Serverless Application

Rubrics:

https://review.udacity.com/#!/rubrics/2574/view

Please use the following settings in the config.ts of the client in order to test our serverless application:

```
const apiId = 'qbasdfm88e'

export const apiEndpoint = `https://${apiId}.execute-api.us-east-1.amazonaws.com/dev`

export const authConfig = {
  domain: 'dev-clq116aa.auth0.com',
  clientId: 'j89rVwvRORYSRJbXWAGRCWk25TeB8FdO',
  callbackUrl: 'http://localhost:3000/callback'
}
```

Path to download the certification for authorization using Auth0 and JWT:

https://dev-clq116aa.auth0.com/.well-known/jwks.json'

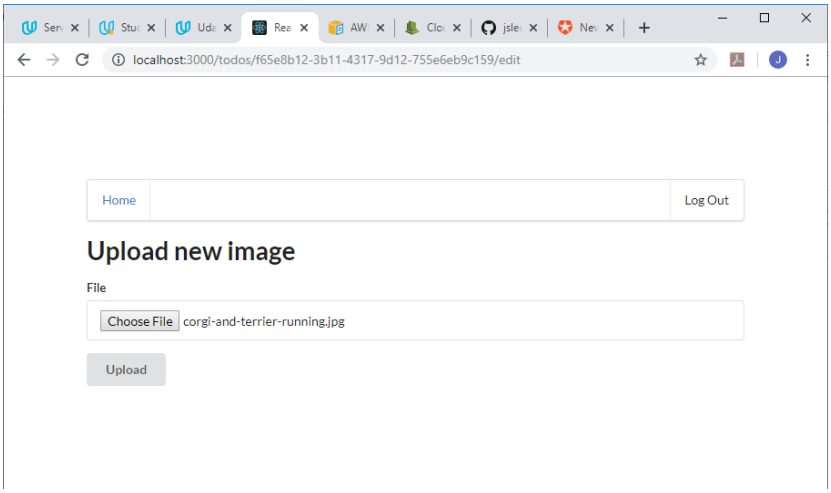
1. Functionality

1.1 The application allows users to create, update, delete TODO items

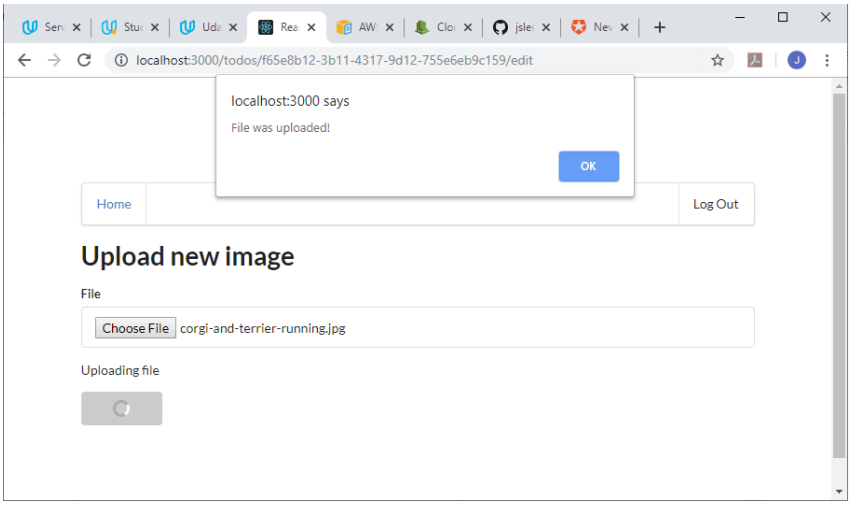
1. Please git clone https://github.com/jsleung1/serverless_application.git
2. In the client folder, execute: npm run start
3. Logged in using Gmail account (supported by https://auth0.com/)
4. User should able to create, update (by placing the “checkmark” of the TODO item to mark the item as Done) and delete TODO items.

1.2 The application allows users to upload a file.

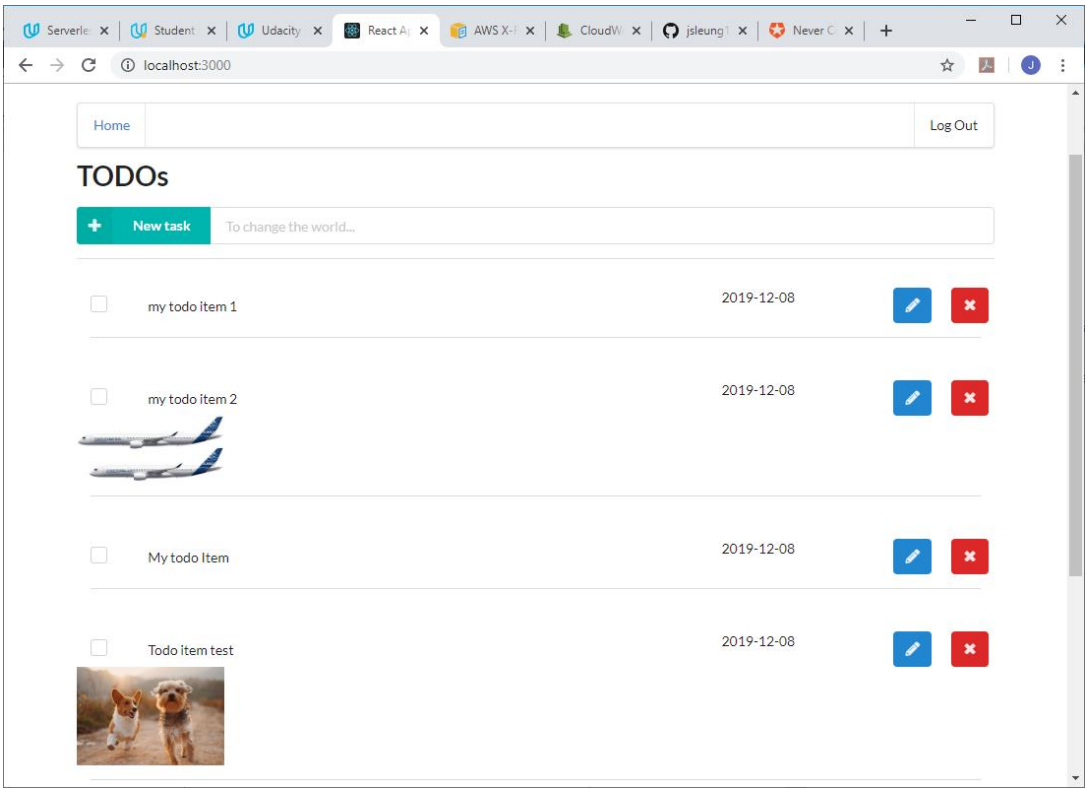
1. User select an image file to upload:



2. User successful uploaded image file:



The image (now stored in S3 bucket) is successfully shown with the TODO item:



1.3 The application only displays TODO items for a logged in user.

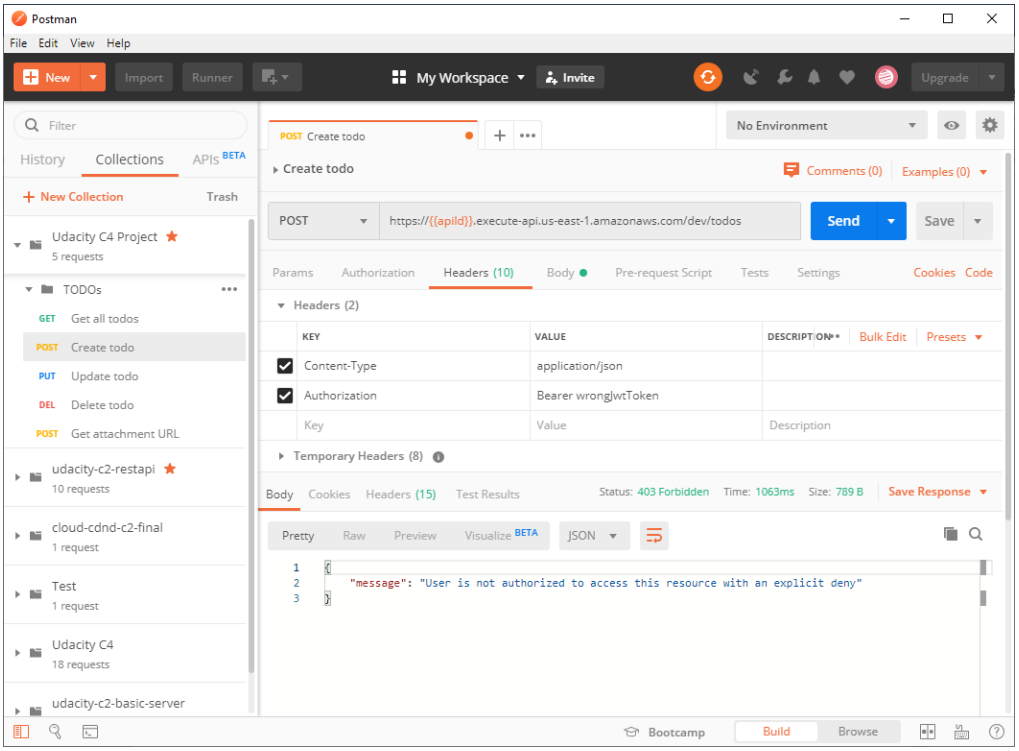
Refer to /backend/src/dataLayer/todoAccess.ts :

```
async getAllTodos(userId: string): Promise<TodoItem[]> {
  this.logger.info('getAllTodos')

  const result = await this.docClient.query({
    TableName: this.todosTable,
    IndexName: this.todosUserIdIndex,
    KeyConditionExpression: 'userId = :userId',
    ExpressionAttributeValues: {
        ':userId': userId
    }
  }).promise()

  const items = result.Items
  return items as TodoItem[]
}
```

getAllTodos (called by lambda function GetTodos) will query all todos by using the userId, which is obtained by calling parseUserId(jwtToken). The client will only display todo items associated with that userId.

