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 Apacche 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)

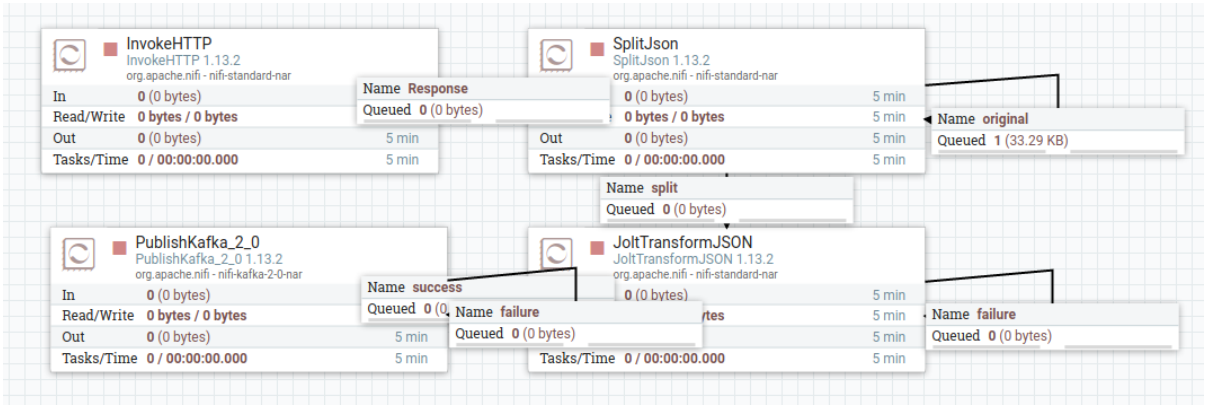
I had experience with Apache Nifi, I had also examined an implemented Kafka environment a bit, but I had never implemented Kafka-Nifi-Spark and database all in one with Docker. So, I did some research and decided to implement the whole data streaming on a Linux environment.

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).

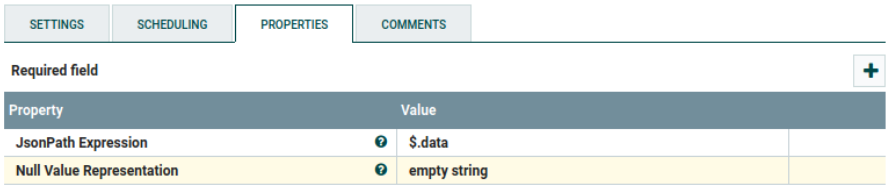


InvokeHTTP is a processor which is used instead of *GetHTTP* processor. It provides collecting data from a website/API. The API URL (http://api.aviationstack.com/v1/flightsaccess_key=ACCESS_KEY) is used. The API Response looks like this:

View as: formatted


```
{
  "pagination" : {
    "limit" : 40,
    "offset" : 0,
    "count" : 40,
    "total" : 280209
  },
  "data" : [ {
    "flight_date" : "2021-04-19",
    "flight_status" : "scheduled",
    "departure" : {
      "airport" : "Yantai",
      "timezone" : "Asia/Shanghai",
      "iata" : "YNT",
      "icao" : "ZSYT",
      "terminal" : null,
      "gate" : null,
      "delay" : null,
      "scheduled" : "2021-04-19T11:55:00+00:00",
      "estimated" : "2021-04-19T11:55:00+00:00",
      "actual" : null,
      "estimated_runway" : null,
      "actual_runway" : null
    },
    "arrival" : {
      "airport" : "Zhengzhou",
      "timezone" : "Asia/Shanghai",
      "iata" : "CGO",
      "icao" : "ZHCC",
      "terminal" : "T2",
      "gate" : null,
      "baggage" : null,
      "delay" : null,
      "scheduled" : "2021-04-19T13:45:00+00:00",
      "estimated" : "2021-04-19T13:45:00+00:00",
      "actual" : null,
      "estimated_runway" : null,
      "actual_runway" : null
    },
    "airline" : {
      "name" : "China Eastern Airlines",
      "iata" : "MU",
      "icao" : "CES"
    },
    "flight" : {
      "number" : "5575",
      "iata" : "MU5575",
      "icao" : "CES5575",
      "codeshared" : null
    },
    "aircraft" : null,
    "live" : null
  }, {
    "flight_date" : "2021-04-19",
    "flight_status" : "scheduled",
    "departure" : {
      "airport" : "Weipa",
      "timezone" : "Australia/Brisbane",
      "iata" : "WEI",
      "icao" : "YBWP",
      "terminal" : null,
      "gate" : null,
      "delay" : null,
      "scheduled" : "2021-04-19T12:40:00+00:00",
      "estimated" : "2021-04-19T12:40:00+00:00",
      "actual" : null,
      "estimated_runway" : null,
      "actual_runway" : null
```

