Python documentation

1. Introduction

This document provides a complete guide to the backend logic of this project, implemented through AWS Lambda functions using Python. Each Lambda function is described in detail, explaining its role in the application, interaction with AWS services like DynamoDB, and response handling. This documentation serves as a walkthrough for anyone interested in understanding the backend processes that power the application's serverless architecture. By the end, readers will grasp how these Lambda functions are configured, why they were chosen, and how they enable a scalable, cost-effective serverless solution.

2. Story

Developing a serverless application involves building a reliable backend infrastructure that can handle various tasks seamlessly, from user data storage to handling shopping cart operations. For this project, AWS Lambda functions in Python were chosen to create an efficient and scalable backend. These Lambda functions interact with DynamoDB to support core functionalities like product listing, user data management, and cart operations, all without the need for a dedicated server.

Using AWS Lambda ensures that the backend scales automatically with demand while only charging for compute time used, making it both efficient and cost-effective. Each Lambda function is tailored for a specific task—whether it’s querying product data, storing user information, or managing a shopping cart—creating a modular, manageable backend that can evolve alongside the application. By leveraging Python's flexibility and the power of AWS services, the backend is optimized for reliability and performance, providing a robust foundation for the application.

Python was chosen for the Lambda functions due to its simplicity, readability, and extensive library support, especially in serverless and cloud environments. With its concise syntax, Python enables rapid development and straightforward debugging, which speeds up iteration during development. Additionally, Python's boto3 library is specifically optimized for AWS, providing seamless integration with services like DynamoDB. These advantages make Python a natural choice over other languages, ensuring that the backend remains both efficient and easy to maintain as the application scales.

3. Design

Here I will explain every line in code what it does and why I need it for my project.

- **lambda_home_page.py** - function created to get ads from fb4u_ads database and sent it to the frontend.
    - It is connecting to the database using boto3 SDK.
    - It is scanning through entire fb4u_ads database to get every item.
    - It is choosing 3 at random and append their details into the list and them return it to the frontend with status code 200.

```python
import simplejson as json
import boto3
import random

def lambda_handler(event, context): # Lambda handler function, called when the Lambda is triggered by an event

    dynamodb = boto3.resource('dynamodb') # Create a DynamoDB resource
    table = dynamodb.Table('fb4u_ads') # Connect to the DynamoDB table

    response = table.scan() # Scan the DynamoDB table for all items
    items = response['Items'] # Get the items from the response

    selected_items = random.sample(items, 3) # Select 3 random items from the items list

    ads = []
    for item in selected_items: # Iterate through the selected items, get the most important data and append it to the ads list
        ad_id = item['ad_id']
        ad_label = item['ad_label']
        ad_photo = f'{ad_id}.png'

        ads.append({
            'ad_id': ad_id,
            'ad_label': ad_label,
            'ad_photo': ad_photo
        })

    return {
        'statusCode': 200,
        'body': json.dumps(ads) # Return the ads list as JSON
    }
```

Home page code. You can see usage of boto3 sdk used to query and get data from the database and send it to the frontend.

- **lambda_load_products.py** - function to query data from fb4u_products based on tag or id parameter.
    - It is connecting to the database using boto3 SDK.
    - It is getting from the frontend the QueryParamater (either tag or product_id) and queries the table based on that.
    - Call out a function that adds specific product variables from the queried data to productReponse list.
    - Return productResponse with 200 code.
    - If there is any problem return status code 500 with error message.

```python
import simplejson as json
import boto3 # AWS SDK for Python
from boto3.dynamodb.conditions import Key, Attr

def lambda_handler(event, context): # Lambda handler function, called when the Lambda is triggered by an event
    dynamodb = boto3.resource('dynamodb') # Create a DynamoDB resource
    table = dynamodb.Table('fb4u_products') # Connect to the DynamoDB table
    operation_bool = False # Set the operation boolean to false
    try:
        query_params = event['queryStringParameters'] # Get the query string parameters
        print(query_params)

        if 'product_tag' in query_params:
            product_tag = query_params['product_tag'] # Get the product tag from the query string
            response = table.query(IndexName='fb4u_tag', KeyConditionExpression=Key('tag').eq(product_tag)) # Query the DynamoDB tab
            data = response["Items"] # Get the items from the response
        elif 'product_id' in query_params:
            product_id = query_params['product_id'] # Get the product id from the query string
            response = table.get_item(Key={'product_id': product_id}) # Get the item with the product id from the DynamoDB table
            data = response["Item"] # Get the items from the response
            operation_bool = True # Set the operation boolean to true
        else:
            raise ValueError("Invalid query parameters")      alissento, 2 weeks ago • rewriting project to vue

        products = json.loads(json.dumps(data)) # Convert the items to JSON
        print(products)

        productResponse = []

        if operation_bool: # If the operation boolean is true, return the product as a single item
            product = products
            productResponse = response_creation(product)
        else: # If the operation boolean is false, return the products as a list
            for product in products: # Iterate through the kits, get the most important data and append it to the products list
                productResponse += response_creation(product)
```

