

Opensource Identity & Access Management with Keycloak

Thomas Darimont | @thomasdarimont

Thomas Darimont

- Fellow @ codecentric
- Pivotal Spring Team Alumni
- Open Source Enthusiast
- Java User Group Saarland
- Keycloak Contributor for over 4 years





@thomasdarimont@jugsaar

The Journey



Keycloak



Single Sign-on



Securing Applications



Keycloak in the Field







Open Source Identity and Access Management

For Modern Applications and Services

Learn Keycloak Basics

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 https://www.keycloak.org

03 Sep

What's Coming To Keycloak

24 Aug

Keycloak 7.0.0 released

12 Aug

Keycloak and JDBC Ping

Project



- Java based Authentication & Authorization Server
- Started in 2013, broad adoption since 2015
- Apache License, Red Hat Developers
- Keycloak Community Free Version (current 7.0.0)
- Red Hat SSO Commercial Offering
- Vital Community with 320+ Contributors 2.100+ Forks
- Very robust, good documentation, many examples

Authentication & Authorization

- Authentication (AuthN)
 - Determines who the user is
 - Internal & Federated User Storage Kerberos, LDAP, Custom
 - Customizable
- Authorization (AuthZ)
 - Determines what the user is allowed to do
 - Hierarchical Role-based Access Control (HRBAC)
 - Authorization Services
 - Flexible <u>Access Control Management</u>
 - More Variants like ABAC, UBAC, CBAC supported

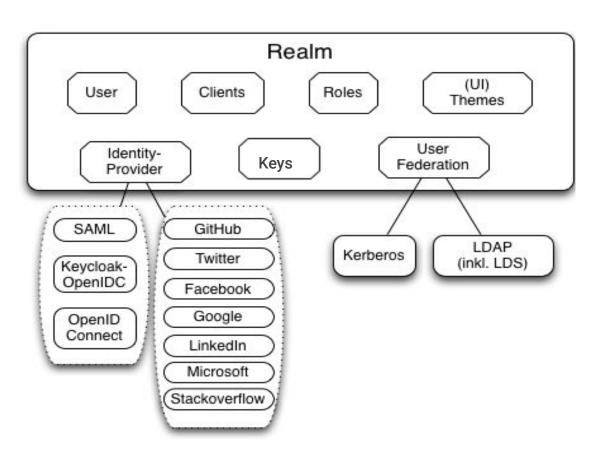
Features



- Single Sign-on and Single Sign-out
- Standard Protocols OAuth 2.0, OIDC 1.0, SAML 2.0
- Flexible Authentication and Authorization
- Multi-Factor Authentication One-time Passwords
- Social Login Google, Facebook, Twitter,..., Azure, ADFS, Auth0
- Provides centralized User Management
- Supports Directory Services
- Customizable and Extensible
- Easy Setup and Integration

Main Concepts







Admin Console

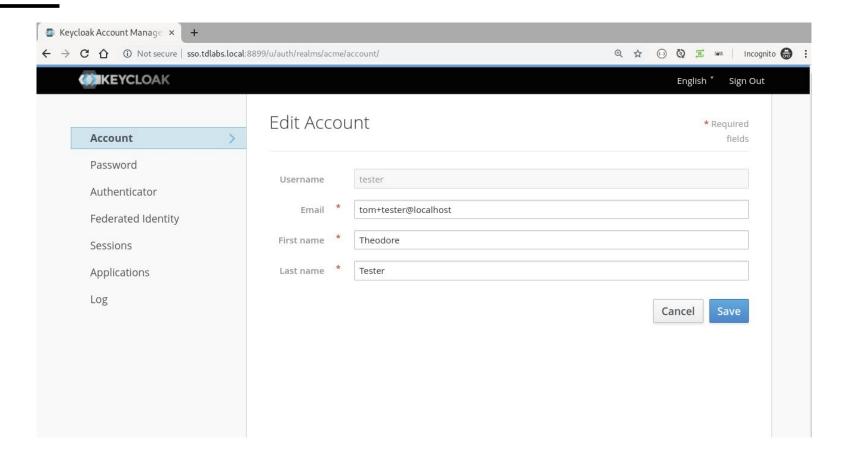


Admin Console



WKEYCLOAK			≜ Admin ∨
Acme ~	Acme 👚		
	General Login	Keys Email Themes Cache Tokens Client Registration Security Defenses	
👭 Realm Settings	* Name	acme	
Clients Client Scopes	Display name	Acme Inc.	
Roles	HTML Display name	Acme Inc.	
□ Identity Providers	Enabled 🚱	ON N	
User Federation Authentication	User-Managed Access @	OFF	
	Endpoints @	OpenID Endpoint Configuration SAML 2.0 Identity Provider Metadata	
4 Groups			
		Save Cancel	
Sessions			
m Events			
☑ Import			

