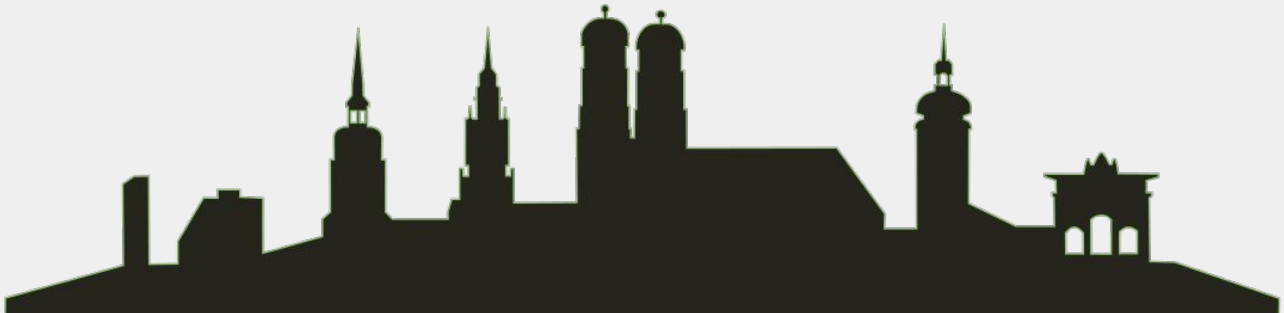
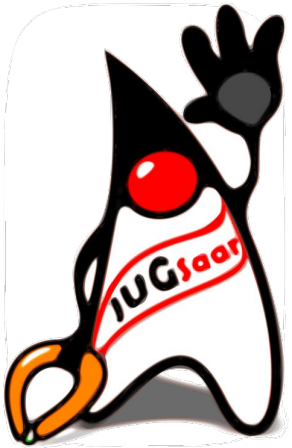
How to secure your Spring Apps with Keycloak

Thomas Darimont
@thomasdarimont



Thomas Darimont

- Software Architect >eurodata
- Spring Data Team Alumni
- Open Source Enthusiast
- Organizer Java User Group Saarland
- Keycloak Contributor for over 3 years



The Journey

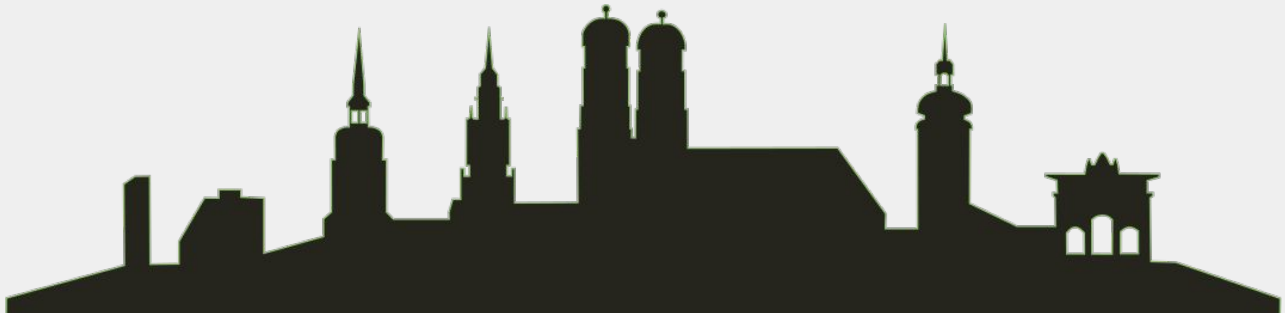

Keycloak

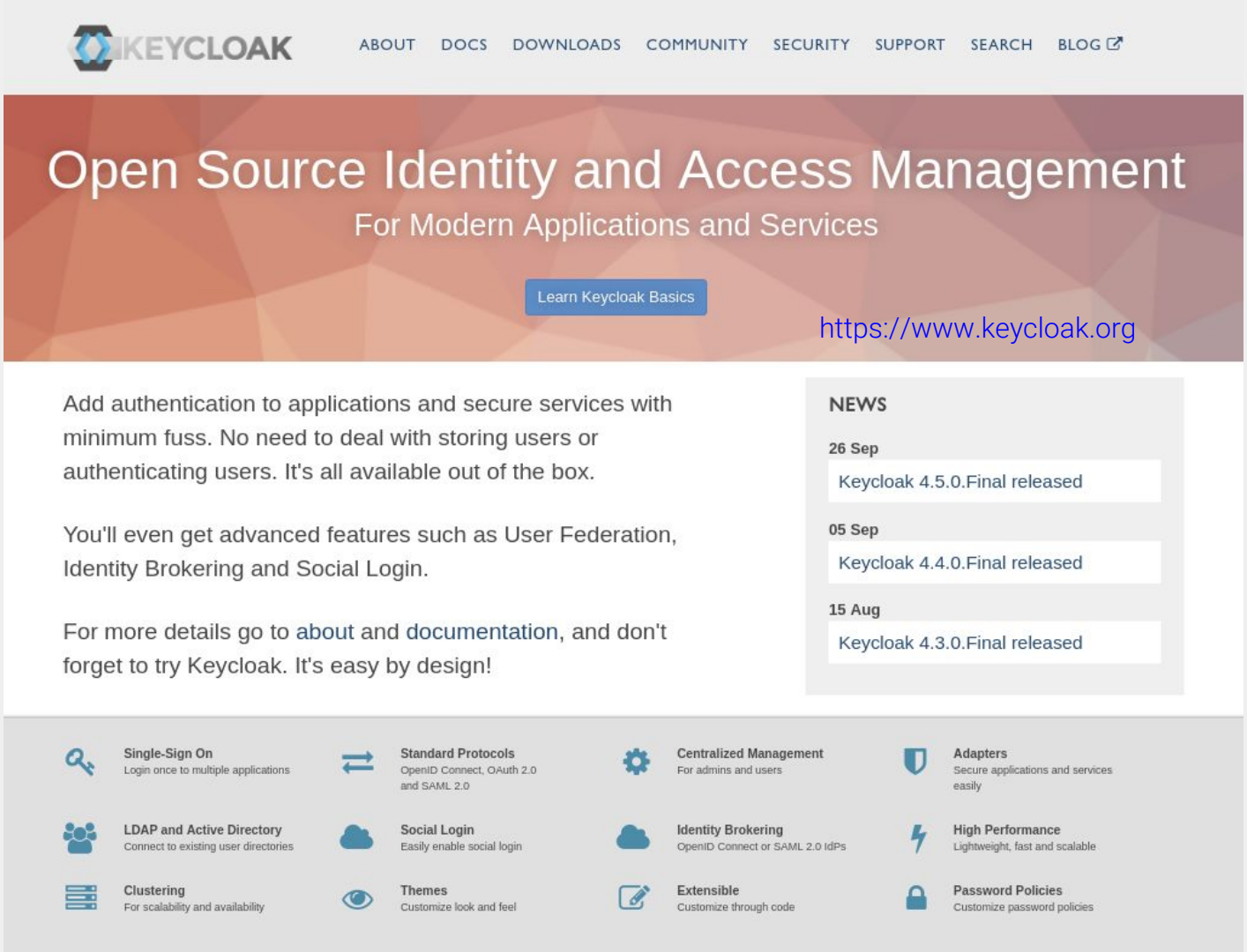

Single Sign-on


Securing Applications


Keycloak Extensions


Keycloak




KEYCLOAK
ABOUT
DOCS
DOWNLOADS
COMMUNITY
SECURITY
SUPPORT
SEARCH
BLOG
Open Source Identity and Access Management
For Modern Applications and Services
Learn Keycloak Basics
https://www.keycloak.org
Add authentication to applications and secure services with minimum fuss. No need to deal with storing users or authenticating users. It's all available out of the box.
You'll even get advanced features such as User Federation, Identity Brokering and Social Login.
For more details go to about and documentation, and don't forget to try Keycloak. It's easy by design!
NEWS
26 Sep
Keycloak 4.5.0.Final released
05 Sep
Keycloak 4.4.0.Final released
15 Aug
Keycloak 4.3.0.Final released
Single-Sign On
Login once to multiple applications
Standard Protocols
OpenID Connect, OAuth 2.0 and SAML 2.0
Centralized Management
For admins and users
Adapters
Secure applications and services easily
LDAP and Active Directory
Connect to existing user directories
Social Login
Easily enable social login
Identity Brokering
OpenID Connect or SAML 2.0 IdPs
High Performance
Lightweight, fast and scalable
Clustering
For scalability and availability
Themes
Customize look and feel
Extensible
Customize through code
Password Policies
Customize password policies

