JavaCro'16

SSO for Modern Applications

Modern Applications

- Big shift in how we do web applications
- Classic:
  - Compose HTML on the server
  - Use server-side frameworks like JSF / JSP, PHP, ASP …
  - In a browser every click reloads a page

Modern Applications – Single Page Web Apps

JavaCro'16

For UI use JavaScript frameworks like Angular, Bootstrap, Foundation …

Web application packaged as static html / js using build frameworks like Yeoman, Bower

Web page does not reload when clicking - no UI flicker

REST calls are used to invoke server-side application logic

Mobile Applications and IoT

- Native mobile apps need user registration, login, server-side logic
- Use same REST endpoints that browser apps use
- Users want 'single' sign-in

  Do you ever type google password in your Android phone?

  Native app / browser session propagation

Authentication vs. Authorization

Authentication

- Prove that you are who you claim you are

- Credentials: password, private key – file on disk, keycard, USB dongle, OTP – calculator, USB dongle, app on your smartphone

- The act of logging into the system

Authentication vs. Authorization

JavaCro'16

Authorization

- Is **authenticated** user allowed to do something

- Examples:

    - Can this user delete that document
    - Can this user create a new user
    - Can this application client get that user's profile info

Social Web

- Not so long ago, there was a time when you downloaded some custom Twitter client, and it would ask you for your Twitter username and password.

- Giving your Twitter (or any other service) username and password to any app other than the official one from that service provider is a huge security risk.

- How can you create publicly available SocialApp that posts tweets if it can't have the user's username and password to perform actions on her behalf?

Web Standards for Auth and Authz

SAML 2.0

OAuth 2.0

OpenID Connect

JWT (JSON Web Tokens)

Token Based Security

- Token is a string issued by **authorization server**, that represents a granted permission to perform some action

- Token is usually encoded document of key-value pairs, accompanied by a cryptographic signature

- Other servers that trust the authorization server which issued a token, can use information within a token directly without doing a REST call to authorization server

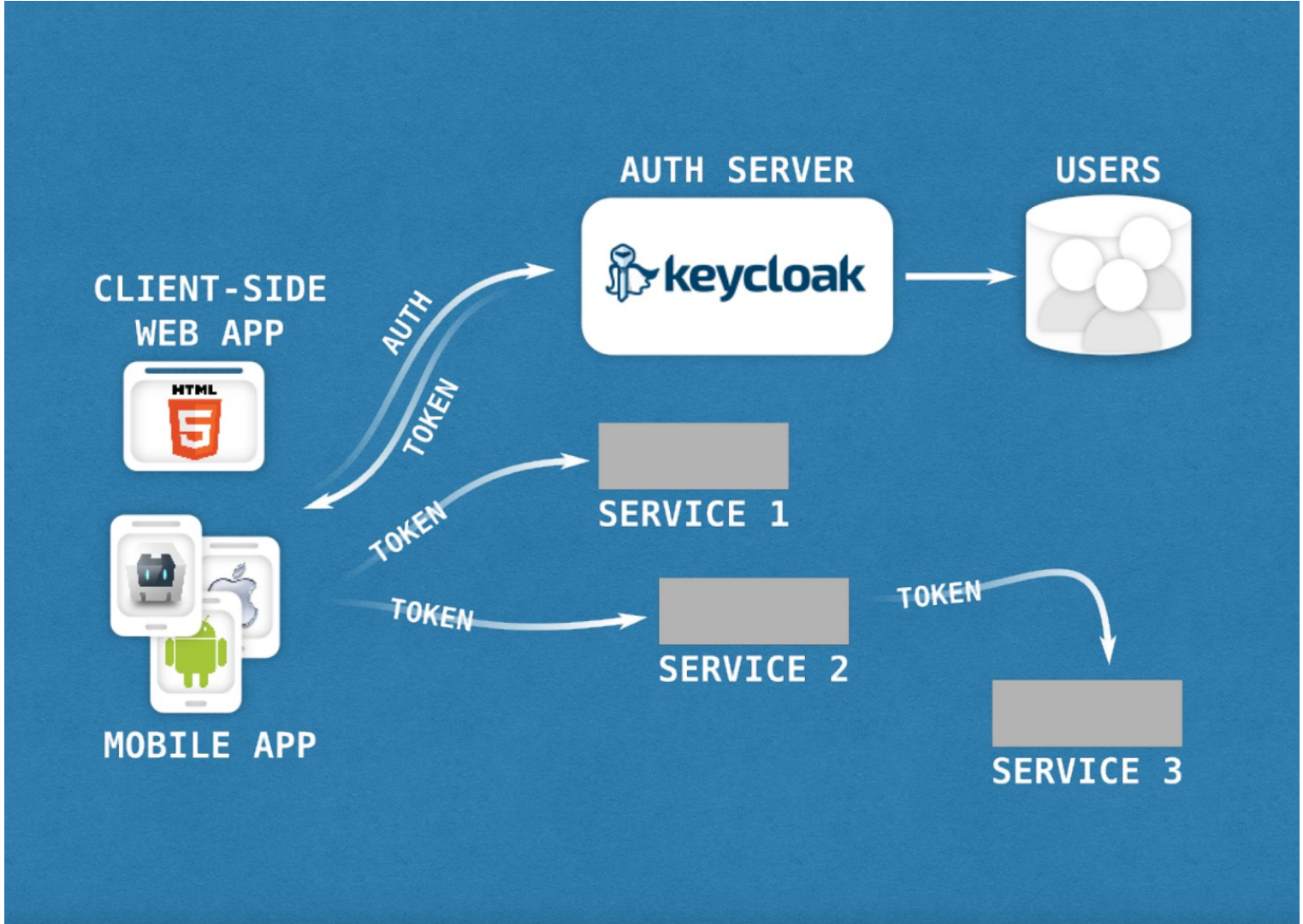
- Token normally has a limited lifespan - it times out.

