SPRING IO

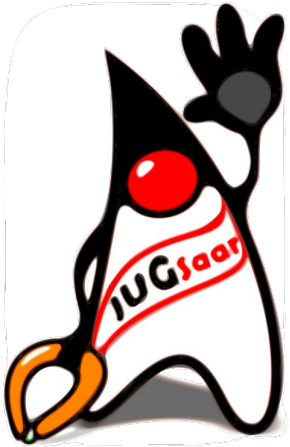
#springio19

How to secure your Spring Apps with Keycloak

Thomas Darimont
@thomasdarimont

Thomas Darimont

- Fellow codecentric
- Pivotal Spring Data 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

Keycloak Overview

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

24 Apr

Keycloak 6.0.1 released

17 Apr

Keycloak 6.0.0 released

06 Mar

Keycloak 5.0.0 released

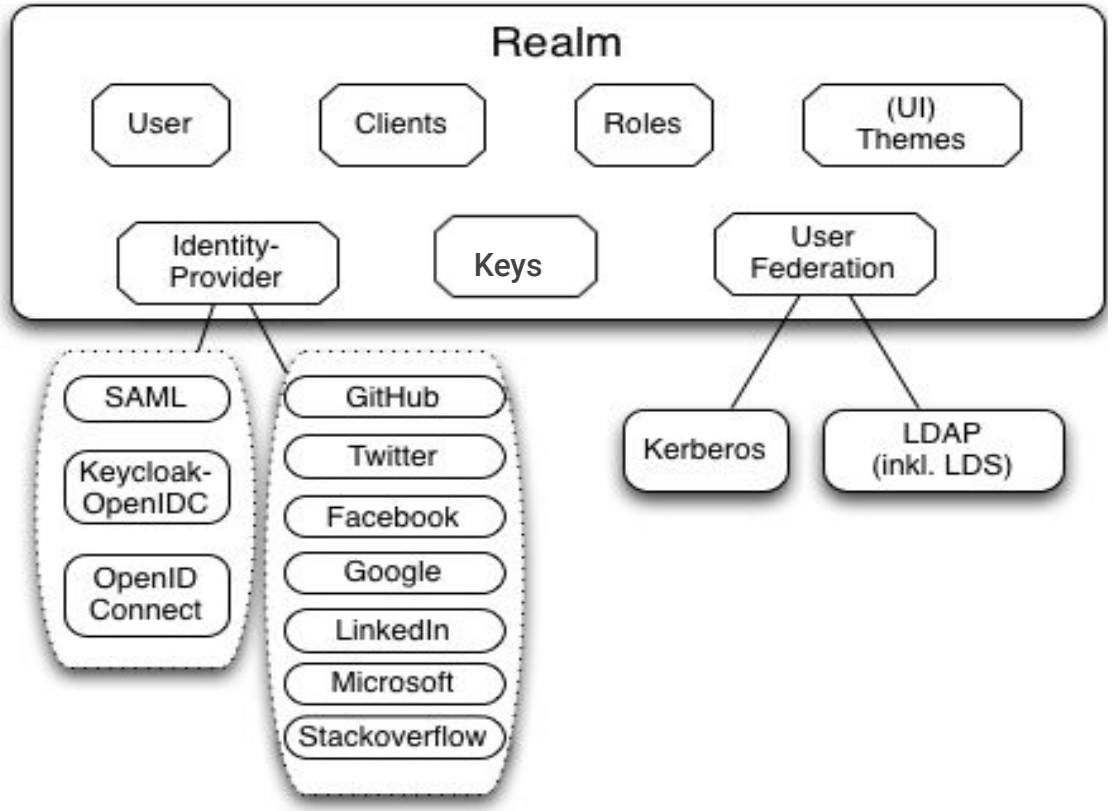
Project

- Java based **Authentication** & **Authorization** Server
- Started in 2013, broad adoption since 2015
- Apache License, **Red Hat** Developers
- Current version **6.0.1.Final** ~ **every 6 Weeks**
- Commercial Offering Available → **Red Hat SSO**
- Vital **Community** with **300+** Contributors **1.900+** Forks
- 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
- Flexible **Authentication** and **Authorization**
- **Multi-Factor Authentication** One-time Passwords
- **Social Login** Google, Facebook, Twitter,...
- Provides centralized **User Management**
- Supports **Directory Services**
- **Customizable** and **Extensible**
- **Easy** Setup and Integration

