



Securing Microservices with OpenID Connect and Spring Security 5

Workshop

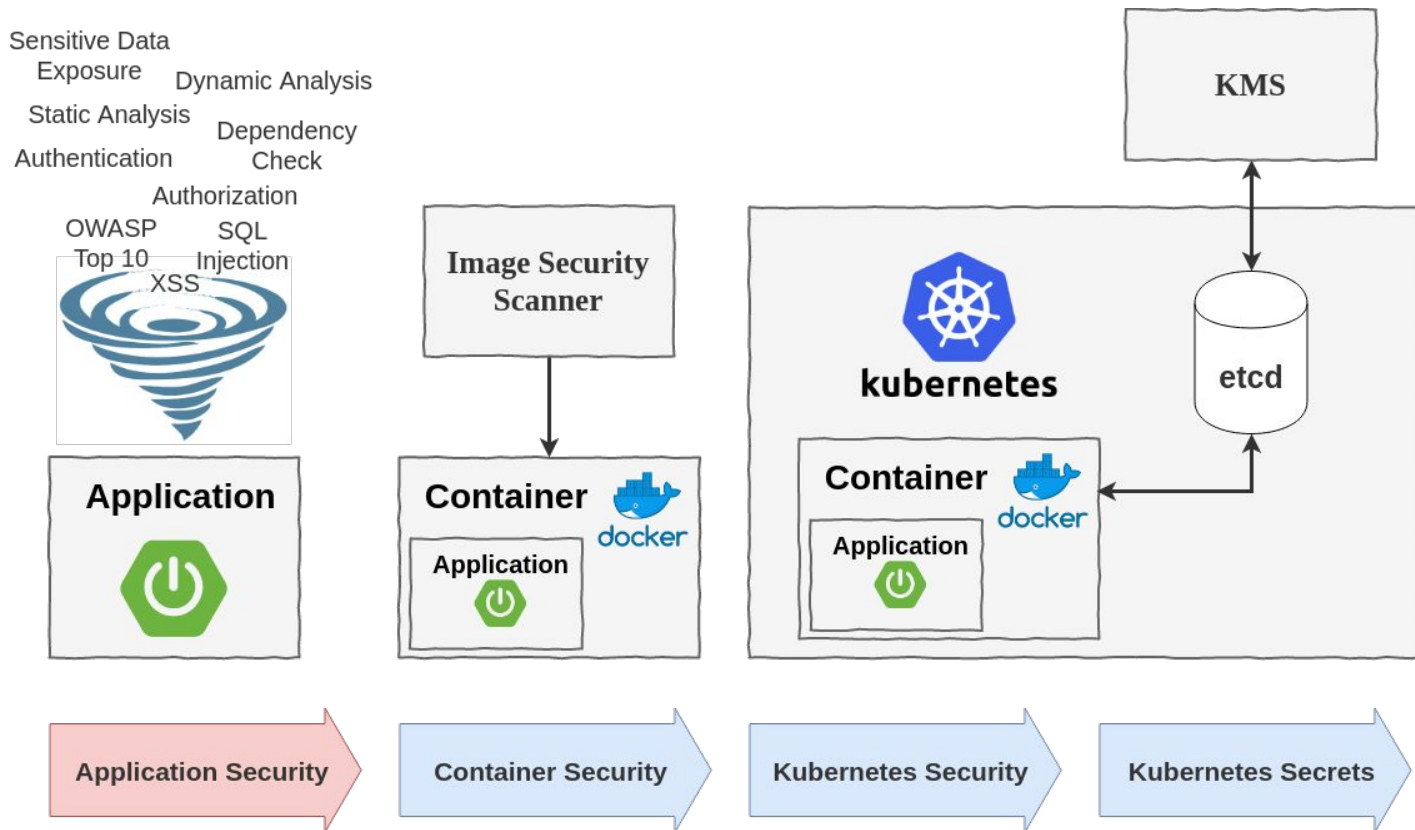
Andreas Falk



Agenda

1. Intro:
 - OAuth 2.0 & OpenID Connect
 - Intro-Lab: OAuth2 in Action
2. Hands-On Part
 - Resource Server
 - Web Client (Auth Code Flow)
3. What's new in Spring Security

Auth & Authz is only ONE part of Security!



Authentication Mechanisms

- Basic Authentication / Digest Access Authentication
- Form-based Authentication (i.e. using Session Cookies)
- Client-Certificates (MTLS)
- Kerberos Tickets
- SAML Assertion Tokens
- JSON Web Tokens
- Proprietary mechanisms like API-Tokens, Siteminder etc.



OAuth 2.0 & OpenID Connect

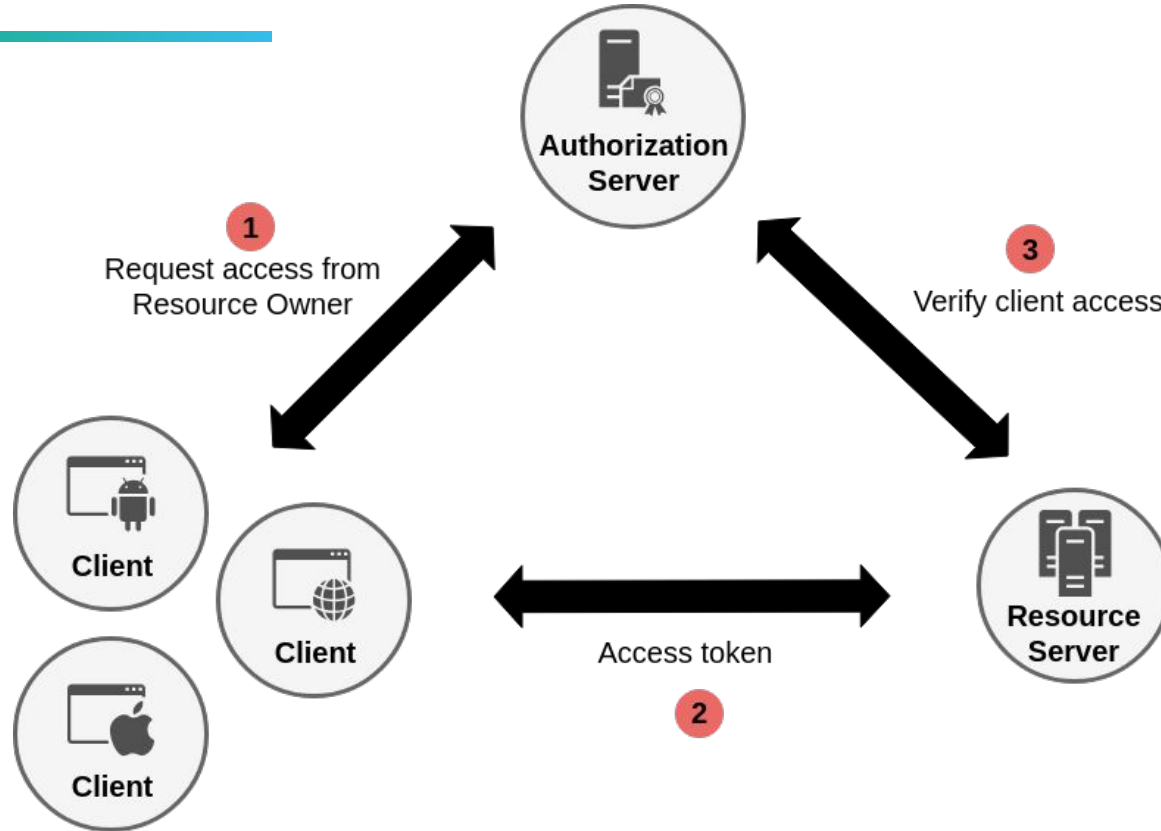


Introduction

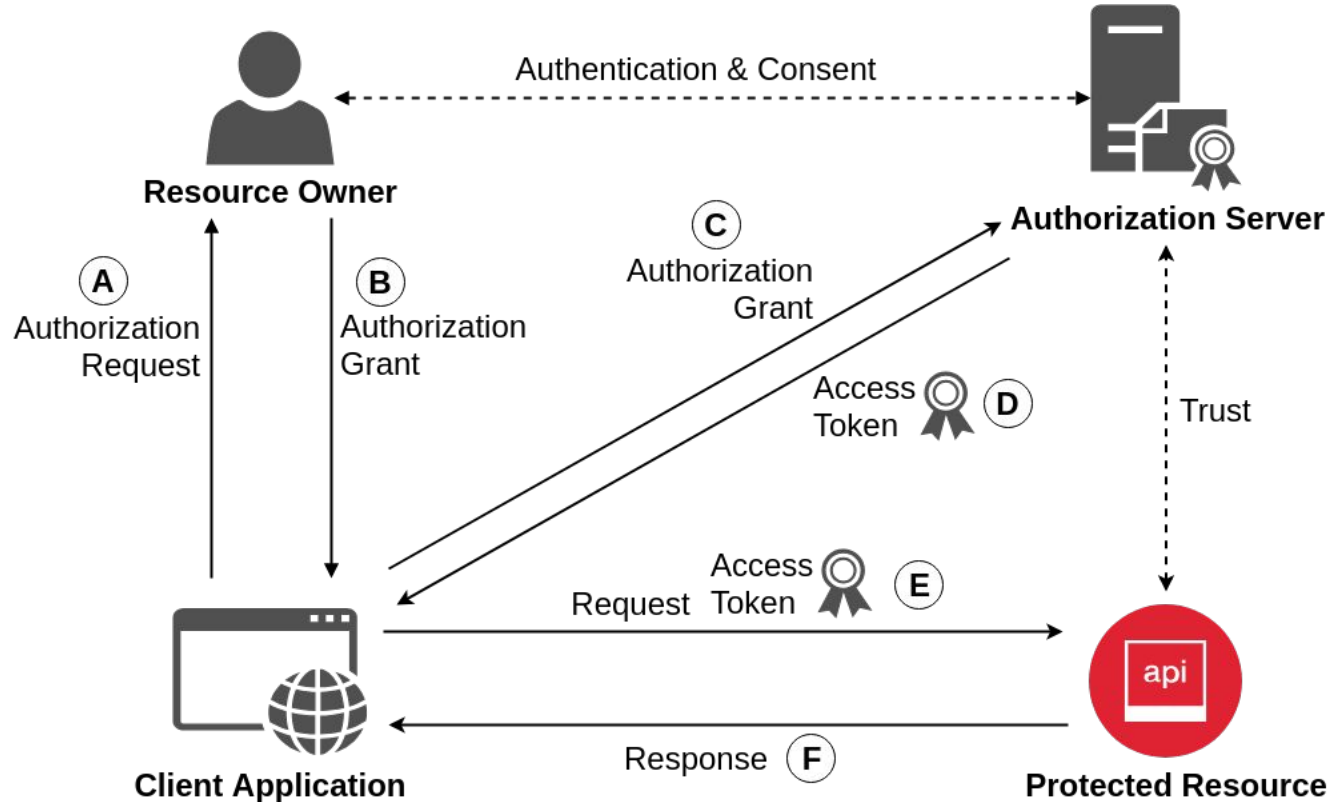
OAuth 2.0 is an authorization delegation framework



