Keycloak
Prosty sposób na bezpieczeństwo i uwierzytelnianie w nowoczesnych aplikacjach

Bolesław Dawidowicz
Red Hat

Andrzej Goławski
PPL, PANSA, OSEC

Before we begin…

Agenda - Part 1 [Slides]

- What is Keycloak ~ 3min
- Token based security ~ 10min
- Quick tour on modern standards ~10min
  - SAML2
  - OAuth2
  - OpenID Connect

Agenda - Part 2

- Key Keycloak features ~ 5min

Agenda - Part 3 [Demo]

- Awesome Demo by Andrzej :)

What is Keycloak?

Identity Solution for Modern Applications, Services and APIs.

Keycloak provides

- Authentication
- SSO
- Session Management
- User Self Services
- Auditing
- Authorization

Keycloak focuses on…

- Modern Applications
  - Web & Mobile
  - Client Side (HTML5/JS) & Server Side
  - (Micro)Services and APIs
  - Any technology being used in them

Is **NOT ONLY** for JAVA applications

Java being implementation detail…

Keycloak integrates well with

- Existing infrastructure
- LDAPs / RDBMSs / Custom user storage
- External Identity Providers
- Kerberos
- Social Providers

Relies on standards

- SAML 2
  - Keycloak is SAML2 IdP and provides SAML2 SP libraries
- OpenID Connect
  - Keycloak is OIDC Authorisation Server and uses this standard in its client adapters
- Kerberos
  - Provides Web SSO integration

Focusing on being super easy to integrate with

Will show how easy during second part of this presentation

Provides very good user experience with extensive Management UI

keycloak

Admin

Master

Configure

- Realm Settings
- Clients
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Users
- Sessions
- Events

Master

General | Login | Keys | Email | Themes | Cache | Tokens | Security Defenses

User registration	ON
Email as username	ON
Edit username	ON
Forget password	ON
Remember Me	ON
Verify email	ON
Require SSL	external request

Save Cancel

Thats all..

Now lets take a step back… and set some context

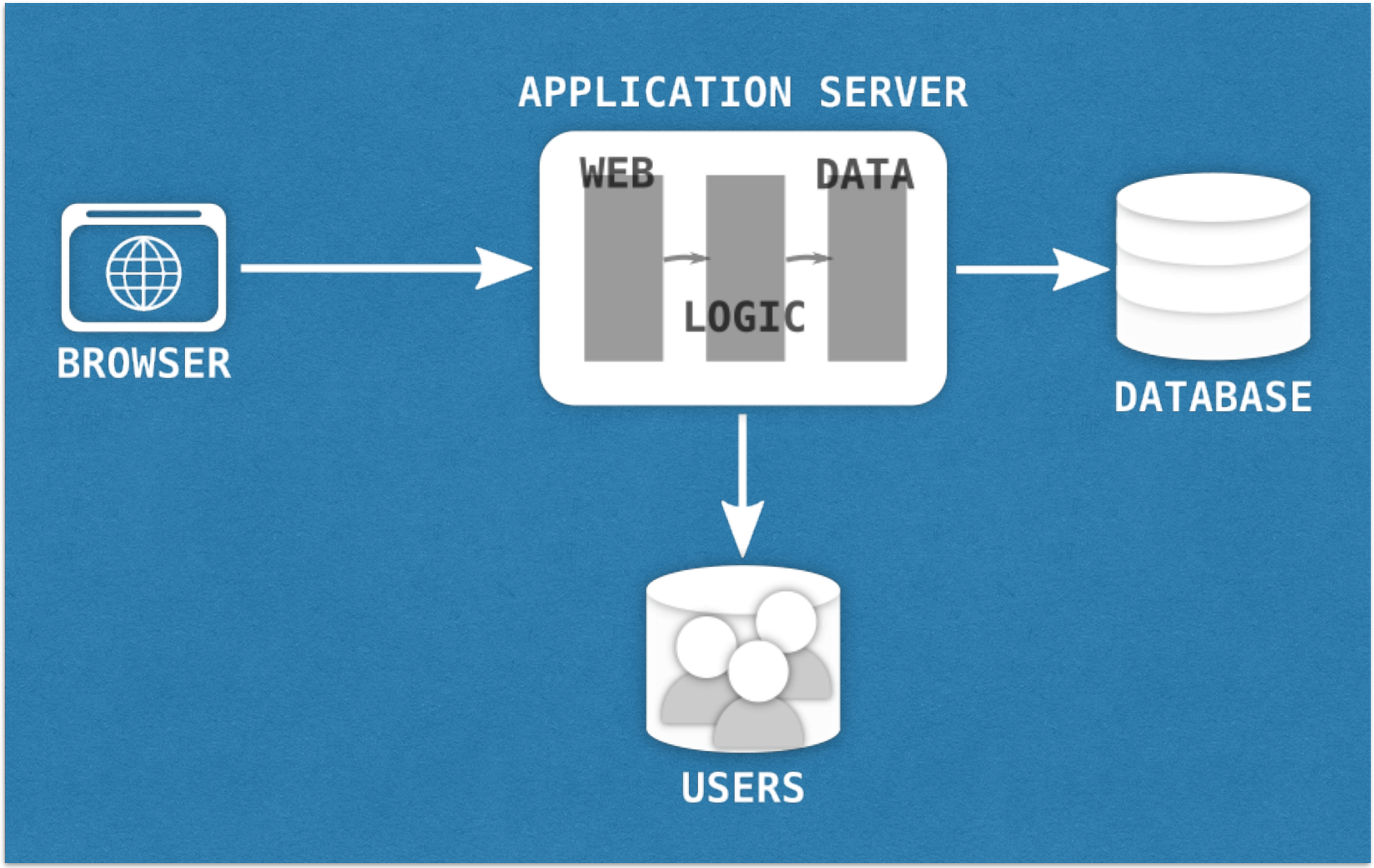
Modern security architecture and standards

and what is happening behind the scenes

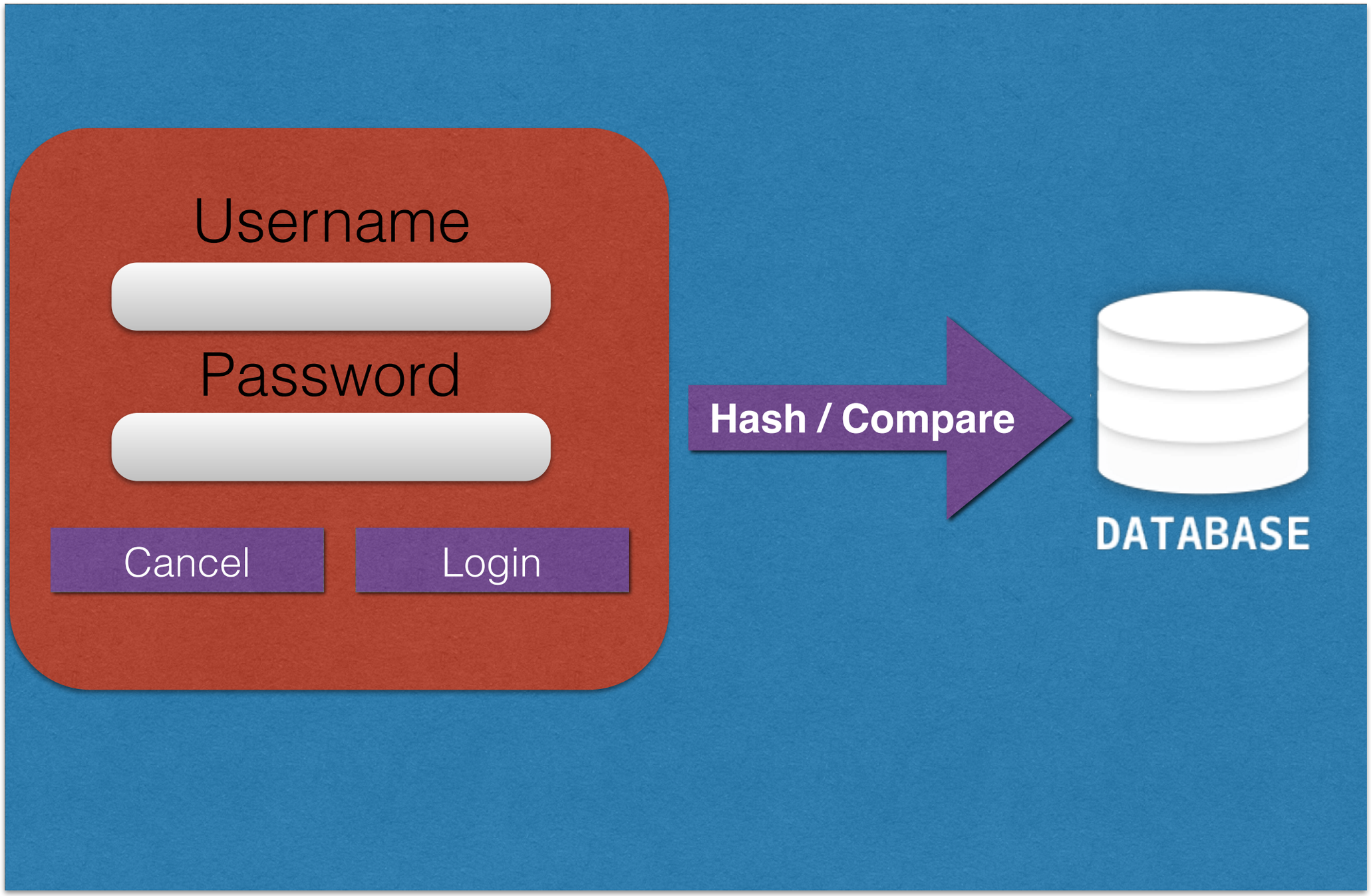
In the old days…

… all used to be simple ;)

Server Side Apps



Form based security



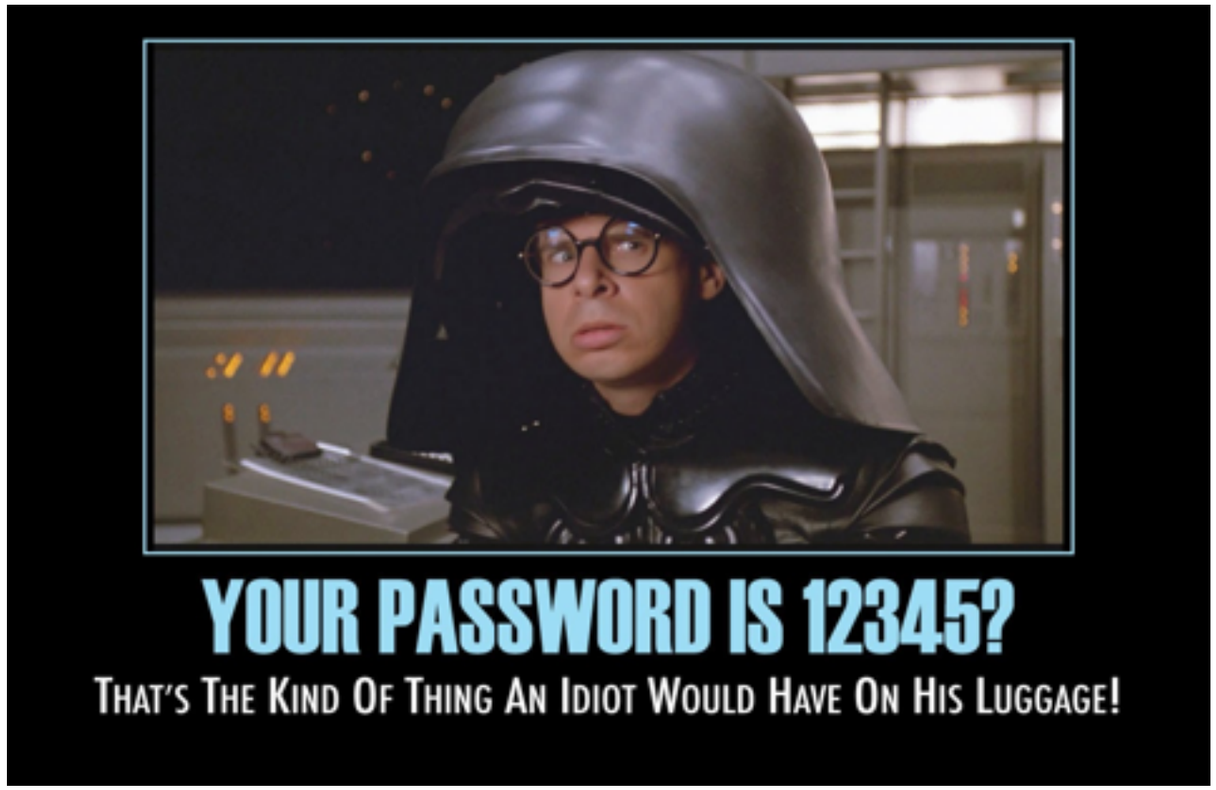
Old Architecture

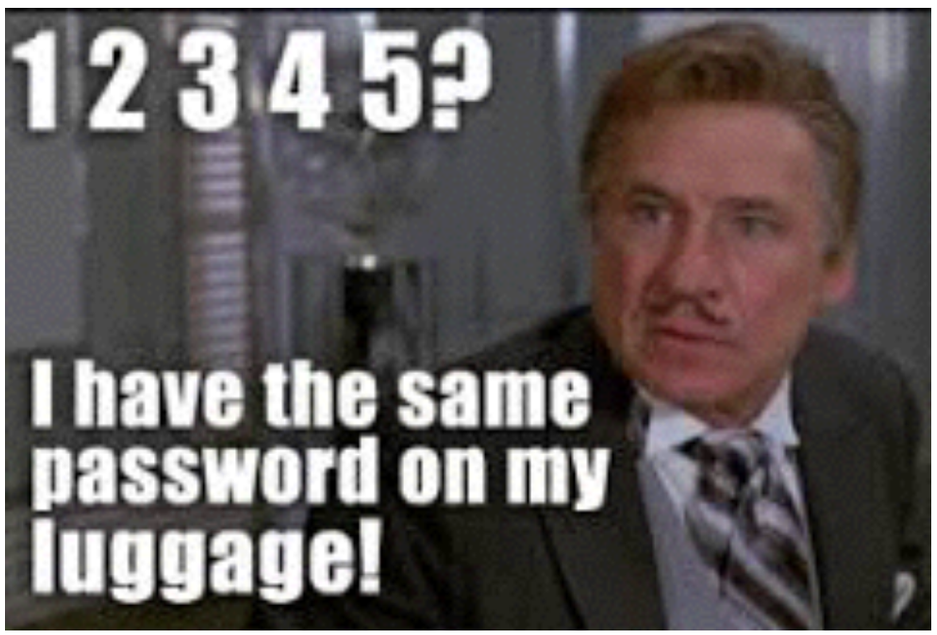
- Usually server side
- Passwords
- Simple session management
  - Invalidate to logout
  - Cookies

