

# TP 3 - Lambda


## Mise en place

Allez sur la plateforme AWS academy et accédez au cours AWS Academy Learner Lab [43226]. Puis cliquez sur `Modules > Learner Lab`. Lancez votre environnement en cliquant sur `Start Lab`. Une fois le cercle passé au vert, cliquez sur `AWS Details` et `AWS CLI`. Les clés que vous voyez vont permettre un accès programmatique à votre compte. Cherchez le dossier `.aws` sur votre machine puis remplacez le contenu du fichier `credentials` par les clés que vous venez de récupérer

## Ma première lambda

### Définition de la Lambda

Une fois sur la console AWS, cherchez le service `Lambda` dans la barre de recherches. Sur le dashboard Lambda cliquez sur `Créer une fonction`. Laissez l'option `Créer à partir de zéro` cochée, donnez un nom à votre fonction lambda, et pour le langage d'exécution sélectionnez `python3.9`. Conservez l'architecture `x86_64`, et dépliez `Modifier le rôle d'exécution par défaut`, sélectionnez `utiliser un rôle existant` et sélectionnez le rôle `LabRole`. Créez votre fonction

 À la différence des instances EC2, une fonction Lambda a besoin d'un rôle pour fonctionner. Sans rentrer dans les détails un rôle va déterminer les droits de la fonction. Comme votre compte n'a pas le droit de création de rôle, vous ne pouvez pas créer un rôle à la volée, et il faut sélectionner le rôle `LabRole` déjà créé.

Une fois sur la page de votre fonction, un code de base est proposé par AWS. Ce code retourne simplement un code 200 et le texte `Hello from Lambda!`. Vous allez lancer cette fonction via le bouton `Test`. Créez un nouvel événement de test, que l'on va appeler `test_basique`, et laissez le json de base. Votre événement de test sera un simple json de la forme


```
1 {  
2   "key1": "value1",  
3   "key2": "value2",  
4   "key3": "value3"  
5 }
```

Enregistrez-le et cliquez de nouveau sur `Test`. Normalement tout devrait bien se passer.

Maintenant vous allez légèrement modifier la fonction. Au lieu de juste retourner une chaîne de caractères fixe, elle va retourner l'heure actuelle. Importez la classe `datetime` du module éponyme et utilisez le code `datetime.now().strftime("%m/%d/%Y, %H:%M:%S")` pour avoir l'heure et la date du jour comme un string. Comme votre fonction a été modifiée, il faut la redéployer avec le bouton `Deploy`. Une fois fait, testez-la de nouveau pour voir si tout fonctionne.

## Ajout de l'invocation toutes les minutes

Sur la page de votre lambda vous allez cliquer sur `+ Ajouter un déclencheur`. La source va être `EventBridge`. Créez une nouvelle règle avec le nom `minuteur`. Le type de règle sera `Expression de planification` et l'expression `rate(1 minute)`.

 EventBridge permet de gérer les événements comme des alarmes quand un seuil est dépassé, mais aussi les événements planifiés.

Votre fonction sera désormais appelée toutes les minutes. Malheureusement, comme il n'y a pas de destination pour votre fonction, les résultats disparaissent dans le néant du cloud. Il est bien possible de voir qu'elle est invoquée en allant sur l'onglet `Surveiller` puis `Journaux`. Vous allez voir une ligne par minute mais comme notre fonction de log rien, vous n'allez voir aucun résultat.

 Il est possible d'ajouter un logger (utile pour le debug) en faisant des `print()` (c'est pas beau), ou en utilisant le module `logging`

```
1 import os
2 import logging
3 logger = logging.getLogger()
4 logger.setLevel(logging.INFO)
5
6 def lambda_handler(event, context):
7     logger.info('## ENVIRONMENT VARIABLES')
8     logger.info(os.environ)
9     logger.info('## EVENT')
10    logger.info(event)
```

## Poussez les résultats dans une file SQS

Maintenant vous allez faire en sorte que votre fonction envoie ses résultats dans une file SQS. Cherchez le service SQS et créez une file. Elle sera du type Standard et donnez lui le nom que vous souhaitez. Gardez toutes les valeurs par défaut et créez votre file. Copiez url de la file.

