KEYCLOAK

Open Source Identity & Access Management with Keycloak

Thomas Darimont | @thomasdarimont

Thomas Darimont

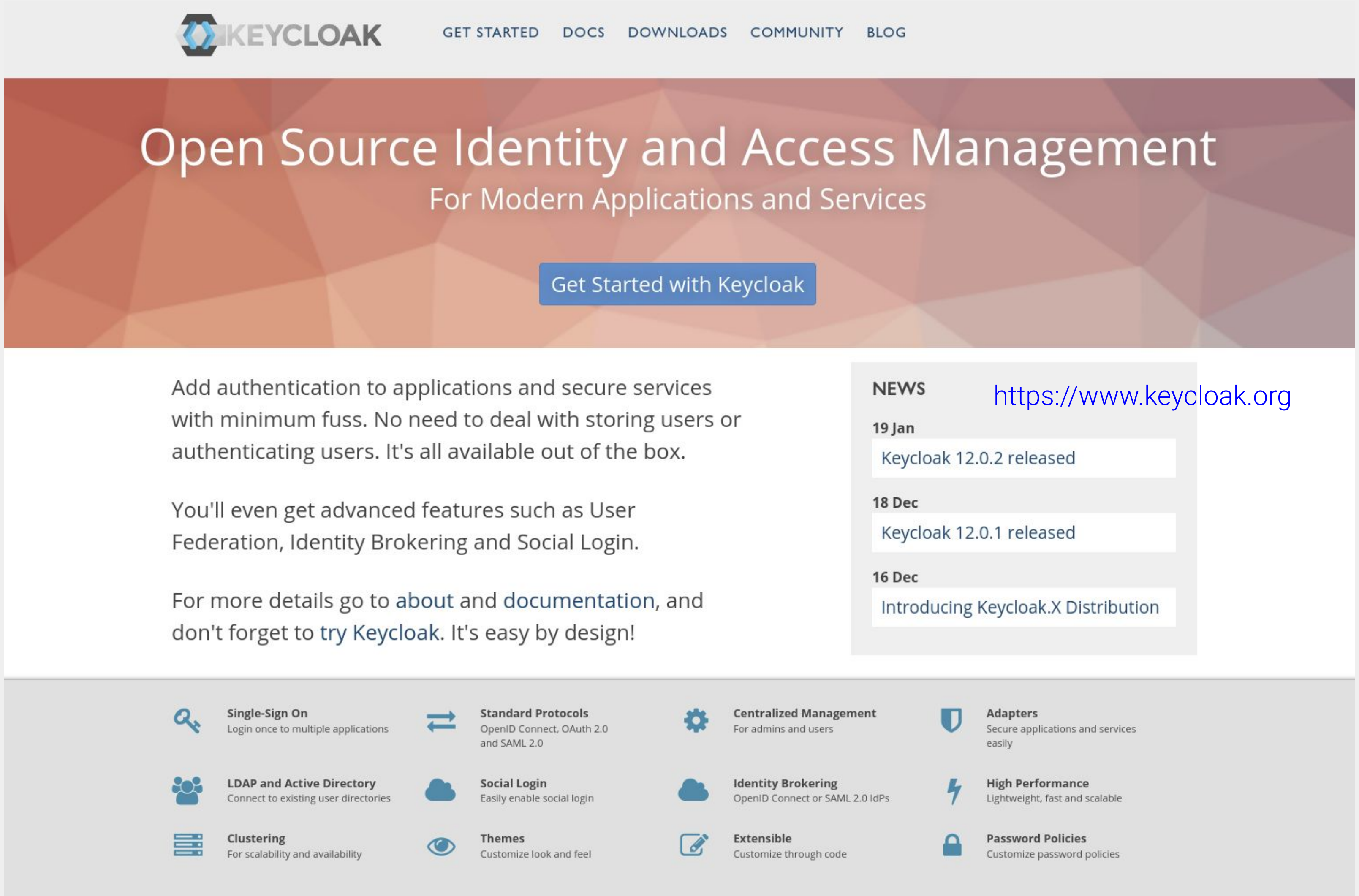
- Fellow codecentric
- Pivotal Spring Team Alumni
- Open Source Enthusiast
- Java User Group Saarland
- Keycloak Contributor for over 5 years

@thomasdarimont
@jugsaar

The Journey

- Keycloak
- Single Sign-on
- Securing Applications
- Keycloak in the Field

Keycloak Overview


GET STARTED DOCS DOWNLOADS COMMUNITY BLOG

Open Source Identity and Access Management

For Modern Applications and Services

Get Started with Keycloak

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

19 Jan
Keycloak 12.0.2 released

18 Dec
Keycloak 12.0.1 released

16 Dec
Introducing Keycloak.X Distribution

https://www.keycloak.org

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

- Java based **Authentication** & **Authorization** Server
- Started in 2013, broad adoption since 2015
- Apache License, **Red Hat** Developers
- **Keycloak Community** Free Version (current 13.0.0)
- **Red Hat SSO** Commercial Offering
- Vital **Community** with **470+** Contributors **3.3k+** Forks
- Very **robust**, good **documentation**, many **examples**

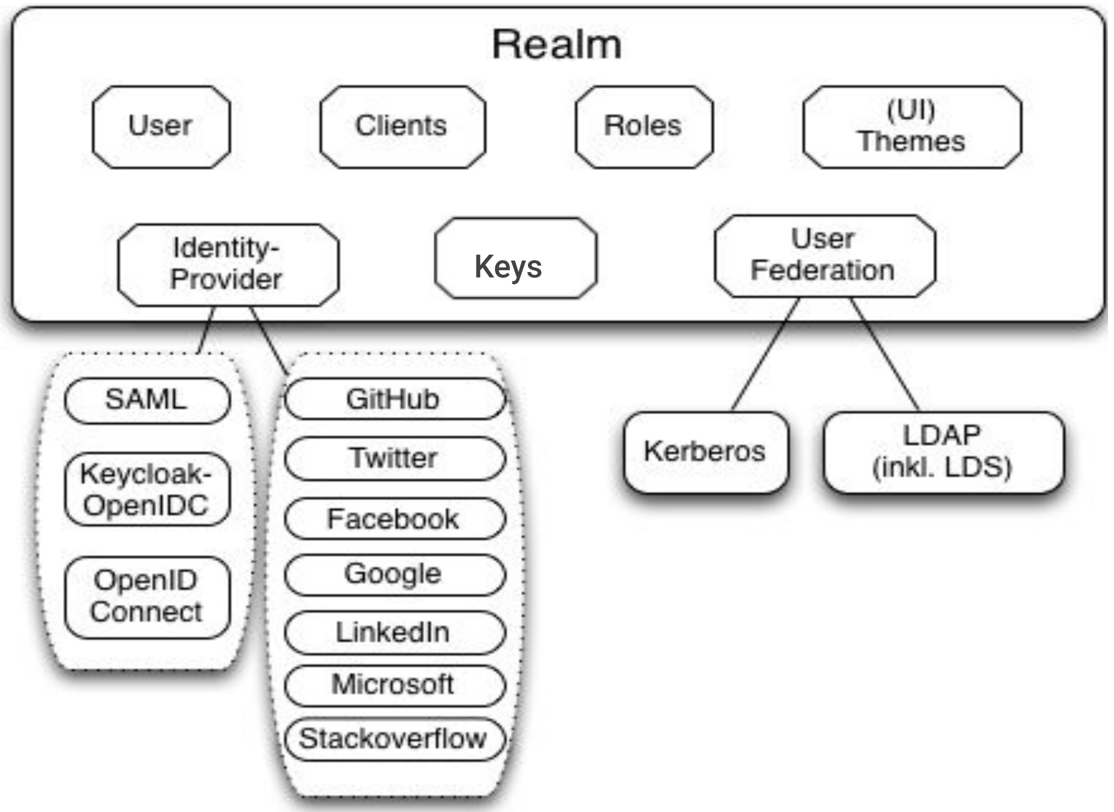
Authentication & Authorization

- 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 Access Control Management
    - 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