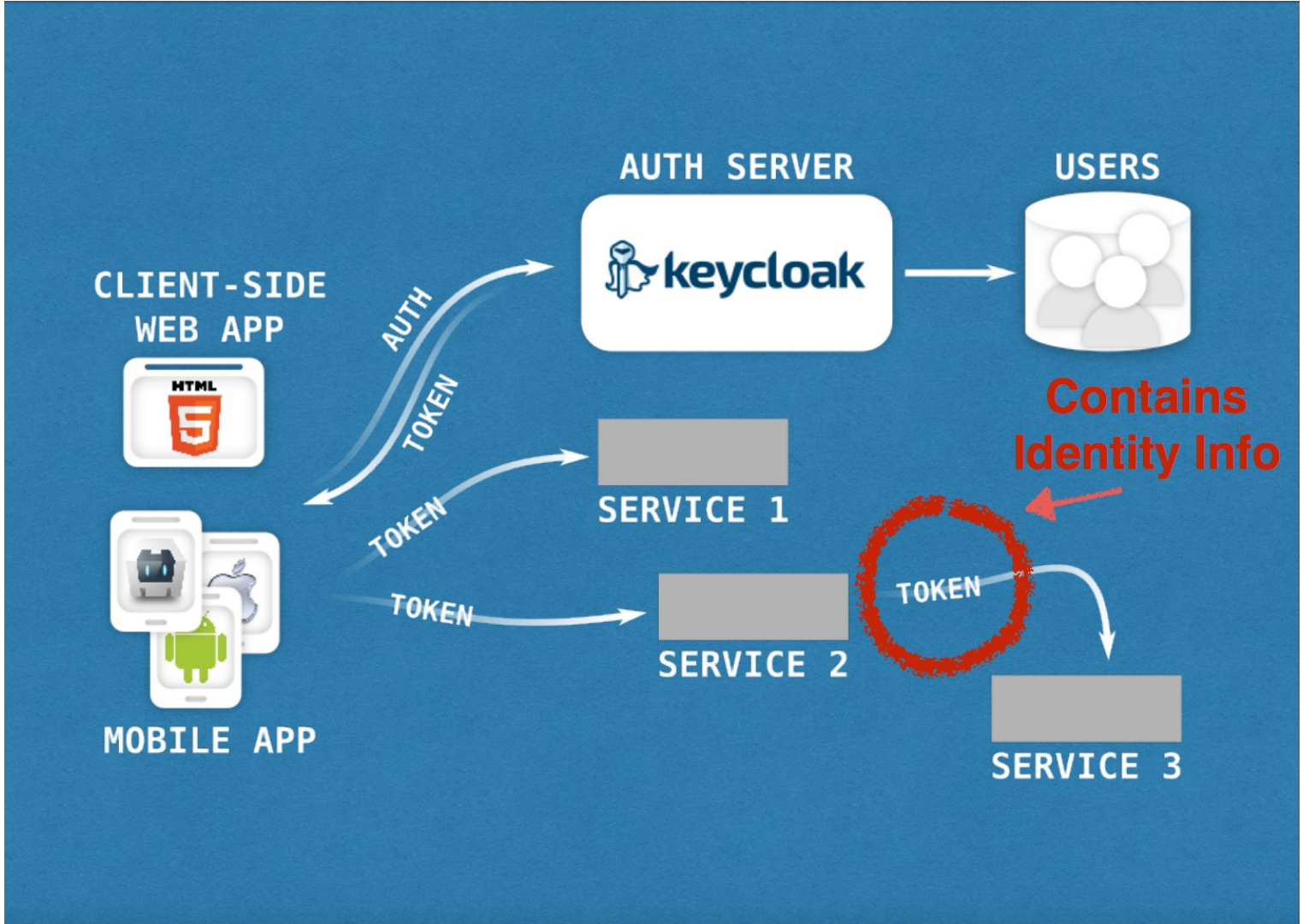
Token Standards

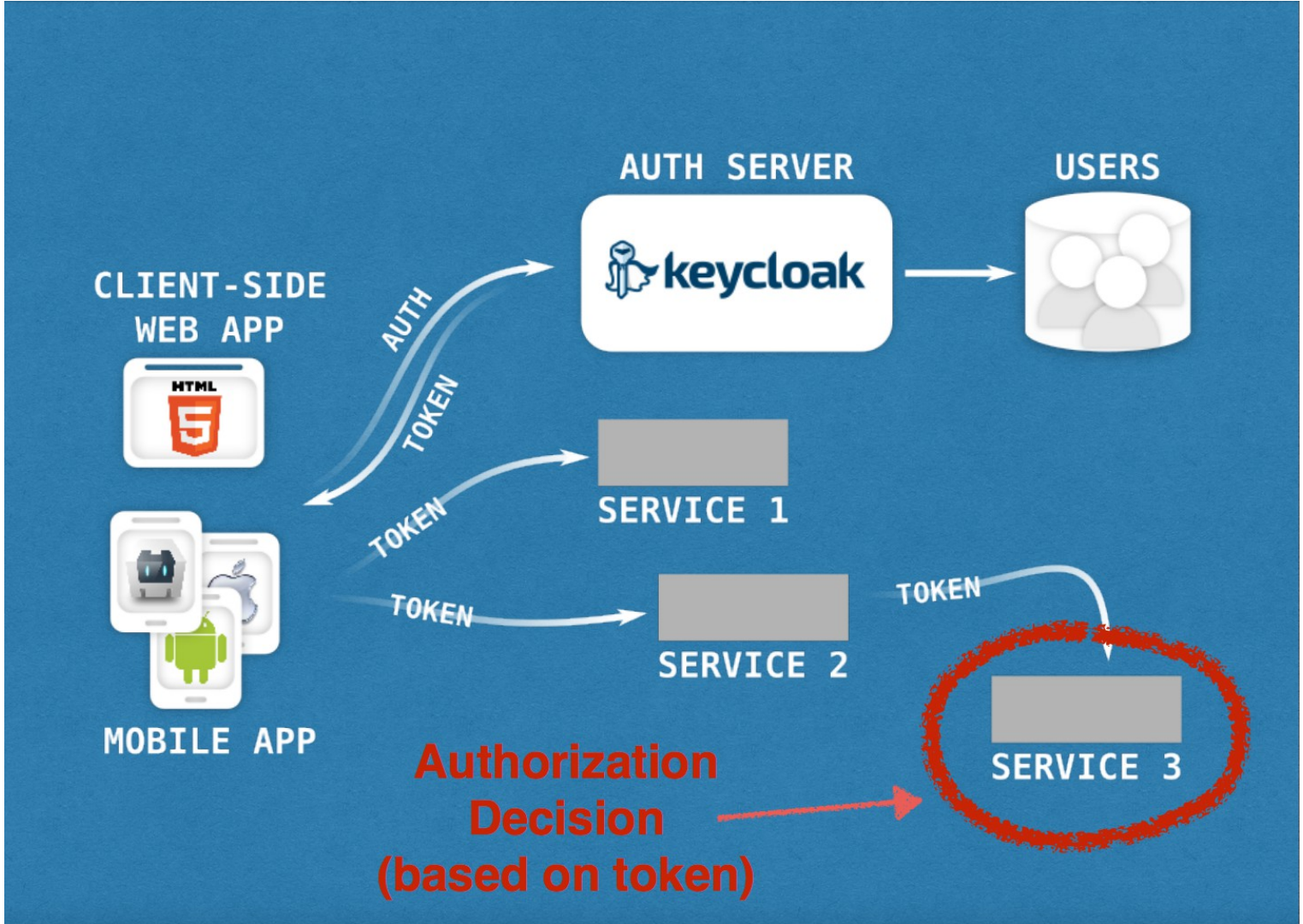
- Kerberos
- SAML 2
- JWT (JSON Web Token)