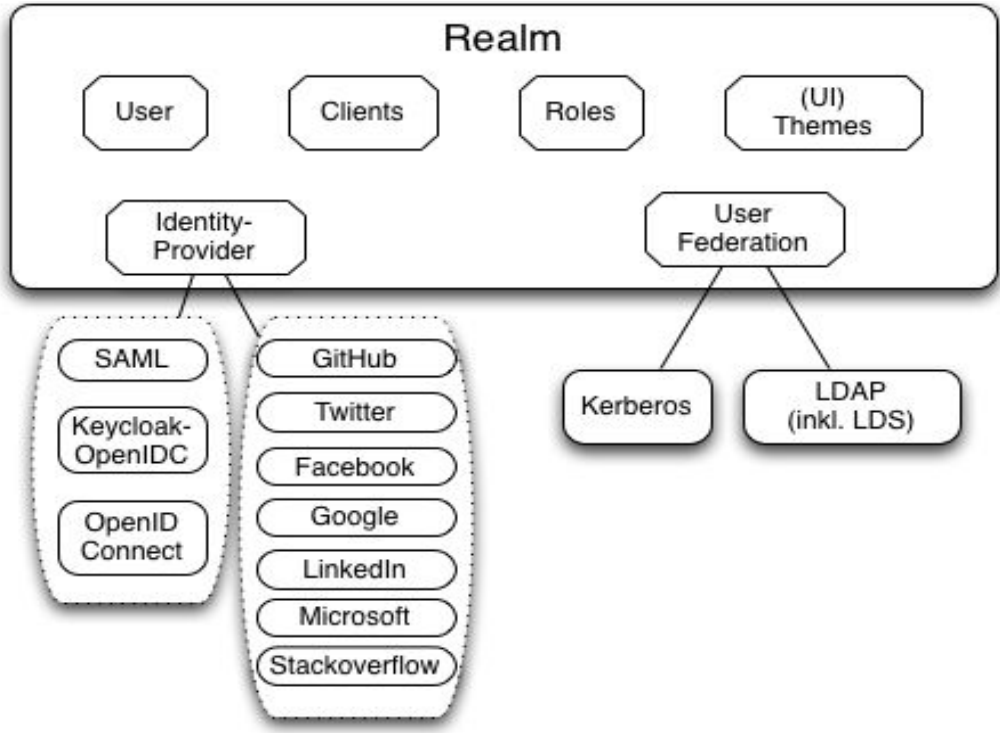
Project

- Open Source **Identity and Access Management**
- **Red Hat** Developers, Apache Licensed
- Versions **3.4.3.Final, 4.5.0.Final** ~ **every 6 Weeks**
- Since 2013, broad adoption since 2015
- Vital **Community**: **283+** Contributors **1.595+** Forks
- Very **robust**, good **documentation**, many **examples**
- Commercial Offering available **Red Hat SSO**

Features

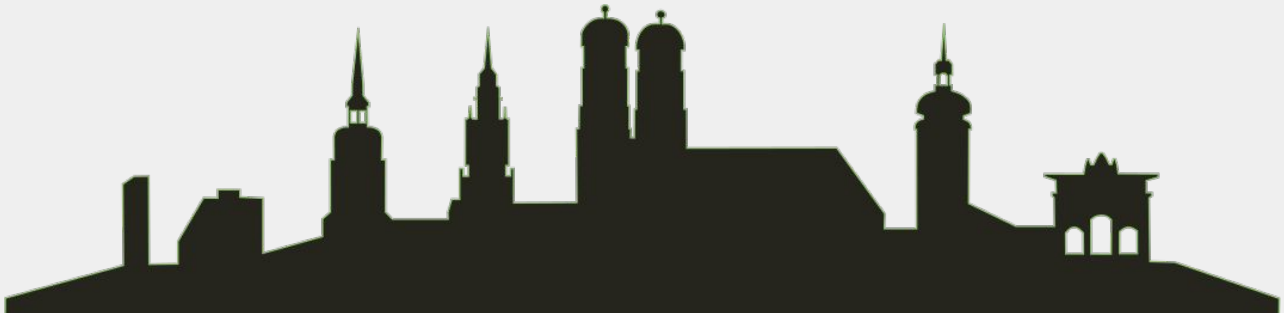
- **Single Sign-on** and Single Sign-out
- Flexible **Authentication** and **Authorization**
- **Standard Protocols** OAuth 2.0, OIDC 1.0, SAML 2.0, Docker Auth
- **Multi-Factor Authentication** One-time Password
- **Social Login** Google, Facebook, Twitter,...
- Provides centralized **User Management**
- Supports **Directory Services**
- **Customizable** and **Extensible**
- **Easy** Setup and Integration