1.4 Authentication is implemented and does not allow unauthenticated access.

Using the wrong jwtToken value or missing Authorization header will create a HTTP respond with error code 403 Forbidden as shown in the following example:

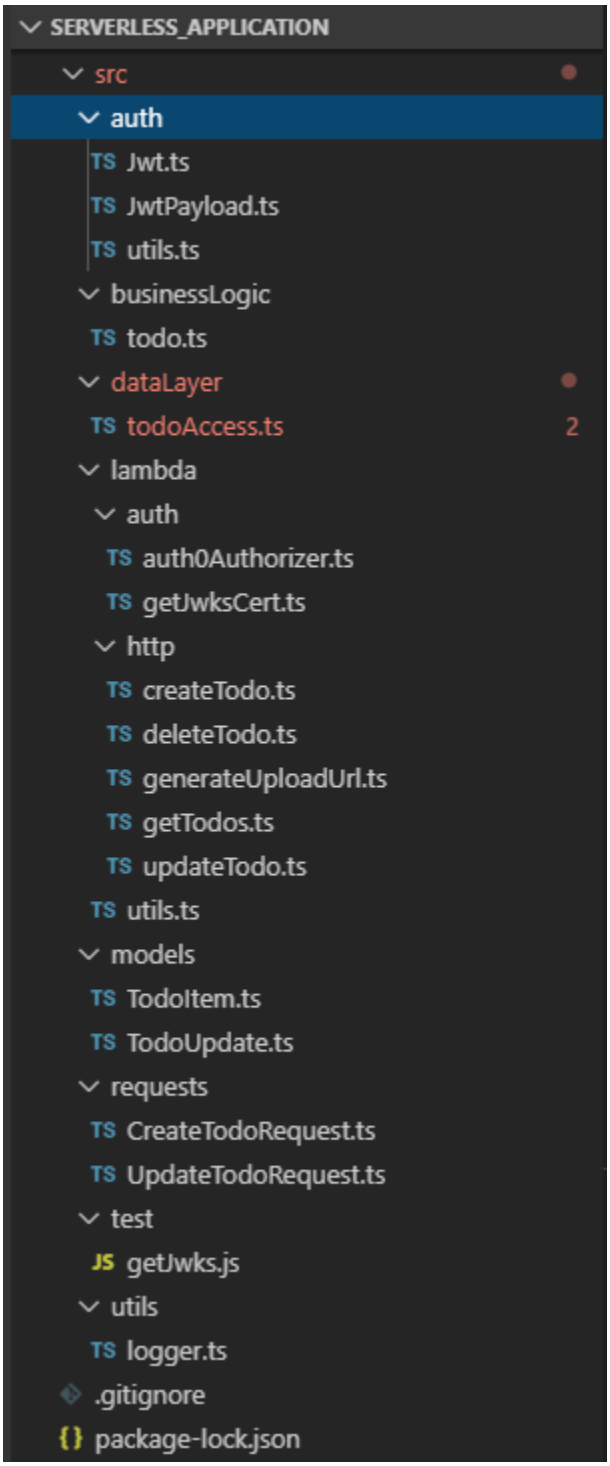


Please note in the auth0Authorizer.ts, it downloads the certificate by calling getJksCert(`https://dev-clq116aa.auth0.com/.well-known/jwks.json`) without hard coding the cert in the authorizer.

2. Code Base

2.1 The code is split into multiple layers separating business logic from I/O related code.

Refer to the code structure in the /backend folder, it shows the code is split into multiple layers separating business logic (businessLogic/todos.ts) from I/O related code (dataLayer/todoAccess.ts).



2.2 Code is implemented using async/await and Promises without using callbacks.

Please check all the code in the backend folder. It contains code using async/await and Promises without using callbacks.

3. Best Practices

3.1 All resources in the application are defined in the "serverless.yml" file

All resources are defined in backend/serverless.yml .

3.2 Each function has its own set of permissions.

In the serverless.yml file, under plugins, the “serverless-iam-roles-per-function” was included. As shown in the following, each Lambda function definition has its own iamRoleStatements:

```
functions:
  Auth0:
    handler: src/lambda/auth/auth0Authorizer.handler

  GetTodos:
    handler: src/lambda/http/getTodos.handler
    events:
      - http:
          method: get
          path: todos
          cors: true
          authorizer: Auth0
    iamRoleStatements:
      - Effect: Allow
        Action:
          - dynamodb:Query
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}
      - Effect: Allow
        Action:
          - dynamodb:Query
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}/index/${self:provider.environment.TODOS_USERID_INDEX}
      - Effect: Allow
        Action:
          - xray:PutTraceSegments
          - xray:PutTelemetryRecords
        Resource: "*"
  CreateTodo:
```

```
    handler: src/lambda/http/createTodo.handler
    events:
      - http:
          method: post
          path: todos
          cors: true
          authorizer: Auth0
          reqValidatorName: RequestBodyValidator
          documentation:
            summary: Create a new Todo
            description: Create a new Todo
            requestModels:
              'application/json': CreateTodoRequest
    iamRoleStatements:
      - Effect: Allow
        Action:
          - dynamodb:PutItem
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}
      - Effect: Allow
        Action:
          - xray:PutTraceSegments
          - xray:PutTelemetryRecords
        Resource: "*"
  UpdateTodo:
    handler: src/lambda/http/updateTodo.handler
    events:
      - http:
          method: patch
          path: todos/{todoId}
          cors: true
          authorizer: Auth0
          reqValidatorName: RequestBodyValidator
          documentation:
            summary: Update existing Todo
            description: Update existing Todo
            requestModels:
              'application/json': UpdateTodoRequest
    iamRoleStatements:
      - Effect: Allow
        Action:
          - dynamodb:Query
          - dynamodb:UpdateItem
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}
      - Effect: Allow
        Action:
          - dynamodb:Query
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}/index/${self:provider.environment.TODOS_TODOID_INDEX}
```

```
      - Effect: Allow
        Action:
          - xray:PutTraceSegments
          - xray:PutTelemetryRecords
        Resource: "*"
  DeleteTodo:
    handler: src/lambda/http/deleteTodo.handler
    events:
      - http:
          method: delete
          path: todos/{todoId}
          cors: true
          authorizer: Auth0
    iamRoleStatements:
      - Effect: Allow
        Action:
          - dynamodb:Query
          - dynamodb:DeleteItem
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}
      - Effect: Allow
        Action:
          - dynamodb:Query
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}/index/${self:provider.environment.TODOS_TODOID_INDEX}
      - Effect: Allow
        Action:
          - xray:PutTraceSegments
          - xray:PutTelemetryRecords
        Resource: "*"
  GenerateUrl:
    handler: src/lambda/http/generateUploadUrl.handler
    events:
      - http:
          method: post
          path: todos/{todoId}/attachment
          cors: true
          authorizer: Auth0
    iamRoleStatements:
      - Effect: Allow
        Action:
          - dynamodb:Query
          - dynamodb:UpdateItem
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}
      - Effect: Allow
        Action:
          - dynamodb:Query
        Resource: arn:aws:dynamodb:${self:provider.region}:*:table/${self:provider.environ
ment.TODOS_TABLE}/index/${self:provider.environment.TODOS_TODOID_INDEX}
```

```
      - Effect: Allow
        Action:
          - s3:PutObject
          - s3:GetObject
        Resource: arn:aws:s3:::${self:provider.environment.IMAGES_S3_BUCKET}/*
      - Effect: Allow
        Action:
          - xray:PutTraceSegments
          - xray:PutTelemetryRecords
        Resource: "*"
```

