Event-Driven cloud Architecture, CQRS and Event Sourcing for user management

2019/08/09

About	
Company	Molo17 S.r.l.
Author	Lorenzo Busin
State	Approved
Use	Internal
Email	lorenzo.busin@gmail.com

Description

This document contains doc about cloud architectures that use an Event-Driven approach and implement CQRS and Event Sourcing for user management.

Table of contents

List of figures

1 Introduction

1.1 Project purpose

This project explains how to build Event-Driven architectures, CQRS and Event Sourcing for user management, showing how should be implemented, extended, strengths and weaknesses. The application for user management implements an authentication function and the CRUD operations for each aggregates, which are: users, roles, authorizations and groups.

1.2 References

- **What do you mean by "Event-Driven"? - Martin Fowler**: martinfowler.com/articles/201701-event-driven.html;
- **CQRS - Martin Fowler**: martinfowler.com/bliki/CQRS.html;
- **Event Sourcing - Martin Fowler**: martinfowler.com/eaaDev/EventSourcing.html;
- **AWS Lambda documentation**: martinfowler.com/articles/201701-event-driven.html;

2 Amazon Web Services

Amazon Web Services (AWS) is a subsidiary of Amazon that provides on-demand cloud computing platforms to individuals, companies, and governments, on a pay-as-you-go basis. These cloud computing web services provide a set of primitive abstract technical infrastructure and distributed computing building blocks and tools. AWS's version of virtual computers emulate most of the attributes of a real computer including, hardware central processing units and graphics processing units, local/RAM memory, hard-disk/SSD storage; a choice of operating systems; networking; and pre-loaded application software such as web servers and databases.

2.1 Lambda

AWS Lambda is an event-driven, serverless computing platform provided by Amazon. It is a computing service that runs code in response to events and automatically manages the computing resources required by that code. The purpose of Lambda is to simplify building smaller, on-demand applications that are responsive to events and new information. AWS targets starting a Lambda instance within milliseconds of an event. Node.js, Python, Java, Go, Ruby and C# through .NET Core are all officially supported.

In this project, lambda functions are the computing core for the execution of all commands and are written in Node.js.

2.1.1 Write mode Lambda functions

2.1.1.1 PushOperationAggregateToSQS

This functions starts the flow and triggered when fetching the corresponding API gateway URL with a POST request like this:

```
fetch("https://domain/resourceAPIpath", {
  method: "post",
  headers: {
    'Accept': 'application/json',
    'Content-Type': 'application/json'
  },

  body: JSON.stringify({
    "attribute1": "value1",
    "attribute2": "value2"
  })
});
```

Figura 1: Fetch POST API

Once triggered, this functions retrieve the event and put it in the *MessageBody* parameter. The event object is not changed, instead in the user functions which encrypt the password before sending the message to corresponding SQS queue.

2.1.1.2 CommandOperationAggregate

This functions validate the values of the attributes before storing the event in the *eventStore* and triggered when a new event arrives to the corresponding queue.
If you have to check for duplicated attributes in the database, the function must be marked *async* because it has to wait for the result of the check operation.

2.1.1.3 Mediator

This function catch the *DynamoDB* event and triggered when a change occurs in a certain table. You have to be careful that the *DynamoDB* events After that, the *QueueUrl* parameter retrieved from the event objec, the payload event is passed with the *MessageBody* and then sending the execution message to corresponding SQS queue.

2.1.1.4 OperationAggregate

2.1.1.5 recovery

2.1.2 Read mode Lambda functions

2.1.2.1 readOperation

2.1.3 CloudWatch Logs

2.2 DynamoDB

2.3 API Gateway

2.4 Simple Queue Service

3 Serverless Framework

3.1 Description

3.2 serverless.yml

4 CQRS

4.1 Architecture overview

4.2 Write model

4.3 Read model

5 Aggregates

5.1 User

5.2 Role

5.3 Authorization

5.4 Group

6 Event Sourcing

6.1 Event store

7 Extension points

7.1 New aggregates

7.2 New write functions

7.3 New read functions

8 Setup