Main Concepts

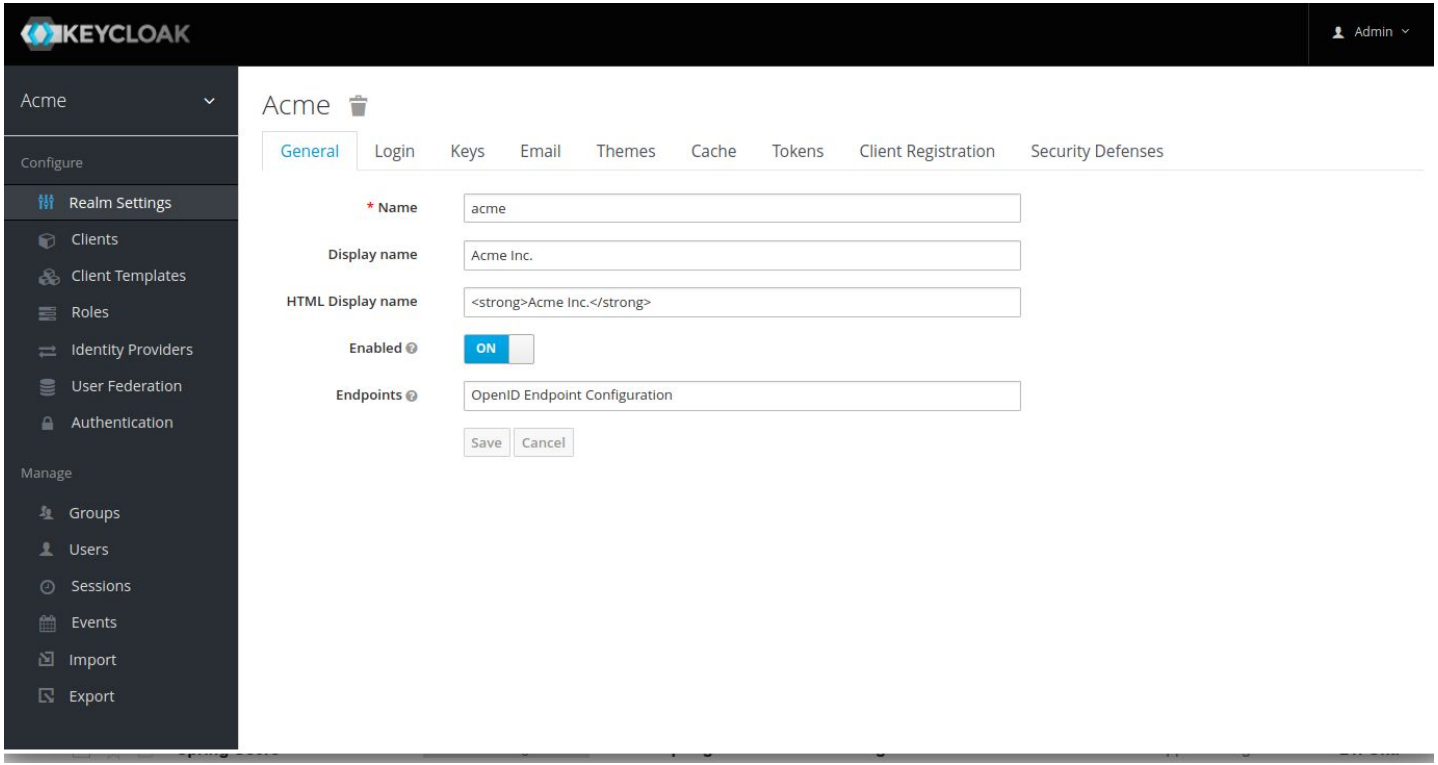


Keycloak Admin Console

Admin Console Login



Admin Console



Technology Stack 4.5.0.RELEASE

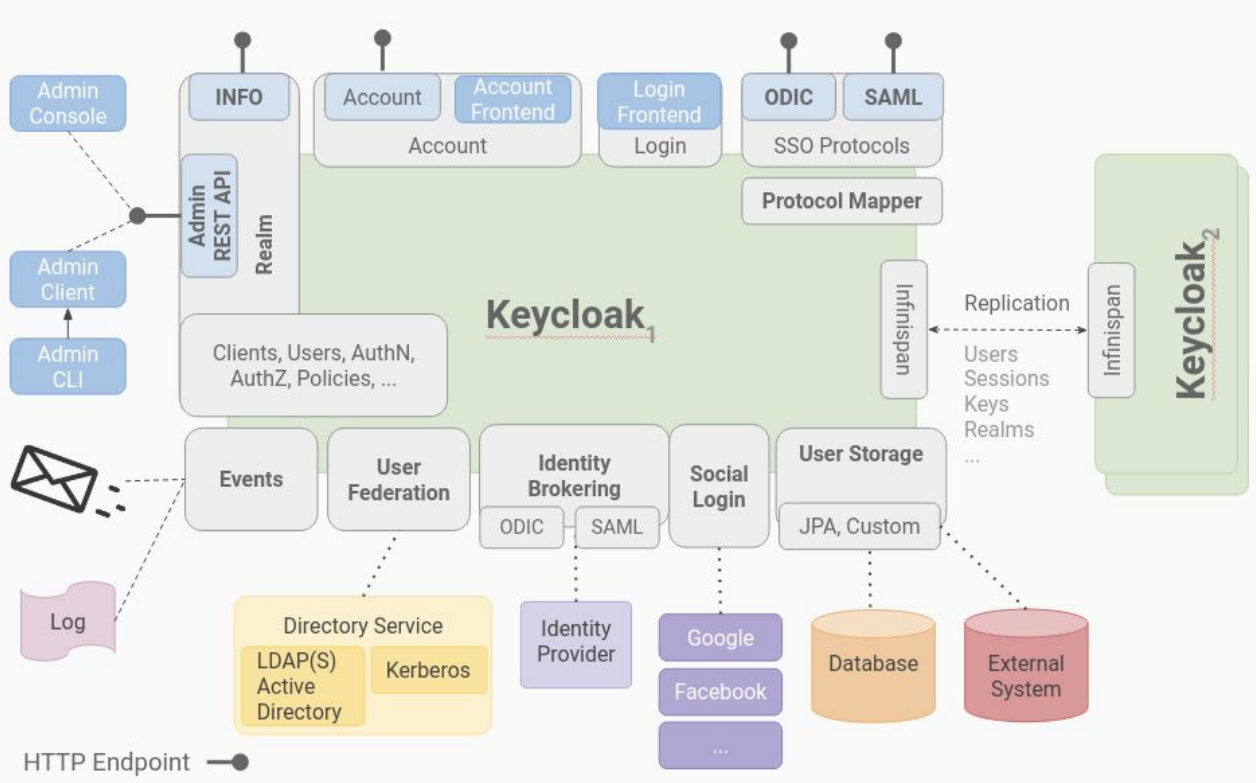
Admin Console

- Angular JS (1.6.10)
- PatternFly
- Bootstrap

Keycloak Server

- Wildfly 14.0.1.x
- JAX-RS (Resteasy)
- JPA (Hibernate)
- Infinispan (JGroups)
- Freemarker
- Jackson 2.x
- JBoss Logging
- Apache Directory API
- Commons HTTP Client

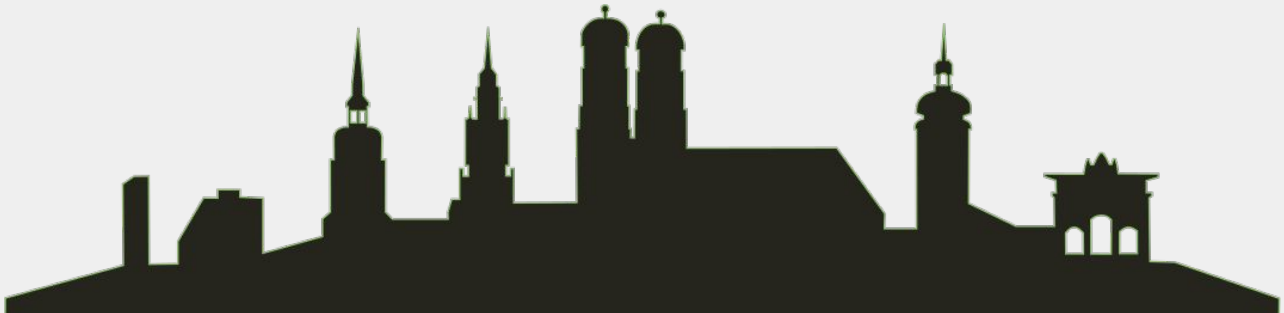
Server Architecture



Authentication & Authorization

- Authentication (AuthN)
  - Determines ***who*** *the user* ***is***
  - via *OIDC, SAML, Docker Auth, Kerberos*
  - Internal & Federated User Storage Kerberos, LDAP, Custom
- Authorization (AuthZ)
  - Determines ***what*** *the user* ***is allowed*** *to do*
  - Hierarchical Role-based Access Control (HRBAC)
  - Authorization Services
    - Flexible Access Control Management
    - More Variants like ABAC, UBAC, CBAC supported

