CS4471b Software Architectures Class Project

(Changes are highlighted in red in the table of contents)

Jerry Yuan Chen, Jiarui Yang, Justin Koh, Reid Jackson

Group 3

Table of Contents (Changes in Red)

Services Offered

(Found in external file: 2_Project Services)

Service Requirements

(Found in external file: 2_Project Services)

Project Plan

(Found in external file: 3_Project Plan)

Cloud Platform

None of our group members have specific experience with IBM Cloud. However, Justin and Jerry have transferable experience from Google Cloud platform, Jordan has similar experience with Amazon Web Services, and Reid has previous experience with Kubernetes/Docker.

We are planning to host our front-end application (VueJS/ReactJS) on the IBM Cloud to reduce the time or cost of managing or maintaining our application. We will use IBM Cloud (serverless) functions for the actual app logic, which will allow our app to scale fluidly and require less maintenance. Additionally, this will allow flexibility upon failures, whereupon only one serverless app would fail at a single time. Furthermore, we are planning to use the Cloudant database to store our dataset, which is compatible with IBM Cloud. We will parse that given stock data beforehand in order to store it into databases. Moreover, the IBM cloud also provides better security and performance for our application, since we do not want to have our data get leaked.

Services and Service-Oriented Architectures (SOA)

All group members are familiar with the Service-Oriented Architecture of services being provided to other components through communication protocols. We all have experience creating programs that follow similar structures to that of Service-Oriented Architecture. We are all also familiar with “services” and creating software that provides different tasks and operations for the user. These include creating web applications, using API’s and other various protocols to create automated procedures that are done for the user. It will also concern database creation and management of sending and receiving data for use in user applications.

Implementation-wise, a service-oriented architecture will allow our team members to work more independently, as the codebase will have less coupling. Moreover, as all the services will not be functionally dependent on one another, their individual development progress will not be blocked by each other.

Views

Section 1: The Primary Presentation

Module View
(Found in external file: 4_Module View)

Component and Connector View
(Found in external file: 5_Component Connector View)

Allocation View
(Found in external file: 6_ Allocation View)

Company Sequence Diagram (Quality View 1)
(Found in external file: 7_Company Sequence Diagram)

Company Communication Diagram (Quality View 2)
(Found in external file: 8_Company Communication Diagram)

Section 2: The Element Catalog

The several elements we have are common throughout our provided architecture views. Therefore, we will explain them all in this single view template.

Elements, Their Properties, and Their Interfaces:

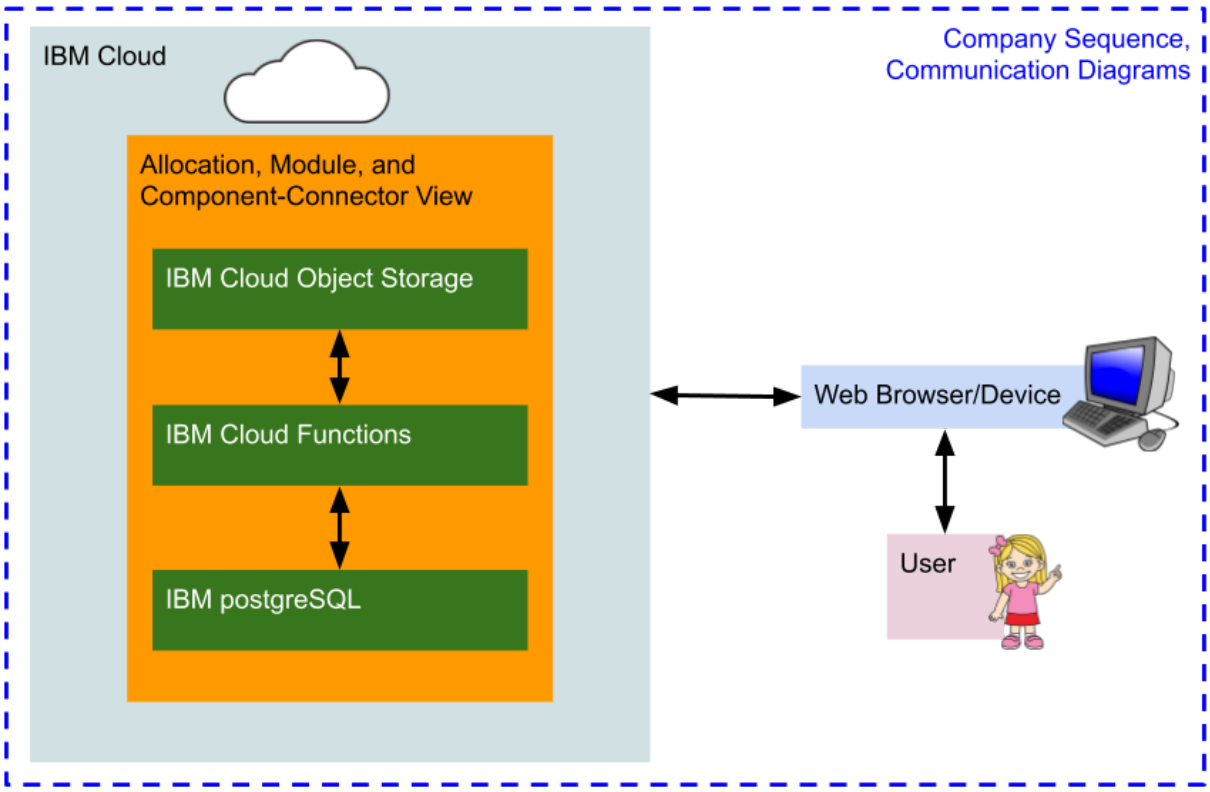
1. UI
    a. UI Module
        - This will be a static website hosted on the IBM cloud. It will have a web interface which the user can interact with. Upon certain inputs, it will make requests to appropriate endpoints to our server-side IBM Cloud functions.
        - It will contain both HTML layouts, CSS stylings, and JavaScript logic.
2. Cloud Functions/Logic
    a. Stock Sales/Query/Trades/Data Module
        - This module contains the serverless/ stateless function/logic for querying stock trades (Stock Query Service) as well as its respective database.
        - It is called by other cloud functions and the UI module as an REST API endpoint.
        - It contains and interfaces with the stock trades DB by calling NoSQL queries.
    b. Flag Suspects Module
        - This module contains the serverless/ stateless function/logic for filtering flagged suspicious stock trades (Flag Suspects Service).
        - It is called by the UI module as an REST API endpoint.