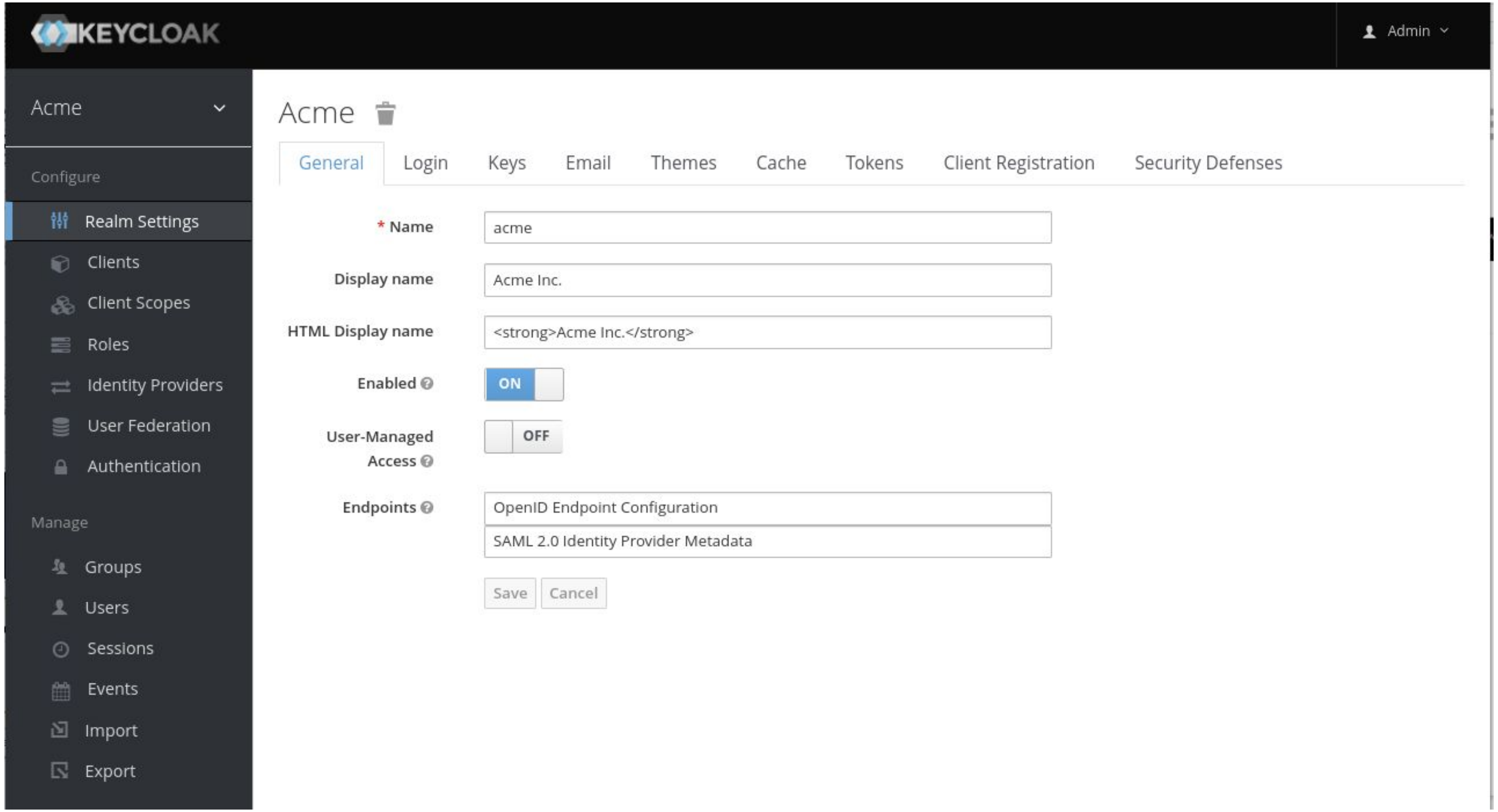
Main Concepts



Keycloak Quick Tour

Admin Console

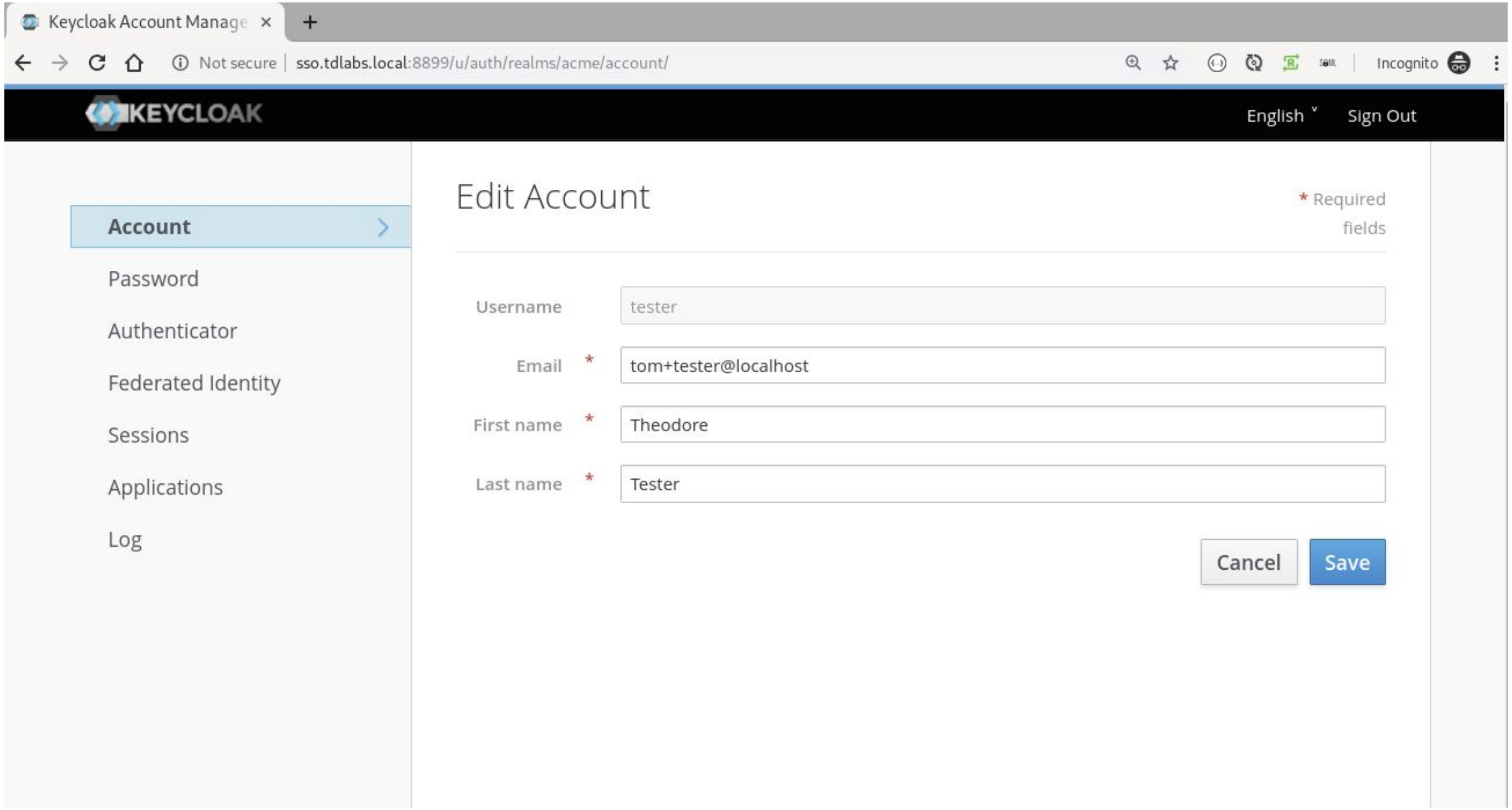
Admin Console



Acme

* Name	acme
Display name	Acme Inc.
HTML Display name	<strong>Acme Inc.</strong>
Enabled	ON
User-Managed Access	OFF
Endpoints	OpenID Endpoint Configuration
	SAML 2.0 Identity Provider Metadata

