

KPMG TEST CASE DOCUMENTATION

PREPARED BY:

DAMLA BUSRA
OZSONMEZ

1. INTRODUCTION

Real time data streaming has become very popular in recent years. The importance of latency is increased and the integration of different sources becomes inevitable. Apache Kafka, RabbitMQ, Redis, Apache NiFi, Hive, Spark are the some of the tools that can be used in a real-time data project. In this project, a data processing service is asked to be done using Apache Kafka as a publish/subscribe system, a loader (from an online streaming source) into Kafka Topic, a consumer from the Kafka topic and a database as the final point with Docker images. Apache Nifi is selected as the data loader (publisher) and Spark streaming as the consumer. Due to the fact that I could not get the permission for the Twitter API, I used **AviationStack** API as the streaming source which provides flight informations in the world.

<https://aviationstack.com/documentation>

I had experience with Apache Nifi, I had also examined an implemented Kafka environment a bit, but I had never implemented Kafka-Nifi-Spark and database all in one with Docker. So, I did some research and decided to implement the whole data streaming on a Linux environment.

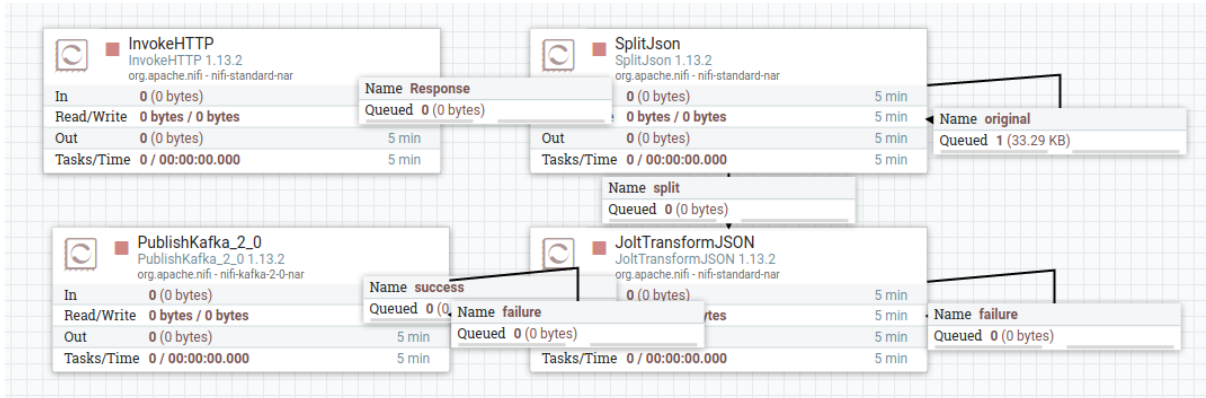
1. Requirements

- Java (Java 11)
 - JAVA_HOME must be set
- Docker & Docker Compose

2. IMPLEMENTATION

Apache NiFi is selected as the producer for Kafka. The first step is to design the data loading system in NiFi. The processors are used to

manipulate the data stream. <http://localhost:8080/nifi/> can be used (8080 is the default port for NiFi) to access the User Interface (UI).



