Intégration de Amazon SQS dans un projet Spring Boot à l'aide de Spring Cloud AWS

Gestion des Notifications Utilisateur avec Spring Cloud AWS et Amazon SQS

Mené par :
NEJIB Hamza

Projet réalisé de 06 à 10 Janvier 2025

Village de l'Emploi

Table des matières

1 Résumé

L'objectif du projet est de créer un système capable d'envoyer et de traiter des notifications utilisateur en s'appuyant sur Amazon SQS (Simple Queue Service) via Spring Cloud AWS. Cela permet de garantir résilience, scalabilité et robustesse dans la gestion des événements clés tels que l'inscription, le changement de mot de passe, ou d'autres actions critiques.

2 Objectifs

Le projet vise à :

- Configurer une file d'attente SQS pour stocker temporairement les notifications.
- Envoyer des messages JSON représentant des notifications dans cette file.
- Consommer ces messages pour les traiter (affichage, envoi d'email ou SMS).
- Simuler des environnements locaux avec LocalStack, si AWS réel n'est pas disponible.

3 Initialisation du projet

Pour initialiser le projet Spring Boot avec Spring Cloud AWS et SQS, j'ai suivi les étapes ci-dessous.

3.1 Accéder au site Spring Initializr

J'ai accédé sur le site Spring Initializr et j'ai rempli les informations suivantes :

Project Maven
Language Java
Spring Boot Version 3.4.1
Group com.vde
Artifact notifications-app
Name notifications-app
Description Gestion des notifications utilisateur avec Spring Cloud AWS et SQS
Packaging Jar
Java Version 17

3.2 Ajouter les dépendances nécessaires

J'ai cliqué sur "Add Dependencies" et j'ai ajouté les dépendances suivantes :

- **Spring Web** pour la création de l'API REST.
- **Lombok** pour générer automatiquement les getters, setters, etc
- **Spring Boot DevTools** pour le rechargement automatique et un développement plus rapide.

3.3 Générer le projet

J'ai cliqué sur "Generate" pour télécharger le projet. Une fois généré, j'ai décompressé-le et le copié dans le dossier du projet créé à partir d'un répertoire git. Enfin, j'ai ajouté, commité et poussé les fichiers sur GitHub.

4 Installer et configurer LocalStack sur Windows

Pour simuler des services AWS en local, j'ai installé LocalStack qui permet de tester les services AWS, comme SQS, sur ma machine sans avoir besoin d'une connexion réelle à AWS.

4.1 Installer Docker Desktop pour Windows

LocalStack fonctionne avec Docker, donc la première étape consiste à installer Docker Desktop.
- J'ai téléchargé Docker Desktop pour Windows depuis le site officiel de Docker.
- J'ai suivi les instructions d'installation (redémarrer la machine si nécessaire).
- Une fois installé, j'ai ouvert Docker Desktop pour vérifier que Docker fonctionne correctement.

4.2 Installer AWS CLI

AWS CLI permet d'interagir avec LocalStack plus facilement, notamment pour créer et gérer des ressources comme SQS.
- J'ai téléchargé l'installer de l'AWS CLI depuis le site officiel.
- J'ai suivi les étapes d'installation et configuré AWS CLI en exécutant la commande suivante dans le terminal : **aws configure**.
- J'ai entré n'importe quelles valeurs pour la clé d'accès et la clé secrète (LocalStack n'en a pas besoin pour fonctionner localement).

4.3 Installer LocalStack via Docker

- Avant d'installer LocalStack, j'ai vérifié que Docker fonctionnait correctement en exécutant cette commande dans le terminal Windows (PowerShell ou Command Prompt) : **aws –version**. Si Docker est installé correctement, il renverra la version de Docker installée.
- Ensuite, j'ai exécuté LocalStack dans un conteneur Docker avec la commande suivante :

docker run –rm -d -p 4566 :4566 -p 4510-4559 :4510-4559 localstack/localstack

Cette commande télécharge et lance LocalStack dans un conteneur Docker en arrière-plan.
- Le port 4566 est utilisé pour accéder à tous les services simulés de LocalStack, tels que SQS, S3, DynamoDB, etc.
- Les ports 4510-4559 sont utilisés pour des services spécifiques comme Kinesis et DynamoDB.