Self-Service Account Management



Keycloak Account Manage

Not secure | sso.tdlabs.local:8899/u/auth/realms/acme/account/

Incognito

KEYCLOAK

English ˅ Sign Out

- Account
- Password
- Authenticator
- Federated Identity
- Sessions
- Applications
- Log

Edit Account

* Required fields

Username	tester
Email *	tom+tester@localhost
First name *	Theodore
Last name *	Tester

Cancel Save

Keycloak Technology Stack

Admin Console

- Angular JS
- PatternFly
- Bootstrap

Keycloak Server

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

Keycloak.X (Preview) Technology Stack

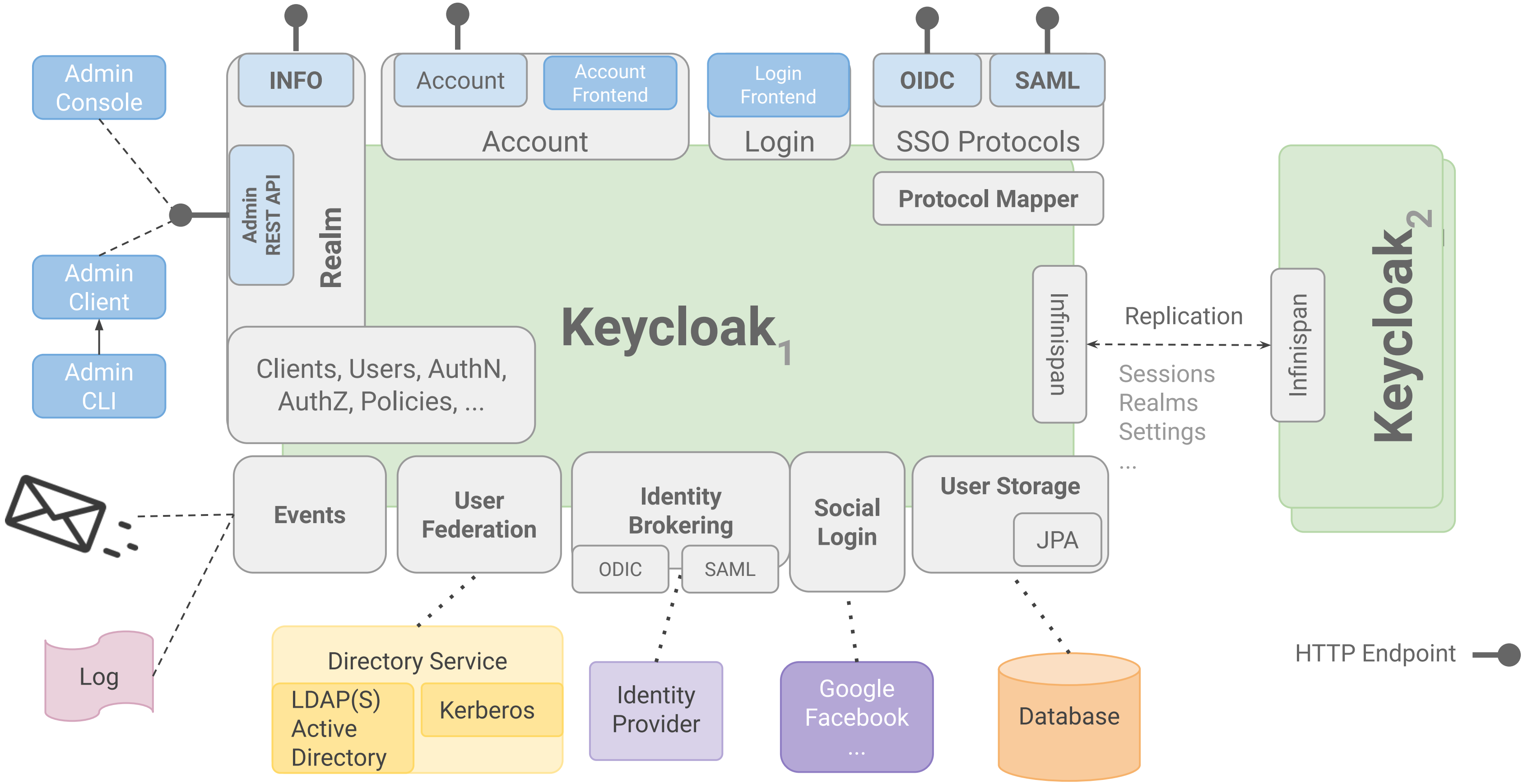
Admin Console

- Angular JS
- PatternFly
- Bootstrap

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

Server Architecture



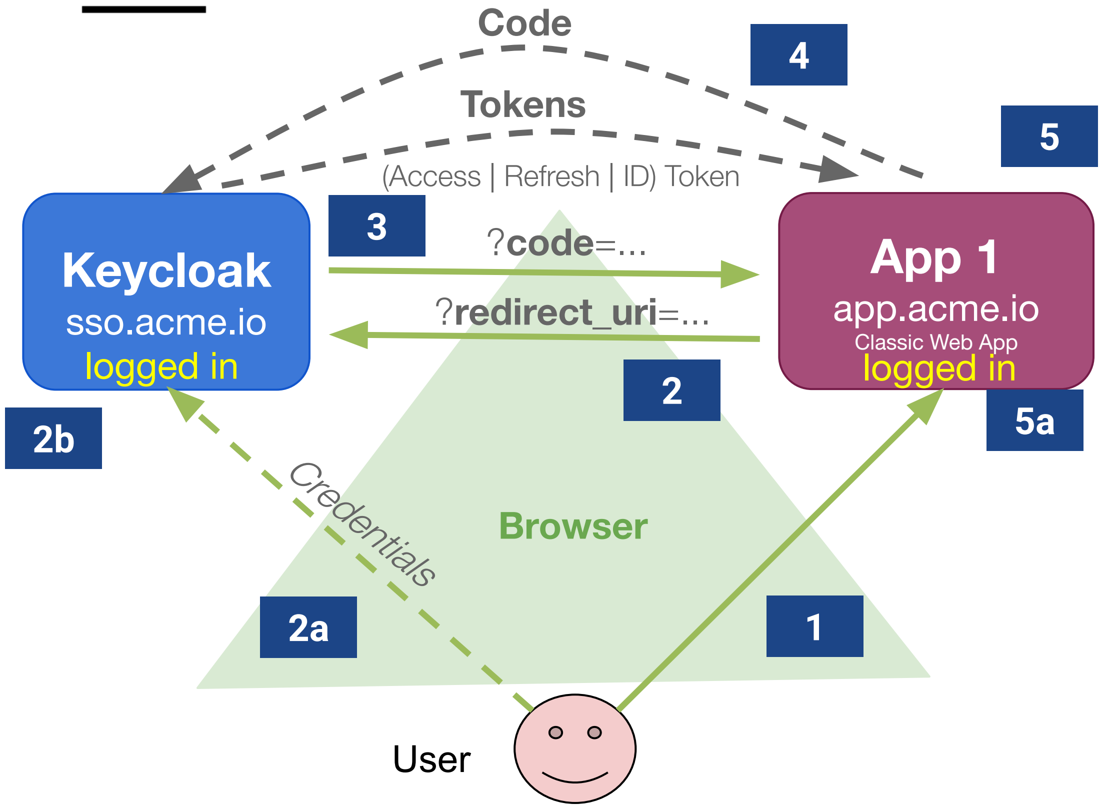
