

L'intégration de Kafka et Keycloak dans le projet de microservices de réseau social

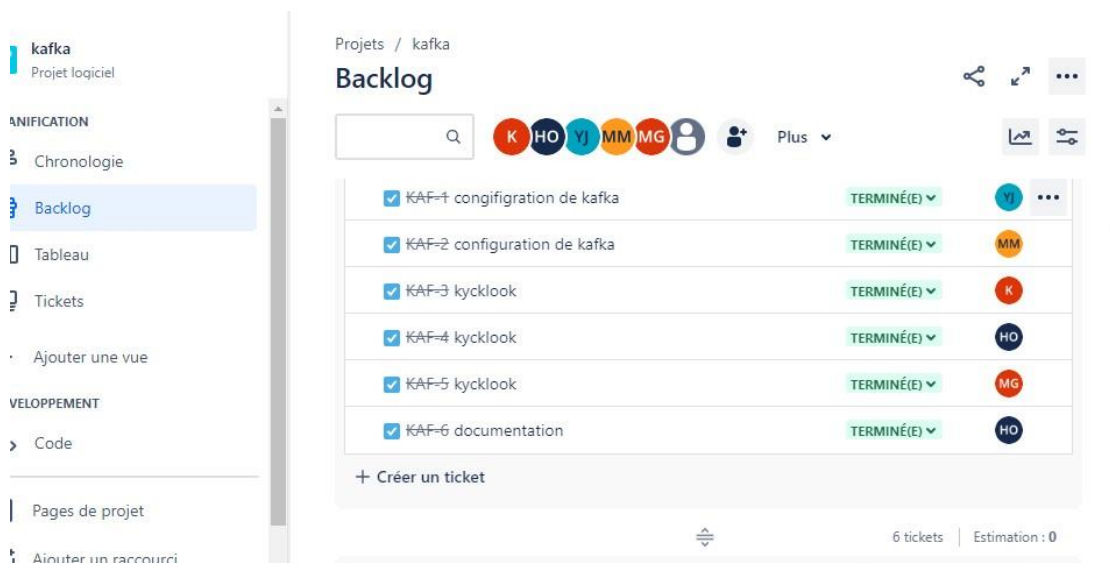
Introduction

Ce rapport présente les détails de l'intégration des technologies Apache Kafka et Keycloak dans notre projet de microservices de réseau social. L'objectif principal de cette intégration est d'améliorer la scalabilité, la sécurité et la réactivité de notre application, tout en offrant une expérience utilisateur fluide et efficace.

Outil de collaboration

Jira :

Pour réussir KANBAN, nous nous servons de 'JIRA' qui est un outil de gestion de projet en ligne. JIRA est une plateforme multifonction qui vise à faciliter la gestion de projet en aidant à suivre les tâches, identifier les blocages et partager des informations entre les membres d'une équipe. Il est basé sur une organisation des projets en tickets, chacune représentant des tâches. De plus, JIRA permet d'assurer le suivi des tickets et la d' définition d'un workflow adapté aux m' méthodes de travail. JIRA génère aussi des graphiques et visuels qui permettent de repérer en un coup d'œil l'état des différentes missions et d'identifier les problèmes à régler en priorité pour pouvoir avancer.



Intégration d'Apache Kafka

Apache Kafka est intégré comme courtier de messages pour faciliter la communication asynchrone entre les microservices. Chaque microservice produira des messages vers des sujets Kafka lorsqu'un événement se produira, par exemple, lors de l'inscription d'un utilisateur ou de la publication de contenu. D'autres microservices consommeront ces messages pour effectuer des actions ou des mises à jour pertinentes. Cette architecture assure une communication lâche et évolutive entre les composants.