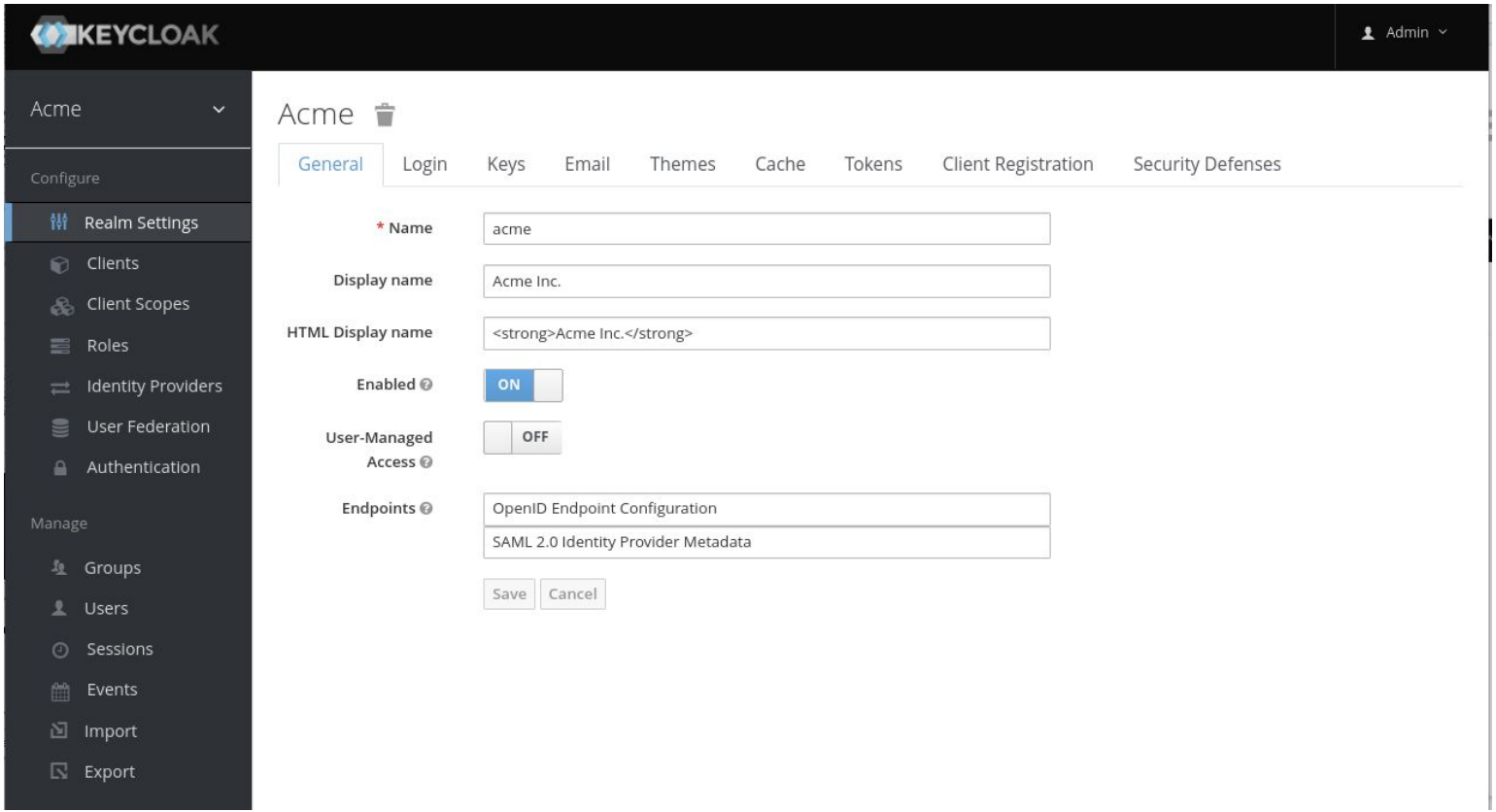
Main Concepts



Keycloak Quick Tour

Admin Console

Admin Console



Technology Stack 6.0.1.RELEASE

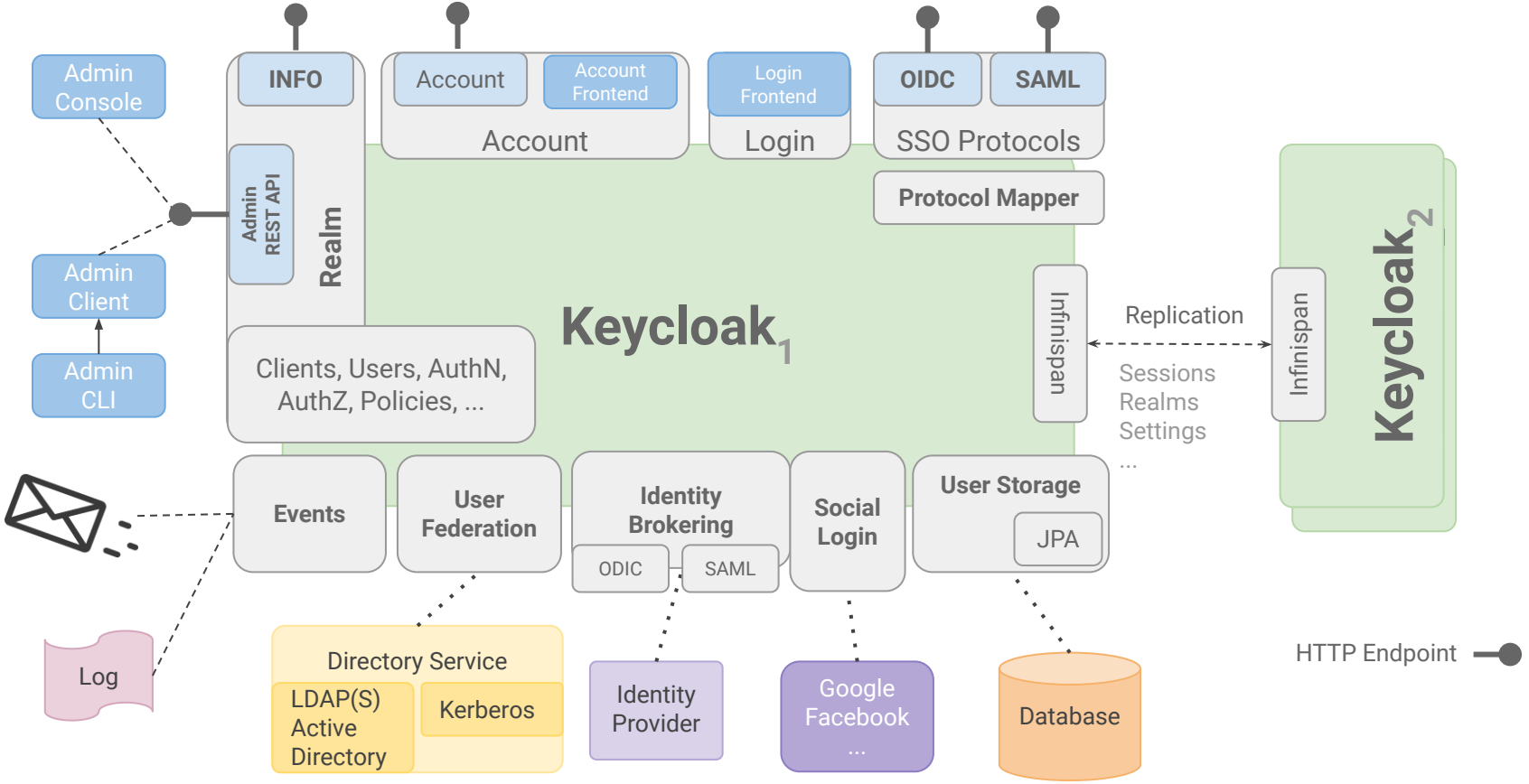
Admin Console

- Angular JS (1.6.x)
- PatternFly
- Bootstrap

Keycloak Server

- Wildfly 16.0.x
- JAX-RS (Resteasy)
- JPA (Hibernate)
- Infinispan (JGroups)
- Freemarker
- Jackson 2.x
- JBoss Logging
- Apache Directory API
- Commons HTTP Client