Single Sign-on with Keycloak

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

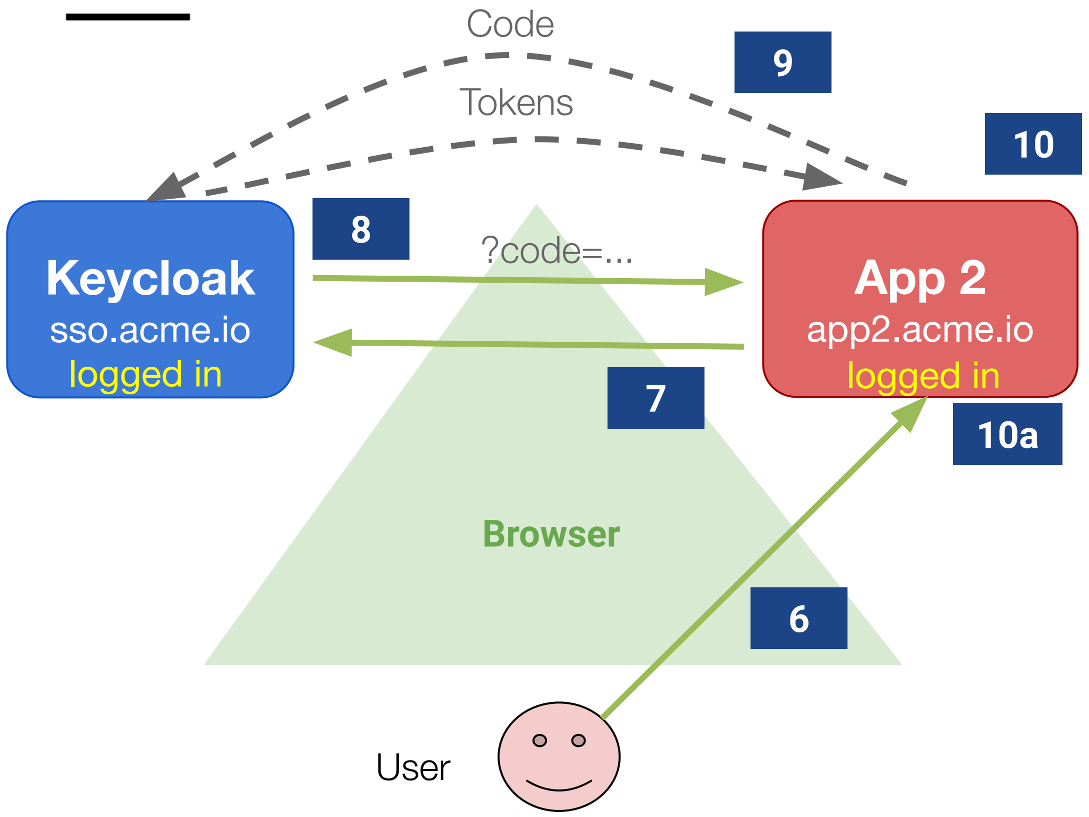
Web SSO with OIDC*: Unauthenticated User



1	Unauthenticated User **accesses** App
2	App **redirects** to Keycloak for Login
2a	User **submits** *Credentials* to Keycloak
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



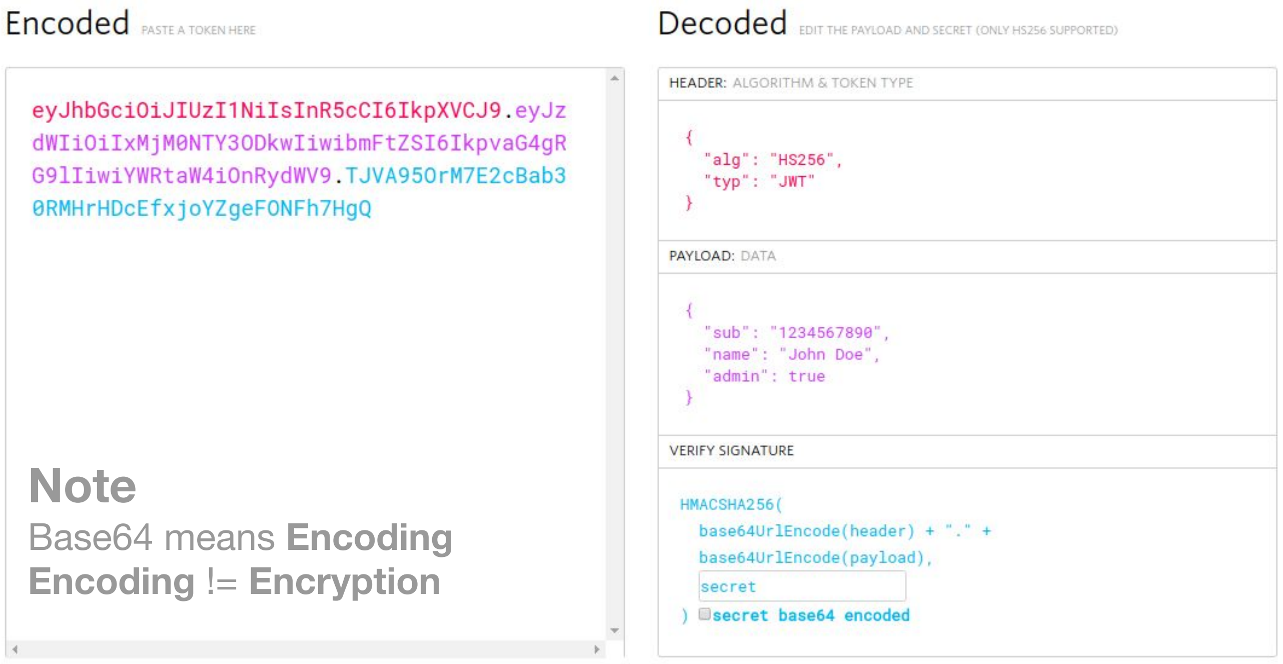
...	
6	Authenticated user **accesses** App 2
7	App 2 **redirects** user to Keycloak for login
8	Keycloak **detects** *SSO Session*, **generates** *code*, **redirects** to App 2
9	App 2 **exchanges** *code* for *tokens* with Keycloak via separate channel
10	App 2 **verifies** *received tokens* and associates it with a *session*
10a	User is now *logged-in* to App 2