Not anymore…

Passwords

- Not easy to store and validate
- Hard to enforce using strong and unique ones
- Problematic to pass around in safe manner





New credential types

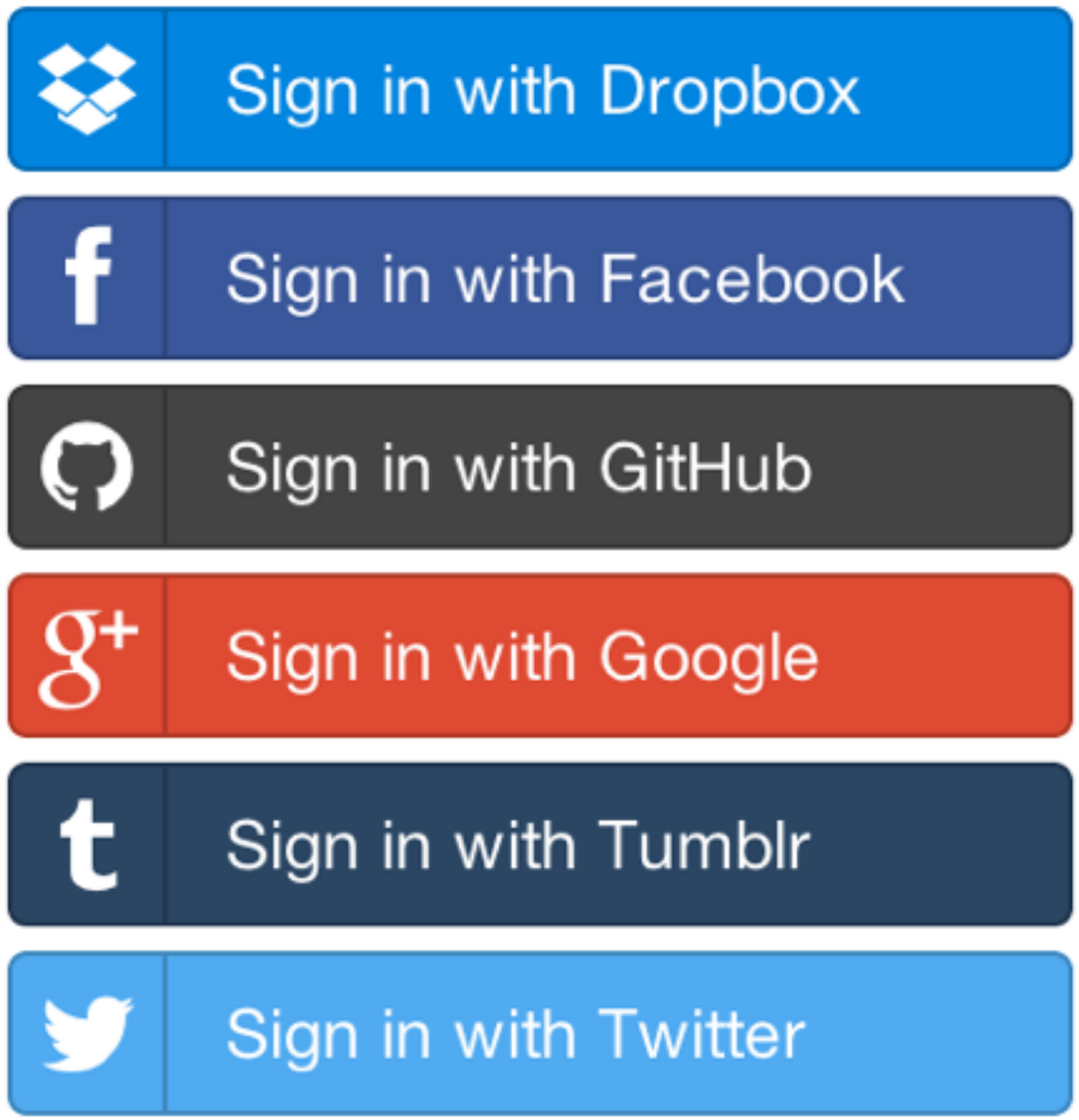
- Not always easy to adapt application to support them

Session Management

- Simple for server side applications
- Complex for mobile and HTML5 applications

External Users

- Other Companies
  - Our partners or customers
  - No access to their database
- Social Networks



Single Sign **On**

Single Sign **Out**

Invalidate all your sessions in one place

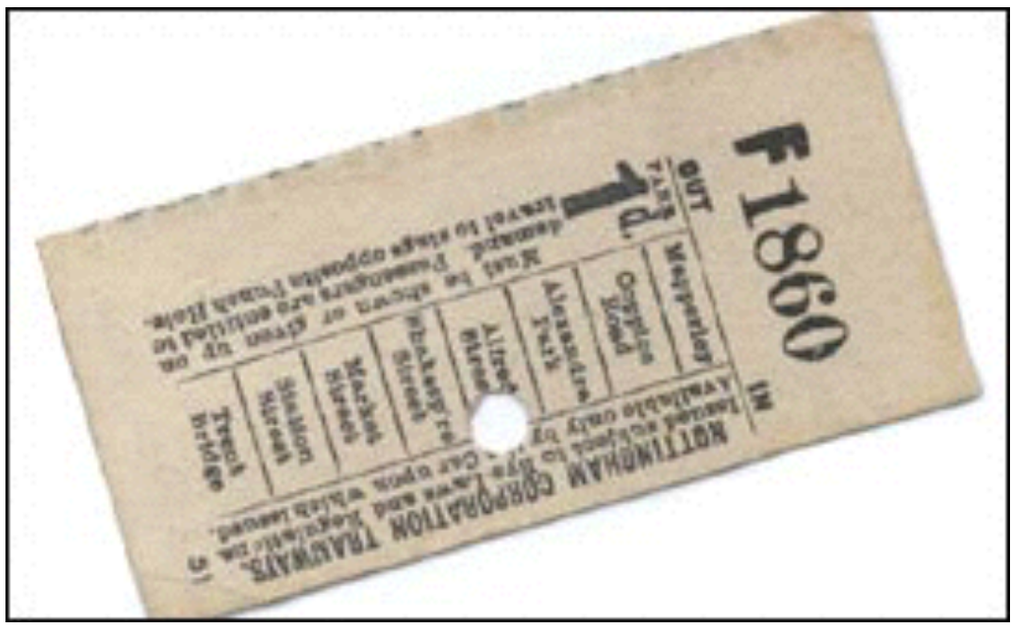
Mobile Devices

- Using app for 10 min every 2 months … and remain logged in!
- Authentication within application
- Safe storage of credentials
- Cut of access when device is lost

Tokens

Modern security standards are all about them…

Tokens / Tickets / Assertions



What is a token?

- (XML / JSON / Whatever) document with some payload
- Information about user
    - Name, organisation, roles, authentication method, timestamp, lifespan, etc.
- Can be signed

Tokens are signed

- Ensure they were not tampered
  - No one altered them in the meantime
- Trust them without additional check with issuer.
  - No communication overhead

Why Tokens?

- Decoupling authentication / authorization
- Not relying on specific method
- Mobile / Server Side and Client Side apps will authenticate differently.
- How this info is passed around

Flexible

You can decide what they contain

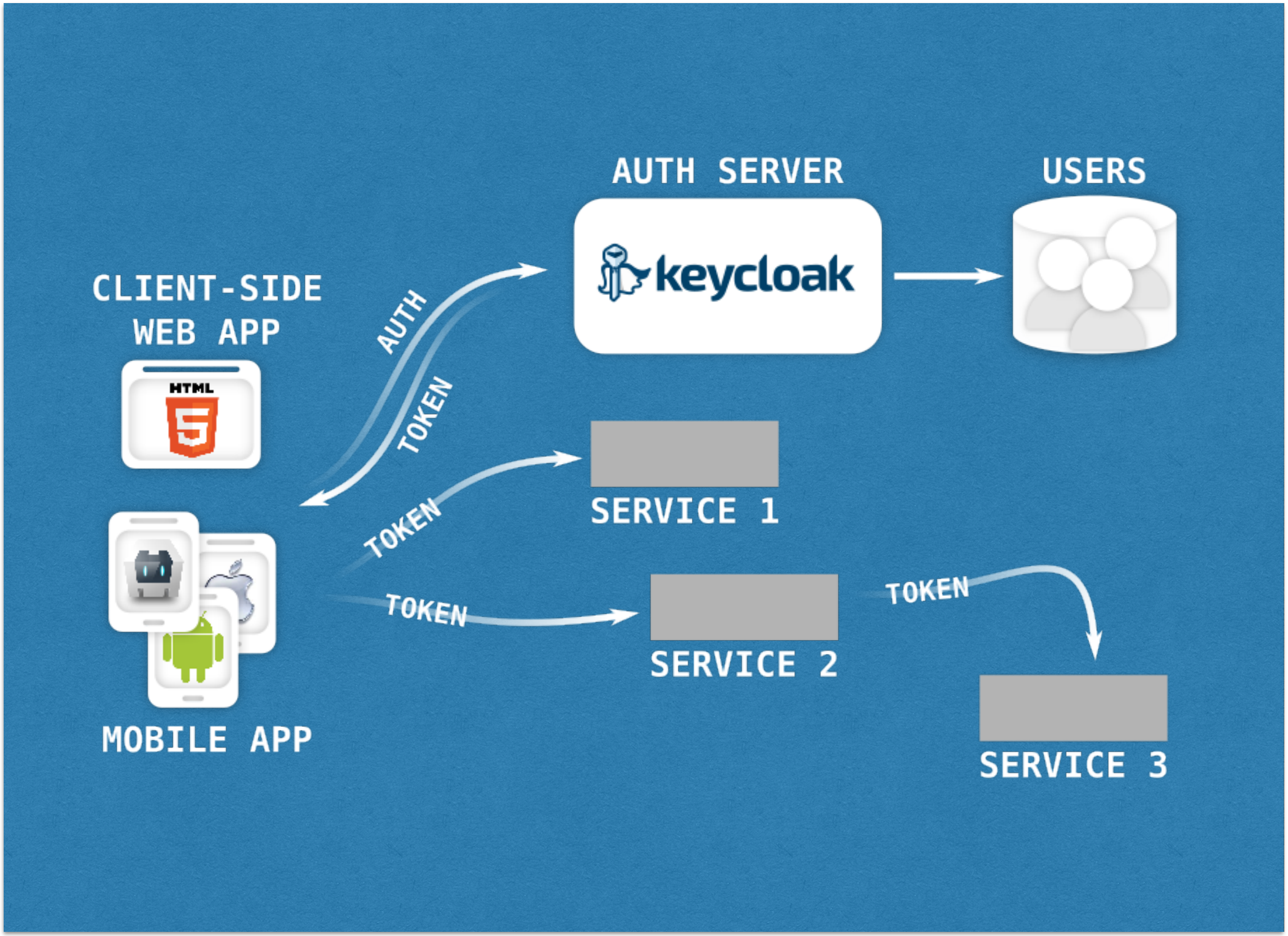
Why Tokens?

- Contain any additional information needed
- Can have defined lifetime
- Can be invalidated or managed in central manner

Ok but… why tokens help?

