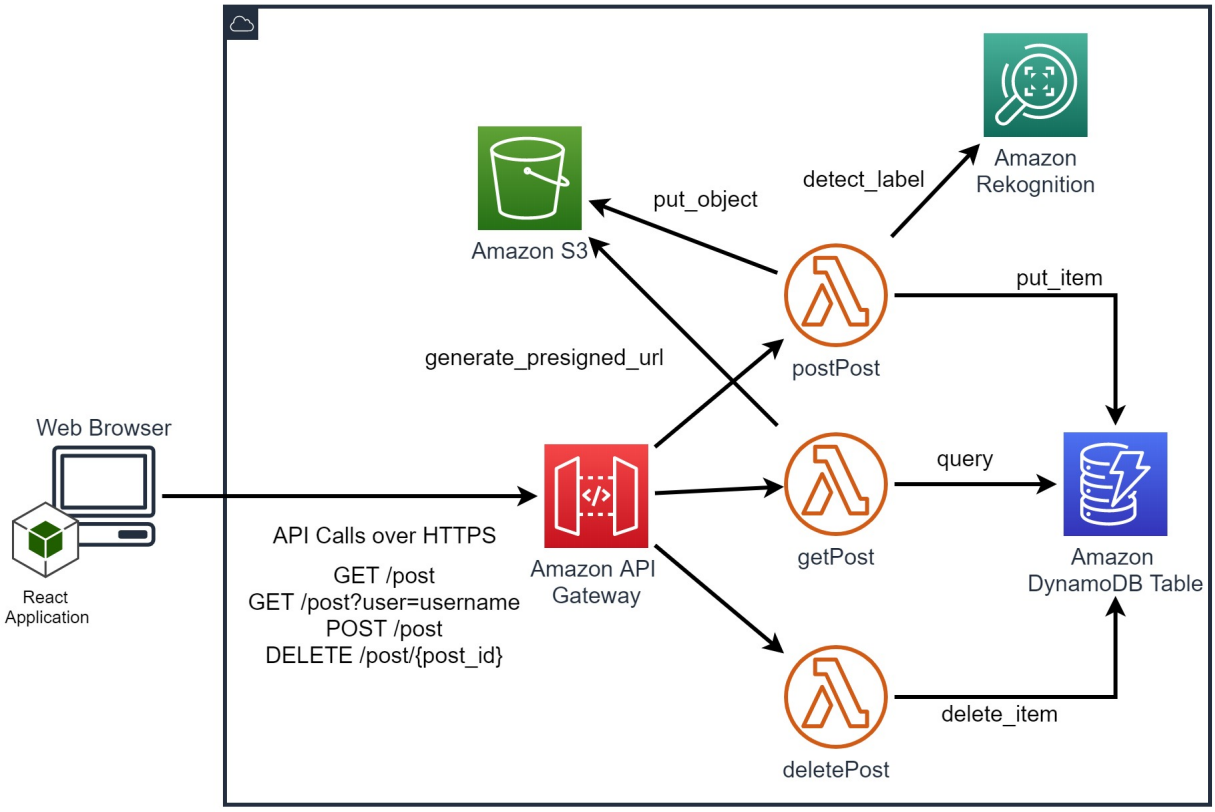
TP noté 2 : un web service serverless

Sujet

Dans ce devoir noté vous allez concevoir la partie backend du nouveau réseau social de partage de photo, **Postagram**. Voici le diagramme d'architecture de l'application.