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



Encoded PASTE A TOKEN HERE

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

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
```

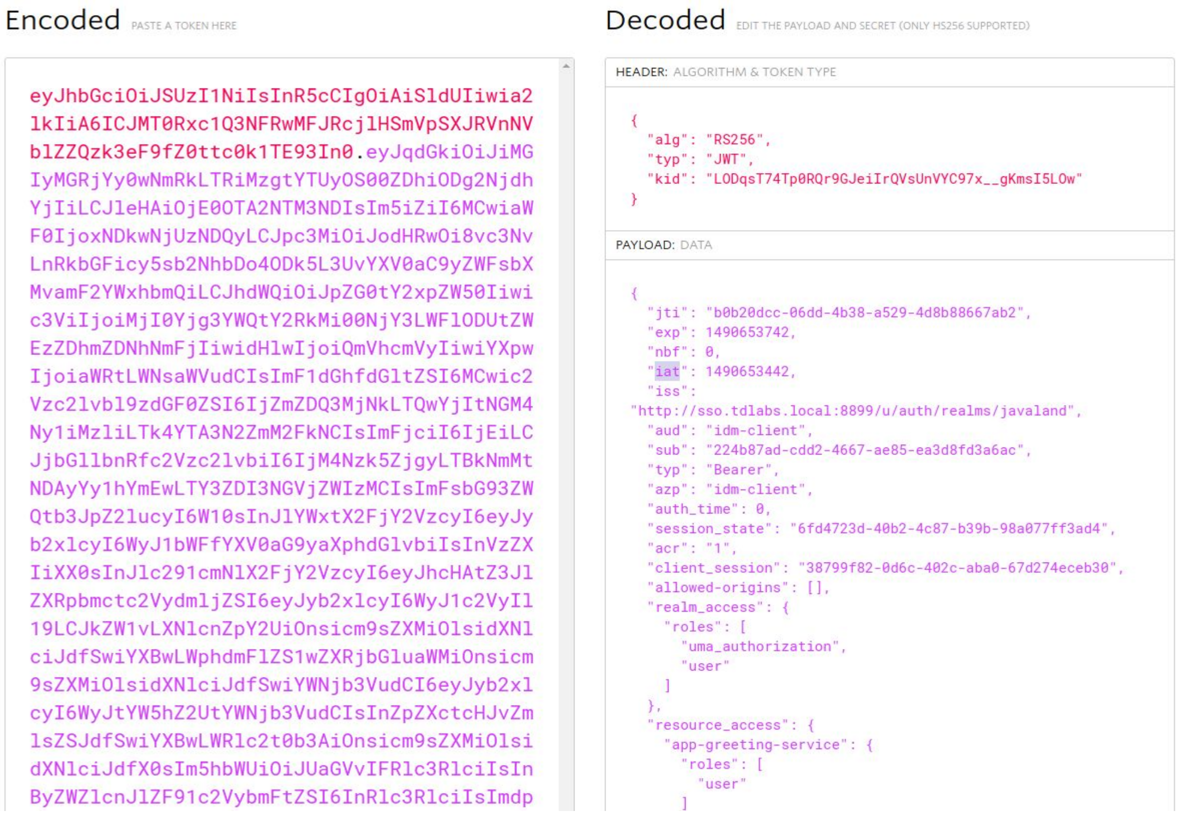
Note

Base64 means **Encoding**

Encoding != **Encryption**

https://jwt.io

Keycloak 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

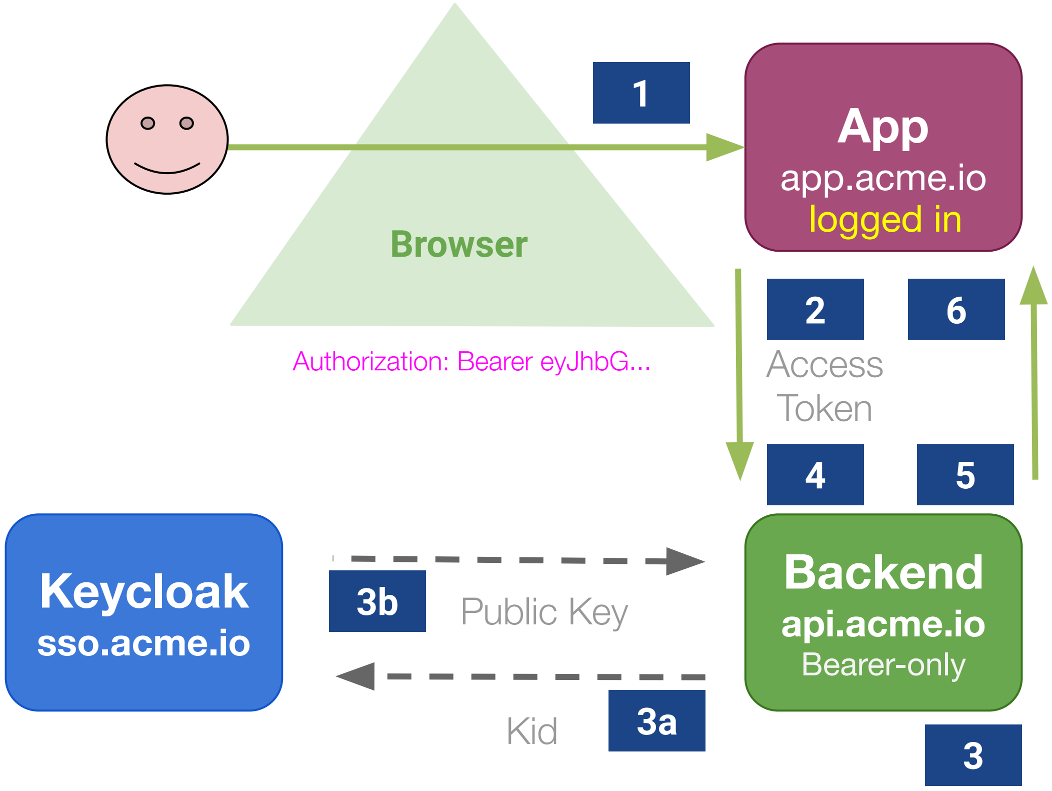
```
{
  "alg": "RS256",
  "typ": "JWT",
  "kid": "LODqsT74Tp0RQr9GJeiIrQVsUnVYC97x__gKmsI5LOw"
}
```

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"
      ]
```

https://jwt.io

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

Keycloak Client Integrations

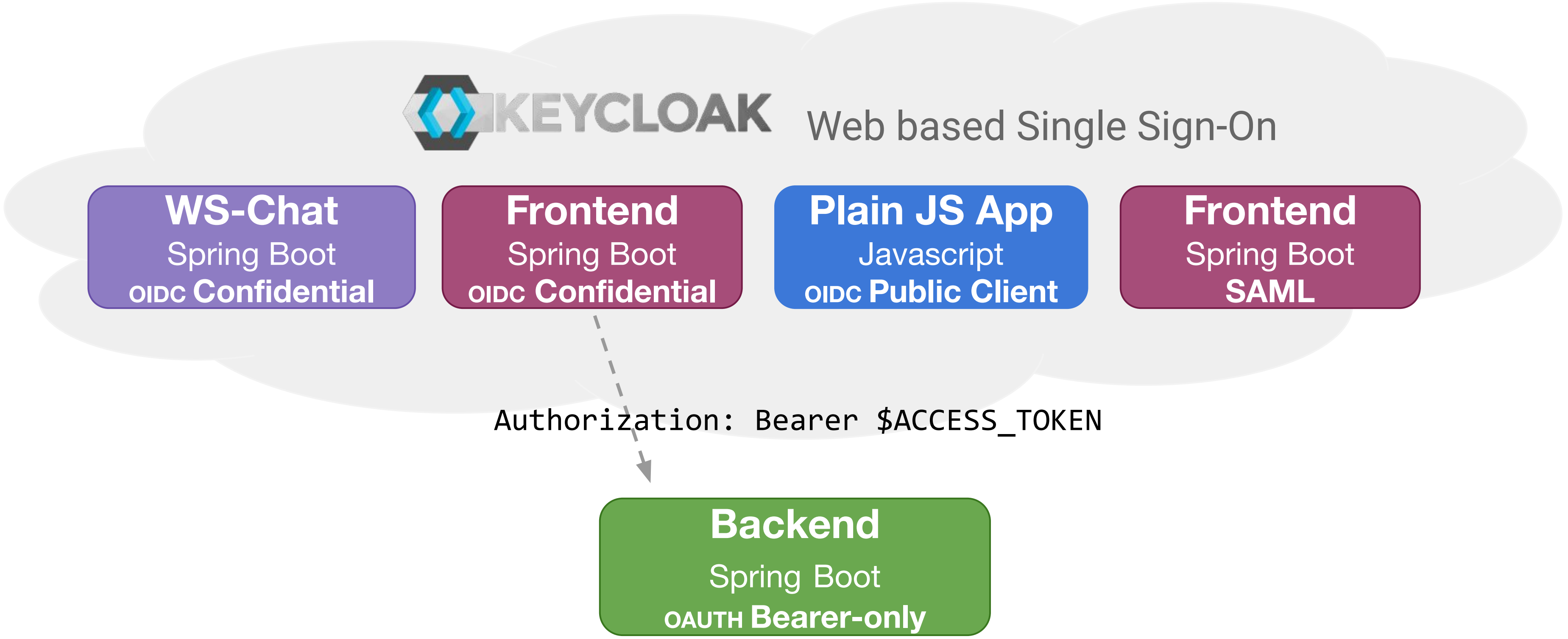
Keycloak Integration Options

- 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 ...
- Many generic library integrations (Java, .Net (Core), Python, Node,...)
  - Spring Security 5 JWT / OAuth / OIDC support might be sufficient for your use-cases
  - see OIDC and SAML
- Reverse Proxies
  - oauth2-proxy, Auth-Proxy, written in Go (Replaces ~~Louketo-Proxy~~ FKA Keycloak Gatekeeper)
  - Apache **mod_auth_openidc** for OpenID Connect and **mod_auth_mellon** for SAML
  - Nginx OpenResty