Single Sign-on with Keycloak



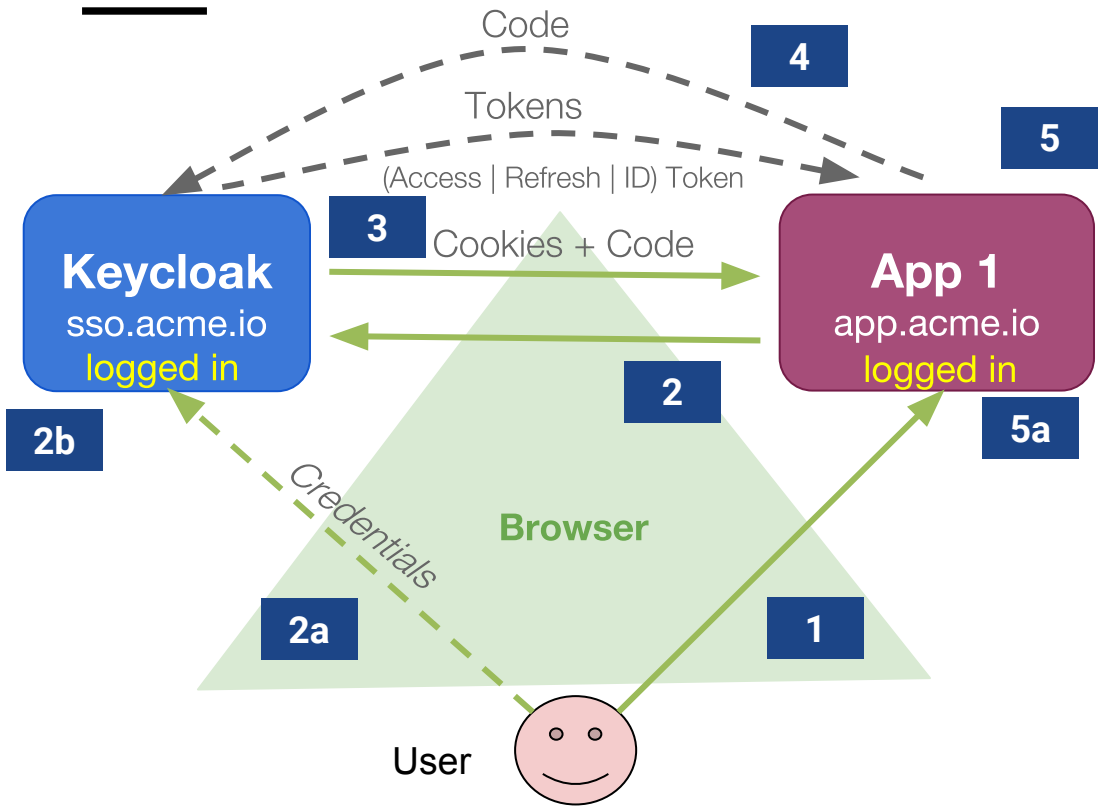
Single Sign-on & Single Sign-out

- **SSO** ⇒ Login **once** to access all applications
- **Standardized Protocols**
  - Open ID Connect 1.0 (OIDC)
  - Security Assertion Markup Language 2.0 (SAML)
- **Browser based “Web SSO”**
- works for Web, Mobile and Desktop Apps
- Support for **Single Sign-out**
  - Logouts can be propagated to clients
  - Clients can opt-in

Supported Single Sign-on Protocols

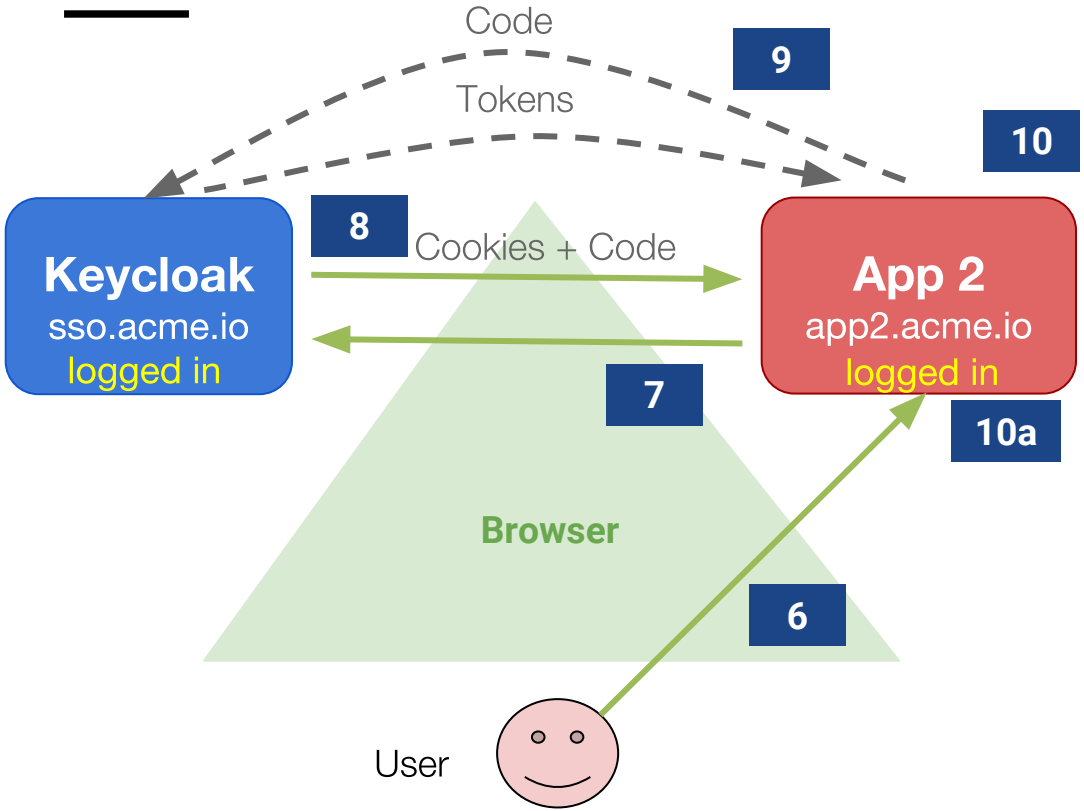
- OpenID Connect 1.0
  - Protocol based on OAuth 2.0
  - Uses OAuth 2.0 tokens + IDToken to encode Identity
  - Tokens are encoded as JSON Web Tokens (JWT)
  - Requires secure channel HTTPS/TLS
- SAML 2.0 Security Assertion Markup Language
  - Very mature standard & common in enterprise environments
  - XML based protocol
  - Uses XML signature and encryption → no secure channel required
- Docker Registry v2 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	Keycloak validates User Credentials
3	Keycloak creates SSO Session + Cookies and redirects User 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

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

Keycloak OAuth / OpenID Connect Tokens

- Tokens contain User information + Metadata claims
  - Signed self-contained **JSON Web Tokens**
  - Issued by Keycloak, **signed** with Realm **Private Key**
  - Limited lifespan; can be revoked
- Tokens can be verified by Clients
  - … by **verifying** the signature with Realm **Public Key**
- Essential Token Types
  - **Access-Token** short-lived (Minutes)→used for accessing Resources
  - **Refresh-Token** long-lived (Days)→used for requesting new Tokens
  - **Offline-Token** special *Refresh-Token* that “never” expires
  - **IDToken** contains information about User (OpenID Connect)

JSON Web Tokens

`<header-base64>.<payload-base64>.<signature-base64>`

Encoded PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJz
dWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4gR
G9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab3
0RMHrHDcEfxjoYZgeFONFh7HgQ
```

Note

Base64 means **Encoding**

Encoding != Encryption

Decoded EDIT THE PAYLOAD AND SECRET (ONLY HS256 SUPPORTED)

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD: DATA

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) secret base64 encoded
```