Retournez sur la page de votre Lambda et modifiez le code pour publier dans la file SQS en vous aidant du code suivant :

```
1 import json
2 import boto3
3 from datetime import datetime
4 sqs = boto3.client('sqs') #client is required to interact with sqs
5
6 def lambda_handler(event, context):
7     # event provenant d'une lambda
8     data = int(json.loads(event["Records"][0]["body"])["data"])
9
10    sqs.send_message(
11        QueueUrl="VOTRE URL SQS",
12        MessageBody=json.dumps({"body" : data})
```

```

13     )
14     return {
15         'statusCode': 200,
16         'body': data
17     }

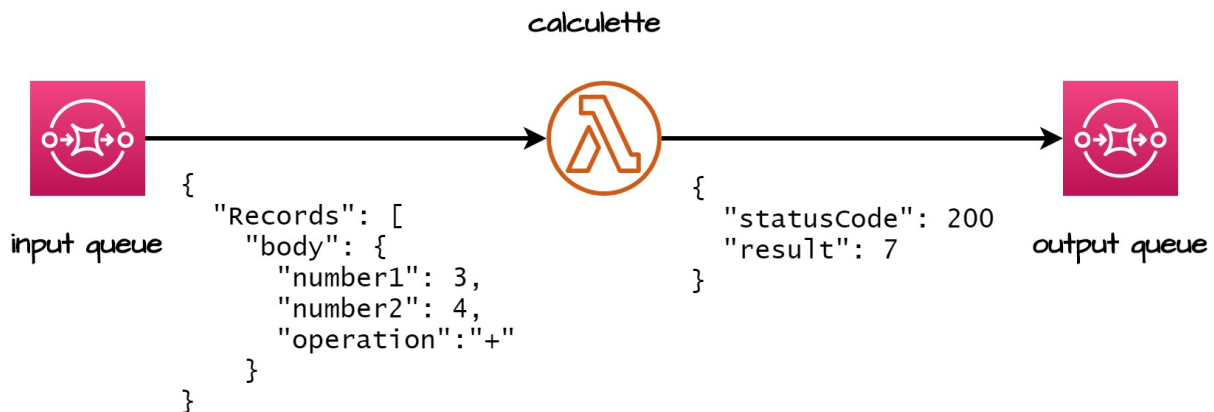
```

Déployer la nouvelle fonction, et attendez quelques minutes puis retournez sur la page de votre file et cliquez sur `Envoyez et recevoir des messages` puis sur `Recherchez des messages`. Vous devrez voir des messages apparaître. Cliquez sur un et vous devrez voir votre message.

🎉 Félicitation vous venez de mettre en place une architecture 100% serverless qui va réaliser un traitement toutes les minutes et pousser le résultat dans une file pour être utilisé par un autre service par la suite. Même si le code python du traitement est assez ridicule, l'architecture elle ne l'est pas. Vous pourriez par exemple avec ce système faire une requête toutes les heures à un web service mettre à jour des données en base.

## 🎲 Une calculatrice

Maintenant vous allez réaliser une calculatrice en utilisant une fonction lambda. Voici le schéma d'architecture globale.



Vous allez devoir créer :

- Deux files SQS, une pour l'input et une pour l'output
- Une fonction Lambda qui va aller chercher les clés `number1`, `number2` et `operation` et faire le calcul demandé. Les opérations que l'on souhaite faire sont l'addition, la soustraction, la multiplication et la division.