Self-Service Account Management



Technology Stack 7.0.0 RELEASE

Admin Console

- Angular JS
- PatternFly
- Bootstrap

Keycloak Server

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

























HIBERNATE



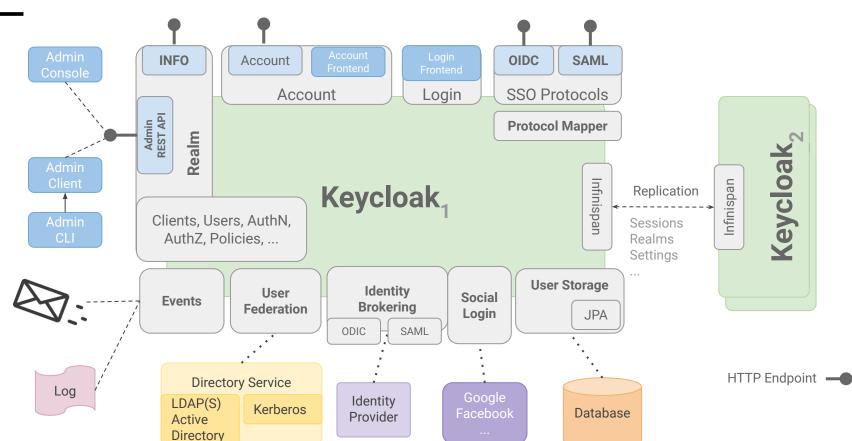






Server Architecture





Single Sign-on with Keycloak

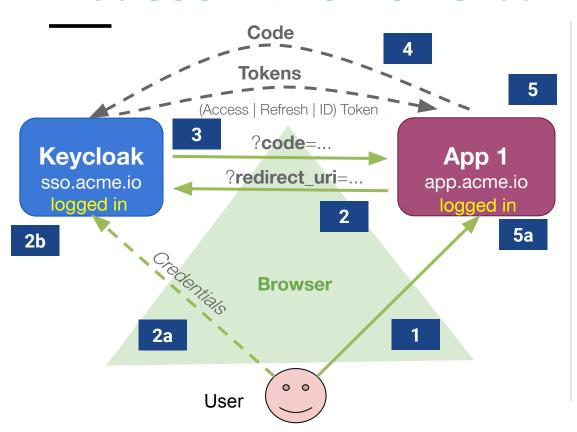
How it works



Single Sign-on

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

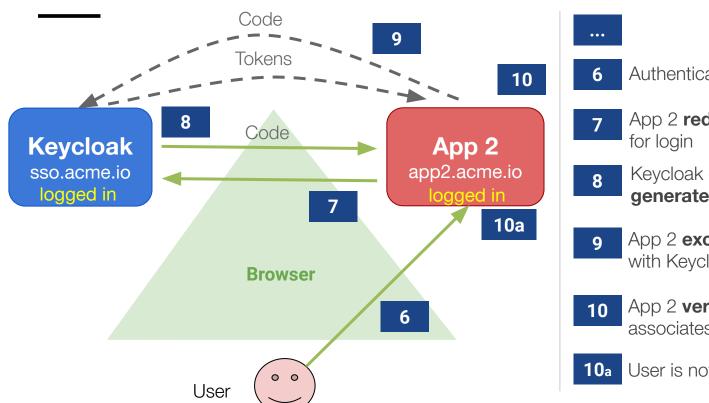
Web SSO with OIDC*: Unauthenticated User



