Keycloak

W mniej typowych scenariuszach

Warsaw 20.06.2023

FORUM 2023

Agenda

- Nudne rzeczy:
  - Agenda ✔
  - O mnie
- Rzeczy teoretyczne:
  - OAuth 2
  - Terminologia
- Rzeczy ciekawe:
  - OAuth Device Flow
  - Implementacja Identity Provider
- Wymagane rzeczy
  - Ocena prezentacji
  - Pytania i odpowiedzi

O mnie

Łukasz Dywicki

W IT od 2005 roku. Pracuję z Keycloak od 2018 roku / v3.4.3, głównie na bazie kodu źródłowego. Współpracuję z OSEC od 2020 r.

Główne osiągnięcia do tej pory:

- [2018] Integracja logowania dla aplikacji mobilnej.
- [2019] Dostosowanie KC dla dużej sieci sklepów.
- [2020] Integracja OAuth Device Grant (KEYCLOAK-7675).
- [2020] Uruchomienie "Cross DC" dla jednego z banków w PL.
- [2021] Migracja powyższego do Red-Hat SSO.
- [2022] Uruchomienie multi-tenant KC dla projektu w chmurze.
- [2023] Integracja w infrastrukturze MObywatel.

cd. O mnie

Łukasz Dywicki

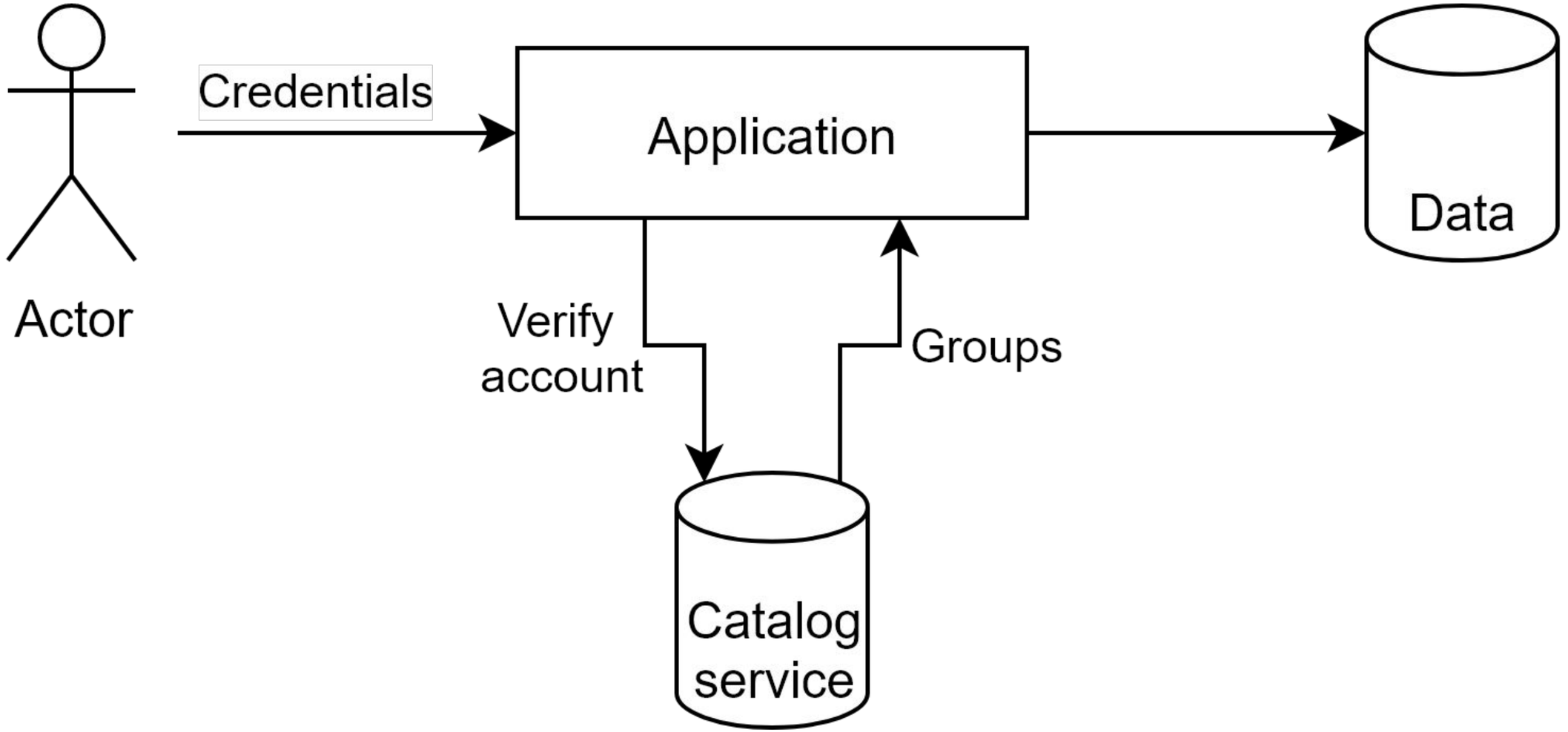
... Jestem również autorem szkolenia, które uczy uczestników jak tworzyć rozszerzenia dla Keycloak w *bardziej typowych scenariuszach*.

Rzeczy teoretyczne

Czym jest OAuth 2

- Popularny standard internetowy
- Opublikowany w 2012 r.
- Protokół delegacji dostępu
- Konkurencja dla OpenID