The main part of the code. It uses boto3, querying data dependent on queryStringParameter and calling out the response_creation function.

```python
    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
        },
        'body': json.dumps(productResponse) # Return the products list as JSON
    }

except Exception as e:
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
        },
        'body': json.dumps({'error': str(e)}) # Return an error message as JSON
    }
```

Returning 200 code with the productResponse when everything goes well, otherwise return 500 with an error message.

```python
def response_creation(product): # Function to create the response for the products
    productResponse = []

    product_id = product['product_id']
    productTag = product['tag']

    if productTag == 'kits':
        team_name = product['team_name']
        season = product['season']
        side = product['side']
        productLabel = f"{team_name} {side} {season}"
    elif productTag == 'boots':
        brand = product['brand']
        label = product['label']
        productLabel = f"{brand} {label}"

    price = product['price']
    photoID = f"{product_id}.png"
    productDescription = product['description']

    productResponse.append({
        'productLabel': productLabel,
        'price': price,
        'photoID': photoID,
        'product_id': product_id,
        'productDescription': productDescription,
        'productTag': productTag
    })

    print(productResponse)

    return productResponse
```

Response_creation function. It creates variables dependant on the product tag and appends them to the productReponse list.

- **lamda_store_user_data.py** - function to store additional data after user registration or details update in dynamodb table “fb4u_users”.
    - It is connecting to the database using boto3 SDK.
    - Loads up data and queryStringParameter from header sent by JavaScript.
    - Depending on the queryStringParameter (it is a register or update operation) it extracts data to specific variables and then puts or updates the user record in the table.
    - After that, send an 200 status code with a “User details stored successfully” message.
    - After a failed operation, return an 500 status code with an error message.

```python
import simplejson as json
import boto3
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError

def lambda_handler(event, context): # Lambda handler function, called when the Lambda is triggered by an event
    dynamodb = boto3.resource('dynamodb') # Create a DynamoDB resource
    table = dynamodb.Table('fb4u_users') # Connect to the DynamoDB table

    try:
        print("Before body")
        body = json.loads(event['body'])
        print(body)
        operation = event['queryStringParameters']['operation']

        if operation == 'register':
            user_id = body['user_id']
            email = body['email']
            first_name = body['firstName']
            last_name = body['lastName']

            response = table.put_item(Item={
                'user_id': user_id,
                'email': email,      alissento, 13 hours ago • why i am doing this to myself
                'first_name': first_name,
                'last_name': last_name
            })

            print(response)
        elif operation == 'update':
            user_id = body['user_id']
            update_expression = "SET " + ", ".join([f"{key} = :{key}" for key in body if key != 'user_id'])
            expression_attribute_values = {f":{key}": value for key, value in body.items() if key != 'user_id'}

            response = table.update_item(
                Key={
                    'user_id': user_id
                },
                UpdateExpression=update_expression,
                ExpressionAttributeValues=expression_attribute_values
            )

            print(response)
```

Main part of the code. It is using boto3 to set up table, loads up data sent by frontend, and either creates or updates user details in the table.

```python
        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps({'message': 'User details stored successfully'})
        }

    except ClientError as e:
        return {
            'statusCode': 500,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps({'error': str(e)})
        }

    except Exception as e:
        return {
            'statusCode': 500,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps({'error': str(e)}) # Return an error message as JSON
        }
```

When the operation is successful, return the 200 code with a successful message. Otherwise, return 500 code with an error message