Server Architecture



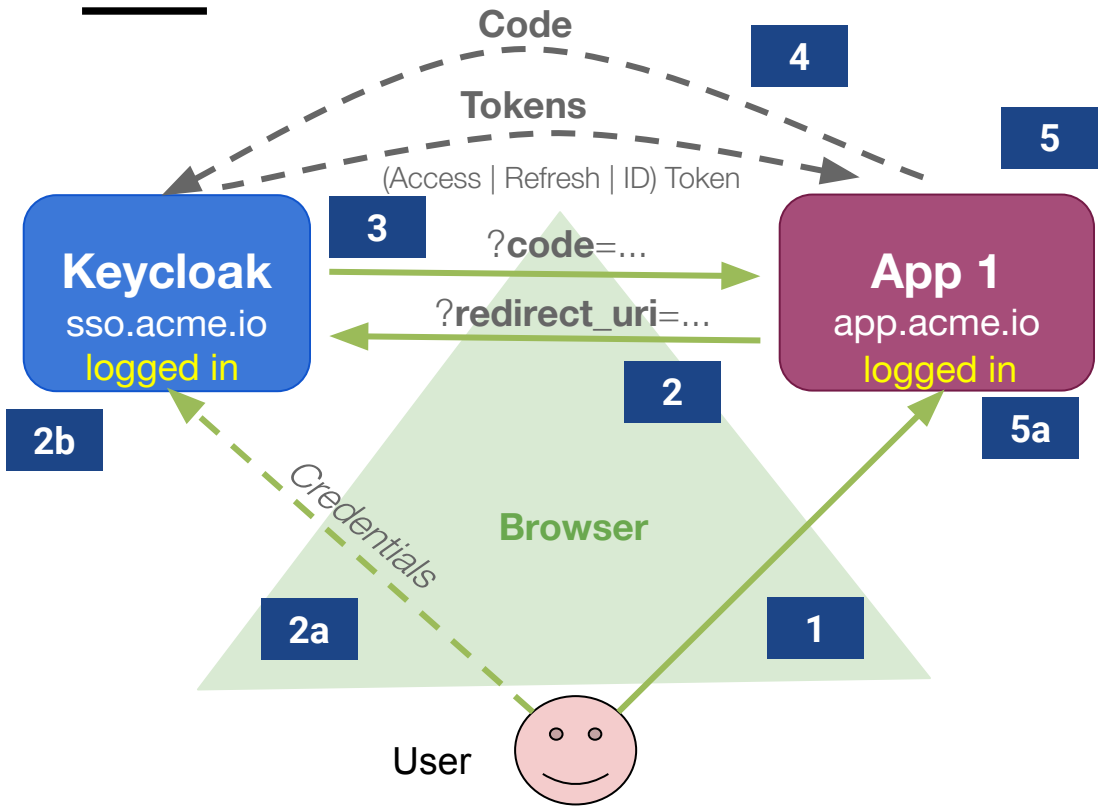
Single Sign-on with Keycloak

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

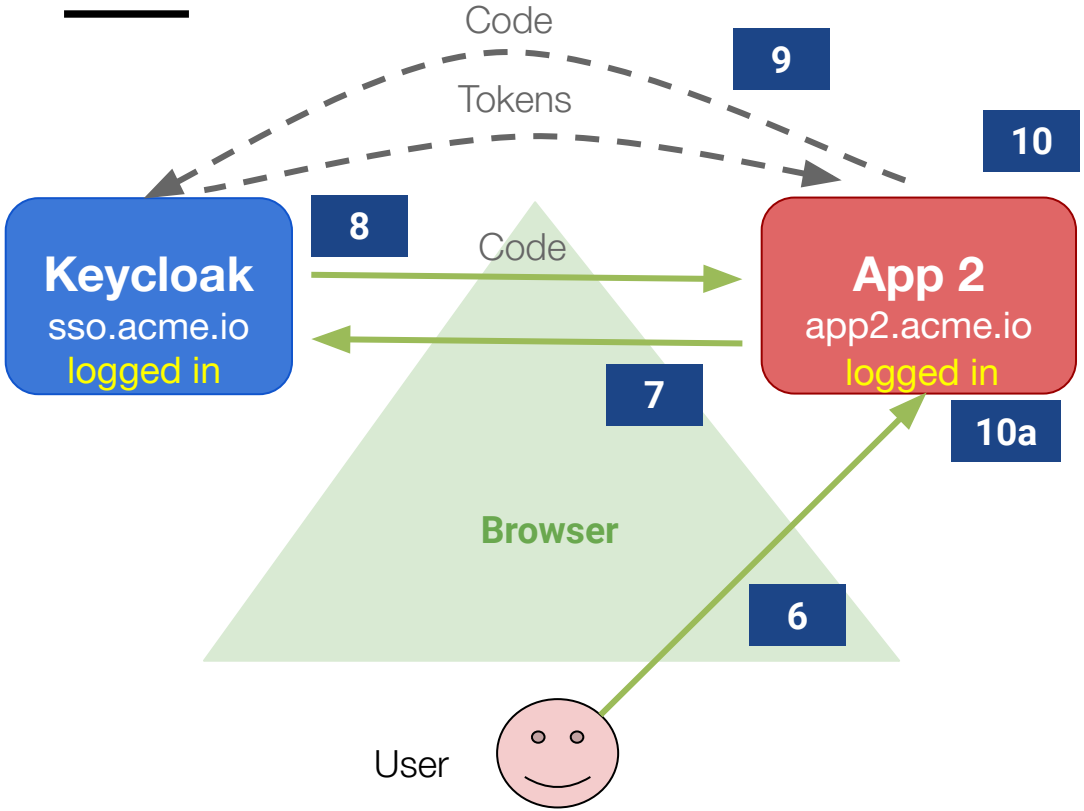
Web SSO with OIDC*: Unauthenticated User



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