Delegation of authentication to dedicated and trusted server / provider

Providing credentials to only one selected entity


AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
SERVICE 1
TOKEN
TOKEN
SERVICE 2
TOKEN
SERVICE 3
MOBILE APP

Here authentication happens

AUTH SERVER

keycloak

USERS

CLIENT-SIDE WEB APP

HTML

AUTH

TOKEN

TOKEN

SERVICE 1

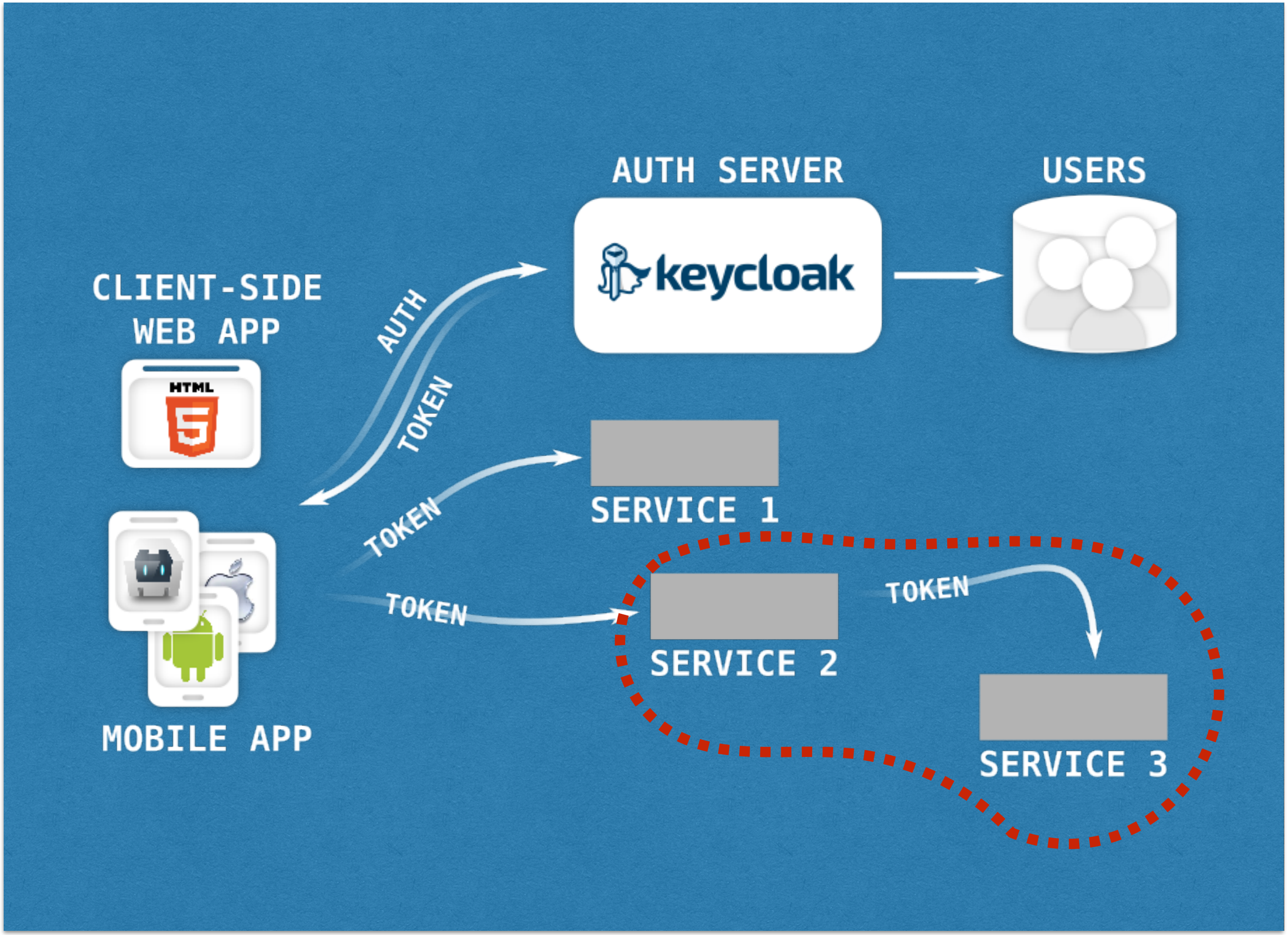
TOKEN

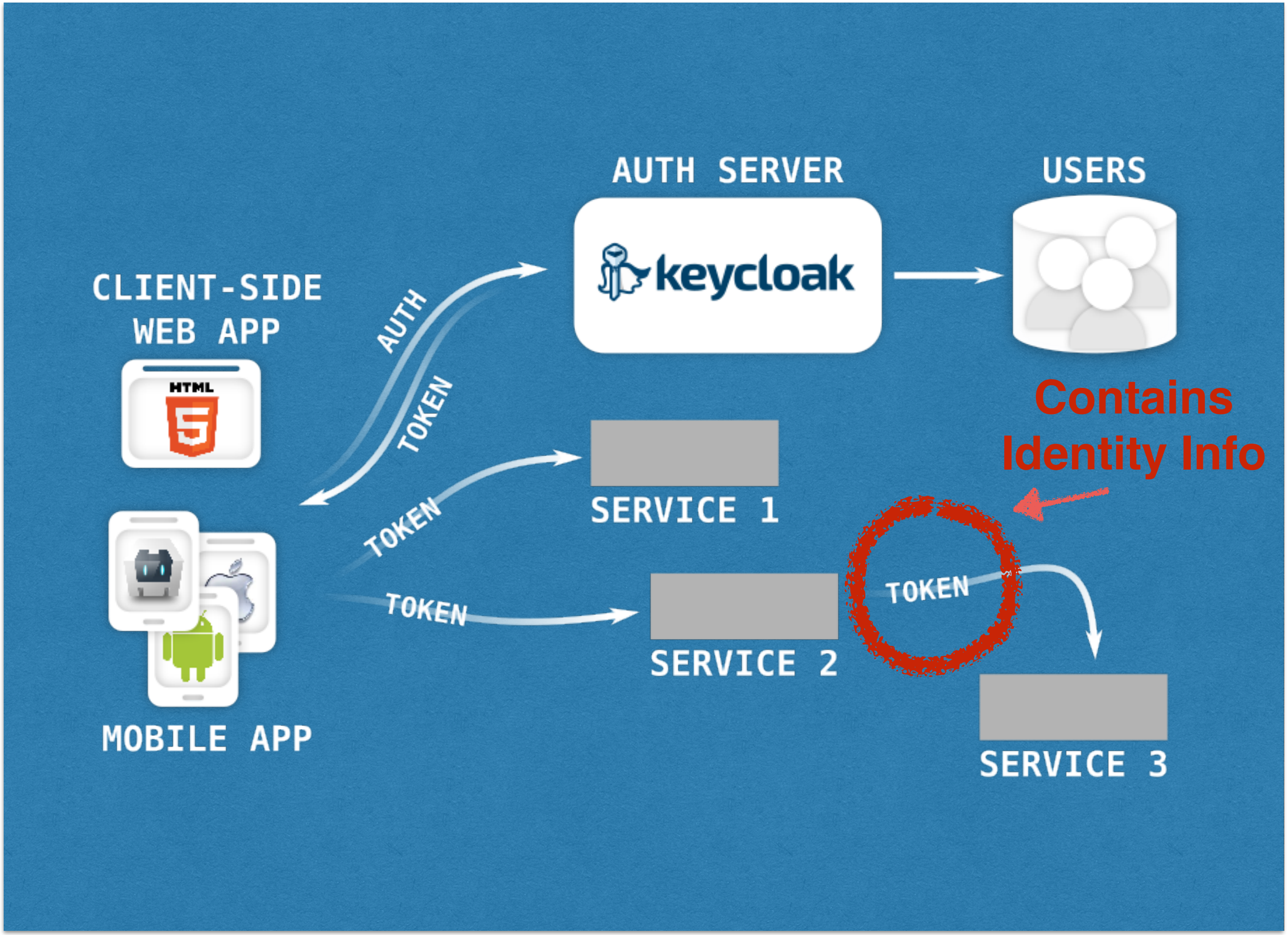
SERVICE 2

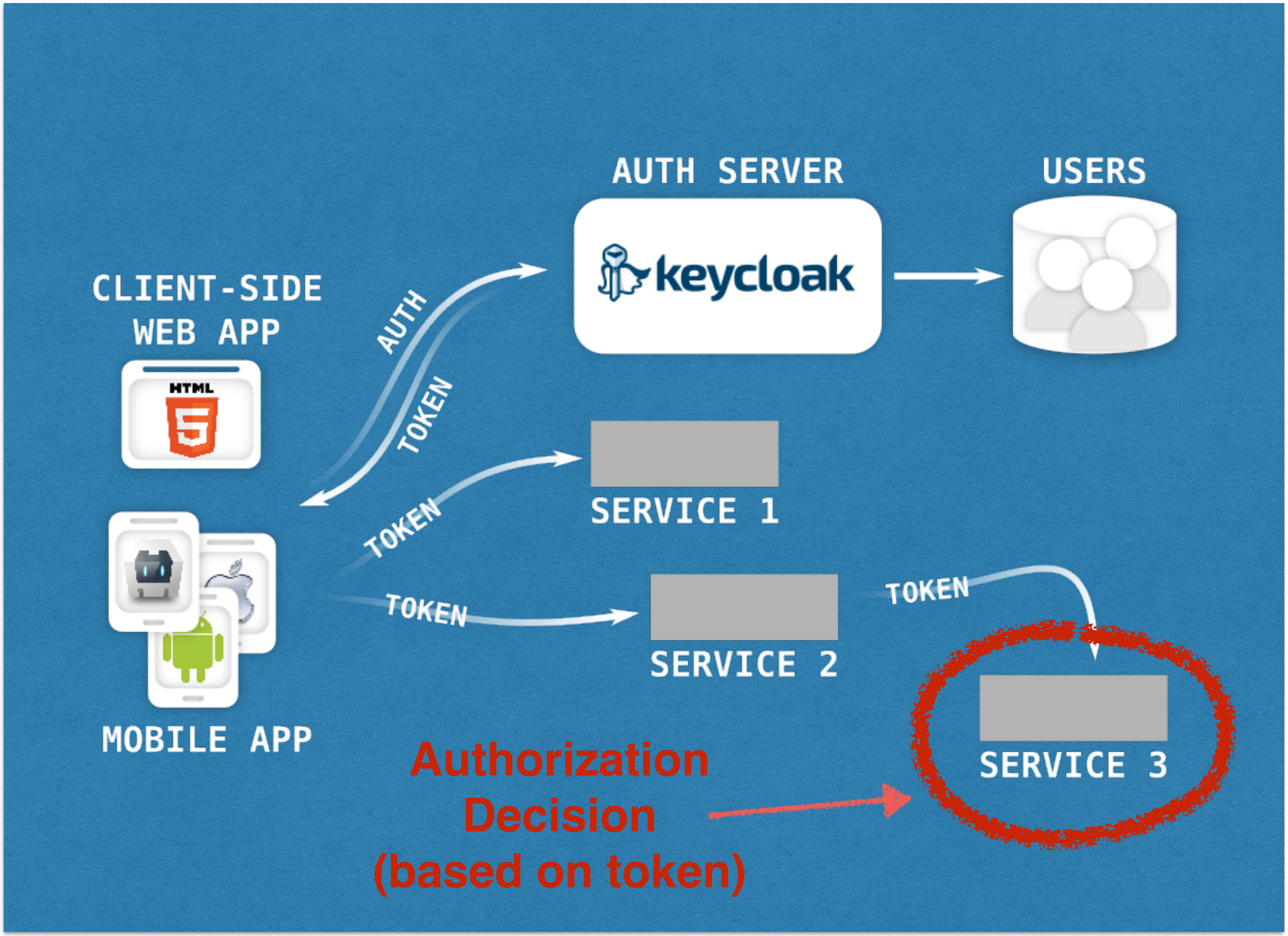
TOKEN

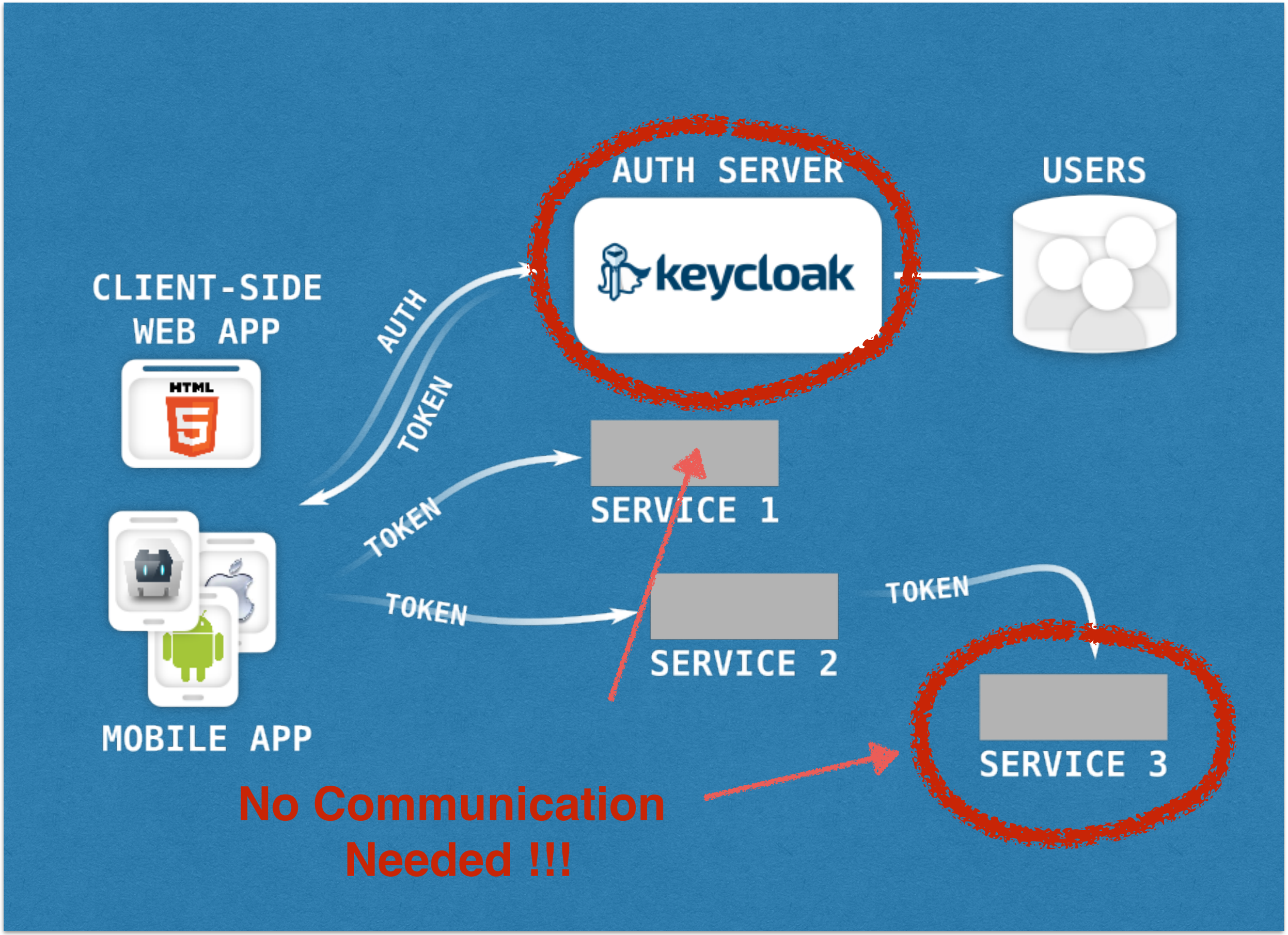
SERVICE 3

MOBILE APP


AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
TOKEN
TOKEN
SERVICE 1
TOKEN
SERVICE 2
SERVICE 3
MOBILE APP


AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
Contains
Identity Info
SERVICE 1
TOKEN
TOKEN
SERVICE 2
TOKEN
SERVICE 3
MOBILE APP


AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
SERVICE 1
TOKEN
TOKEN
SERVICE 2
TOKEN
SERVICE 3
MOBILE APP
Authorization
Decision
(based on token)


AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
SERVICE 1
TOKEN
TOKEN
SERVICE 2
TOKEN
SERVICE 3
MOBILE APP
No Communication
Needed !!!

Typical Components

Identity Provider / Authorization Server

- Stores user data and credentials
- Performing authentication
- Issuing, verifying and revoking tokens

Service Provider / Resource / Resource Server

- Exposing data or operations to be accessed by:
  - user
  - application on behalf of the user

User / Client / Relying Party

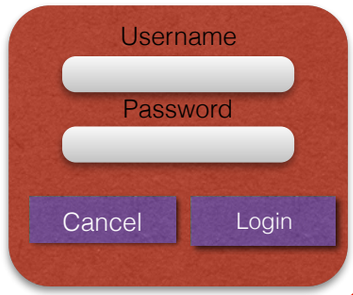
- User (e.g. via Browser)
- Applications acting on behalf of the user
  (e.g. Mobile App)
- Backend Service or App (e.g. Microservices)

In practice?

ACME

ACME

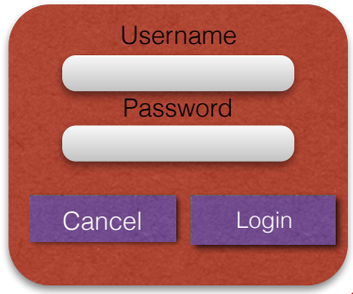
Token



Here you authenticate (providing credential). Within company firewall/VPN

ACME

Token



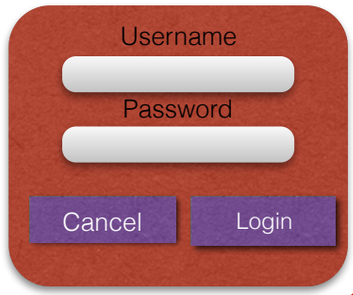
Here you authenticate (providing credentials). Within company firewall/VPN

ACME

Token

Here you get access (no credentials)

Gmail™ by Google



Here you authenticate (providing credential). Within company firewall/VPN

ACME

Token

Here you get access (no credentials)

Gmail™ by Google

ACME remains in control of shared user information

Web SSO
(Single Sign On)

Federation

SAML 2

SAML 2

- OASIS Standard
- Version 1.0 in 2001
- Version 2.0 in 2005

SAML 2 Facts

- XML token - verbose and heavy
- Complex flows, lot of different profiles
- Very mature and widely adopted
- Many binding types - SOAP for e.g.
- High learning curve
- Doesn’t fit mobile use cases well.

SAML 2 - Federation

- Authentication and authorization between organisations
- Not sharing access or ownership to user data
- Trusting external Identity Provider

SAML2 is very widely adopted

de facto standard

What is the issue with SAML2 and mobile?

SAML2 was not designed with mobile and modern web in mind…

POST vs GET

There are ways to workaround it…

… still it always remains a workaround…

Newcomers…

OAuth2 & OpenID Connect

Modern security needs

- Client Side Apps (HTML 5 / JS / Stateless)
- Mobile Apps
- (Micro)Services / APIs

First of all...

Tokens! :)

Worth understanding how they really look like…

Token standards

- Kerberos
- SAML 2
- JWT (JSON Web Token)

JSON Web Token

JOSE

- **JSON Object Signing and Encryption**
- Set of **IETF** open standards (RFCs)
- **JWT** - JSON Web Token
- **JWS** - JSON Web Signature
- **JWE** - JSON Web Encryption
- **JWA** - JSON Web Algorithms
- **JWK** - JSON Web Key

JWT (JSON Web Token)

- JSON Document
- Key [String] : Value [JSON]
- Wide choice of signing algorithms

Anatomy

- Simple JSON, Encoded in Base64
- Three parts
  - Header
  - Payload
  - Signature

JSON Web Token

eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ