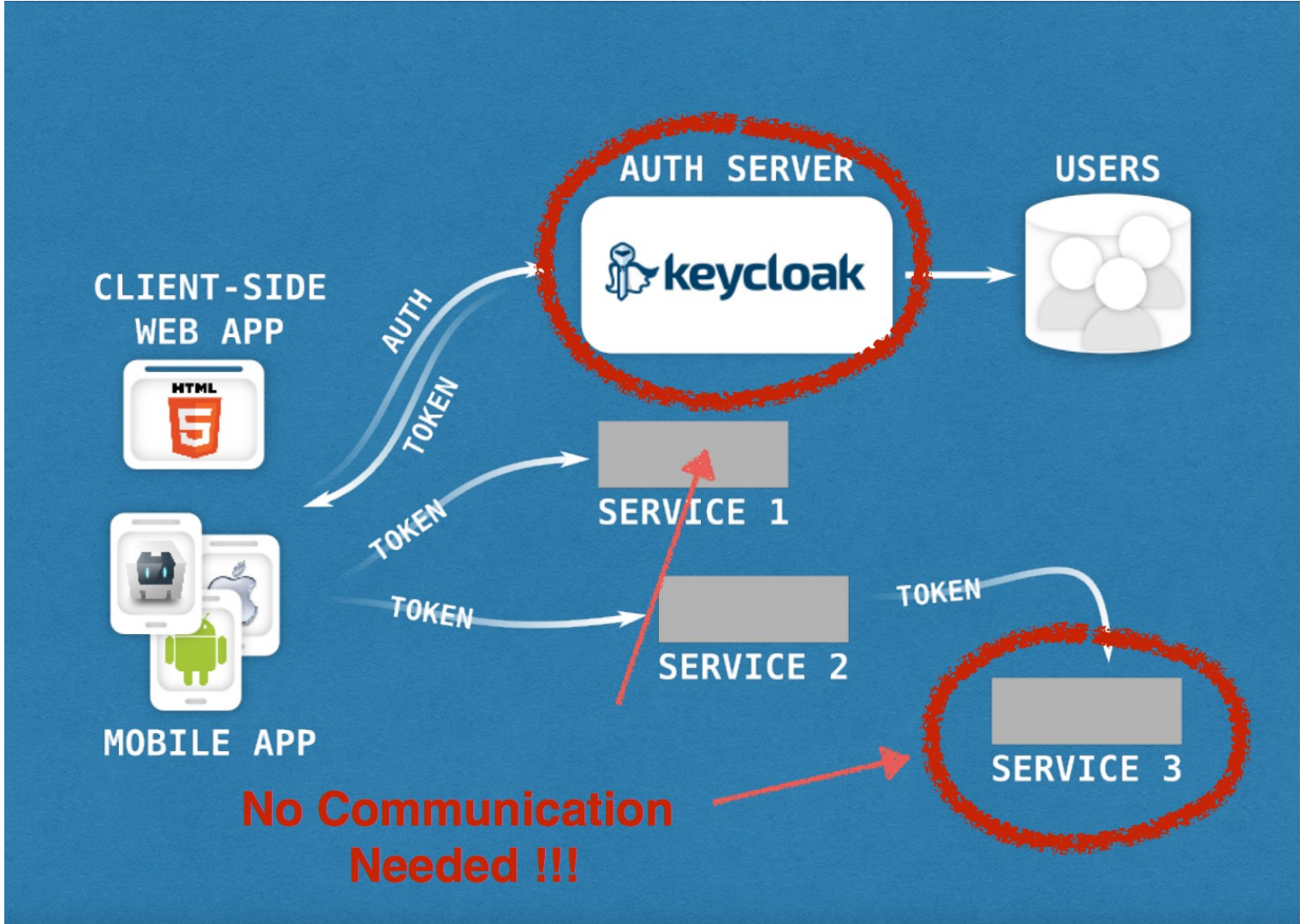
Distributed Security

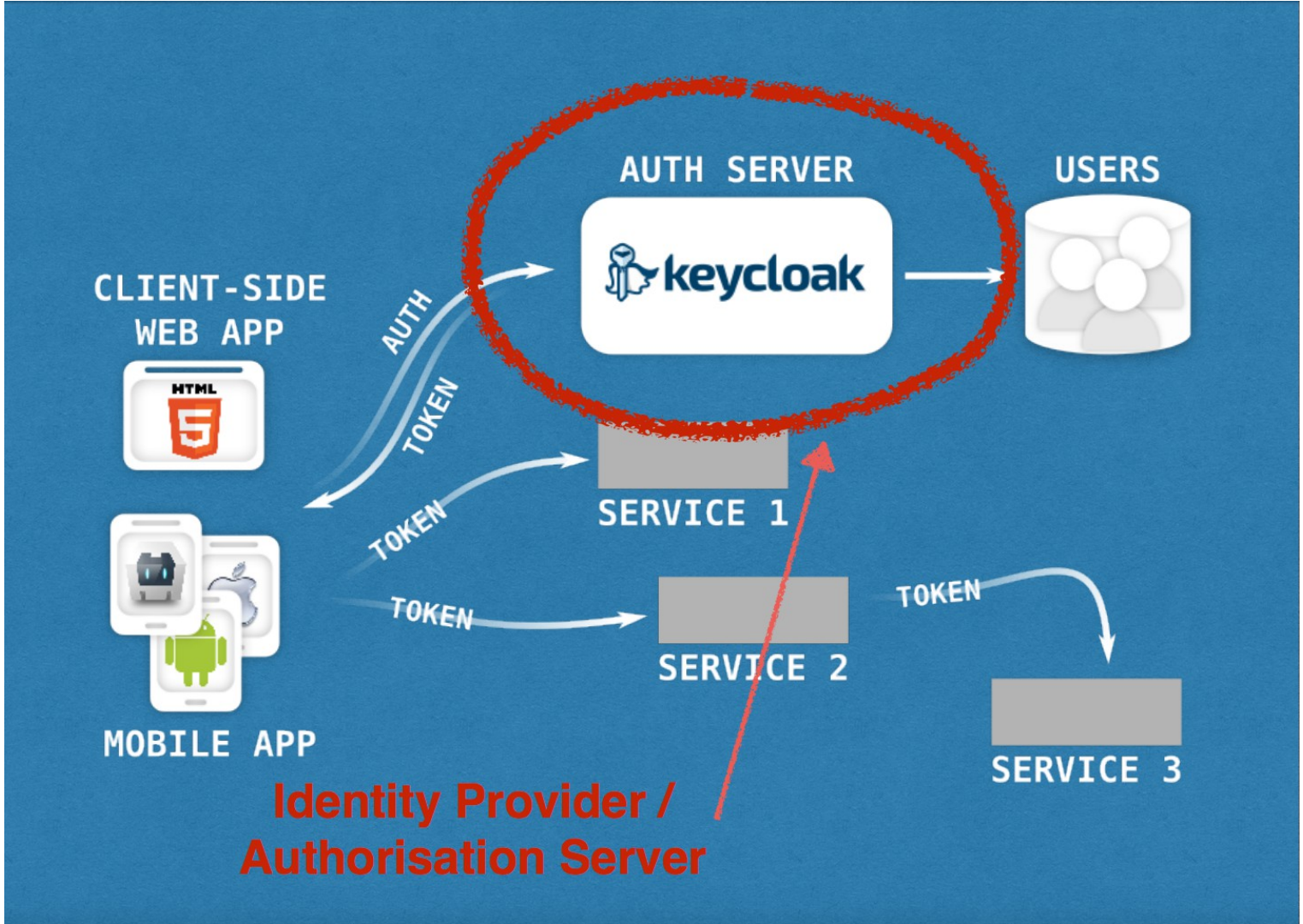
- Authentication server vs. Content servers
  - Server that provides REST endpoints for your application, and requires authentication does not have to be the same as the server that performs authentication
  - There may be multiple content servers, delegating authentication to the same authentication server
- Single Sign-On / Single Logout
  - You sign in once in your browser or in your mobile app, and obtain a token which you use for all subsequent requests even to multiple different applications as long as they trust the same authentication server