Le déclencheur de la fonction lambda passent des paramètres dans le dictionnaire `event`. Pour obtenir la clé `number1` vous devez faire `float(json.loads(event["Records"][0]["body"])["number1"])`

🧑 Pour vous aidez à comprendre ce code, les messages récupérés par la lambda sont dans la clé `Records`. SQS n'envoie pas les messages à la fonction lambda, c'est la fonction qui les pull. Sauf qu'elle peut en récupérer plusieurs à la fois, et la clé `Records` est une liste. Pour simplifier on ne regarde que le premier message d'où le `event["Records"][0]`, si vous avez envie, vous pouvez boucler sur les éléments de la liste. Ensuite chaque message est contenu

dans la clé `body`. Les messages sont considérés comme des strings car il n'y a aucune raison que ce soit un json, donc il nous faut le transformer en dictionnaire avec un `json.loads()`. Enfin il est possible de récupérer les clés que l'on souhaite. Attention à leur type ! Il faut spécifier que les nombres sont bien des nombres.

Pour tester votre application, vous pouvez :

- Faire un test via l'onglet test de la fonction lambda avec le json suivant :

```
1 {
2   "Records": [
3     {
4       "messageId": "bc8007e9-6a6d-41d4-ba09-2fcf16e5e6c3",
5       "receiptHandle": "AQEB92BoJQl1WctZSi iIQ69fXXX4ac7cpxxcbTirw4/b+ziBTzAx1wXFMbj3w6wbOPom4jP
usM9453dZDXi4iVH/vf97fFk6yg/EkP9UZRYrK5OwfwIXQJkk1we8ZKK84uYVhGIDi5kBfw
TCnsX6u83+GE59g/Uwc0+jbYvOArOLwCCOTRqbH3spkG/GhDH1yxVwPv/K+xNM+7pqQX21yj
SQdiLww1k7dDJwiNGatRq9D1vIDHduabmHn2I1sLrq778ZkZXS4YJ6IYeFXC+kwVY1Sy+1Xy
VxHfXBVXQCU8PsSNv6MsoBDgjU1LD43NFikQLVI5F/+HnBEX2AzhoJPBMz/eijkW1miJNZ48
G9gg2H2D0t0x2OQtg2M2VqtxROMD06gHUPsr67vvBH2J5m770xw==",
6       "body": "{\n\"number1\":1,\n\"number2\":5,\n\"operation\":\"\n+\n\"}\",
7       "attributes": {
8         "ApproximateReceiveCount": "18",
9         "SentTimestamp": "1681393246569",
10        "SenderId": "AROAZ2UVGELJYYC7FJZIV:user2476414=__tudiant_test",
11        "ApproximateFirstReceiveTimestamp": "1681393246569"
12      },
13      "messageAttributes": {},
14      "md5OfBody": "cb76cceb2fbc7622690cdf4f256ea8e0",
15      "eventSource": "aws:sqs",
16      "eventSourceARN": "arn:aws:sqs:us-east-1:675696485075:lab-input-
queue",
17      "awsRegion": "us-east-1"
18    }
19  ]
20 }
```

- Créer un message dans la queue d'input et voir si le résultat apparaît dans la queue d'output. Voici un exemple de message

```
1 {"number1":1,"number2":5,"operation":+"}
```

🎉 Félicitation vous venez de mettre en place une architecture 100% serverless avec trois services qui communiquent les uns les autres. mettre des files entre des services permet de découpler les services et d'avoir un système plus modulable. Par exemple dans notre cas, notre lambda ne sait pas d'où proviennent les données, elle sais juste les prendre depuis une file. Ainsi la source des données peut changer, du moment que la nouvelle source alimentera la file SQS il n'y aura pas de raison de changer la lambda. De la même façon, notre lambda ne se préoccupe pas du service qui va utiliser les

données qu'elle produit. Elle les posent simplement dans une file pour qu'un consommateur puisse les chercher. Les file SQS agissent comme des zone tampon entre les services.

S'il vous reste du temps pendant le TP commencez le TP noté.