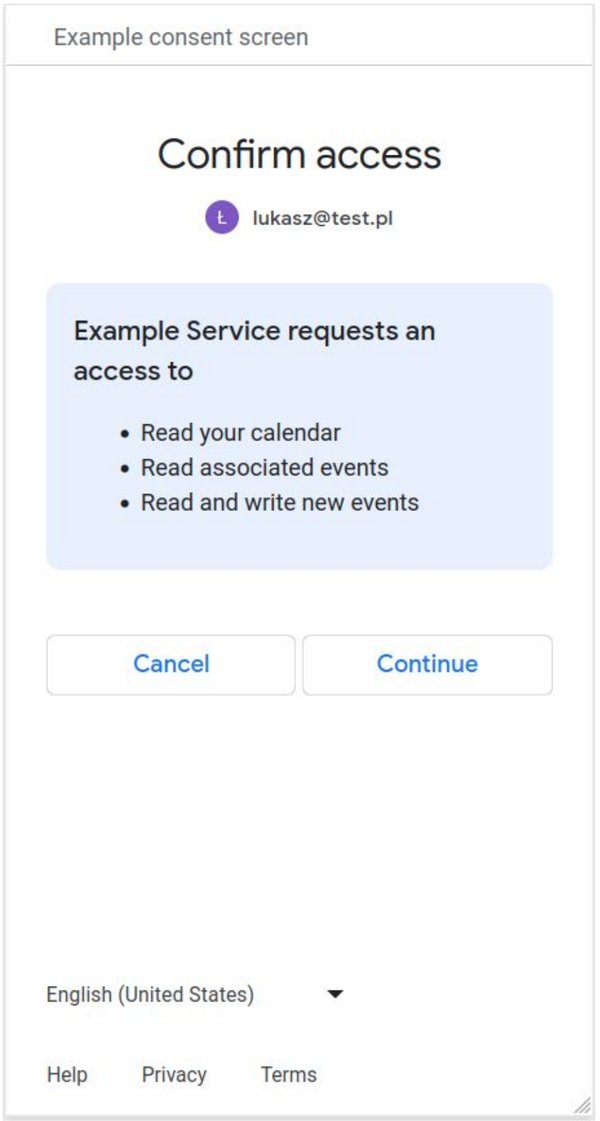
Usługi katalogowe

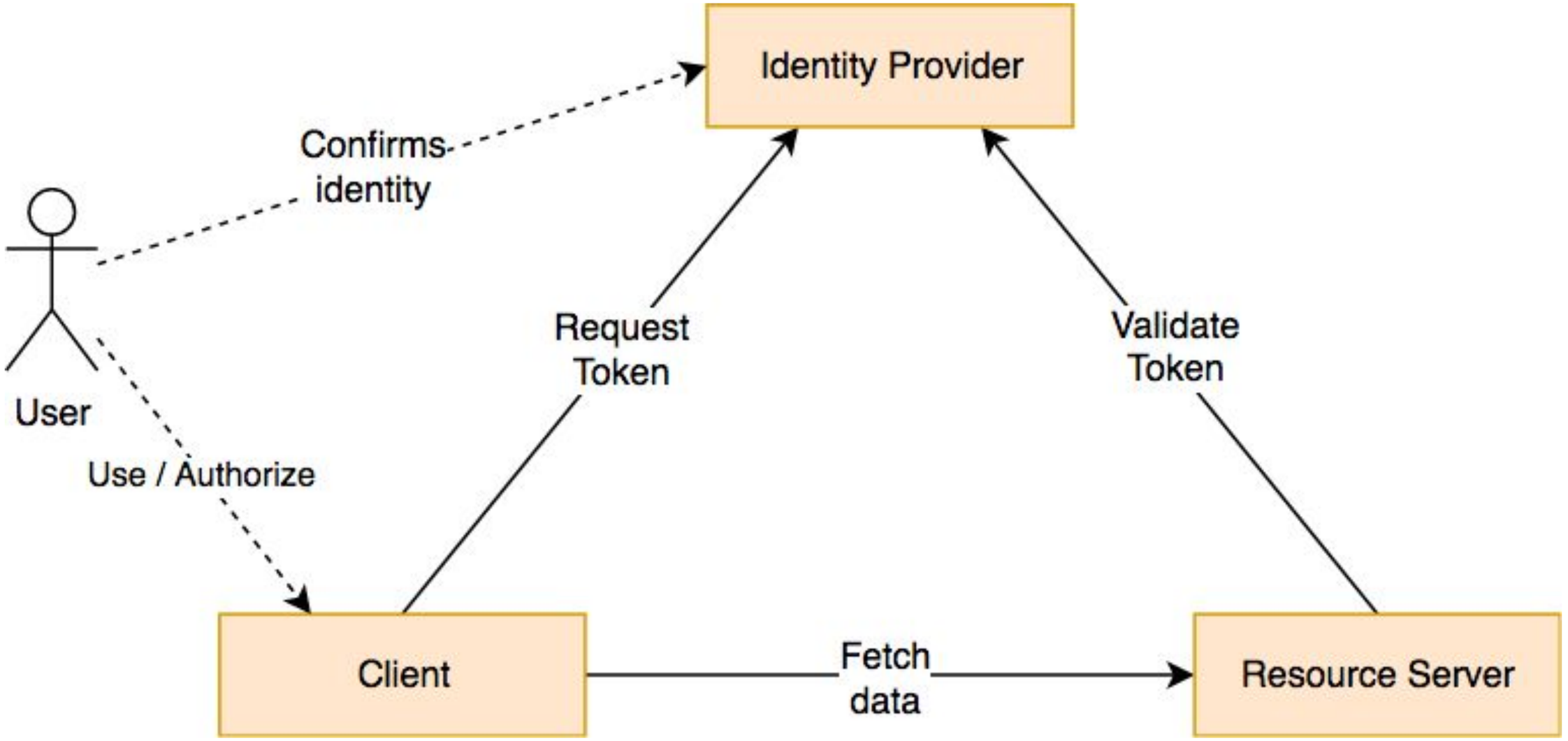


OAuth 2 - delegacja dostępu

- Podstawowe zwroty, które opisują założenia OAuth 2:
  - Jestem użytkownikiem i jestem właścicielem zasobów
  - Aplikacja potrzebuje dostępu do zasobów użytkownika
  - Jestem użytkownikiem i mogę pozwolić aplikacji na dostęp do moich zasobów
- W standardowych scenariuszach aplikacja:
  - Nie powinna przetwarzać hasła użytkownika
  - Powinna poprosić i korzystać z tokenu dostępowego
- Aplikacja wie o zakończeniu autoryzacji tylko i wyłącznie wtedy, gdy otrzyma token.
  Brak tokenu = brak użytkownika, brak autoryzacji.

