# Introduction

The repository hosts a serverless exchange rate tracking application developed in the AWS Lambda environment, deployable using AWS CDK. The application is designed to fetch exchange rate data from the [European Central Bank](https://www.ecb.europa.eu/stats/policy_and_exchange_rates/euro_reference_exchange_rates/html/index.en.html) and store it in a DynamoDB table. It's structured to update the exchange rates periodically and expose a REST API endpoint for fetching the latest exchange rates.

## Key Components:

* DynamoDB Table: Used for storing exchange rates.
* Lambda Functions: Two main Lambda functions are involved. One for periodically updating exchange rates, scheduled to run daily, and another for reading exchange rates from DynamoDB.
* API Gateway: Provides the latest exchange rates published by the ECB, along with the change with respect to the last published update.

The primary objectives of the serverless exchange rates tracking application are to provide a scalable, efficient, and automated solution for tracking currency exchange rates. Here's a detailed breakdown of the main objectives:

1. Automated Exchange Rates Retrieval: The application is designed to automatically fetch exchange rate data from the European Central Bank. This ensures that the data is always up-to-date and reflects the latest published rates.

2. Scheduled Updates: A Lambda function is scheduled to run daily at 17:00:00 UTC to update the exchange rates in the DynamoDB table. This scheduling ensures consistency and regularity in the data updates.

3. Data Storage: The application utilizes a DynamoDB table to store exchange rates. This provides a robust and scalable solution for data storage, allowing for efficient retrieval and management of exchange rates data.

4. REST API Exposure: The application exposes a REST API endpoint that provides the latest exchange rates, along with the change with respect to the last published update. This API can be consumed by other applications or services to access the exchange rate data.

6. Testing Framework: The codebase includes a testing framework to ensure the functionality and reliability of the application. This aids in maintaining code quality and detecting potential issues early in the development cycle.

7. Scalability and Efficiency: The application is designed to scale seamlessly with demand by leveraging serverless architecture and AWS services. This ensures that the application can handle varying loads without manual intervention.

In summary, the objectives of the application encompass automated data retrieval, scheduled updates, efficient storage, API exposure, testing support, and scalable architecture. These objectives align to provide a comprehensive solution for tracking and accessing currency exchange rates.

## Sample Output:

{

"update\_date":"2023-08-08",

"publish\_date":"2023-08-08",

"base\_currency":"EUR",

"exchange\_rates":[

{

"currency":"AUD",

"rate":"1.6694",

"change":"-0.0054",

"change\_percentage":"-0.3224 %"

},

{

"currency":"BGN",

"rate":"1.9558",

"change":"0.0",

"change\_percentage":"0.0 %"

},

……

# Architecture

The architecture of the serverless exchange rates tracking application is a blend of AWS services, serverless components, and custom code. It's designed to provide a seamless flow from data retrieval to data exposure. Let's delve into the architectural components and their relationships.

### Components:

1. Data Source - European Central Bank: The primary data source for the application is the European Central Bank. The application fetches exchange rate data from this external source.
2. AWS Lambda:

Update Exchange Rates Lambda: This function fetches the exchange rates from the European Central Bank and updates the DynamoDB table. It's scheduled to run daily at 17:00:00 UTC.

Get Exchange Rates Lambda: This function reads the exchange rates from the DynamoDB table and serves it through the REST API.

1. DynamoDB Table:

Exchange Rates Table: This table stores the exchange rate data. It's designed for efficient read-and-write operations, ensuring quick updates and retrievals.

1. AWS API Gateway:

Exchange Rates API: The application exposes a REST API endpoint through AWS API Gateway. This API provides the latest exchange rates and their changes. It's connected to the "Get ECB Exchange Rates Lambda" for data retrieval.

1. AWS CDK: The application uses AWS CDK for defining and provisioning the cloud resources. It provides an abstraction over AWS CloudFormation and allows for defining cloud resources using familiar programming languages.
2. AWS Eventbridge:

Scheduled Event: An event is scheduled to trigger the "Update ECB Exchange Rates Lambda" function daily. This ensures automated and regular updates to the exchange rate data.

1. Trigger Function: A trigger function populates initial exchange rate data in the table after deployment.

In essence, the architecture is a well-orchestrated combination of AWS services, serverless components, and custom code, working in harmony to provide an automated, scalable, and efficient solution for tracking currency exchange rates.

