Yfinance Stock Price Alarm Processor

In this code we use YFinance API to extract, in real time, the current stock price of a list of entities, (as of now: meta, ibm, amazon, apple, general motors and vanguard_etf), continuously processing their current-price moving average for 5, 10 and 15 minute windows. Duplicates, i.e., input values from YFInance with the same update time, are filtered and ignored in the ETL process.

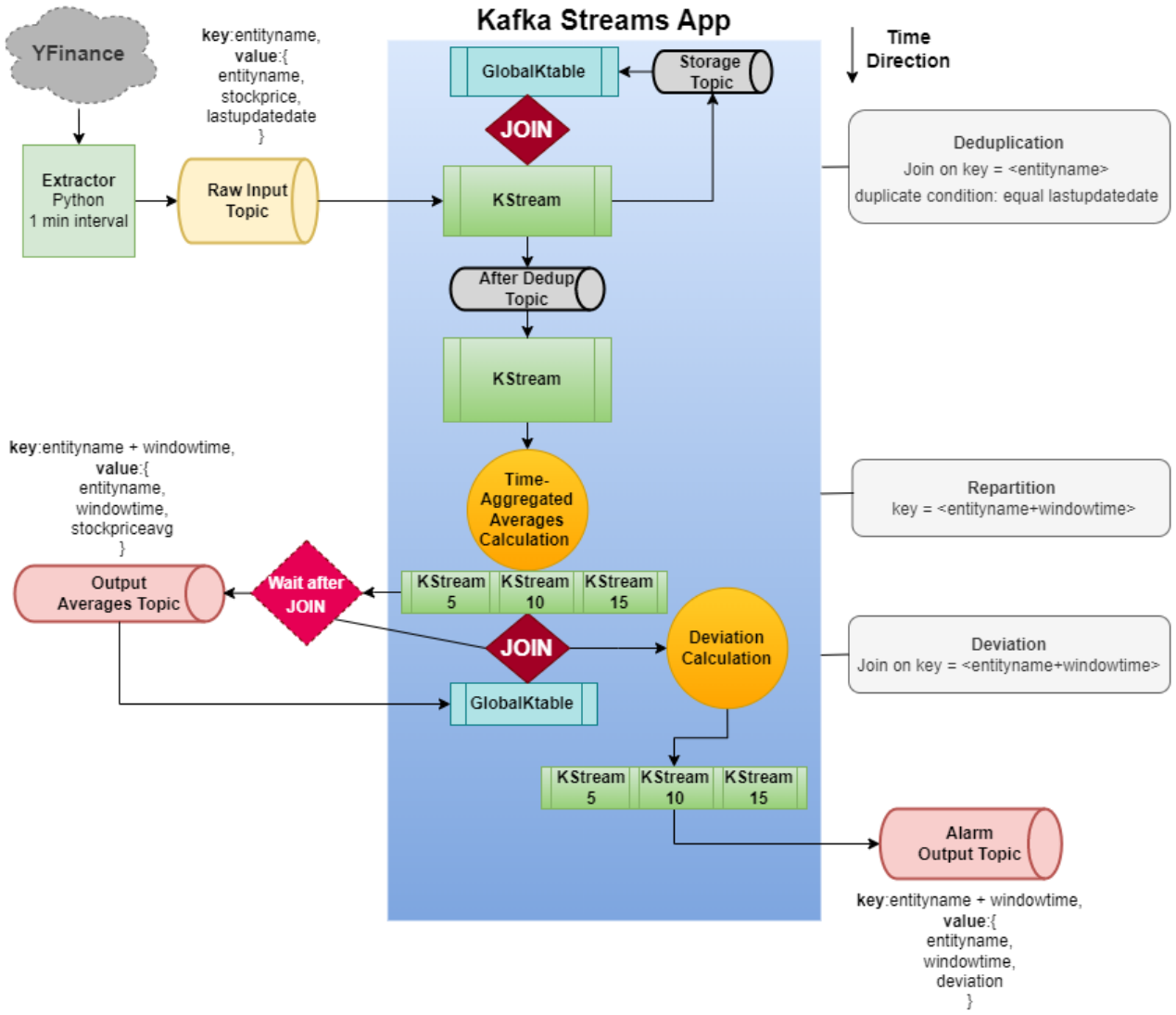
For each new ocurrence, and for each average-window, we compare the newest average aggregation with the previous stored one, computing its absolute deviation from the latter. If the deviations are larger than a user-defined tolerance, an alarm is produced.