OAuth 2 - delegacja dostępu

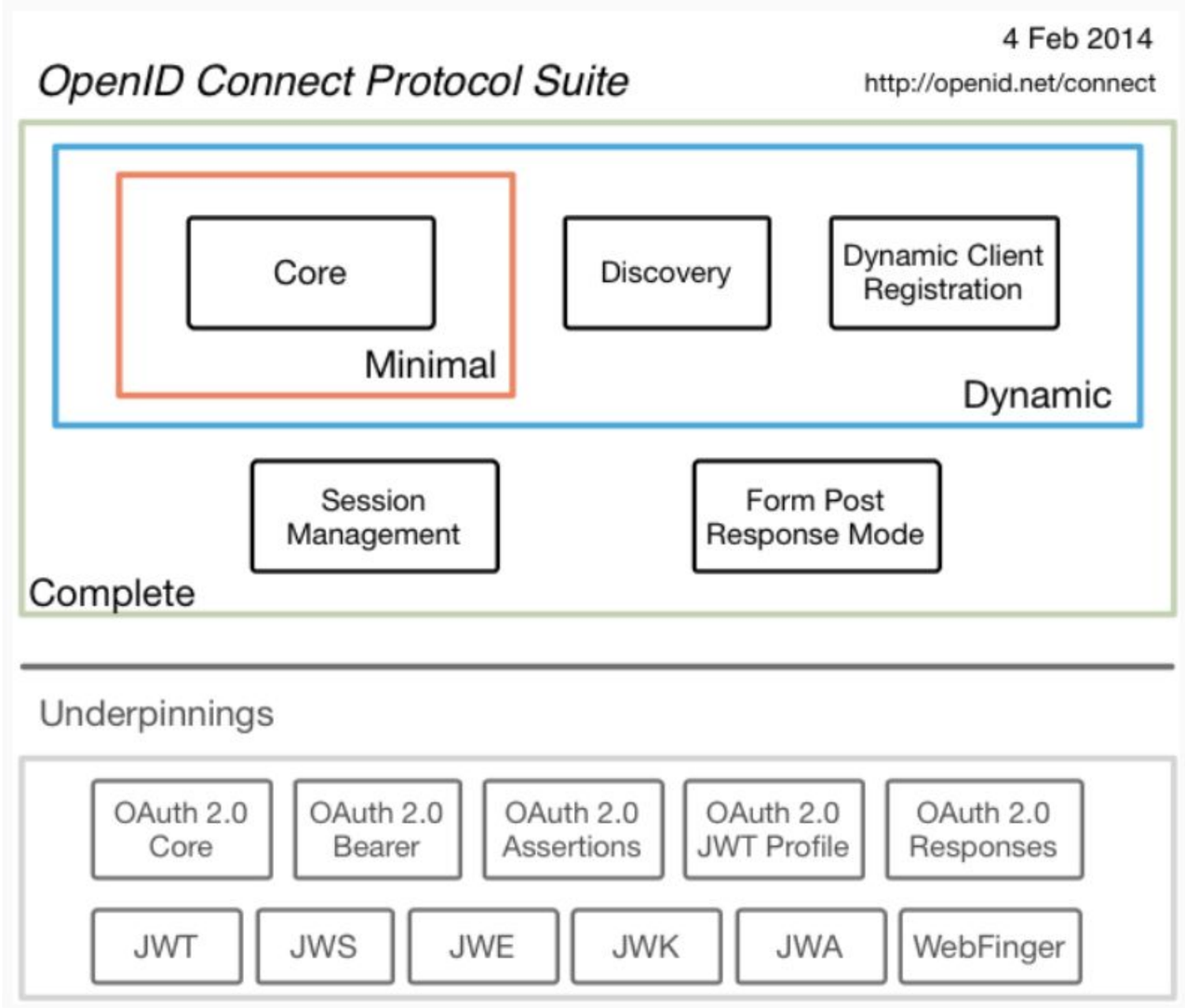



Identity Provider
Confirms identity
User
Use / Authorize
Request Token
Validate Token
Client
Fetch data
Resource Server

OpenID Connect

- Warstwa "uwierzytelniania i tożsamości" zbudowana na bazie OAuth 2
- Protokół opublikowany w 2014, w stosunku do OAuth 2 dodaje między innymi:
  - możliwość odczytania tożsamości użytkownika
  - możliwość odczytania profilu użytkownika
- OIDC nie należy mylić z OpenID
- OIDC definiuje:
  - Standard opisu dla dostawców tożsamości
  - Sposób dynamicznej rejestracji klientów
  - Sposób zarządzania sesją użytkownika i wylogowania
- Promuje użycie JSON Web Token (JWT)

Protokoły OpenID Connect



Terminologia OAuth2/OIDC

Definicje w OAuth 2

- Aktorzy
  - Client
  - Authorization server
  - Resource server

Rodzaje tokenów

- access token
- refresh token
- id token

- "Końcówki"
  - Authorization endpoint
  - Token Endpoint
  - User info Endpoint

JWT (związane z OIDC):

- Claim

OAuth 2 - Klient

Klient może również być określany jako:

- application
- "Relying Party"

Klient prosi użytkownika (pośrednio) o dostęp do zasobów, który jest potwierdzony autentykacją. Autentykacja jest obsługiwana przez serwer autoryzacyjny (authorization server).

Klient może być uruchomiony gdziekolwiek, może być czymkolwiek. Od wiersza poleceń poprzez aplikację mobilną po serwery, skończywszy na .. samym serwerze autoryzacyjnym (np. account console).

cd. OAuth 2 - Klient

Istnieje kilka typów Klientów:

- public
- confidential
- bearer-only (zależnie od implementacji)

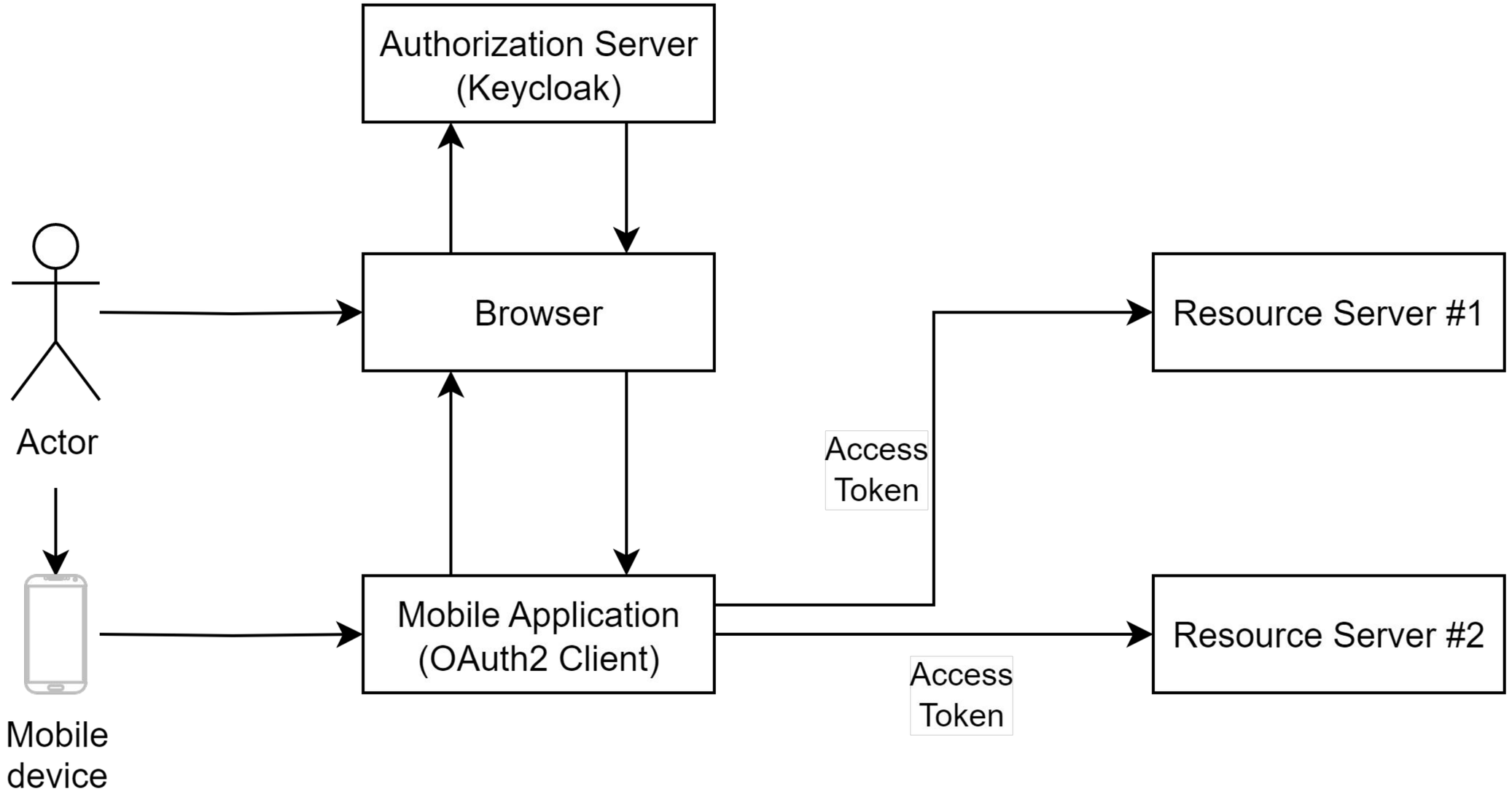
Różnica wynika z tego kto/co kontroluje daną aplikację. Jeżeli aplikacja działa na urządzeniu zewnętrznym zazwyczaj będzie to klient publiczny. Jeżeli w infrastrukturze wewnętrznej organizacji jako serwer - wówczas może to być klient prywatny.