OAuth 2.0 Roles



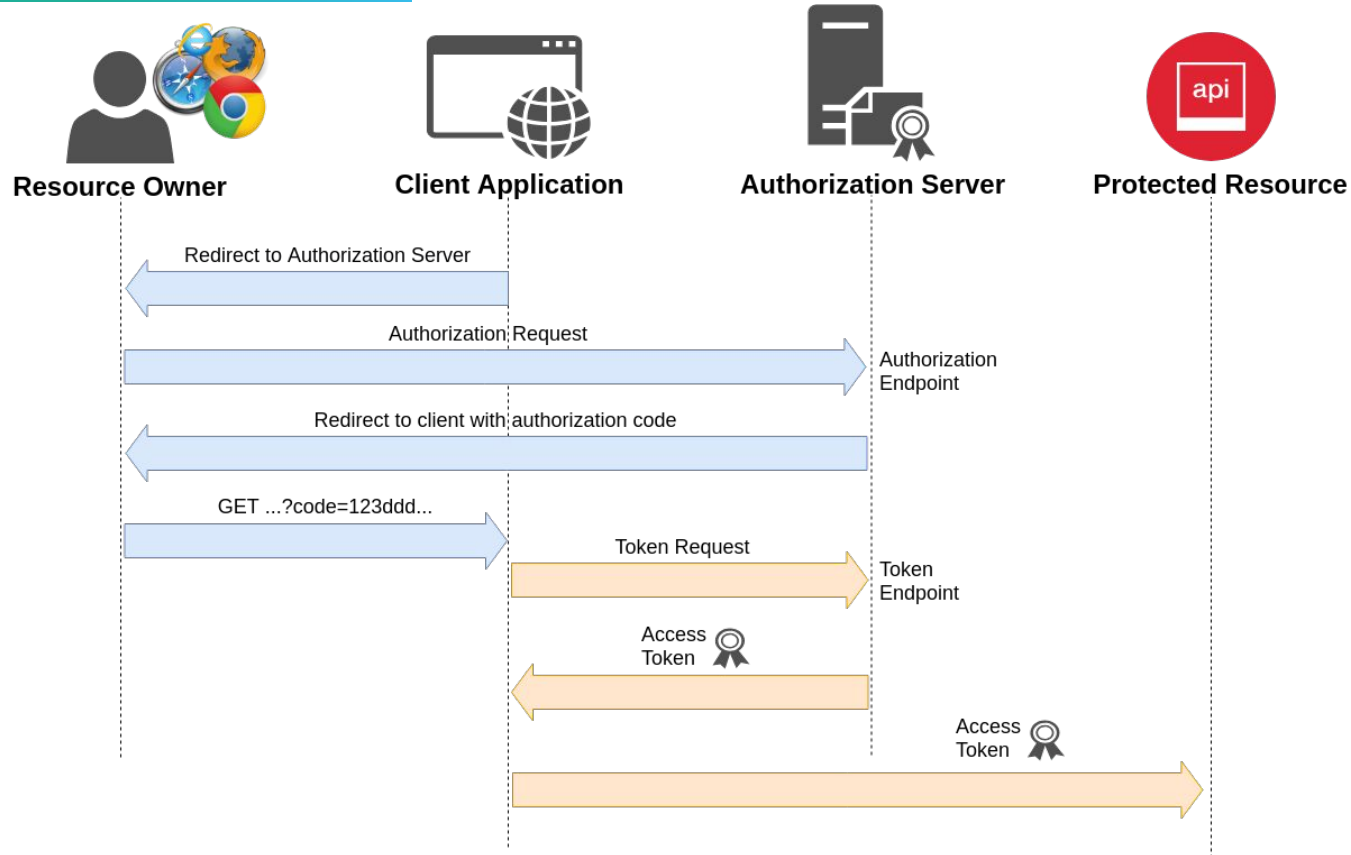
OAuth 2.0 Protocol Flow



OAuth 2.0 Grant Flows

Client Type	Flow	Refresh Tokens
Confidential	Authorization Code	X
Public (Native)	Authorization Code (PKCE)	X
Public (SPA)	Implicit	--
Trusted	RO Password Creds	X
No Resource Owner	Client Credentials	--

OAuth 2.0 Authorization Code Grant Flow





Practice Time

Intro-Lab: Authorization Code Grant Flow in Action

OAuth 2.0 is NOT an Authentication Protocol!



Jim Manico

@manicode

Follow



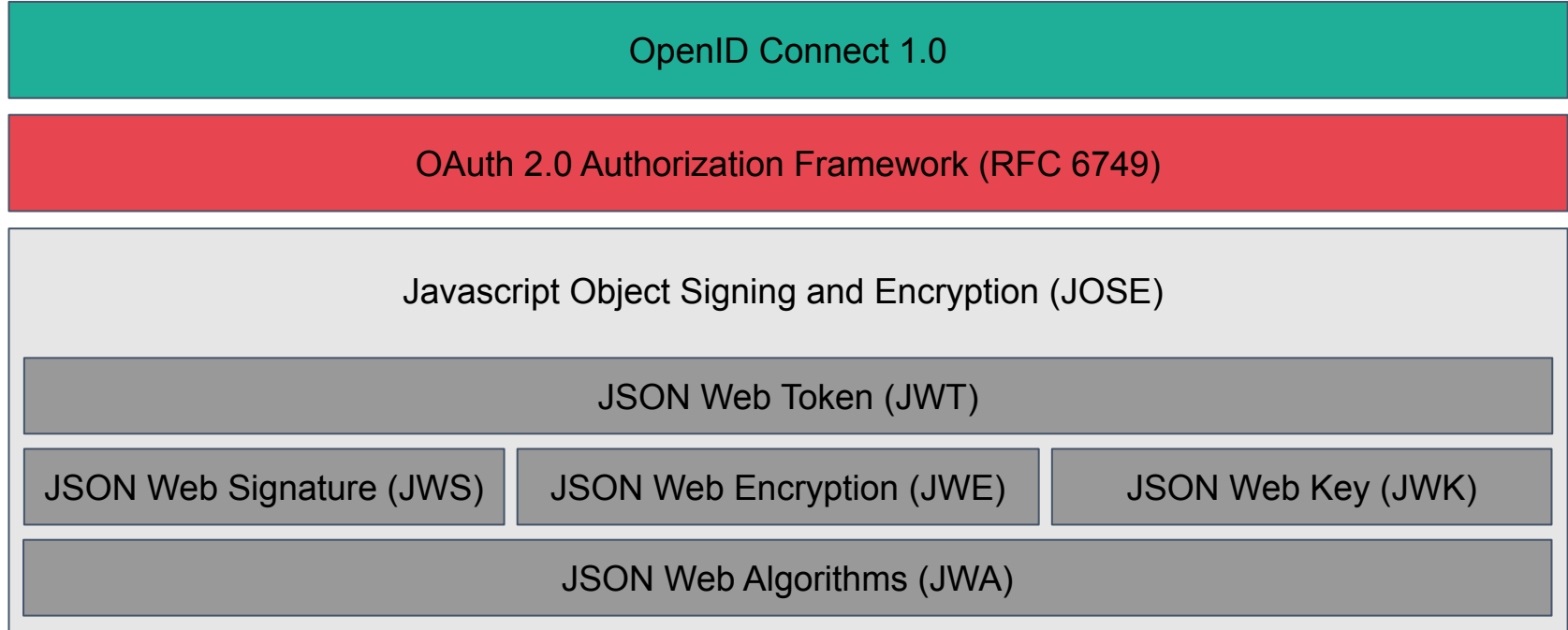
Repeat after me: OAuth 2.0 is NOT AN
AUTHENTICATION PROTOCOL.

oauth.net/articles/authenticate ...

12:22 PM - 2 Feb 2017 from [Kapaa, HI](#)

[OAuth 2.0 is not an authentication protocol](https://oauth.net/articles/authenticate)

OpenID Connect 1.0 Standards Layer



JSON Web Algorithms (JWA)

- Cryptographic algorithms and identifiers for JWS, JWE, and JWK specifications
- Digital Signatures and MACs
- Algorithms for Key Management
- Algorithms for Content Encryption
- Algorithms for Keys

<https://tools.ietf.org/html/rfc7518>

JSON Web Key (JWK)

```
{
  "keys": [
    {
      "kty": "EC",
      "crv": "P-256",
      "x": "MKBCTNIcKUSDii11ySs3526iDZ8AiTo7Tu6KPAqv7D4",
      "y": "4EtI6SRW2YiLUrN5vfvVHuhp7x8PxltmWWIbbM4IFyM",
      "use": "enc",
      "kid": "1"
    },
    {
      "kty": "RSA",
      "n": "0vx7agoebGcQSuuPiLJXZptN9nn...",
      "e": "AQAB",
      "alg": "RS256",
      "kid": "2011-04-29"
    }
  ]
}
```

<https://tools.ietf.org/html/rfc7517>

JSON Web Signature (JWS)

- JSON Web Signature (JWS) represents content secured with digital signatures or Message Authentication Codes (MACs) using JSON-based data structures
- A document using JWS can answer two questions about the JSON payload:
 - Has the JSON object been altered after creation?
 - Who created this JSON object?

<https://tools.ietf.org/html/rfc7515>

JSON Web Encryption (JWE)

- Data structure representing an encrypted and integrity-protected message
- As of July 2019 only identity server of PingIdentity supports JWE
- NOT supported by Spring Security 5.x (See github issue 4435) !

<https://tools.ietf.org/html/rfc7516>

<https://github.com/spring-projects/spring-security/issues/4435>

JSON Web Token (JWT)

- JSON Web Tokens consist of three parts separated by dots ("."), which are:
 - Header
 - Payload
 - Signature
- Each part is Base64Url encoded
- Signature supports symmetric or asymmetric algorithms (e.g. HMAC or RSA)
- Signature = HMACSHA256(
base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

<https://tools.ietf.org/html/rfc7519>

<https://tools.ietf.org/html/draft-ietf-oauth-jwt-bcp>

<https://tools.ietf.org/html/draft-ietf-oauth-proof-of-possession>

JSON Web Token (JWT) - Decoded Form

```
{  
  "alg": "RS256",  
  "typ": "JWT",  
  "kid": "lp_FcMZ8D7U6EEUCiZyWAF21NcwjX_ddwJ5a3eCPMwQ"  
}
```

HEADER

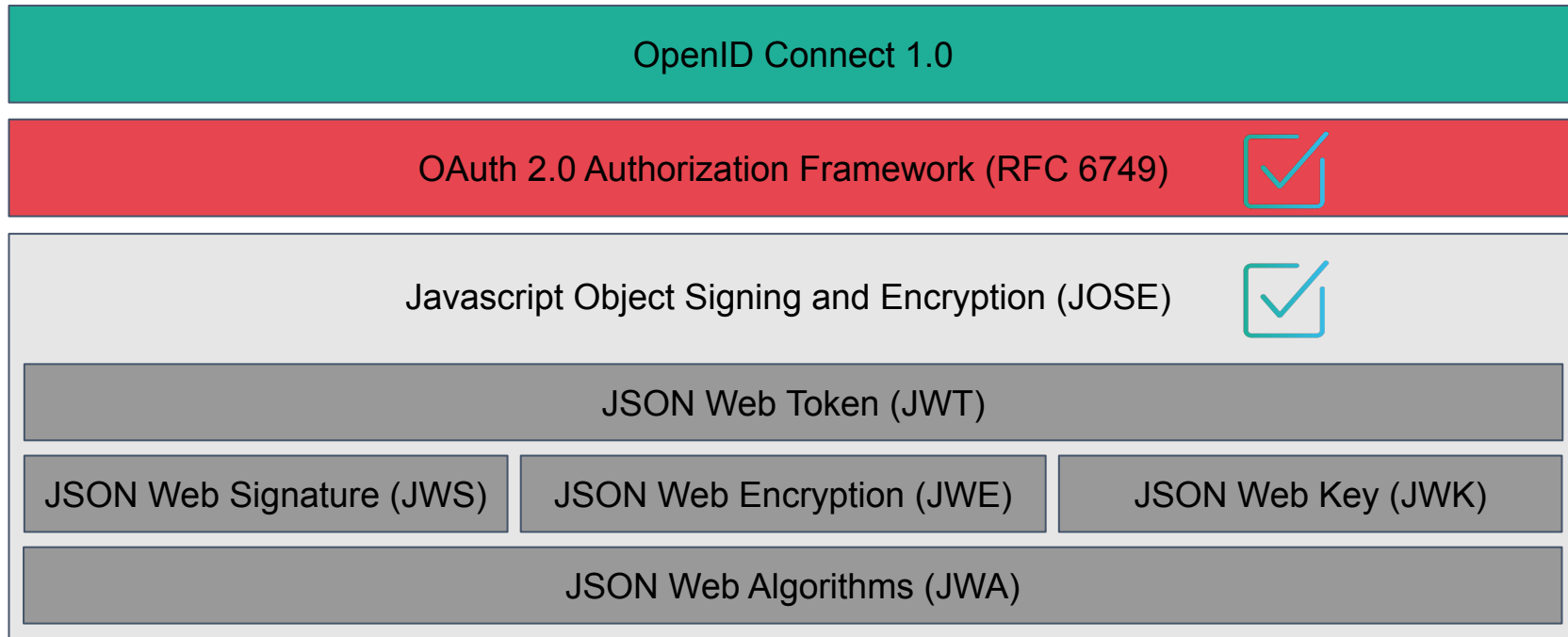
```
{  
  "exp": 1571745342,  
  "iat": 1571745042,  
  "iss": "http://localhost:8080/auth/realms/workshop",  
  "aud": ["library-service","account"],  
  "sub": "08d3bcaa-5ffd-4b8d-909e-bb567881384b"  
}
```

PAYLOAD

20



OpenID Connect 1.0 Standards Layer



OpenID Connect 1.0 (OIDC)