Le service est accessible via une API gateway qui répond à plusieurs endpoints :

- POST /posts pour créer une publication
- DELETE /posts/{id} pour supprimer une publication
- GET /posts pour lire toutes les publications stockées en base
- GET /posts?user=username pour lire toutes les publications d'un utilisateur
- GET /getSignedUrlPut qui va retourner une url signée pour pouvoir uploader une image

En plus de cela un mécanisme de détection des labels des images est mis en place dans l'application. Dés qu'une image est uploadée sur amazon S3 une lambda est déclenchée pour l'analyser via le service Amazon Rekognition et les labels retournés sont stockés dans la base dynamoDB

Attendu

Votre but n'est pas de réaliser l'intégralité de l'application, mais seulement la partie création d'une publication et détection des labels via le service Rekognition, récupération des publications et suppression des publications. Le reste vous sera déjà donnée. Ainsi vous avez à votre disposition:

- Un application web écrite en Reactjs (le dossier `webapp`) qui va communiquer avec votre application. Aucune connaissance en Reactjs n'est attendue, ce code est uniquement là pour requêter votre webservice. Vous avez néanmoins à modifier la ligne 12 du fichier `index.js` pour mettre à la place l'adresse de votre API gateway. Attention l'url ne doit pas avoir de / à la fin !! Pour lancer cette application placez vous dans le dossier `webapp` et faite `npm start`
- Un projet Amazon SAM déjà initialisé. Vous allez devoir modifier ce projet car c'est lui qui va contenir votre application ! Le fichier template.yaml contient déjà certains éléments. Ne les modifiez pas ! Surtout la partie `Cors`. Sans elle, votre API gateway va bloquer les appels de votre application.
- La fonction lambda qui permet de générer une url présignés pour uploader les images est déjà faite.

Le code est à récupérer avec un `git clone git@github.com:HealerMikado/postagram_ensai.git`

Cet exercice est à faire par groupe de 3 max. Vous pouvez ainsi le faire seul à deux ou à trois. Vous noterez les membres du groupe dans un fichier `groupe.md` et ce même si vous êtes seul ! Vous rendrez une Moodle une archive .zip contenant le code du projet SAM. **Attention votre code doit fonctionner tel quel**. C'est à dire qu'il suffit de faire un `sam deploy --guided` pour tout lancer.

Si vous faite ce projet en groupe, je vous encourage à rapidement mettre en place un dépôt git et à travailler en parallèle.

Voici le macro barème qui sera appliqué si vous êtes 3 :

- Vous avez tout qui fonctionne correctement : 20
- Vous avez les fonctionnalités de base qui fonctionnent sans la génération url signées pour l'affichage des images et la détection des labels : 16
- Vous avez la possibilité de poster et afficher des publications : 14
- Vous avez seulement la partir création de publication : 10

Je pars du principe que le code python est propre à chaque fois et que le template SAM fonctionne. Je n'attends pas des commentaires, mais un code lisible.

Si vous faites ce travail seul ou à deux cela sera pris en compte. Considérez que si vous êtes seul, faire la fonctionnalité de création de publication avec détection des labels vaut un 20. Si vous êtes à deux le 20 il faut ajouter la partie récupération des publications.

✨Aides

Ce projet contient des choses que vous avez déjà vu, ainsi que des choses nouvelles. Voici pour vous aider de nombreux exemple de code. N'hésitez pas retourner dans le cours ou aller sur internet pour vous aider. Bien entendu ce ne sont que des aides, et pas la solution à l'exercice.

🛑 Les rôles des lambdas

Il n'est pas possible de passer la champ `Role: !Sub arn:aws:iam::${AWS::AccountId}:role/LabRole` dans les globals du template. Pensez à ajouter ce champ à chaque lambda !

Base Dynamodb

Votre base Dynamodb aura comme clé de partition les utilisateurs, et comme clé de trie l'id des tâches. Pour éviter tout chevauchement entre les concepts, je valoriserai les groupes qui préfixent ses deux attributs comme dans le TP 4.

