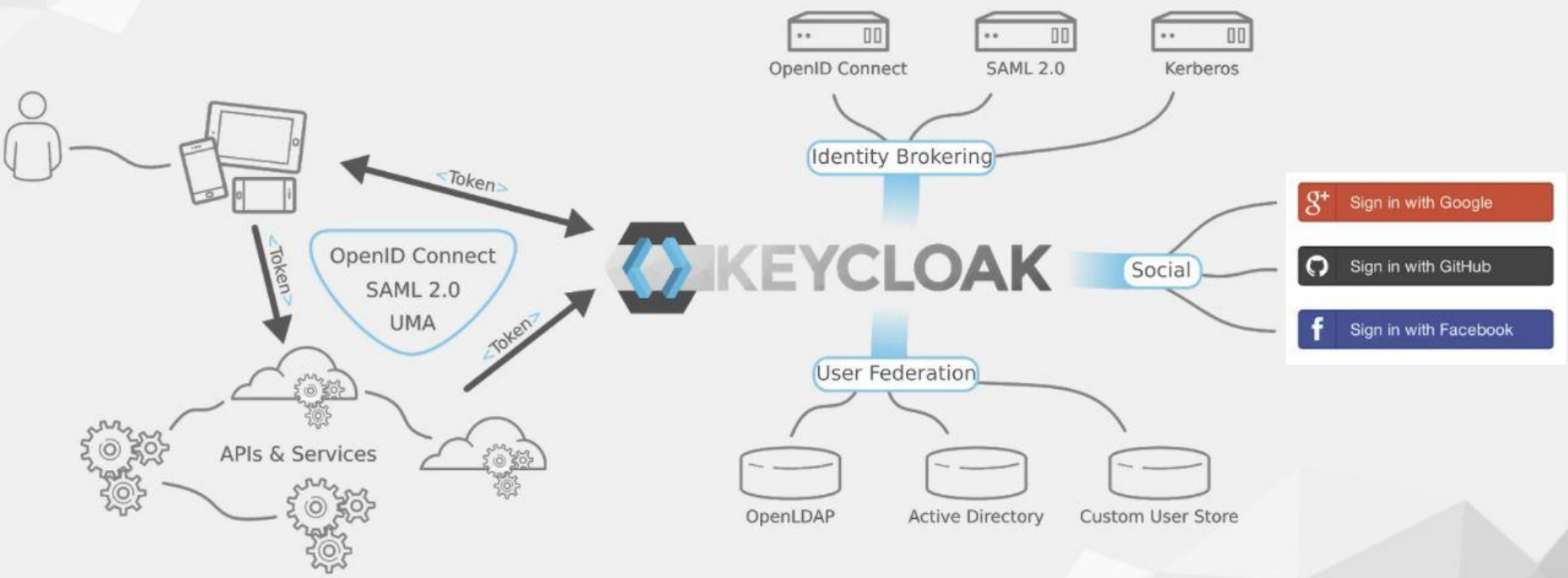
Keycloak

Open Source Identity & Access Management with Keycloak

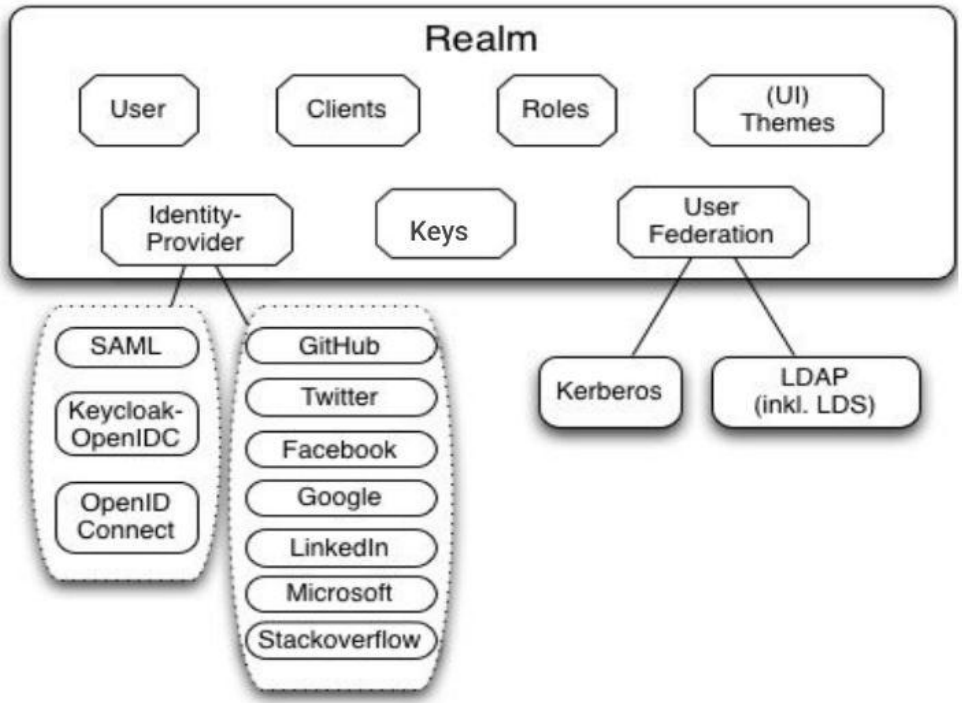
Authenticate users without code



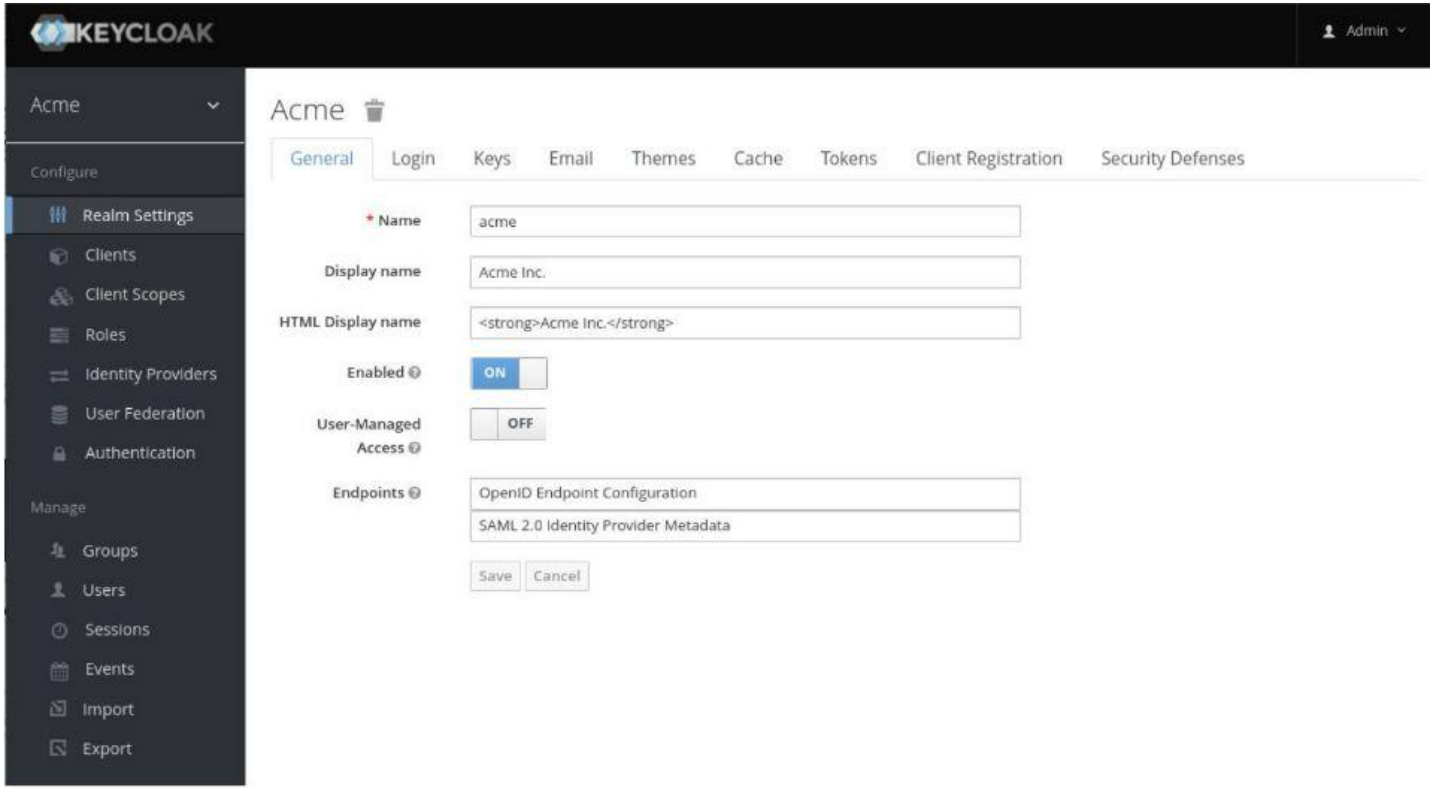
Features

- Single Sign-on and Single Sign-out
- Standard Protocols OAuth 2.0, OIDC 1.0, SAML 2.0
- Flexible Authentication and Authorization
- Social Login Google, Facebook, Twitter...
- Provides centralized User Management
- Customizable and Extensible
- Easy Setup and Integration