- 1 Unauthenticated User accesses App
- 2 App **redirects** to Keycloak for Login
- 2a User **submits** Credentials to Keycloak
- Credentials OK? → Keycloak **creates**SSO Session and emits Cookies
- **Generates** *Code* and **redirects** User back to App
- App **exchanges** Code to Tokens with Keycloak via separate Channel
- App **verifies** received *Tokens* and associates it with a session
- 5a User is now *logged-in* to App

^{*)} OpenID Connect with OAuth 2.0 Authorization Code Grant Flow

Web SSO with OIDC: Authenticated User



- 6 Authenticated user **accesses** App 2
- 7 App 2 **redirects** user to Keycloak for login
- Keycloak **detects** SSO Session, **generates** code, **redirects** to App 2
- App 2 **exchanges** code for tokens with Keycloak via separate channel
- 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 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
- IDToken → contains User information (OIDC)
- Offline-Token long-lived (Days++) "Refresh-Token" that "never" expires

JSON Web Tokens



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

Encoded PASTE A TOKEN HERE



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

Keycloak JSON Web Token Example

Encoded PASTE A TOKEN HERE

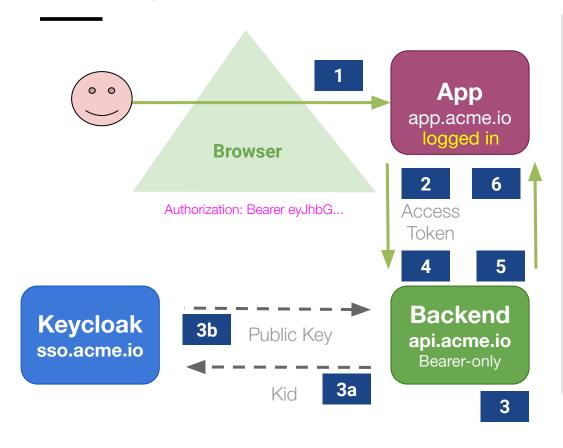
evJhbGciOiJSUzI1NiIsInR5cCIqOiAiSldUIiwia2 lkIiA6ICJMT0Rxc1Q3NFRwMFJRcj1HSmVpSXJRVnNV blZZQzk3eF9fZ0ttc0k1TE93In0.eyJqdGki0iJiMG IyMGRjYy0wNmRkLTRiMzgtYTUy0S00ZDhi0Dg2Njdh YjIiLCJleHAiOjE00TA2NTM3NDIsIm5iZiI6MCwiaW F0IjoxNDkwNjUzNDQyLCJpc3Mi0iJodHRw0i8vc3Nv LnRkbGFicv5sb2NhbDo40Dk5L3UvYXV0aC9vZWFsbX MvamF2YWxhbmQiLCJhdWQi0iJpZG0tY2xpZW50Iiwi c3ViIjoiMjI0Yjg3YWQtY2RkMi00NjY3LWF10DUtZW EzZDhmZDNhNmFjIiwidHlwIjoiQmVhcmVyIiwiYXpw IjoiaWRtLWNsaWVudCIsImF1dGhfdGltZSI6MCwic2 Vzc2lvb19zdGF0ZSI6IjZmZDQ3MjNkLTQwYjItNGM4 Ny1iMzliLTk4YTA3N2ZmM2FkNCIsImFjciI6IjEiLC JjbGllbnRfc2Vzc2lvbiI6IjM4Nzk5ZjgyLTBkNmMt NDAyYy1hYmEwLTY3ZDI3NGVjZWIzMCIsImFsbG93ZW Qtb3JpZ2lucyI6W10sInJlYWxtX2FjY2VzcyI6eyJy b2xlcyI6WyJ1bWFfYXV0aG9yaXphdGlvbiIsInVzZX IiXX0sInJlc291cmNlX2FjY2VzcyI6eyJhcHAtZ3J1 ZXRpbmctc2VydmljZSI6eyJyb2xlcyI6WyJ1c2VyI1 19LCJkZW1vLXNlcnZpY2UiOnsicm9sZXMiOlsidXNl ciJdfSwiYXBwLWphdmFlZS1wZXRjbGluaWMiOnsicm 9sZXMiOlsidXNlciJdfSwiYWNjb3VudCI6eyJyb2xl cyI6WyJtYW5hZ2UtYWNjb3VudCIsInZpZXctcHJvZm lsZSJdfSwiYXBwLWRlc2t0b3AiOnsicm9sZXMiOlsi dXNlciJdfX0sIm5hbWUiOiJUaGVvIFRlc3RlciIsIn ByZWZlcnJlZF91c2VybmFtZSI6InRlc3RlciIsImdp