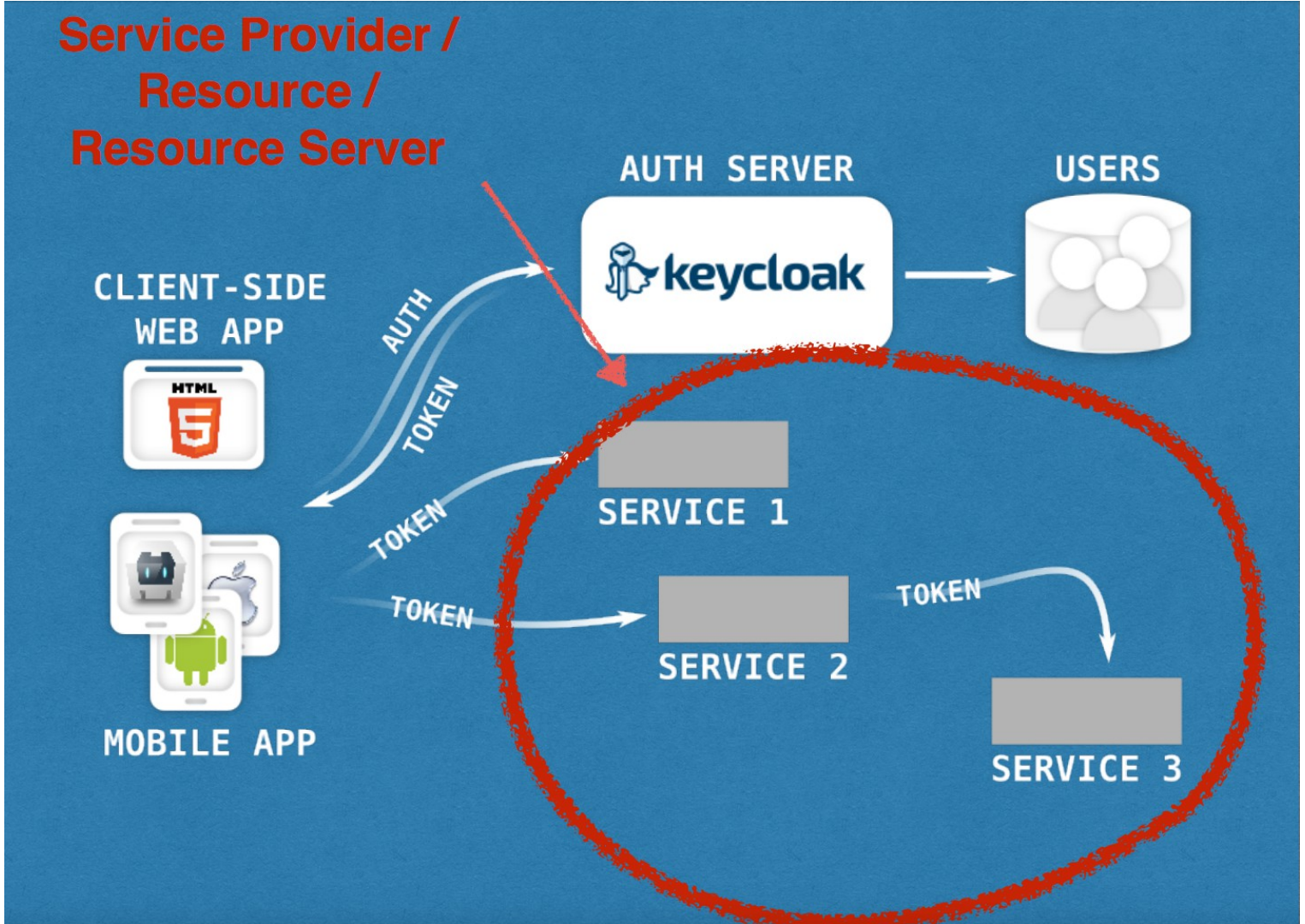

AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
TOKEN
SERVICE 1
TOKEN
TOKEN
SERVICE 2
SERVICE 3
MOBILE APP

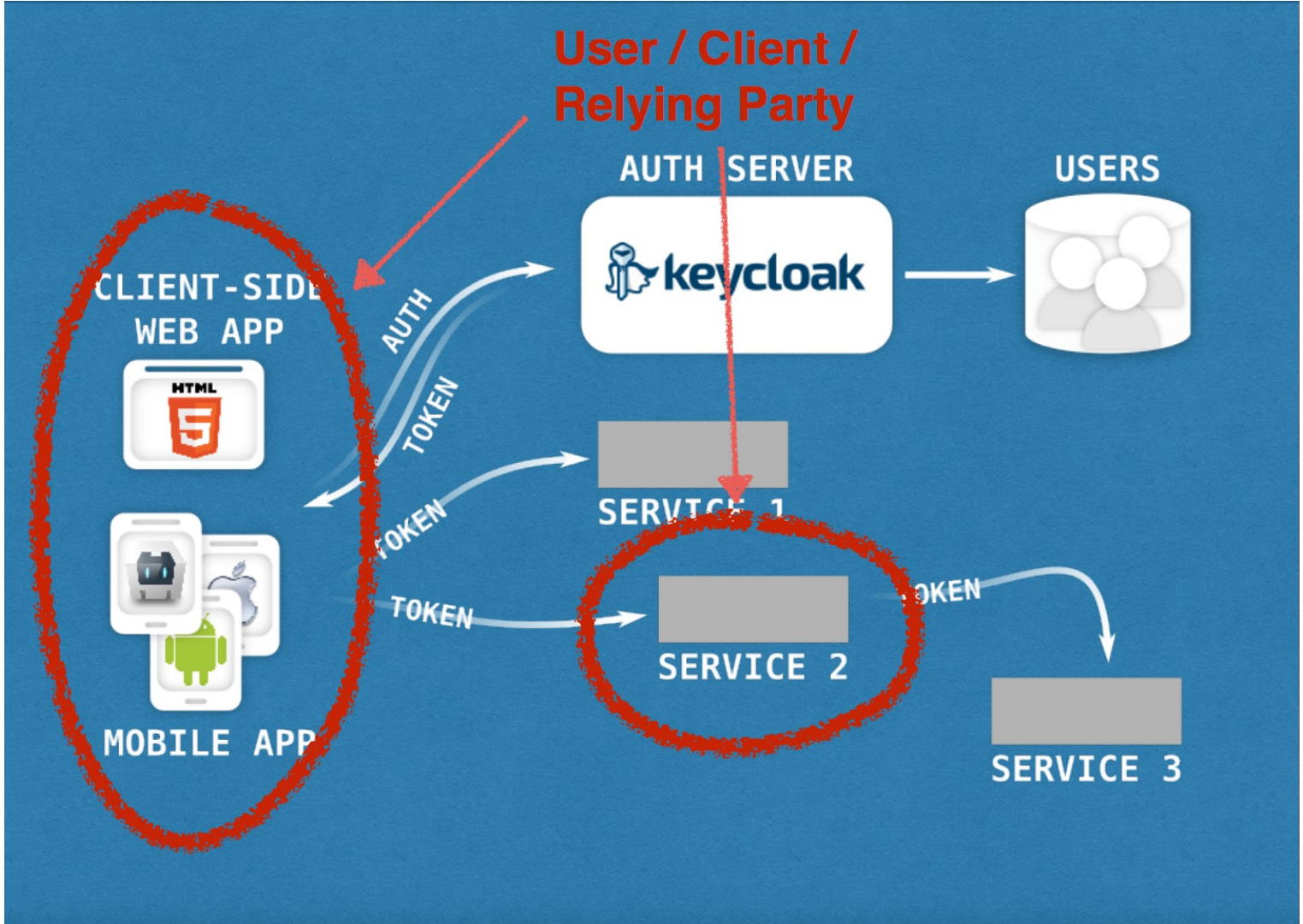

AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
Contains
Identity Info
SERVICE 1
TOKEN
TOKEN
TOKEN
SERVICE 2
SERVICE 3
MOBILE APP


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


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


AUTH SERVER
keycloak
USERS
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
TOKEN
SERVICE 1
TOKEN
SERVICE 2
TOKEN
SERVICE 3
MOBILE APP
Identity Provider /
Authorisation Server


Service Provider /
Resource /
Resource Server
AUTH SERVER
keycloak
USERS
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
TOKEN
TOKEN
TOKEN
SERVICE 1
SERVICE 2
SERVICE 3
MOBILE APP


User / Client /
Relying Party
AUTH SERVER
USERS
keycloak
CLIENT-SIDE
WEB APP
HTML
AUTH
TOKEN
TOKEN
TOKEN
SERVICE 1
SERVICE 2
SERVICE 3
MOBILE APP

SAML 2

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

- Very widely adopted – De facto standard in enterprise security integration

SAML 2

- XML token - verbose and heavy
- Complex flows, lots 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

- Not fun, but it's **widespread**

SAML 2

- Predates the mobile web revolution
- Doesn't fit mobile well:

  - To receive a token authentication server (Service Provider in SAML speech) needs to connect to some configured URL.

  - You don't run a server on your mobile phone do you :)
  - There are workarounds

JWT – JSON Web Token

- JSON Object Signing and Encryption (JOSE)
- 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
- Simple JSON, Encoded in Base64

JWT – JSON Web Token

Three parts: Header, Payload, Signature

<header-base64>.

<payload-base64>.

<signature-base64>

OAuth 2

- Standard that defines workflows for token based security for modern apps
- Allows for applications to integrate with different online services in the same way - once you've integrated with one it's very easy to integrate with others
- Used with social logins - SignIn with Facebook / Google
- Solves the problem of applications & services acting on behalf of the user

OAuth 2: Token based solution to a problem

- Traditionally some service requires username:password from client app
- Client app then has full access to user's resources