Main Concepts



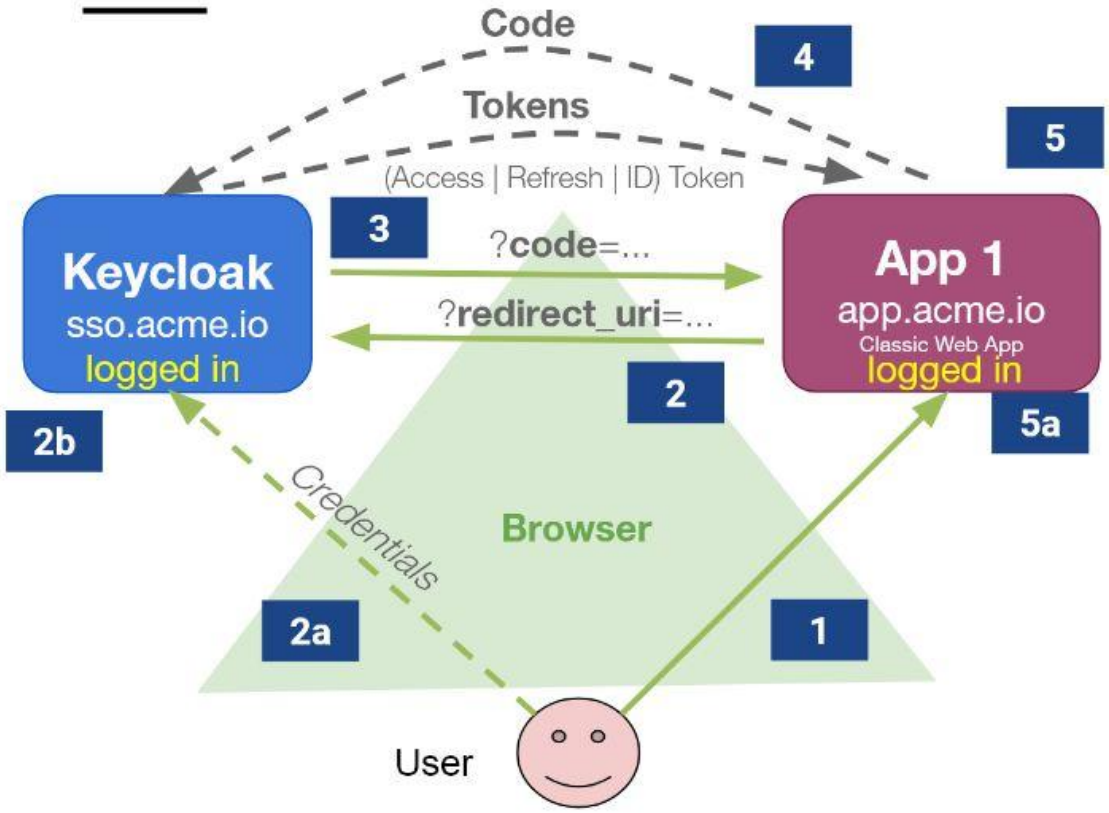
Admin Console



Single Sign-on

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

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