Klient może być tylko jednego typu. Typ określa czy wywołania Token Endpoint wymagają dodatkowej autoryzacji (tak dla confidential).

OAuth 2 - Aktor Bearer only

Jest to specjalny typ klienta, który nie korzysta z autoryzacji z udziałem użytkownika.
Ten typ klienta może wchodzić w interakcję z serwerem autoryzacyjnym, ale nie może poprosić o dostęp w imieniu użytkownika.

Autoryzacja aplikacji mobilnej



Rzeczy ciekawe

Bezpieczeństwo urządzeń/IoT

Bezpieczeństwo w przypadku urządzeń końcowych to problem wieloaspektowy.

Delegacja dostępu dla tych urządzeń do kolejny krok poprawiający bezpieczeństwo warstwy aplikacyjnej.

Bezpieczeństwo na warstwie sprzętowej wymaga odpowiedniego przygotowania instalacji systemu i komponentów sprzętowych (np. TPM2).

Delegacja dostępu dla urządzenia

- Wracamy do założeń OAuth 2 i założeń:
  - Jestem użytkownikiem i jestem właścicielem zasobów
  - **Urządzenie** potrzebuje dostępu do zasobów użytkownika
  - Jako użytkownik muszę najpierw pozwolić urządzeniu na dostęp do moich zasobów
- Takie podejście pozwala na powiązanie urządzenia z użytkownikiem
- Urządzenie po zakończeniu procesu autoryzacji może działać jako standardowy klient OAuth 2/OIDC