- Based on OAuth 2.0
- Additions:
 - ID Token (JWT format is mandatory)
 - User Info Endpoint (Mandatory)
 - Hybrid Grant Flow (Mandatory)
 - OpenID Provider Configuration Information (Discovery, Optional)

https://openid.net/specs/openid-connect-core-1_0.html

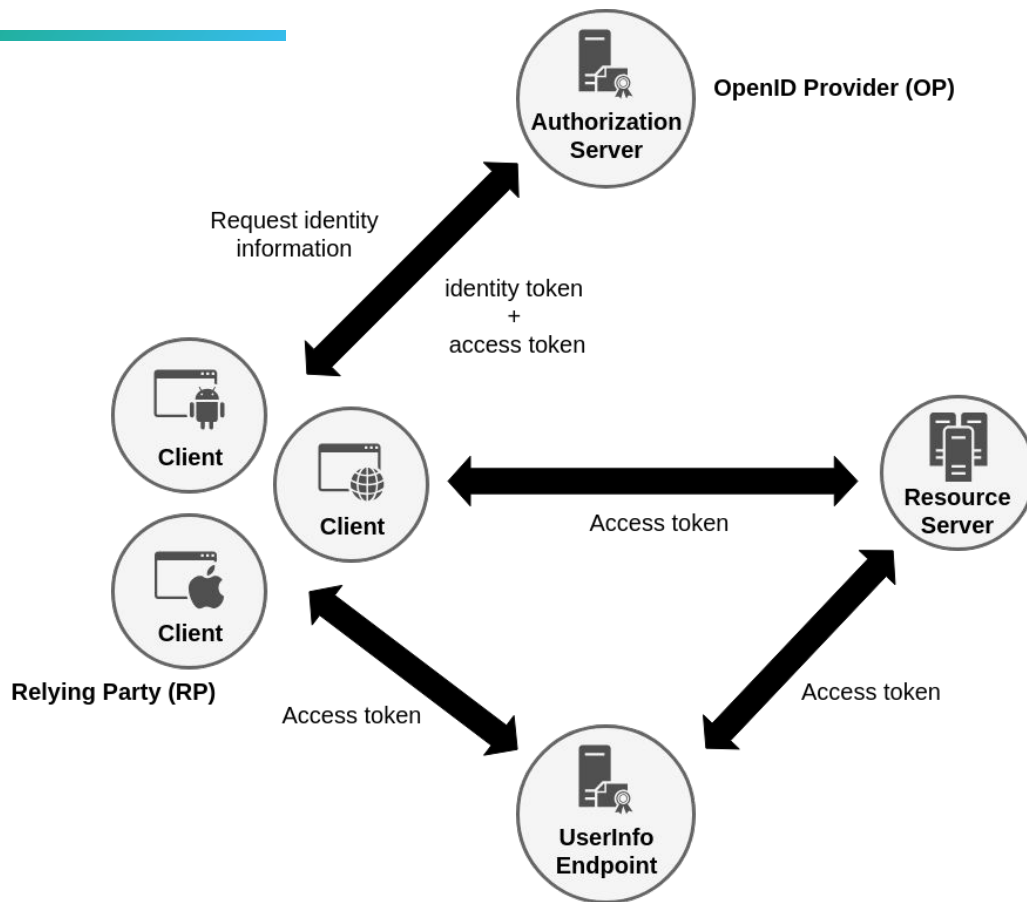
https://openid.net/specs/openid-connect-registration-1_0.html

https://openid.net/specs/openid-connect-discovery-1_0.html

OpenID Connect 1.0 Claims

Claim	Required	Description
iss	<input checked="" type="checkbox"/>	Issuer Identifier
sub	<input checked="" type="checkbox"/>	Unique Subject Identifier
aud	<input checked="" type="checkbox"/>	Target audience(s) of an ID Token
exp	<input checked="" type="checkbox"/>	Expiration time
iat	<input checked="" type="checkbox"/>	Time at which the JWT was issued
auth_time	<input checked="" type="checkbox"/>	Time of End-User authentication
nonce		Used to associate a client with an ID Token

OpenID Connect 1.0 Roles



OpenID Connect 1.0 User Info Endpoint

```
GET /userinfo HTTP/1.1  
Host: identityserver.example.com  
Authorization: Bearer SLAV32hkKG
```

```
HTTP/1.1 200 OK  
Content-Type: application/json
```









```
{  
  "sub": "248289761001",  
  "name": "Jane Doe",  
  "given_name": "Jane",  
  "family_name": "Doe",  
  "preferred_username": "j.doe",  
  "email": "janedoe@example.com",  
  "picture": "http://example.com/janedoe/me.jpg"  
}
```

OpenID Connect 1.0 Discovery

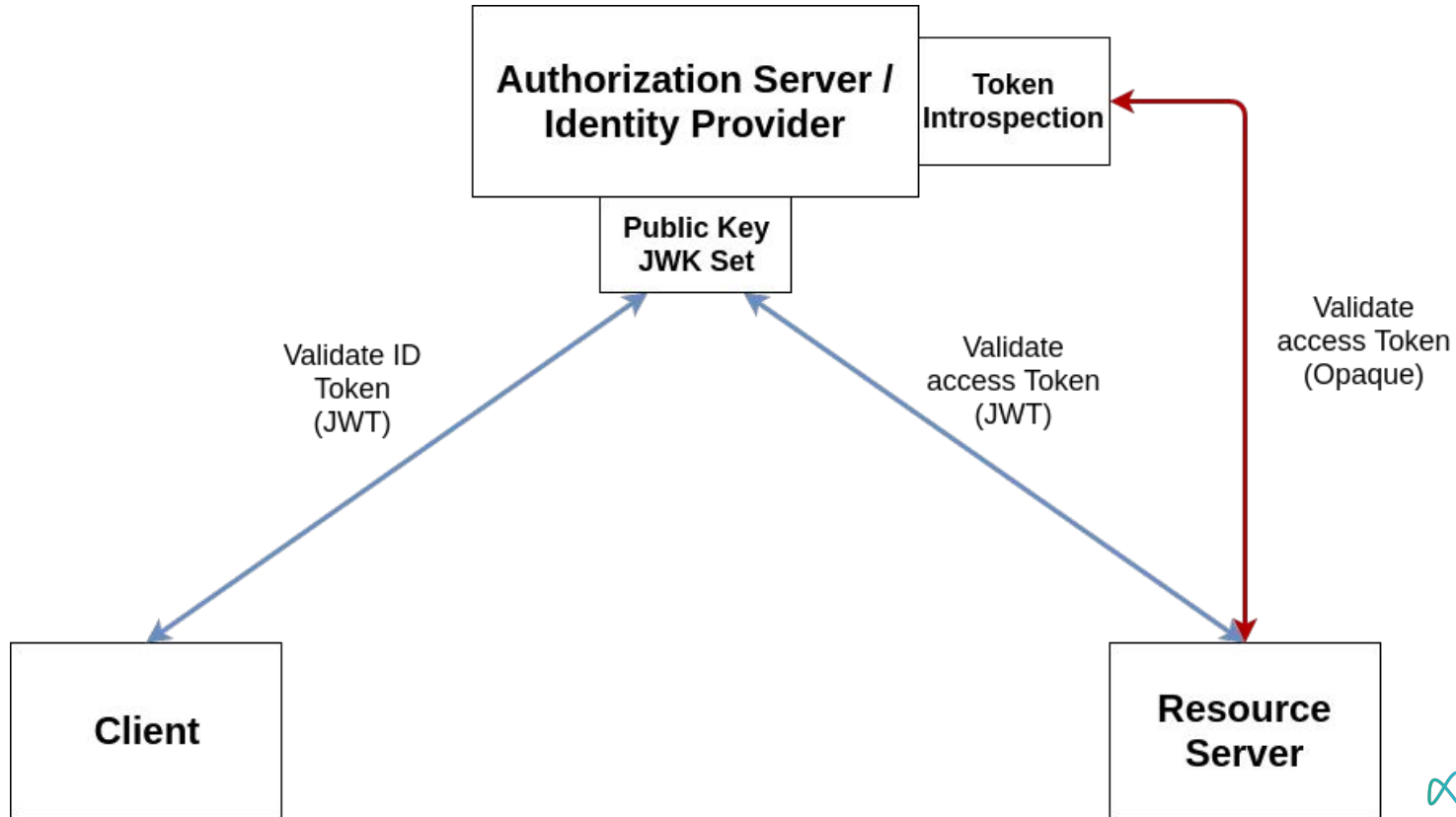
```
{  
  "authorization_endpoint": "https://idp.example.com/auth",  
  "grant_types_supported": [  
    "authorization_code",  
    "implicit",  
    "refresh_token"  
  ],  
  "issuer": "https://idp.example.com",  
  "jwks_uri": "https://idp.example.com/keys",  
  "token_endpoint": "https://idp.example.com/token",  
  "userinfo_endpoint": "https://idp.example.com/userinfo",  
  ...  
}
```

<https://example.com/.well-known/openid-configuration>

OpenID Connect 1.0: Access Token Types

JWT Token (Self-contained)	Opaque Token (Reference)
 Offline-Validation (Signature/Expiration)	 Validation call to introspection-endpoint
 Contains all required information	 Additional call to get required information
 Protocol agnostic	 Bound to Http
 Cannot be revoked	 May be revoked

Token Validation



OAuth 2.0 Access Token JWT Profile (Draft)

- Required claims: iss, exp, aud, sub, client_id
- Consider privacy restrictions for identity claims
- Authorization claims according to SCIM Core (RFC7643):
 - Groups
 - Entitlements
 - Roles

System for Cross-domain Identity Management (SCIM)
JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens

OpenID Connect Identity Providers

- RedHat/JBoss Keycloak (<https://www.keycloak.org>)
- Auth0 (<https://auth0.com>)
- Okta (<https://www.okta.com>)
- ForgeRock (<https://www.forgerock.com/platform/identity-management>)
- CloudFoundry UAA (<https://github.com/cloudfoundry/uaa>)
- PingFederate
(<https://www.pingidentity.com/en/platform/single-sign-on/sso-overview.html>)
- Azure Active Directory
(<https://azure.microsoft.com/en-us/services/active-directory>)
- ...

See: <https://openid.net/developers/certified/#OPServices>

JBoss Keycloak as OIDC Identity Provider

- Open Source identity and access management product by RedHat/JBoss
- Currently based on JBoss Wildfly Application Server
- Implements OpenID Connect 1.0, OAuth 2.0 and SAML 2.0
- Provides a centralized user management
- Keycloak-X will move away from WildFly to Quarkus (<https://quarkus.io>)

