Serverless Development with AWS

Imagine a modern enterprise needing a flexible and efficient way to manage their API interactions – a system that could not only store and retrieve configurations for API calls but also trigger these calls on demand. They seek a solution that adapts quickly to their evolving needs, whether it's polling different APIs for weather forecasts or the latest news headlines.

In this hands-on workshop, you'll construct a robust serverless CRUD (Create, Read, Update, Delete) application using the AWS Cloud Development Kit (CDK) and various AWS services to fulfill this need. Your application will interact with a DynamoDB table designed to persist API settings such as endpoints, parameters, and additional attributes. By leveraging Lambda functions, you'll expose a REST API through API Gateway, allowing for seamless management of these settings.

You'll also implement a trigger mechanism to initiate API calls based on the stored configurations, capturing the essence of serverless scalability and responsiveness.

By the end of this workshop, you'll have a fully operational serverless application, ready to serve as a dynamic backbone for API interaction management, mirroring the adaptability required in the fast-paced digital landscape.

Prerequisites

We recommend to use Typscript as your CDK language. Therefore you can use the provided “package.json” to install most of the requirements via “npm ci”. Further information is available in the repositories README. Additionally, you will need to have:

- An AWS Account (we can provide you access if necessary).
- AWS CLI: Install the latest AWS Command Line Interface version (2.13.32).
- AWS CDK: Install the latest AWS Cloud Development Kit version (2.104.0).
- Install Node.js version 21.1.0 (or any LTS ≥ 14.15.0).
- Make sure curl is working or install an alternative like Insomnia to make API calls.
- Docker Desktop
- Repository to clone: https://github.com/patbauer/ServerlessWithAWS-Workshop42

Resources to create

Lambda Functions

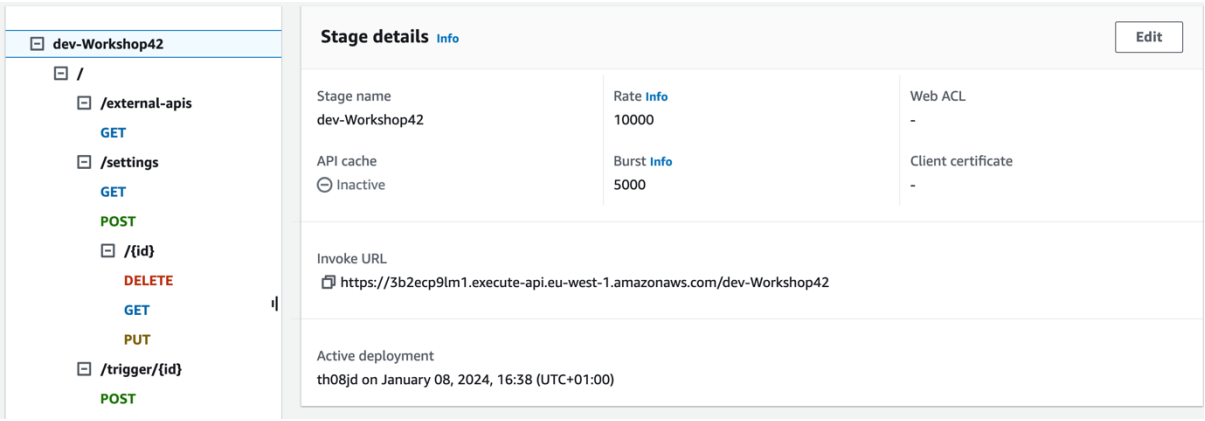
The code for an example Lambda functions is provided to you. The code is written in Typescript and found in the “lambda/settings” folder. There is a test file located in the “resources” folder to test the example Lambda function through APIGateway. The other Lambda functions must be created by you.

REST API WITH APIGATEWAY

The Lambda functions shall be triggered from a **REST API**, deployed in APIGateway. The following resources shall be created in APIGateway:

- **Resource /settings:**
    - **POST method**: Integrates with **CreateApiSettingLambda** to allow the creation of new API settings. This method and its integration serve as an example for the workshop.
    - **GET method:** Integrates with **ListSettingsLambda** to list all API settings (to be implemented).
- **Nested Resource {id} under /settings:**
    - **GET method**: Retrieves the details of a specific API setting (to be implemented).
    - **PUT method**: Updates an existing API setting (to be implemented).
    - **DELETE method**: Removes an API setting (to be implemented).
- **Resource /trigger and Nested Resource {id} under /trigger:**
    - **POST method**: Initiates the API call based on the stored API settings (to be implemented).
- **Resource /external-apis:**
    - **GET method**: Lists the available external APIs that are configured (to be implemented).