- Modern web problem:
  - Allow application to add a meeting to your Google Calendar **without giving it your username, and password**

Access Tokens

- Never have to blindly give away your password to cool third party apps any more

- When you uninstall an application or if it misbehaves, you can revoke its access, without affecting any other third party application, and without changing your Google password (if you used Google to Sign In)

- Secure - they can be short-lived, so they expire quickly, for online usage (during online user session)

- Convenient - they can be long-lived in safe environments, saved in a database, and used for offline access.

Refresh Tokens

- Access tokens are often short lived, when they expire they don't work any more
- But, we don't want to redirect user to login server every few minutes
- Solution: another type of token which can be used to get a new Access Token
- When your application sees that Access Token is about to expire or already has expired it makes a call to Authorization Server's Token Endpoint, sending Refresh Token, and getting back a new Access Token.

OAuth 2 explained by example

- Meetup.com and Google Calendar example
- User is a ‘resource owner’
- Google server hosting Google Calendar API is a ‘resource server’
- Google server where user signs in and grants permission to Meetup.com to access her calendar is a ‘authorization server’
- Meetup.com server which wants to add event to user’s Google Calendar is a ‘client’
- Client's goal is to obtain an Access Token.

OAuth 2 standard flow

Also called Authorization code flow

1) User triggers Add event to my Google Calendar on Meetup.com

2) Meetup.com sends a redirect HTTP response to user's browser, redirecting it to Google's Authorization Endpoint

3) At Google's Authorization Endpoint user is asked to login, then to confirm access to her Google Calendar to Meetup.com

4) User's browser is then redirected back to Meetup.com with an extra query parameter called 'code', that represents user grant - but is not yet an Access Token

OAuth 2 standard flow ...

5) Meetup.com extracts code query parameter, and makes a REST call to Google's Token Endpoint, sending there code, and at the same time authenticating with client_id, and client_secret received when application registered with Google.