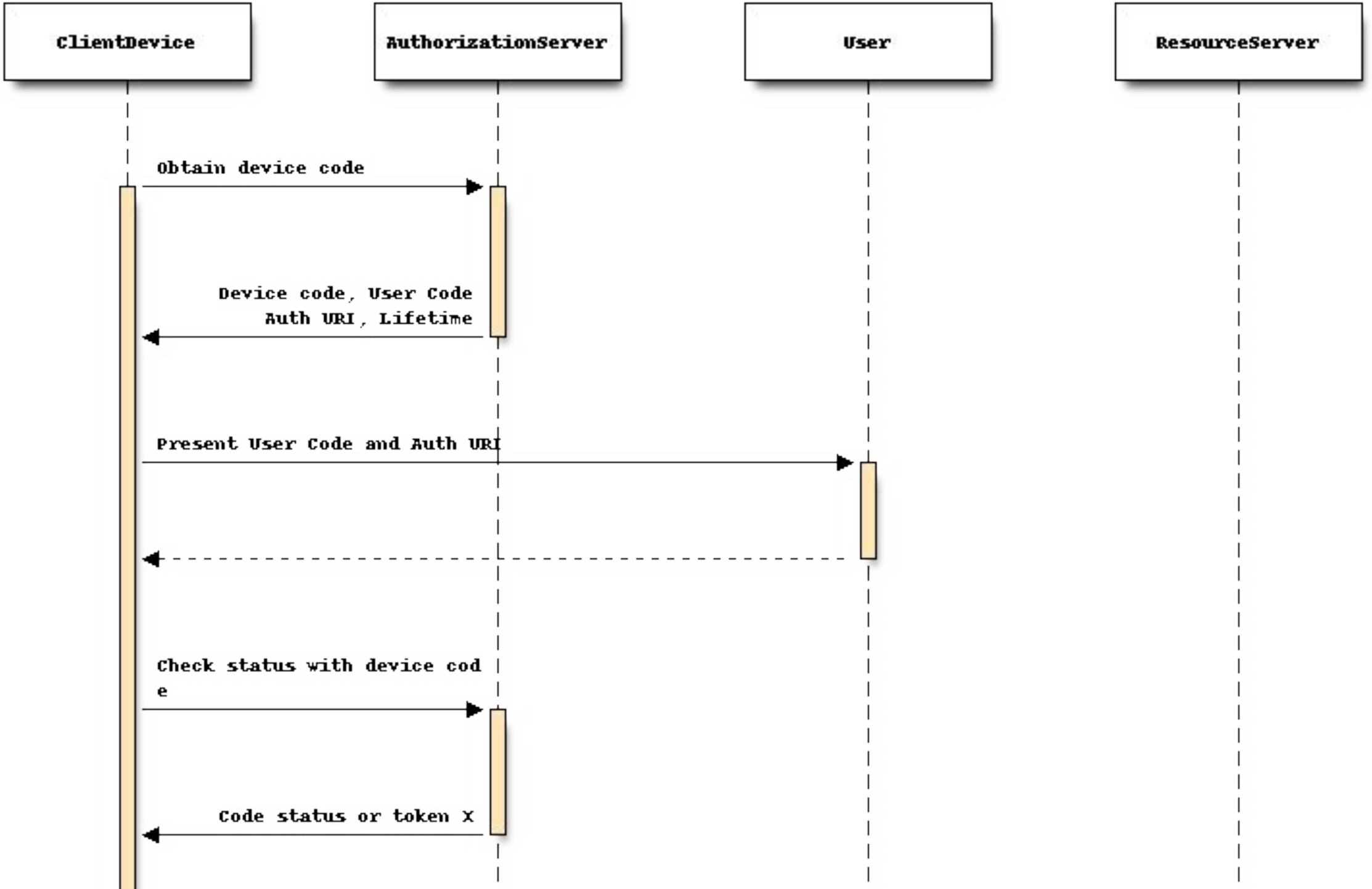
Przebieg OAuth 2 - Device flow

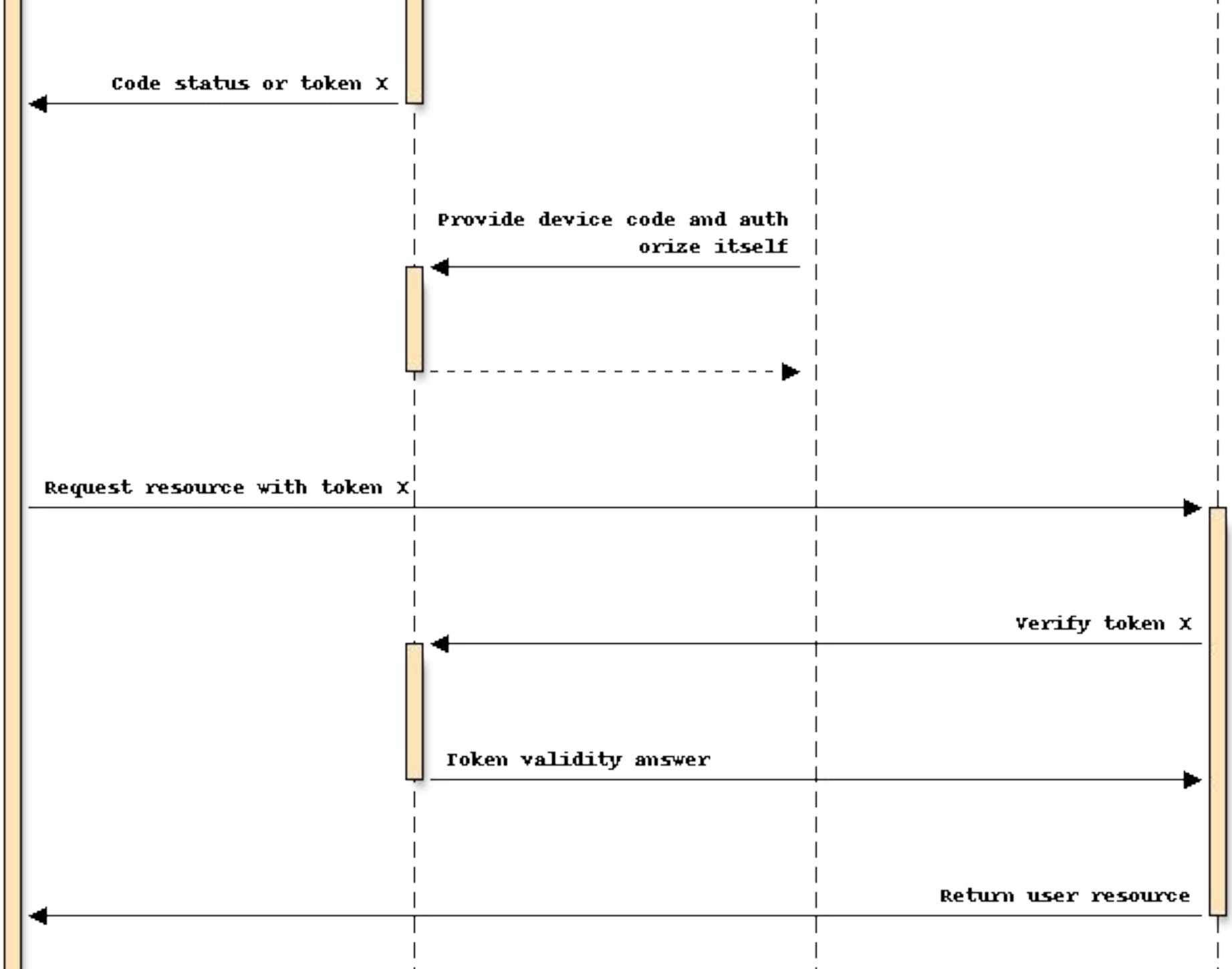
1. Urządzenie prosi o kod (wysyła client id i secret)
   a. Kod jest ważny przez określony czas, wskazany przez serwer
   b. Urządzenie rozpoczyna cykliczne sprawdzanie Token Endpoint
2. Urządzenie wyświetla kod/link dla użytkownika
3. Użytkownik dokańcza proces na telefonie/laptopie niezależnie od urządzenia, które prosi o dostęp do zasobów.