https://jwt.io

JSON Web Token Example

Encoded PASTE A TOKEN HERE


```
eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2
lkIiA6ICJMT0Rxc1Q3NFRwMFJRcjlHSmVpSXJRVnNV
blZZQzk3eF9fZ0ttc0k1TE93In0.eyJqdGkiOiJiMG
IyMGRjYy0wNmRkLTRiMzgtYTUyOS00ZDhiODg2Njdh
YjIiLCJleHAiOjE0OTA2NTM3NDIsIm5iZiI6MCwiaW
F0IjoxNDkwNjUzNDQyLCJpc3MiOiJodHRwOi8vc3Nv
LnRkbGFicy5sb2NhbDo4ODk5L3UvYXV0aC9yZWFsbX
MvamF2YWxhbmQiLCJhdWQiOiJpZG0tY2xpZW50Iiwi
c3ViIjoiMjI0Yjg3YWQtY2RkMi00NjY3LWFlODUtZW
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)

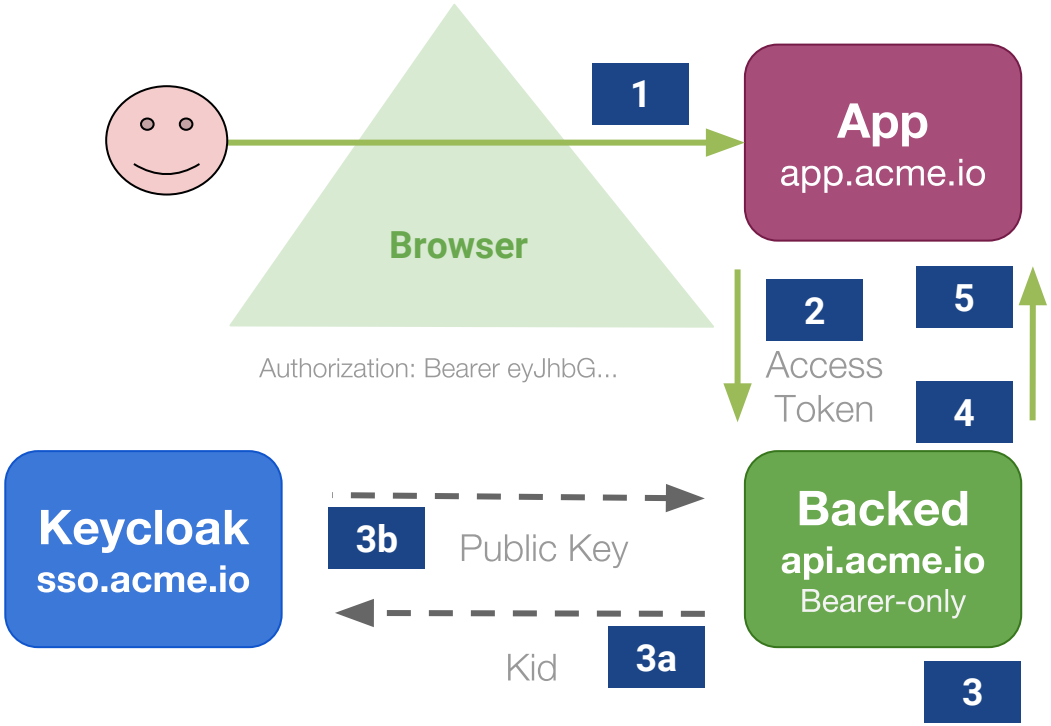
HEADER: ALGORITHM & TOKEN TYPE


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


PAYLOAD: DATA

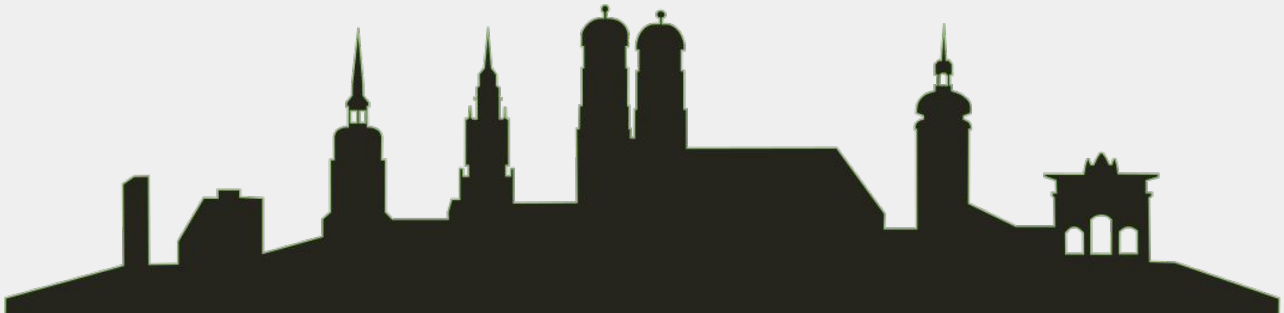