6) Meetup.com receives Access Token. It may also receive a Refresh Token.

7) Meetup.com makes a REST call to Google Calendar API with new event info to add to the calendar, passing with it Access Token.

OAuth 2 standard flow ...

Important points

- The most secure standard way of obtaining an Access Token
- User (resource owner) never sees an Access Token
- Meetup.com (client) is assumed to be able to keep a secret
- Meetup.com (client) direct communication with Google (authorization server) is direct via TLS, and bypasses user's browser completely
- Meetup.com (client) can get a Refresh Token for offline use, and can for example add new emails to event attendees in your calendar event.

OAuth 2 implicit flow

A shorter way of retrieving an Access Token for JavaScript and native applications where user, or user's web agent acts as a client - making REST API calls from browser or from local native application.

Let's say we have a Meetup.com single page web application running on the phone.

1) User triggers Add event to my Google Calendar in the app
2) App needs an Access Token so it redirect browser to Google's Authorization Endpoint

OAuth 2 implicit flow ...

3) At Google's Authorization Endpoint user is asked to login, then to confirm access to her Google Calendar to Meetup.com

4) User's browser is then redirected back, this time without Authorization Code - instead it receives an Access Token in the fragment part or the redirect response url.

5) JavaScript in browser app makes a REST call to Google Calendar API with new event info to add to the calendar, passing with it Access Token.

OAuth 2 implicit flow ...

Important points

- Browser app (client) loaded from Meetup.com is assumed to NOT be able to keep a secret
- User (resource owner) CAN see an Access Token.
- Third party browser plugins can intercept it, browser log files can contain it, user can copy / paste it using browser debug tools ...
- Browser app (client) has to use TLS to communicate with Google Calendar REST API (resource server), and with Google Authorization Endpoint (authz server)
- There is no Refresh Token, there is no Authorization Code.
- One less round-trip.

OAuth2 other standard flows - Password flow

- Client gets user's username and password, and exchanges them for Access Token
- The benefit - no need to store user's login info, with all the associated issues
- Requires strong trust since users gives username, and password to the application which can then do anything it likes with them.

- This should only be used with official applications released by the service provider.

OAuth2 other standard flows - Client credentials flow

- When client has to access some resource provider's resources in its own name, and for itself, rather than on behalf of any user.
- For example, Facebook uses application token which client has to obtain using client credentials to then be able to access Facebook's endpoint for validating Access Tokens

OpenID Connect

- OAuth 2 only allows client (Meetup.com) to get from authorization server (Google) the Access Token, and Refresh Token

- What if we want to know user’s unique user id, or if we want some info from user's profile?

- OIDC is an extension of OAuth 2 that adds this information about authentication to the Token Endpoint response - next to Access Token we also receive an ID Token

- There is an additional UserInfo Endpoint where user profile info can be retrieved

Keycloak

Authentication server

- Using token based security
- Following latest web standards (OAuth2, OpenID Connect, SAML 2, JWT)
- Extensible, pluggable, customizable
- Administration UI
- Administration REST API
- Comes with themable UI for integrating your applications
- Provides to your application users: Registration, Login, Logout, Forgotten password, OTP, Captcha, Self-service account administration

keycloak

Admin

Master

Configure

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

Manage

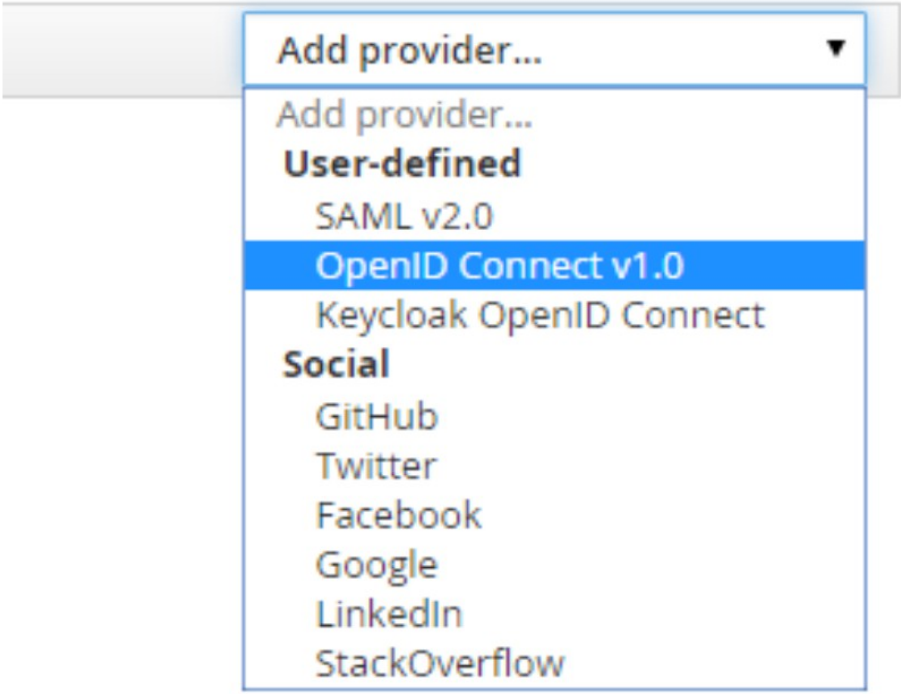
- Users
- Sessions
- Events

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

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!

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 TOTP	☑	☐
Update Password	☑	☐
Terms and Conditions	☑	☐
Update 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

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

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

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 [illegible]C 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

