INFORMATIQUE
TECHNIQUE

Expert

AWS Lambda

Développez des micro-services en Java sur la plateforme serverless d'Amazon

En téléchargement

exemples et projets du livre

Version en ligne
OFFERTE !
pendant 1 an

Nicolas DUMINIL

eni

Table des matières

Les éléments à télécharger sont disponibles à l'adresse suivante :
http://www.editions-eni.fr
Saisissez la référence de l'ouvrage **EIAWSL** dans la zone de recherche et validez. Cliquez sur le titre du livre puis sur le bouton de téléchargement.

Introduction au Serverless

Chapitre 1
Introduction à AWS Lambda

Chapitre 2

AWS Lambda - Développement en Java

Chapitre 3
AWS Lambda - Implémentation en Java

Chapitre 4
Le développement d'API Serverless

Chapitre 5
Fonctions Lambda événementielles

Chapitre 6
Le service SQS (Simple Queue Service)

Chapitre 7
Le service DynamoDB

Chapitre 8
Tester les fonctions Lambda

Chapitre 9
La sécurité des fonctions Lambda

Chapitre 10
Les fonctions Lambda et Eclipse Microprofile

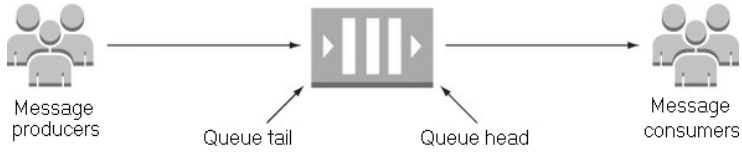
Chapitre 11
Pour aller plus loin

Chapitre 6
Le service SQS (Simple Queue Service)

1. Introduction

SQS est le service de messaging d'AWS. Il s'agit d'un service qui se veut « simple », comme son nom l'indique. Il est basé sur la notion de file d'attente ou *queue*, dans laquelle on peut stocker et récupérer, de manière plus ou moins ordonnée, des messages. Tout fonctionne selon une dynamique fournisseur/consommateur de messages, où le premier produit des messages et les stocke dans des files d'attente et le dernier les consomme.

SQS est un service distribué, à tolérance des pannes, qui permet à des fournisseurs/consommateurs multiples d'interagir par l'intermédiaire des files d'attente. Les messages échangés sont caractérisés par un cycle de vie standard, avec une période de rétention et une date d'expiration, au-delà desquelles ils sont supprimés. L'interaction qui a lieu entre les fournisseurs et les consommateurs de messages est complètement découplée, dans le sens où ils ne se connaissent pas, ce qui évite la création d'adhérences entre les composants.



Fournisseurs et consommateurs de messages SQS

Le service SQS propose deux types de files d'attente : standard et FIFO (*First In First Out*). Bien que très similaires, ces deux types de files d'attente présentent quand même des particularités qu'on va essayer de détailler ici.

2. Files d'attente SQS standard

Une des premières particularités des files d'attente standard est le fait qu'elles ne garantissent pas l'ordre selon lequel les messages sont stockés et, par conséquent, consommés. Bien que la loi du *best effort* s'y applique et que, par conséquent, l'ordre de l'arrivée des messages est dans la plupart des cas respecté, cela n'est pas garanti et il peut arriver que des messages soient traités en désordre. Ainsi, un consommateur de messages d'une file d'attente SQS standard ne peut pas présumer que l'ordre selon lequel il consomme les messages est bien celui dans lequel ils ont été initialement reçus.

De plus, l'unicité des messages dans la file n'est pas garantie non plus et, par conséquent, il peut arriver occasionnellement que plus d'une copie du même message soit délivrée à ses consommateurs. Ces deux particularités font que la fiabilité des files d'attente SQS est limitée, mais leur utilisation reste très intéressante dans des cas de figure moins contraignants, car elles proposent des rapports coûts/performances attractifs.

Ceci d'autant plus que les files d'attente standard peuvent supporter un nombre virtuellement illimité de transactions par seconde. Elles sont supportées par toutes les régions et par tous les services AWS, dont Lambda.

3. Files d'attente SQS FIFO

Ce type de file d'attente SQS a été spécialement conçu pour des cas d'utilisation dans lesquels garantir la consommation des messages dans l'ordre de leur arrivée est critique. Car les files d'attente SQS FIFO garantissent cet ordre. De plus, elles garantissent également l'unicité des messages, rendant ainsi impossible l'existence de doublons.

Les SQS FIFO se caractérisent par une fiabilité supérieure, comparées à leurs homologues standards. En revanche, contrairement à ces dernières, elles ne sont pas disponibles dans toutes les régions et sont limitées à maximum 300 transactions par seconde. Et si on rajoute le fait qu'elles ne sont pas supportées par tous les services AWS et que, s'agissant du service Lambda, leur support est seulement limité à un concept connu sous le nom de DLQ (*Dead Letters Queues*), qu'on va analyser dans un instant, alors il nous apparaît clairement que leur utilisation dans notre contexte spécifique n'est pas spécialement utile.