9.eyJpc3MiOiJqb2U
iLCJleHAiOjEzMDA
4MTkzODAsImh1b
WFuIjp0cnVlLCJvd
mVyIDE4Ijp0cnVlfQ.
0LtIxMRXW0VItgVq
hQGUJCEjzWpnXpt
QOWxn9wGDyW4

Header

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

Header in Base64:
eyJhbGciOiJIUzI1NiI
sInR5cCI6IkpXVCJ9

Payload

```
{
  "iss": "joe",
  "exp": 1300819380,
  "human": true,
  "over 18": true
}
```

Payload in Base64:
eyJpc3MiOiJqb2UiLCJleHAiOjE
zMDA4MTkzODAsImh1bWFuIjp
0cnVlLCJvdmVyIDE4Ijp0cnVlfQ

JSON Web Token

eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ
9.eyJpc3MiOiJqb2U
iLCJleHAiOjEzMDA
4MTkzODAsImh1b
WFuIjp0cnVlLCJvd
mVyIDE4Ijp0cnVlfQ.
0LtIxMRXW0VItgVq
hQGUJCEjzWpnXpt
QOWxn9wGDyW4

Header

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

Header in Base64:
eyJhbGciOiJIUzI1NiI
sInR5cCI6IkpXVCJ9

Payload

```
{
  "iss": "joe",
  "exp": 1300819380,
  "human": true,
  "over 18": true
}
```

Payload in Base64:
eyJpc3MiOiJqb2UiLCJleHAiOjE
zMDA4MTkzODAsImh1bWFuIjp
0cnVlLCJvdmVyIDE4Ijp0cnVlfQ

JSON Web Token

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJqb2UiLCJleHAiOjE2MDA4MTkzODAsImh1bWFuIjp0cnVlLCJvdmVyIDE4Ijp0cnVlfQ.0LtIxMRXW0VItgVqhQGUJCEjzWpnXptQOWxn9wGDyW4

Header

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

Header in Base64:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9

Payload

```
{
  "iss": "joe",
  "exp": 1300819380,
  "human": true,
  "over 18": true
}
```

Payload in Base64:
eyJpc3MiOiJqb2UiLCJleHAiOjE2MDA4MTkzODAsImh1bWFuIjp0cnVlLCJvdmVyIDE4Ijp0cnVlfQ

JSON Web Token

eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ
9.eyJpc3MiOiJqb2U
iLCJleHAiOjEzMDA
4MTkzODAsImh1b
WFuIjp0cnVlLCJvd
mVyIDE4Ijp0cnVlfQ.
0LtIxMRXW0VItgVq
hQGUJCEjzWpnXpt
QOWxn9wGDyW4

Header

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

Header in Base64:
eyJhbGciOiJIUzI1NiI
sInR5cCI6IkpXVCJ9

Payload

```
{
  "iss": "joe",
  "exp": 1300819380,
  "human": true,
  "over 18": true
}
```

Payload in Base64:
eyJpc3MiOiJqb2UiLCJleHAiOjE
zMDA4MTkzODAsImh1bWFuIjp
0cnVlLCJvdmVyIDE4Ijp0cnVlfQ

JSON Web Token

eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ
9.**eyJpc3MiOiJqb2**
UiLCJleHAiOjEzMD
A4MTkzODAsImh1
bWFuIjp0cnVlLCJv
dmVyIDE4Ijp0cnVlf
Q.
0LtIxMRXWv8vItgVq
hQGUJCEjzWpnXpt
QOWxn9wGDyW4

Header

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

Header in Base64:
eyJhbGciOiJIUzI1NiI
sInR5cCI6IkpXVCJ9

Payload

```
{
  "iss": "joe",
  "exp": 1300819380,
  "human": true,
  "over 18": true
}
```

Payload in Base64:
eyJpc3MiOiJqb2UiLCJleHAiOj
EzMDA4MTkzODAsImh1bWFul
jp0cnVlLCJvdmVyIDE4Ijp0cnV
lfQ

JSON Web Token

eyJhbGciOiJIUzI1Ni
IsInR5cCI6IkpXVCJ
9.eyJpc3MiOiJqb2U
iLCJleHAiOjEzMDA
4MTkzODAsImh1b
WFuIjp0cnVlLCJvd
mVyIDE4Ijp0cnVlfQ.
0LtIxMRXW0VltgVq
hQGUJCEjzWpnXp
tQOWxn9wGDyW4

Hash

Header

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

Header in Base64:
eyJhbGciOiJIUzI1NiI
sInR5cCI6IkpXVCJ9

Payload

```
{
  "iss": "joe",
  "exp": 1300819380,
  "human": true,
  "over 18": true
}
```

Payload in Base64:
eyJpc3MiOiJqb2UiLCJleHAiOjE
zMDA4MTkzODAsImh1bWFuIjp
0cnVlLCJvdmVyIDE4Ijp0cnVlfQ

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

Simple!

All you need to know to parse it in your app! ;)

OAuth2

Who knows or used OAuth2?

Everyone ;)

Email *

Password *

Sign In

Forgot your password?

Facebook

Google

Don’t have an account? Create Account

▾ Pocket would like to:

Have offline access

Cancel

Accept

source: aaronparecki.com

OAuth2 - Some facts

- Many major companies behind it
  - Google, Facebook, Twitter and etc.
- 2.0 incompatible with 1.x
- Becoming de facto standard security framework for
  - Mobile Apps
  - Client Side
  - APIs / Services

Major usecase?

NOT for authentication

. . .

NOT for authorization

. . .