Technologies:

- Extractor is done with python3.
- Data is stored using Kafka.
- ETL is performed using Kafka Streams.
- Environment is local and self-contained, deployed and managed using kubernetes in a minikube cluster and docker images.

1 - Topology and functional architecture

- Functional/topology diagram:



- Functional steps:
  - YFinance current stock price and its last update date are fetched using YFinance API and a Python extractor. Data is fetched for several entities once each minute into a Kafka topic, which will feed the Kafka Streams application.
  - Kafka Streams application will have three main functional stages:
    a. Deduplication of events with the same entity and last update date.
    b. Moving average aggregation of the current stock price into 5, 10 an 15-minute windows.
    c. Calculation of the deviation between the ongoing average (calculated with the ongoing event) and last calculated one (without the ongoing event), sending it into an alarm-output kafka topic if the deviation, calculated for each windowing time, is greater than a given user tolerance.
  - Steps a. and c. are stateful operations, and therefore a GlobalKTable is used to store the state of the exact-previous event, needed for the deduplication and deviation calculus, respectively.
  - Output topics are the averages and the alarm outputs. All other topics are internal.

2 - Pre-requirements

The following pre-requirements are needed to configure and execute the integration tests:

- Docker engine

- Maven
- Helm
- Minikube
- Kubectl
- Python3.9

3 - Running instructions

- Install dependencies stated on **2**.
- Start minikube:

```
minikube start
```

- Set the tolerance values for each window within <root>/DockerizedApp/yfinance.properties .
- Open a new shell and go to <root>/DockerizedApp. The following commands will dockerize your application in the minikube docker environment, so that the kubernetes deployment can recognize and find the app docker image:

```
eval $(minikube docker-env)
docker build -t yfinance-stock-price-alarm-processor-001 .
```

- Open a new shell and go to <root>/DockerizedPythonExtractor. The following commands will do the same as above but for the python extractor:

```
eval $(minikube docker-env)
docker build -t yfinance-stock-price-alarm-processor-001 .
```

- In project root, to start the kubernetes deployment with the helm command:

```
helm install --debug yfinance-stock-price-alarm-processor yfinance-stock-price-alarm-processor
```

4 - Functional validation

Run:

```
eval $(minikube docker-env)
docker ps
```

and search for the container id with name with the form 'k8s_server-k8s_server-k8s-deployment-< hashcode >'. Using that container id run:

```
docker exec -it <container_id> bash
```

Inside the container bash create a consumer for a relevant topic, running:

```
kafka-console-consumer --bootstrap-server server-k8s-service:9092 --topic <topic_name> --formatter
```

where < topic_name > may be:

- yfinance-averages-output (output)
- yfinance-deviation-alarm (output)
- yfinance-raw-input (input)
- yfinance-after-dedup-storage-table (intermediary)
- yfinance-after-dedup-stream (intermediary)

You should see the data being consumed by the above-mentioned topics in real-time. Take in consideration that some input data may not be consumed by some topics due to the input tolerances given, or due to duplication filtering.

3 - Installing instructions

If you want to change the source code, in the root folder, run:

```
mvn clean
mvn package
```

run:

```
mv -i ./target/BdeOnBoardingExerciseFirstDraft-1.0-SNAPSHOT-jar-with-dependencies.jar ./Dockerize
```

and follow the running instructions **3** to run the new code.