*) 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

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*

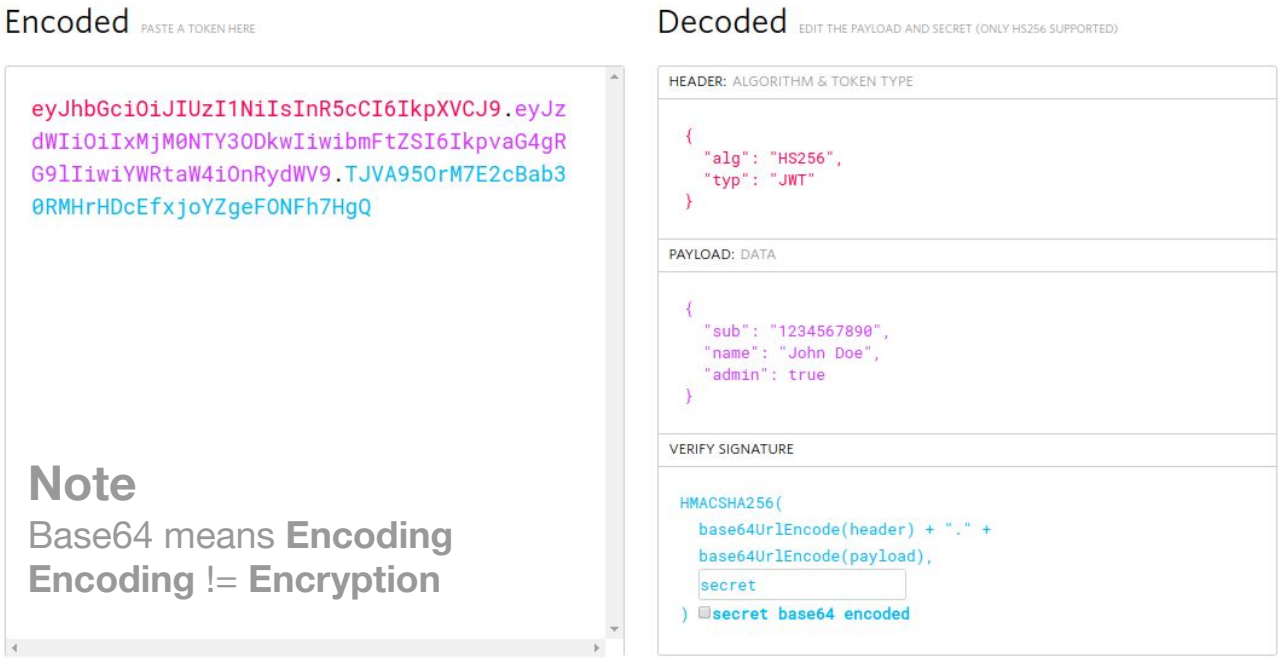
10a User is now *logged-in* to App 2

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** 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