DELEGATION

protocol

Applications & Services

acting on behalf of the user

OAuth2 Actors

- **Resource owner** - YOU! :)
- **Client** - application acting on your behalf
- **Resource / Resource Server** - APIs accessed by Client
- **Authorization Server** - Place where you authenticate and where tokens are being issued

Authorization flows

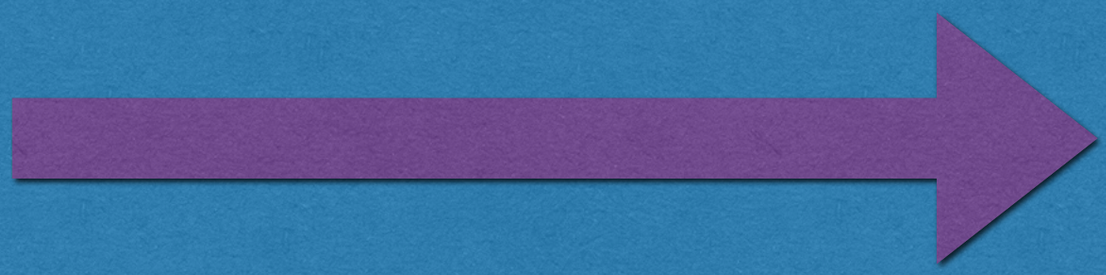
- **Authorization Code** - Server Side Application
- **Implicit** - Client Side and Mobile Apps
- **Resource Owner** - Directly using username / password
- **Client Credentials** - For app related operations - app identity

Authorization Code Flow

Server Side Applications

Authorization Server

Resource Server



Resource Owner

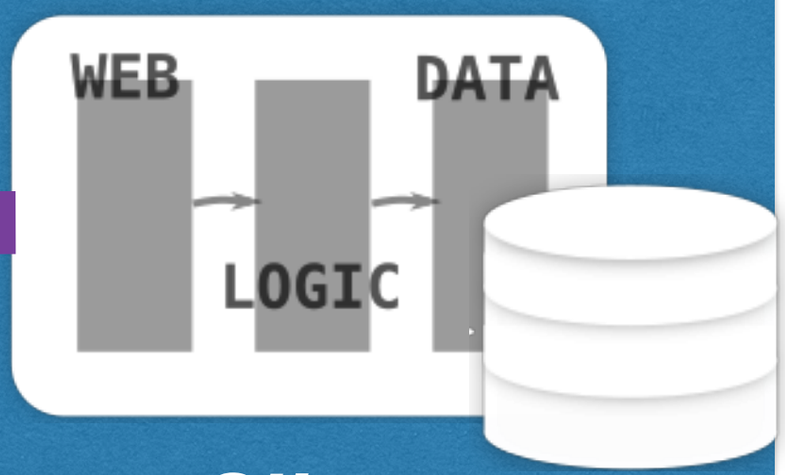
Client

Authorization Server

Resource Server

Picasa

Client ID



Resource Owner

Client

Authorization Server

Resource Server

Authorization Code

Resource Owner

Client

Authorization Server

Resource Server

Picasa

Authorization Code

Client ID

Client Secret

Resource Owner

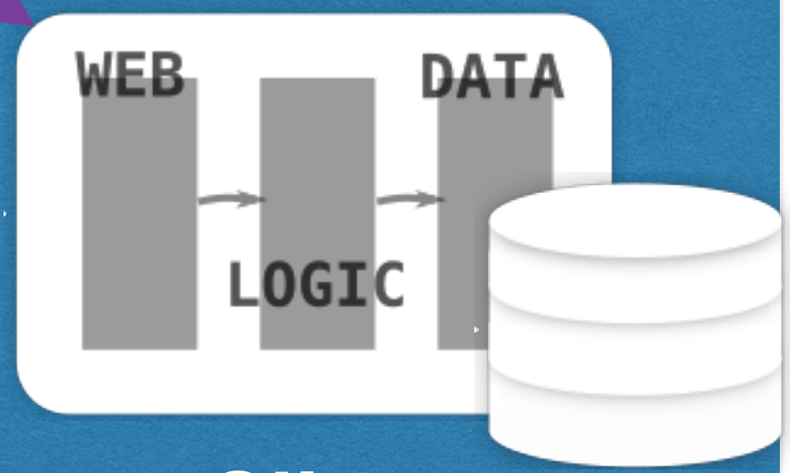
Client

Authorization Server

Resource Server

Picasa

Access Token
Refresh Token



Resource Owner

Client

Authorization Server

Resource Server

Access Token

Resource Owner

Client

Implicit

Client Side & Mobile

Authorization Server

Resource Server

Client

Authorization Server

Resource Server

Access Token

Client

Authorization Server

Resource Server

Access Token

Client

OAuth2 - Scopes

- During authorization user grants application access with given scope
  - e.g. Basic profile info only (full name & email)
  - e.g. Information about all your social graph (friends on FB)
  - e.g. Write access to FB wall

▾ Pocket would like to:

Have offline access

Cancel

Accept

OAuth2 - Summary

- Is pretty generic
  - Doesn't define specific token standard
  - Lacks on authentication and user info side
- Many consider it more "*a protocol framework*" then protocol itself.

Scales and adapts nicely

Just teach your (micro)Service to understand JWT!

AUTH SERVER

USERS

Username

Password

Cancel

Login

CLIENT-SIDE WEB APP

HTML

AUTH

TOKEN

SERVICE 1

TOKEN

TOKEN

SERVICE 2

TOKEN

SERVICE 3

MOBILE APP

OpenID Connect

OpenID Connect

- Builds on top of OAuth2
- Adds two new flows
- Adds session management -
    - e.g. Single Sign On & Out for HTML5 apps
- ID token
- User info endpoint
- Discovery & Clients self registration

Getting widely adopted…

and being pushed by Google, MSFT and etc.

OpenID Connect is what people should consider for new development

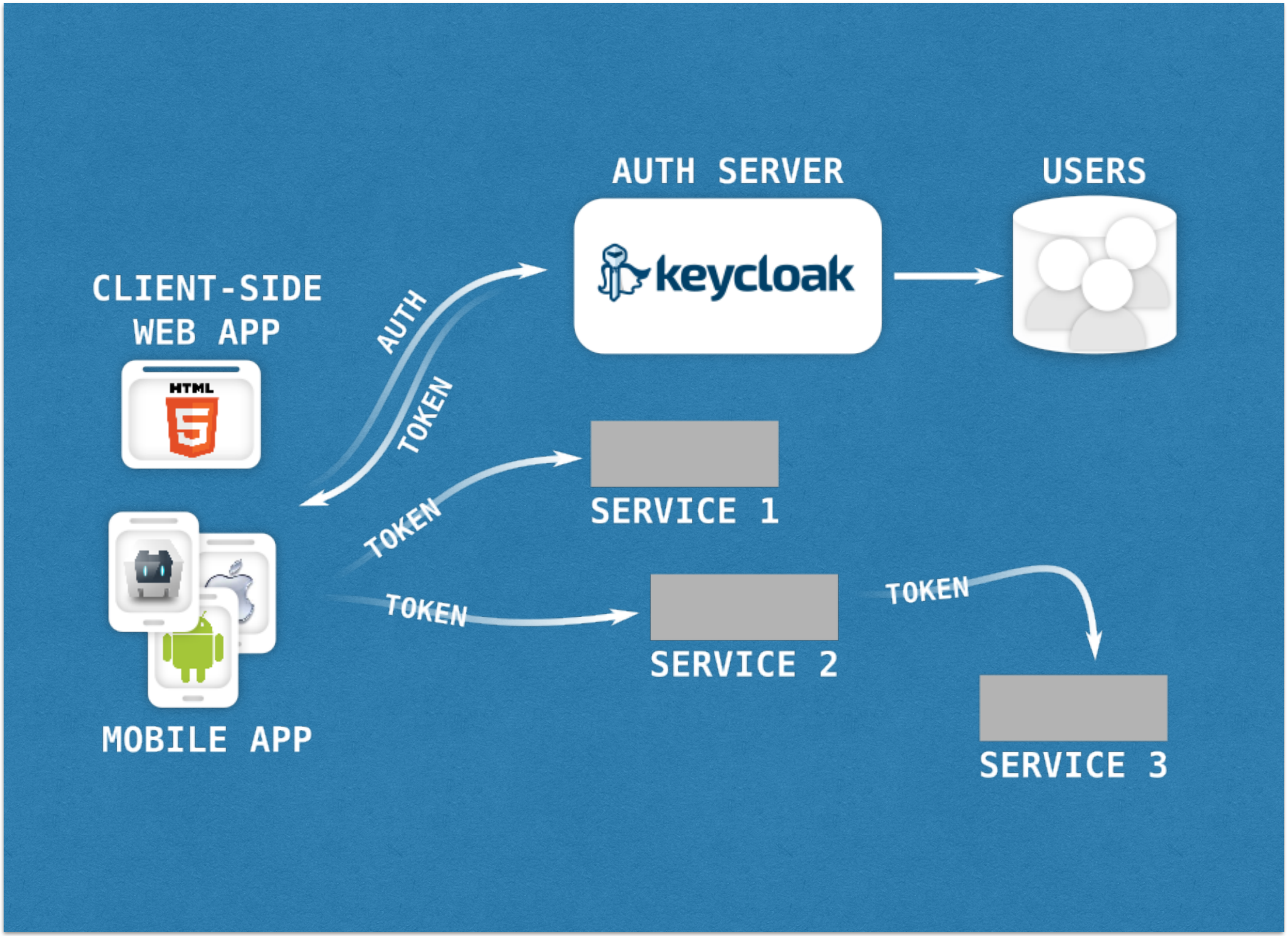
For any new modern application

OpenID Connect is what Keycloak is using

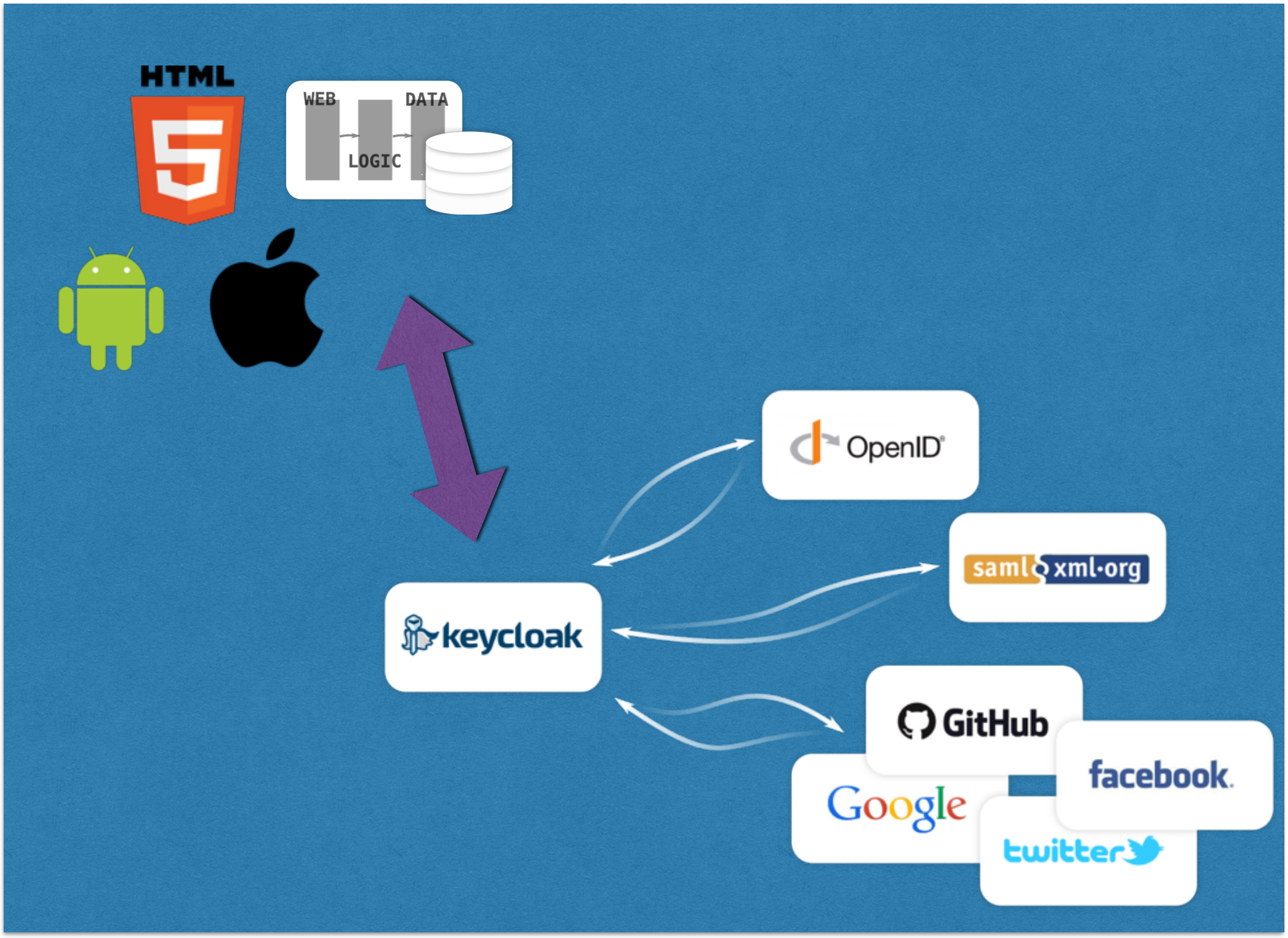
Communication protocol between Keycloak Server and Keycloak client adapters installed in each secured application

