II 3502 - Lab : Introduction to Apache Kafka

Hervé RIVIERE - herve.riviere@isep.fr

January 2020

1 Kafka use cases

Question 0 :
Indicate if Apache Kafka can fit or not for the following use cases :

1. Server log aggregation (being able to collect and read log or events from multiple servers and / or applications)
2. Transfert of files (from couple of kB to GB) between server and / or application
3. Event bus allowing distinct applications to share and consume events (example : a web server is producing login events and we want the anti-fraud application and the CRM consuming and reacting to this event)
4. Event store allowing to store (and if needed, reread in the same order) events produced by an application
5. Database (like MySQL or postgresSQL)
6. Messaging system (like Whatsapp or Hangout)
7. Make data pipeline / streaming ETL (for instance transfert specific records from a database to elastic search and a file in FTP)

Question 1 :
What's a Kafka topic ?

Question 2 :
True or false ?
A event inside Kafka can be modified ?

2 The lab environement : a fully configured remote desktop

A server with Apache Guacamole (web browser remote desktop solution) was configured for each student. Therefore only a web browser like Chrome or Firefox is needed.

If not yet given, ask the instructor for ip, username and password.

Warning : Be careful with keyboard shortcuts and copy and paste commands : copy and paste will only work inside the remote desktop.
For Mac users : Please use "Windows like" shortcuts inside the remote desktop (so CTRL + C and not CMD +C)

Warning : For copy-past inside the remote desktop, best is to open this PDF in Chromium inside the remote desktop (file on the desktop)

2.1 Adapt screen resolution

If needed you can change the screen resolution of the session by following instructions of the README file located on the desktop.
For a better usuability you can also switch your web browser to a full-screen mode and open these instructions on the remote session (file is located on the desktop)

2.2 The lab environment

The lab environment is simulating a multi-servers kafka infrastucture. It contains

- A running zookeeper server (port 2181)
- Three running kafka servers - a kafka server is often called also a kafka broker (port 9092, 9093 and 9094)
- Basic monitoring of the Kafka servers using prometheus (port 9090)
- A java IDE (InteliJ) with some java classes to interact with the Kafka cluster

Machine is connected to internet and your user has paswordless root permission

Warning : Having only one Zookeeper and multiple kafka brokers in a same server is a setup only valid for dev / experimental use cases. In a production grade environment each of these components will be on separate hosts and Zookeeper will be at least deployed on three servers.

2.3 Source code

All the java code used in this lab and also all the code (Ansible) to setup the enviroment is available on this github : https://github.com/HerveRiviere/isep-kafka-lab

3 Why Zookeeper ?

3.1 Initial checks before starting to play...

In a terminal, execute the following command to check all service are well running. Command result should indicate 'running' for each.

```
sudo systemctl status zookeeper
sudo systemctl status kafka-1
sudo systemctl status kafka-2
sudo systemctl status kafka-3
```

If one service is indicated as dead you can execute the following command. If it's not fixing the issue please call the instructor.

```
sudo systemctl start <service>
# example :  sudo systemctl start kafka-3
```

3.2 Zookeeper, a discovery solution

Zookeeper is an external component allowing kafka brokers to :

- Discover them each others : which servers are part of the cluster
- Discover cluster metadata : topic list, users and permissions...
- What's the responsability of each server (leader or follower) and if needed, play some consensus algorithm. Example Zookeeper is used to elect one and only one kafka server as cluster controller.

Warning : Zookeeper is only storing kafka cluster metadata. Kafka data (ie. the events) are stored in the local file system of the Kafka brokers.

3.3 Play with zookeeper-shell

You can connect and explore to the Zookeeper cluster (here localhost:2181) using the following command in a terminal. You can open a terminal with a shortcut on the Desktop.

```
/opt/kafka/bin/zookeeper-shell.sh localhost:2181
# Zookeeper is organized like a unix file system
# where each node (called a znode) can contains data and / or have children
#
# Display children of /
ls /

# Get data contained in /controller znode
get /controller

# Display children of /brokers and /brokers/ids
ls /brokers
ls /brokers/ids

# Get data contained in /brokers/ids/1 (the kafka broker id 1)
get /brokers/ids/1
```

Let kill the Kafka cluster contoller indicated in the znode /controller.
In another terminal

```
# If the cluster contoller is the kafka-1 (brokerid 1)
sudo systemctl stop kafka-1
```

Re-execute the command in the previous section in zookeeper-shell to check who is the new controller (get /controller) and the number of alive brokers (ls /brokers/ids)
You can then restart the stopped broker

```
# If kafka-1 was the killed broker
sudo systemctl start kafka-1
```

Warning : Checking manually data inside Zookeeper is here only to explain how kafka servers exchange metadata internally. In a normal use of Kafka you will use command line tool provided by kafka (example kafka-topic.sh) that will abstract the use of Zookeeper.

Question 3 :
Is a Kafka cluster can still working if Zookeeper cluster is unavailable ?