Le tableau ci-dessous permet de se repérer quant aux cas d'utilisation des deux types de files d'attente proposées par l'infrastructure AWS :

Critère	SQS FIFO	SQS Standard
Performances	300 TPS (*Transaction Per Second*)	Virtuellement illimitées
Ordre	Garanti	Non garanti
Livraison	Unique	Doublons possibles
Disponibilité	Nombre de régions limité	Toutes les régions
Consommation asynchrone	Supportée	Pas supportée
Services supportés	Limité	Tous

Comparaison des files d'attente SQS standard et FIFO

4. La production/consommation des messages SQS

La manipulation d'une file d'attente SQS standard, qui inclut des opérations comme la création, la production/consommation/suppression des messages, etc., peut se faire très simplement avec AWS CLI, comme suit :

```
nicolas@BEL20:~$ aws sqs create-queue --queue-name send-money-queue
{
    "QueueUrl": "https://sqs.eu-west-3.amazonaws.com/459436678662/
send-money-queue"
}
nicolas@BEL20:~$ aws sqs get-queue-attributes --queue-url
https://sqs.eu-west-3.amazonaws.com/459436678662/send-money-queue
nicolas@BEL20:~$ aws sqs get-queue-attributes --queue-url
https://sqs.eu-west-3.amazonaws.com/459436678662/send-money-queue
--attribute-name
ApproximateNumberOfMessages
{
    "Attributes": {
        "ApproximateNumberOfMessages": "0"
    }
}
nicolas@BEL20:~$ aws sqs get-queue-attributes --queue-url
https://sqs.eu-west-3.amazonaws.com/459436678662/send-money-queue
--attribute-name All
{
    "Attributes": {
        "QueueArn": "arn:aws:sqs:eu-west-3:459436678662:send-money-queue",
        "ApproximateNumberOfMessages": "0",
        "ApproximateNumberOfMessagesNotVisible": "0",
        "ApproximateNumberOfMessagesDelayed": "0",
        "CreatedTimestamp": "1597749946",
        "LastModifiedTimestamp": "1597749946",
        "VisibilityTimeout": "30",
        "MaximumMessageSize": "262144",
        "MessageRetentionPeriod": "345600",
        "DelaySeconds": "0",
        "ReceiveMessageWaitTimeSeconds": "0"
    }
}
nicolas@BEL20:~$ aws sqs list-queues
{
    "QueueUrls": [
        "https://sqs.eu-west-3.amazonaws.com/459436678662/send-money-queue"
    ]
}
nicolas@BEL20:~$ aws sqs send-message --queue-url https://sqs.eu-west-3.
amazonaws.com/459436678662/send-money-queue --message-body "Information
about the largest city in Any Region."
```

```
{
    "MD5OfMessageBody": "51b0a3256d59467f973009b739163aa0",
    "MessageId": "f90792ae-d0ff-437e-a2cd-b1d03cb38230"
}
nicolas@BEL20:~$ aws sqs receive-message --queue-url https://sqs.eu-west-3.
amazonaws.com/459436678662/send-money-queue
{
    "Messages": [
        {
            "MessageId": "f90792ae-d0ff-437e-a2cd-b1d03cb38230",
            "ReceiptHandle": "AQEBGKXlTIbXvny/C/Jvc8iRmYFLLZ7+JDgx1fa67Wys5o/
O116ZgQ2QNfmYLwWVLjfDD1NsBPoAF5130E7ExCf2xXKeBux67fItFBclcZSI596InBsxt54vBzrJf
S6gYsXCGsBCeynvTPx/EPh1PSOr6xOGyIj1Ko2Q7tws/36l67j0niwdBFde8QaCS5bbftGpYob3qz
EFfF5YJCoaL0hH14vWMX+ggODPORAirAypLmEQ4d8WGbxGKBwdSxubs1cw3+XWXtXCC2geR1yts9F
5nVu8mDJhquaGtOotzg/UvgAGQe+24THFYqb8f8OdgiwS/imhWvEznJptRTjMBrok5gH38IUK7gz
f4KgVz30Iba3jjUSt9KC2dduHbdSa9haS76JogjHOSqQ/xfsuChLCNA==",
            "MD5OfBody": "51b0a3256d59467f973009b739163aa0",
            "Body": "Information about the largest city in Any Region."
        }
    ]
}
nicolas@BEL20:~$ aws sqs purge-queue --queue-url https://sqs.eu-west-3.
amazonaws.com/459436678662/send-money-queue
nicolas@BEL20:~$ aws sqs receive-message --queue-url https://sqs.eu-west-3.
amazonaws.com/459436678662/send-money-queue
nicolas@BEL20:~$ aws sqs delete-queue --queue-url https://sqs.eu-west-3.
amazonaws.com/459436678662/send-money-queue
nicolas@BEL20:~$ aws sqs list-queues
nicolas@BEL20:~$
```

Le listing ci-dessus montre d'abord la commande `aws sqs create` queue qui permet de créer une file d'attente SQS standard. Une fois créée, cette file peut être interrogée avec `aws sqs get-attributes`. Notez bien la manière selon laquelle on passe le nom de l'attribut dont la valeur nous intéresse, par exemple `ApproximateNumberOfMessages`, ou `All` pour tous les attributs.