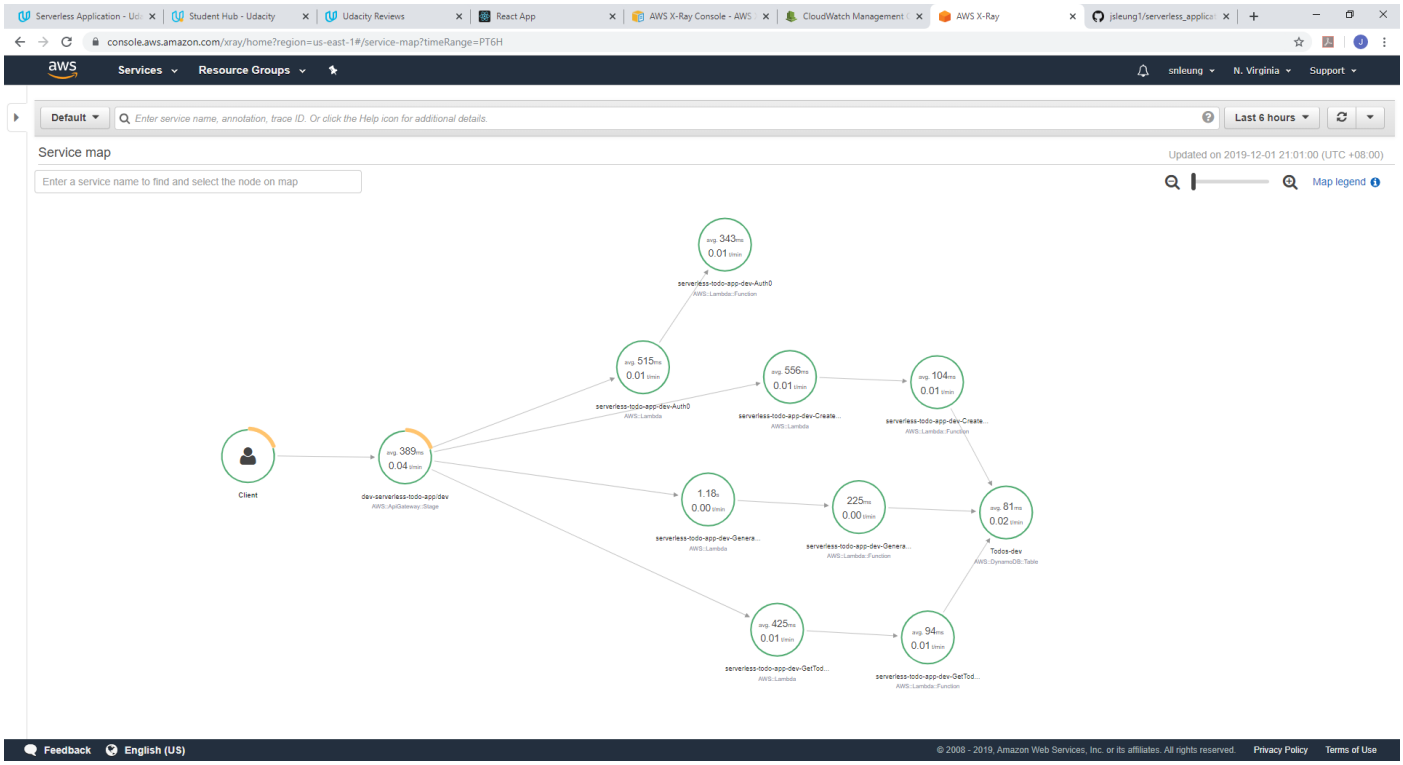
3.3 Application has sufficient monitoring.