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

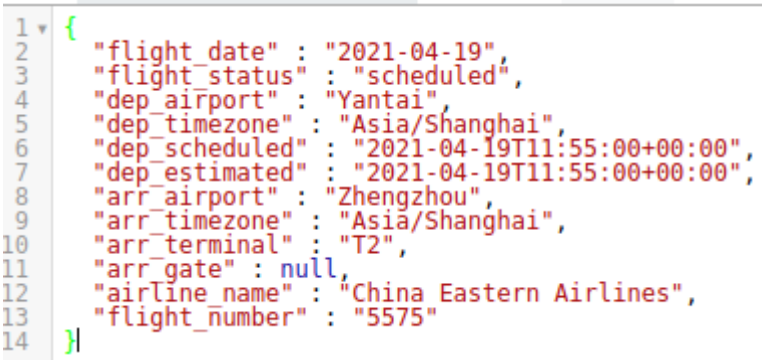


The result is as follows:


```
{
  "flight_date" : "2021-04-19",
  "flight_status" : "scheduled",
  "departure" : {
    "airport" : "Yantai",
    "timezone" : "Asia/Shanghai",
    "iata" : "YNT",
    "icao" : "ZSYT",
    "terminal" : null,
    "gate" : null,
    "delay" : null,
    "scheduled" : "2021-04-19T11:55:00+00:00",
    "estimated" : "2021-04-19T11:55:00+00:00",
    "actual" : null,
    "estimated_runway" : null,
    "actual_runway" : null
  },
  "arrival" : {
    "airport" : "Zhengzhou",
    "timezone" : "Asia/Shanghai",
    "iata" : "CGO",
    "icao" : "ZHCC",
    "terminal" : "T2",
    "gate" : null,
    "baggage" : null,
    "delay" : null,
    "scheduled" : "2021-04-19T13:45:00+00:00",
    "estimated" : "2021-04-19T13:45:00+00:00",
    "actual" : null,
    "estimated_runway" : null,
    "actual_runway" : null
  },
  "airline" : {
    "name" : "China Eastern Airlines",
    "iata" : "MU",
    "icao" : "CES"
  },
  "flight" : {
    "number" : "5575",
    "iata" : "MU5575",
    "icao" : "CES5575",
    "codeshared" : null
  },
  "aircraft" : null,
  "live" : null
}
```


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"
  }
```



```
{
  "flight_date" : "2021-04-19",
  "flight_status" : "scheduled",
  "dep_airport" : "Yantai",
  "dep_timezone" : "Asia/Shanghai",
  "dep_scheduled" : "2021-04-19T11:55:00+00:00",
  "dep_estimated" : "2021-04-19T11:55:00+00:00",
  "arr_airport" : "Zhengzhou",
  "arr_timezone" : "Asia/Shanghai",
  "arr_terminal" : "T2",
  "arr_gate" : null,
  "airline_name" : "China Eastern Airlines",
  "flight_number" : "5575"
}
```

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
```

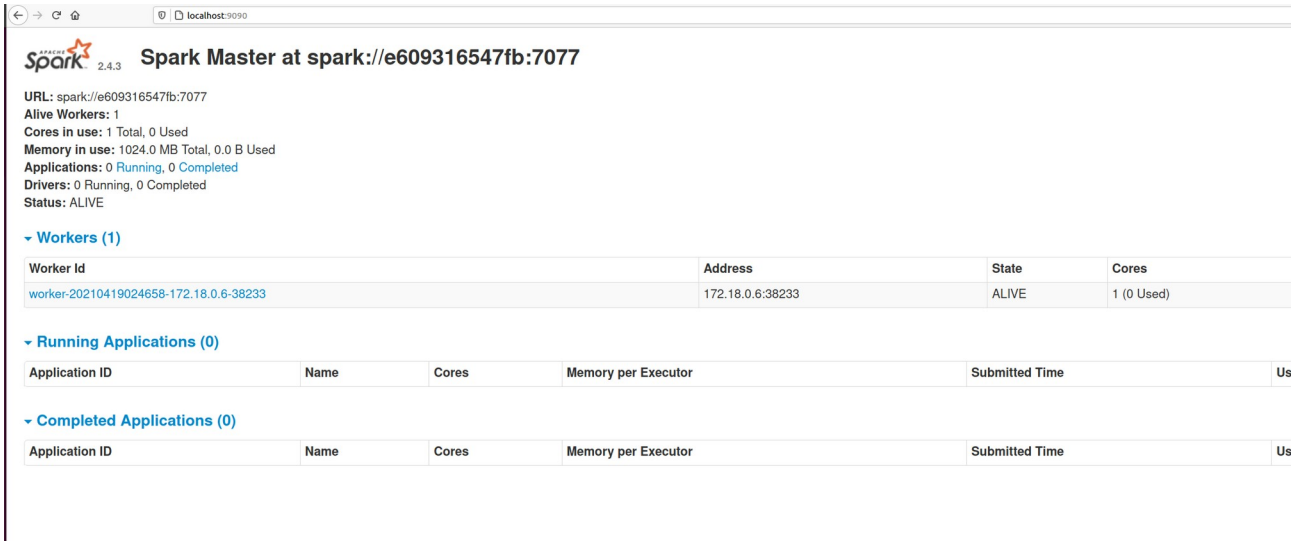
If the containers are healthy (up), we can create a Kafka Topic. First, we have to access the container using **docker exec -it <<CONTAINER_NAME>> *bin*/sh.** The command **/bin/kafka-topics.sh --create --zookeeper zookeeper:2181 --replication-factor 1 --partitions 1 --topic <<TOPIC_NAME>>** is used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

