Y_2 D1 Design Details

Shantanu Kulkarni (Shan)
Gaurung Rastagi
Malavika Thakore
Ben Phoebus
Haodong Mao
Jordan Aronson

Describe how you intend to develop the API module and provide the ability to run it in Web service mode

We intend to develop the API module using GraphQL as an alternative to REST. To expose this as a Web service, we will use AWS AppSync as a managed GraphQL service to sit behind a React-based front-end.

Discuss your current thinking about how parameters can be passed to your module and how results are collected. Show an example of a possible interaction.(e.g.-sample HTTP calls with URL and parameters)

The following two examples illustrate how queries to the GraphQL API Server can yield different data. The data format and hierarchy matches the one specified in the spec.

This query:

```
{
  article (date: "2018-xx-xx xx:xx") {
    url
    headline
  }
}
```

returns all article URLs and headlines from 2018:

```
{
  "data": {
    "article": [
      {
        "url": "https://cnn.com/world/zika-outbreak-mumbai/index.html",
        "headline": "Mumbai detects Zika outbreak"
      },
      {
        "url": "https://foxnews.com/world/marburg-lagos-kills/index.html",
        "headline": "Marburg virus kills 10 in deadly Lagos outbreak"
      },
      {
        "url": "https://washingtonpost.com/world/bhutan-ebola-
lockdown/index.html",
        "headline": "Bhutan enters lockdown amongst Ebola fears"
      }
    ]
  }
```

```
}
```

A more complex query such as this one:

```
{
   article {
             report(locations: ["London"], diseases: ["Marburg", "Zika"]) {
                 diseases
                 syndromes
                 event_date
                 locations
             }
             url
             headline
   }
}
```

returns all article URLs and headlines which have reports that contain the Marburg virus in the location of London:


```
{
  "data": {
    "article": {
      "reports": [
        {
          "diseases": [
            "Marburg"
          ],
          "syndrome": [],
          "event_date": "2021-01-06 17:00:19",
          "location": [
            "London"
          ]
        },
        {
          "diseases": [
            "Marburg"
          ],
          "syndrome": [],
          "event_date": "2021-02-24 12:23:09",
          "location": [
            "London"
          ]
        }
      ],
      "url": "https://foxnews.com/world/marburg-london-kills/index.html",
      "headline": "January Marburg Virus outbreak kills another 10 in growing
London cluster"
    }
  }
}
```

Present and justify implementation language, development and deployment environment (e.g. Linux, Windows) and specific libraries that you plan to use.

We felt the specification fitted naturally with a solution that used GraphQL, given the unstructured nature of the various data sources. Since the data format for how things like articles and reports is specified in the spec, it makes defining a GraphQL schema simple. From this we can reap the following advantages from using GraphQL:

- Improved performance
    - allows you to write rich specific queries that only fetch the fields that you want. This eliminates over-fetching or under-fetching problems thereby reducing network overhead. Overall 'chatter' between endpoints is reduced since only one endpoint - the GraphQL server - is hit initially to resolve a query. As such multiple `GET` requests do not have to be made to multiple endpoints.
- Reduced complexity in front-end development
    - the development of the front-end is optimised as front-end code does not have to fetch data from multiple endpoints, but can rather write a rich SQL-like query that will fetch all necessary data in one request. Unlike REST, a new endpoint is not required for each different type of 'view' required by the front-end.

Moreover, we'd like to maximise the scalability, performance and fault-tolerance of the GraphQL API and have therefore decided to combine with a micro-services architecture that uses server-less compute. Micro-services decouples services, improves scalability and performance, and in this case is reduced in complexity due to the use of a GraphQL schema.

The API back-end (e.g resolver functions, schema) will be written in Golang, using the `graphql-go` library for GraphQL. Golang is a minimalist, garbage collected yet performant, strongly and statically typed compiled programming language. For our purposes, we favour the performance, ease of development and strong type system which reduces compile time errors.

To assist the API in resolving data, we will use the fully managed proprietary NoSQL database service Amazon DynamoDB; the unstructured nature of much of the data makes it unsuitable for a mapping to a relational database.

Since we are deploying cloud-native with AWS, we will be using AWS Code-pipeline for our build process. From a broader DevOps point of view, we will be using Trello to manage user-stories, tasks and features in a simple scrum board for each sprint.

For our deployment environment, we have chosen to opt for server-less compute to integrate seamlessly with our micro-services architecture. Given that we are AWS-native, we will use AWS Lambda, effectively abstracting away all server/compute considerations as each service will run as a Lambda function running Golang functions.