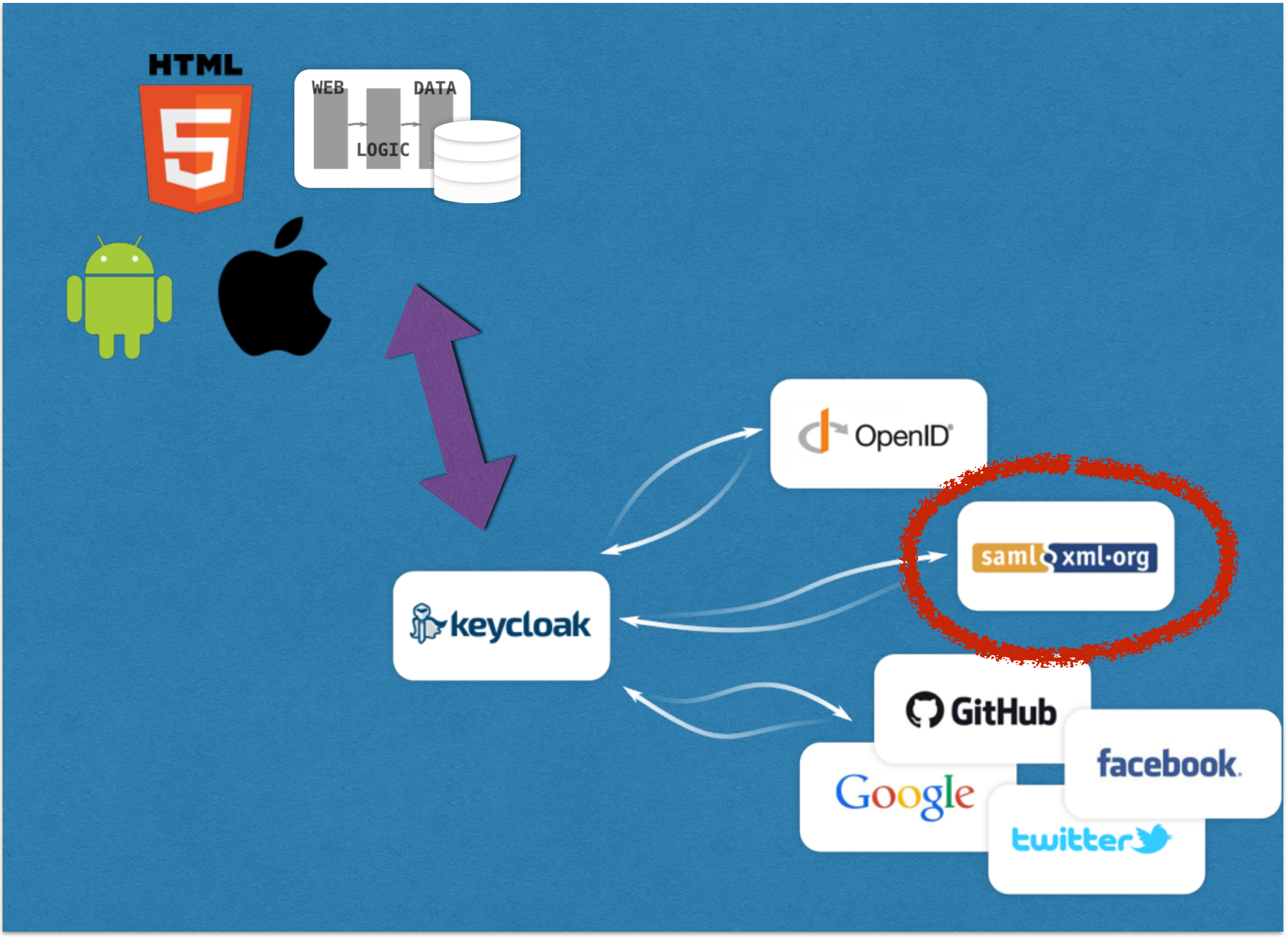
Still there is a lot of existing infrastructure in place…

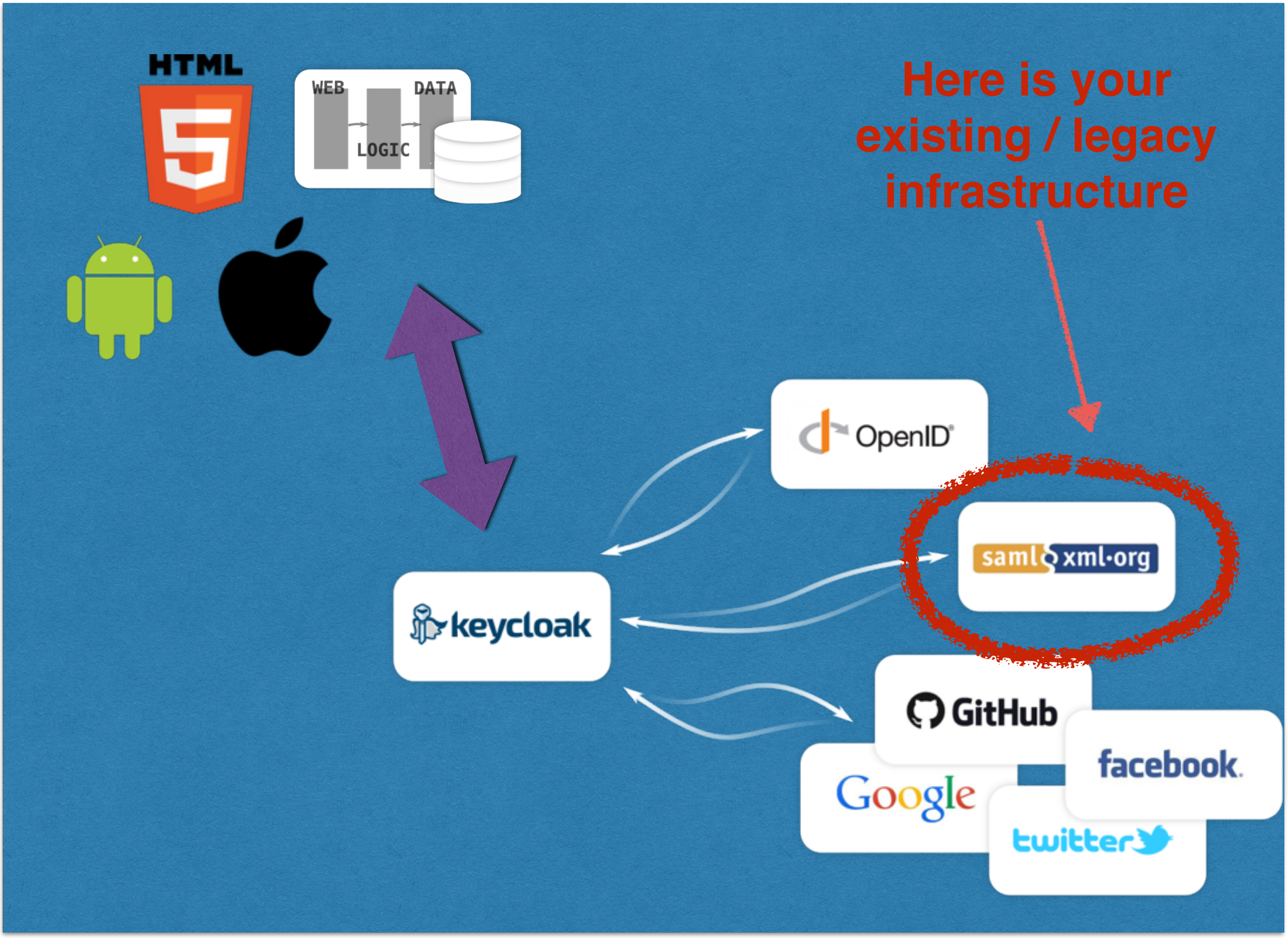
Keycloak aim is to provide the easiest way to integrate with it.