Zastosowanie Device Flow

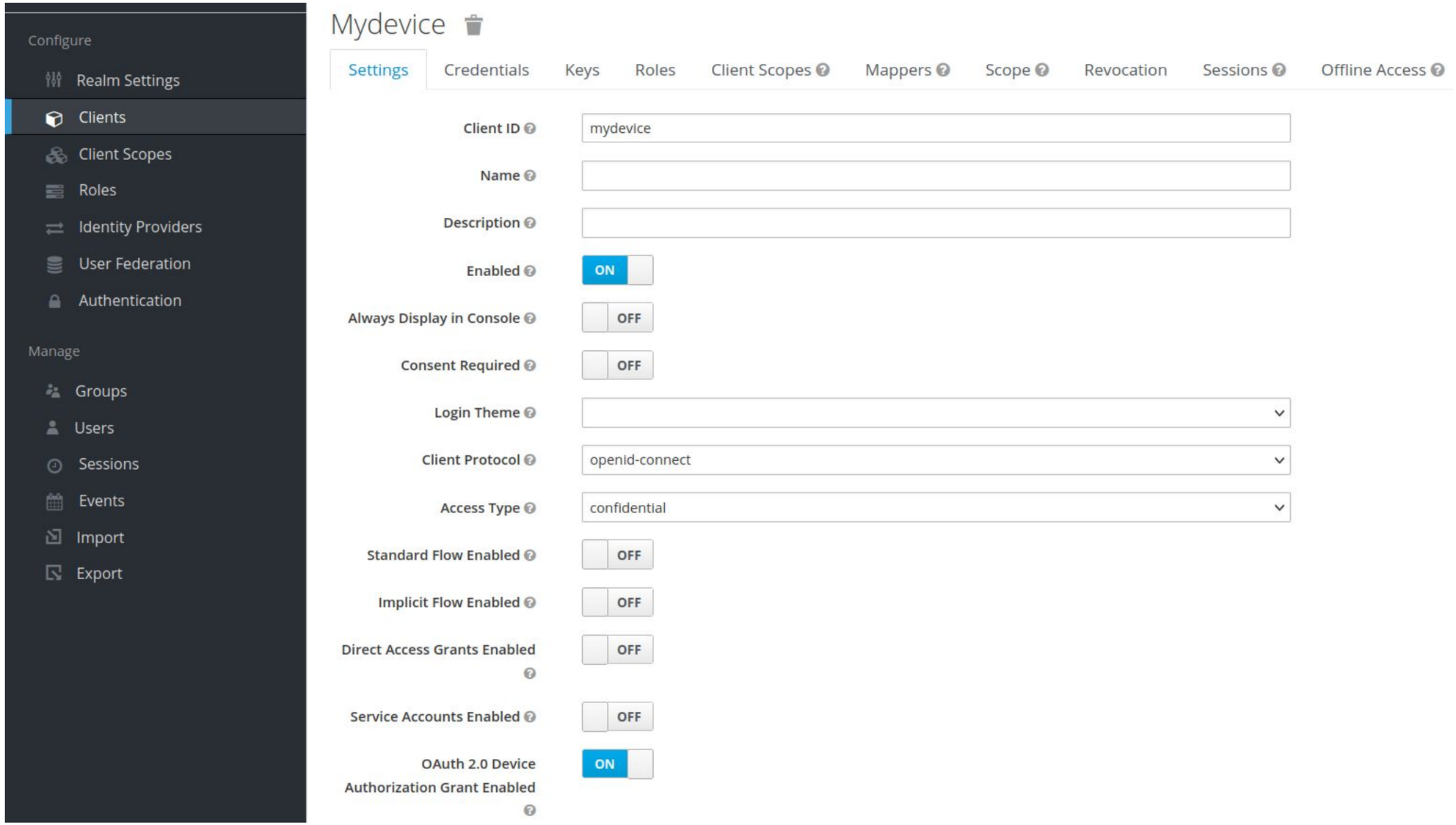
OAuth 2 Device Flow zakłada brak możliwości wprowadzenia danych dostępowych na samym urządzeniu. Według założeń - urządzenie powinno mieć tylko możliwość dostarczenia linku do potwierdzenia i kodu użytkownika.

Flow ma sens tylko i wyłącznie jeżeli urządzenie ma uzyskać dostęp do zasobów użytkownika lub powiązanych z tymże.

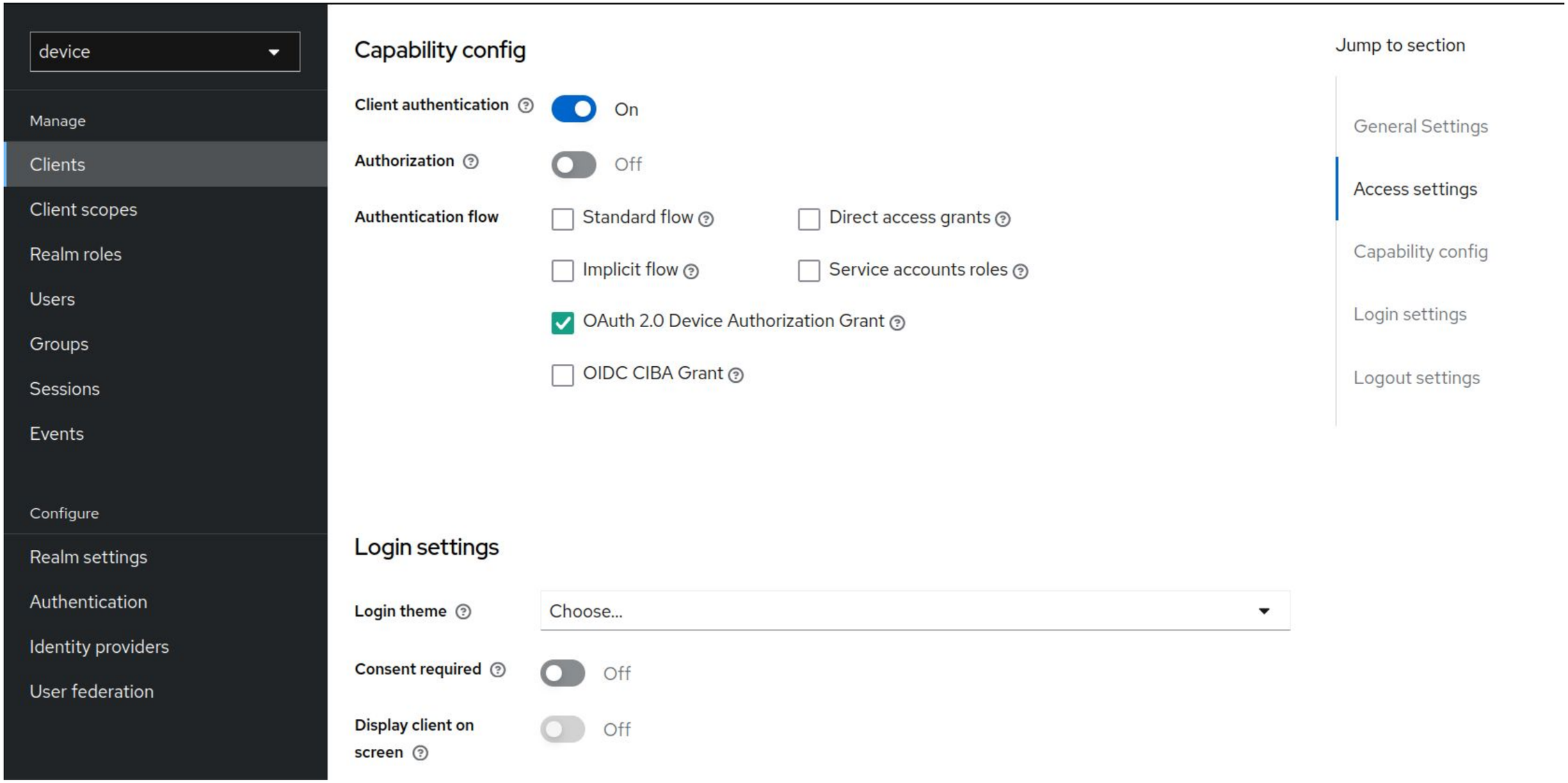

ClientDevice
AuthorizationServer
User
ResourceServer
Obtain device code
Device code, User Code
Auth URI, Lifetime
Present User Code and Auth URI
Check status with device cod
e
Code status or token X


Code status or token X
Provide device code and auth
orize itself
Request resource with token X
Verify token X
Token validity answer
Return user resource