Keycloak Demo

Securing Apps

Demo Environment



Keycloak Demo

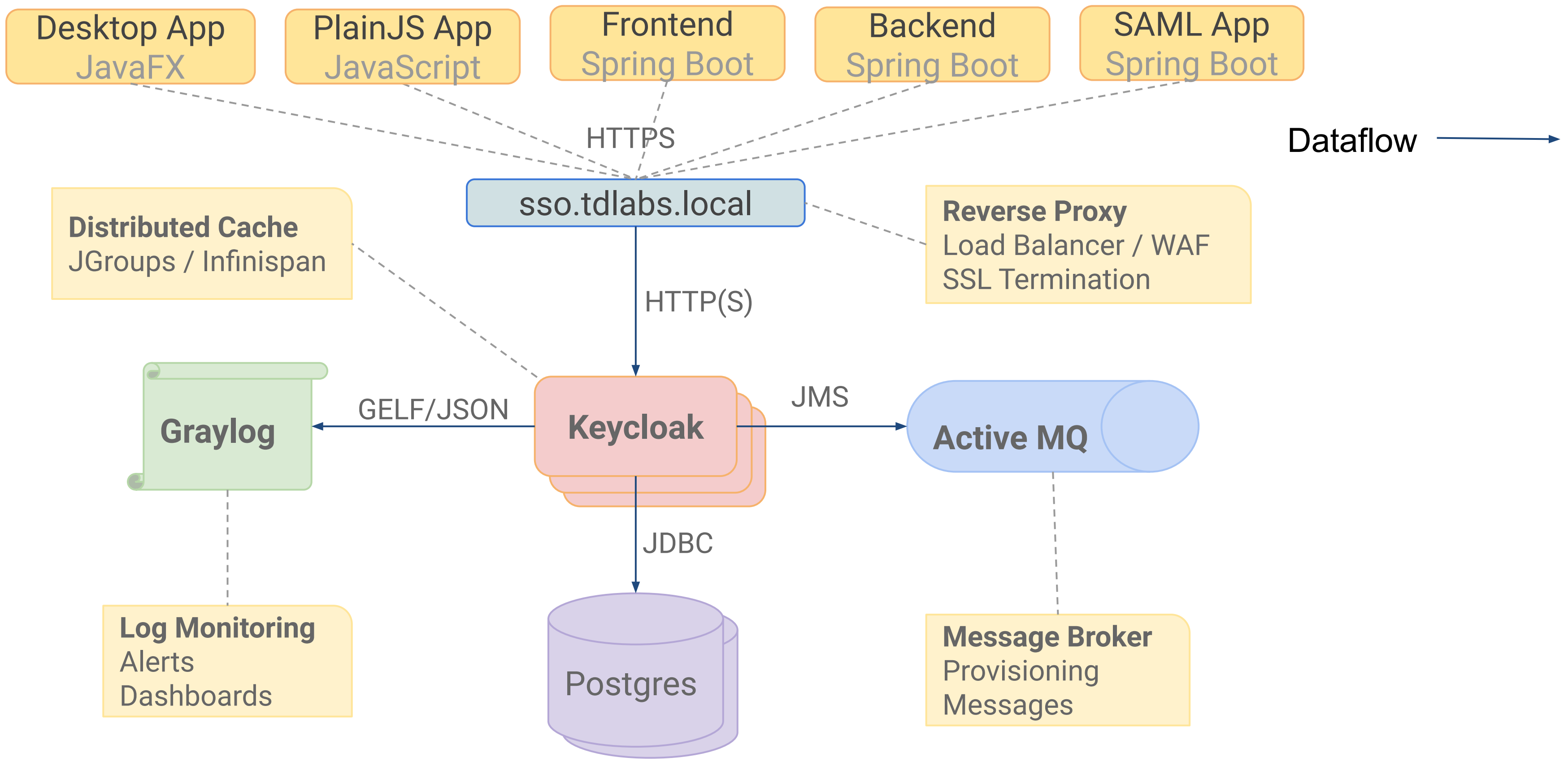
Securing Apps

thomasdarimont/keycloak-docker-demo

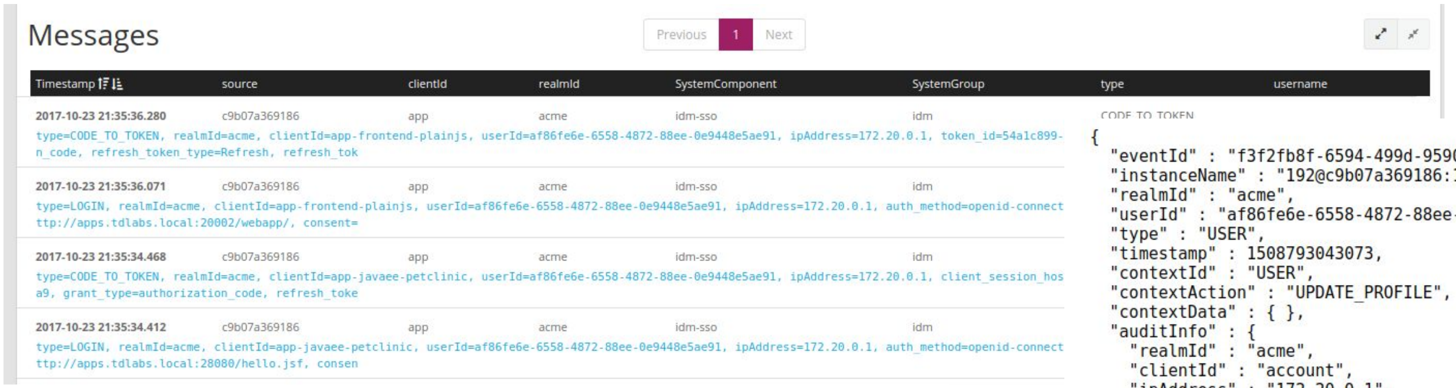
Keycloak in the Field

How can a Keycloak environment look like?