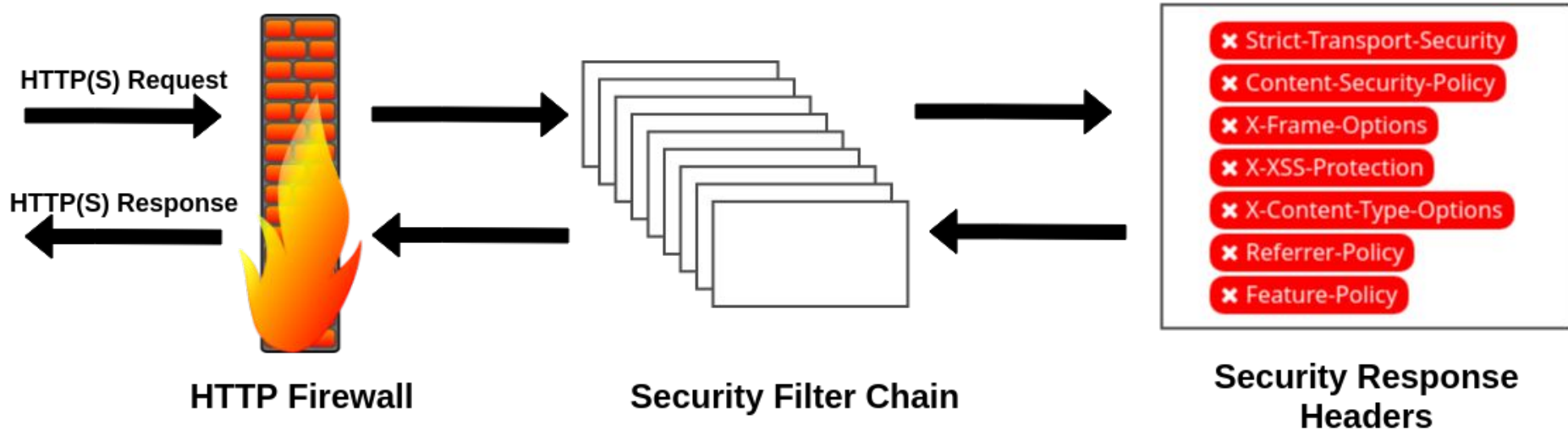
<https://www.keycloak.org/>

<https://www.keycloak.org/2019/10/keycloak-x.html>

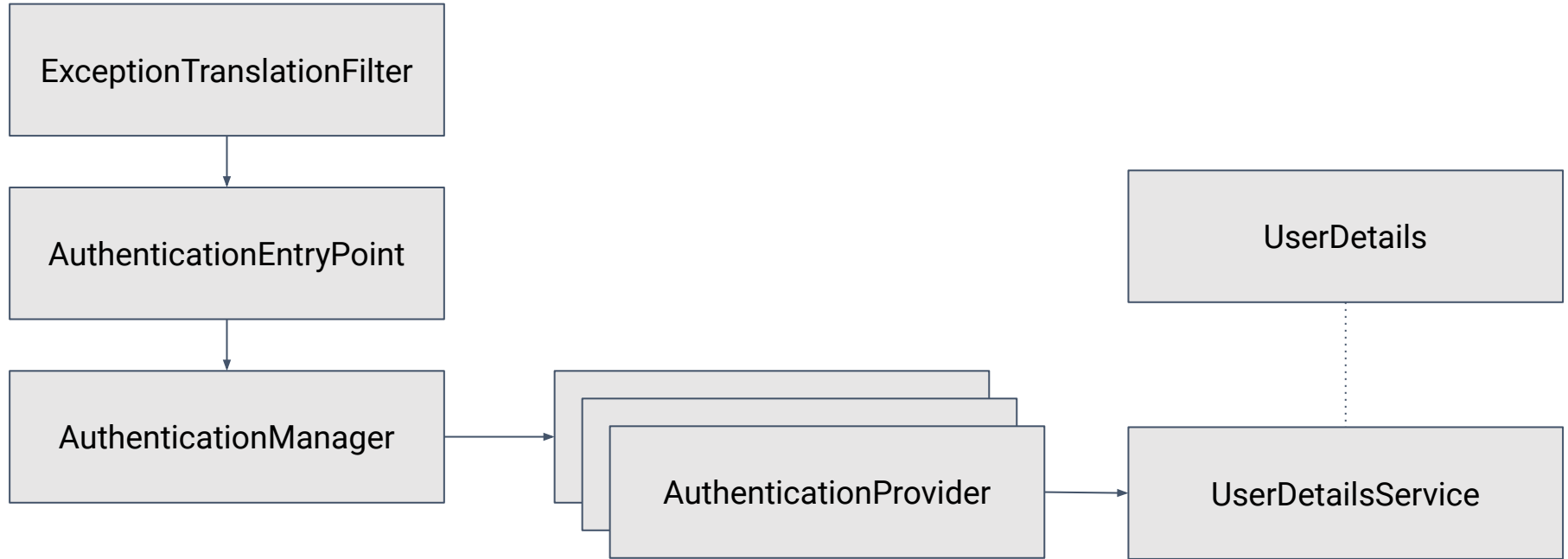


Spring Security 5 Basics

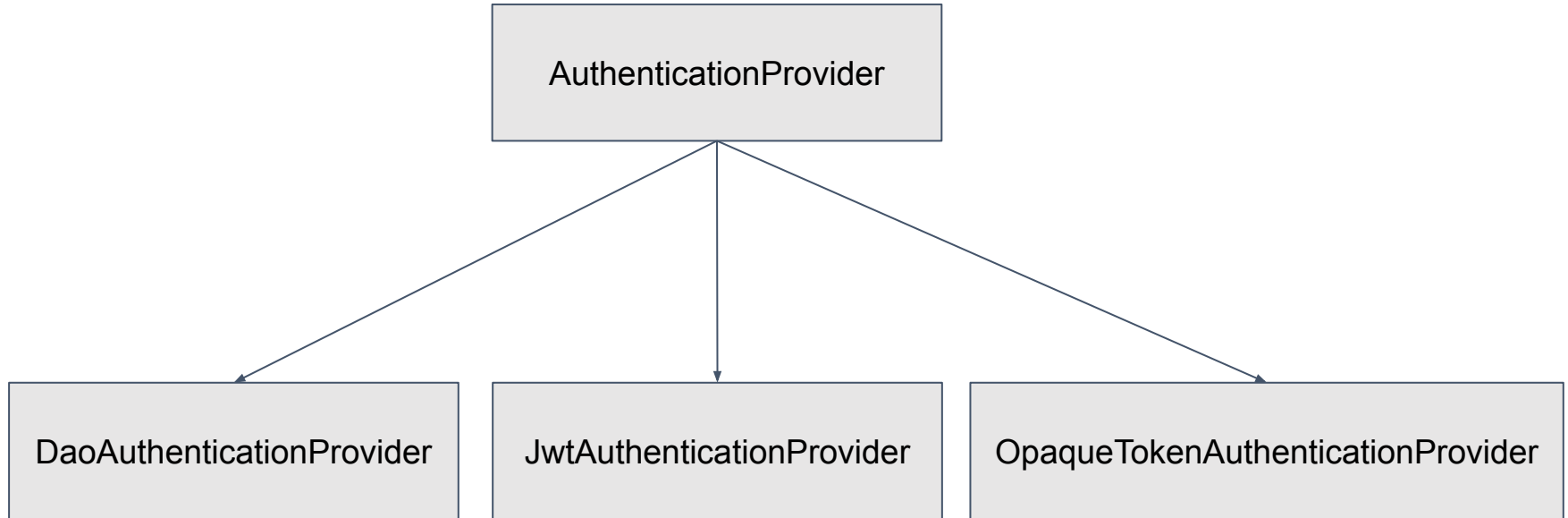
Spring Security High Level View



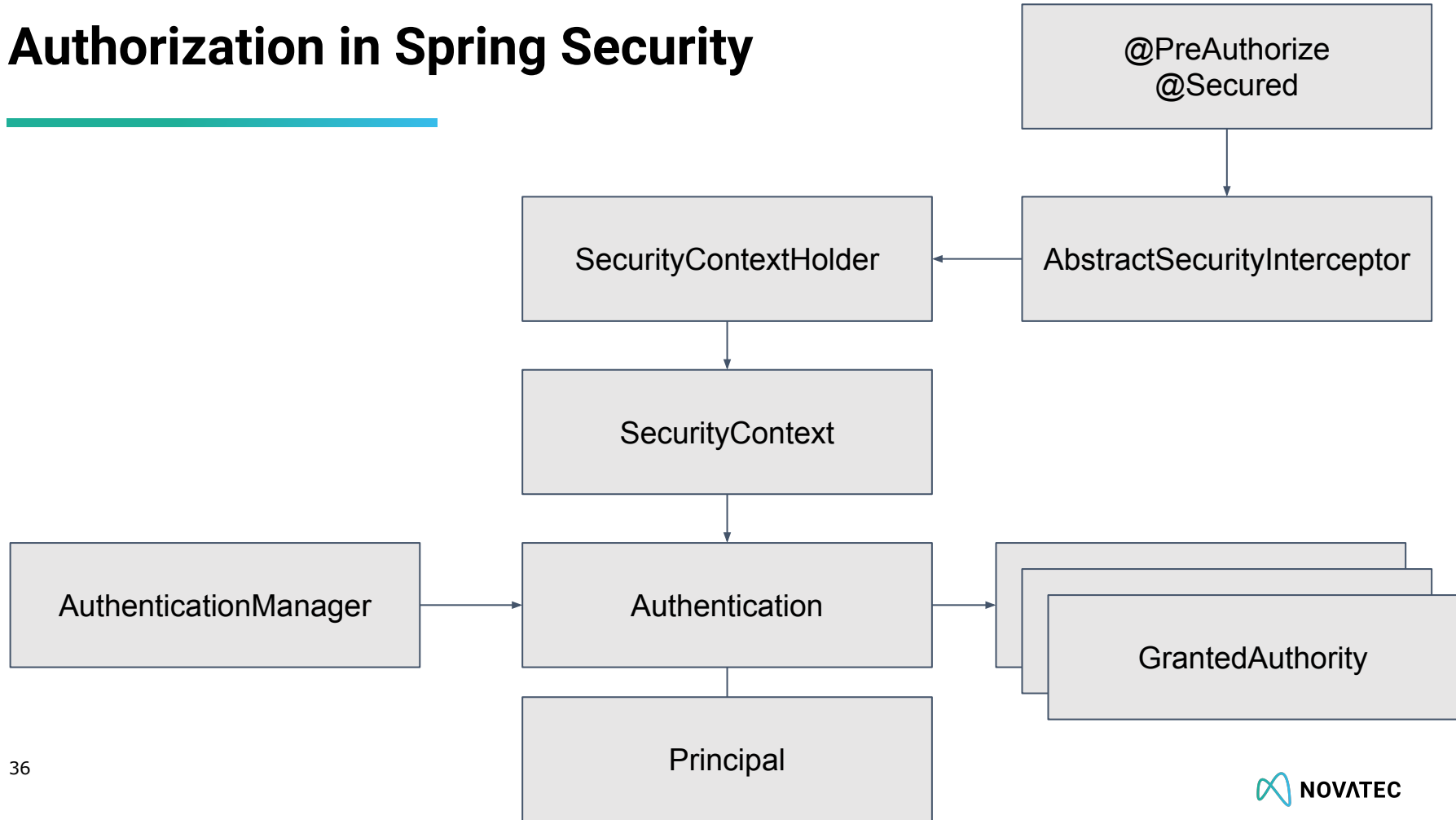
Authentication in Spring Security



Authentication in Spring Security



Authorization in Spring Security



“Legacy” Spring Security 4.x OAuth2 Technology Stack

spring-security-oauth2-autoconfigure

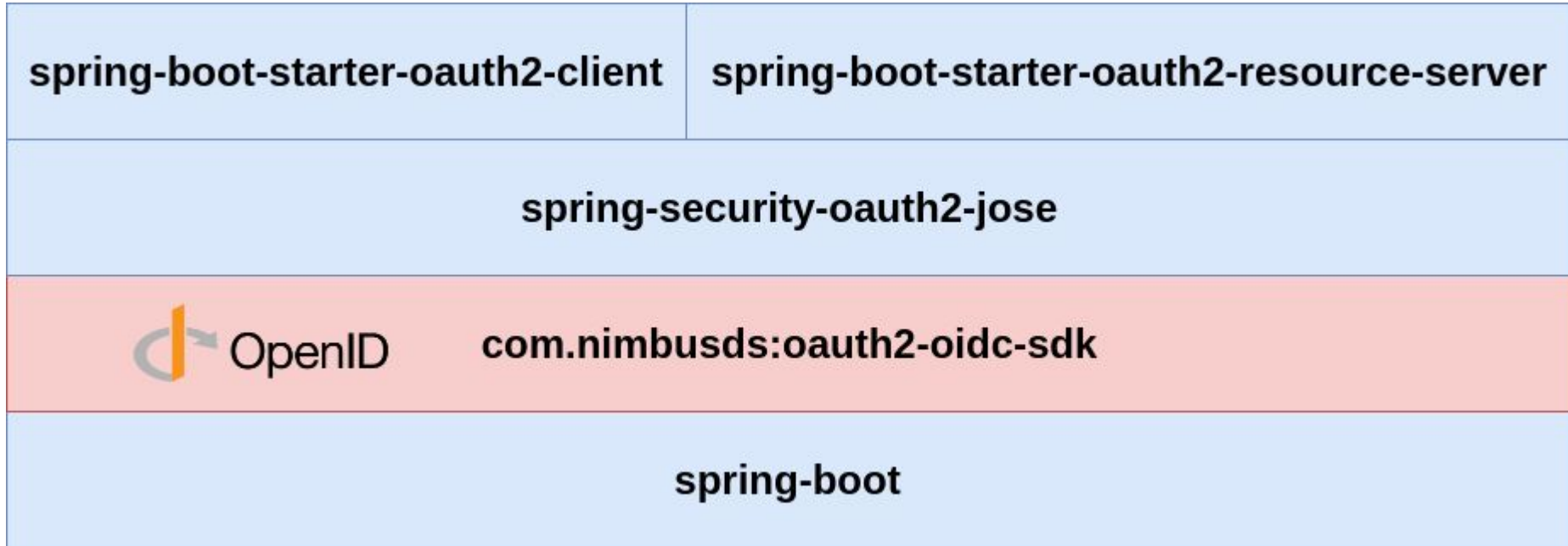
spring-security-oauth2

spring-security-jwt

spring-boot-starter-security

spring-boot

New Spring Security 5.x OIDC Technology Stack



<https://connect2id.com/products/nimbus-oauth-openid-connect-sdk>



OpenID Connect on the Server side (The Resource Server)