Voici un rappel des méthodes qui vous seront utiles :

```
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table(table_name)

data = table.put_item(
    Item={...})

data = table.delete_item(
    Key={'name' : 'jon',
    "lastname" : 'doe'}
)

table.update_item(
Key={
    "name": "jon",
    "lastname": "doe"
},
AttributeUpdates={
    "tel": {
        "Value": "12345678",
        "Action": "PUT"
    }
},
ReturnValues='UPDATED_NEW'
)
data = table.scan()

data = table.query(
    KeyConditionExpression=Key('user').eq('jon' & Key("lastname".eq("doe")))
)


```

✅Les retours des lambdas

Pour que l'application Reactjs fonctionne elle attend un certain type de retour. Voici à quoi devra ressembler vos retours. Le status code 200 permet de dire que tout est ok, le header `'Access-Control-Allow-Origin': '*'` est la pour que votre navigateur ne bloque pas la réponse, et le body contiendra votre réponse.

```python
import logging
import json

logger = logging.getLogger()
logger.setLevel(logging.INFO)
def lambda_handler(event, context):
    # Code de la fonction
    # ...
    response = {
            "statusCode": 200,
            "headers": {
            'Access-Control-Allow-Origin': '*'
            },
            "body": # Dépend de la fonction
        }

    logger.info(f'response from: ${event["path"]} statusCode: ${response["statusCode"]} body: {response["body"]}')
    return response
```

A noter que j'ai ajouter aussi un logger. Vous pouvez vous en sortir avec des `print()` mais un logger est plus propre.

 Récupérer le username

Pour simuler une vraie application, le nom de l'utilisateur sera, sauf mention contraire, récupéré dans le header de la requête. En effet, les utilisateurs authentifiés envoient à chaque requête un jeton avec diverses information, dont leur username. Dans notre cas pour simplifier, ce n'est pas un jeton qui va être envoyé, mais simplement le username dans un header de la requête.

Ainsi pour récupérer le username vous allez devoir faire :

```python
user = event["headers"]["Authorization"]
```

Cela sera utile pour :

- poste une publication
- supprimer une publication

Poster une publication

Voici le corps de la requête pour poster une publication que vous enverra l'application Reactjs :

```
{
   'title' : string,
   'body' : string,
}
```

Il vous faut mettre cela en base en calculant un id pour la publication. Le username est à récupérer dans le header comme dit ci-dessus.

Pour le retour attendu :

```
data = table.put_item(...)
response = {
        "statusCode": 200,
        "headers": {
        'Access-Control-Allow-Origin': '*'
        },
        "body":json.dumps(data)
    }
```

Bucket S3 et détection des labels

Dans le template fourni, un bucket S3 est déjà définit. La détection des labels sera exécutée dés qu'un objet sera uploadé sur S3. Pour faire cela, la lambda ne doit pas être déclenchée par un appel API mais par la création d'un objet sur S3. Voici un exemple pour vous aider :

```
  S3EventFunctionFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: src/handlers/s3event
      Runtime: python3.9
      Handler: app.lambda_handler
      Events:
        ObjectCreatedEvent:
          Type: S3
          Properties:
            Bucket: !Ref UploadsBucket
            Events: s3:ObjectCreated:*
```

Le code qui gère l'uploade des fichiers les met dans votre bucket à l'adresse : `user/id_publication/image_name`. Comme votre fonction est déclenchée par l'ajout dans objet dans un bucket, l'event va être différent de celui d'un appel API. Voici pour vous aider les premières lignes de code de cette lambda

```
import boto3
import os
```

```
import logging
import json
from datetime import datetime
from urllib.parse import unquote_plus

table_name = os.getenv("TASKS_TABLE")
dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table(table_name)
reckognition = boto3.client('rekognition')
logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
    # Pour logger
    logger.info(json.dumps(event, indent=2))
    # Récupération du nom du bucket
    bucket = event["Records"][0]["s3"]["bucket"]["name"]
    # Récupération du nom de l'objet
    key = unquote_plus(event["Records"][0]["s3"]["object"]["key"])
    # extration de l'utilisateur et de l'id de la tâche
    user, task_id = key.split('/')[:2]
```