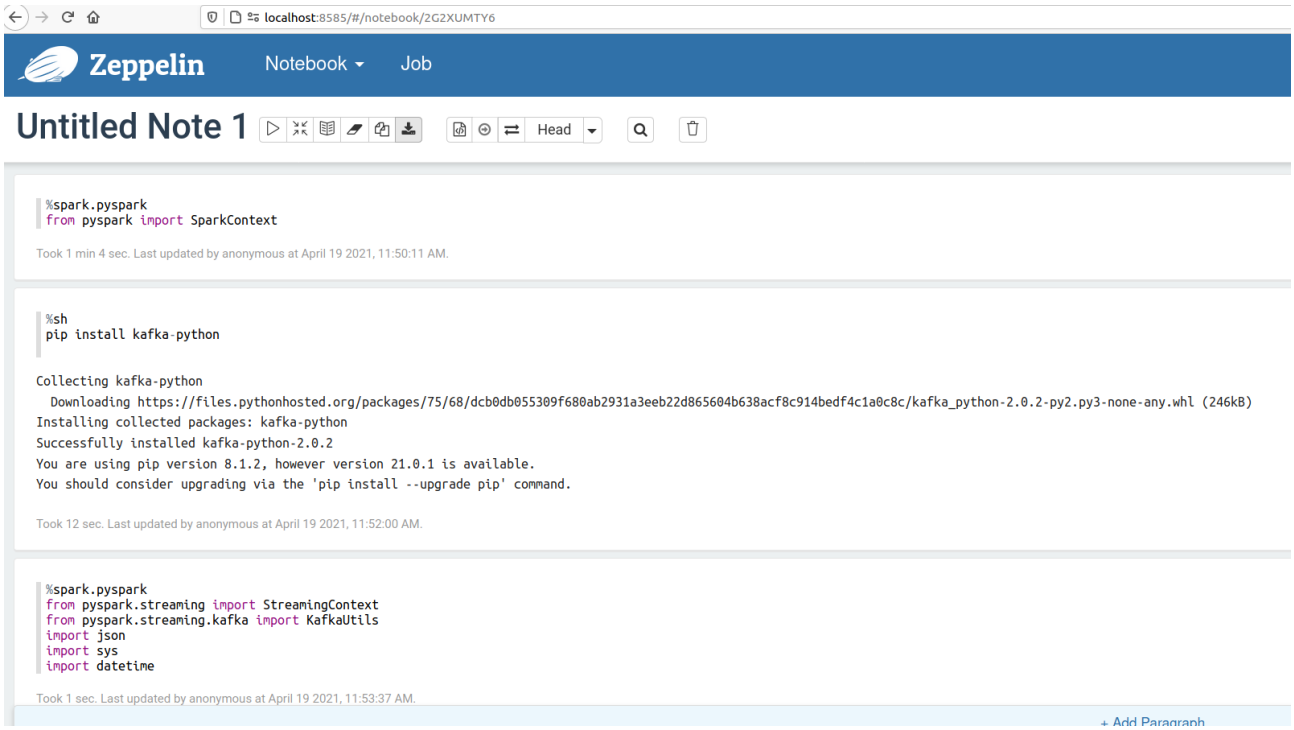
```
/opt/kafka_2.13-2.7.0 # ./bin/kafka-topics.sh --list --zookeeper zookeeper:2181
__consumer_offsets
__transaction_state
aviation_topic
test_aviation
```

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





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