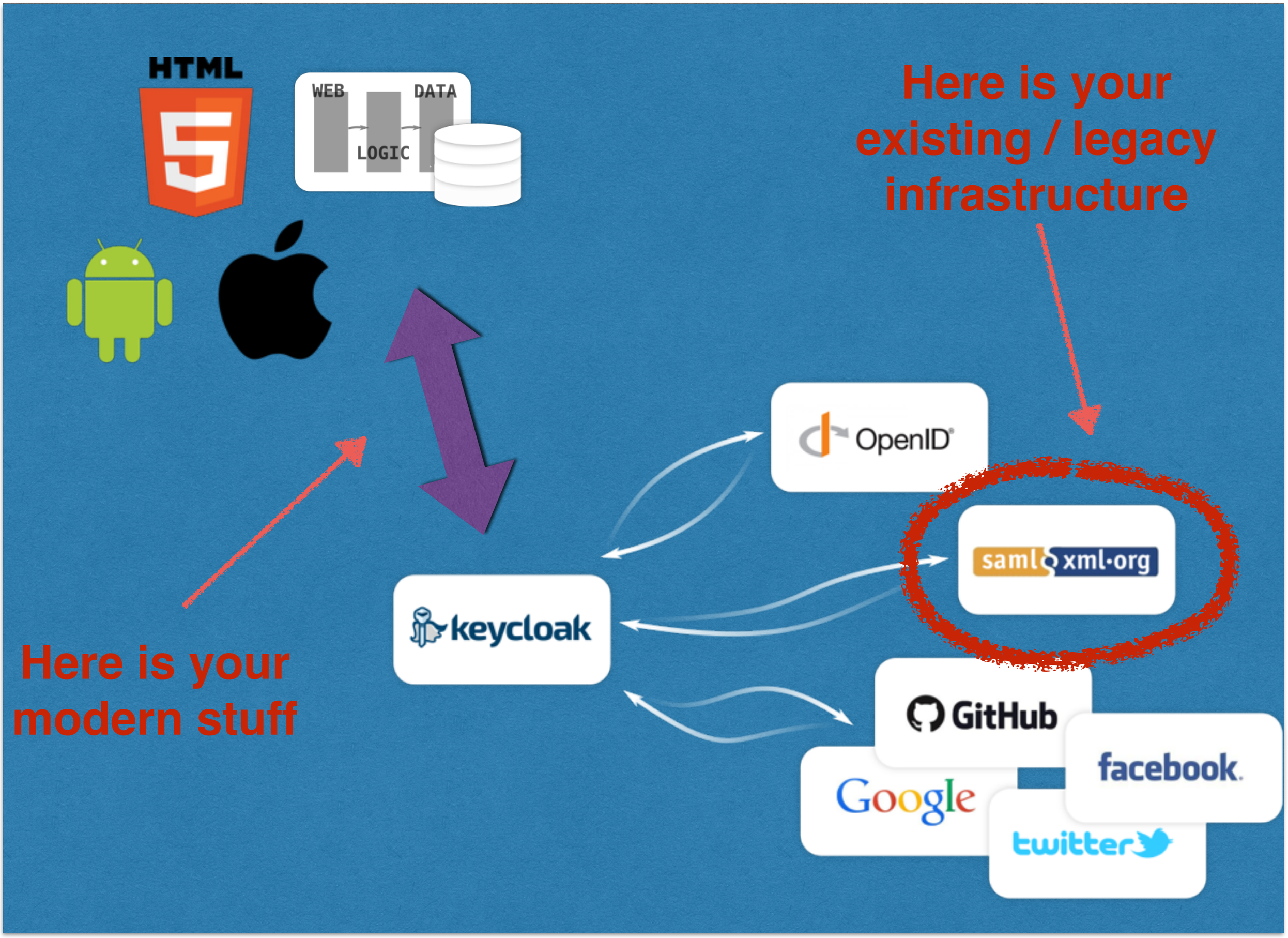

AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
SERVICE 1
TOKEN
TOKEN
SERVICE 2
TOKEN
SERVICE 3
MOBILE APP

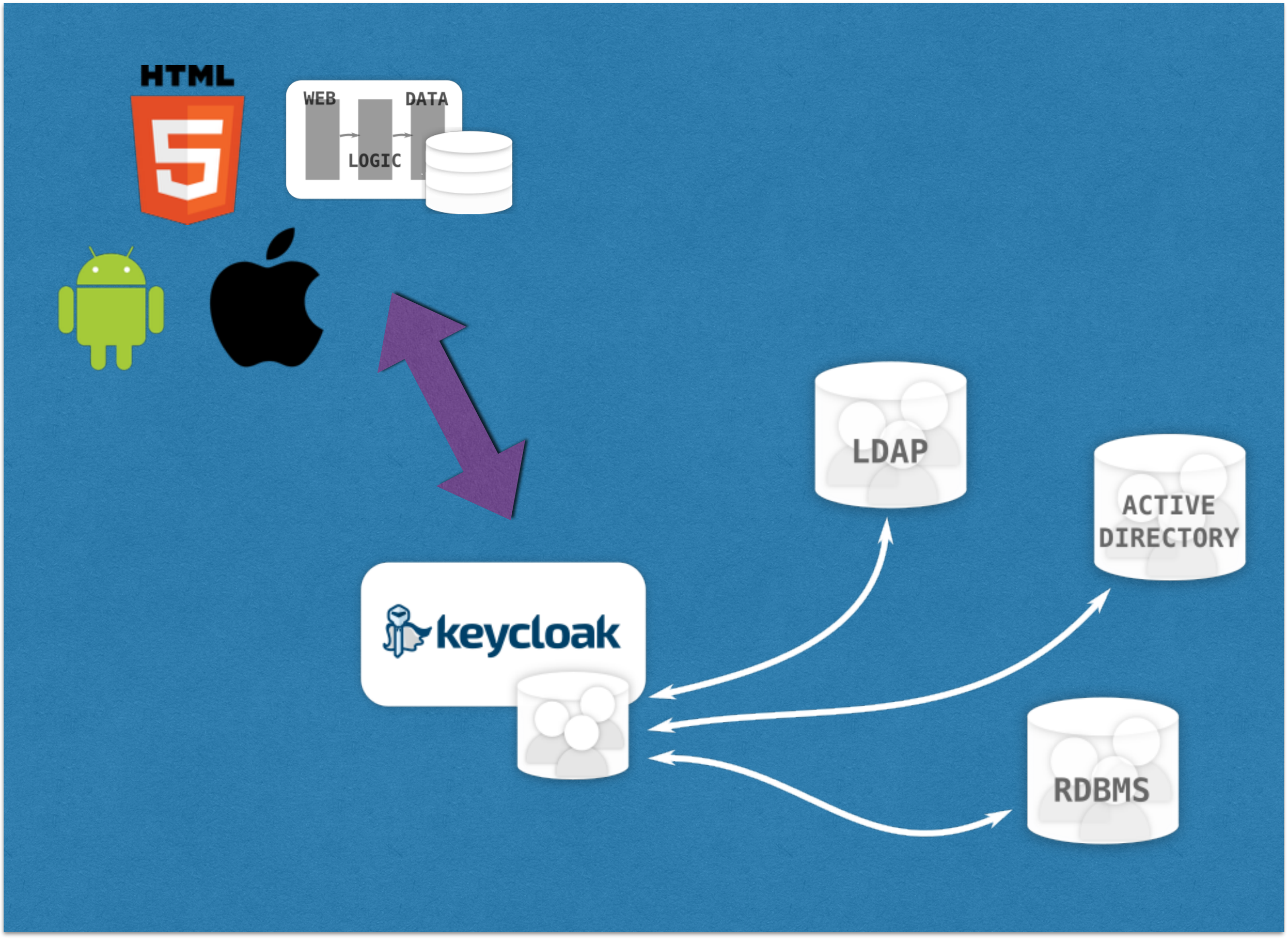
Identity Brokering


HTML
WEB
DATA
LOGIC
OpenID
saml xml·org
keycloak
GitHub
Google
facebook
twitter


HTML
WEB
DATA
LOGIC
OpenID
saml xml·org
keycloak
GitHub
Google
facebook
twitter


HTML
WEB
DATA
LOGIC
Here is your existing / legacy infrastructure
OpenID
saml xml·org
keycloak
GitHub
Google
facebook
twitter


HTML
WEB
DATA
LOGIC
Here is your existing / legacy infrastructure
Here is your modern stuff
OpenID
saml xml·org
keycloak
GitHub
facebook
Google
twitter


HTML
WEB
DATA
LOGIC
LDAP
ACTIVE
DIRECTORY
keycloak
RDBMS

HTML

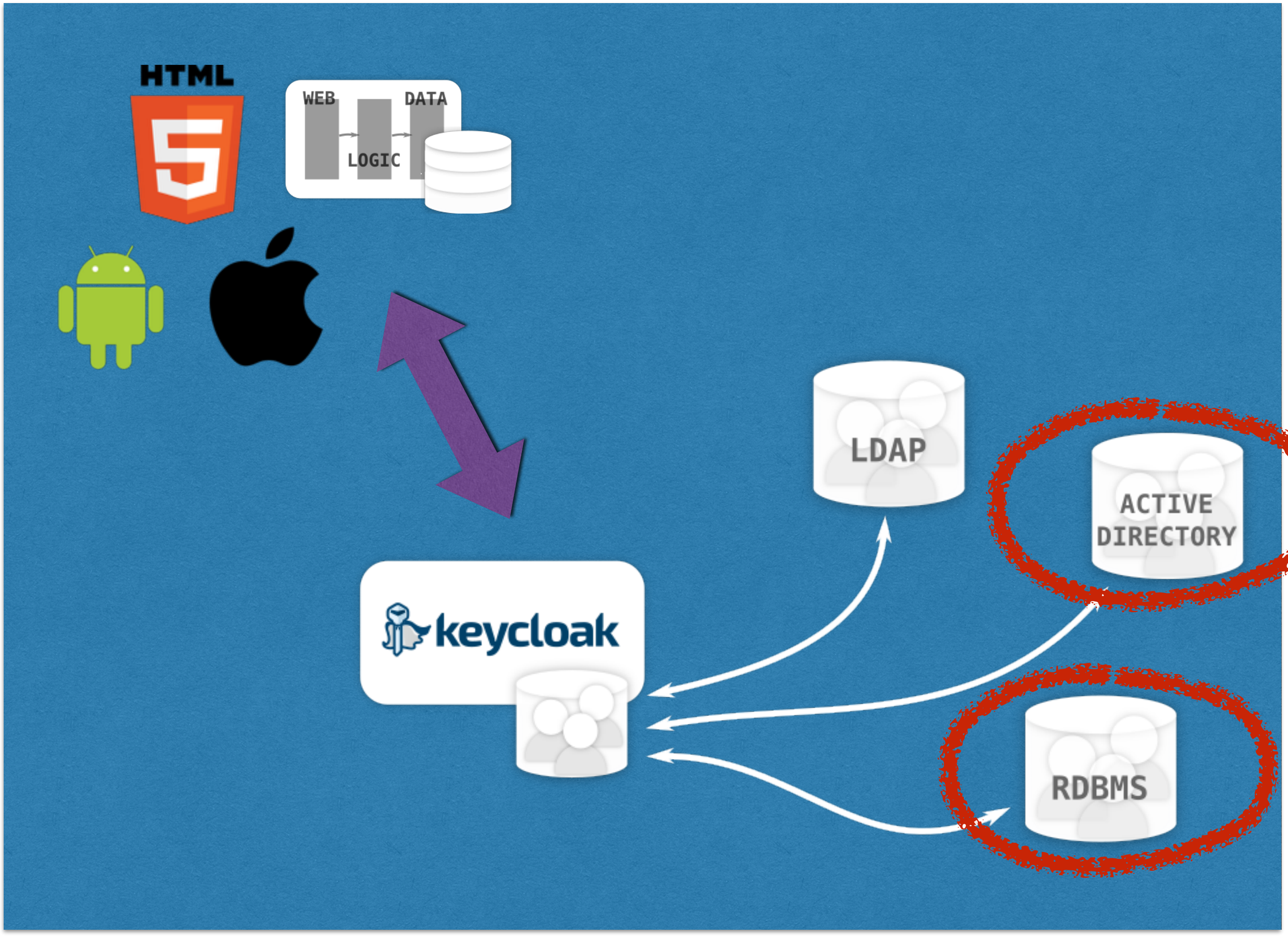
WEB DATA LOGIC

LDAP

ACTIVE DIRECTORY

keycloak

RDBMS


HTML
WEB
DATA
LOGIC
LDAP
ACTIVE
DIRECTORY
keycloak
RDBMS

Will handle existing / legacy infra integration for you !!!

AUTH SERVER

keycloak

USERS

CLIENT-SIDE WEB APP

HTML

AUTH

TOKEN

SERVICE 1

TOKEN

TOKEN

SERVICE 2

TOKEN

SERVICE 3

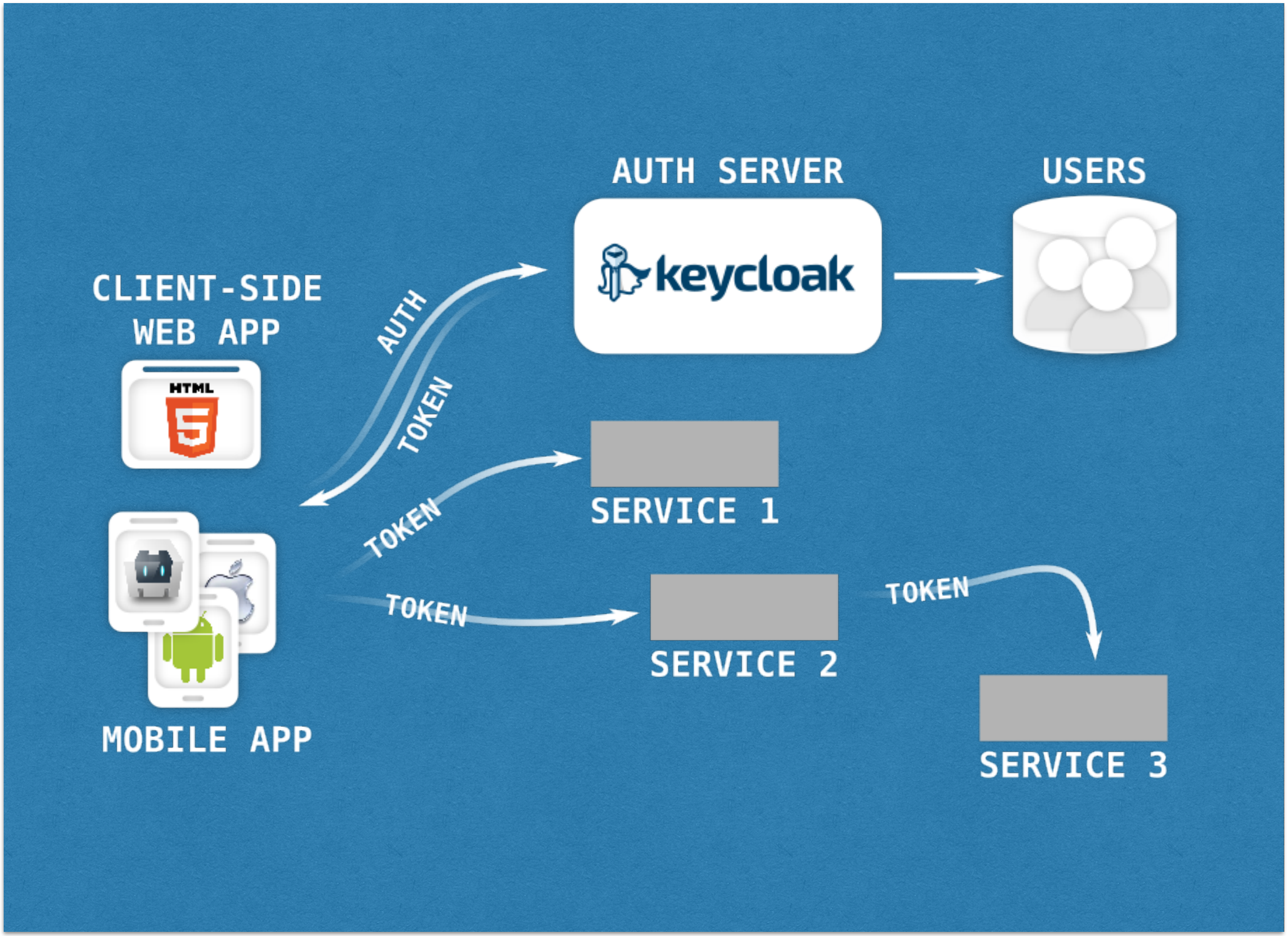
MOBILE APP

Externalise Security

Integrate with OOTB Identity Solution

Security is hard

Don’t reinvent (or reimplement) things yourself…


AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
SERVICE 1
TOKEN
TOKEN
SERVICE 2
TOKEN
SERVICE 3
MOBILE APP

It should be easy for developers!

Major mission of Keycloak

How does it work?