```
{
  "jti": "b0b20dcc-06dd-4b38-a529-4d8b88667ab2",
  "exp": 1490653742,
  "nbf": 0,
  "iat": 1490653442,
  "iss":
"http://sso.tdlabs.local:8899/u/auth/realms/javaland",
  "aud": "idm-client",
  "sub": "224b87ad-cdd2-4667-ae85-ea3d8fd3a6ac",
  "typ": "Bearer",
  "azp": "idm-client",
  "auth_time": 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"
      ]
```

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
3b. Keycloak returns Realm Public Key
4. Backend verifies Access-Token Signature with Realm Public Key
5. Backend Service grants access and returns user data

Keycloak Client Integrations

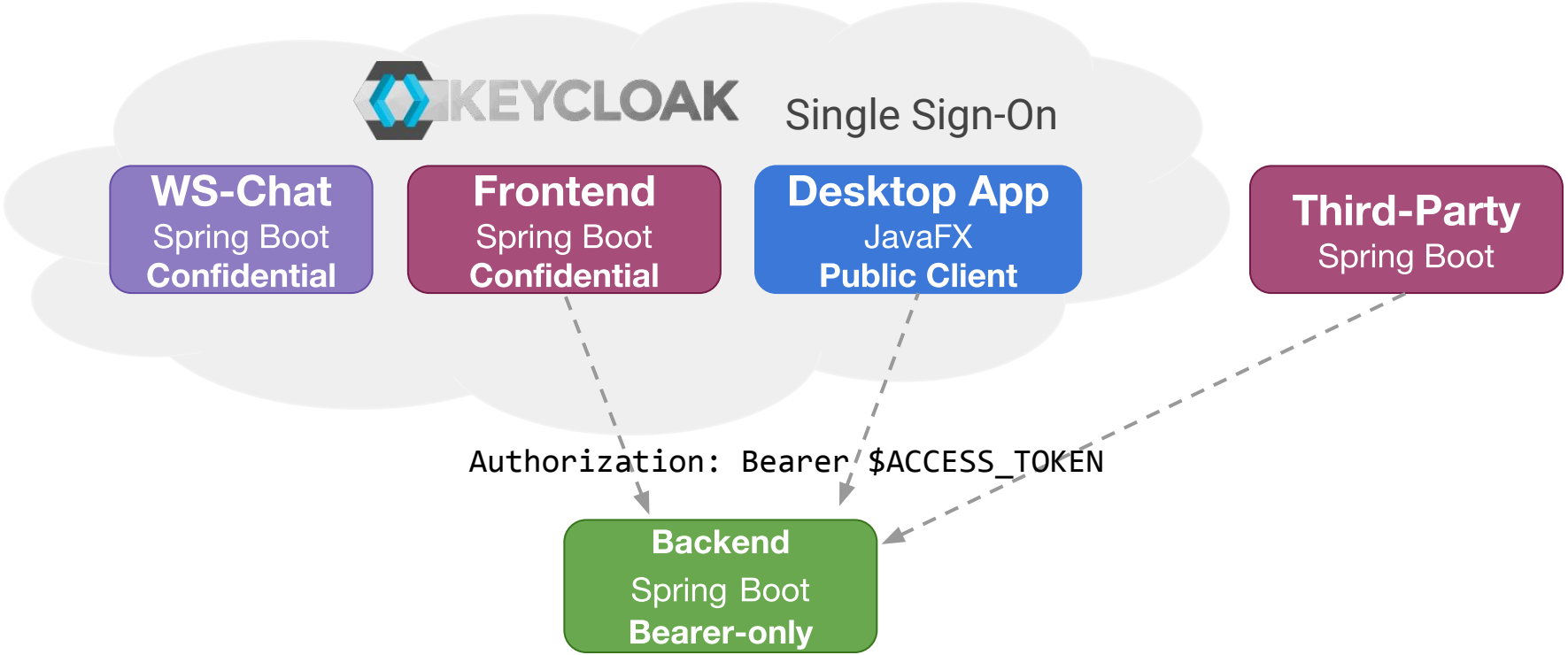


Keycloak Integration Options

- OpenID Connect Adapters
  - Spring Security, Spring Boot, ServletFilter, Tomcat, Jetty, Undertow, Wildfly, JBoss EAP,…
  - NodeJS, JavaScript, Angular, AngularJS, Aurelia, CLI & Desktop Apps…
- SAML Adapters
  - ServletFilter, Tomcat, Jetty, Wildfly
- Apache Modules
  - mod_auth_oidc for OpenID Connect - maintained by Ping Identity
  - mod_auth_mellon for SAML - maintained by Red Hat
- Reverse Proxies
  - Official Keycloak Proxy injects auth info into HTTP headers
  - keycloak-proxy on github… same written in Go
- Others see ODIC and SAML

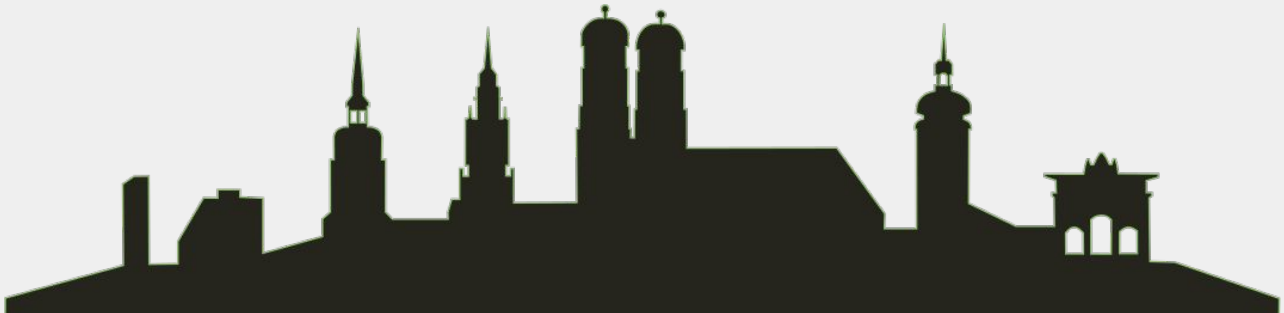
Keycloak Demo Securing Apps

Demo Environment

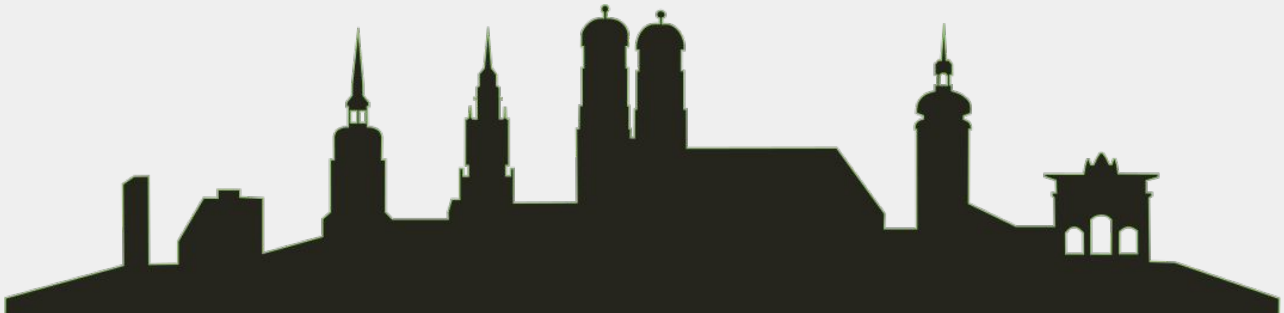


Demo Securing Apps

thomasdarimont/wjax2018-spring-keycloak



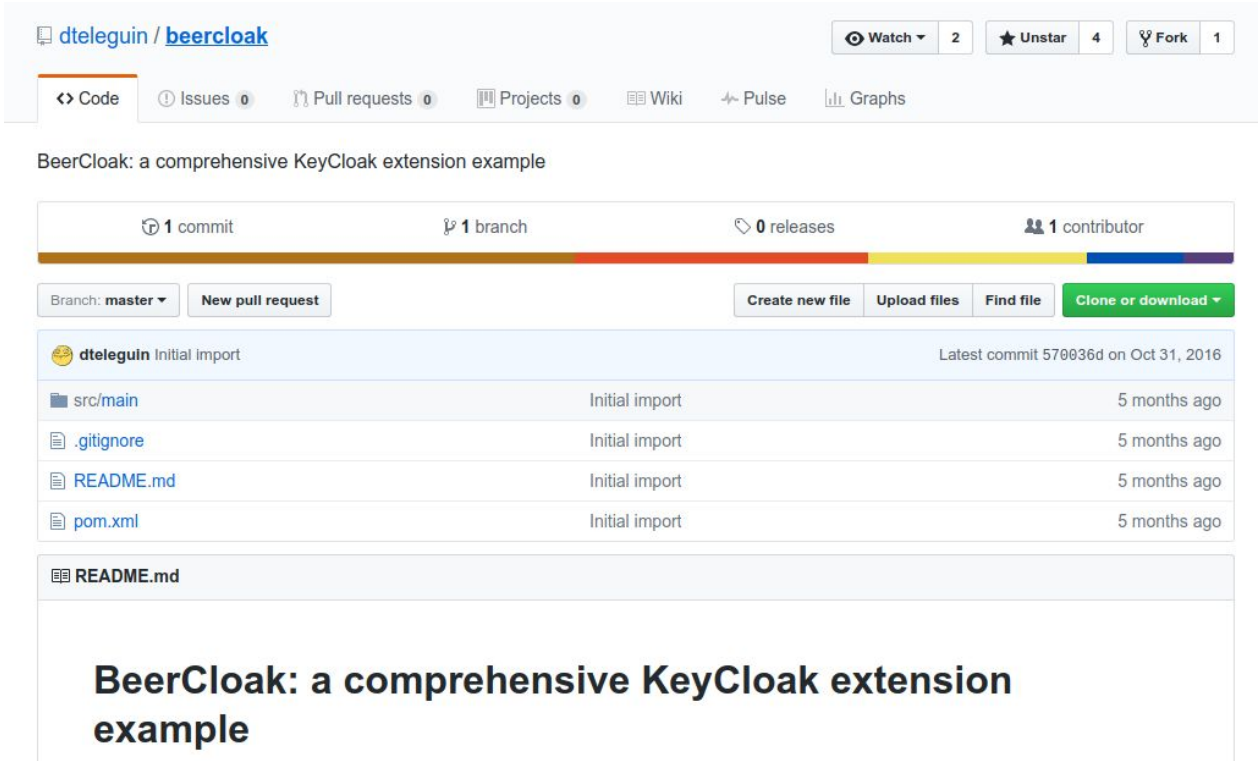
Keycloak Extensions



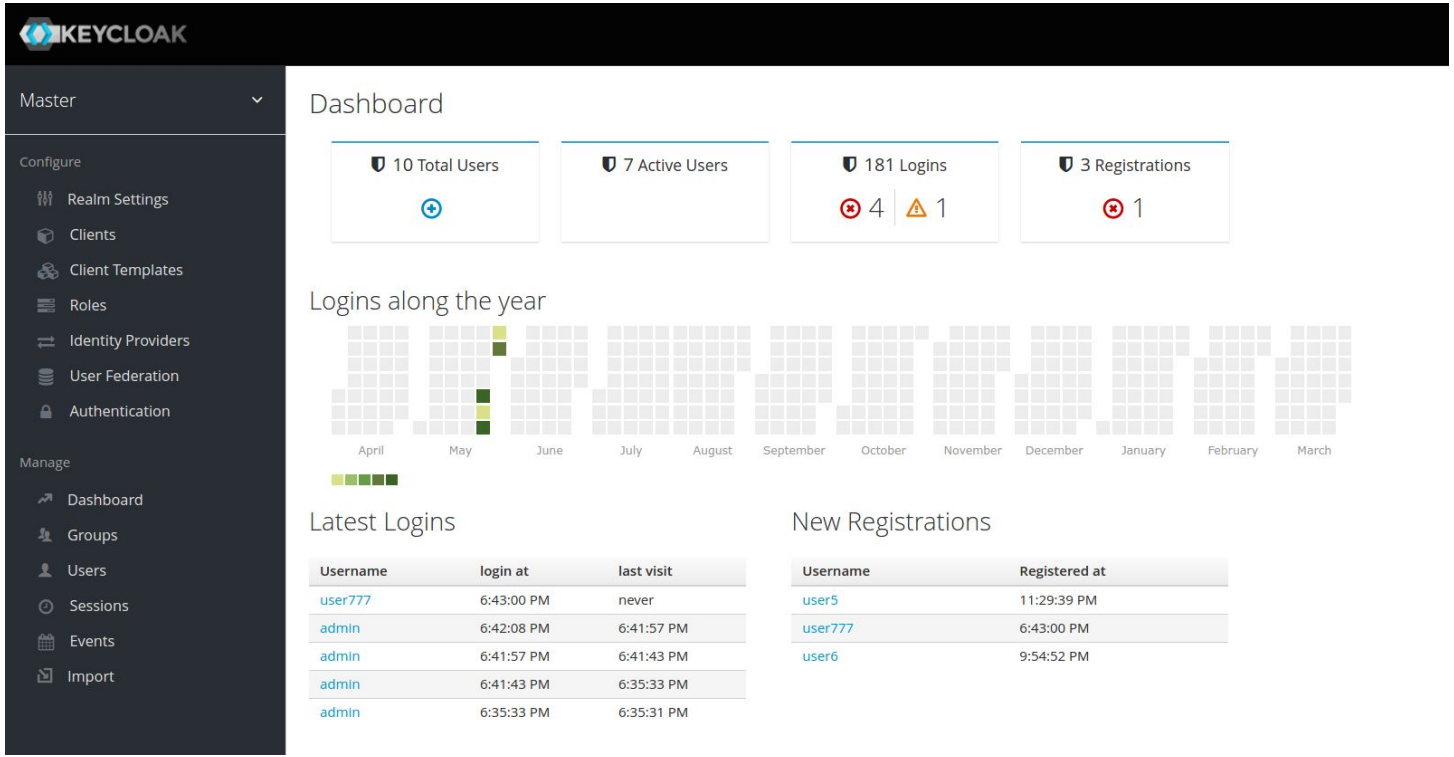
Keycloak Extension Points

- Extensions via *Service Provider Interfaces*
- Custom Authentication Mechanisms
- Custom “Required Actions”
- Custom User Storage (JDBC, REST, etc.)
- Event Listener (Provisioning, JMS)
- Credential Hashing Mechanisms
- Custom REST Endpoints
- Custom Persistent Entities
- Custom Themes
- … many more

Keycloak Extensions Example dteleguin/beercloak

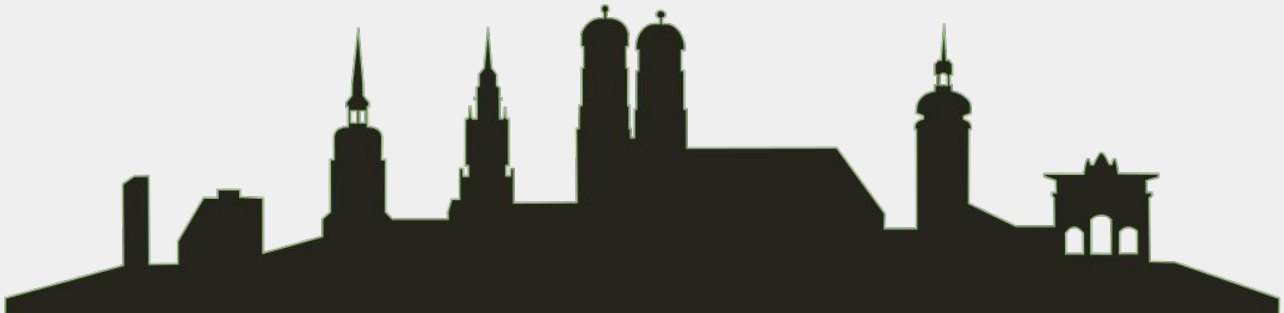


Custom Dashboard Extension

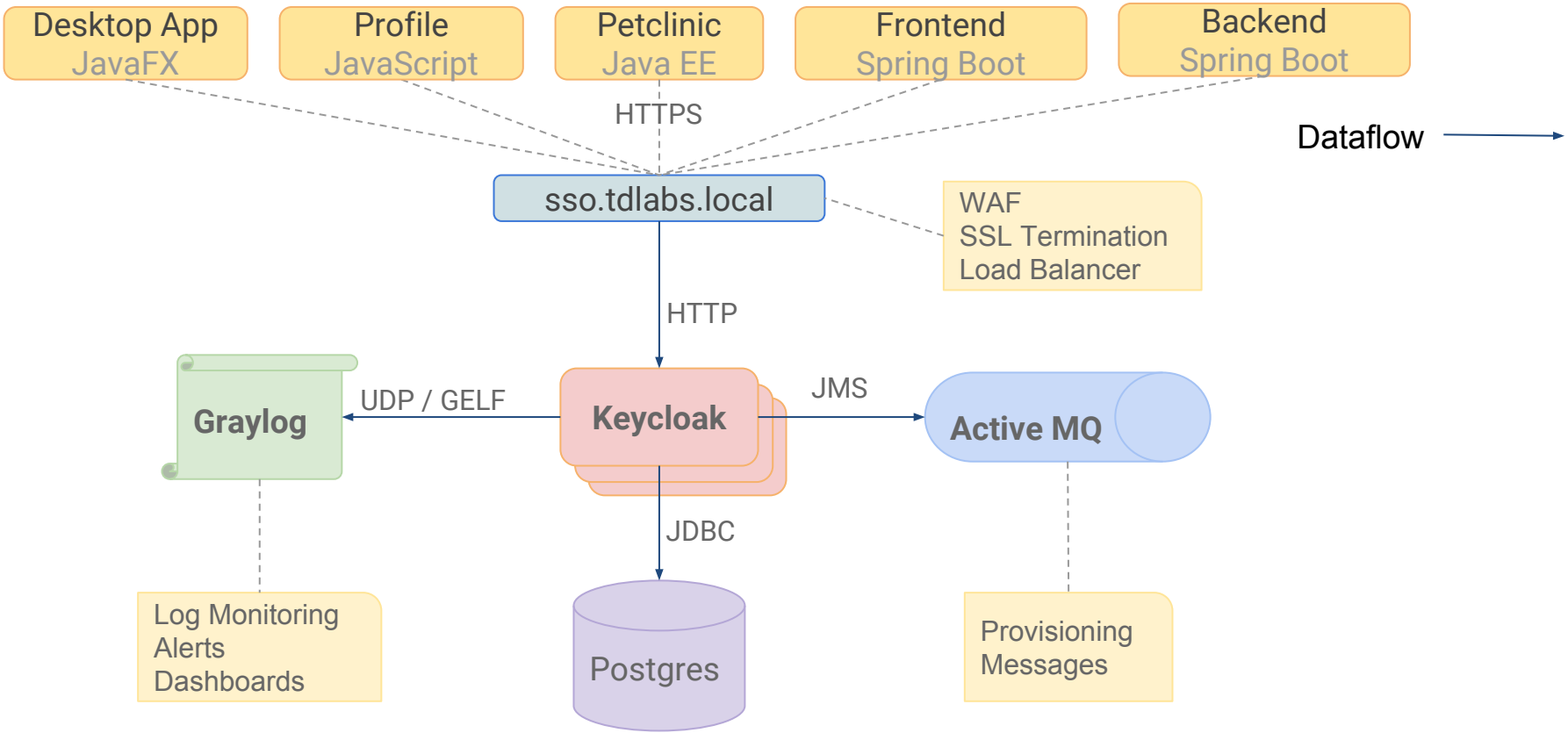


Please vote :) https://issues.jboss.org/browse/KEYCLOAK-1840

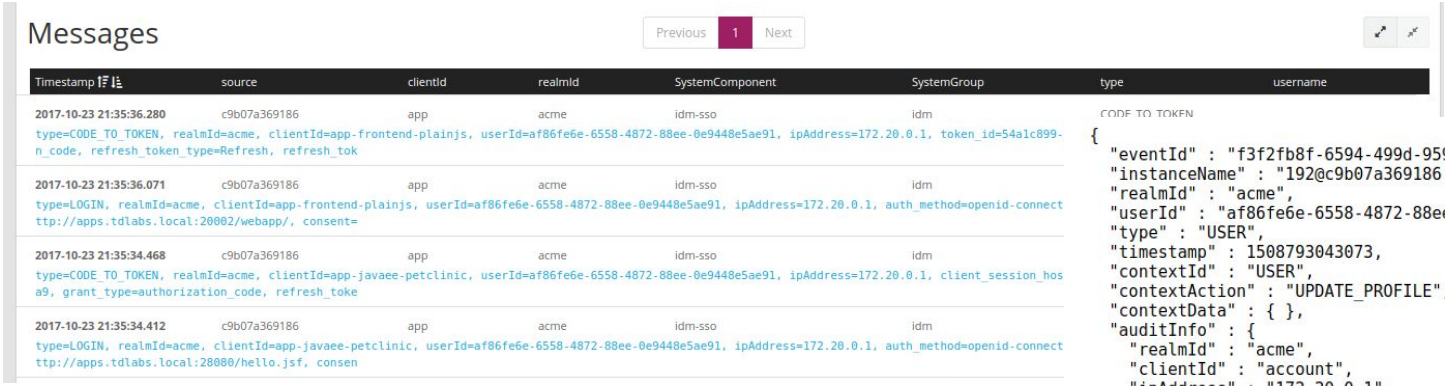
Keycloak in the field

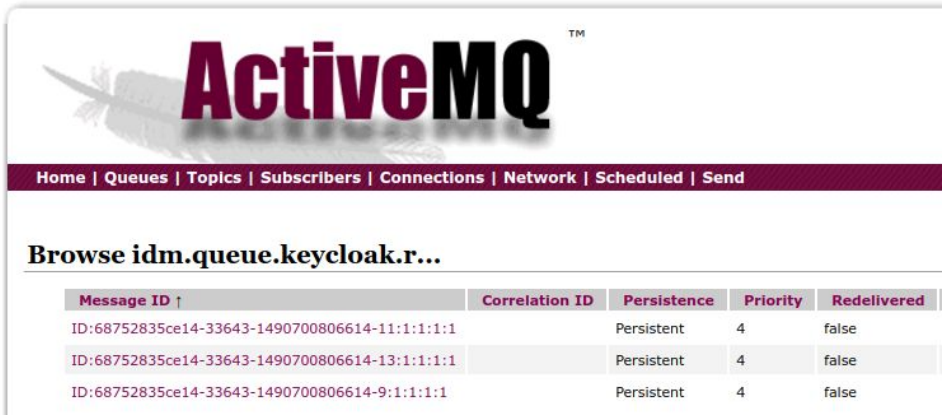


Demo Environment



Demo: Keycloak, Graylog, ActiveMQ





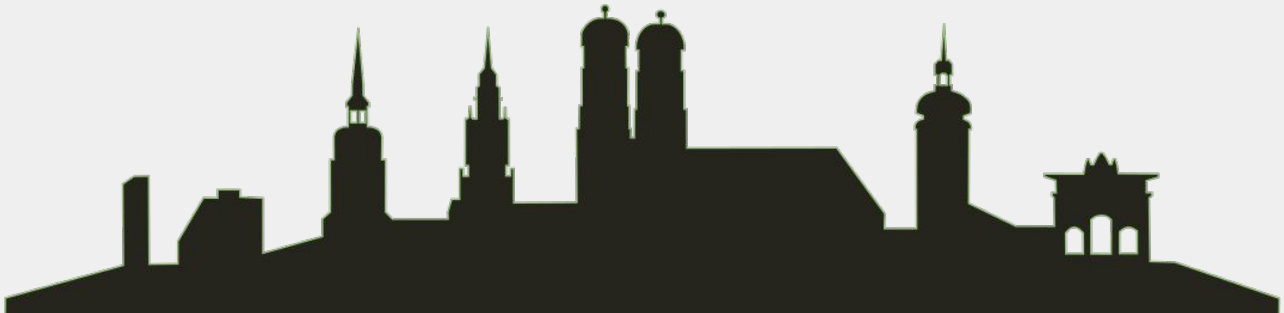