Authentication in a single Resource server



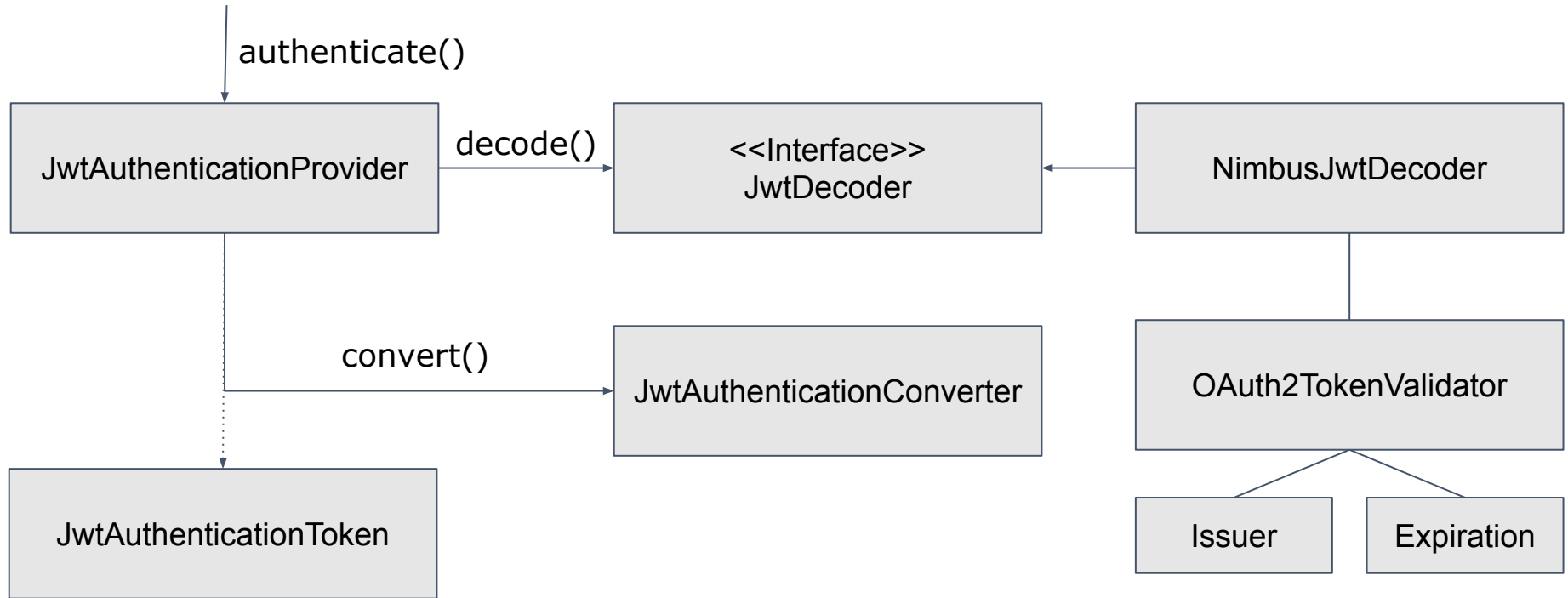
GET / HTTP/1.1

Host: localhost:8080

Authorization: Bearer

eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1N...

JWT Authentication in Spring Security 5





Practice Time

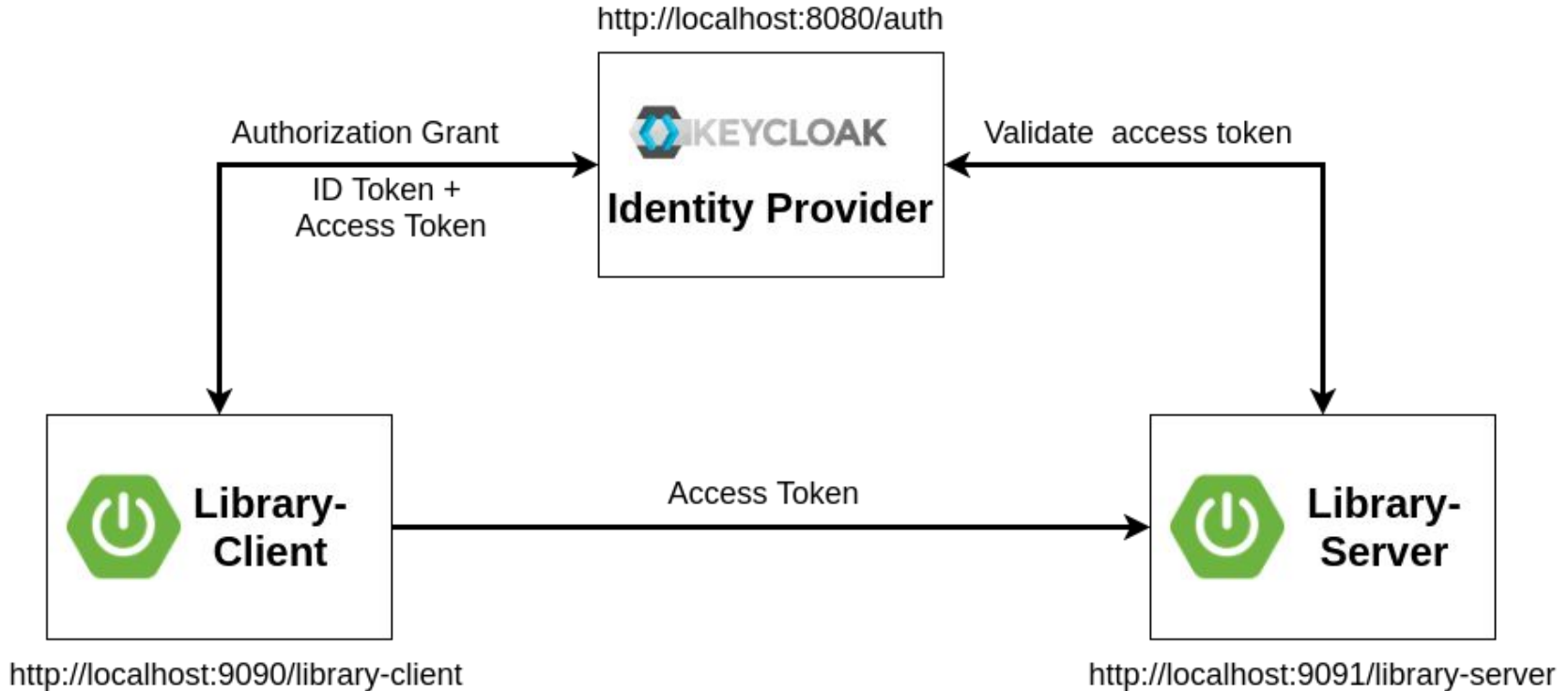
Lab 1: Implementing a Resource Server

The Hands-On Application: Use Cases

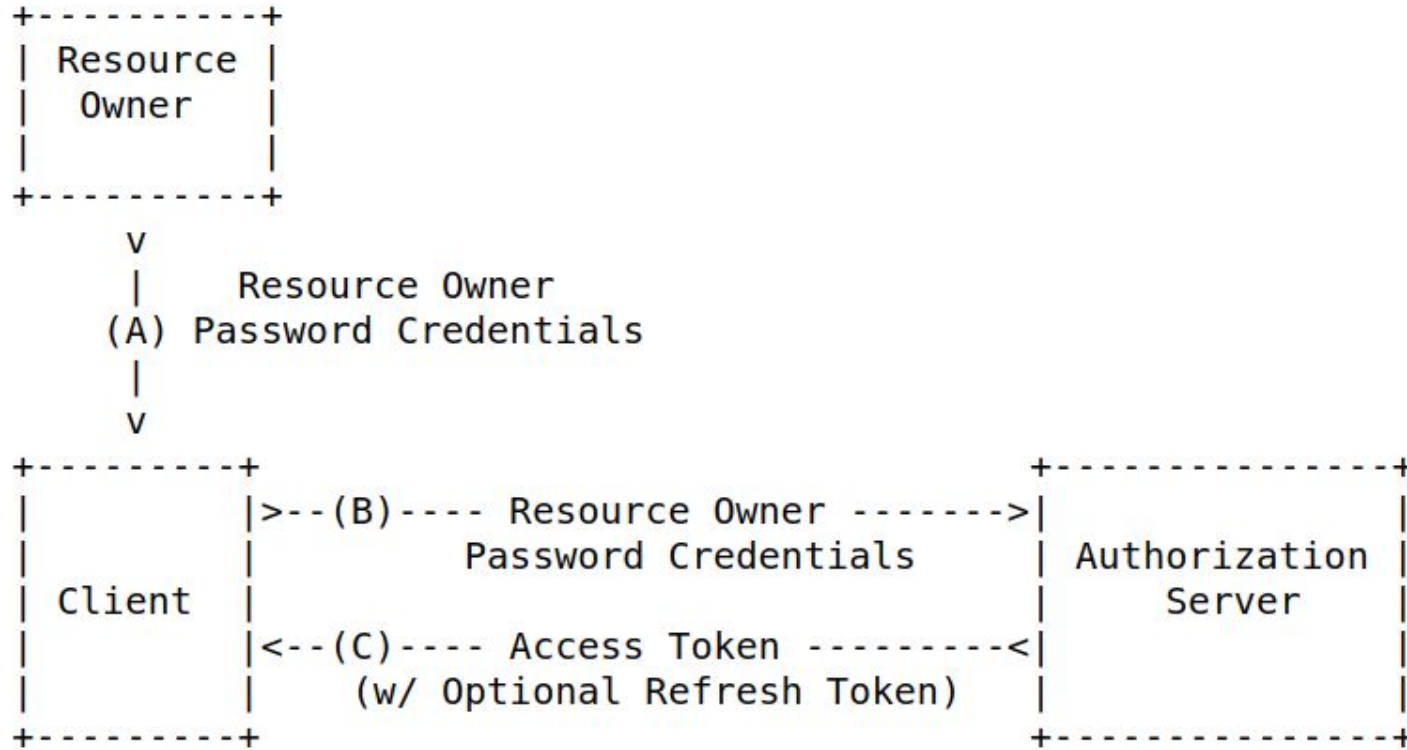
- Online Book Library
 - Administer Library Users
 - Administer Books
 - List available Books
 - Borrow a Book
 - Return a previously borrowed Book

<https://github.com/andifalk/secure-oauth2-oidc-workshop>

The Hands-On Application: Architecture



OAuth 2.0 Resource Owner Password Credentials Flow (1)



OAuth 2.0 Resource Owner Password Credentials Flow (2)

POST /token HTTP/1.1

Host: server.example.com

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded

grant_type=password&username=johndoe&password=A3ddj3w

POST /token HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded

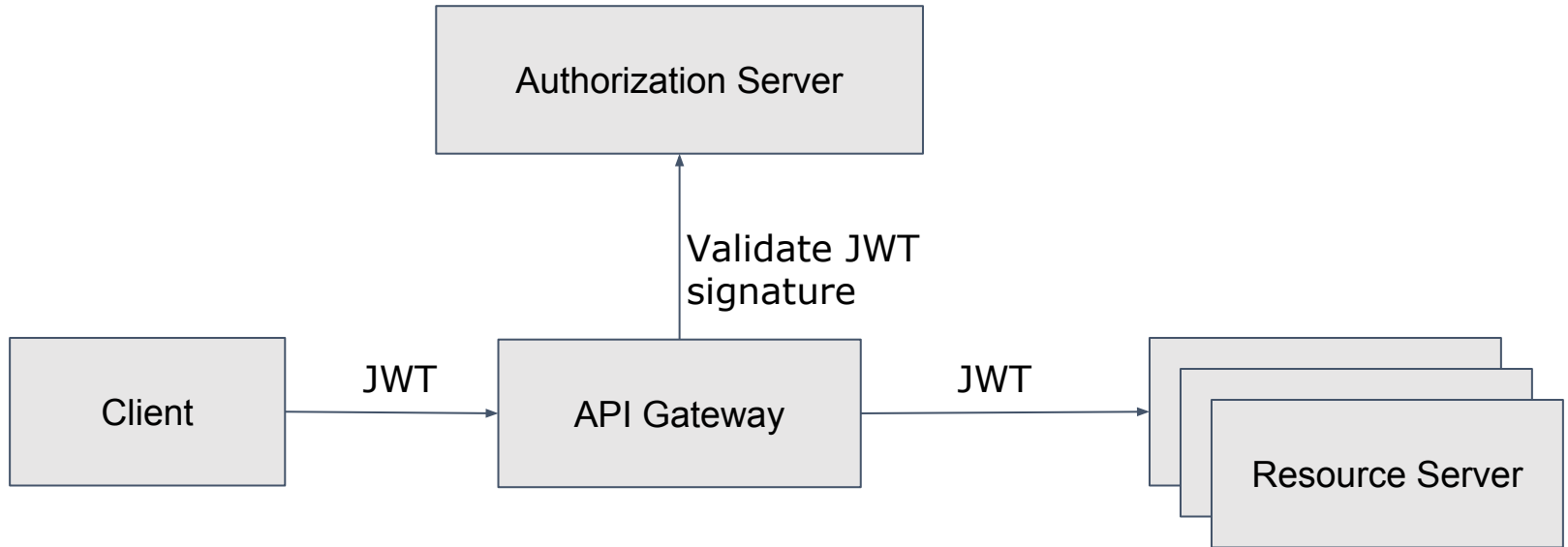
grant_type=password&username=johndoe&password=A3ddj3w
&client_id=123&client_secret=xyz