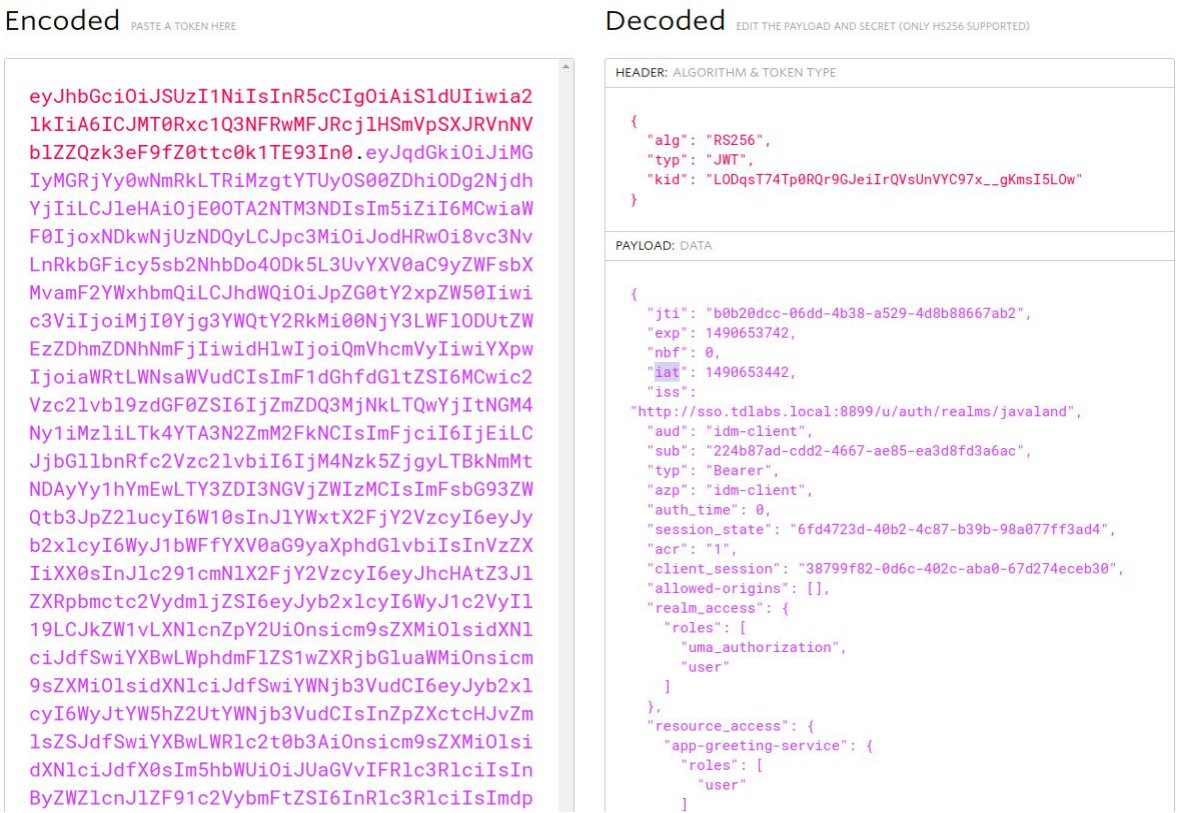
JSON Web Tokens

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



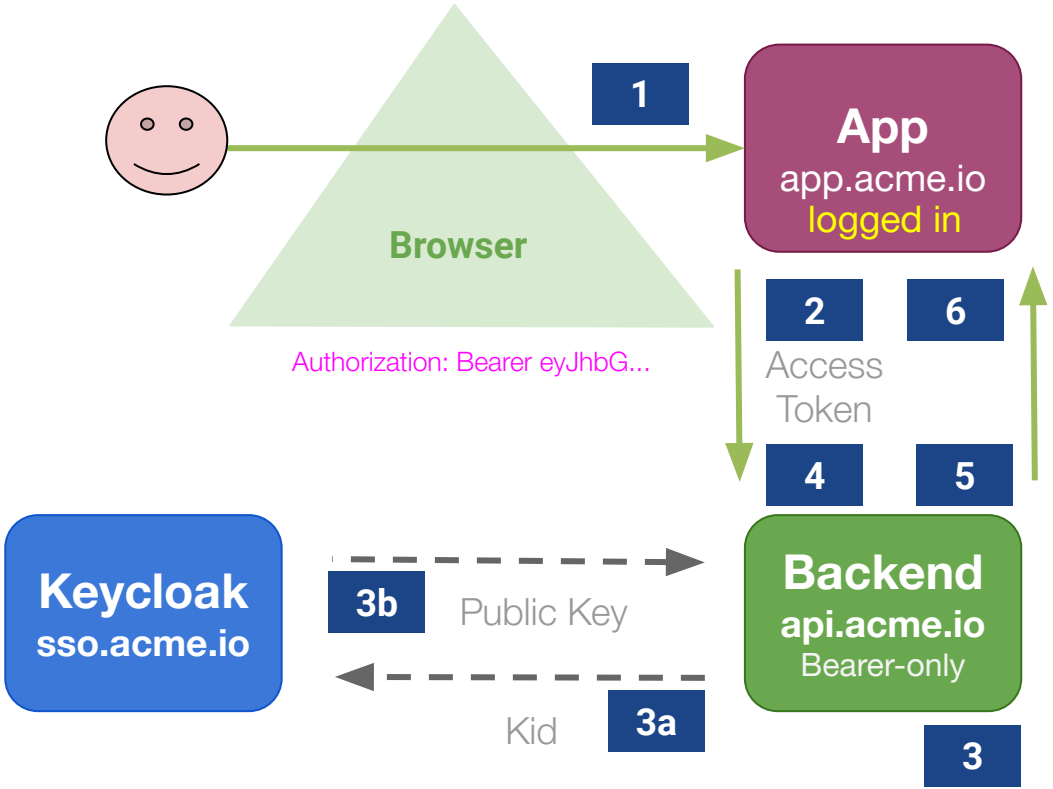
https://jwt.io

Keycloak JSON Web Token Example



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
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

