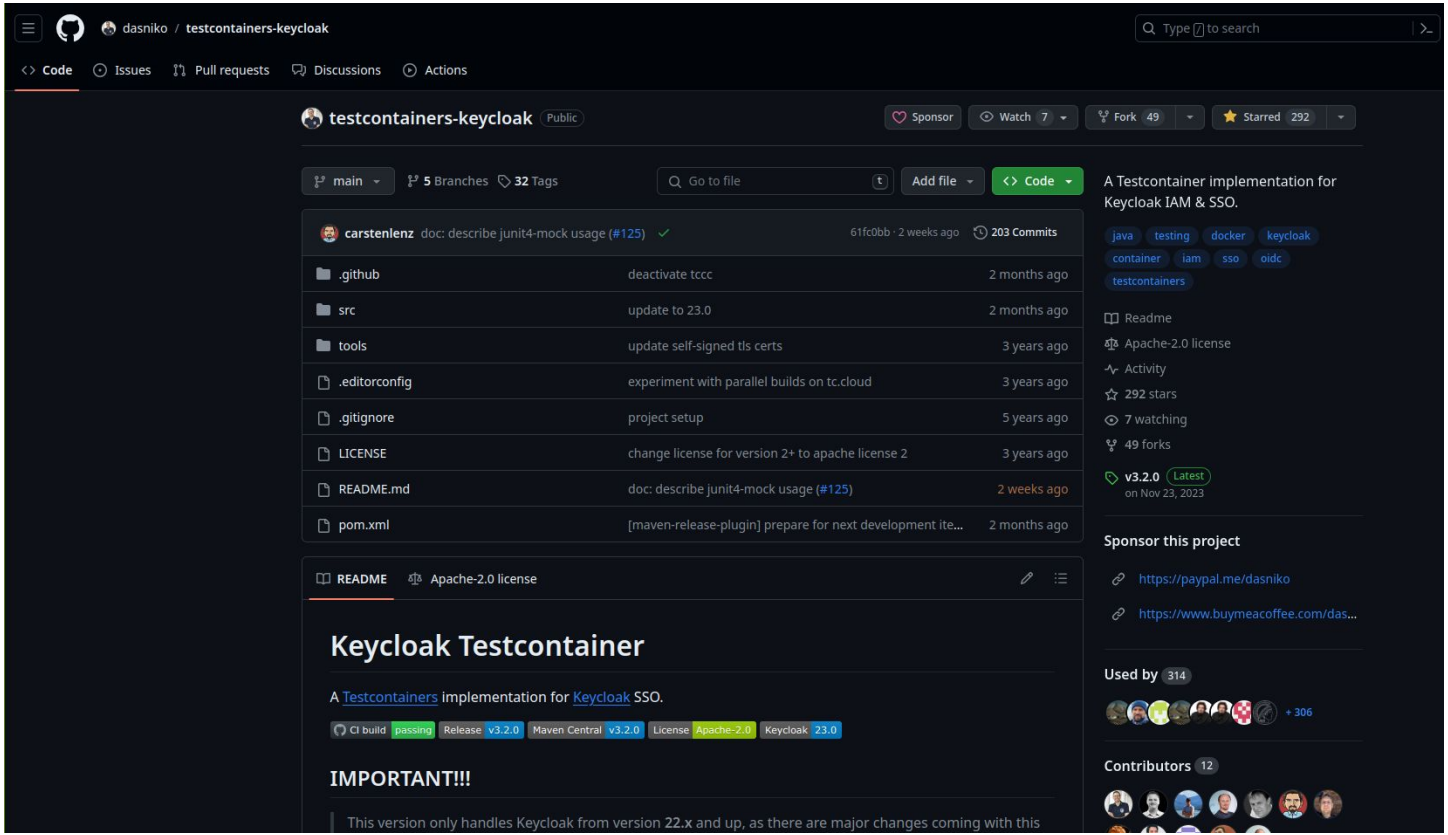


Testing Keycloak Extensions

- Testing Keycloak Extensions can be tricky…
  - You need start the Keycloak environment
  - Setup Keycloak with test data → e.g. Import predefined realm.json
  - Provide Stubs / Mocks for external services