If you have created the APIGateway as specified, the method tree in the management console should look like this:



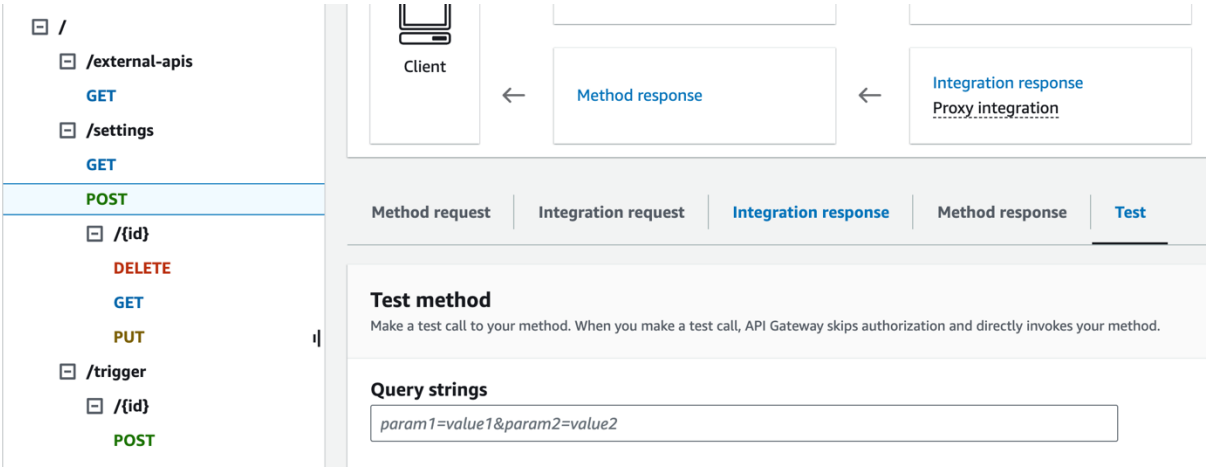
DynamoDB

The api configuration objects shall be stored in a DynamoDB table. It is already provided to you as part of the example project:

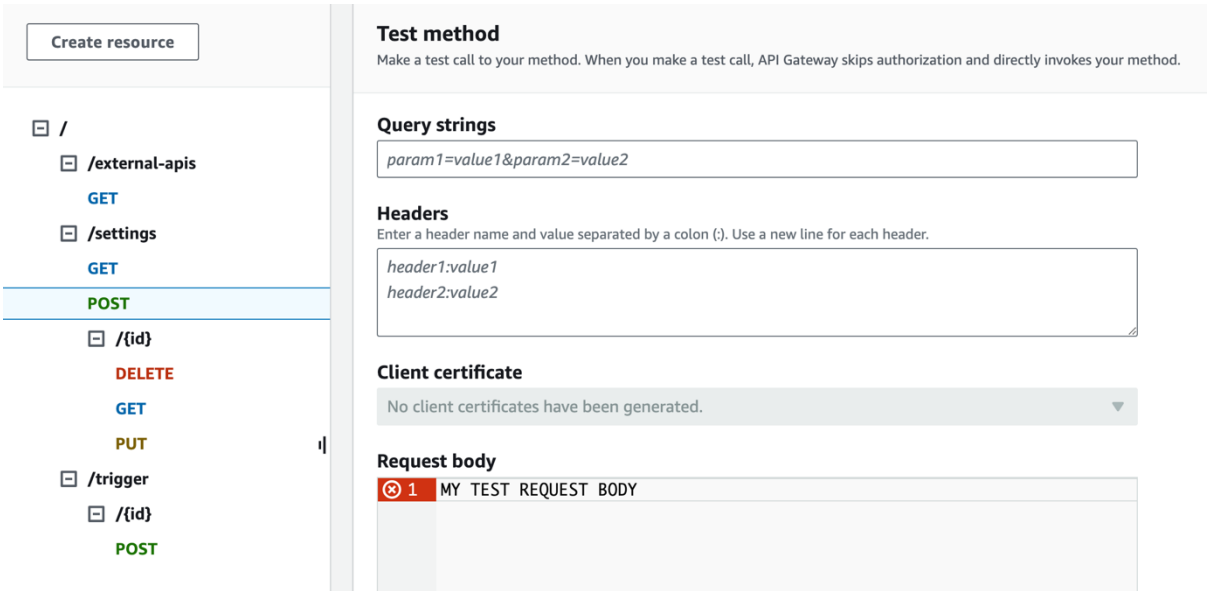
1. Resource type: **Table**
2. Partition Key name: **PK**, type: **String ,** Sort Key name: **SK**, type: **String**

Testing

We provided you test data (JSON file) to test the application the CreateSettingLambda and its API Gateway endpoint. To test “CreateSetting”, go to the created APIGateway in the AWS Management Console. Click on the “POST” method under the resource “/settings”. There, click on “Test”.



In the window that opens, post the content of the test file “**createSettingTestData.json**” in the body section of the formular.



If the test fails, analyze the stack trace display below the test window and debug your CDK resources.

Bonus

If you have successfully deployed and tested the above resources: Congratulations!

But it is not over yet... as there are over 300 AWS Services to explore :)

AWS Systems Manager ParameterStore and AWS SecretsManager are two services that could be interesting for the “API interaction management Service” you just created.

By incorporating these services into your "API interaction management Service," you can enhance security and manage your configuration data more effectively.

ParameterStore

ParameterStore is a service that provides secure, hierarchical storage for configuration data management. You can store data such as database name strings, or configuration data like API endpoints and use them as part of your automation and deployment processes. It's integrated with AWS Identity and Access Management (IAM), allowing fine-grained access control to the parameters. ParameterStore is often used to manage settings for serverless applications and other AWS services, keeping them secure and easily accessible without hardcoding them into your application.

SecretsManager

SecretsManager is a service designed to help you protect access to your applications, services, and IT resources without the upfront investment and on-going maintenance costs of operating your own infrastructure. The service enables you to easily rotate, manage, and retrieve database credentials, API keys, and other secrets throughout their lifecycle. Using SecretsManager, you can secure and manage access to secrets used by your applications and services to access resources, ensuring an additional layer of security for sensitive information.