Demo Environment



Keycloak with Graylog + ActiveMQ

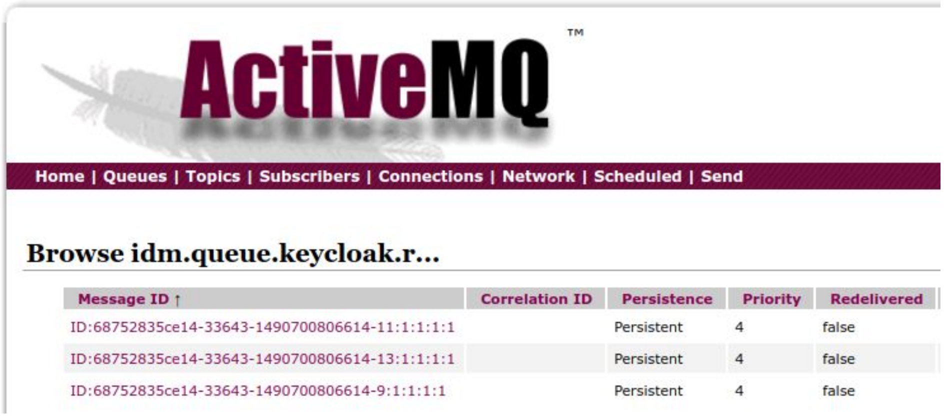


Messages

Previous 1 Next

```
Timestamp                 source          clientId   realmId   SystemComponent   SystemGroup   type            username
2017-10-23 21:35:36.280   c9b07a369186    app        acme      idm-sso           idm           CODE_TO_TOKEN
type=CODE_TO_TOKEN, realmId=acme, clientId=app-frontend-plainjs, userId=af86fe6e-6558-4872-88ee-0e9448e5ae91, ipAddress=172.20.0.1, token_id=54a1c899-
n_code, refresh_token_type=Refresh, refresh_tok
2017-10-23 21:35:36.071   c9b07a369186    app        acme      idm-sso           idm
type=LOGIN, realmId=acme, clientId=app-frontend-plainjs, userId=af86fe6e-6558-4872-88ee-0e9448e5ae91, ipAddress=172.20.0.1, auth_method=openid-connect
ttp://apps.tdlabs.local:20002/webapp/, consent=
2017-10-23 21:35:34.468   c9b07a369186    app        acme      idm-sso           idm
type=CODE_TO_TOKEN, realmId=acme, clientId=app-javaee-petclinic, userId=af86fe6e-6558-4872-88ee-0e9448e5ae91, ipAddress=172.20.0.1, client_session_hos
a9, grant_type=authorization_code, refresh_toke
2017-10-23 21:35:34.412   c9b07a369186    app        acme      idm-sso           idm
type=LOGIN, realmId=acme, clientId=app-javaee-petclinic, userId=af86fe6e-6558-4872-88ee-0e9448e5ae91, ipAddress=172.20.0.1, auth_method=openid-connect
ttp://apps.tdlabs.local:28080/hello.jsf, consen
```

```
{
  "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" ]
    }
  }
}
```



ActiveMQ™
Home | Queues | Topics | Subscribers | Connections | Network | Scheduled | Send

Browse idm.queue.keycloak.r...

Message ID ↑	Correlation ID	Persistence	Priority	Redelivered
ID:68752835ce14-33643-1490700806614-11:1:1:1:1		Persistent	4	false
ID:68752835ce14-33643-1490700806614-13:1:1:1:1		Persistent	4	false
ID:68752835ce14-33643-1490700806614-9:1:1:1:1		Persistent	4	false

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 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

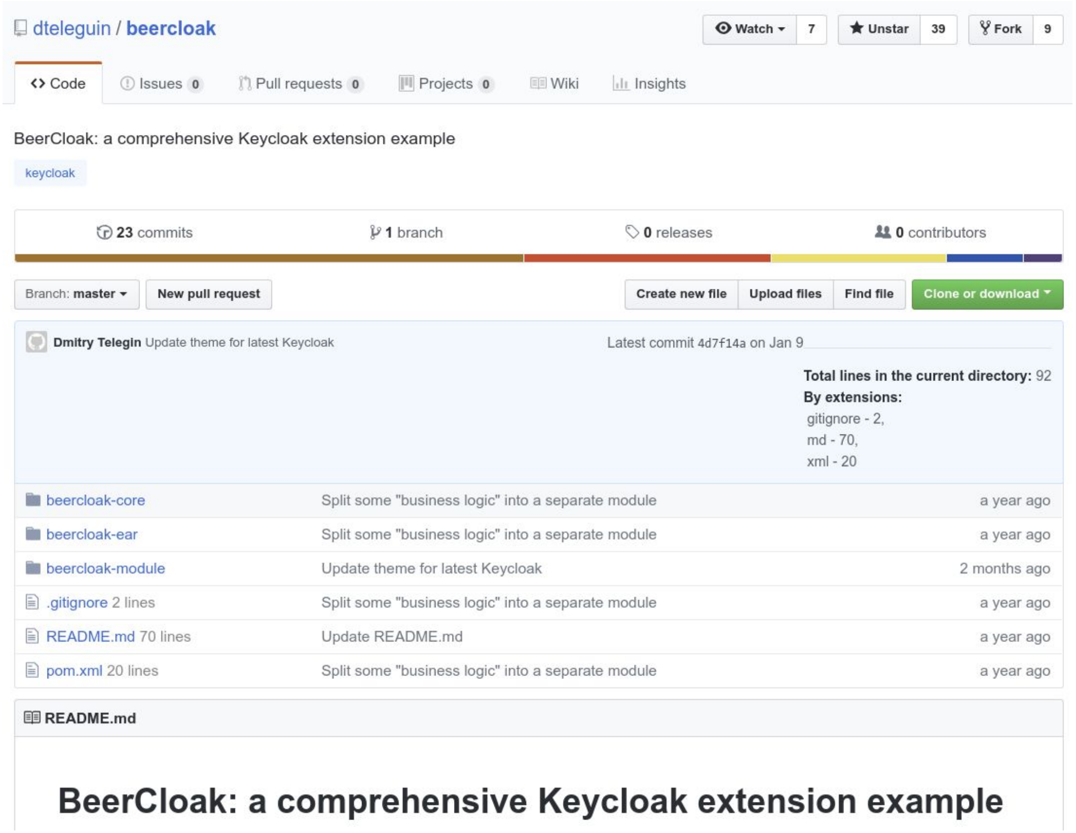
Tips for working with Keycloak

- 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!
  - 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



dteleguin / beercloak

BeerCloak: a comprehensive Keycloak extension example

keycloak

23 commits · 1 branch · 0 releases · 0 contributors

Dmitry Telegin Update theme for latest Keycloak — Latest commit 4d7f14a on Jan 9

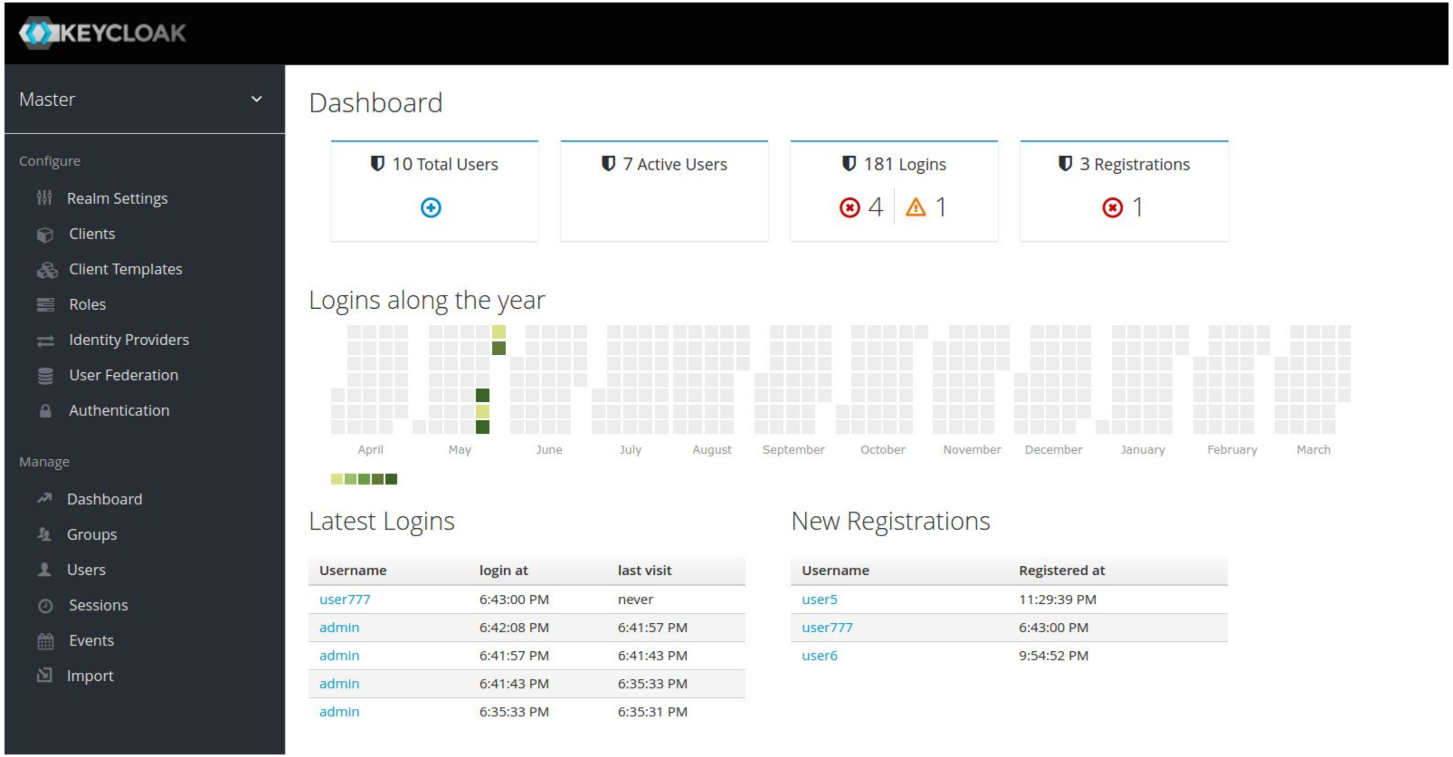
Total lines in the current directory: 92
By extensions:
gitignore - 2,
md - 70,
xml - 20