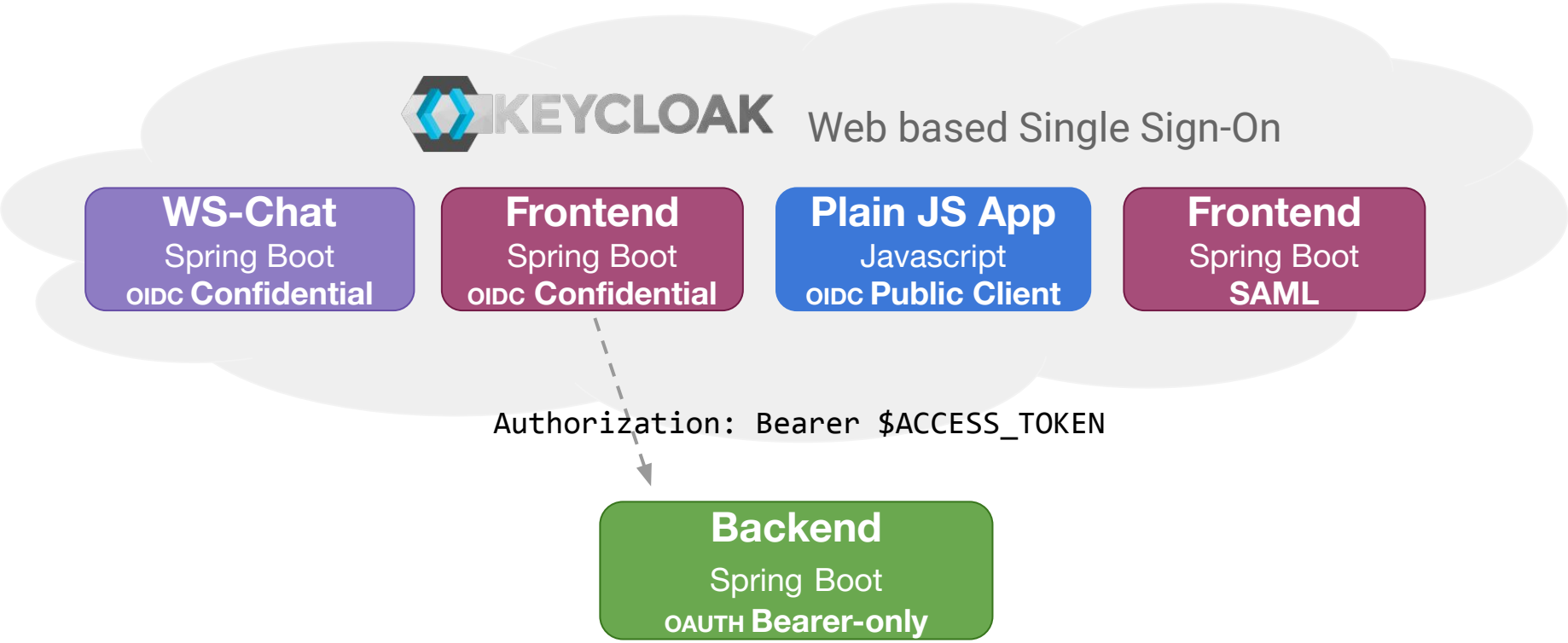
Keycloak Integration Options

- 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 ...
- 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 OIDC and SAML