# Deployment Steps and Execution:

Deploying the serverless exchange rates tracking application involves a series of steps to ensure that all components are correctly provisioned and interconnected. Here's a step-by-step guide to deploying and executing the application:

## Deployment Steps:

1. Prerequisites:

1. Ensure you have the AWS Command Line Interface (CLI) installed and configured with the necessary access rights.
2. Install AWS CDK using the command: `npm install -g aws-cdk`
3. Python 3.8 or higher should be installed on your machine.
4. Install the required Python packages: `pip install -r requirements.txt`

2. Clone the Repository:

1. Clone the repository to your local machine using the command: `git clone https://github.com/FaizanSh/CCL\_task.git `
2. Navigate to the repository directory: `cd CCL\_task`

3. Bootstrap the CDK App:

1. If you haven't used CDK before with your AWS account and region, you need to bootstrap the environment: `cdk bootstrap`

4. Deploy the Application:

1. Deploy the application using the command: `cdk deploy`
2. This command will provision all the necessary AWS resources, including the Lambda functions, DynamoDB table, and API Gateway.

# Execution Steps:

### 1. Access the REST API:

Once the deployment is successful, the CDK will output the URL of the API Gateway endpoint. It will look something like:

`https://<unique-id>.execute-api.<region>.amazonaws.com/prod/ecbexchangerates`

You can use this URL to fetch the latest exchange rates.

### 3. Test the API:

Use a tool like `curl`, ‘requests’ or any API client like Postman to test the API.

Example using `curl`:

curl https://<unique-id>.execute-api.<region>.amazonaws.com/prod/ecbexchangerates`

This will return the latest exchange rates along with the change with respect to the last published update.

Example Using ‘Requests’

import requests

url = "https://<unique-id>.execute-api.us-east-1.amazonaws.com/prod/ecbexchangerates"

response = requests.get(url)

print(response)

if response.status\_code == 200:

data = response.json()

print(data)

else:

print("Request failed with status code:", response.status\_code)

### 4. Monitoring & Logs:

You can monitor the execution of the Lambda functions and view logs in the AWS Management Console under the AWS Lambda and Amazon CloudWatch sections, respectively.

5. Cleanup:

To avoid incurring unnecessary costs, you can destroy the provisioned resources once you're done testing.

Use the command: `cdk destroy` to remove all the resources associated with the application.

By following the above deployment and execution steps, you'll have the serverless exchange rates tracking application up and running. The REST API provides an easy way to fetch the latest exchange rates, and the architecture ensures that the data is always up-to-date and accurate.

# Tests

Testing is a critical aspect of any software development process, and in the context of the serverless exchange rates tracking application, it plays a vital role in validating the architecture created by AWS CDK. The tests are designed to ensure that the application's components are correctly defined, interconnected, and possess the required properties. Here's an overview of the testing approach and its significance:

## Test Objectives:

1. Validate CDK Architecture: The tests are aimed at confirming that the CDK architecture is defined as expected. This includes validating the properties of various AWS resources like Lambda functions, DynamoDB tables, API Gateway, and more.

2. Confirm Resource Properties: The tests confirm that the resources have the required properties, such as the correct runtime for Lambda functions, the correct partition key for the DynamoDB table, and the proper scheduling for the update function.

4. Automated Testing Framework: The tests are automated and can be run as part of a continuous integration (CI) pipeline. This ensures that any changes to the codebase are automatically validated, maintaining the integrity of the application.

## Test Execution:

1. Install Test Dependencies: Ensure that the required test dependencies are installed using the command: `pip install -r requirements.txt`

2. Run Tests: Execute the tests using a testing framework like pytest. The command to run the tests is: `python3 -m pytest -v`

3. Analyze Test Results: Review the test results to ensure that all tests have passed. Any failures will provide insights into discrepancies between the expected and actual architecture.

The tests provide a layer of quality assurance, ensuring that the code meets the expected standards and behaves as intended. By validating the architecture and properties, the tests help in early detection of errors, reducing the risk of deploying a faulty application. Thus, developers and operators can have confidence in deploying and scaling the application, knowing that the core architectural components have been validated.