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

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

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

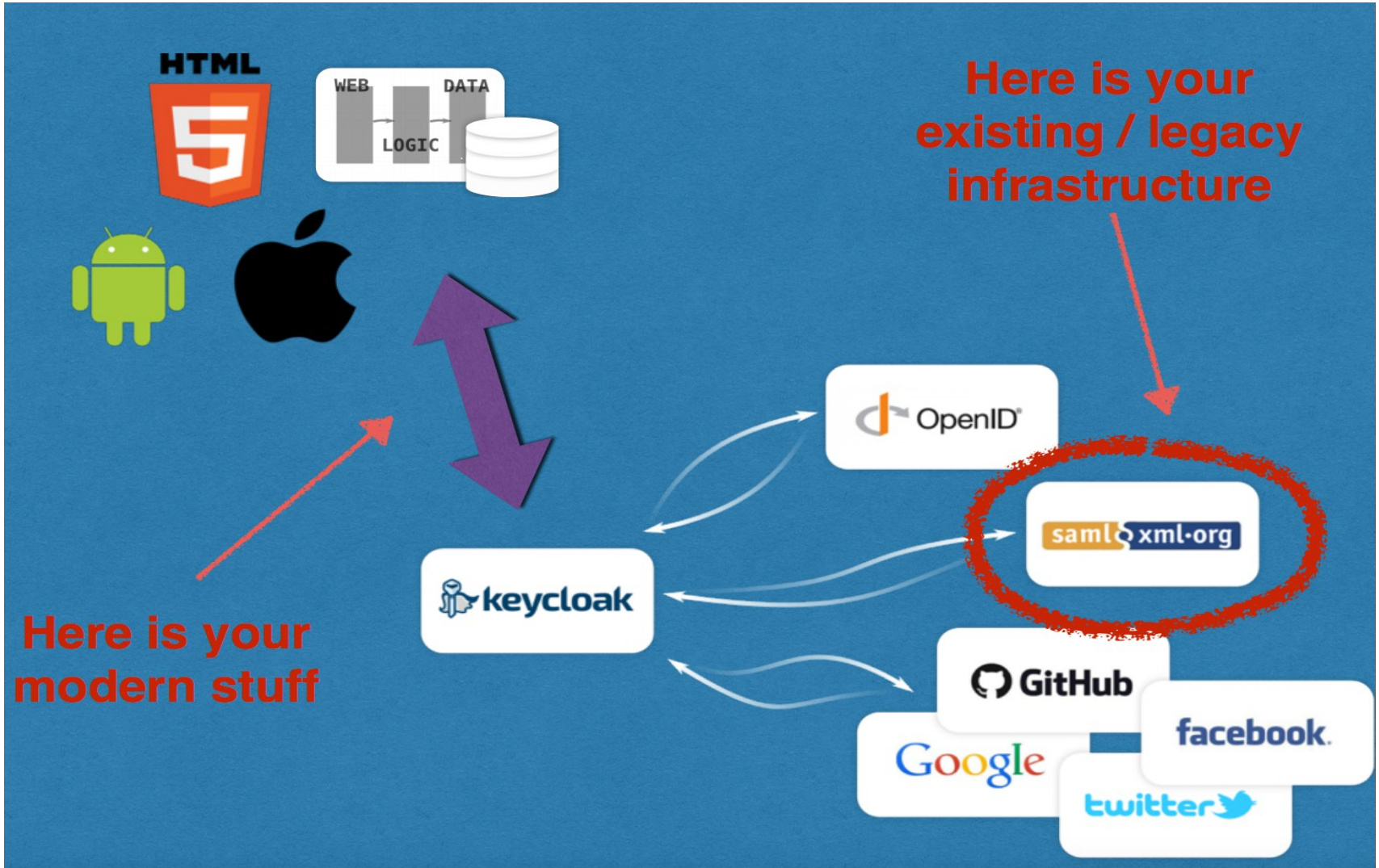
Test-example

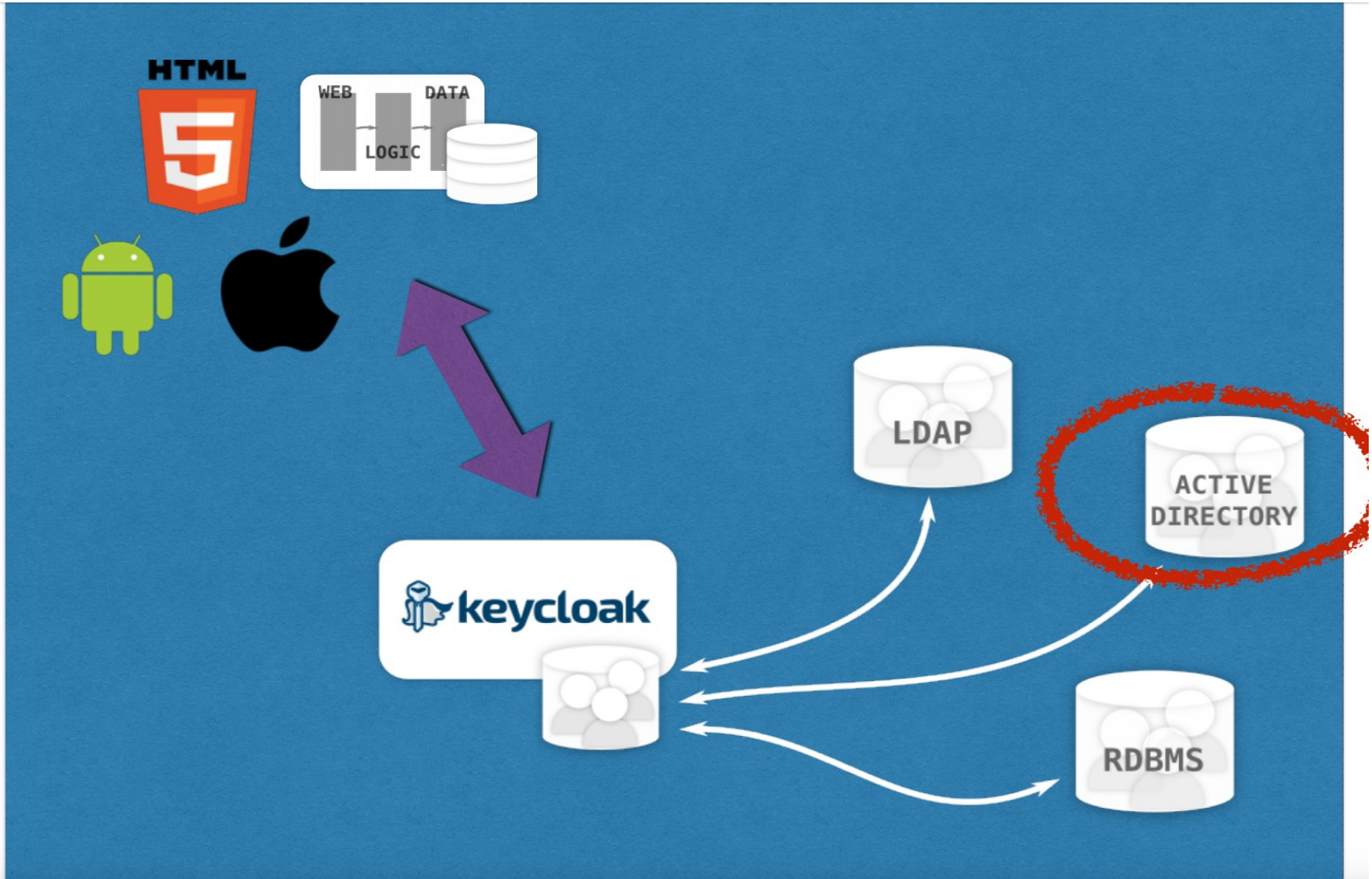
Settings | Credentials | Roles | Mappers | Scope | Revocation | Sessions | Offline Access | Clustering | Installation