        - It calls the stock query module through it's REST API endpoint.
    c. Stock Analysis Module
        - This module contains the serverless/ stateless function/logic for querying stock trades (Stock Analysis Service).
        - It is called by the UI module as an REST API endpoint.
        - It calls the stock query module through it's REST API endpoint.
    d. Stock Sales Volume Visualization Module
        - This module contains the serverless/ stateless function/logic for generating visualization data for a specified stock (Stock Sales Volume Visualization Module).
        - It is called by the UI module as an REST API endpoint.
        - It calls the stock query module through it's REST API endpoint.
    e. Exchange Sales Volume Visualization Module
        - This module contains the serverless/ stateless function/logic for generating visualization data for a specified exchange (Exchange Sales Volume Visualization Module).
        - It is called by the UI module as an REST API endpoint.
        - It calls the stock query module through it's REST API endpoint.
    f. Company Query/Data Module
        - This module contains the serverless/ stateless function/logic for querying stock trades (Company List Service) as well as its respective database.
        - It is called by other cloud functions and the UI module as an REST API endpoint.
        - It contains and interfaces with the company query DB by calling NoSQL queries.
    g. Company Comparison Report Module
        - This module contains the serverless/ stateless function/logic for querying stock trades (Company List Service).
        - It is called by the UI module as an REST API endpoint.
        - It calls the stock query and the company query module through it's REST API endpoint.
    h. Currency Rates/Conversion Module
        - This module contains the serverless/ stateless function/logic for providing currency conversion rates (Currency Conversion Service) as well as its respective database.
        - It is called by the UI module as an REST API endpoint.
        - It contains and interfaces with the currency rates DB by calling NoSQL queries.
3. Databases
    a. Stock Trades DB
        - This database is a part of the stock query module.
        - It contains all the query-able stock trades.
        - It is interfaced by the stock query module through NoSQL queries.

   b. Company Query DB
      - This database is a part of the company query module.
      - It contains all the query-able company names.
      - It is interfaced by the company query module through NoSQL queries.
   c. Currency Rates DB
      - This database is a part of the currency conversion module.
      - It contains all the query-able currency rates.
      - It is interfaced by the currency conversion module through NoSQL queries.
4. Company Sequence/Communication Objects
   a. Main Page
      - Contains the landing web interface for the user
      - Contains navigation options to each of the services (company pages)
   b. Company Page
      - Contains the web interface for each service
      - Allows the user to specify their input and view retrieved data in a comprehensible way.
      - Sends requests and processes responses from company queries
   c. Company Query
      - Contains a REST API interface to be called on by the company page, or other company queries.
      - Contains the logic of data processing required by the service specification.
      - Executes queries on the company database.
   d. Company Database
      - Contains the set of persistent financial data to be processed and retrieved.
      - Interfaced by the company query through NoSQL queries.

<u>Relationships:</u>
Unless otherwise specified, all the arrows in the views designate a “relies on” relationship. “Has-a” relationships are shown by elements being inside or subsuming one another.

Section 3: Context Diagram



This diagram shows the general context that each of our views concern. The allocation, module, and component-connector views all deal with the structure of our code as hosted on the IBM Cloud. They all concern modules which interact between IBM Cloud Object Storage, Cloud Functions and Cloudant databases. On the other hand, the company sequence and communication quality views concern the interaction flow from the users to the metal.

Section 4: Variability Guide

Because we are approaching development in staggered stages (in prioritizing certain services over others), the later modules to be developed are open to changes up until that point. In fact, only the stock query module is invariable at an early point, as it is the only module which is depended-upon by other modules. Therefore, all the other service modules can be adapted at any time in development, given that they are not yet set in stone. Moreover, since each module acts as a black box for others, its internal logic may also be patched and change without impact on its interface.

Elements which cannot be changed, or require extraordinary effort to change:

- Reliance on IBM Cloud technologies
- Parsing and insertion of data into NoSQL storage.
- Interfaces of crucial modules (i.e. the stock-query modules)

Elements which can be changed at any point (including patched after deployment):

- Any module's internal logic
- Web user-interface and presentation

Elements which would require some refactoring:

- Interfaces of non-crucial modules.
- Addition/removal of new modules.

Configurable parameters that can be changed on deployment and redeployment (may require more/less money):

- Rate limits for requests.
- Database capacity.

Section 5: Rationale

Microservices and reusability were the patterns predominately used in this design. Microservices allow each module to be seperate from others, so that they don't directly interact with each other and do a specific task. This allows for different modules to process requests and information without affecting others. This is especially useful to insure the errors or downtime of one module doesn't affect the entire system. For the UI part these modules can be easily added and removed as well, creating easy changeability and adaptability for the system. Coupling between systems is also reduced, while cohesion is increased as modules of the system perform specific tasks.

Reusability was used with the Stock Module as errors with the upper systems will be more easily traced back to the source. Using this pattern, redundancies were decreased, having similar stock code for each separate module is not ideal, and it also introduces more areas for errors in code duplication or logic for each individual stock Module that would exist.