Decoded EDIT THE PAYLOAD AND SECRET (ONLY H5256 SUPPORTED)

```
HEADER: ALGORITHM & TOKEN TYPE
   "alg": "RS256".
   "tvp": "JWT",
   "kid": "LODgsT74Tp0RQr9GJeiIrQVsUnVYC97x__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",
   "client_session": "38799f82-0d6c-402c-aba0-67d274eceb30",
   "allowed-origins": [],
   "realm_access": {
     "roles": [
       "uma_authorization",
       "user"
    "resource_access": {
      "app-greeting-service": {
       "roles": [
          "user"
```

https://jwt.io

Calling Backend Services with Access-Token



- 1 Authenticated User accesses App
- App **uses** Access-Token in HTTP Header to access backend
- 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
- Backend **verifies** signature of Access-Token with Realm Public Key
- Backend Service **grants** access and **returns** user data
- 6 App can now display user data

Keycloak Client Integrations



Keycloak Integration Options

OpenID Connect Keycloak Adapters

- Spring Security, Spring Boot, ServletFilter, Tomcat, Jetty, Undertow, Wildfly, JBoss EAP,...
- NodeJS, JavaScript, Angular, AngularJS, Aurelia, CLI & Desktop Apps...

SAML Keycloak Adapters

ServletFilter, Tomcat, Jetty, Wildfly ...

Reverse Proxies

- Keycloak Gatekeeper, dedicated Proxy, written in Go, injects auth info into HTTP headers
- Apache mod_auth_oidc for OpenID Connect
- Apache mod_auth_mellon for SAML
- Many more generic integrations see <u>OIDC</u> and <u>SAML</u>