Keycloak Demo

Securing Apps

Demo Environment



Keycloak Demo

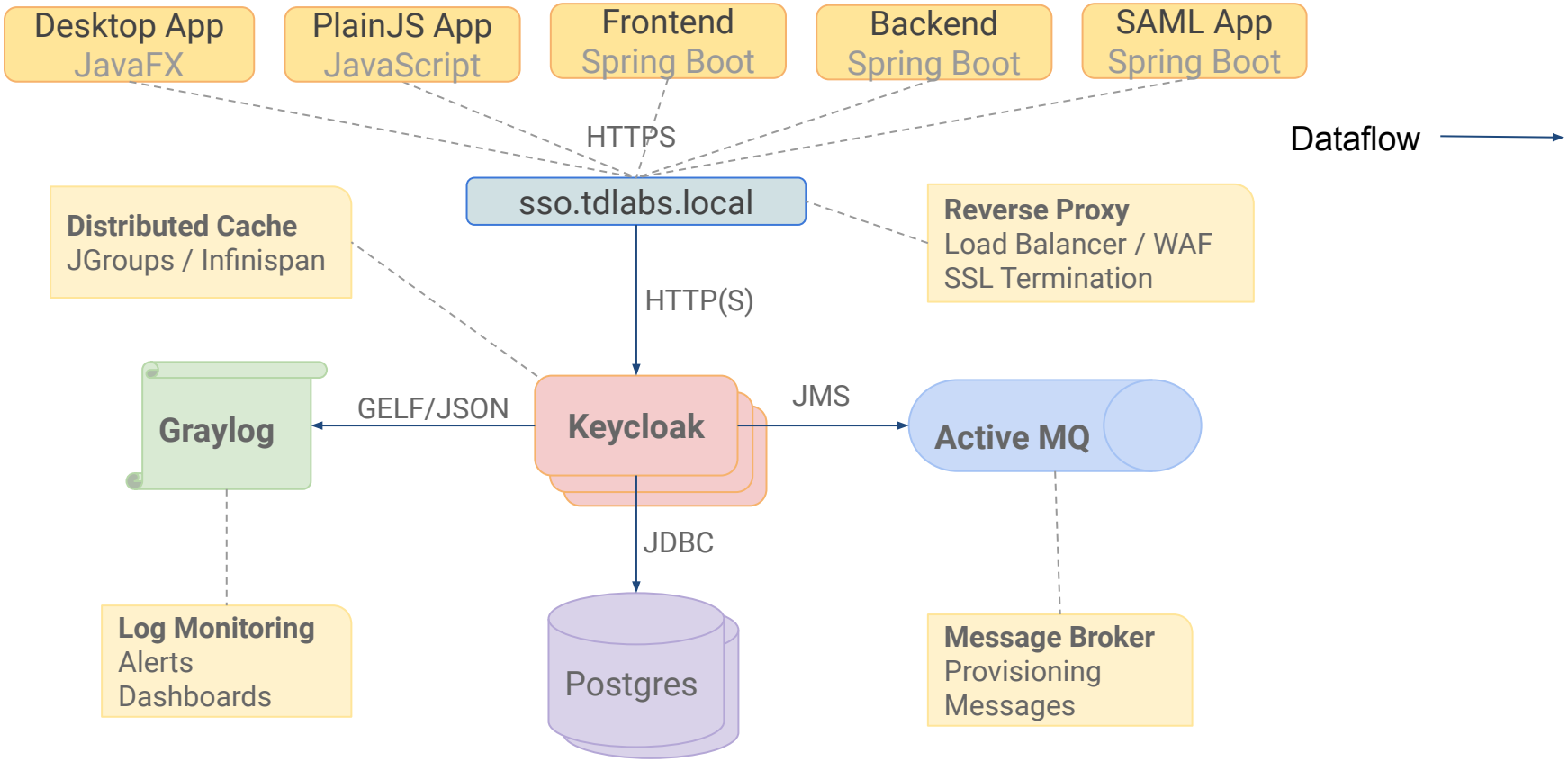
Securing Apps

thomasdarimont/keycloak-docker-demo

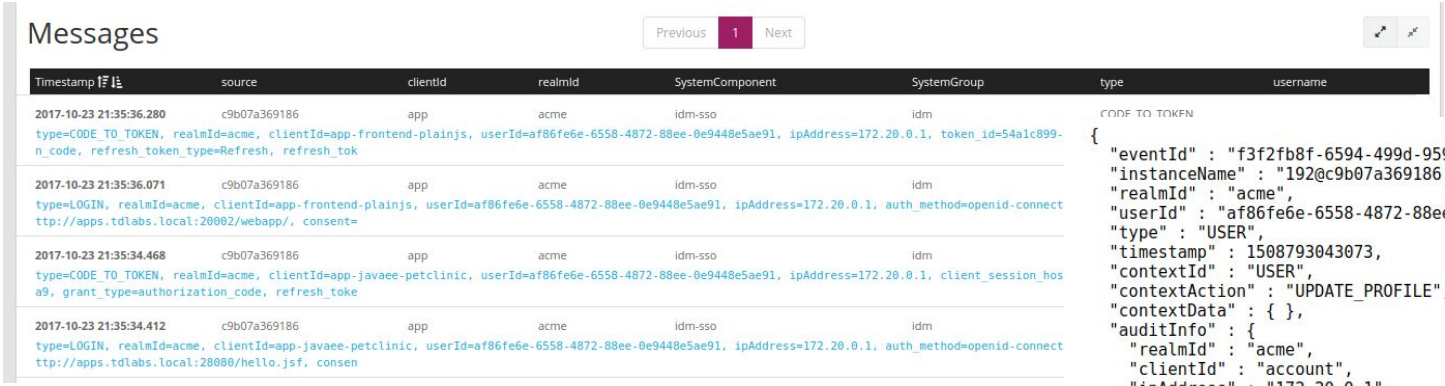
Keycloak in the Field

How can a Keycloak environment look like?