Device flow / Keycloak admin



Device flow / Keycloak admin v2



Przykładowy klient z Device Flow

1. Skorzystamy z biblioteki OAuth 2.0 SDK (com.nimbusds/oauth2-oidc-sdk) https://bitbucket.org/connect2id/oauth-2.0-sdk-with-openid-connect-extensions
2. Kod w Javie - aplikacja uruchamiana w “konsoli”

```java
String uri = "http://id.connectorio.com:8080/realms/device/";
URI deviceAuthURI = new URI( str: uri + "protocol/openid-connect/auth/device");
URI tokenURI = new URI( str: uri + "protocol/openid-connect/token");
ClientID clientID = new ClientID( value: "mydevice");
Secret secret = new Secret( value: "7r1HHmyDJsUt72EY6qODiZ5rvQHa2cz7");
Scope scope = new Scope( ...values: "openid", "profile");

DeviceAuthorizationRequest authorizationRequest = new Builder(new ClientSecretBasic(clientID, secret))
        .scope(scope)
        .endpointURI(deviceAuthURI)
        .build();
HTTPRequest httpRequest = authorizationRequest.toHTTPRequest();
httpRequest.setHeader( name: "Accept", ...values: "application/json");
HTTPResponse httpResponse = httpRequest.send();
```

```
DeviceAuthorizationResponse response = DeviceAuthorizationResponse.parse(httpResponse);
if (!response.indicatesSuccess()) {
  DeviceAuthorizationErrorResponse errorResponse = response.toErrorResponse();
  throw new RuntimeException("Could not finish request. Server returned "
      + " HTTP status code " + errorResponse.getErrorObject().getHTTPStatusCode()
      + " error code " + errorResponse.getErrorObject().getCode()
      + " error description " + errorResponse.getErrorObject().getDescription());
}
```

```java
long now = Clock.systemUTC().millis();
DeviceAuthorizationSuccessResponse successResponse = response.toSuccessResponse();
long maxLifetime = TimeUnit.SECONDS.toMillis(successResponse.getLifetime()) + now;

System.out.println("Received code response");
System.out.println("Your user code: " + successResponse.getUserCode());
System.out.println("Valid to: " + new Date(maxLifetime));
System.out.println("Please enter code in page: " + successResponse.getVerificationURI());
System.out.println("Or confirm linking: " + successResponse.getVerificationURIComplete());

DeviceCodeGrant grant = new DeviceCodeGrant(successResponse.getDeviceCode());
TokenRequest request = new TokenRequest(tokenURI, new ClientSecretBasic(clientID, secret),
    grant, scope, resources: null, customParams: null);
```

```
final HTTPRequest httpRequest = tokenRequest.toHTTPRequest();
HTTPResponse httpResponse = httpRequest.send();

if (httpResponse.getStatusCode() == 400) {
  System.out.println("User did not complete operation with code " + userCode);
  System.out.println(httpResponse.getContent());
} else if (httpResponse.indicatesSuccess()) {
  System.out.println("User completed authentication with " + userCode);
  OIDCTokenResponse tokenResponse = OIDCTokenResponse.parse(httpResponse);
  AccessToken accessToken = tokenResponse.getTokens().getAccessToken();
  RefreshToken refreshToken = tokenResponse.getTokens().getRefreshToken();
  JWT idToken = tokenResponse.getOIDCTokens().getIDToken();
  System.out.println("Access token: " + accessToken);
  System.out.println("Refresh token: " + refreshToken);
  System.out.println("ID token: " + idToken);
  break;
}
sleep();
```

Demo #1, Device Flow

Rzeczy Ciekawe, Identity Broker

Integracja zewnętrznego SSO

Część dużych organizacji boryka się z zagadnieniem migracji ze “starego” rozwiązania SSO do czegoś nowszego.

Coś nowszego zazwyczaj powinno obsługiwać OpenID Connect oraz SAML.

Oczywiście - z obsługą tych dwóch rzeczy pomoże nam Keycloak.
Pytanie - co zrobić ze starym SSO?

Co zrobić ze starym SSO?

Nieodzowna część każdej zmiany to okres przejściowy. Możemy tu zastosować kilka podejść:

- Powrót do usługi katalogowej
  - Masowa zmiana w konfiguracji istniejących aplikacji.
  - Krok wstecz w porównaniu do stanu obecnego.
  - Często niemożliwe do wykonania.
- Funkcjonowanie nowego i starego SSO równocześnie
  - Stopniowe przenoszenie konfiguracji klientów.
  - Konieczna migracja/synchronizacja użytkowników.

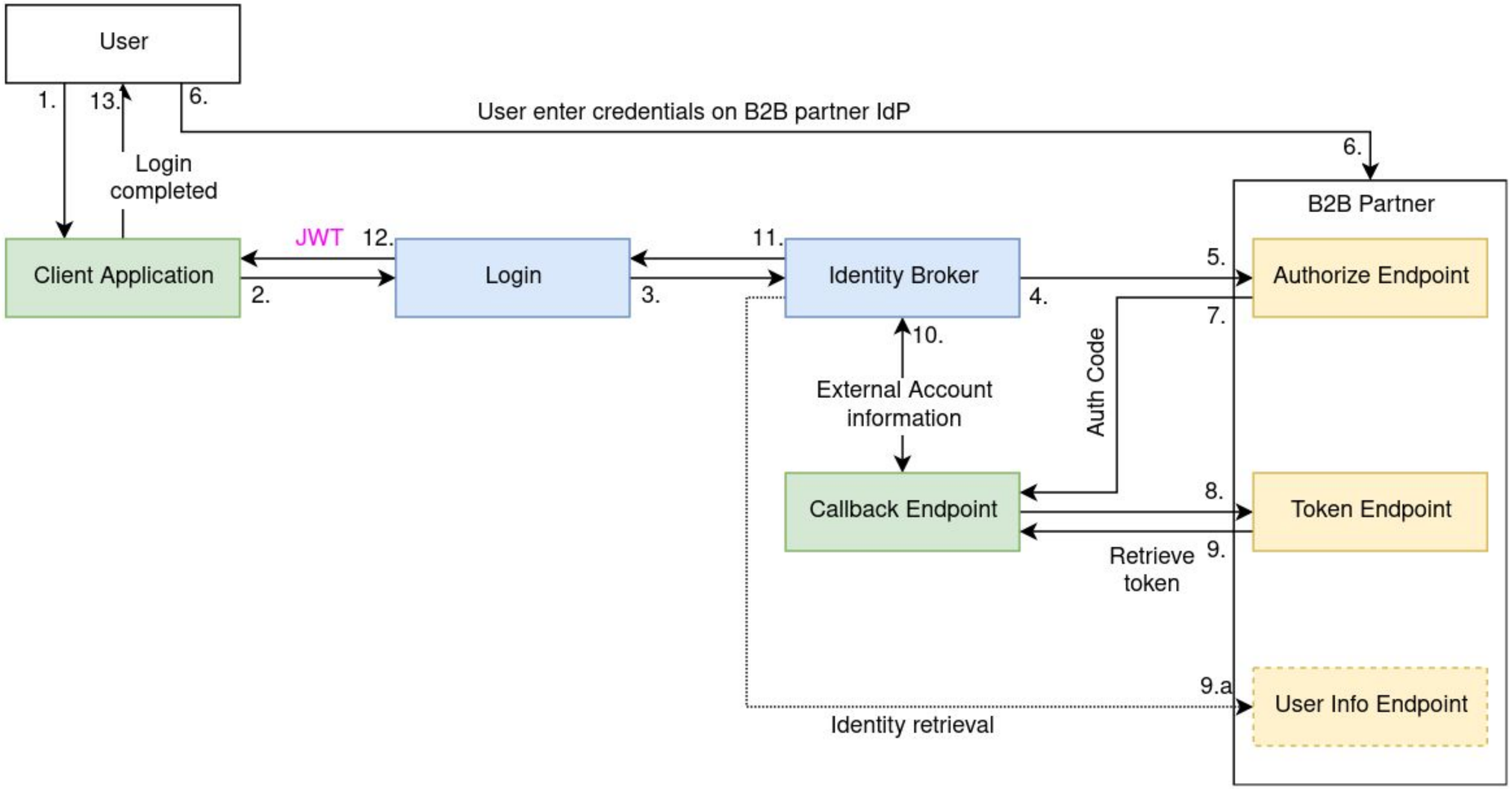
cd. Co zrobić ze starym SSO?

