Koyfin Data Processing

Overview:

Designed a data processing solution using Spring-boot Microservice application with relational MYSQL db. The application streams data files from a landing zone and loads them to a mysql database. A REST api is built on top of the database to serve data as requested.

Features:

- Application listens for files in a landing zone and process them to the database.
- In case of errors, data file is sent to an error folder to be checked.
- In case of success, the data file is moved to an archived folder.
- Rest API that serves data in time series to be easily graphed.
- Application is designed to be easily scaled vertically or horizontally.
- Application is designed for collaboration and easily adding different sources

How Is it Built:

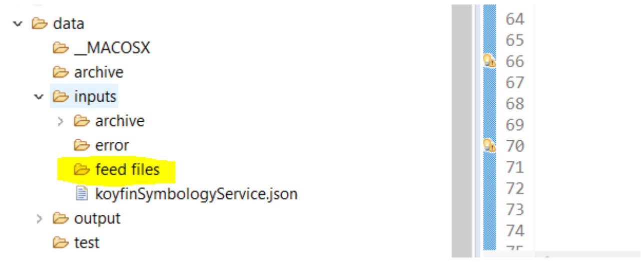
Database: Using functionality of computed values in mysql db to compute the adjusted price when a change occurs, and easily joins tables together for data analysis. Suggested to have a table for each source, however the current design also supports one table for all sources.

For the application, I use Spring-boot which is mainly used for microservice architecture. The application has a cache value to keep track for the max adjusted value. In case of failure, the application query the table to obtain the most recent value and save it in the cache. Data is read from a landing zone location, parsing the json, extracting the values and finally sending the data to the database. Once the data is processed. The file gets sent to the archive folder and a log is written to indicate the status of the processed file. The application listens to the landing zone for any change to be able to process the next ones. The same scenario in case of failures. All exceptions are captured and delivered in a log error message, and the file is moved to the error landing zone to be reprocessed.

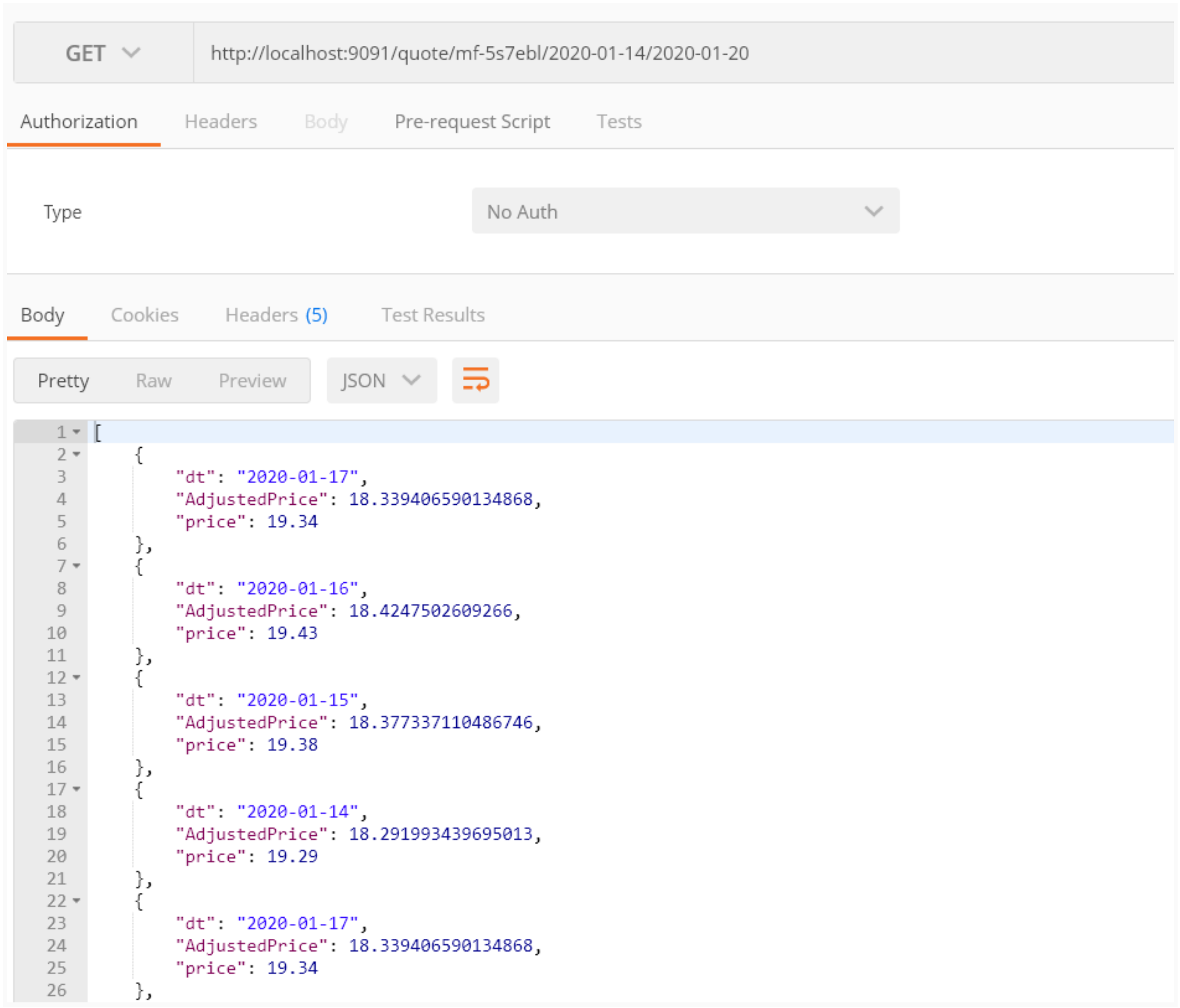
API: is a rest API that takes KID, start date and end date and serves back the price and the computed adjusted price. The API checks the data before processing them to the db. Returning an appropriate error response is very important and it’s properly handled in the application.