beercloak-core	Split some "business logic" into a separate module	a year ago
beercloak-ear	Split some "business logic" into a separate module	a year ago
beercloak-module	Update theme for latest Keycloak	2 months ago
.gitignore 2 lines	Split some "business logic" into a separate module	a year ago
README.md 70 lines	Update README.md	a year ago
pom.xml 20 lines	Split some "business logic" into a separate module	a year ago

README.md

BeerCloak: a comprehensive Keycloak extension example

Custom Dashboard Extension



Dashboard

10 Total Users	7 Active Users	181 Logins	3 Registrations
⊕		⊗ 4 ⚠ 1	⊗ 1

Logins along the year

Latest Logins

Username	login at	last visit
user777	6:43:00 PM	never
admin	6:42:08 PM	6:41:57 PM
admin	6:41:57 PM	6:41:43 PM
admin	6:41:43 PM	6:35:33 PM
admin	6:35:33 PM	6:35:31 PM

New Registrations

Username	Registered at
user5	11:29:39 PM
user777	6:43:00 PM
user6	9:54:52 PM

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 (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
- 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
  - Uses Resource Owner Password Credentials Grant Flow (`grant_type=password`)
  - Sends HTTP `POST` request to `/token` Endpoint (`client_id, username, password`)
  - Keycloak returns Tokens (Access-, ID-, Refresh-Token)
  - Client needs to parse & validate tokens
  - Client sees password → *Password Anti-Pattern*
- KeycloakInstalled Adapter
  - Uses OAuth2 *Authorization Code Flow* for Desktop / CLI apps (grant_type=code)
  - 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

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

2 Export keycloak.json for Client

```json
{ "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

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

4 Trigger Browser login

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

5 Read current username

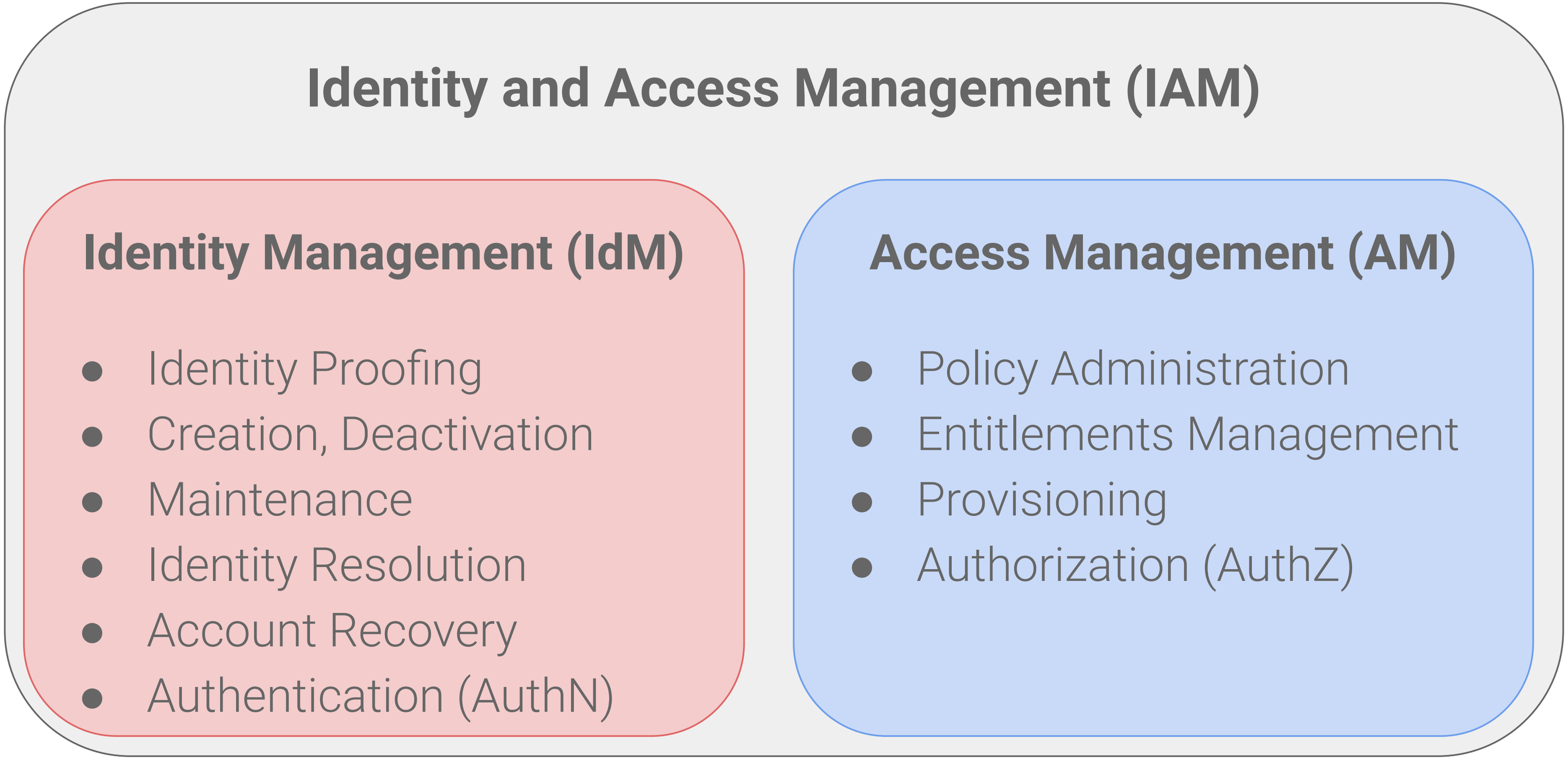
```java
keycloak.getIdToken().getPreferredUsername()
```

6 Read & use AccessToken

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

7 Trigger Browser Logout

```java
keycloak.logout()
```



Identity and Access Management (IAM)

Identity Management (IdM)

- Identity Proofing
- Creation, Deactivation
- Maintenance
- Identity Resolution
- Account Recovery
- Authentication (AuthN)

Access Management (AM)

- Policy Administration
- Entitlements Management
- Provisioning
- Authorization (AuthZ)