Keycloak Demo

Securing Apps



Demo Environment



WS-Chat

Spring Boot onc Confidential

Frontend

Spring Boot OIDC Confidential

Plain JS App

Javascript **OIDC Public Client**

Frontend

Spring Boot **SAML**

Authorization: Bearer \$ACCESS TOKEN

Backend

Spring Boot **OAUTH Bearer-only**



Keycloak Demo

Securing Apps

thomasdarimont/keycloak-docker-demo

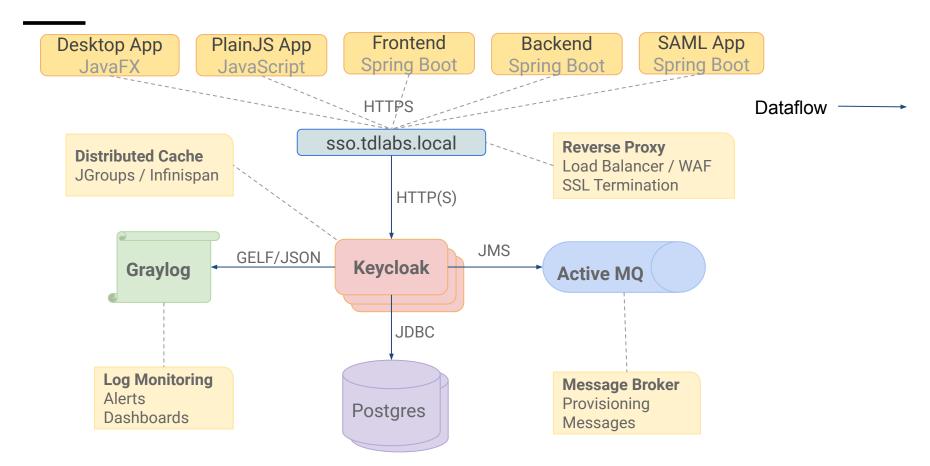


Keycloak in the Field

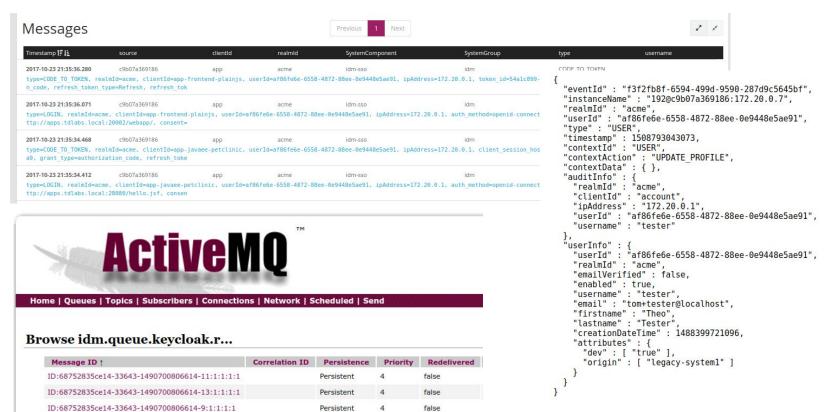
How can a Keycloak environment look like?



Demo Environment



Keycloak with Graylog + ActiveMQ





- 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 a modern Identity Management

Thanks!



@thomasdarimont



Links

- Keycloak Website
- Keycloak Docs
- Keycloak Blog
- Keycloak User Mailing List
- Keycloak Developer Mailing List
- OpenID Connect
- Keycloak Community Extensions

- SAML
- JSON Web Tokens
- Awesome Keycloak
- Keycloak Dockerized Examples
- Keycloak Quickstart Projects
- Keycloak Extension Playground

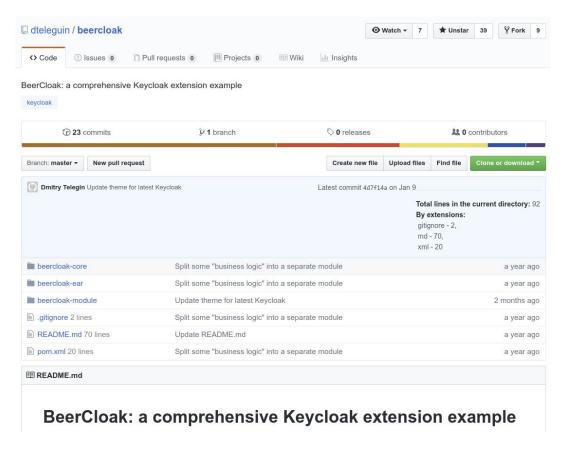
Tips for working with Keycloak

- 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

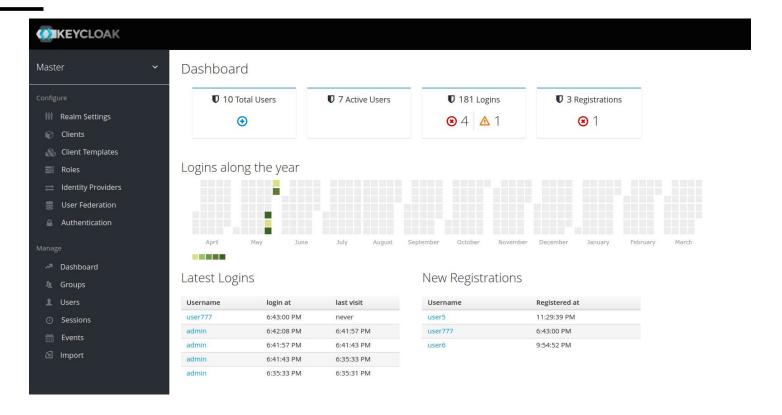
Keycloak Extension Points

- 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 Themes
- ... many more

Keycloak Extension Example



Custom Dashboard Extension



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

Supported Authentication 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 (<u>JWT</u>)
 - 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
- Docker Registry v2 Authentication

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

```
Add Maven Dependency
                                <dependency>
                                   <groupId>org.keycloak
                                   <artifactId>keycloak-installed-adapter</artifactId>
                                   <version>${keycloak.version}</version>
                                </dependency>
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 }
Create KeycloakInstalled
                                 KeycloakInstalled keycloak = new KeycloakInstalled();
Trigger Browser login
                                 keycloak.loginDesktop();
Read current username
                                 keycloak.getIdToken().getPreferredUsername()
Read & use AccessToken string
                                 String token = keycloak.getTokenString(10, TimeUnit.SECONDS);
                                 httpClient.header("Authorization", "Bearer " + token);
Trigger Browser Logout
                                                                                            41
                                 keycloak.logout()
```