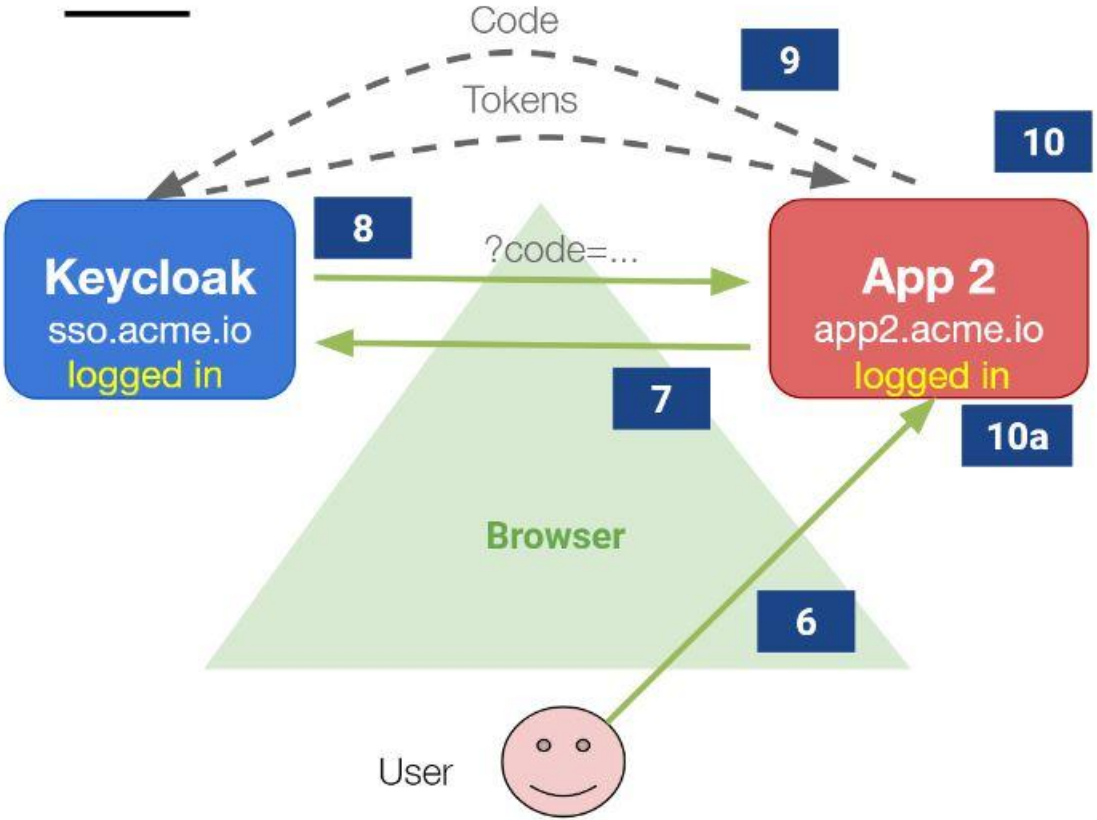
4 App **exchanges** *Code* to *Tokens* with Keycloak via separate Channel

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

Web SSO with OIDC: Authenticated User



...