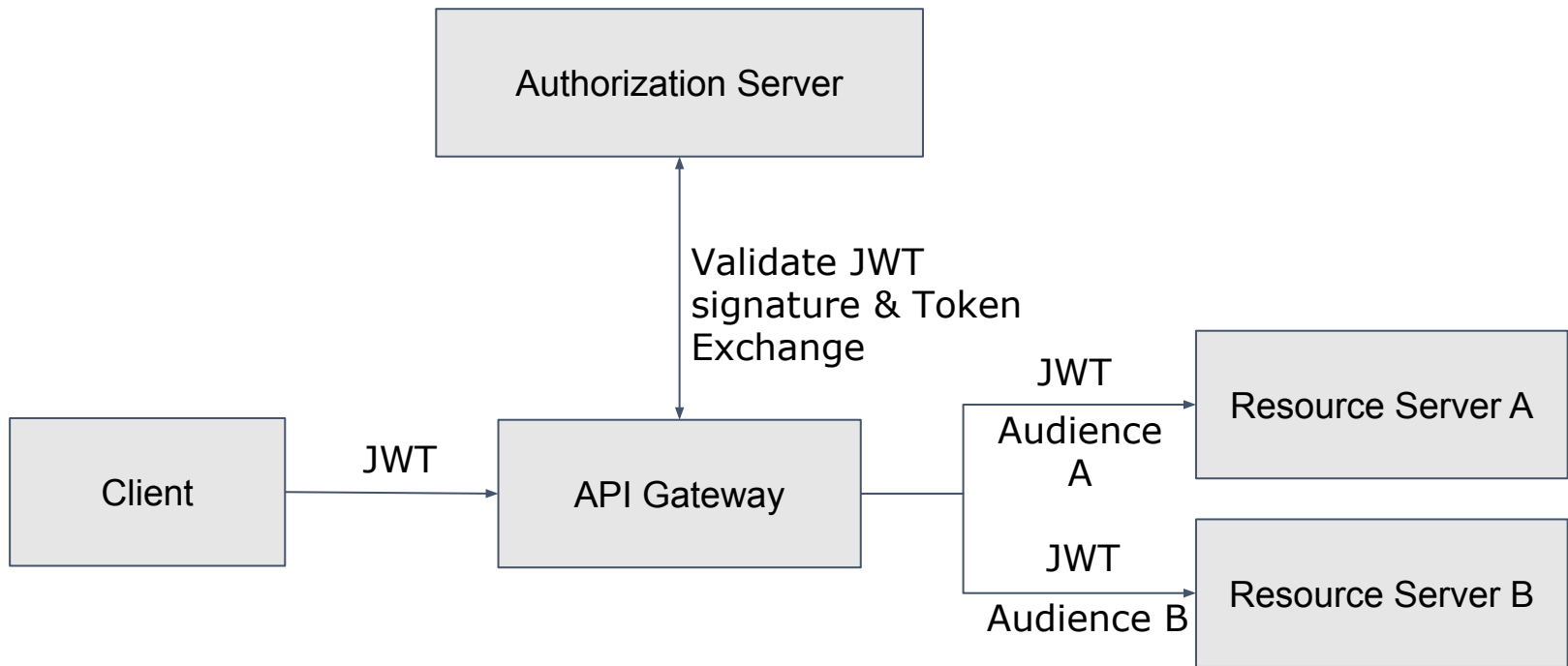
Advanced OpenID Connect Scenarios



Token-Relay with an API Gateway

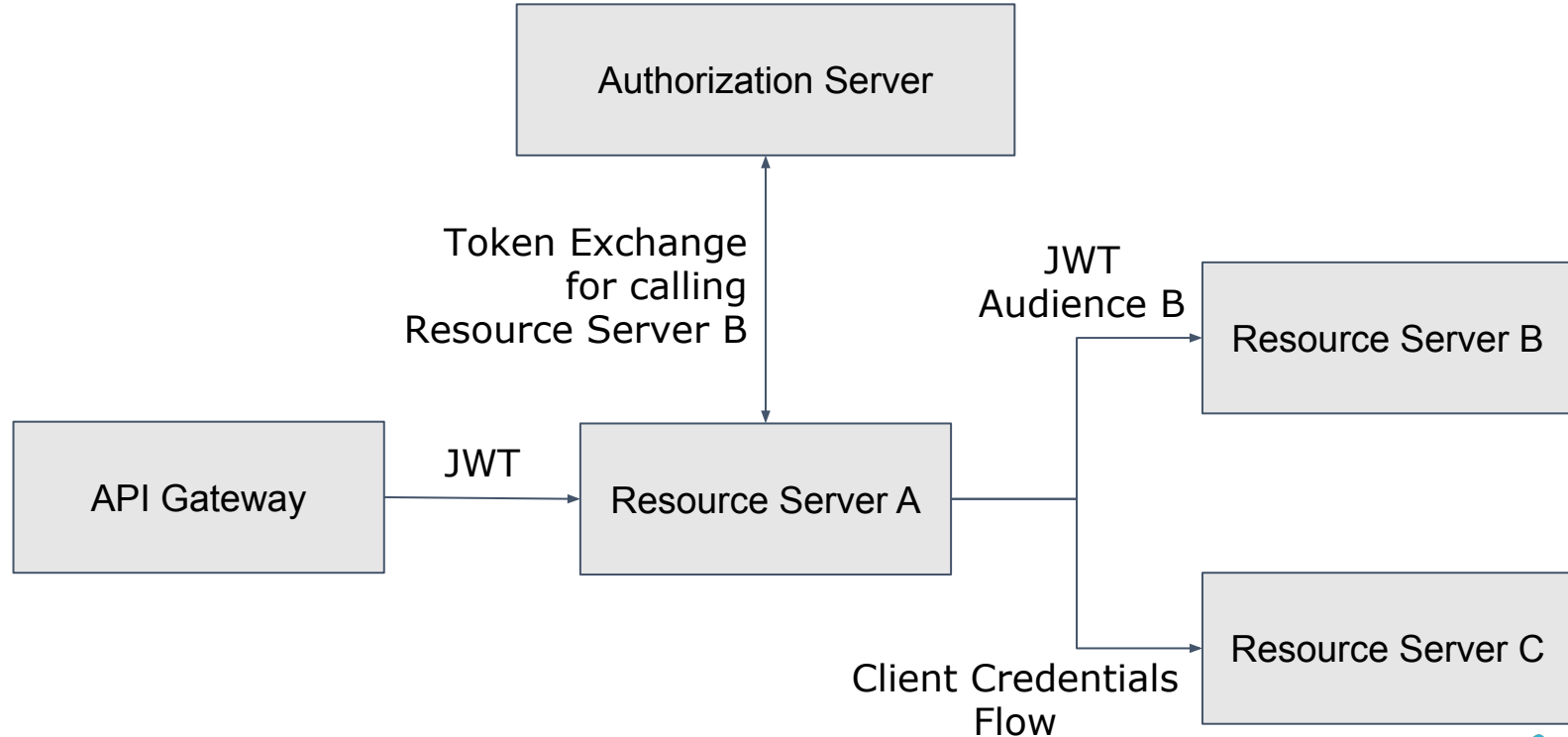


Token-Exchange with an API Gateway



<https://tools.ietf.org/html/draft-ietf-oauth-token-exchange-19>

Microservice-to-Microservice calls





OpenID Connect on the Client side

OAuth 2.0 Grant Flows

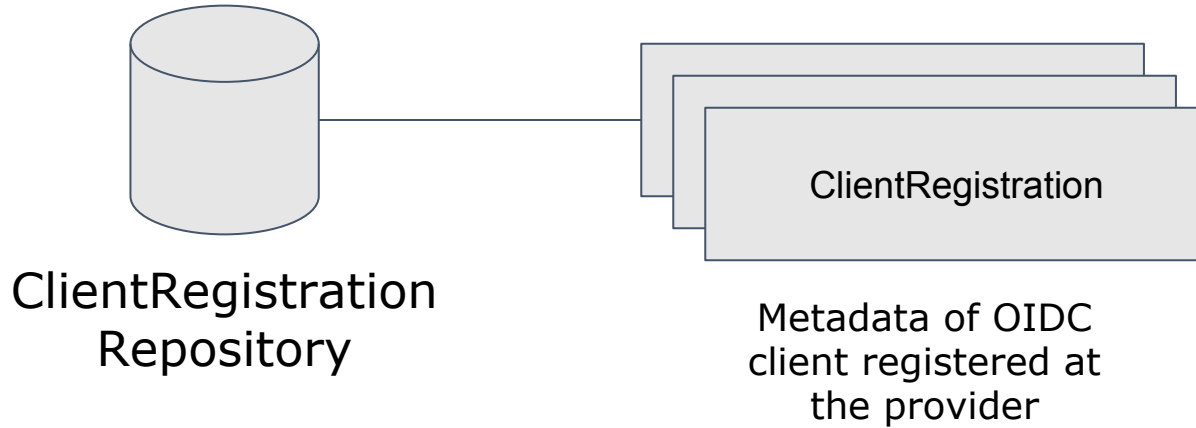
Client Type	Flow	Refresh Tokens
Confidential	Authorization Code	X
Public (Native)	Authorization Code (PKCE)	X
Public (SPA)	Implicit	--
Trusted	RO Password Creds	X
No Resource Owner	Client Credentials	--

OpenID Connect Libraries

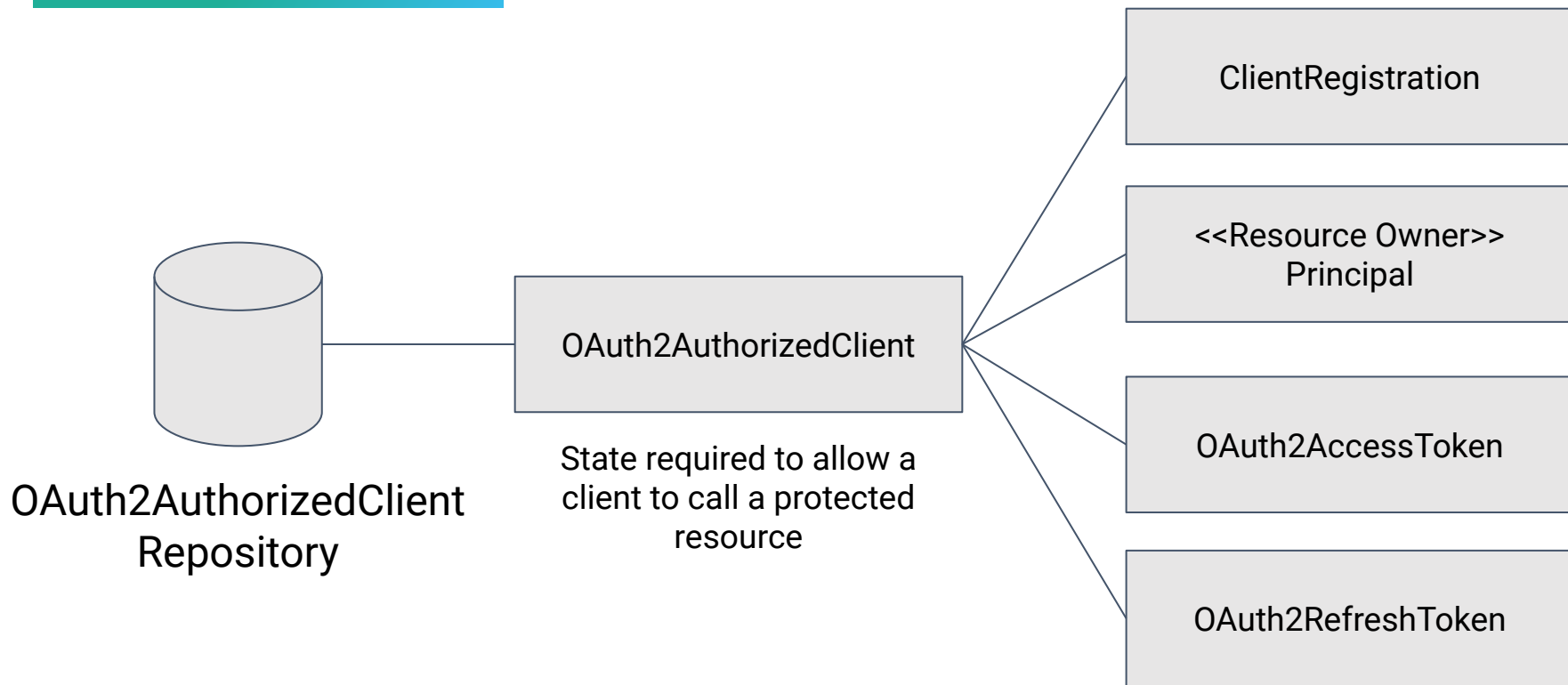
- oidc-client (Javascript) <https://github.com/IdentityModel/oidc-client-js>
- angular-oauth2-oidc (Typescript)
<https://github.com/manfredsteyer/angular-oauth2-oidc>
- angular-auth-oidc-client (Typescript)
<https://github.com/damienbod/angular-auth-oidc-client>
- IdentityModel.OidcClient (C#/.Net)
<https://github.com/IdentityModel/IdentityModel.OidcClient>
- Nimbus OAuth 2.0 SDK (Java)
<https://connect2id.com/products/nimbus-oauth-openid-connect-sdk>
- OIDC RP library (Python) <https://github.com/openid/JWTConnect-Python-OidcRP>
- ...