Intégration de Keycloak

Keycloak est intégré pour gérer l'authentification et l'autorisation des utilisateurs. Chaque microservice déléguera l'authentification des utilisateurs à Keycloak, garantissant ainsi un mécanisme d'authentification centralisé et sécurisé. Keycloak émettra des jetons JWT lors d'une authentification réussie, utilisés par les microservices pour autoriser les actions des utilisateurs en fonction des rôles et des permissions définis.

Les étapes :

1. Installation du keycloak et la création d'un nouveau realm

The screenshot displays the Keycloak administration interface. At the top, the 'KEYCLOAK' logo is visible alongside a user profile 'admin'. A left-hand navigation menu lists various management options: Manage, Clients, Client scopes, Realm roles, Users, Groups, Sessions, Events, Configure, Realm settings, Authentication, Identity providers, and User federation. The main content area shows a 'Welcome to Connexia' message with a brief instruction on how to manage the realm.

Below this, the 'Clients' section is active, showing the 'Client details' for 'Connexia-ressource'. The 'OpenID Connect' protocol is selected, and the client is currently 'Enabled'. A 'Jump to section' sidebar on the right lists: General Settings, Access settings, Capability config, Login settings, and Logout settings. The 'General Settings' tab is selected, showing fields for 'Client ID' (Connexia-ressource), 'Name' (Connexia-ressource), and 'Description' (une application réseaux sociaux). The 'Always display in UI' toggle is currently 'Off'. At the bottom, 'Save' and 'Revert' buttons are available.

2. Creation d'un nouveau client connexia-ressource

Create client

Clients are applications and services that can request authentication of a user.

1 General Settings

2 **Capability config**

3 Login settings

Client authentication ⓘ
☒ On

Authorization ⓘ
☐ Off

Authentication flow ⓘ

☐ Standard flow ⓘ☐ Direct access grants ⓘ
☐ Implicit flow ⓘ☒ Service accounts roles ⓘ
☐ OAuth 2.0 Device Authorization Grant ⓘ
☐ OIDC CIBA Grant ⓘ

Next

Back

Cancel

3. On crée un autre client pour oauth

[Clients](#) > Client details

Connexia-oauth

OpenID Connect

☒ Enabled ⓘ Action ▾

Clients are applications and services that can request authentication of a user.

Settings

Keys

Credentials

Roles

Client scopes

Sessions

Advanced

General Settings

Client ID ⓘ ⓘ

Connexia-oauth

Name ⓘ

Connexia-oauth

Description ⓘ

Always display in UI ⓘ

☐ Off

Access settings

Save

Revert

Jump to section

General Settings

Access settings

Capability config

Login settings

Logout settings

Capability config

Client authentication ☒ On [?](#)

Authorization [?](#) ☐ Off

Authentication flow ☒ Standard flow [?](#) ☐ Direct access grants [?](#)
☐ Implicit flow [?](#) ☐ Service accounts roles [?](#)
☐ OAuth 2.0 Device Authorization Grant [?](#)
☐ OIDC CIBA Grant [?](#)

4. On crée un user

Create user

Required user actions [?](#)

Username *

Email

Email verified [?](#) ☐ No

First name

Last name

Groups [?](#)

5. Après la création on assigne un mot de passe

Users > User details

moon Enabled Action

Details Attributes Credentials Role mapping Groups Consents Identity provider links Sessions

ID * ce423764-1e67-4b26-a88e-ba993d98cb47

Created at * 3/14/2024, 1:46:26 PM

Required user actions Select action

Username * moon

Email mounaguelsa@gmail.com

Email verified ☐ No

First name

Last name

Save Revert

Users > User details

moon Enabled Action

Details Attributes Credentials Role mapping Groups Consents Identity provider links Sessions

Credential Reset

Type	User label	Data
Password		Show data Reset password

6. On assigne un role au user

Users > User details

moon Enabled Action

Details Attributes Credentials Role mapping Groups Consents Identity provider links Sessions

Search by name → ☒ Hide inherited roles Assign role Unassign 1-2 < >

<input type="checkbox"/> Name	Inherited	Description
<input type="checkbox"/> USER	False	utilisateur de l'application
<input type="checkbox"/> default-roles-connexia	False	\${role_default-roles}

