CLOUD COMPUTING, CM3

Pépin Rémi, Ensai, 2023

remi.pepin@ensai.fr

LE SERVERLESS

UN SERVICE AVEC SERVEUR

Cas d'un service classique type web service

- Je choisi la configuration de mon instance EC2
- J'installe mon web service
- J'ai un accès total à ma machine

Je suis facturé au temps d'utilisation de ma mon service. Tant qu'il est allumée je le paye. Qu'il traite 2 ou 100 000 requêtes le prix sera le même.

UN SERVICE SANS SERVEUR ?

Cas d'une fonction as a Service

- Je choisi la puissance et le déclencheur de ma fonction
- J'upload le code de ma fonction
- Quand un événement déclenche ma fonction, mon CP la lance

Je suis facturé au temps d'exécution de ma fonction. Si elle est souvent déclenché je paye beaucoup, si elle ne l'est pas je ne paye rien.

LE SERVERLESS

- Terme mal choisi : des serveurs sont bien utilisés
- Mais ils sont 100% gérés par le CP et pas par vous !

Un service dit "serverless" est un service qui masque totalement la partie serveur au client, qui devient à la charge du CP. Ce n'est pas un SaaS car fourni des ressources informatiques.

DIFFÉRENTS SERVICES SERVERLESS

- Function as a Service : AWS Lambda, GCP Cloud Functions, Azure Functions
- Base de données : AWS Aurora serverless, AWS DynamoDB, GCP Datastore
- Stockage objet : Amazon S3, GCP Cloud Storage, OVH Object Storage
- Et bien d'autres

AVANTAGES / INCONVÉNIENTS

Avantages

- Coût (si utilisation imprévisible)
- Scalability/Elasticity : théoriquement pas de limite car géré par CP
- Productivité : facile à déployer car moins de configuration, maintenance

Désavantages possible

- Latence : temps de lancement à prendre en compte (il existe des solutions)
- Puissance limitée : conçu pour des traitements courts et léger
- Monitorage et débogage : si plusieurs services communiquent entre eux vite difficile de comprendre ce qu'il se passe.

Permet de se concentrer sur le métier.

AWS LAMBDA

- Compatible avec des nombreux langages (dont python)
- Configuration de la puissance, timeout, concurrence, variable d'env
- Pas de serveurs à gérer
- Facturation au temps d'exécution (ms)
- Environnement éphémère géré par AWS
- Temps exécution max : 15 min

Bout de code qui sera exécuté en fonction de déclencheurs (périodicité, requête HTTP, alerte etc)

AWS LAMBDA : EXEMPLE

```
import json
import os
print('Loading function')

def lambda_handler(event, context):
    print("Received event: " + json.dumps(event, indent=2))
    print("value1 = " + event['key1'])
    print(os.getenv("foo"))
    return event['key1']  # Echo back the first key value
```

- lambda_handler : la fonction qui sera appelée
- event : un dictionnaire avec les params. Dépend du déclencheur
- context : des métadonnées (nom de la fonction, mémoire max etc)

CODE CDKTF

```
class LambdaStack(TerraformStack):
    def init(self, scope: Construct, id: str):
        super().init(scope, id)
        AwsProvider(self, "AWS", region="us-east-1")

        # Packagage du code
        code = TerraformAsset(
            self, "code",
            path="./lambda",
            type= AssetType.ARCHIVE
        )

        # Create Lambda function
        lambda_function = LambdaFunction(self,
                "lambda",
```

AWS SQS : SIMPLE QUEUE SERVICE

- Service pour avoir une queue d'événements (~FIFO)
- Peut déclencher une lambda ou servir d'output
- Pas de serveurs à gérer
- Facturation au nombre d'évènements

Permet de faire le lien entre différents service.

AWS SQS : SIMPLE QUEUE SERVICE

```
{
    "Records": [
        {
        "messageId": "19dd0b57-b21e-4ac1-bd88-01bbb068cb78",
        "receiptHandle": "MessageReceiptHandle",
        "body": "Hello from SQS!", // <<==== event ici, souvent un objet json (clé:valeur)
        "attributes": {
            "ApproximateReceiveCount": "1",
            "SentTimestamp": "1523232000000",
            "SenderId": "123456789012",
            "ApproximateFirstReceiveTimestamp": "1523232000001"
        },
        "messageAttributes": {},
        "md5OfBody": "{{{md5_of_body}}}",
        "eventSource": "aws:sqs",
```

AWS SQS + LAMBDA

```
class LambdaStack(TerraformStack):
    def init(self, scope: Construct, id: str):
        super().init(scope, id)
        AwsProvider(self, "AWS", region="us-east-1")
        # Packagage du code
        code = TerraformAsset(...)
        # Create Lambda function
        lambda_function = LambdaFunction(...)
        # Create SQS queue
        queue = SqsQueue(
            self,
            "queue",
            name="input_queue",
            visibility_timeout_seconds=60
        )
```

TP AWS LAMBDA

Objectif

- Créer une fonction Lambda simple qui se déclenche périodiquement via la console
- Créer une fonction Lambda qui somme 2 nombres provenant d'une queue SQS, puis écris le résultat dans une autre queue SQS

THAT'S ALL FOLKS