- **lambda_get_user_data.py** - queries user data based on QueryParameter with a user_id and returns them to the JavaScript.
    - It is connecting to the database using boto3 SDK.
    - Gets user_id from QueryParameter
    - Queries user data based on user_id and if there is existing data in the database, return it to the JavaScript otherwise give a 404 error user not found.
    - If there are any other problems return the 500 code with an error message.

```python
import simplejson as json      alissento, 3 days ago • added fully functiona
import boto3
from botocore.exceptions import ClientError

def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('fb4u_users')

    try:
        user_id = event['queryStringParameters']['user_id']

        try:
            response = table.get_item(Key={'user_id': user_id})
            if 'Item' in response:
                user_data = response['Item']
                return {
                    'statusCode': 200,
                    'headers': {
                        'Access-Control-Allow-Origin': '*',
                        'Access-Control-Allow-Headers': 'Content-Type',
                        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
                    },
                    'body': json.dumps(user_data)
                }
            else:
                return {
                    'statusCode': 404,
                    'headers': {
                        'Access-Control-Allow-Origin': '*',
                        'Access-Control-Allow-Headers': 'Content-Type',
                        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
                    },
                    'body': json.dumps({'error': 'User not found'})
                }
        except Exception as e:
            print("Error:", str(e))
            raise e
```

Very simple functionality. Based on user_id it queries the users' data and sends it to the frontend

```
except ClientError as e:
    print("ClientError:", str(e))
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps({'error': str(e)})
    }

except Exception as e:
    print("Exception:", str(e))
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps({'error': str(e)})
    }
```

This part of the code returns an error if anything goes bad.

- **lambda_add_to_cart.py** - function created to add specific products to the users cart table.
    - It is connecting to the database using boto3 SDK.
    - Gets data from JavaScript from a header and extracts it into specific variables.
    - Queries current user cart data. If the cart is empty (no user data) then create a new cart data based on user_id and items variables from a header.
    - If the cart is not empty, simply add a new item to the items list.
    - If everything works, return 200 status code with a successful message. Otherwise, return 500 status code with an error message.

```python
import simplejson as json
import boto3 # AWS SDK for Python
from boto3.dynamodb.conditions import Key, Attr

def lambda_handler(event, context): # Lambda handler function, called when the Lambda is triggered by an event
    dynamodb = boto3.resource('dynamodb') # Create a DynamoDB resource
    table = dynamodb.Table('fb4u_cart') # Connect to the DynamoDB table

    try:
        body = json.loads(event['body']) # Parse the JSON body of the event
        print(f"Body: {body}")
        user_id = body['user_id']
        product_id = body['product_id']
        size = body['size']

        try: # Try to get the user's cart from the DynamoDB table
            response = table.get_item(Key={'user_id': user_id})
            print(f"Response: {response}")

            if 'Item' in response: # If the user's cart exists, add the product to the cart
                cart = response['Item']
                print(f"Cart: {cart}")
                if 'items' not in cart:
                    cart['items'] = []

                cart['items'].append({
                    'product_id': product_id,
                    'size': size
                })

                table.put_item(Item=cart)
            else: # If the user's cart does not exist, create a new cart with the product
                table.put_item(Item={
                    'user_id': user_id,
                    'items': [{
                        'product_id': product_id,
                        'size': size
                    }]
                })
        except Exception as e: # Catch exceptions from the DynamoDB client
            print(f"Error: {e}")
            raise e
```

Main part of the code. It is using the boto3 to get table from AWS. Then queries based on the body data to check if a cart exists or not. Then either add items to already existing cart or create a new cart with the items.

```python
    return { # Return a 200 response if the product is added to the cart successfully
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps({'message': 'Product added to cart successfully'})
    }
except Exception as e: # Catch all other exceptions
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps({'error': str(e)})
    }
```

Returns either a successful message or an error.