See: <https://openid.net/developers/certified/#OPServices>

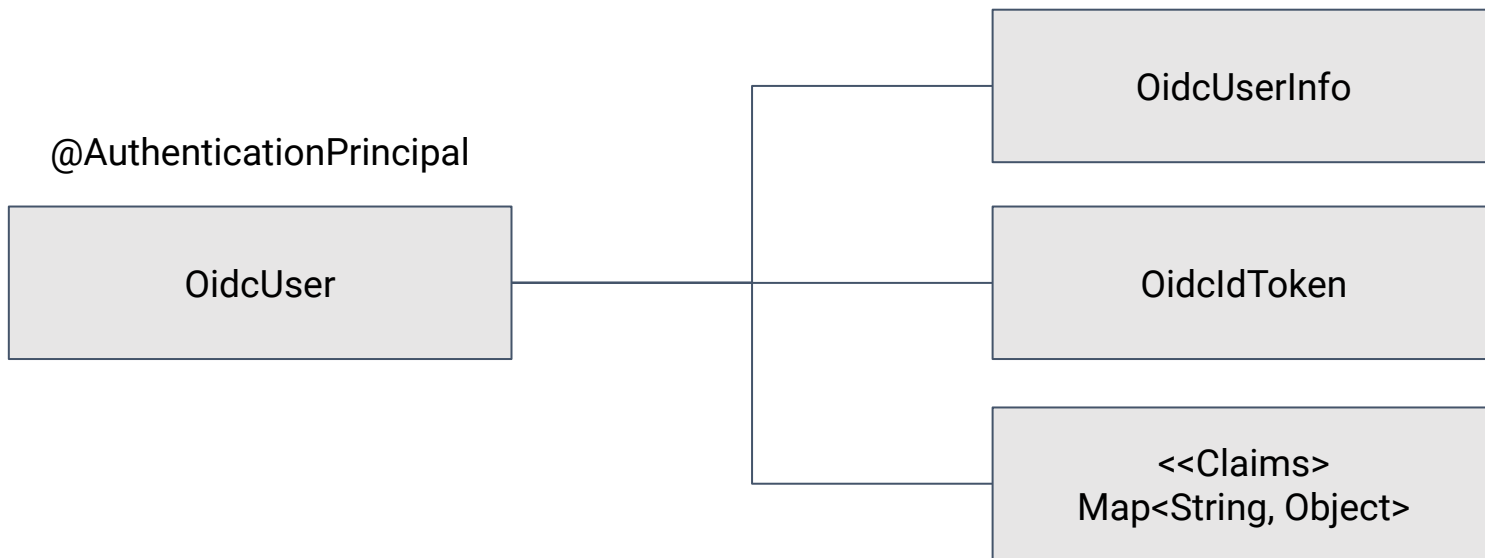
Spring Security: ClientRegistration



Spring Security: OAuth2AuthorizedClient



Spring Security: OidcUser





Practice Time

Lab 2: Implementing the client side (Authorization Code Flow)

OAuth 2.0 / OIDC Client (Spring MVC / Thymeleaf)

Library Client

A nice library to borrow books

Hello Bruce Wayne

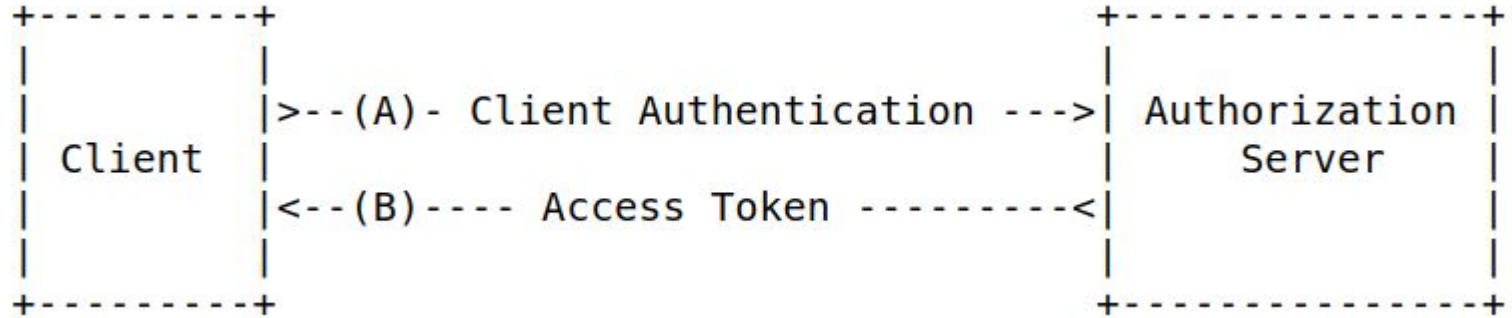
Author(s)	Title	Description	ISBN	
[Bob C. Martin]	Clean Code	Even bad code can function. But if code isn't clean, it can bring a development organization to its knees. Every year, countless hours and significant resources are lost because of poorly written code. But it doesn't have to be that way. Noted software expert Robert C. Martin presents a revolutionary paradigm with Clean Code: A Handbook of Agile Software Craftsmanship. Martin has teamed up with his colleagues from Object Mentor to distill their best agile practice of cleaning code "on the fly" into a book that will instill within you the values of a software craftsman and make you a better programmer—but only if you work at it.	9780132350884	Return
[Josh Long, Kenny Bastiani]	Cloud Native Java	What separates the traditional enterprise from the likes of Amazon, Netflix, and Etsy? Those companies have refined the art of cloud native development to maintain their competitive edge and stay well ahead of the competition. This practical guide shows Java/JVM developers how to build better software, faster, using Spring Boot, Spring Cloud, and Cloud Foundry.	9781449374648	
[Craig Walls]	Spring in Action: Covers Spring 4	Spring in Action, Fourth Edition is a hands-on guide to the Spring Framework, updated for version 4. It covers the latest features, tools, and practices including Spring MVC, REST, Security, Web Flow, and more. You'll move between short snippets and an ongoing example as you learn to build simple and efficient J2EE applications. Author Craig Walls has a special knack for crisp and entertaining examples that zoom in on the features and techniques you really need.	9781617291203	Borrow
[Gene Kim, Jez Humble, Patrick Debois]	The DevOps Handbook	Wondering if The DevOps Handbook is for you? Authors, Gene Kim, Jez Humble, Patrick Debois and John Willis developed this book for anyone looking to transform their IT organization—especially those who want to make serious changes through the DevOps methodology to increase productivity, profitability and win the marketplace.	9781942788003	Borrow



Practice Time

Lab 3: Implementing the client side (Client Credentials Flow)

OAuth 2.0 Client Credentials Grant Flow (1)



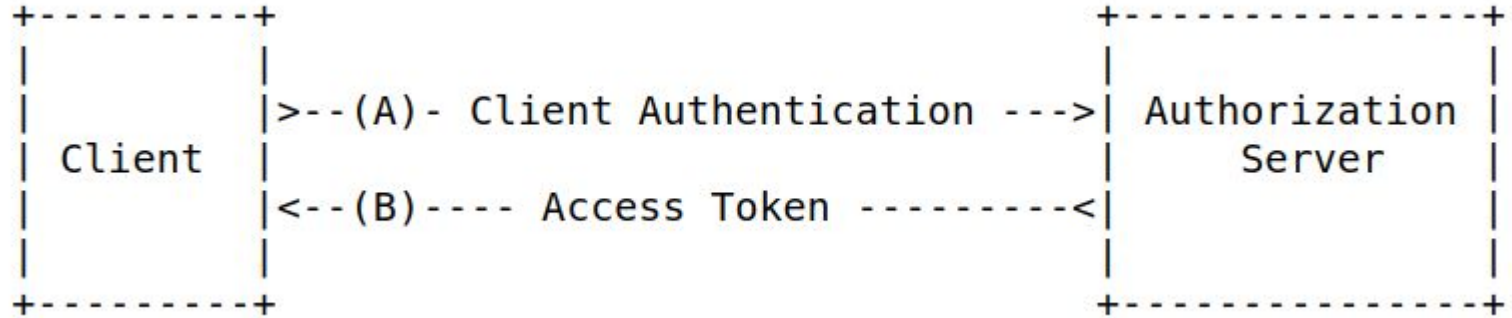
POST /token HTTP/1.1

Host: server.example.com

Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

Content-Type: application/x-www-form-urlencoded
grant_type=client_credentials

OAuth 2.0 Client Credentials Grant Flow (2)



POST /token HTTP/1.1

Host: server.example.com

Content-Type: application/x-www-form-urlencoded

grant_type=client_credentials&client_id=123&client_secret=xyz



Practice Time

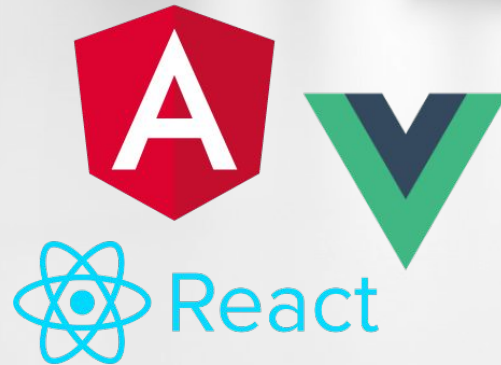
Lab 4: Testing JWT Auth+Authz

General Testing Strategies for JWT Auth+Authz

- Using self-signed JWT Tokens
- Using real identity provider (using Testcontainers)
- Only test the authorization layer

<https://www.testcontainers.org/>

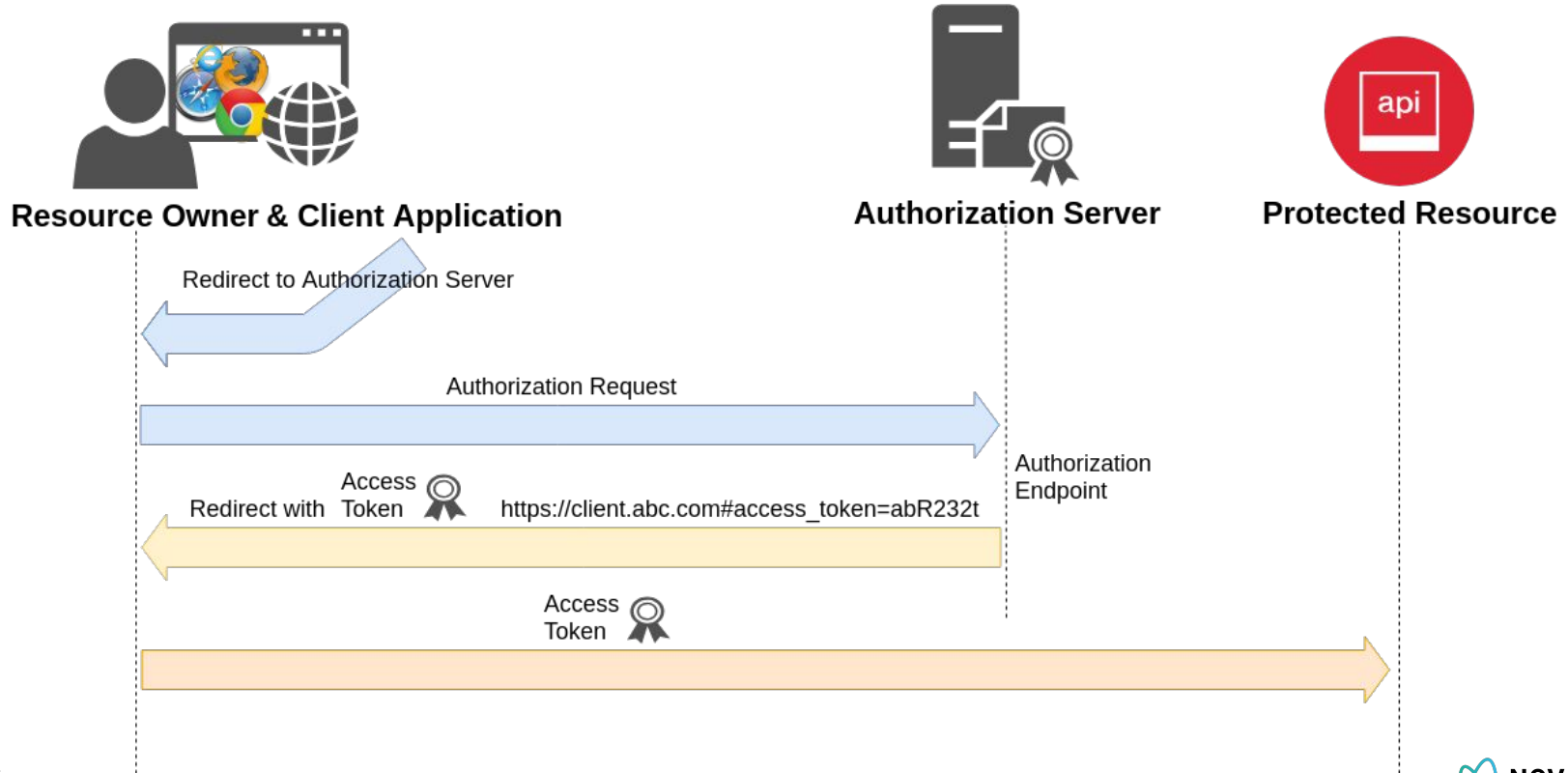
What about Single Page Applications?