Documentation Beyond Views

Section 1: Documentation Roadmap

Scope and Summary

Explain the purpose of the document and briefly summarize what is covered and (if you think it will help) what is not covered. Explain the relation to other documents (such as downstream design documents or upstream system engineering documents).

The purpose of this document is to provide:

1. *General documentation on the system diagram can be used for the purpose of education*: The views and documents we created can be a great means of introducing our solutions or system components to new people, including new members of the team and stakeholders.

- Module View & Allocation View: give us a general idea of how the system looks like inside, and how components interact with each other.
- Company Sequence Diagram: give us the general idea of how the users/clients interact with the system.

2. *A primary vehicle for communication among stakeholders*: Our diagrams and documents should be the primary way for us to explain our system to these people, especially without technical backgrounds, such as stakeholders, higher-level management teams. The documentation and views help us get approvals for proceeding the project.

3. *The basis for system analysis and construction for Developers, QA testers, and project manager*s: The documents and views can be used to keep track of the project for project managers and give a general scope for QA testers. As for developers, it gives the basic ideas of how system components communicate with other components in the system, but more importantly, other systems from other teams.

Things covered: interaction between system components, User interactions. The general architecture of each component.

Things can be covered: Requirements and purpose of each component. Input/output data format of each component.

How the Documentation is Organized

For each section in the documentation, give a short synopsis of the information that can be found there. An alternative to this is to use an annotated table of contents. This is a table that doesn't just list section titles and page numbers, but also gives a synopsis with each entry. It provides one-stop shopping for a reader attempting to look up a particular kind of information.

1. *Module View Documentation*: Mainly used for developers. The document gives general information on how each component is going to be implemented. It also gives the general requirements and responsibilities of each component that are needed to be fulfilled at the completion of the project.

2. *Component-and-Connector View Documentation*: Mainly used for developers, QA testers. The document gives general ideas how each component processes the data, what the data format of each component's input and output, and the relationship between each components.

3. *Allocation View Documentation*: Mainly used for developers, QA testers, and architects. It gives general ideas of how each component is mapped, it is easier for providing reasons about performance, availability, security.

4. *Powerpoint Documentation*: Mainly used for stakeholders. The powerpoint slides are used for explaining our ideas and solutions to stakeholders, and high-level management teams. The powerpoint documentation is used for getting approval to proceed the project.

View Overview

The major part of the map describes the views that the architect has included in the package. For each view, the map gives the following information:

- The name of the view and what pattern it instantiates, if any.
- A description of the view's element types, relation types, and property types. This lets a reader begin to understand the kind of information that is presented in the view.
- A description of language, modeling techniques, or analytical methods used in constructing the view.

- *Module Views*: In the module view, the elements are modules which are representing each implementation unit of software that provides a coherent set of responsibilities.There are 3 types of relations: is part of (defines part/whole relationship), Depends on(defines Dependency relationship), is a (defines generalization/specialization relationship). The view is mainly used for a blueprint for the construction of the code and communicating the functionalities of each component.

- *Component-and-Connector Views*: In the Component-and-Connector View, the elements are processing units, data stores, and connectors between components. There are two types of relationships, the first one is attachment(Components and their connectors), and interface(association with one more port in an internal sub-architecture). The purpose of the view is to show the working principle of the system and guide the development of the software.

- *Allocation Views*: In the Allocation view, the elements are software elements that have properties that are required of the environment. The only relationship that this view has is "Allocated to"(a software element is mapped as an environmental element). The allocation view can be used to provide reasoning about performance, availability, security, and distributed development.

How Stakeholders can use the Documentation

The map follows with a section describing which stakeholders and concerns are addressed by each view; this is conveniently captured as a table. This section shows how various stakeholders might use the documentation to help address their concerns. Include short scenarios, such as "A maintainer wishes to know the units of software that are likely to be changed by a proposed modification. The maintainer consults the decomposition view to understand the responsibilities of each module in order to identify the modules likely to change.

The maintainer then consults the user's view to see what modules use the affected modules (and thus might also have to change)." To be compliant with ISO/IEC 42010-2007, you must consider the concerns of at least users, acquirers, developers, and maintainers.

1. Module View Documentation: Mainly used for developers. The document gives general information on how each component is going to be implemented(Guide the development).

2. Component-and-Connector View Documentation: Mainly used for developers, QA testers. The document gives general ideas how each component processes the data, the data format of each input and output.

3. Allocation View Documentation: Stakeholder, Users. The document can be viewed by members without good technical background. Allocation can mainly be used to track the progress of a project for stakeholder and Users.

Section 2: How a View Is Documented

A developer wants to know the relationships between the components and the databases, developers can consult the component connector view to understand the interactions between components in the system.

A user wants to know how reliable the overarching system is; they can refer to either the allocation view, or component connector view to understand the partitioning of the software components and understand the scalability of the platform.

An acquirer would like to know about the tech stack used to create our system. The acquirers can consult the allocation view to understand the technologies that the system is built upon to potentially transition the system to another development cycle.

A maintainer may want to know about the components of the system that are modular and susceptible to change. The maintainer can consult the module view to get a sense of the microservice architecture, and understand the abstraction of each module. This will allow the maintainer to to understand what changes will affect certain modules within the system.

Section 3: System Overview

Our system aims to provide a variety of financial stock services on a user-facing web page. We target users for personal use. This system will adopt a microservices architecture distributed across multiple SQL databases and serverless cloud functions.

Our microservice architecture consists of various individual modules, with each fully representing a proposed service. They operate and are requested independently. Each module is made to be as highly coherent and lowly coupled as possible, existing independently as cloud

functions and only interacting with each other through structured REST API endpoints. Each one of these endpoints/services/modules are stateless.Through this structure, we hope to curb ripple effects in boosting modifiability and stability.