- Distributed tracing is enabled

Using AWS X-Ray, distributed tracing is enabled by define the following in serverless.yml:

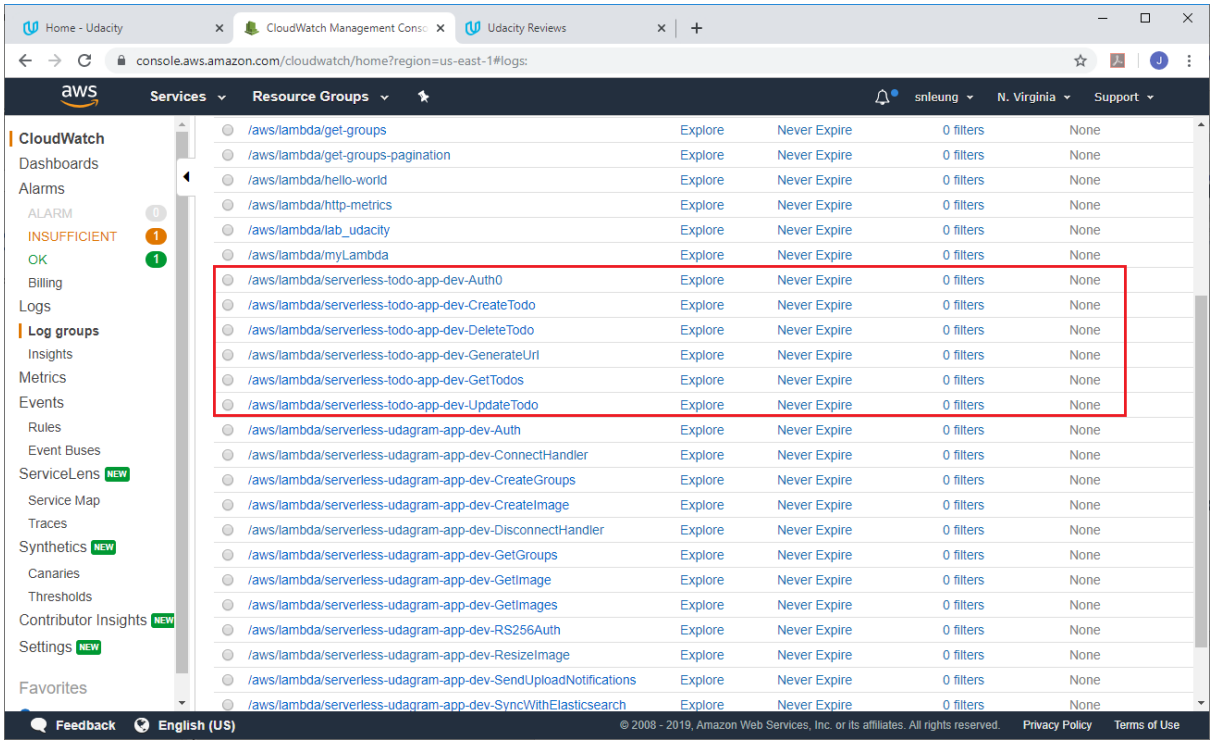
```
provider:
  ....
  tracing:
    lambda: true
    apiGateway: true
```

In the AWS Console, under AWS X-Ray, it shows the following trace:

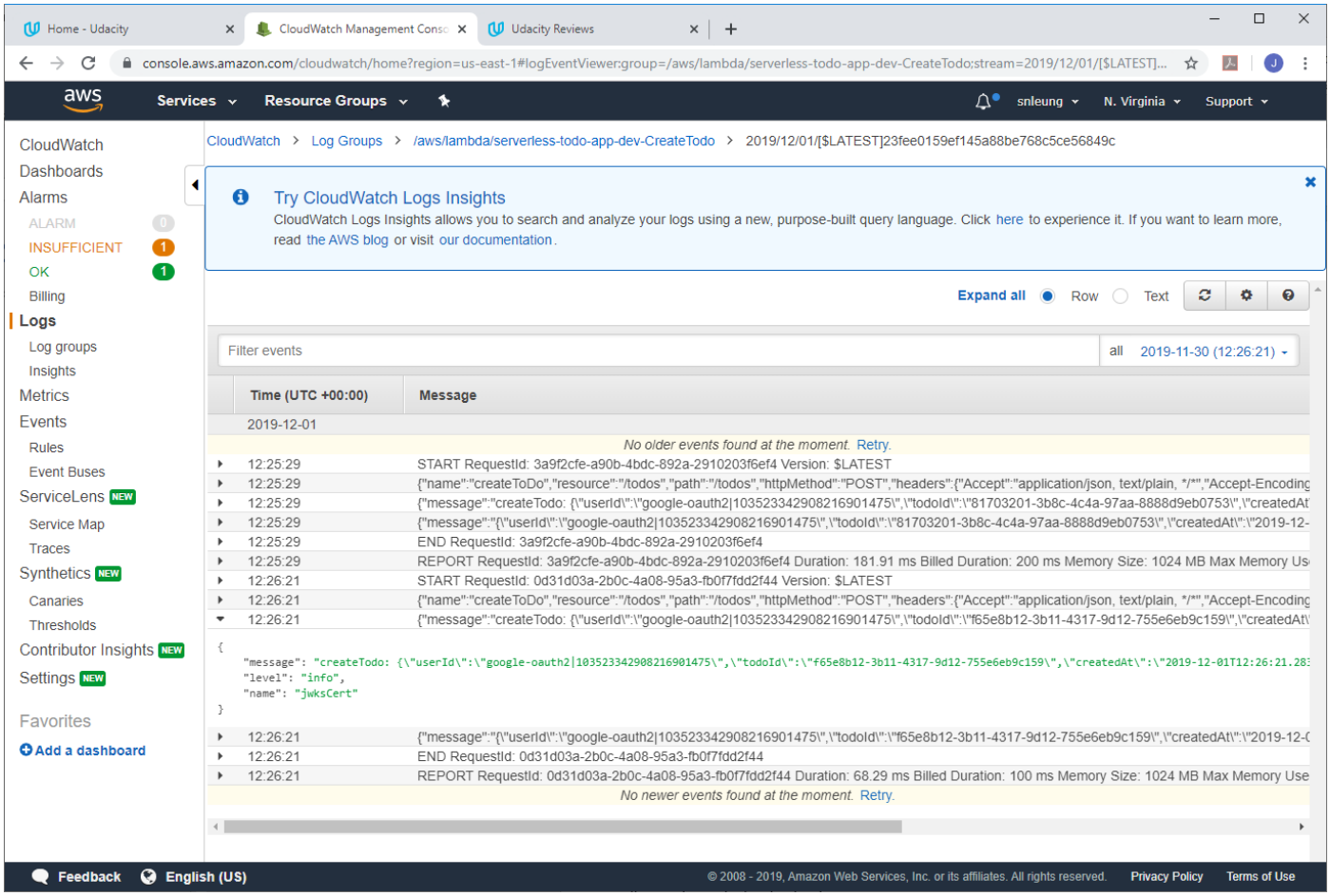


- It has a sufficient amount of log statements

Our application uses the Winston logger which outputs the logs to AWS CloudWatch:

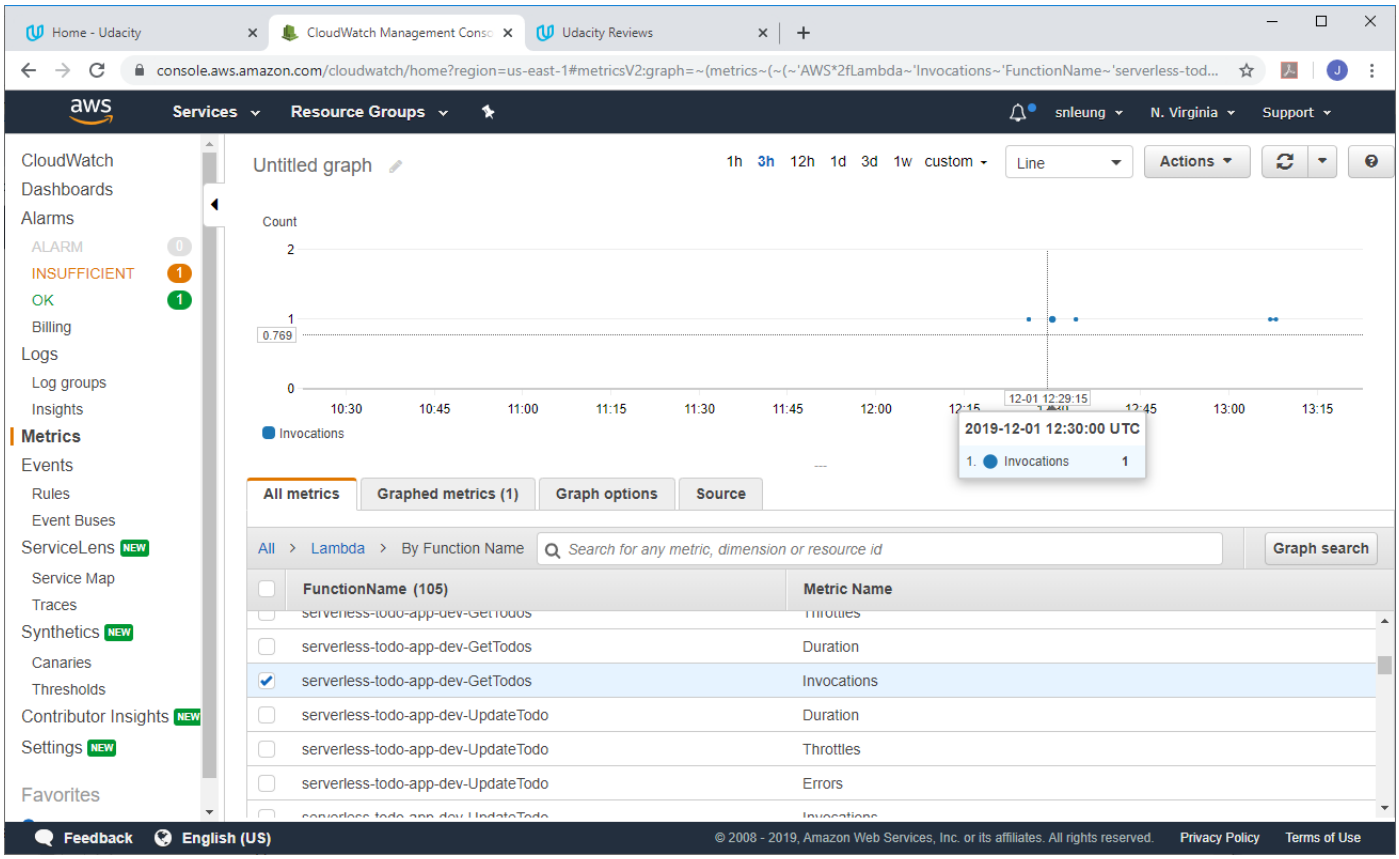


For example, for log group “aws/lambda/serverless-todo-app-dev-CreateTodo”, we can check the JSON object right before the object is being saved to DynamoDB in the Lambda function “CreateToDo”:



- It generates application level metrics

Under AWS CloudWatch, we can find the application metrics of our application by going to Metrics → Lambda → By Function Name. The following example display the graph for the number of invocations of the Lambda function "GetTodos":



3.4 HTTP requests are validated

Incoming HTTP requests are validated using request validation in API Gateway (to reduce the number of unwanted Lambda function calls). In serverless.yml,

```
custom:
  ..........
  documentation:
    ...........
    models:
      - name: CreateTodoRequest
        contentType: application/json
        schema: ${file(models/create-todo-request.json)}
      - name: UpdateTodoRequest
        contentType: application/json
        schema: ${file(models/update-todo-request.json)}
```

The CreateTodoRequest and UpdateTodoRequest validation models are defined in models/create-todo-request.json and models/update-todo-request.json respectively:

create-todo-request.json :

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "todo",
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dueDate": {
        "type": "string"
      }
    },
    "required": [
      "name",
      "dueDate"
    ],
    "additionalProperties": false
}
```

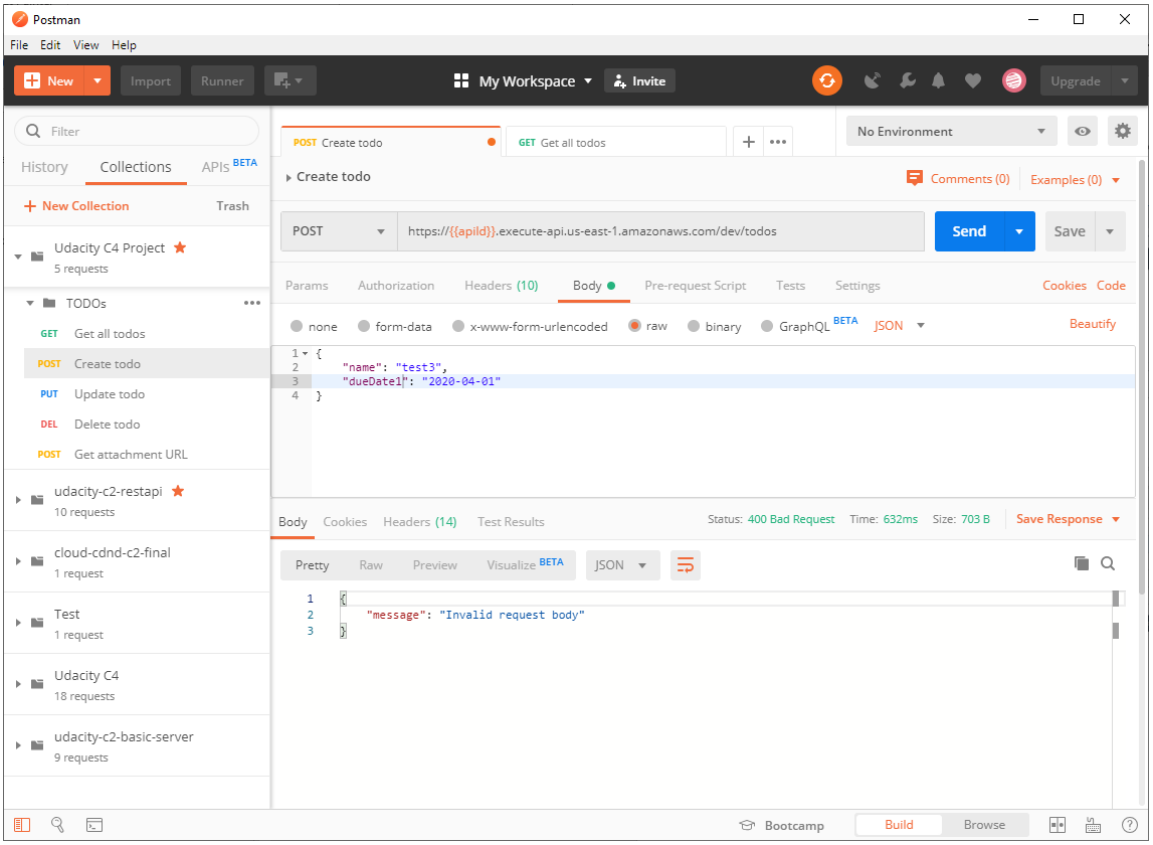
update-todo-request.json :

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",
    "title": "todo",
    "type": "object",
    "properties": {
      "name": {
        "type": "string"
      },
      "dueDate": {
        "type": "string"
      },
      "done": {
        "type": "boolean"
      }
    },
    "required": [
      "name",
      "dueDate",
      "done"
    ],
    "additionalProperties": false
}
```