6 Authenticated user **accesses** App 2

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

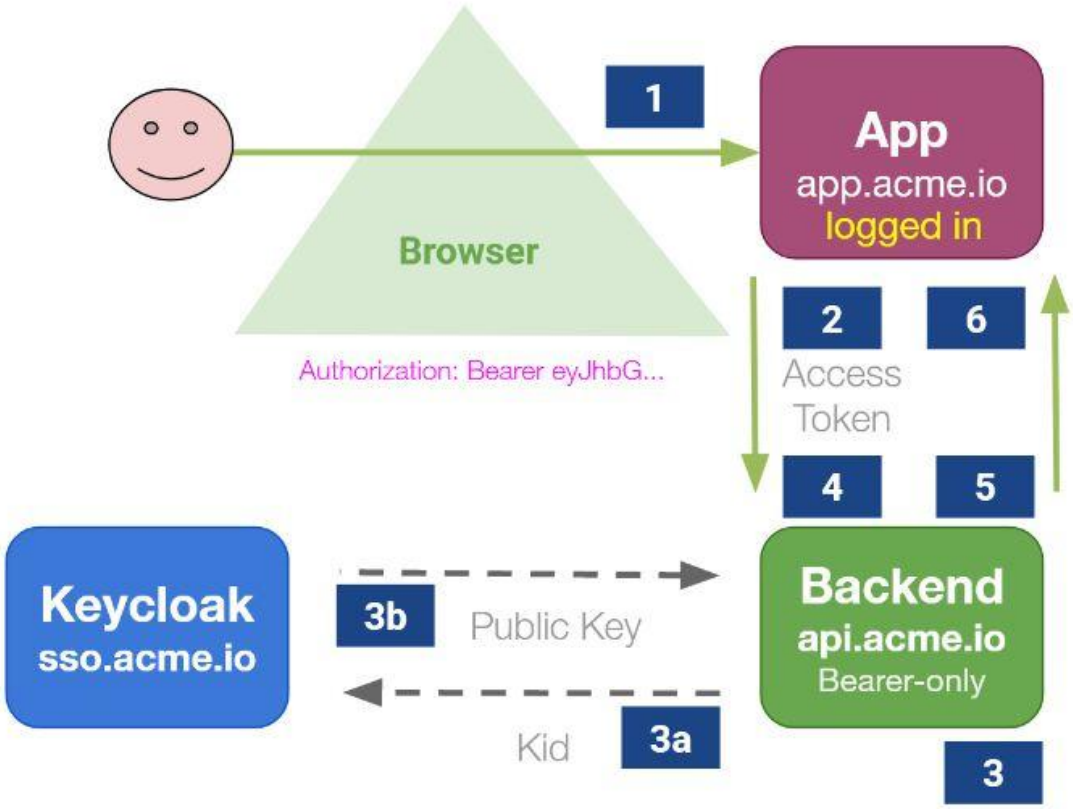
8 Keycloak **detects** *SSO Session*, **generates** *code*, **redirects** to App 2

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

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 Key* with *Kid* from Keycloak's JWKS endpoint
3b. Keycloak **returns** *Realm Public Key*
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

Relational Database Setup

Declare Your JDBC Drivers

```
  <subsystem xmlns="urn:jboss:domain:datasources:5.0">
     <datasources>
       ...
       <datasource jndi-name="java:jboss/datasources/KeycloakDS" pool-name="KeycloakDS"
enabled="true" use-java-context="true">
           <connection-url>jdbc:postgresql://localhost/keycloak</connection-url>
           <driver>postgresql</driver>
           <pool>
               <max-pool-size>20</max-pool-size>
           </pool>
           <security>
               <user-name>William</user-name>
               <password>password</password>
           </security>
       </datasource>
        ...
     </datasources>
  </subsystem>
```

OpenId Endpoints configuration


```
issuer: "http://localhost:8080/auth/realms/mlbd-keycloak-auth"
authorization_endpoint: "http://localhost:8080/auth/realms/mlbd-keycloak-auth/protocol/openid-connect/auth"
token_endpoint: "http://localhost:8080/auth/realms/mlbd-keycloak-auth/protocol/openid-connect/token"
introspection_endpoint: "http://localhost:8080/auth/realms/mlbd-keycloak-auth/protocol/openid-connect/token/introspect"
userinfo_endpoint: "http://localhost:8080/auth/realms/mlbd-keycloak-auth/protocol/openid-connect/userinfo"
end_session_endpoint: "http://localhost:8080/auth/realms/mlbd-keycloak-auth/protocol/openid-connect/logout"
jwks_uri: "http://localhost:8080/auth/realms/mlbd-keycloak-auth/protocol/openid-connect/certs"
check_session_iframe: "http://localhost:8080/auth/realms/mlbd-keycloak-auth/protocol/openid-connect/login-status-iframe.html"
grant_types_supported:
    0: "authorization_code"
    1: "implicit"
    2: "refresh_token"
    3: "password"
    4: "client_credentials"
    5: "urn:ietf:params:oauth:grant-type:device_code"
    6: "urn:openid:params:grant-type:ciba"
```

Email Login Flow with OpenId

Secure Spring Boot App with Keycloak

Secure a Frontend App with Keycloak

Resources

How to secure your Spring apps with Keycloak by Thomas Darimont

Keycloak core features and concepts
Authorization Services Guide
Server Administration Guide
Keycloak Admin REST API