**Project Documentation: Serverless Exchange Rate Tracker**

**Introduction**

This documentation outlines the architecture and functionality of a serverless application designed to monitor and retrieve exchange rate data from the European Central Bank (ECB). Developed within the AWS Lambda environment and deployable via AWS CDK, this application integrates with DynamoDB for data storage and schedules updates to ensure data remains current, exposing a REST API for data retrieval.

**Key Components Overview**

* **DynamoDB Table**: Serves as the primary storage for exchange rates, ensuring durability and scalability.
* **Lambda Functions**: Comprises two functions; one updates exchange rates daily, while the other retrieves data from DynamoDB.
* **API Gateway**: Facilitates access to the latest ECB exchange rates and provides updates on changes from the last retrieval.

**Detailed** Objectives

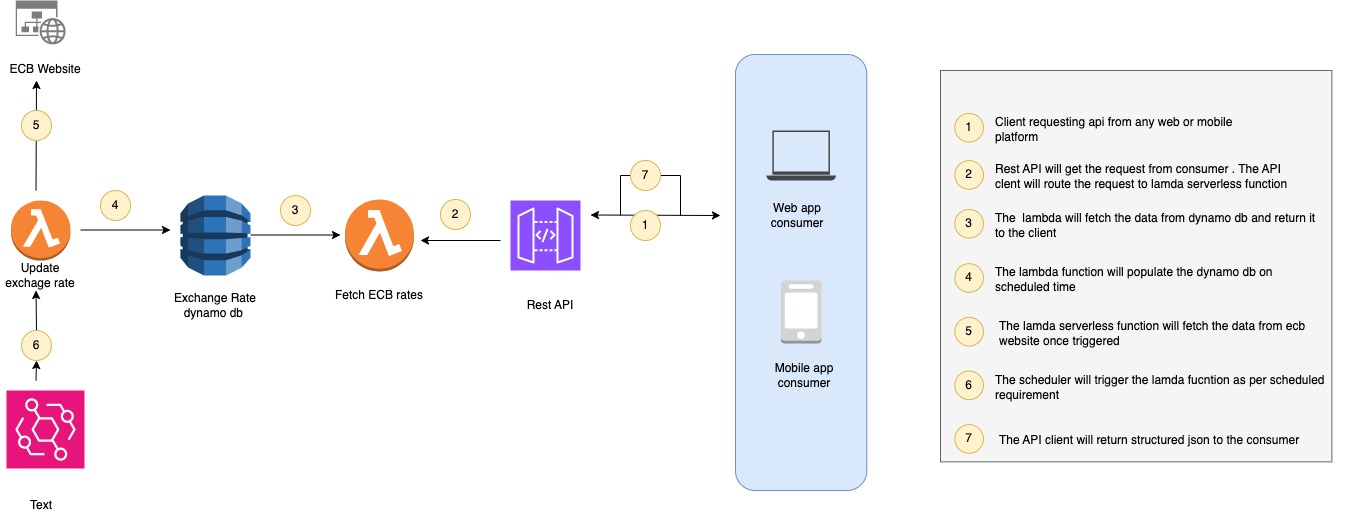
1. **Automated Exchange Rates Retrieval**
   * **Purpose**: Automates the fetching of exchange rates from the ECB to guarantee up-to-date information.
   * **Mechanism**: Utilizes scheduled Lambda functions to fetch and update data without manual intervention.
2. **Scheduled Updates**
   * **Schedule**: Executes a Lambda function daily at 17:00:00 UTC to refresh stored exchange rates.
   * **Consistency**: Ensures data regularity and timeliness in updates.
3. **Data Storage**
   * **Technology**: Employs DynamoDB, offering a robust and scalable data management solution.
   * **Benefits**: Facilitates efficient data retrieval and management, optimizing for high-performance access.
4. **REST API Exposure**
   * **Functionality**: Provides an endpoint for accessing the latest exchange rates and their recent changes.
   * **Integration**: Allows seamless consumption by other applications, supporting broader accessibility to the data.
5. **Testing Framework**
   * **Role**: Validates functionality and reliability through comprehensive testing.
   * **Benefits**: Helps maintain high code quality and early detection of potential issues, promoting reliable application performance.
6. **Scalability and Efficiency**
   * **Architecture**: Leverages the serverless model to scale effortlessly according to demand.
   * **Optimization**: Reduces the need for manual scaling and maintenance, enhancing operational efficiency.