```
{
  "eventId" : "f3f2fb8f-6594-499d-9590-287d9c5645bf",
  "instanceName" : "192@c9b07a369186:172.20.0.7",
  "realmId" : "acme",
  "userId" : "af86fe6e-6558-4872-88ee-0e9448e5ae91",
  "type" : "USER",
  "timestamp" : 1508793043073,
  "contextId" : "USER",
  "contextAction" : "UPDATE_PROFILE",
  "contextData" : { },
  "auditInfo" : {
    "realmId" : "acme",
    "clientId" : "account",
    "ipAddress" : "172.20.0.1",
    "userId" : "af86fe6e-6558-4872-88ee-0e9448e5ae91",
    "username" : "tester"
  },
  "userInfo" : {
    "userId" : "af86fe6e-6558-4872-88ee-0e9448e5ae91",
    "realmId" : "acme",
    "emailVerified" : false,
    "enabled" : true,
    "username" : "tester",
    "email" : "tom+tester@localhost",
    "firstname" : "Theo",
    "lastname" : "Tester",
    "creationDateTime" : 1488399721096,
    "attributes" : {
      "dev" : [ "true" ],
      "origin" : [ "legacy-system1" ]
    }
  }
}
```

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 extension points & customization options
- A pivotal part of an Identity Management infrastructure

THANKS!
Q & A

Thomas Darimont
@thomasdarimont



Links

- Keycloak Website
- Keycloak Docs
- Keycloak Blog
- Keycloak User Mailing List
- Keycloak Developer Mailing List
- OpenID Connect
- SAML
- JSON Web Tokens
- Awesome Keycloak
- Keycloak Dockerized Examples
- Keycloak Quickstarts Example Projects

Accessing the API Backend with CURL

1 Request new Tokens via Password Credentials Grant (Direct Access Grants in Keycloak)