- **lambda_load_cart.py** - loads specific user carts to display them in the summary page.
    - It is connecting to the databases using boto3 SDK.
    - Gets user_id from QueryParameter.
    - Queries the cart table for users' cart data. If it is empty it will return 200 codes with an empty list (JavaScript will handle the rest).
    - Extract cart data into variables and query the products table based on product_id to get more product details.
    - Based on product_tag extract specific variables and append them into a list.
    - Return 200 status codes with the list. If anything fails, return the 500 code with an error message.

```python
import simplejson as json
import boto3 # AWS SDK for Python
from boto3.dynamodb.conditions import Key, Attr

def lambda_handler(event, context): # Lambda handler function, called when the Lambda is triggered by an event
    dynamodb = boto3.resource('dynamodb') # Create a DynamoDB resource
    tableCart = dynamodb.Table('fb4u_cart') # Connect to the DynamoDB table
    tableProducts = dynamodb.Table('fb4u_products') # Connect to the DynamoDB table

    try:
        user_id = event['queryStringParameters']['user_id']

        try:
            responseCart = tableCart.get_item(Key={'user_id': user_id})
            print(f"Response cart: {responseCart}")
            cartData = responseCart['Item']
            print(f"Cart data: {cartData}")      alissento, 2 days ago • -fully working login and registering
            usersCart = []
        except Exception as e:
            print(f"Error: {e}")
            usersCart = []

            return {
                'statusCode': 200,
                'headers': {
                    'Access-Control-Allow-Origin': '*',
                    'Access-Control-Allow-Headers': 'Content-Type',
                    'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
                },
                'body': json.dumps(usersCart) # Return the products list as JSON
            }
```

This part of the code queries the cart data from DynamoDB and if anything goes wrong like the cart does not exist it will return an empty userCart list to the frontend.

```
for product in cartData['items']:
    product_id = product['product_id']
    size = product['size']

    response = tableProducts.get_item(Key={'product_id': product_id})
    print(f"Response1: {response}")
    responseProducts = response['Item']
    print(f"Response products: {responseProducts}")

    productTag = responseProducts['tag']

    if productTag == 'kits':
        team_name = responseProducts['team_name']
        season = responseProducts['season']
        side = responseProducts['side']
        productLabel = f"{team_name} {side} {season}"
    elif productTag == 'boots':
        brand = responseProducts['brand']
        label = responseProducts['label']
        productLabel = f"{brand} {label}"

    price = responseProducts['price']
    photoID = f"{product_id}.png"

    usersCart.append({
        'productLabel': productLabel,
        'price': price,
        'photoID': photoID,
        'product_id': product_id,
        'size': size,
        'user_id': user_id
    })

    print(f"Users cart: {usersCart}")
```

For a function to iterate through every product in the cart data from the products table to get product information. Necessary to construct a better-looking cart.

```
    return {
        'statusCode': 200,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps(usersCart) # Return the products list as JSON
    }
except Exception as e:
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps({'error': str(e)})
    }
```

If everything goes well return a 200 code with user's cart, otherwise return a 500 code with an error.

- **lambda_clear_cart.py** - function to delete user cart from the dynamodb table.
    - It is connecting to the databases using boto3 SDK.
    - Gets user_id from a header.
    - Deletes the item based on user_id from the cart table.
    - Return 200 status code with the list. If anything fails, return the 500 code with an error message.

```python
import simplejson as json
import boto3 # AWS SDK for Python
from boto3.dynamodb.conditions import Key, Attr

def lambda_handler(event, context): # Lambda handler function, called when the Lambda is triggered by an event
    dynamodb = boto3.resource('dynamodb') # Create a DynamoDB resource
    table = dynamodb.Table('fb4u_cart') # Connect to the DynamoDB table

    try:
        body = json.loads(event['body'])
        print(body)
        user_id = body['user_id']      alissento, 2 days ago • -fully working login and registering functional…
        print(user_id)

        try:
            response = table.delete_item(Key={'user_id': user_id}) # Delete the item from the DynamoDB table
            print(f"Response: {response}")

            return {
                'statusCode': 200,
                'headers': {
                    'Access-Control-Allow-Origin': '*',
                    'Access-Control-Allow-Headers': 'Content-Type',
                    'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
                },
                'body': json.dumps({'message': 'Cart cleared successfully'})
            }
        except Exception as e:
            print(f"Error: {e}")
            raise e

    except Exception as e:
        return {
            'statusCode': 500,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps({'message': 'Error clearing cart'})
        }
```

Very simple functionality that gets user_id from the body, and deletes the user record from the cart database.