InvokeHTTP is a processor which is used instead of *GetHTTP* processor. It provides collecting data from a website/API. The API URL (http://api.aviationstack.com/v1/flightsaccess_key=ACCESS_KEY) is used. The API Response looks like this:

```
View as: formatted
1 {
2   "pagination": {
3     "limit": 40,
4     "offset": 0,
5     "count": 40,
6     "total": 280209
7   },
8   "data": [
9     {
10      "flight_date": "2021-04-19",
11      "flight_status": "scheduled",
12      "departure": {
13        "airport": "Yantai",
14        "timezone": "Asia/Shanghai",
15        "iata": "YNT",
16        "icao": "ZSYT",
17        "terminal": null,
18        "gate": null,
19        "delay": null,
20        "scheduled": "2021-04-19T11:55:00+00:00",
21        "estimated": "2021-04-19T11:55:00+00:00",
22        "actual": null,
23        "estimated_runway": null,
24        "actual_runway": null
25      },
26      "arrival": {
27        "airport": "Zhengzhou",
28        "timezone": "Asia/Shanghai",
29        "iata": "CGO",
30        "icao": "ZHCC",
31        "terminal": "T2",
32        "gate": null,
33        "baggage": null,
34        "delay": null,
35        "scheduled": "2021-04-19T13:45:00+00:00",
36        "estimated": "2021-04-19T13:45:00+00:00",
37        "actual": null,
38        "estimated_runway": null,
39        "actual_runway": null
40      },
41      "airline": {
42        "name": "China Eastern Airlines",
43        "iata": "MU",
44        "icao": "CES"
45      },
46      "flight": {
47        "number": "5575",
48        "iata": "MUS575",
49        "icao": "CES5575",
50        "codeshared": null
51      },
52      "aircraft": null,
53      "live": null
54    },
55    {
56      "flight_date": "2021-04-19",
57      "flight_status": "scheduled",
58      "departure": {
59        "airport": "Weipa",
60        "timezone": "Australia/Brisbane",
61        "iata": "WEI",
62        "icao": "YBWP",
63        "terminal": null,
64        "gate": null,
65        "delay": null,
66        "scheduled": "2021-04-19T12:40:00+00:00",
67        "estimated": "2021-04-19T12:40:00+00:00",
68        "actual": null,
69        "estimated_runway": null,
70        "actual_runway": null
71      },
72      "arrival": null,
73      "airline": null,
74      "flight": null,
75      "aircraft": null,
76      "live": null
77    }
78  ]
79 }
```

Since the “data” part is to be used, *SplitJson* processor is used as follows:

SETTINGS	SCHEDULING	PROPERTIES	COMMENTS
Required field			
Property		Value	
JsonPath Expression		\$.data	
Null Value Representation		empty string	

The result is as follows:

```
1 {
2   "flight_date" : "2021-04-19",
3   "flight_status" : "scheduled",
4   "departure" : {
5     "airport" : "Yantai",
6     "timezone" : "Asia/Shanghai",
7     "iata" : "YNT",
8     "icao" : "ZSYT",
9     "terminal" : null,
10    "gate" : null,
11    "delay" : null,
12    "scheduled" : "2021-04-19T11:55:00+00:00",
13    "estimated" : "2021-04-19T11:55:00+00:00",
14    "actual" : null,
15    "estimated_runway" : null,
16    "actual_runway" : null
17  },
18  "arrival" : {
19    "airport" : "Zhengzhou",
20    "timezone" : "Asia/Shanghai",
21    "iata" : "CGO",
22    "icao" : "ZHCC",
23    "terminal" : "T2",
24    "gate" : null,
25    "baggage" : null,
26    "delay" : null,
27    "scheduled" : "2021-04-19T13:45:00+00:00",
28    "estimated" : "2021-04-19T13:45:00+00:00",
29    "actual" : null,
30    "estimated_runway" : null,
31    "actual_runway" : null
32  },
33  "airline" : {
34    "name" : "China Eastern Airlines",
35    "iata" : "MU",
36    "icao" : "CES"
37  },
38  "flight" : {
39    "number" : "5575",
40    "iata" : "MU5575",
41    "icao" : "CES5575",
42    "codeshared" : null
43  },
44  "aircraft" : null,
45  "live" : null
46 }
```