```
KC_RESPONSE=$(curl -X POST \
  http://sso.tdlabs.local:8899/u/auth/realms/acme/protocol/openid-connect/token \
  -d 'grant_type=password' \
  -d 'username=tester&password=test' \
  -d 'client_id=app-frontend-springboot&client_secret=4822a740-20b9-4ff7-bbed-e664f4a70eb6' \
)
```

2 Extract AccessToken

```
KC_ACCESS_TOKEN=$(echo $KC_RESPONSE | jq -r .access_token)
# eyJhbGciOiJSUzI1NiIsInR5cCIgOiAiSldUIiwia2lkIiA6ICJGY3RMVHJqeWRxYkpISGZ0d29U ...
```

3 Use AccessToken in Authorization Header

```
curl \
  -H "Authorization: Bearer $KC_ACCESS_TOKEN" \
  http://apps.tdlabs.local:20000/todos/search/my-todos
```

Desktop Applications

- Two ways to integrate Desktop Applications
  - Direct Access Grants - ***no*** SSO
  - KeycloakInstalled Adapter - SSO
- Direct Access Grants
  - Client sends HTTP `POST` request to Keycloaks `/token` Endpoint
  - `client_id, username, password, grant_type=password`
  - Keycloak returns Tokens (Access-, ID-, Refresh-Token)
  - Client needs to parse & validate tokens
  - Client sees password → *Password Anti-Pattern*
- KeycloakInstalled Adapter
  - Enables OAuth2 *authorization code flow* for Desktop / CLI apps
  - Code to Token exchange via short lived `ServerSocket@localhost`
  - Uses Keycloak Login via Browser
  - Can reuse existing SSO session

Using the KeycloakInstalled Adapter

1. Add Maven Dependency

```
<dependency>
    <groupId>org.keycloak</groupId>
    <artifactId>keycloak-installed-adapter</artifactId>
    <version>${keycloak.version}</version>
</dependency>
```

2. Export keycloak.json for Client

```
{ "realm": "acme",
 "auth-server-url": "http://sso.tdlabs.local:8899/u/auth",
 "ssl-required": "external",
 "resource": "app-frontend-javafx",
 "public-client": true, "use-resource-role-mappings": true }
```

3. Create KeycloakInstalled

```
KeycloakInstalled keycloak = new KeycloakInstalled();
```

4. Trigger Browser login

```
keycloak.loginDesktop();
```

5. Read current username

```
keycloak.getIdToken().getPreferredUsername()
```

6. Read & use AccessToken string

```
String token = keycloak.getTokenString(10, TimeUnit.SECONDS);
httpClient.header("Authorization", "Bearer " + token);
```

7. Trigger Browser Logout

```
keycloak.logout()
```

Tips for working with Keycloak

- Learn to configure Wildfly → Booktip: Wildfly Cookbook
- Keep your Tokens small → HTTP Header limits!
  - Only put in the tokens what you really need → Full Scope Allowed = off
- Keycloak provides a Realm-scoped Admin Console
  - http://kc-host:8080/auth/admin/my-realm/console
  - Admin users need permissions for realm-management in `my-realm`
- Secure your Keycloak Installation!
  - Keycloak exposes some undocumented Endpoints by default on server AND client!
  - Inspect other Keycloak instances to learn what to hide
    - Google Search for Keycloak Endpoints
    - Shodan search for Keycloak

Credits

Silhouette of Munich.svg - Stefan Meister