1-2 < >

L'implémentation

Cette fonction `convert` est une implémentation de la méthode `convert` de l'interface `Converter` dans Java. Dans ce cas particulier, cette fonction est conçue pour convertir un objet Jwt (JSON Web Token) en une collection de rôles accordés (`GrantedAuthority`)

```

package com.exemple.geteway.security;

import org.springframework.core.convert.converter.Converter;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.oauth2.jwt.Jwt;

import java.util.ArrayList;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import java.util.stream.Collectors;

1 usage  Mouna GUELSA
public class KeycloakRoleConverter implements Converter<Jwt, Collection<GrantedAuthority>> {

    Mouna GUELSA
    @Override
    public Collection<GrantedAuthority> convert(Jwt source) {
        Map<String, Object> realmAccess = (Map<String, Object>) source.getClaims().get("realm_access");
        if (realmAccess == null || realmAccess.isEmpty()) {
            return new ArrayList<>();
        }
        Collection<GrantedAuthority> returnValue = ((List<String>) realmAccess.get("roles"))
            .stream().map(roleName -> "ROLE_" + roleName) Stream<String>
            .map(SimpleGrantedAuthority::new) Stream<SimpleGrantedAuthority>
            .collect(Collectors.toList());
        return returnValue;
    }
}

```

Configuration de sécurité

```

@Configuration
@EnableWebFluxSecurity
public class SecurityConfig {

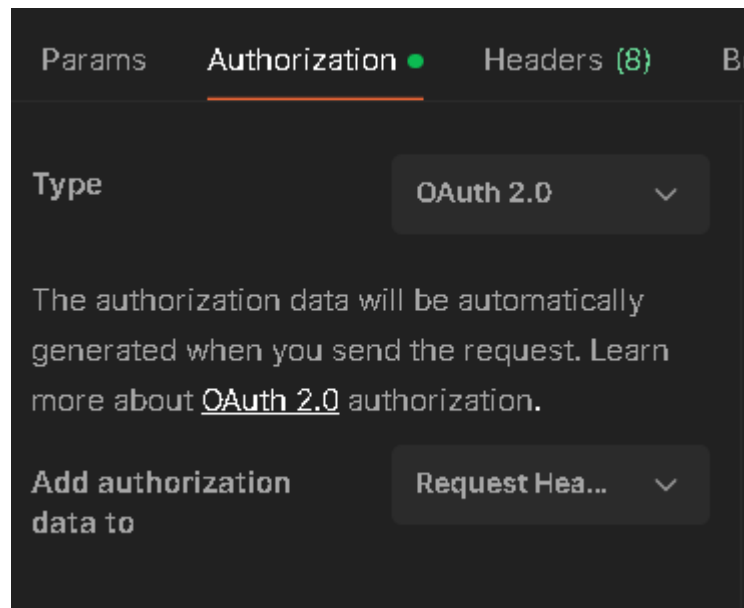
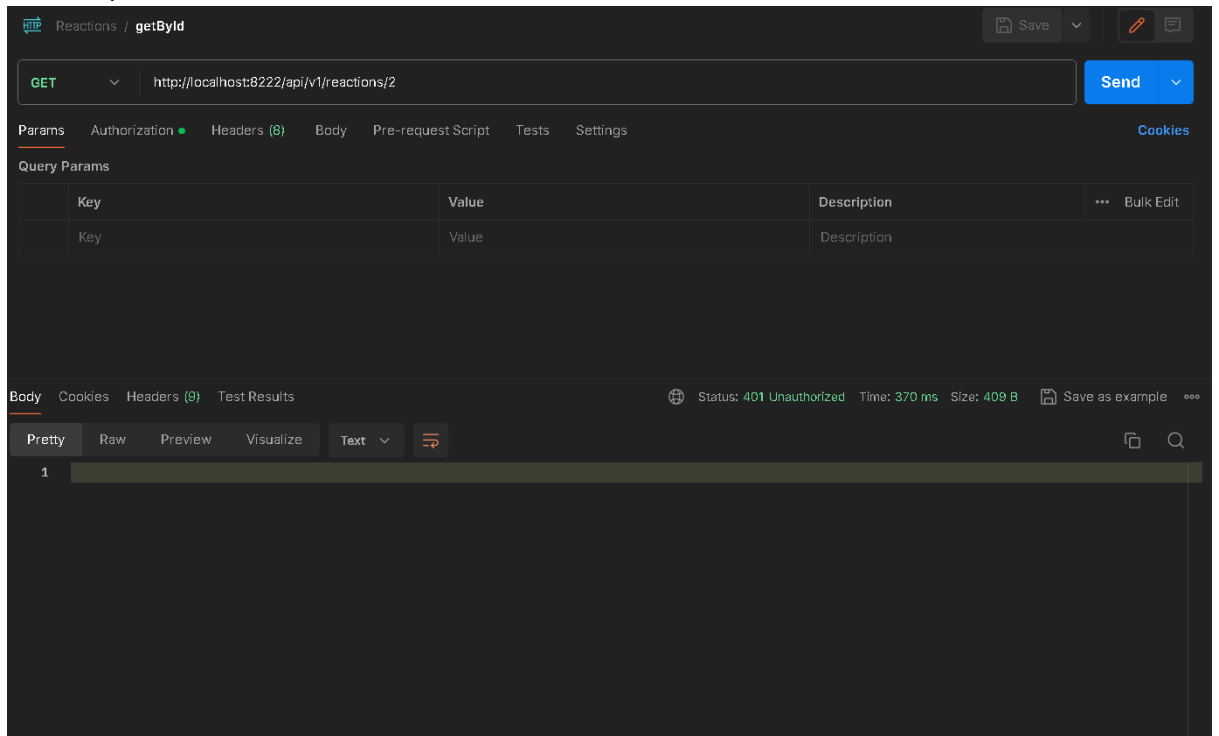
    Mouna GUELSA
    @Bean
    public SecurityWebFilterChain springSecurityFilterChain(ServerHttpSecurity serverHttpSecurity) {