- Several Test Options
  - Classic Unit-Tests with Keycloak API Mocks
  - Integration Tests with Testcontainers and Keycloak-Testcontainers
  - … or the new Keycloak Testing Framework
  - Stub APIs with WireMock or MockServer
  - UI / Acceptance Tests with Selenide, Cypress or Playwright

Testcontainers Keycloak



AuditListener Testing Implementation Sketch

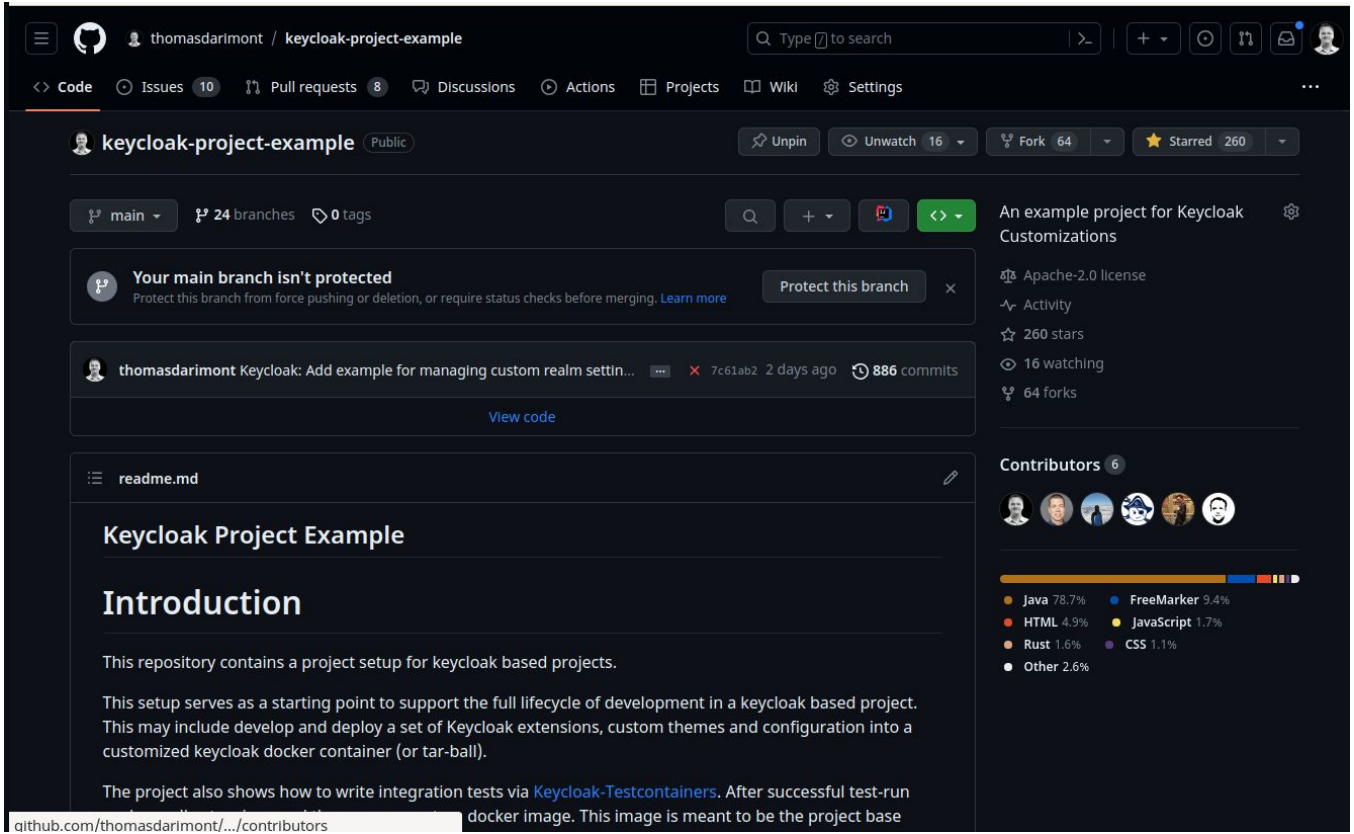
```xml
<dependency>
    <groupId>com.github.dasniko</groupId>
    <artifactId>testcontainers-keycloak</artifactId>
    <version>3.2.0</version>
    <scope>test</scope>
</dependency>
```

```
public static KeycloakContainer keycloak;

@BeforeAll
public static void beforeAll() throws Exception {

    keycloak = createKeycloakContainer(REALM_IMPORT_FILE);
    keycloak.withReuse(true);
    keycloak.start();
    keycloak.followOutput(new Slf4jLogConsumer(log));
}


 public static KeycloakContainer createKeycloakContainer(File realmImportFile) {
     …

     return new KeycloakContainer("quay.io/keycloak/keycloak:23.0.4");
 }
```

```
@Test
public void auditListenerShouldPrintLogMessage() throws Exception {

    ToStringConsumer consumer = new ToStringConsumer();
    keycloak.followOutput(consumer);

    TokenService tokenService = KeycloakTestSupport.getTokenService(keycloak);

    // trigger user login via ROPC
    tokenService.grantToken(ACME_REALM,
            new Form()
            .param("grant_type", "password")
            .param("username", "tester")
            .param("password", TEST_USER_PASSWORD)
            .param("client_id", TEST_CLIENT)
            .param("scope", "openid acme.profile acme.ageinfo")
            .asMap());

    // Allow the container log to flush
    TimeUnit.MILLISECONDS.sleep(750);

    // Check for audit log entry
    assertThat(consumer.toUtf8String()).contains("audit userEvent");
}
```