Demo Environment



Keycloak with Graylog + ActiveMQ




```
{
  "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 modern Identity Management

Thanks!

Thomas Darimont
@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

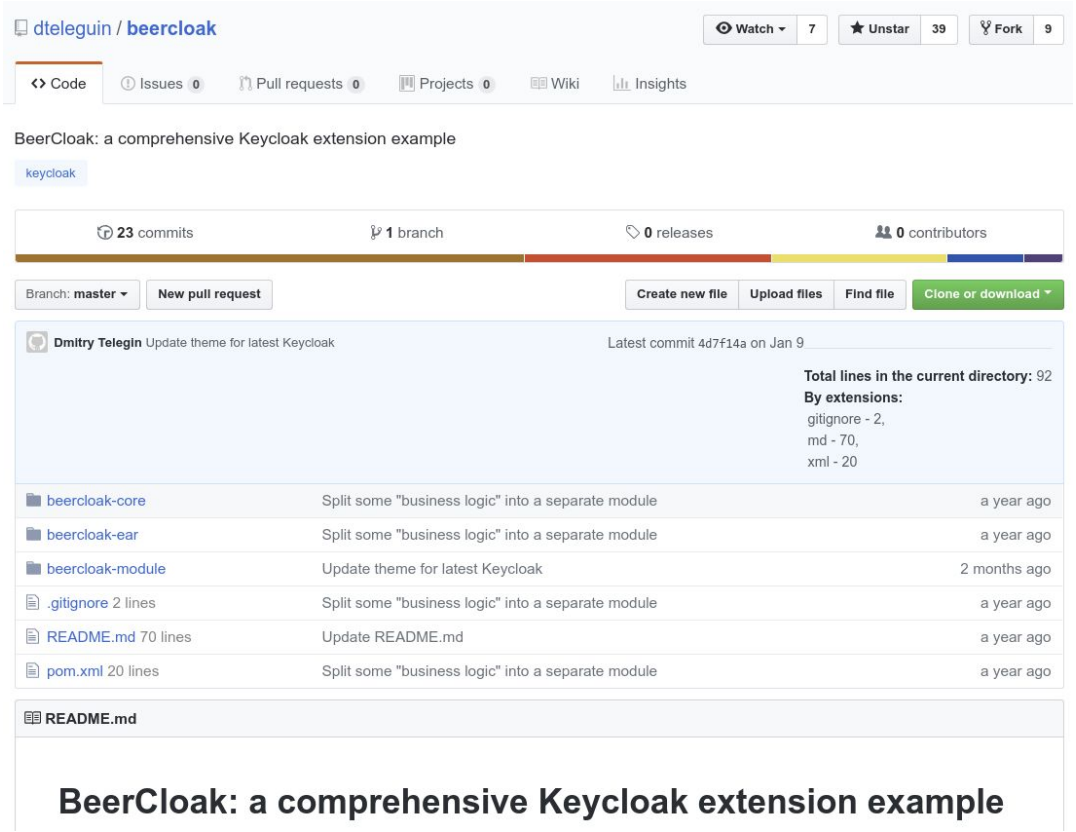
Tips for working with Keycloak

- 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

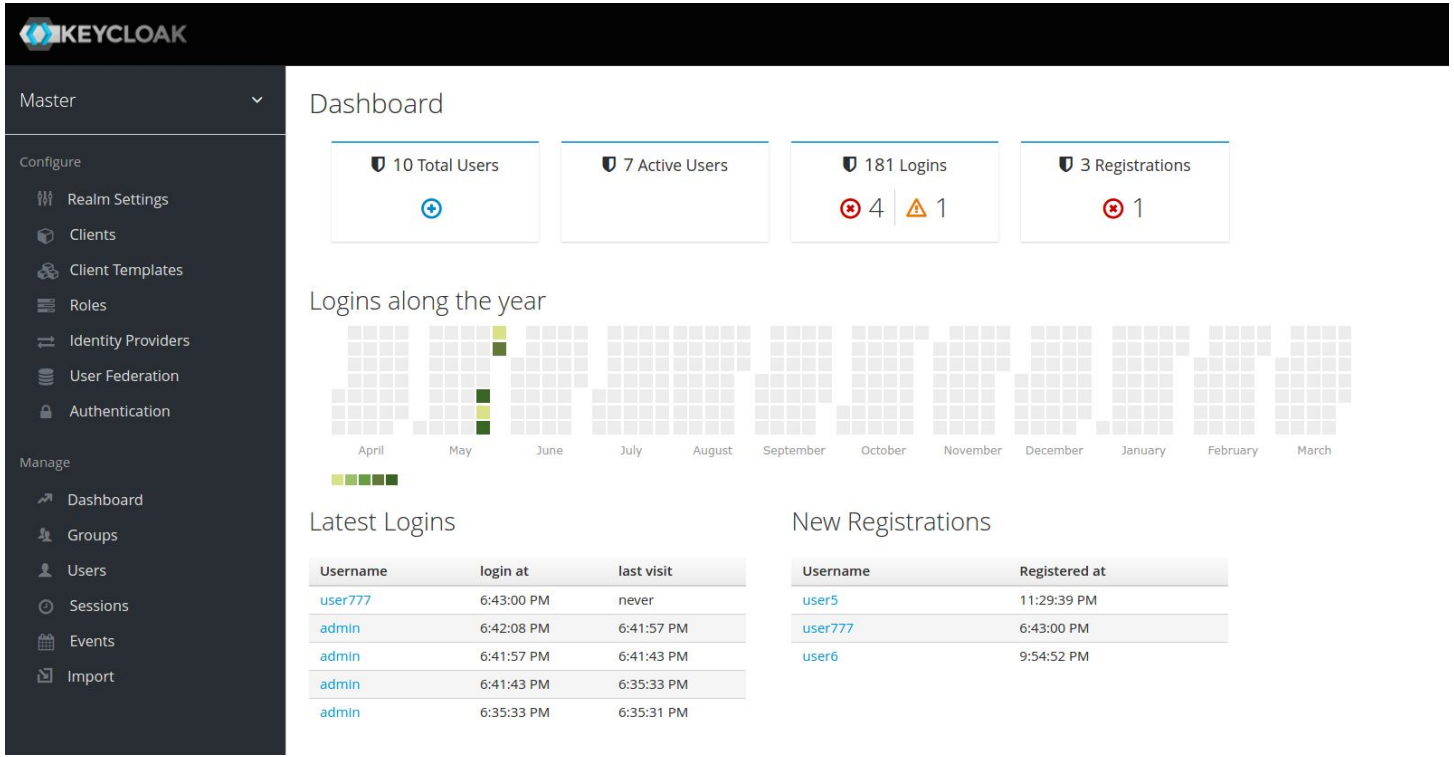
Keycloak Extension Points

- 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

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

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
  - 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()
```