**Project Objectives Summary**:

The system is engineered to achieve comprehensive automation in currency exchange rate monitoring. Key objectives include:

* **Automated Data Retrieval:** Direct integration with the European Central Bank to fetch the latest exchange rate data automatically.
* **Scheduled Updates:** Updates are systematically scheduled, ensuring data freshness daily at 17:00:00 UTC.
* **Efficient Data Storage:** Utilizes DynamoDB for high-efficiency data storage and retrieval.
* **API Accessibility:** Provides a REST API for easy access to the latest exchange rates.
* **Robust Testing Framework:** Incorporates a testing framework to guarantee system reliability and performance.
* **Scalable Architecture:** Designed to automatically scale with increased demand without manual intervention

**System Architecture**: The architecture of this serverless exchange rate tracking system integrates various AWS services and serverless components to streamline the flow from data retrieval to user accessibility.



**Architectural Components:**

1. **Data Source:**
   * **European Central Bank:** Acts as the primary source of exchange rate data, interfaced directly for updates.
2. **AWS Lambda Functions:**
   * **Update Exchange Rates Lambda:** Scheduled to fetch and update rates daily from the ECB.
   * **Get Exchange Rates Lambda:** Retrieves and serves current rates via the REST API.
3. **DynamoDB Table:**
   * **Exchange Rates Table:** Optimized for rapid read and write operations, ensuring swift data updates and retrievals.
4. **AWS API Gateway:**
   * **Exchange Rates API:** Exposes a RESTful endpoint for accessing the latest rates, interfacing with the Lambda function for data retrieval.
5. **AWS Cloud Development Kit (CDK):**
   * Facilitates cloud resource definition and provisioning, leveraging familiar programming paradigms for infrastructure as code.
6. **AWS Event Bridge:**
   * **Scheduled Event:** Triggers the daily update Lambda function, ensuring regular data refreshes.
   * **Trigger Function:** Initializes the DynamoDB table with data upon deployment.

**Deployment and Execution Steps** : Deployment involves several sequential steps to ensure all system components are properly configured and functional:

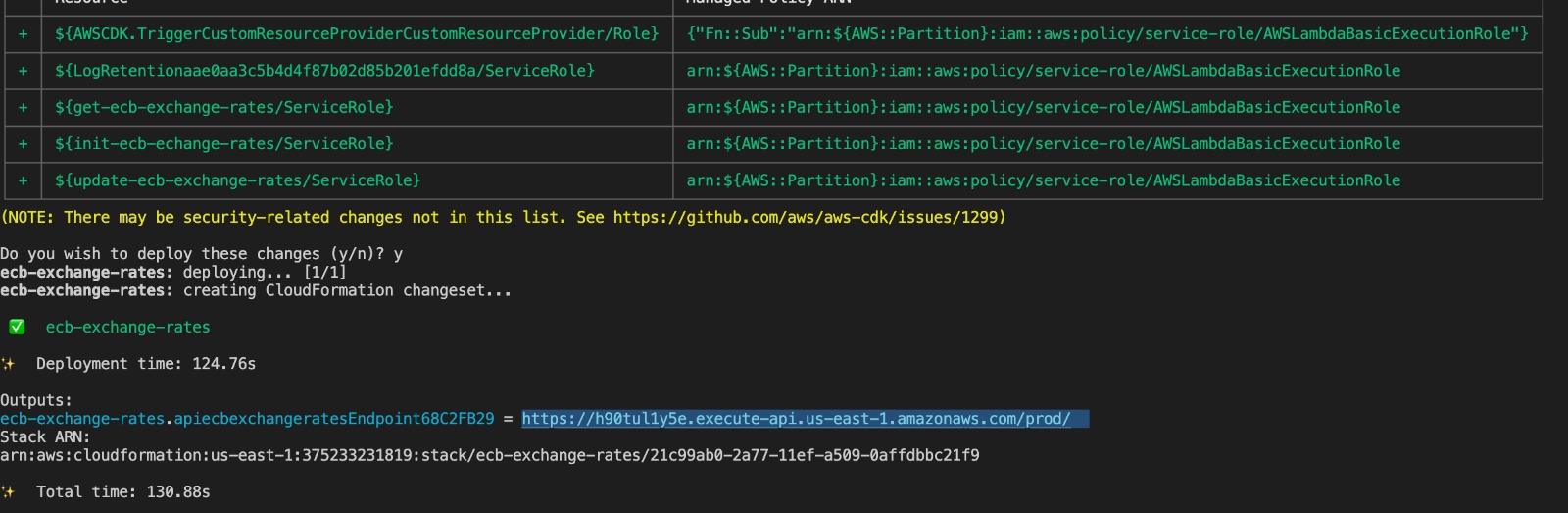
1. **Prerequisites:**

* Install AWS CLI and configure access.
* Install AWS CDK globally: npm install -g aws-cdk.
* Ensure Python 3.8+ is installed.
* Install Python dependencies: pip install -r requirements.txt.
  + **Repository Setup:**
* Clone the repository: git clone <https://github.com/subhani-git/CCL_ASSIGNMENT.git>
* Here is the link for repository
* https://github.com/subhani-git/CCL\_ASSIGNMENT
* Navigate to the project directory: cd CCL\_RESTAPI\_TASK.

1. **CDK Bootstrapping:**

Bootstrap the AWS environment if not previously done: cdk bootstrap.

1. **System Deployment:**
   * Deploy using AWS CDK: cdk deploy.
   * This step provisions all necessary AWS resources.

Out put for CDK should be something like this   
  


**Operational Guide**

1. **Accessing the API:** Post-deployment, use the provided API Gateway endpoint to query the latest exchange rates.
2. **Testing the API:** Utilize tools like curl or Postman to validate API functionality.
3. **Monitoring and Logs:** AWS Lambda and CloudWatch provide comprehensive monitoring and logging capabilities.
4. **Cleanup:** Use cdk destroy to decommission resources when necessary to avoid unnecessary costs.

**Execution Steps**

1. **Accessing the REST API:**

Post-deployment, the AWS Cloud Development Kit (CDK) will provide the URL for the API Gateway endpoint. This URL serves as your access point to retrieve the latest exchange rates. An example URL might look like this:

https://<uniqueid>.executeapi.<region>.amazonaws.com/prod/getecbexchangeres  
  
Utilize this URL to query the most recent exchange rate data.

1. **Testing the API:** To ensure the API's functionality, employ testing tools such as python requests library , Postman or curl:

Curl Example:

Curl https://<uniqueid>.executeapi.<region>.amazonaws.com/prod/getecbexchangerates

**Python Requests Example:**

import requests

url="https://<uniqueid>.executeapi.<region>.amazonaws.com/prod/getecbexchangerates"

response = requests.get(url)

if response.status\_code == 200:

data = response. json()

print(data)

else:

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

**Sample API Output**

{

'update\_date': '2024-06-14',

'publish\_date': '2024-06-14',

'base\_currency': 'EUR',

'exchange\_rates': [

{

'currency': 'AUD',

'rate': '1.6156',

'change': '-0.0076',

'change\_percentage': '-0.4682 %'

}

}

1. **Monitoring and Logging:** Monitor the performance and health of the Lambda functions via the AWS Management Console under AWS Lambda and Amazon CloudWatch sections. This will provide insights into operational aspects and potential improvements.  
     
   **4. Cleanup Procedure:** To avoid unnecessary charges, ensure to remove the

provisioned resources after testing:

**cdk destroy**

### This command decommissions all resources associated with the application to manage costs effectively.

### Testing Protocol

**Objective:** The testing framework is crucial for validating the integrity and performance of the serverless architecture designed with AWS CDK. These tests verify that all components are correctly defined and interconnected, and that they function as intended.

**Test Execution Steps:**

1. **Install Dependencies:** Ensure all necessary test dependencies are installed:  
     
   pip install -r requirements.txt
2. **Run Tests:** Utilize a framework like pytest to conduct the tests  
     
   python3 -m pytest -v
3. **Review Test Outcomes:** Analyze the results from the tests to confirm all components pass the expected criteria. Failures should be addressed promptly to ensure system reliability.

**Significance of Testing**   
  
Testing ensures that the application not only meets design specifications but also operates reliably under various conditions. By integrating these tests into a CI/CI pipeline, continuous validation of the application's functionality and performance is achieved, reinforcing the system's robustness and efficacy in production environments.

This documentation is crafted to provide clarity and comprehensive insights into the practical steps and testing procedures necessary for managing the serverless exchange rate tracking application effectively, tailored for a technical manager's requirements.