Since dealing with nested json array can cause problems, it makes more sense to use single level JSON. *JoltTransformJSON* is used to select the key values and to make the JSON object with a single level. *JoltSpecification* is configured as follows:

```
{
  "operation": "shift",
  "spec": {
    "flight_date": "flight_date",
    "flight_status": "flight_status",
    "departure": {
      "airport": "dep_airport",
      "timezone": "dep_timezone",
      "scheduled": "dep_scheduled",
      "estimated": "dep_estimated"
    },
    "arrival": {
      "airport": "arr_airport",
      "timezone": "arr_timezone",
      "terminal": "arr_terminal",
      "gate": "arr_gate"
    },
    "airline": {
      "name": "airline_name"
    },
    "flight": {
      "number": "flight_number"
    },
    "aircraft": {
      "registration": "aircraft_registration"
    },
    "live": {
      "updated": "LiveUpdated",
      "latitude": "latitude",
      "longitude": "longitude",
      "altitude": "altitude",
      "direction": "direction",
      "speed_horizontal": "speedHorizontal",
      "speed_vertical": "speedVertical",
      "is_ground": "isGround"
    }
  }
}
```

```
1 {
2   "flight_date" : "2021-04-19",
3   "flight_status" : "scheduled",
4   "dep_airport" : "Yantai",
5   "dep_timezone" : "Asia/Shanghai",
6   "dep_scheduled" : "2021-04-19T11:55:00+00:00",
7   "dep_estimated" : "2021-04-19T11:55:00+00:00",
8   "arr_airport" : "Zhengzhou",
9   "arr_timezone" : "Asia/Shanghai",
10  "arr_terminal" : "T2",
11  "arr_gate" : null,
12  "airline_name" : "China Eastern Airlines",
13  "flight_number" : "5575"
14 }
```

After extracting the relevant information from the data, we must publish the JSON flowfiles to a Kafka Topic. Since the version of Kafka is 2.0, the processor *PublishKafka_2.0* is used with the Kafka topic and the IP/PORT of the topic as Kafka Brokers.

3. ZOOKEEPER/KAFKA/NIFI & DOCKER

Docker is used to create images from Kafka. As a starting point, the Docker Kafka image of *Wurstmeister* (<https://github.com/wurstmeister/kafka-docker>) is cloned. The **docker-compose.yml** file contain the configuration for the images that will be created for Kafka/Zookeeper & Apache NiFi. After setting the configurations in this file, the Docker Compose must be running with the command **docker-compose up -d**. We can check the states using **docker-compose ps**.

```
busra@busra-TULPAR-T5-V1:~/kafka-docker$ docker-compose ps
```

Name	Command	State	Ports
kafka-docker_kafka_1	start-kafka.sh	Up	0.0.0.0:9094->9094/tcp,:::9094->9094/tcp
kafka-docker_nifi_1	./scripts/start.sh	Up	10000/tcp, 8000/tcp, 0.0.0.0:8080->8080/tcp,:::8080->8080/tcp, 8443/tcp
kafka-docker_zeppelin_1	/usr/bin/tini -- bin/zeppe ...	Up	8080/tcp, 0.0.0.0:8181->8181/tcp,:::8181->8181/tcp
kafka-docker_zookeeper_1	/bin/sh -c /usr/sbin/sshd ...	Up	0.0.0.0:2181->2181/tcp,:::2181->2181/tcp, 22/tcp, 2888/tcp, 3888/tcp

If the containers are healthy (up), we can create a Kafka Topic. First, we have to access the container using `docker exec -it <<CONTAINER_NAME>> bin/sh`. The command `/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic <<TOPIC_NAME>>` is used to create a topic in the container. The created topics can be listed as follows:

```
/opt/kafka_2.13-2.7.0 # ./bin/kafka-topics.sh --list --zookeeper zookeeper:2181
__consumer_offsets
__transaction_state
aviation_topic
test_aviation
```