OAuth 2.0 Grant Flows

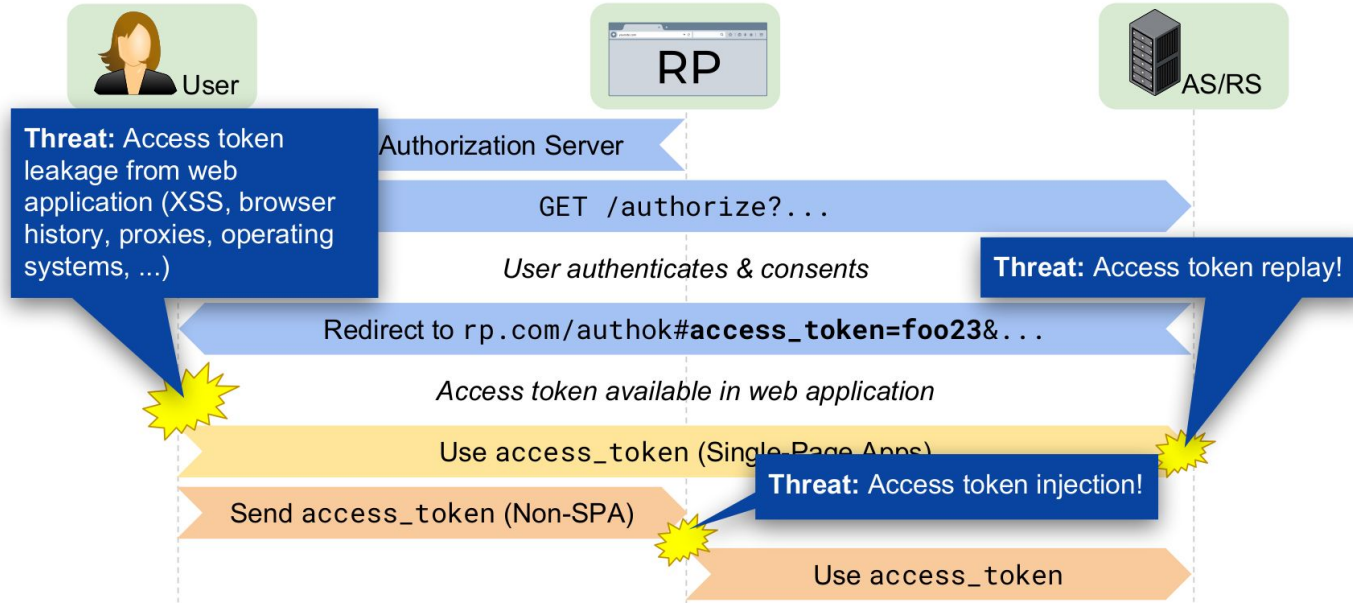
Client Type	Flow	Refresh Tokens
Confidential	Authorization Code	X
Public (Native)	Authorization Code (PKCE)	X
Public (SPA)	Implicit	--
Trusted	RO Password Creds	X
No Resource Owner	Client Credentials	--

OAuth 2.0 Implicit Flow



OAuth 2.0 Implicit Grant Flow is at Risk !

Don't use the OAuth Implicit Grant any longer!



<https://tools.ietf.org/html/draft-ietf-oauth-security-topics>

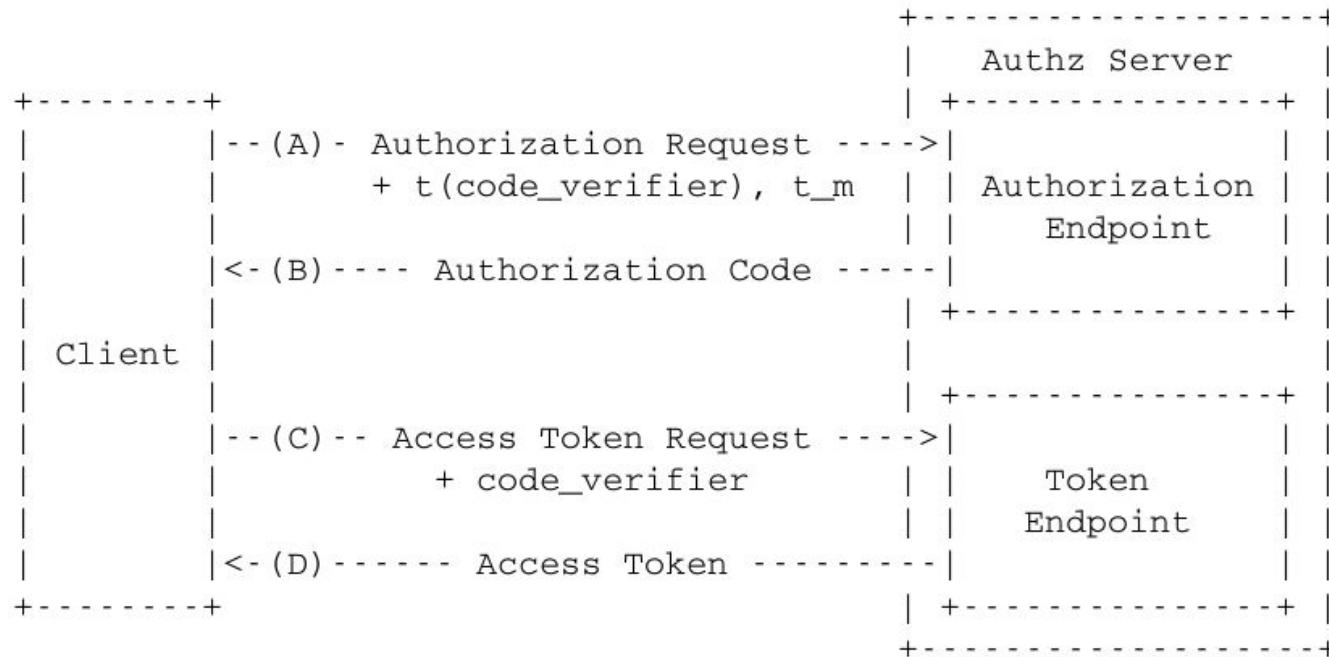
“OAuth 2.1” Grant Flows

Client Type	Flow	Refresh Tokens
Confidential	Authorization Code (PKCE)	X
Public (Native)	Authorization Code (PKCE)	X
Public (SPA)	Authorization Code (PKCE)	--
Trusted	RO Password Creds	X
No Resource Owner	Client Credentials	--

Authorization Code + PKCE Grant Flow

- PKCE-Proof Key for Code Exchange (“pixy”)
- Mitigates authorization code interception attack
- Public clients cannot keep secrets (“client_secret”)
 - PKCE adds a dynamically created cryptographically random key: The “code verifier”

OAuth 2.0 Auth Code + PKCE Grant Flow



code_verifier:
Random string
(43-128
characters)

t(code_verifier):
"code_challenge",
Hashed
code_verifier

t_m: Hashing
algorithm to be
used
("S256" or
"None")

<https://tools.ietf.org/html/rfc7636>



What's new in Spring Security 5.2&5.3

Spring Security 5.2

- Client Support for PKCE
- OpenID Connect RP-Initiated Logout
- Support for OAuth 2.0 Token Introspection
- Resource Server Multi-tenancy (Servlet & Reactive)
- Use symmetric keys with JwtDecoder
- JWT Flow API in Test Support

Support for Opaque Tokens

```
class ResSrvConfig extends WebSecurityConfigurerAdapter {  
  
    @Override  
    protected void configure(HttpSecurity http)  
        throws Exception {  
        http.oauth2ResourceServer()  
            .opaqueToken()  
                .introspectionUri(this.introspectionUri)  
                .introspectionClientCredentials(  
                    this.clientId, this.clientSecret);  
    }  
}
```

Resource Server Multi-Tenancy

```
class ResSrvConfig extends WebSecurityConfigurerAdapter {  
    @Override protected void configure(HttpSecurity http) {  
        http.oauth2ResourceServer()  
            .authenticationManagerResolver(  
                multitenantAuthenticationManager());  
    }  
  
    @Bean AuthenticationManagerResolver<HttpServletRequest>  
        multiTenantAuthMgr() {...}  
  
    AuthenticationManager jwt() {...}  
    AuthenticationManager opaque() {...}  
}
```

Support for Symmetric Keys

```
class ResSrvConfig extends WebSecurityConfigurerAdapter {  
    @Value("${spring.security.oauth2.resourceserver.  
            jwt.key-value}") RSAPublicKey key;  
  
    @Override protected void configure(HttpSecurity http) {  
        http.oauth2ResourceServer().jwt().decoder(jwtDecoder());  
    }  
  
    @Bean JwtDecoder jwtDecoder() throws Exception {  
        return NimbusJwtDecoder.  
            withPublicKey(this.key).build();  
    }  
}
```

JWT Flow API in Test Support

```
public class OAuth2ResourceServerTest {  
  
    @Test  
    public void testRequestPostProcessor() {  
        mockMvc.perform(get("/message")  
            .with(mockAccessToken().scope("message:read")))  
            .andExpect(status().isOk())  
  
        mockMvc.perform(get("/")  
            .with(jwt().claim(SUB, "the-subject")))  
            .andExpect(status().isOk())  
    }  
}
```

Spring Security 5.3

- Support OAuth 2.0 / OIDC Authorization Server:
 - OpenID Connect 1.0 (Authorization Code Flow)
 - PKCE
 - OAuth 2.0 Client Credentials Grant
 - JWT Access Token format
 - JWK Set Endpoint
 - Opaque Access Token format



What about other Microframeworks?

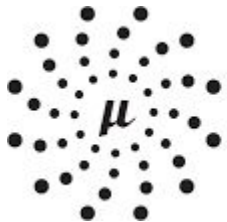
OpenID Connect in other Micro-Frameworks

Already good support for JWT based authentication in:

- Micronaut
<https://micronaut-projects.github.io/micronaut-security/latest/guide/#jwt>
- Quarkus <https://quarkus.io/guides/jwt-guide>



QUARKUS



MICRONAUT™

OpenID Connect in other Micro-Frameworks

See Bonus-Lab containing a demo for Micronaut!





Books and Online References

Books and Online References (1)

- Justin Richer et.al: OAuth2 in Action (Manning, 2017, ISBN 978-1617293276)
- Michael Schwartz et.al: Securing the Perimeter (Apress, 2018, ISBN 978-1484226001)
- [RFC 6749: The OAuth 2.0 Authorization Framework](#)
- [RFC 6750: OAuth 2.0 Bearer Token Usage](#)
- [RFC 6819: OAuth 2.0 Threat Model and Security Considerations](#)
- [RFC 7636: Proof Key for Code Exchange \("Pixy"\)](#)
- [OpenID Connect Core 1.0 Specification](#)
- [OpenID Connect Dynamic Client Registration 1.0](#)
- [OpenID Connect Discovery 1.0](#)
- [RFC 7519: JSON Web Token \(JWT\)](#)
- [JSON Web Token Best Current Practices](#)

Books and Online References (2)

- [Why you should stop using the OAuth implicit grant](#)
- [OAuth 2.0 Security Best Current Practice](#)
- [OAuth 2.0 for Browser-Based Apps](#)
- [OAuth 2.0 Mutual TLS Client Authentication and Certificate-Bound Access Tokens](#)
- [JSON Web Token \(JWT\) Profile for OAuth 2.0 Access Tokens](#)
- [OAuth 2.0 Token Exchange](#)

Books and Online References (3)

- [Resource Indicators for OAuth 2.0](#)
- [Spring Security 5.2 Reference Documentation](#)
- [Microservices Security Patterns & Protocols with Spring Security \(Devoxx Video\)](#)
- [Microservices Security Patterns & Protocols \(SpringOne Platform 2019 Video\)](#)
- [How to secure your Spring apps with Keycloak by Thomas Darimont \(Video\)](#)



Andreas Falk

Managing Consultant

Mobil: +49 151 46146778

E-Mail: andreas.falk@novatec-gmbh.de

Novatec Consulting GmbH

Dieselstraße 18/1

D-70771 Leinfelden-Echterdingen

T. +49 711 22040-700

info@novatec-gmbh.de

www.novatec-gmbh.de