On peut facilement produire des messages dans la file d'attente SQS standard qu'on vient de créer avec `aws sqs send-message` et les consommer avec `aws sqs receive-message`. Enfin, on peut purger une file d'attente avec `aws sqs purge` et la supprimer avec `aws sqs delete-queue`. À tout moment on peut obtenir la liste des files d'attente créées pour l'utilisateur courant avec `aws sqs list-queues`.

La documentation complète de la commande `aws sqs` se trouve ici :
https://docs.aws.amazon.com/cli/latest/reference/sqs

Évidemment, toutes les opérations présentées sont également accessibles via SAM, comme on va le voir un peu plus tard. Mais en tant que développeur Java, ce qui nous intéresse en premier lieu ce n'est pas la ligne de commande avec AWS CLI ou SAM, mais leur implémentation en Java.

Tout se passe dans l'interface `com.amazonaws.services.sqs.AmazonSQS` qui définit toutes les méthodes nécessaires et dont la documentation se trouve ici : https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-sqs.html. Voici quelques-unes des méthodes les plus utilisées :

```
CreateQueueResult createQueue(String queueName);
GetQueueUrlResult getQueueUrl(String queueName);
ListQueuesResult listQueues();
SendMessageResult sendMessage(String queueUrl,
                  String messageBody);
SendMessageBatchResult sendMessageBatch(String queueUrl,
                       String messageBody);
ReceiveMessageResult receiveMessage(String queueUrl);
PurgeQueueResult purgeQueue(PurgeQueueRequest purgeQueueRequest);
DeleteQueueResult deleteQueue(String queueUrl);
```

On peut même tenter de définir les étapes principales d'une session SQS ainsi :

– Obtention d'une instance de `AmazonSQS`. Pour ce faire on peut utiliser des constructeurs, mais la méthode à privilégier reste l'utilisation de `AmazonSQSClientBuilder`.

– En supposant que la file d'attente cible a déjà été créée par le processus de *build*, ce qui est typiquement le cas, on récupère son URL via la méthode `GetQueueUrlResult getQueueUrl (String queueName)`.

– On produit des messages via la méthode `SendMessageResult sendMessage (String queueName, String messageBody)`. On peut aussi produire des messages en masse avec `SendMessageBatchResult sendMessageBatch (String queueName, String messageBody)`.

– On peut consommer des messages avec `ReceiveMessageResult receiveMessage (String queueUrl)`.

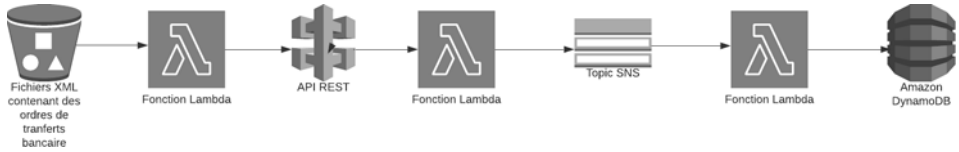
– À tout moment on peut obtenir des informations sur nos files d'attente avec `ListQueuesResult listQueues()`.
– On peut purger une file d'attente avec `PurgeQueueResult purgeQueue (PurgeQueueRequest purgeQueueRequest)`.
– Enfin, on peut supprimer une file d'attente avec `DeleteQueueResult deleteQueue (String queueUrl)`.

5. Le scénario de test

Lorsque nous nous sommes occupés du service API Gateway, au chapitre Le développement d'API Serverless, nous avions fourni une implémentation minimaliste des fonctions Lambda auxquelles les points d'entrée de notre API étaient connectés. Il est temps maintenant que nous complétions cette implémentation.

Ainsi, nous allons remplacer l'implémentation actuelle qui ne fait qu'une simple journalisation des messages, avec une autre qui publie, dans une file d'attente dédiée, chaque nouvel ordre de virement bancaire. Ceci afin que cet ordre puisse être récupéré par le service qui effectue le processus de transfert a proprement dit, service qui est en général un service externe à l'organisation. Et s'agissant d'une opération non déterministe, dont on ne peut pas présumer la durée, on ne peut pas non plus, par conséquent, ni anticiper ni attendre le résultat. D'où la nécessité d'un traitement asynchrone et complètement découplé, consistant à déposer les ordres de virements dans une file d'attente, pour qu'ils soient récupérés et traités en temps voulu.

Pour mémoire, notre scénario général est reproduit ci-dessous.



Le scénario général