Il vous faut maintenant appeler le service Rekognition pour obtenir les labels de l'image. Voici un exemple de code

```
# Appel au service, en passant l'image à analyser (bucket et key)
# On souhaite au maximum 5 labels et uniquement les labels avec un taux de confiance > 0.75
# Vous pouvez faire varier ces valeurs.
label_data = reckognition.detect_labels(
        Image={
            "S3Object": {
                "Bucket": bucket,
                "Name": key
            }
        },
        MaxLabels=5,
        MinConfidence=0.75
    )
logger.info(f"Labels data : {label_data}")
# On extrait les labels du résultat
labels = [label["Name"] for label in label_data["Labels"]]
logger.info(f"Labels detected : {labels}")
```

Puis il vous faut enfin insérer les labels et la localisation de l'image dans s3 dans la base Dynamodb avec la méthode update_item. Vous stockerez en base la contenu de la variable `key` qui est en quelque sort le chemin de l'image dans votre bucket.

Récupérer des publications

Cette fonctionnalité demande simplement d'aller récupérer les données stockées dans la base. Attention néanmoins car un *query parameter* peut être utilisé pour récupérer les posts d'un utilisateur en particulier. Récupérer un query parameter va utiliser la clef `queryStringParameters` de l'event. Exemple pour récupérer le query parameter `toto` :

```
def handler(event, context):
    print(event["queryStringParameters"]["toto"])
```

Votre code va devoir gérer les deux cas. Je vous conseil pour une meilleur lisibilité de faire deux méthodes qui seront appelées dans le handler de la lambda en fonction du cas dans lequel vous vous trouvez. L'application web attend une réponse qui contient une liste de post, chaque poste ayant les informations suivante

```
{
   'user' : string,
   'title' : string,
   'body' : string,
   'image' : string //contient l'url de l'image,
   'label' : liste de string
}
```

Pour obtenir l'url de l'image vous allez utiliser une url dite signée. En effet comme votre bucket est privé, pour rendre accessible les images, il faut créer une url spéciale. Voici le code pour générer cette url :

```
bucket = os.getenv("S3_BUCKET")
s3_client = boto3.client('s3')

url = s3_client.generate_presigned_url(
    Params={
    "Bucket": bucket,
    "Key": object_name,
},
    ClientMethod='get_object'
)
```

Pour une publication donnée vous trouverez l'image dans le bucket de votre application au chemin suivant : `user/id_publication/image_name`

Pour le retour attendu :

```
data = table.query(...)
response = {
        "statusCode": 200,
        "headers": {
        'Access-Control-Allow-Origin': '*'
        },
        "body":json.dumps(data["Items"])
    }
```

❌Supprimer des posts

Enfin la suppression des posts va appeler la méthode `delete_item` présente dans le CM4. Pour ajouter un *path parameter* il vous faut modifier légèrement la clef path dans le yaml de défintion de l'architecture. Par exemple :

```
  GetTaskByIdFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: src/handlers/getTaskById
      Runtime: python3.9
      Handler: app.lambda_handler
      Environment:
        Variables:
          TASKS_TABLE: !Ref TasksTable
      Events:
        GetByIdFunctionApi:
          Type: Api
          Properties:
            RestApiId: !Ref TasksApi
            Path: /tasks/{id}
            Method: GET
```

Ensuite pour récupérer cette valeur dans votre lambda vous allez devoir faire :

```
def lambda_handler(event, context):
    logger.info(f"event : {event}")
    id = event["pathParameters"]["id"]
```

La subtilité de la suppression est que vous allez devoir supprimer la publication de la base de donnée, mais aussi supprimer son image dans le bucket S3 si elle en a une.

Pour le retour attendu :

```
data = table.delete_item()
response = {
    "statusCode": 200,
    "headers": {
        'Access-Control-Allow-Origin': '*'
    },
    "body": json.dumps(data)
}
```