Section 4: Mapping Between Views

Elements which are shared between the module view, component and connector view, and the allocation view include all the cloud function modules, the UI module, and the databases:

- UI Module
- Stock Sales/Query/Trades/Data Module
- Flag Suspects Module
- Stock Analysis Module
- Stock Sales Volume Visualization Module
- Exchange Sales Volume Visualization Module
- Company Query/Data Module
- Company Comparison Report Module
- Currency Rates/Conversion Module
- Service Registry Module
- Stock Trades DB
- Company Query DB
- Currency Rates DB

The elements between the company sequence and company communication diagrams are also shared and equivalent:

- Main Page
- Company Page
- Company Query
- Company Database

Now, here is the mapping between the two different sets of elements:

- Main Page (is implemented by)
    - UI Module
- Company Page (is implemented by)
    - UI Module
- Company Query (is implemented by)
    - Stock Sales/Query/Trades/Data Module
    - Flag Suspects Module
    - Stock Analysis Module
    - Stock Sales Volume Visualization Module
    - Exchange Sales Volume Visualization Module
    - Company Query/Data Module
    - Company Comparison Report Module
    - Currency Rates/Conversion Module
    - Service Registry Module

- Company Database (is implemented by)
    - Stock Trades DB
    - Company Query DB
    - Currency Rates DB

Section 5: Rationale

Statelessness spans the different views, as of right now there are no users so there is not captured state within the system of each individual user.

Microservices/Modularity spans the different views, this was decided upon to decrease coupling and allow for different services to run independently of each other. This is shown in the views with each module being separate from each other to reduce dependencies. Although some dependencies still exist, these are to reduce redundant code as this allows more places for logic or coding errors to occur.

Section 6: Directory

The directory is a set of reference material that helps readers find more information quickly. It includes an index of terms, a glossary, and an acronym list.

IBM Cloud Object Storage - discussed in 'Section 3: Context Diagram', it is "a highly scalable cloud storage service, designed for high durability, resiliency and security"
https://www.ibm.com/ca-en/marketplace/object-storage

IBM Cloud Functions - discussed in 'Section 3: Context Diagram', allow the user to "use your favorite programming language to write lightweight code that runs app logic in a scalable way. You can run code on-demand with HTTP-based API requests from applications or run code in response to IBM Cloud services and third-party events"
https://cloud.ibm.com/functions/

IBM Cloud Cloudant - discussed in 'Section 3: Context Diagram', "is a NoSQL cloud database deployment"
https://cloud.ibm.com/catalog/services/cloudant

Cloudant - "IBM Cloudant is a fully managed JSON document database that offers independent serverless scaling of provisioned throughput capacity and storage."
https://cloud.ibm.com/catalog/services/cloudant#about

Acronyms:
DB - database; a structured set of data held in a computer, especially one that is accessible in various ways. In this case IBM's Cloud Database using Cloudant and NoSQL
JSON - JavaScript Object Notation; is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types

UI - User Interface; the means by which the user and a computer system interact, in particular the use of input devices and software. In this case, the web browser of choice and the web page displayed
NoSQL - NoSQL uses JSON documents and interfaces with the IBM Cloud Cloudant database.

Source Files

Frontend Source Files
(Found in external file: 9_Source Frontend)

Backend Source Files
(Found in external file: 10_Source Backend)

How to Run the System

Our system is currently running online. You can access the home page with this link: **https://sda-site-cos-standard-fw5.s3.us-east.cloud-object-storage.appdomain.cloud/index.html**

From there, you can interact with our various services.

1. Homepage and Service Registry

The homepage contains a service registry. It calls a REST API endpoint which returns all the current online and offline services. If one of the services is offline, it will disappear from this list as well as the navigation bar.
From here, you can click on any service in the navigation bar.

2. Stock Query

Here, you can query using any stock ticker. It will return all trades for that company starting from the latest one.

3. Company Query

Here you can enter a stock ticker to see the name of companies whose tickers contain your specified substring.

4. Flag Suspects

This service allows the user to query the database for trades that may be considered suspicious. The dataset provided had no suspicious trades, so we manually flagged a couple for example. You can query using the ‘aat’ stock ticker.

5. Stock Analysis

Here, you can enter a stock ticker to retrieve some statistics about its recent performance.

6. Stock Visualization

Here, you can enter a stock ticker to generate a graph of recent stock trade sales.

7. Exchange Visualization

Select an exchange to view a graph of its most recent stock trades.

8. Company Comparison

The Company Comparison feature allows the user to visualize two companies' stock prices over time, which is displayed in a JavaScript chart, calculated on the server-side and rendered on the client's machine.

9. Currency Conversion

Here, you can choose which currency to convert, as well as the amount you would like to have converted.

