**[##](https://github.com/PJ64/apigw_lambda_dynamodb) Example**

This example is designed for a concept mobile application called Skip the Line, which allows user to pre-order takeaway coffee while they are in transit. Just as the train pulls into the station, the user can order a coffee and pick it up on the way past the coffee shop.

The example deploys an Amazon API Gateway, multiple AWS Lambda functions and an Amazon DynamoDB table with two indexes. The example demonstrates using DynamoDB indexes to support the different query patterns of a microservice.

The Amazon DynamoDB table is partitioned on the accountid attribute and it also includes a sort key on the vendorid attribute, together they form the primary key. This allows us to find orders using accountid and a datetime range. For example, we can query for customer orders on a particular day.

The local secondary index is partitioned on the accountid attribute, the same as the base table, which is a requirement for this type of index. The orderdate attribute has been used as the sort key.

The global seondary index is partitioned on vendorid and orderdate is used as the sort key. The global secondary index partition key does not need to be the same as the base table. This index allows orders to be queried using vendorid and a datetime range. For example, we can find the most recent vendor orders.

![architecture](./images/architecture\_3.png "Architecture")

\*\*Base Table\*\*

Partition key = accountid

Sort key = vendorid

1. The third script will get items using a query operation against the local secondary index using an accountid and orderdate range.

\*\*Local Secondary Index\*\*

Partition key = accountid

Sort key = orderdate

4. The fourth script will get items using a query operation against the global secondary index using a vendorid and orderdate range.

\*\*Global Secondary Index\*\*

Partition key = vendorid

Sort key = orderdate

## Setup

You will need to download and install [Node.js](https://nodejs.org/en/download/) before you can start using the AWS Cloud Development Kit.

This example is developed using the AWS CDK and Typescript, so you will need to install both Typescript and the CDK using the following commands

```

npm install -g typescript

npm install -g aws-cdk@latest

```

Since this CDK project uses ['Assests'](https://docs.aws.amazon.com/cdk/latest/guide/assets.html), you might need to run the following command to provision resources the AWS CDK will need to perform the deployment.

```bash

cdk bootstrap

```

The testing scripts can be executed using Jupyter Notebook. There are a few methods for installing Jupyter Notebooks. These instructions will help you get to started with [JupyterLab](https://jupyter.org/install) installation.

You can also install Jupyter Notebooks as part of [Anaconda](https://docs.anaconda.com/anaconda/install/index.html) installation.

To download this example, you will need to install [Git](https://github.com/git-guides/install-git). After installing git follow these [instructions](https://github.com/git-guides/git-clone) to learn how to clone the repository.

After the repository has been cloned set the command prompt path to the cloned directory and run the following command to install the project dependencies.

```bash

npm install

```

\*\*cdk synth\*\* executes the application which translates the Typescript code into an AWS CloudFormation template.

```bash

cdk synth

```

After the synth command has generated the template use the  \*\*cdk deploy\*\* command to deploy the template to AWS CloudFormation and build the stack. You will be prompted to confirm the deployment with y/n.

```bash

cdk deploy

```

## Test the Stack

We need to install Jest since we are using the Jest framework to test the stack. Testing the stack is optional.

```

npm install --save-dev jest @types/jest @aws-cdk/assert

```

## Run the Example

Open the Jupyter Notebook in the \*\*jupyter\_notebook directory\*\* follow the instructions.

## Cleanup

From the command prompt execute the following command: \*\*cdk destroy\*\*

**Jupyter Notebook**

This example demonstrates using Amazon EventBridge to invoke Lambda functions when events are added to the EventBus. In the example event rules are used to trigger the Lambda functions. There is an event rule created for new orders and a rule that catches update operations. One Lambda function inserts into a DynamoDB table while the other function updates the item in the table.

 The first script sends new orders to the gateway endpoint. The event rule for new orders filters on the 'eventtype'::['new\_order'] attribute.

 The second script sends an order update to the gateway endpoint. The event rule for order updates filters on the 'eventtype':['update\_order'] attribute

Before you run the final script you need to create an event rule to catch delete operations by using the 'eventtype':['delete\_order'] attribute, the delete script includes the ```'eventtype':['delete\_order']```.

Amazon EventBridge. Using Amazon EventBridge create a rule that catches delete operations using the following settings.

Event Bus: ```event\_sourcing\_eventbridge```

Name: ``` event\_sourcing\_eventbridge\_delete\_order```

Target: ```Lambda```

Function: ```event\_sourcing\_eventbridge\_delete\_item```

Event pattern:

```

{

"detail": {

"order": {

"eventtype": ["delete\_order"]

}

}

}```