<button onclick="**keycloak.login()**">
Login
</button>

Most basic HTML5 Hello World example…

WELCOME

LOGIN

OOTB Client Adapters

- JBoss EAP & WildFly
- JBoss Fuse
- JBoss BxMS
- Spring
- Tomcat / Jetty
- … custom …

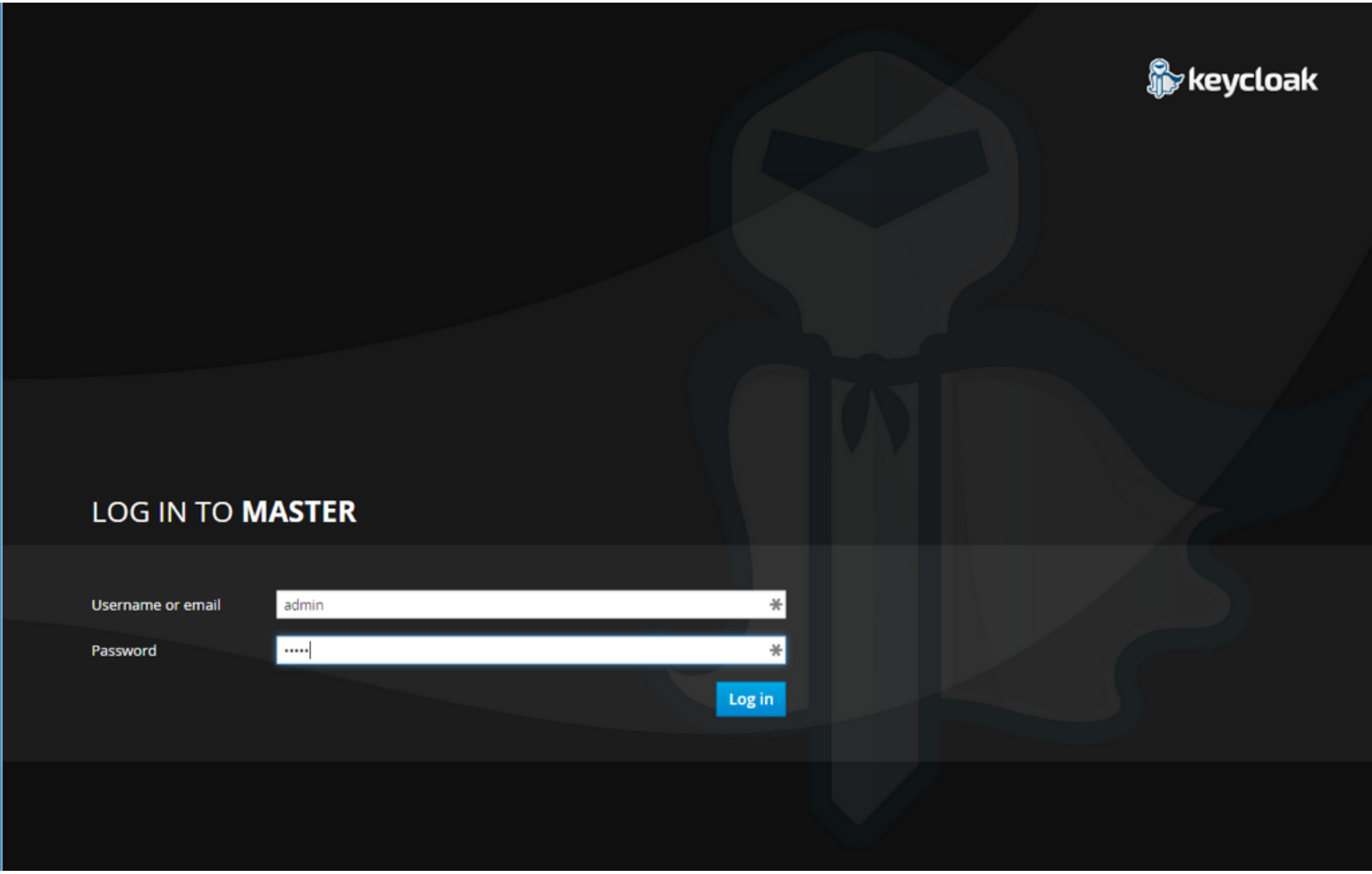
- **JavaScript**
- **NodeJS**
- Mobile (Cordova / Native)
- Any OIDC compat library
- **mod-auth-mellon** on Apache
- Reverse Web Proxy

What do you get?

When you click on this “Login” button…

WELCOME

LOGIN


keycloak
LOG IN TO MASTER
Username or email
admin
Password
Log in

keycloak

Admin

Master

Configure

- Realm Settings
- Clients
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

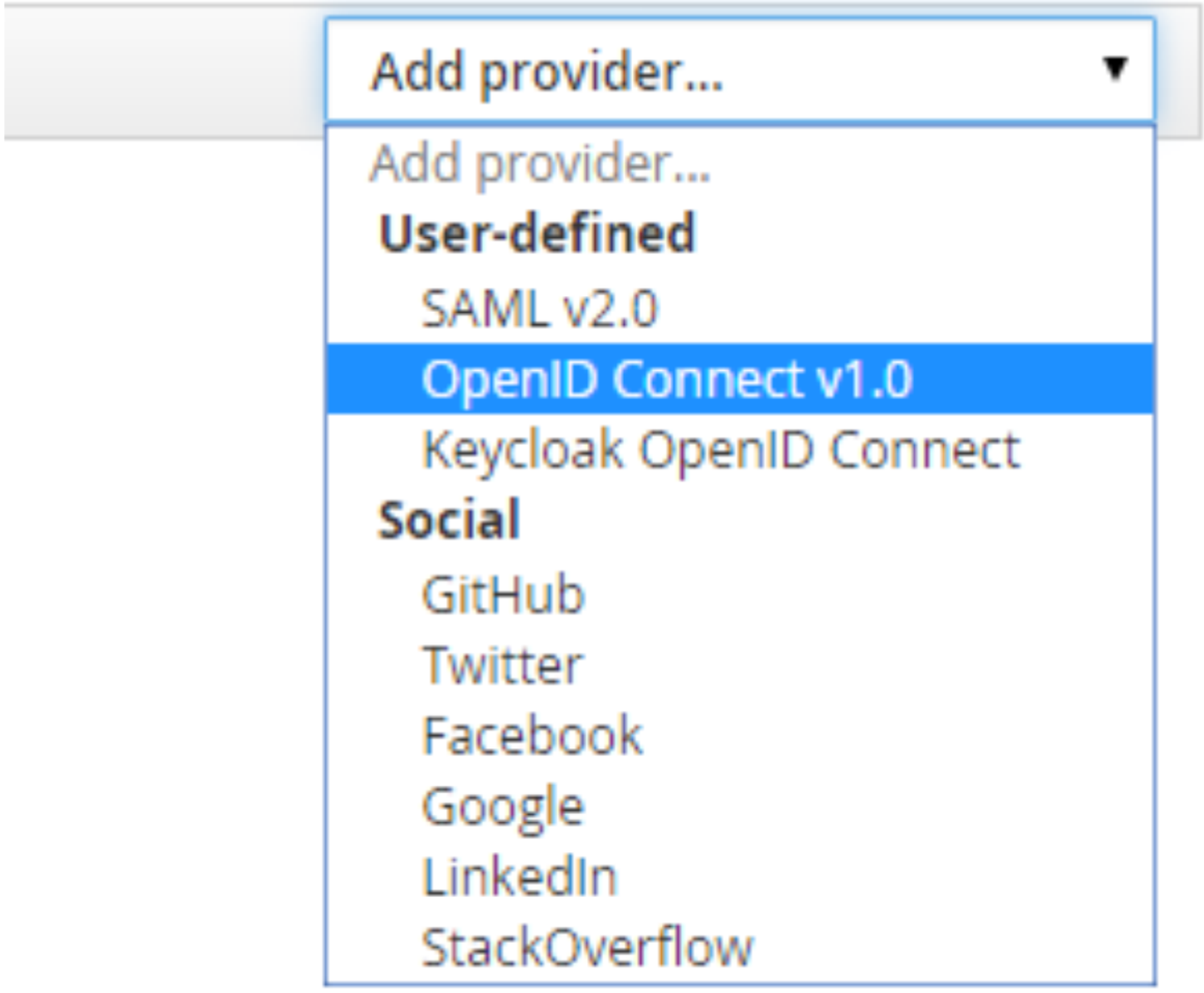
- Users
- Sessions
- Events

Master

General | Login | Keys | Email | Themes | Cache | Tokens | Security Defenses

User registration	ON
Email as username	ON
Edit username	ON
Forget password	ON
Remember Me	ON
Verify email	ON
Require SSL	external request

Save Cancel


Add provider...
Add provider...
User-defined
SAML v2.0
OpenID Connect v1.0
Keycloak OpenID Connect
Social
GitHub
Twitter
Facebook
Google
LinkedIn
StackOverflow

keycloak

LOG IN TO **MASTER**

Email

Password

Remember me

Forgot Password?

Log in

New user? Register

saml

ACME

JDD.ORG.PL

github

twitter

facebook

google+

stackoverflow

linkedin

keycloak

LOG IN TO **MASTER**

Email

Password

Remember me

Forgot Password?

Log in

New user? Register

saml

ACME

JDD.ORG.PL

github

twitter

facebook

google+

stackoverflow

linkedin

keycloak

LOG IN TO **MASTER**

Email

Password

Remember me

Forgot Password?

Log in

New user? Register

saml

ACME

JDD.ORG.PL

github

twitter

facebook

google+

stackoverflow

linkedin

keycloak

LOG IN TO **MASTER**

Email

Password

Remember me

Forgot Password?

Log in

New user? Register

saml

ACME

JDD.ORG.PL

github

twitter

facebook

google+

stackoverflow

linkedin

keycloak

LOG IN TO **MASTER**

Email

Password

Remember me

Forgot Password?

Log in

New user? Register

saml

ACME

JDD.ORG.PL

github

twitter

facebook

google+

stackoverflow

linkedin

All of this within few minutes (and clicks)

Want to make it look like part of your application?

Sure!

Welcome back to Red Hat Developers

Choose how you would like to log in to your account:

LOG IN WITH YOUR EXISTING ACCOUNT

LOGIN WITH YOUR GITHUB ACCOUNT

LOGIN WITH YOUR STACKOVERFLOW ACCOUNT

LOGIN WITH YOUR LINKEDIN ACCOUNT

Don't see the account you prefer?

See more options ...

Don't have an account? Create one today!

My users need to accept “Terms & Conditions” page during auth…

keycloak

Admin

Master

Configure

- Realm Settings
- Clients
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Users
- Sessions
- Events

Authentication

Flows | Bindings | Required Actions | Password Policy | OTP Policy

Required Action	Enabled	Default Action
Configure T…	☑	☐
…date Password	☑	☐
Terms and Conditions	☑	☐
…date Profile	☑	☐
Verify Email	☑	☐

keycloak

Admin

Master

Configure

- Realm Settings
- Clients
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Users
- Sessions
- Events

Authentication

Flows | Bindings | Required Actions | Password Policy | OTP Policy

Browser

New | Copy

Auth Type		Requirement		
Cookie		ALTERNATIVE	DISABLED	
Kerberos		ALTERNATIVE	REQUIRED	DISABLED
Forms		ALTERNATIVE	REQUIRED	DISABLED
	Username Password Form	REQUIRED		
	O T P Form	REQUIRED	OPTIONAL	DISABLED

User Self Management?

keycloak

Back to security-admin-console Sign Out

- Account
- Password
- Authenticator
- Sessions
- Applications

Edit Account

* Required fields

Username admin

Email *

First name *

Last name *

Cancel Save

Let them setup OTP?

keycloak

Back to security-admin-console Sign Out

Account

Password

Authenticator

Sessions

Applications

Authenticator

1. Install FreeOTP or Google Authenticator on your device. Both applications are available in Google Play and Apple App Store.
2. Open the application and scan the barcode or enter the key.

IVUU YYTC KFRE S5CI M5KW 2ZKW PAYW 4VJT

3. Enter the one-time code provided by the application and click Save to finish the setup.

One-time code

Cancel Save

Let them manage all their active sessions in one place?

Sessions

IP	Started	Last Access	Expires	Clients
127.0.0.1	Nov 6, 2015 12:04:48 PM	Nov 6, 2015 12:05:36 PM	Nov 6, 2015 10:04:48 PM	account

Log out all sessions

Want to logout users yourself as an admin?

keycloak

Admin

Master

Configure

- Realm Settings
- Clients
- Roles
- Identity Providers
- User Federation
- Authentication

Manage

- Users
- Sessions
- Events

Sessions

Realm Sessions | Revocation

Logout All

Client	Active Sessions
security-admin-console	1
account	2

Impersonate user?

Users

Search... | View all users | Unlock Users | Add User

Username	Last Name	First Name	Email	Actions
admin				Edit Impersonate Delete
john	Doe	John	john@example.com	Edit Impersonate Delete

And much more…

… of stuff you don’t need to implement yourself

Some (of many) features

- Session management
- Identity Management (with UI screens)
- Security Defences
- Password policies
- User impersonation
- ….

Lot of things challenging to implement securely…

… they take time…. and remember about possible vulnerabilities as well…

What remains for you?

- Configure client adapter
- Get (more) information from the token
- Call REST endpoints
  - Obtain additional information
  - Perform additional operations

And btw. how hard is to register a new authentication client with it?

Just copy paste configuration snippet…

Test-example

Settings | Credentials | Roles | Mappers | Scope | Revocation | Sessions | Offline Access | Clustering | Installation

Format Option Keycloak JSON

Download


```
{
  "realm": "master",
  "realm-public-key":
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAkF+UEF7oTS+qRjaasM3JKV0BWAoLGuEUtCHAu3xz0pZ1SJTJJJ50FLzA8MpbpzWV3pP0JKrnPYVUxqwl/I3LWOfJJ8fFOXLFF
PYpLUCcEql06aqiFvblxnd++qSv0jGao8KHU//4UoeiR0dgikiLpQiwCObOpJtdivhz4xh+nKOUuNw+IML41bsFQ500nGUy72iX8UaDFTQq4pxOfj6bPa0f+tuW+f4q1bPNIsGvLP24Qr9
yRt1QAWnQypBZ63kC69XdLmBN82KRi0g4JcTg82jkMM5wYM8YuED4GCxKhRjCQmok5I2kKQDrTZzTXa6ZeSeKrPrn+gYxh2FID/G+swIDAQAB",
  "auth-server-url": "http://localhost:8080/auth",
  "ssl-required": "external",
  "resource": "test-example",
  "credentials": {
    "secret": "6c0bf660-280a-45a5-937c-ea0c54fd386d"
  }
}
```

or use client self registration capabilities…

Single Sign Out of many HTML5 applications?

Sure…

and etc.

DEMO TIME