Rozwiązanie z Keycloak:

- Integracja starego SSO jako dostawcy tożsamości Keycloak:
  - Możliwość mapowania użytkowników ze “starego” do “nowego”.
  - Bieżące uzupełnienie bazy Keycloak.
- Integracja bazy starego SSO.

- W przypadku aplikacji, które nie mogą zostać zmigrowane:
  - Implementacja protokołu starego SSO.

Identity broker



Najważniejsze punkty

Identity Broker działa z OIDC / OAuth 2 / OAuth 1 / SAML.

Zewnętrzny system może przeprowadzać autoryzację użytkownika "po swojemu" - przez przeglądarkę lub np. aplikację mobilną i "push" do Keycloak.

Z punktu widzenia kodu Keycloak wymaga implementacji "pierwszego kroku" oraz "powrotu" z informacją o użytkowniku.

Przykład Discourse

- Popularne forum internetowe.
- Integruje się z usługami / dostawcami tożsamości
  - OpenID Connect
  - SAML
- Ma również własny protokół SSO tzw. Discourse Connect
  - Pozwala na płynną rejestrację użytkowników w Discourse
  - .. ale również na autentykację za pośrednictwem Discourse

Protokół Discourse Connect

Żądanie wymaga 2 parametrów:

- sso
  - nonce - zwracany do wywołującego
  - return_sso_url - adres zwrotny, weryfikowany w konfiguracji Discourse
- sig
  - HmacSHA256 (sso), kluczem jest "Client Secret"

https://keycloak.discourse.group

KEYCLOAK

all categories ▸ | all tags ▸ | Latest | Top | Categories

Topic

Authorization server with 3rd party authentication
Getting advice authentication

Request URI did not match SAML request destination

Keycloak clustering in 21.1.1 vs 20.x.x
Configuring the server clustering

Use OTP validation using request

cd. Protokół Discourse Connect

Odpowiedź zawiera kilka parametrów

- sso
  - nonce
  - external_id (Discourse user id)
  - username
  - email
  - + opcjonalnie dodatkowe flagi
- sig - sygnatura sso do zweryfikowania z client secret

```java
public class DiscourseIdentityProvider extends AbstractIdentityProvider<DiscourseIdentityProviderConfig>
    implements SocialIdentityProvider<DiscourseIdentityProviderConfig> {

    1 usage   Łukasz Dywicki
    public DiscourseIdentityProvider(KeycloakSession session, DiscourseIdentityProviderConfig config) {
        super(session, config);
    }

    Łukasz Dywicki
    @Override
    public Object callback(RealmModel realm, AuthenticationCallback callback, EventBuilder event) {
        return new DiscourseIdentityProvider.Endpoint(session, realm, callback, event);
    }
```

```java
public Response performLogin(AuthenticationRequest request) {
  try {
    String nonce = request.getState().getEncoded();
    String payload = "nonce=" + nonce + "&return_sso_url=" + request.getRedirectUri();
    String base64payload = new String(Base64.getEncoder().encode(payload.getBytes()));

    String hexSignature = hmac(base64payload, getConfig().getSsoSecret());
    String encodedPayload = URLEncoder.encode(base64payload, StandardCharsets.UTF_8);

    String address = getConfig().getDiscourseAddress();
    address += "/session/sso_provider?";
    address += "sso=" + encodedPayload;
    address += "&sig=" + hexSignature;
    URI authenticationUrl = URI.create(address);

    return Response.seeOther(authenticationUrl).build();
  } catch (Exception e) {
    throw new IdentityBrokerException("Could send authentication request to Discourse.", e);
  }
}
```

```java
@GET
public Response authResponse(@QueryParam("sso") String sso, @QueryParam("sig") String signature)
    throws Exception {
    Map<String, String> data = parse(getConfig().getSsoSecret(), sso, signature);
    String state = data.get("nonce");

    if (state == null) {
        return callback.error("Could not correlate request");
    }
    AuthenticationSessionModel authSession = callback.getAndVerifyAuthenticationSession(state);
    if (authSession == null) {
        return callback.error("Could not identify request");
    }
```

```
try {
  BrokeredIdentityContext identity = new BrokeredIdentityContext(data.get("external_id"));
  identity.setAuthenticationSession(authSession);
  identity.setIdp(DiscourseIdentityProvider.this);
  identity.setUsername(data.get("username"));
  identity.setEmail(data.get("email"));
  identity.setUserAttribute( attributeName: "moderator", data.get("moderator"));
  identity.setUserAttribute( attributeName: "admin", data.get("admin"));
  identity.setIdpConfig(getConfig());
  return callback.authenticated(identity);
} catch (Exception e) {
  return callback.error("Error while logging in via identity provider");
}
```

Demo #2, Identity Broker

Wymagane rzeczy
Ocena prezentacji
Pytania i odpowiedzi

Kod źródłowy:

http://github.com
/Code-House
/keycloak-deviceflow
/keycloak-discourse