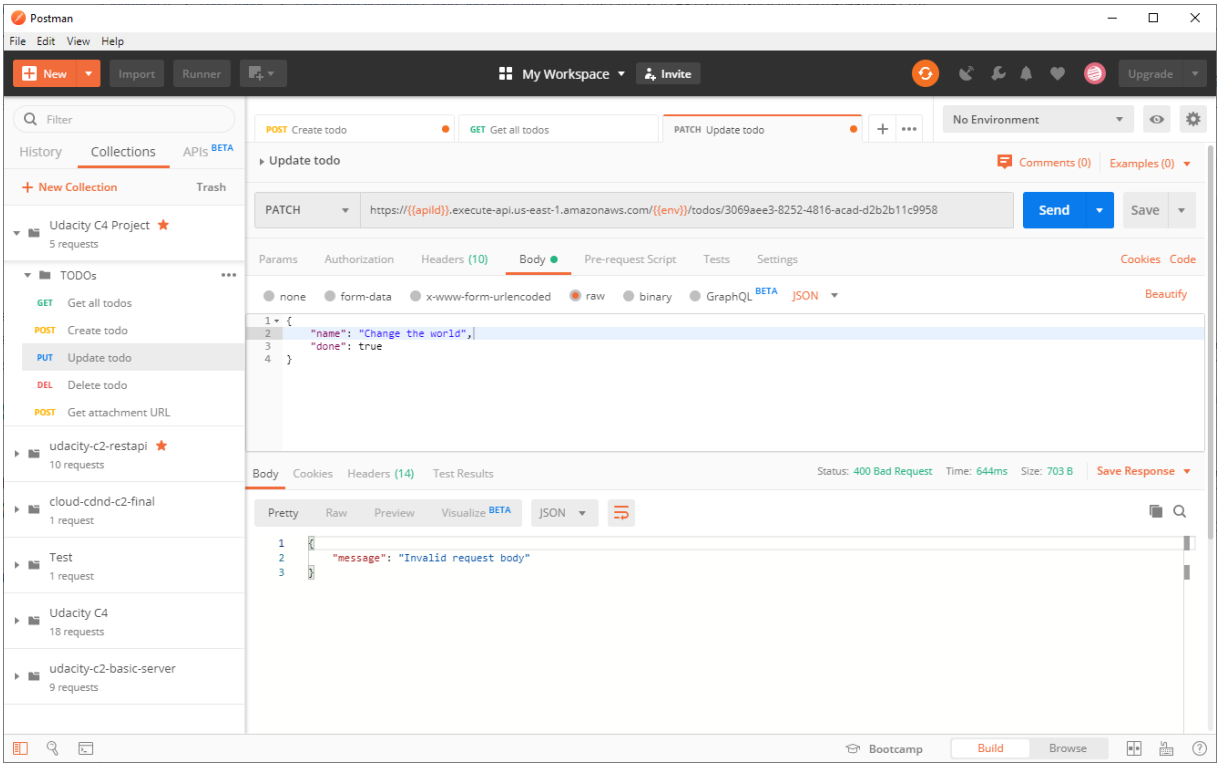
In the lambda functions definition of CreateToDo and UpdateToDo, we defined the RequestBodyValidator and requestModels:

```
  CreateTodo:
    handler: src/lambda/http/createTodo.handler
    events:
      - http:
          method: post
          path: todos
          cors: true
          authorizer: Auth0
          reqValidatorName: RequestBodyValidator
          documentation:
            summary: Create a new Todo
            description: Create a new Todo
            requestModels:
              'application/json': CreateTodoRequest
      ......
  UpdateTodo:
    handler: src/lambda/http/updateTodo.handler
    events:
      - http:
          method: patch
          path: todos/{todoId}
          cors: true
          authorizer: Auth0
          reqValidatorName: RequestBodyValidator
          documentation:
            summary: Update existing Todo
            description: Update existing Todo
            requestModels:
              'application/json': UpdateTodoRequest
      ......
```

In the first example, creating a TODO item using an incorrect attribute name “dueDate1” instead of “dueDate” will result a response code 400 Bad request with message “Invalid request body”:



In the second example, updating a TODO item with missing attribute “dueDate” will result a response code 400 Bad request with message “Invalid request body”:



4. Architecture

4.1 Data is stored in a table with a composite key.

1:M (1 to many) relationship between users and TODO items is modeled using a DynamoDB table that has a composite key with both partition and sort keys.

In the following definition of our DynamoDB, we defined **todoId**, **userId** and **createdAt** as the main attributes:

- **todoId** is a unique value assigned during the creation of new TODO item. It is required in order to update or delete the TODO item in DynamoDB.
- **userId** represents one UserId to many TODO items relationship. It is required when we query the TODO items for each user in order to display the list of TODO items in the client (after the user logged in).
- **createdAt** is used as the sort key of the TODO items in DynamoDB.

For the KeySchema, we used todoId as the partiton key, and createdAt as the sort key.

```
      KeySchema:
        - AttributeName: todoId
          KeyType: HASH
        - AttributeName: createdAt
          KeyType: RANGE
```

To query TODO items using the userId, we include the userId in the GlobalSecondaryIndexes. The following listed out the entire definition of our DynamoDB:

```
  TodosDynamoDBTable:
    Type: "AWS::DynamoDB::Table"
    Properties:
      AttributeDefinitions:
        - AttributeName: todoId
          AttributeType: S
        - AttributeName: userId
          AttributeType: S
        - AttributeName: createdAt
          AttributeType: S
      KeySchema:
        - AttributeName: todoId
          KeyType: HASH
        - AttributeName: createdAt
          KeyType: RANGE
      GlobalSecondaryIndexes:
        - IndexName: ${self:provider.environment.TODOS_TODOID_INDEX}
          KeySchema:
          - AttributeName: todoId
            KeyType: HASH
          Projection:
            ProjectionType: ALL
        - IndexName: ${self:provider.environment.TODOS_USERID_INDEX}
          KeySchema:
```

```
            - AttributeName: userId
              KeyType: HASH
            Projection:
              ProjectionType: ALL
        BillingMode: PAY_PER_REQUEST
        TableName: ${self:provider.environment.TODOS_TABLE}
```

4.1 Scan operation is not used to read data from a database.

The application uses query method (by specifying the userId). This is possible because the userId is added to the GlobalSecondaryIndexes. No scan method is ever used in the application.