- **lambda_checkout.py** - function to generate order data and create it in the orders database and clear the user cart.
    - It is connecting to the databases using boto3 SDK.
    - Gets user_id from a body.
    - Queries cart table based on user_id and gets products from the users' cart.
    - Generates custom ID, current date and specific status for the order information and appends cart items to the order_items list.
    - After that, it puts all order data into the order table and clears the user cart.
    - Return 200 status code with a successful message. If anything fails, return the 500 code with an error message.

```python
import simplejson as json      alissento, 14 hours ago • why i am doing
import boto3
from boto3.dynamodb.conditions import Key
import uuid  # For generating unique order IDs
from datetime import datetime  # For adding the order date

def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    tableCart = dynamodb.Table('fb4u_cart')
    tableOrders = dynamodb.Table('fb4u_orders')

    try:
        body = json.loads(event['body'])
        user_id = body.get('user_id')

        try:
            responseCart = tableCart.get_item(Key={'user_id': user_id})
            cartData = responseCart.get('Item', {})
            if not cartData or 'items' not in cartData:
                raise Exception("Cart is empty or invalid.")
        except Exception as e:
            return {
                'statusCode': 400,
                'headers': {
                    'Access-Control-Allow-Origin': '*',
                    'Access-Control-Allow-Headers': 'Content-Type',
                    'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
                },
                'body': json.dumps({'error': str(e)})
            }

        order_id = str(uuid.uuid4())
        order_date = datetime.now().strftime('%d-%m-%Y')
        order_status = 'Pending'
```

This part of the code queries the cart data based on user_id and generates custom order information.

```
order_items = []
for product in cartData['items']:
    product_id = product['product_id']
    size = product['size']

    order_items.append({
        'product_id': product_id,
        'size': size
    })

order_data = {
    'order_id': order_id,
    'user_id': user_id,
    'items': order_items,
    'order_date': order_date,
    'order_status': order_status
}

tableOrders.put_item(Item=order_data)
tableCart.delete_item(Key={'user_id': user_id})

return {
    'statusCode': 200,
    'headers': {
        'Access-Control-Allow-Origin': '*',
        'Access-Control-Allow-Headers': 'Content-Type',
        'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
    },
    'body': json.dumps({'message': 'Order placed successfully!', 'order_id': order_id})
}
```

This part of the code adds all product info like product_id and size to the order_items and puts whole data about the order into the table. Then it clears the user's cart and returns a successful message.

```
except Exception as e:
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps({'error': str(e)})
    }
```

Return an error message if anything goes bad.

- **lambda_get_orders.py** - simple function to get users orders list and send them to the frontend
    - Connects to the database with boto3.
    - Loads user_id from query string parameter and uses is to query the database with that specific user_id.
    - Returns the orders to the frontend with 200 code. Otherwise return 500 code with an error message.

```python
import simplejson as json
import boto3
from boto3.dynamodb.conditions import Key

def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    tableOrders = dynamodb.Table('fb4u_orders')

    try:
        user_id = event['queryStringParameters']['user_id']
        if not user_id:
            return {
                'statusCode': 400,
                'headers': {
                    'Access-Control-Allow-Origin': '*',
                    'Access-Control-Allow-Headers': 'Content-Type',
                    'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
                },
                'body': json.dumps({'error': 'Missing or invalid user_id'})
            }

        response = tableOrders.query(
            IndexName='fb4u_user_orders',
            KeyConditionExpression=Key('user_id').eq(user_id)
        )

        orders = response.get('Items', [])
        return {
            'statusCode': 200,
            'headers': {
                'Access-Control-Allow-Origin': '*',
                'Access-Control-Allow-Headers': 'Content-Type',
                'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
            },
            'body': json.dumps(orders)
        }
```

Main part of the code. Connects to the database and queries it with user_id. Then send it to the frontend with 200 code.

```python
except Exception as e:
    return {
        'statusCode': 500,
        'headers': {
            'Access-Control-Allow-Origin': '*',
            'Access-Control-Allow-Headers': 'Content-Type',
            'Access-Control-Allow-Methods': 'OPTIONS,POST,GET'
        },
        'body': json.dumps({'error': str(e)})
    }
```

If anything goes bad, return error message.

4. Conclusion

This documentation has outlined the design and purpose of the backend Lambda functions that form the backbone of this serverless application. By utilizing Python for AWS Lambda, the project achieves a streamlined, scalable, and cost-effective backend infrastructure. Each function is designed to handle specific tasks efficiently, ensuring modularity and maintainability. This Lambda-based backend not only simplifies infrastructure management but also enables the application to meet dynamic user demands seamlessly. As the application grows, this serverless approach provides the flexibility needed to scale and adapt, making AWS Lambda an invaluable component of the project’s architecture.