How to Create and Configure the System Yourself

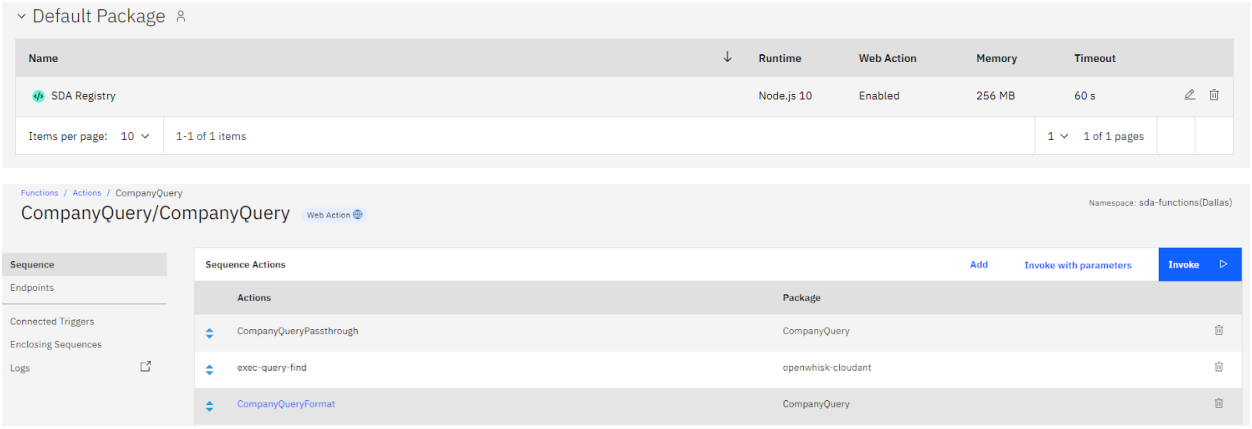
It will be more difficult to replicate your own version of the system, as you will need to recreate the various databases, reupload the cloud functions, and host the static frontend website in another instance. But if you do still wish to do it, here are the instructions to do so.

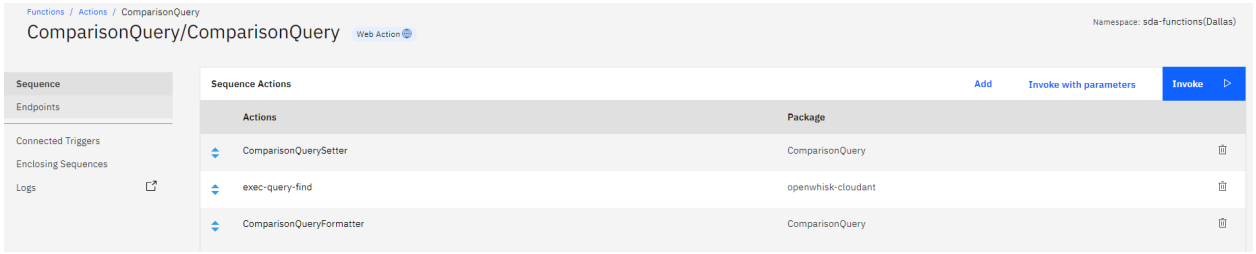
Static Website

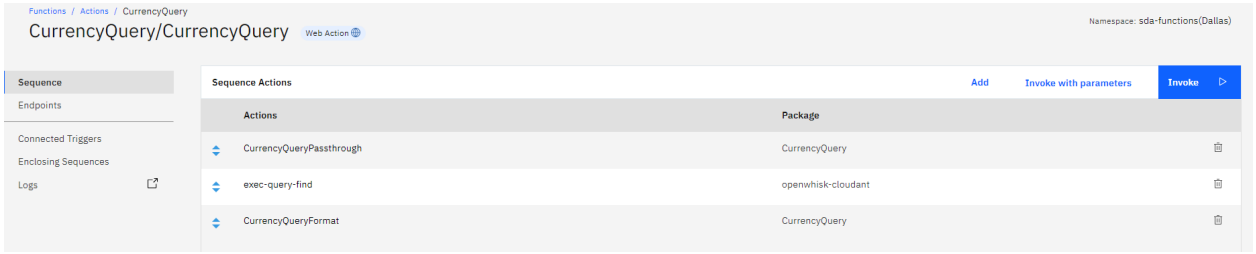
1. Host the static frontend on some server. We currently use the IBM Cloud Object Storage.

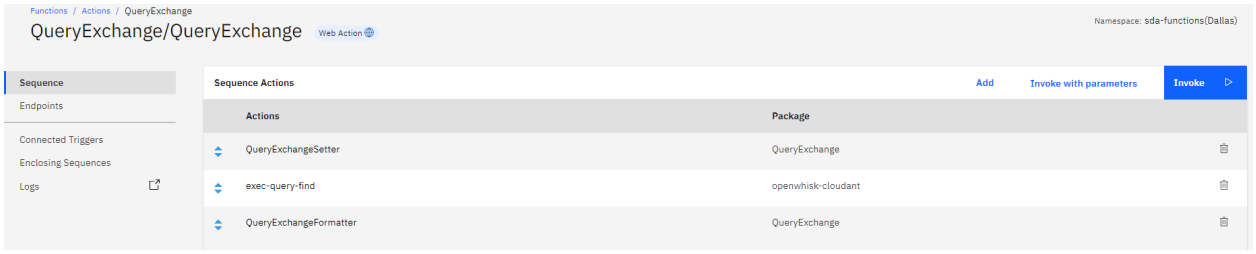
Cloud Functions

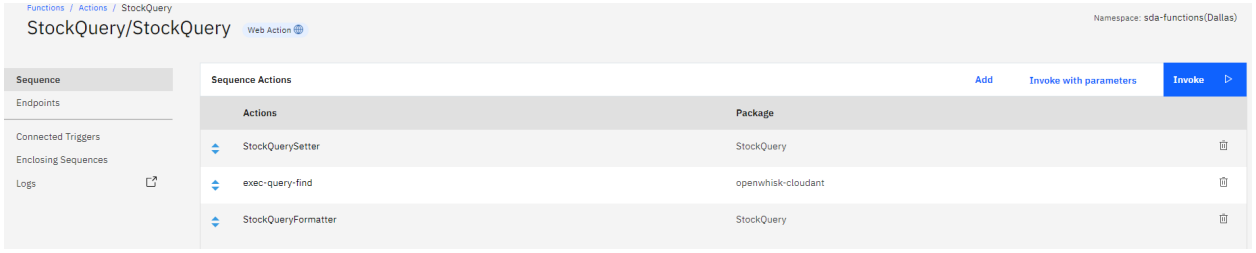
1. Create an IBM Cloud Functions Namespace
2. Create the following sets of function sequences. (Source files are in included in this documentation)