4.4 Vérifier que LocalStack fonctionne

Après avoir lancé LocalStack avec Docker, j'ai vérifié que tout fonctionnait correctement :
- Vérification via le navigateur : J'ai accédé à http ://localhost :4566 dans mon navigateur. Si LocalStack fonctionnait correctement, j'ai vu la page d'accueil de LocalStack.
- Vérification via AWS CLI : J'ai utilisé AWS CLI pour lister les files SQS (ou d'autres services), en m'assurant que j'utilisais l'endpoint LocalStack. Par exemple :

  aws –endpoint-url=http ://localhost :4566 sqs list-queues

  Cette commande a retourné une liste des files SQS disponibles dans mon instance LocalStack.

4.5 Configurer Spring Boot pour utiliser LocalStack

Dans cette étape, j'ai configuré mon projet Spring Boot pour utiliser LocalStack en local. J'ai ajouté ces propriétés dans mon fichier application.properties :

```
cloud.aws.stack-name=localstack
cloud.aws.region.static=us-east-1
cloud.aws.credentials.access-key=mockAccessKey
cloud.aws.credentials.secret-key=mockSecretKey
cloud.aws.endpoint=http://localhost:4566

sqs.queue.name=notifications-queue
```

Cela m'a permis de rediriger les services AWS vers l'instance locale de LocalStack en m'assurant que mes clés d'accès étaient bien des valeurs factices

4.6 Ajouter les dépendances nécessaires

```xml
<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>io.awspring.cloud</groupId>
            <artifactId>spring-cloud-aws</artifactId>
            <version>3.2.0</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

```xml
<dependency>
    <groupId>io.awspring.cloud</groupId>
    <artifactId>spring-cloud-aws-starter-sqs</artifactId>
    <version>3.0.0</version>
</dependency>

<dependency>
    <groupId>io.awspring.cloud</groupId>
    <artifactId>spring-cloud-aws-starter</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-testcontainers</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.testcontainers</groupId>
    <artifactId>localstack</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.awaitility</groupId>
    <artifactId>awaitility</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>com.amazonaws</groupId>
    <artifactId>aws-java-sdk-sqs</artifactId>
    <version>1.12.340</version>
</dependency>
```

5 Étapes Pratiques

5.1 Créer le modèle de données pour les notifications

J'ai créé une classe Notification.java dans un package "model", cette classe représente une notification utilisateur.

```java
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Notification {
    private String userId;
    private String type;
    private String message;
}
```

5.2 Créer le service d'envoi des notifications

Pour publier des messages dans SQS, j'ai implémenté la classe NotificationSenderService. Elle a été conçue pour encapsuler la logique d'envoi des notifications.

```java
@Service
public class NotificationSenderService {
    private final AmazonSQS sqsClient;
    private final ObjectMapper objectMapper;

    @Value("${sqs.queue.name}")
    private String queueName;

    public NotificationSenderService(AmazonSQS sqsClient, ObjectMapper objectMapper) {
        this.sqsClient = sqsClient;
        this.objectMapper = objectMapper;
    }

    public void sendNotification(Notification notification) throws JsonProcessingException {
        String queueUrl = sqsClient.getQueueUrl(queueName).getQueueUrl();
        String messageBody = objectMapper.writeValueAsString(notification);

        SendMessageRequest request = new SendMessageRequest()
                .withQueueUrl(queueUrl)
                .withMessageBody(messageBody);

        sqsClient.sendMessage(request);
    }
}
```

5.3 Créer le service de réception des notifications

Pour consommer les messages depuis SQS, j'ai implémenté une classe NotificationListener. Cette classe sert à écouter la file SQS, récupérer les messages et traiter chaque notification.

```java
@Service
public class NotificationListener {

    private final AmazonSQSAsync sqsClient;

    @Value("${sqs.queue.name}")
    private String queueName;

    public NotificationListener(AmazonSQSAsync sqsClient) {
        this.sqsClient = sqsClient;
    }

    @MessageMapping
    public void listenNotifications() {
        String queueUrl = sqsClient.getQueueUrl(queueName).getQueueUrl();
        List<Message> messages = sqsClient.receiveMessage(queueUrl).getMessages();

        for (Message message : messages) {
            System.out.println("Received notification: " + message.getBody());
            sqsClient.deleteMessage(queueUrl, message.getReceiptHandle());
        }
    }
}
```