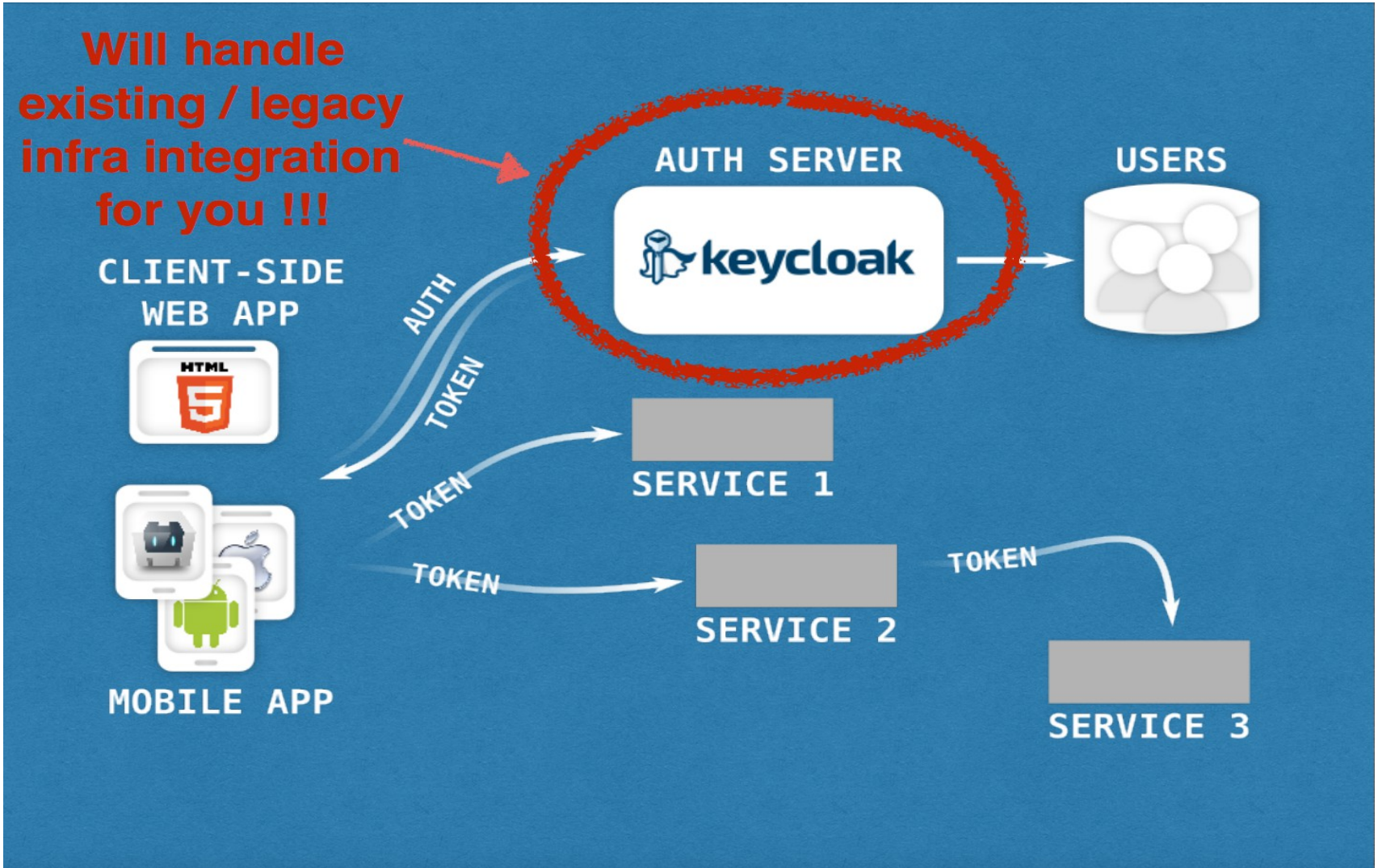
Format Option: Keycloak JSON

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"
  }
}
```


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


HTML
WEB
DATA
LOGIC
keycloak
LDAP
ACTIVE DIRECTORY
RDBMS


Will handle existing / legacy infra integration for you !!!
AUTH SERVER
keycloak
USERS
CLIENT-SIDE WEB APP
HTML
AUTH
TOKEN
TOKEN
TOKEN
TOKEN
SERVICE 1
SERVICE 2
SERVICE 3
MOBILE APP

Keycloak server and adapters

- Server
  - Stand alone server where your application redirects users to login
  - Serves the login form, registration form, lost password flow, user account settings page ...
  - Authentication server, and user account management server

- Adapters
  - Provide Keycloak server integration for your application
  - Redirect to authorization endpoint to retrieve access token
  - Acts as a client in OAuth 2 terminology to exchange authorization code for access token
  - Automatically validates tokens, checks timeouts, and signatures
  - Automatically refreshes tokens using refresh token against token endpoint

Keycloak Features

- OpenID Connect and SAML 2.0 SSO and Single Log Out for browser applications
- Social Broker
  - Enable Google, Facebook, Yahoo, Twitter social login with no code required.
- Identity Broker
  - Delegate to an external SAML 2.0 or OIDC broker for auth.
- Optional LDAP/Active Directory integration
- Optional User Registration, with optional Recaptcha ability
- Password and TOTP support (via Google Authenticator).
  - Client cert auth in the plans
- User session management from both admin and user perspective

Keycloak Features ...

- Customizable themes for user facing pages:
  - login, grant pages, account management, emails, and admin console all customizable!
- OAuth 2.0 Bearer token auth for REST Services
- Integrated Browser App to REST Service token propagation
- Admin REST API
- CORS Support
- Completely centrally managed user and role mapping metadata.
  - Minimal configuration at the application side
- Admin Console for managing users, roles, role mappings, applications, user sessions, allowed CORS web origins, and OAuth clients.

Keycloak Features ...

- Deployable as standalone server, Wildfly Swarm microservice, or an Openshift cloud service - docker image.
- Supports Wildfly, JBoss AS7, EAP 6.x, Tomcat, Jetty, and Pure Javascript applications. Plans to support Node.js, Rails, Grails, and other non-Java applications - Python, Go, PHP.
- HTTP Security Proxy for environments/platforms/languages that don't have a client adapter
- JavaScript/HTML 5 adapter for pure JavaScript apps
- Session management from admin console

Keycloak Features ...

- Claim / assertion mappings.
  - Make your tokens and assertion XML look however you want.
- Revocation policies
- Password policies
- Impersonation. Allow your admins to impersonate a user to debug problems.

And more.

Getting Keycloak

- keycloak.jboss.org
- Current stable version: 1.9.4.Final
- Download Keycloak server
- Startup Keycloak server standalone
- Download adapter zip for your application container
- Follow installation instructions inside the adapter zip to install

Demo

Thank you!