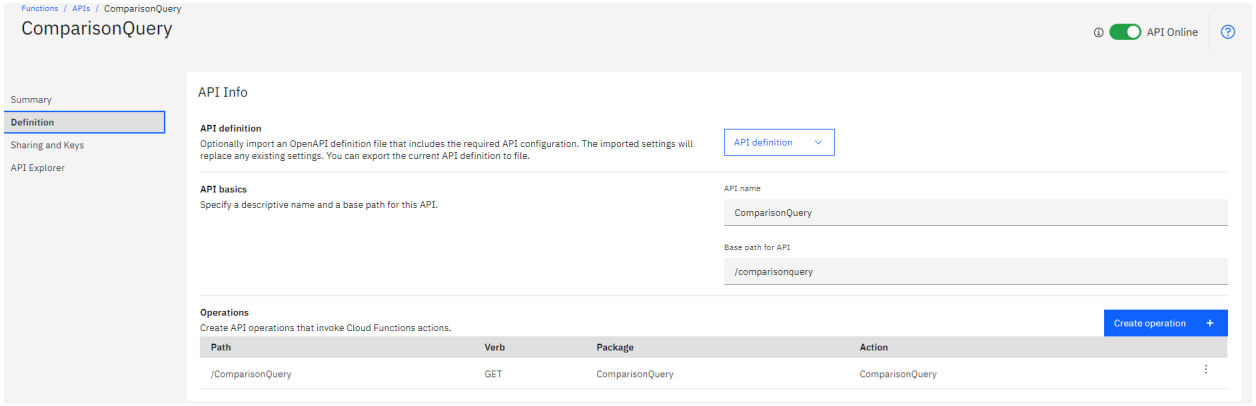


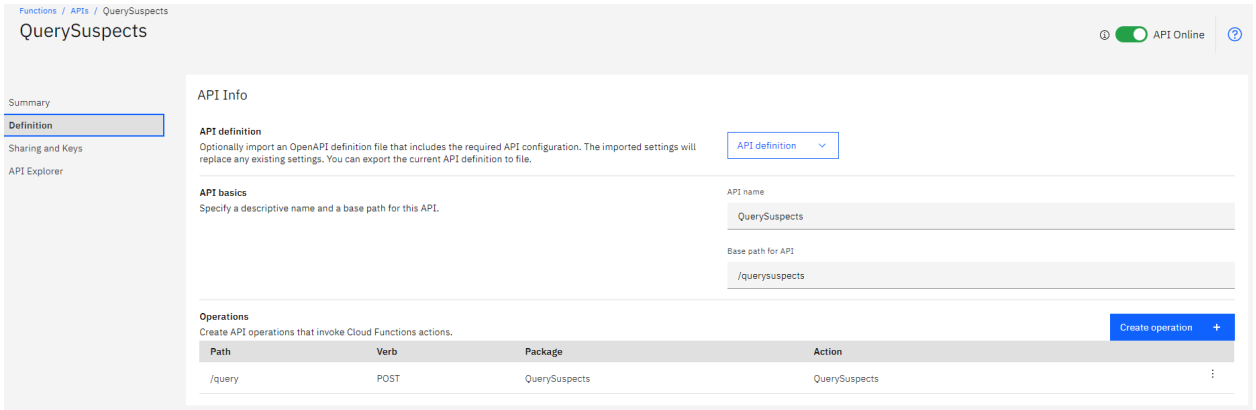


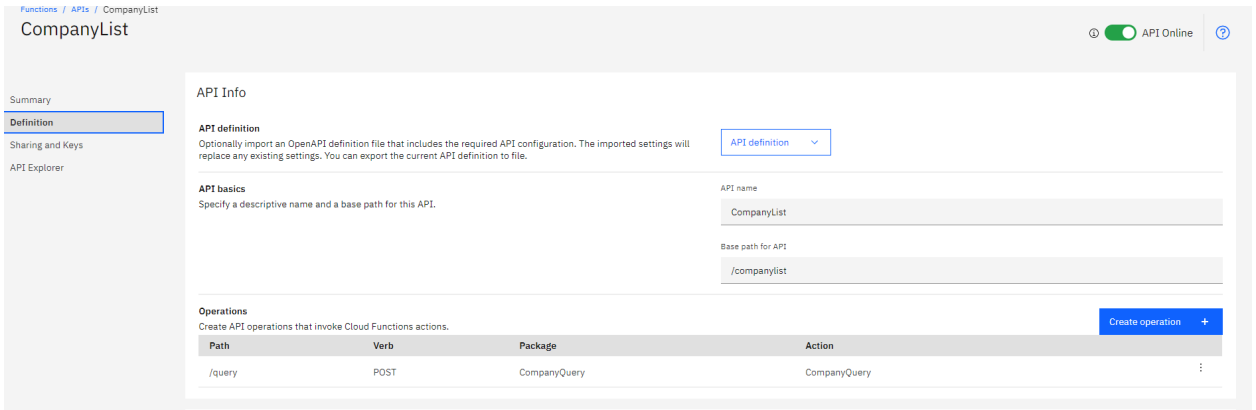


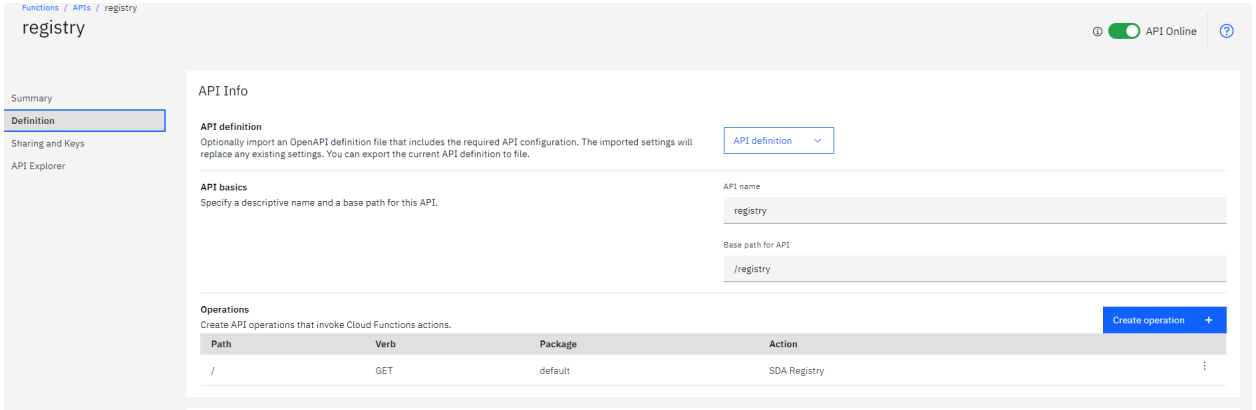


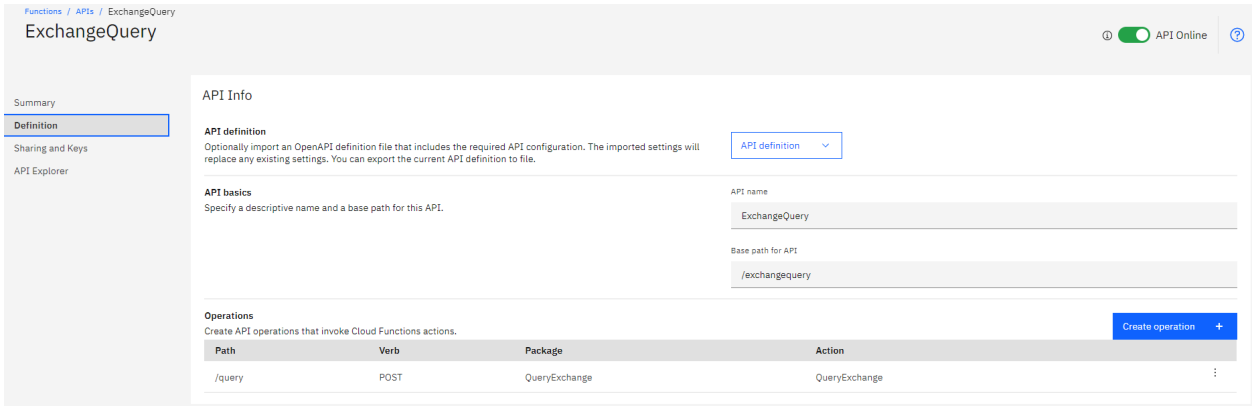
3. Then, set up the proper function APIs as follows.

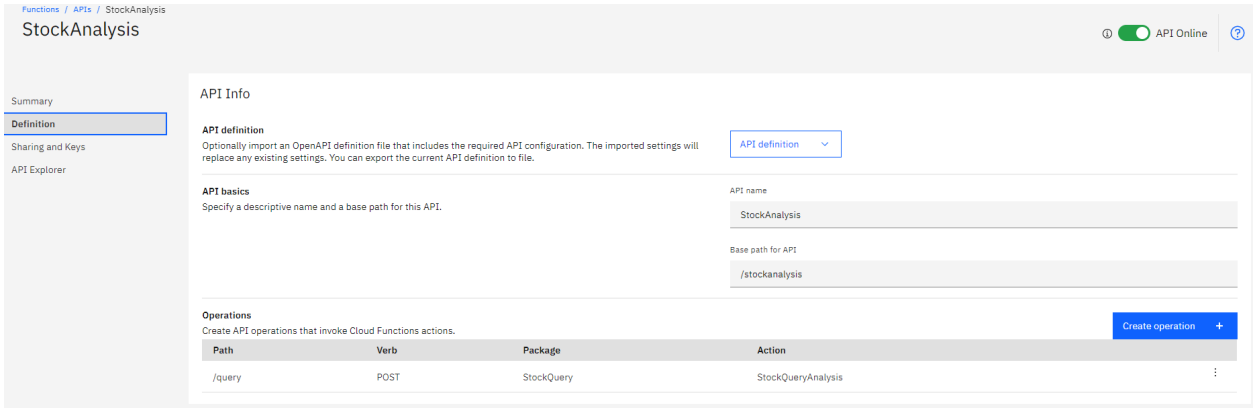

Functions / APIs / ComparisonQuery
ComparisonQuery
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
ComparisonQuery
Base path for API
/comparisonquery
Operations
Create API operations that invoke Cloud Functions actions.
Create operation +
Path
Verb
Package
Action
/ComparisonQuery
GET
ComparisonQuery
ComparisonQuery