After NiFi publishes the data to the Kafka Topic, we can observe the messages sent to Kafka as follows:

```
/opt/kafka_2.13-2.7.0 # ./bin/kafka-console-consumer.sh --bootstrap-server kafka:9092 --topic test_aviation --from-beginning --max-messages 10
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Tolmachevo","dep_timezone":"Asia/Novosibirsk","arr_airport":"Domodedovo",
":null","airline_name":"Royal Air Maroc","flight_number":"9006"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Tolmachevo","dep_timezone":"Asia/Novosibirsk","arr_airport":"Domodedovo",
":null","airline_name":"Belavia","flight_number":"152"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Tolmachevo","dep_timezone":"Asia/Novosibirsk","arr_airport":"Domodedovo",
":null","airline_name":"El Al","flight_number":"8978"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Tolmachevo","dep_timezone":"Asia/Novosibirsk","arr_airport":"Domodedovo",
":null","airline_name":"TAP Air Portugal","flight_number":"8157"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Tolmachevo","dep_timezone":"Asia/Novosibirsk","arr_airport":"Domodedovo",
":null","airline_name":"Etihad Airways","flight_number":"8693"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Xichang","dep_timezone":"Asia/Shanghai","arr_airport":"Nanning","arr_time
airline_name":"Sichuan Airlines","flight_number":"8025"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Vnukovo","dep_timezone":"Europe/Moscow","arr_airport":"Ufa International
":1","arr_gate":null,"airline_name":"Turkish Airlines","flight_number":"9048"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Abu Dhabi International","dep_timezone":"Asia/Dubai","arr_airport":"Schip
arr_gate":null,"airline_name":"Etihad Airways","flight_number":"77"}
{"flight_date":"2021-04-18","flight_status":"active","dep_airport":"Abu Dhabi International","dep_timezone":"Asia/Dubai","arr_airport":"Frankfur
r_terminal":"1","arr_gate":"B48A","airline_name":"Malaysia Airlines","flight_number":"5289"}
{"flight_date":"2021-04-18","flight_status":"scheduled","dep_airport":"Domodedovo","dep_timezone":"Europe/Moscow","arr_airport":"Orenburg","arr_
":null,"airline_name":"Smartavia","flight_number":"231"}
```

3.SPARK & ZEPPELIN

← → ↻ 🏠

localhost:9090

 2.4.3

Spark Master at spark://e609316547fb:7077

URL: spark://e609316547fb:7077

Alive Workers: 1

Cores in use: 1 Total, 0 Used

Memory in use: 1024.0 MB Total, 0.0 B Used

Applications: 0 Running, 0 Completed

Drivers: 0 Running, 0 Completed

Status: ALIVE

Workers (1)

Worker Id	Address	State	Cores
worker-20210419024658-172.18.0.6-38233	172.18.0.6:38233	ALIVE	1 (0 Used)


Running Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	Us
----------------	------	-------	---------------------	----------------	----

Completed Applications (0)

Application ID	Name	Cores	Memory per Executor	Submitted Time	Us
----------------	------	-------	---------------------	----------------	----

← → ↻ 🏠 localhost:8585/#/notebook/2G2XUMTY6

 **Zeppelin** Notebook ▾ Job

Untitled Note 1

▶ ⌂ 📄 ✂ 📎 ⬇ 📄 🔄 🔍 🗑

`%spark.pyspark`
`from pyspark import SparkContext`

Took 1 min 4 sec. Last updated by anonymous at April 19 2021, 11:50:11 AM.

`%sh`
`pip install kafka-python`

Collecting kafka-python
Downloading https://files.pythonhosted.org/packages/75/68/dcb0db055309f680ab2931a3eeb22d865604b638acf8c914bedf4c1a0c8c/kafka_python-2.0.2-py2.py3-none-any.whl (246kB)
Installing collected packages: kafka-python
Successfully installed kafka-python-2.0.2
You are using pip version 8.1.2, however version 21.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.

Took 12 sec. Last updated by anonymous at April 19 2021, 11:52:00 AM.

`%spark.pyspark`
`from pyspark.streaming import StreamingContext`
`from pyspark.streaming.kafka import KafkaUtils`
`import json`
`import sys`
`import datetime`

Took 1 sec. Last updated by anonymous at April 19 2021, 11:53:37 AM.

[+ Ardi Pararanh](#)

Due to the fact that the Zeppelin throws error for Kafka libraries,
I cannot go forward.