        serverHttpSecurity.authorizeExchange(exchanges -> exchanges.anyExchange().hasRole("USER"))
            .oauth2ResourceServer(oAuth2ResourceServerSpec -> oAuth2ResourceServerSpec
                .jwt(jwtSpec -> jwtSpec.jwtAuthenticationConverter(grantedAuthoritiesExtractor())));
        serverHttpSecurity.csrf(csrfSpec -> csrfSpec.disable());
        return serverHttpSecurity.build();
    }

    1 usage  Mouna GUELSA
    private Converter<Jwt, Mono<AbstractAuthenticationToken>> grantedAuthoritiesExtractor() {
        JwtAuthenticationConverter jwtAuthenticationConverter =
            new JwtAuthenticationConverter();
        jwtAuthenticationConverter.setJwtGrantedAuthoritiesConverter
            (new KeycloakRoleConverter());
        return new ReactiveJwtAuthenticationConverterAdapter(jwtAuthenticationConverter);
    }
}

```

Postman

1. Dans la partie Autorisation on choisit : oauth2.0



2. On remplit les champs demandes

The screenshot shows the 'Authorization' tab of a REST client interface. The request method is 'GET' and the URL is 'http://localhost:8222/api/v1/reactions/2'. The 'Authorization' tab is selected, showing the 'OAuth 2.0' type. The 'Current Token' section displays a token 'eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXLTUwLiwia2' and its expiration time 'Expires at 10:24 pm today. Refresh'. The 'Header Prefix' is set to 'Bearer'. The 'Auto-refresh Token' toggle is turned on. The 'Share Token' toggle is turned off. The 'Configure New Token' section is visible at the bottom, showing the 'Token Name' as 'connexia'.