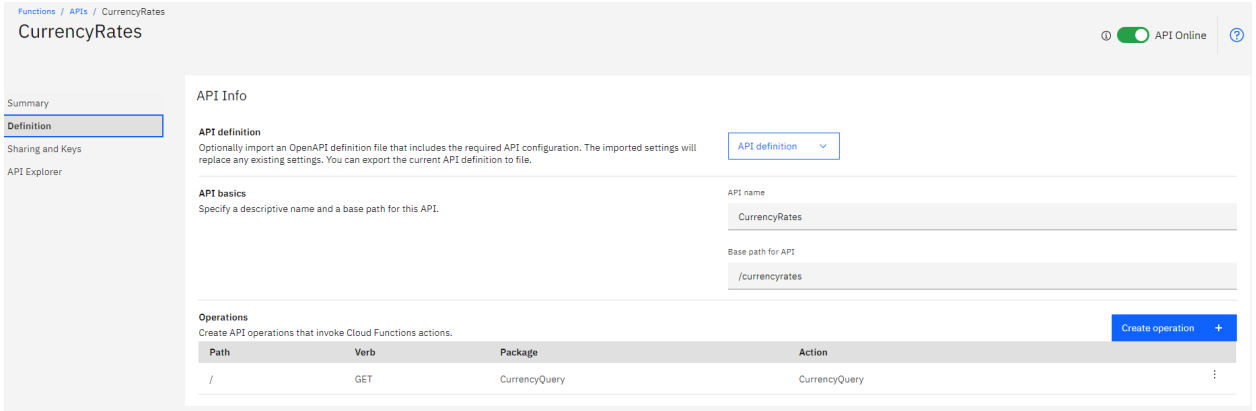

Functions / APIs / QuerySuspects
QuerySuspects
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
QuerySuspects
Base path for API
/querysuspects
Operations
Create API operations that invoke Cloud Functions actions.
Create operation +
Path
Verb
Package
Action
/query
POST
QuerySuspects
QuerySuspects

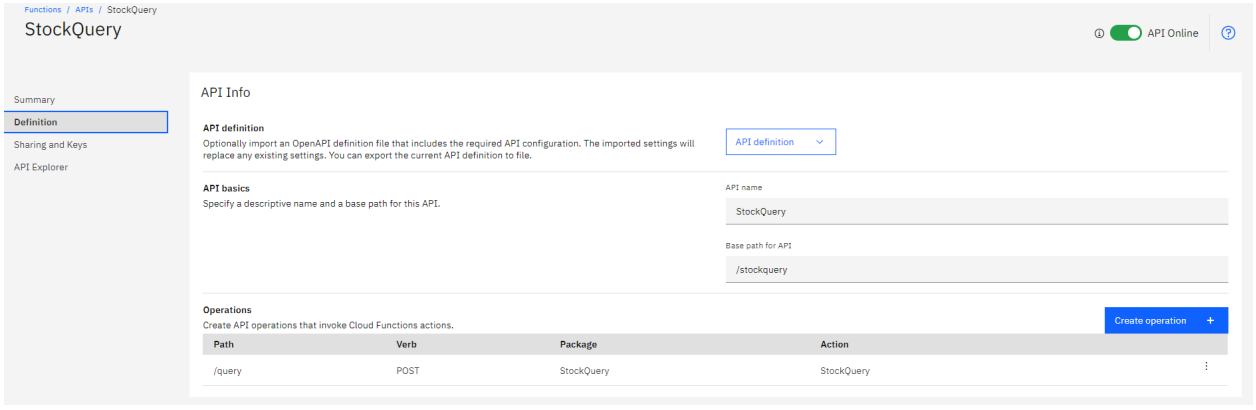

Functions / APIs / CompanyList
CompanyList
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
CompanyList
Base path for API
/companylist
Operations
Create API operations that invoke Cloud Functions actions.
Create operation +
Path
Verb
Package
Action
/query
POST
CompanyQuery
CompanyQuery


Functions / APIs / registry
registry
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
registry
Base path for API
/registry
Operations
Create API operations that invoke Cloud Functions actions.
Create operation +
Path
Verb
Package
Action
/
GET
default
SDA Registry


Functions / APIs / ExchangeQuery
ExchangeQuery
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
ExchangeQuery
Base path for API
/exchangequery
Operations
Create API operations that invoke Cloud Functions actions.
Create operation
Path
Verb
Package
Action
/query
POST
QueryExchange
QueryExchange


Functions / APIs / StockAnalysis
StockAnalysis
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
StockAnalysis
Base path for API
/stockanalysis
Operations
Create API operations that invoke Cloud Functions actions.
Create operation
Path
Verb
Package
Action
/query
POST
StockQuery
StockQueryAnalysis


Functions / APIs / CurrencyRates
CurrencyRates
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
CurrencyRates
Base path for API
/currencyrates
Operations
Create API operations that invoke Cloud Functions actions.
Create operation
Path
Verb
Package
Action
/
GET
CurrencyQuery
CurrencyQuery


Functions / APIs / StockQuery
StockQuery
API Online
Summary
Definition
Sharing and Keys
API Explorer
API Info
API definition
Optionally import an OpenAPI definition file that includes the required API configuration. The imported settings will replace any existing settings. You can export the current API definition to file.
API definition
API basics
Specify a descriptive name and a base path for this API.
API name
StockQuery
Base path for API
/stockquery
Operations
Create API operations that invoke Cloud Functions actions.
Create operation
Path
Verb
Package
Action
/query
POST
StockQuery
StockQuery

4. Finally, make sure that all the frontend requests are properly routed to their respective proper cloud function APIs URLs.

Cloudant NoSQL

1. Create a Cloudant NoSQL resource on IBM Cloud.
2. Create 3 databases:
    a. Company, which should have values
        i. COMNAM (Company Name)
        ii. PERMTICK (Company Ticker)
    b. Trades, which should have values
        i. Time (See TickView Documentation)
        ii. Price (See TickView Documentation)
        iii. Size (See TickView Documentation)
        iv. Exchange (See TickView Documentation)
        v. Sale Condition (See TickView Documentation)
        vi. Suspicious (See TickView Documentation)
        vii. Company (Company Ticker)
        viii. Date (See TickView Documentation)
    c. Currency
        i. Country (Country Name)
        ii. CurrencyRate (Relative to USD)
3. Fill the databases with entries to your liking.
4. Make sure your cloud functions are configured with the correct database credentials.