Usage:

- Set up a mysql DB (tables schemas will be provided later)
- Run the java application using maven (mvn spring-boot:run)
- In the project folder, look for data Folder, and add your file you want to process



-
- For the API, hit the localhost url and provide the (KID),(startDate) and (endDate) of your time series Ex. http://localhost:9091/quote/mf-5s7ebl/2020-01-14/2020-01-14



```
GET http://localhost:9091/quote/mf-5s7ebl/2020-01-14/2020-01-20
```

Type: No Auth

```
[
    {
        "dt": "2020-01-17",
        "AdjustedPrice": 18.339406590134868,
        "price": 19.34
    },
    {
        "dt": "2020-01-16",
        "AdjustedPrice": 18.4247502609266,
        "price": 19.43
    },
    {
        "dt": "2020-01-15",
        "AdjustedPrice": 18.377337110486746,
        "price": 19.38
    },
    {
        "dt": "2020-01-14",
        "AdjustedPrice": 18.291993439695013,
        "price": 19.29
    },
    {
        "dt": "2020-01-17",
        "AdjustedPrice": 18.339406590134868,
        "price": 19.34
    },
```

GOAL:

- Here is a design of the mysql db schema:

```
create table marketData(
   dt DATE NOT NULL,
   assetId INT,
   price DOUBLE,
   maxAdjustedNav DOUBLE,
   adjustedNav DOUBLE,
   AdjustedPrice DOUBLE GENERATED ALWAYS AS
(price*adjustedNav/maxAdjustedNav) STORED,
   PRIMARY KEY ( dt,assetId )
);

create table koyfinData(
      KID VARCHAR(100) NOT NULL,
      ticker VARCHAR(50),
      ISIN VARCHAR(50),
      acmeAssetID INT,
      qualifiedTicker VARCHAR(50),
      PRIMARY KEY ( KID,acmeAssetID )
);
```

- **Trade off:** for the DB choice. I was having a very hard time choosing between (mongoDB, ELasticSearch) or MYSQL DB. Mysql is very good for data analysis which Koyfin is mainly known for. Mysql is easier to query based on fields which can help giving back a certain time series through the api. Scaling a relational db could be challenging. NoSQL db has an advantage for fast retrieving, and multi clustering meaning it's easier to scale and faster to fetch data as long as you know the primary key. IN our case, our data is represented by the KID which references the ISIN, in the case joining tables might not be easily achievable in a nosql db. Finally, in noSQL db data is returned in a json format which can be easily served over the REST API. FOr SQL db, I had to write a class to convert result set to json format.
- For the application, The main trade off is doing the calculation (adjPrice(t) = marketPrice(t) * adjFactor(t) / adjFactor(max(t))) for every request vs a change every time there is an update. Calculation on the client side may slow down the request, however the chance of data change is low. On sql sied, you may consume un updated data. Since the frequency of the adjFactor was low, I choose to do the calculation and update the table data on the server side.

- **TESTING:**
    1. Creating a test plan for the application.
    2. Build test cases from a client prospective.
    3. Automate these test cases. In our case using Junit in the test suite built in the application to support CI/CD.
    4. Perform functional tests to make sure the requirements are met.
    5. Perform a break test to look for unhandled exceptions, and break points
    6. Using SONAR, perform a security and vulnerability test.
    7. Finally, perform a load/stress test to test the process performance.
- **Monitoring the System**:

  There are so many different ways to do monitoring. Adding Logs to different components of the system.

  Overview:
  In the application,build an actuator to reflect matrices of when we receive bad data, missing points and application health. Using Prometheus and alert manager, we read those matrices and alert the team based on the configuration. The same thing for infrastructure. Using kubernetes, export health metrics to Prometheus and use alert manger to distribute the alerts to the engineer.

  Secondly also possible to
    1. Check the data points and write a log if something is missing based on a predetermined logic.
    2. When processing the data write logs when bad data/ parse exceptions.
    3. Watch for exceptions due to outings or performance issues.

Using Logstash to collect the logs which are written in a certain format. Stream the logs using (Kafka) for example from the application to ElasticSearch. From elastic build rules to read certain events and alert the team via emails when certain events occur while indexing.

Finally using kibana to view the performance of the applications which makes it easy and simple to handle.