Reactions / getById

GET http://localhost:8222/api/v1/reactions/2 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Type OAuth 2.0

The authorization data will be automatically generated when you send the request. Learn more about [OAuth 2.0](#) authorization.

Add authorization data to Request Header

Current Token
This token is only available to you. Sync the token to let collaborators on this request use it.

Token connexia
eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXLTUwLiwia2
Expires at 10:24 pm today. [Refresh](#)

Header Prefix ① Bearer

Auto-refresh Token
Your expired token will be auto-refreshed before sending a request. ☒

Share Token
This will allow anyone with access to this request to view and use it. ☐

Configure New Token

Token Name connexia

The screenshot shows the 'Configure New Token' section of the 'Authorization' tab. The 'Token Name' is 'connexia'. The 'Grant type' is 'Authorization Code'. The 'Callback URL' is 'http://your-application.com/registered/callback'. The 'Authorize using browser' checkbox is unchecked. The 'Auth URL' is 'http://localhost:8080/realm/Connexia/prot...'. The 'Access Token URL' is 'http://localhost:8080/realm/Connexia/prot...'. The 'Client ID' is 'Connexia-oauth' with a warning icon. The 'Client Secret' is '4xcjcyS937ytkuDDOocXHNt0nR2pyc80P' with a warning icon. The 'Scope' is 'e.g. read:org'. The 'State' is 'State'.

GET http://localhost:8222/api/v1/reactions/2 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Configure New Token

Token Name connexia

Grant type Authorization Code

Callback URL ① http://your-application.com/registered/callback

☐ Authorize using browser

Auth URL ① http://localhost:8080/realm/Connexia/prot...

Access Token URL ① http://localhost:8080/realm/Connexia/prot...

Client ID ① Connexia-oauth ⚠

Client Secret ① 4xcjcyS937ytkuDDOocXHNt0nR2pyc80P ⚠

Scope ① e.g. read:org

State ① State

3. Pour l client secret on extraire from keycloak

[Clients](#) > Client details

Connexia-oauth OpenID Connect Enabled Action

Clients are applications and services that can request authentication of a user.

Settings Keys **Credentials** Roles Client scopes Sessions Advanced

Client Authenticator ⓘ Client Id and Secret Save

Client secret ⓘ Regenerate

Registration access token ⓘ Regenerate

4. Pour authURL on l'extrait from

KEYCLOAK admin

Connexia

Manage
Clients
Client scopes
Realm roles
Users
Groups
Sessions
Events
Configure
Realm settings
Authentication
Identity providers
User federation

Realm ID ⓘ Connexia ⓘ

Display name

HTML Display name

Frontend URL ⓘ

Require SSL ⓘ External requests

ACR to LoA Mapping ⓘ
No attributes have been defined yet. Click the below button to add attributes; key and value are required for a key pair.
[Add an attribute](#)

User-managed access ⓘ ☐ Off

Endpoints ⓘ
[OpenID Endpoint Configuration](#)
[SAML 2.0 Identity Provider Metadata](#)

Save Revert

Impact sur le projet

L'intégration de ces nouvelles technologies aura un impact significatif sur notre projet. Elle améliorera la scalabilité en permettant à nos microservices de gérer efficacement les charges de travail élevées. La sécurité sera renforcée grâce à l'authentification centralisée et aux mécanismes de contrôle d'accès fournis par Keycloak

Conclusion

En intégrant Apache Kafka et Keycloak dans notre projet de microservices de réseau social, nous visons à améliorer la scalabilité, la sécurité et la réactivité de notre application. Cette intégration nous permettra d'offrir une expérience utilisateur fluide et efficace. Nous sommes confiants que ces nouvelles technologies renforceront notre architecture et amélioreront nos fonctionnalités, conduisant ainsi à un produit final de haute qualité.