AuditListener Testing Demo

Keycloak Project Example (New)



Tip: Expose Extension Metadata in Admin UI

- Implement `ServerInfoAwareProviderFactory` interface
  - … with your ProviderFactory implementation
  - Allows to expose additional metadata in providers view in Server-Info
  - Version Information
  - Factory Configuration

```
public interface ServerInfoAwareProviderFactory {

    /** Return actual info about the provider. This
    Map<String, String> getOperationalInfo();

}
```

Tip: Encapsulate configuration Models

- Wrap generic configuration models in type-safe components

```
public class SmsAuthenticator implements Authenticator {
@Override
  public void authenticate(AuthenticationFlowContext authenticationFlowContext) {
    var config = new SmsAuthenticatorConfig(authenticationFlowContext.getAuthenticatorConfig());
    if (config.isSandboxMode()) {
    // ...
    }
...
```

Tip: Create your own SPIs

- Make Keycloak extensible your way with custom SPIs

```java
@AutoService(Spi.class)
public final class ScheduledTaskSpi implements Spi {

    public boolean isInternal() {
        return true;
    }

    public String getName() {
        return "acme-scheduled-task";
    }

    public Class<? extends Provider> getProviderClass() {
        return ScheduledTaskProvider.class;
    }

    public Class<? extends ProviderFactory<ScheduledTaskProvider>> getProviderFactoryClass() {
        return ScheduledTaskProviderFactory.class;
    }
}
```

```java
public interface ScheduledTaskProvider extends Provider {

    ScheduledTask getScheduledTask();

    long getInterval();

    String getTaskName();
}
```

```java
public abstract class ScheduledTaskProviderFactory implements ProviderFactory<ScheduledTaskProvider> {

    private KeycloakSessionFactory keycloakSessionFactory;

    public void postInit(KeycloakSessionFactory keycloakSessionFactory) {
        ...
    }

    public void close() {
        ...
    }
}
```

Tip: Use Custom Admin Resources

- If you want to add a custom Admin API implement `AdminRealmResourceProvider`

```
public class CustomAdminRealmResourceProvider implements AdminRealmResourceProvider {
  public CustomAdminRealmResourceProvider() {}

  public Object getResource(KeycloakSession session, RealmModel realm,
                            AdminPermissionEvaluator adminPermissionEvaluator,
                            AdminEventBuilder adminEventBuilder) {
    return new CustomAdminResource(session, realm, adminPermissionEvaluator, adminEventBuilder);
  }

  public void close() {}
}
```

Keycloak Labs

- The **student folder** contains the following **labs**
  - **lab401**_keycloak_theme
  - **lab402**_keycloak_extension
- Follow the **instructions** in the **readme** files
- We'll will do this lab together

Summary

- You learned the fundamentals of Keycloak
- You learned the OAuth2 / OpenID Connect basics
- You know how to secure Spring Applications
- You know the basics of Keycloak customizations

That’s quite an achievement!

Congratulations!

Well done folks!
Thank you very much!

@thomasdarimont

Bewertung des Schulungstages 2025