4 Produce and consume some events through Kafka

4.1 Create a topic

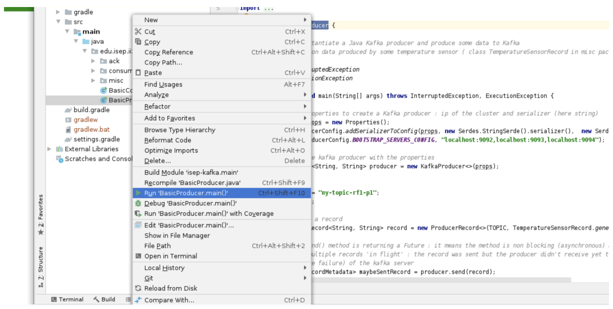
The first step is to create a topic we can after use in our Kafka producer (exaclty like a table in a database). At this point we will create the simplest kafka topic possible : 1 partition and 1 replica (more about these parameters after)

```
/opt/kafka/bin/kafka-topics.sh --create --topic my-topic-rf1-p1 \
--replication-factor 1 --partitions 1 --zookeeper localhost:2181

# Note we are giving zookeeper server ip
# to the command line as topic name are stored in Zookeeper
```

4.2 Produce messages

Using IntelliJ, open and run the java class edu.isep.ii3502.kafka.BasicProducer



```
import ...
ducer {
tantiate a Java Kafka producer and produce some data to Kafka
on data produced by some temperature sensor ( class TemperatureSensorRecord in misc pac
uptedException
ionException
d main(String[] args) throws InterruptedException, ExecutionException {
operties to create a Kafka producer : ip of the cluster and serializer (here string)
ops = new Properties();
cerConfig.addSerializerToConfig(props, new Serdes.StringSerde().serializer(),  new Serd
ducerConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092,localhost:9093,localhost:9094");
e kafka producer with the properties
<String, String> producer = new KafkaProducer<>(props);
= "my-topic-rf1-p1";
;
 a record
ecord<String, String> record = new ProducerRecord<>(TOPIC, TemperatureSensorRecord.gene
nd() method is returning a Future : it means the method is non blocking (asynchronous)
ultiple records 'in flight' : the record was sent but the producer didn't receive yet t
e failure) of the kafka server
cordMetadata> maybeSentRecord = producer.send(record);
```

Figure 1: Click right on the java class and then click on run

By checking prometheus (localhost:9090 in chromium), type in expression field

`kafka_server_topic_messagesinpersec{topic='my-topic-rf1-p1'}`

You should be able to observe the number of message in your topic (you can click on graph to get the metric history) and get the broker$_{i}dreceivingtheload(ifneededscrolldownthepagetoseethegraphlegend)$

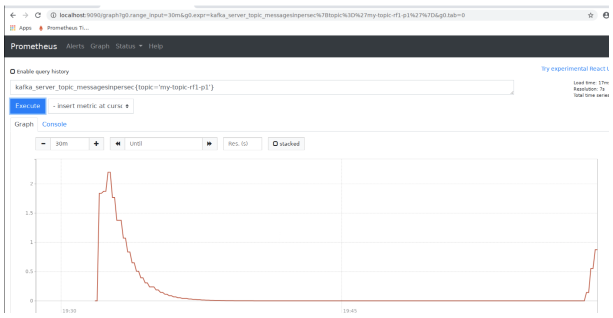

localhost:9090/graph?g0.range_input=30m&g0.expr=kafka_server_topic_messagesinpersec%7Btopic%3D%27my-topic-rf1-p1%27%7D&g0.tab=0
Apps
Prometheus Ti...
Prometheus
Alerts
Graph
Status
Help
Enable query history
Try experimental React UI
kafka_server_topic_messagesinpersec{topic='my-topic-rf1-p1'}
Load time: 17ms
Resolution: 7s
Total time series
Execute
- insert metric at cursor
Graph
Console
30m
Until
Res. (s)
stacked
2
1.5
1
0.5
0
19:30
19:45

Figure 2: Number of messages per second in prometheus

4.3 Consume messages

With the java class edu.isep.ii3502.kafka.BasicProducer still running, run now the class
edu.isep.ii3502.kafka.BasicConsumer Follow the instructions in the code to consume from beginning, from tail or from a specific offset.

Question 4 :
What's a kafka offset ?

5 Experiment failure

5.1 Failure with non replicated topic

Keep the producer and consumer application running. Go back to prometheus and check with the metric label which kafka broker is handling the load.
Kill the corresponding service

```
sudo systemctl stop kafka-2 # if broker_id 2 was handling the load
```

Check the messages per second in prometheus and your java application logs.

Question 5 :
What's happening ? Is your application is fault tolerant ? How to solve the issue ?

Restart the killed broker

```
sudo systemctl start kafka-2 # if broker_id 2 was handling the load
```

5.2 Create a replicated topic

Execute the following command to create a topic with a replicated factor of two

```
/opt/kafka/bin/kafka-topics.sh --create --topic my-topic-rf2-p1 \
--replication-factor 2 --partitions 1 --zookeeper localhost:2181
```

Check the status of your new topic with the following command

```
/opt/kafka/bin/kafka-topics.sh --describe --topic my-topic-rf2-p1 \
--zookeeper localhost:2181
```

Keep note of the leader id indicated.
Modify the java code of the producer and consumer to change the topic name to my-topic-rf2-p1

Do the same operation than the previous section and kill again the broker handling the load (change the topic name to my-topic-rf2-p1 in prometheus metric to get the broker id). What is happening, is the application is now fault-tolerant ?

Recheck the status of your topic with the same command than before

```
/opt/kafka/bin/kafka-topics.sh --describe --topic my-topic-rf2-p1 \
--zookeeper localhost:2181
```

What's the leader id now ?

Question 6 :
Is the kafka replication model is a master-follower or a master-master ?

Question 7 :
Is this configuration is scalable ? Why ?

Stop all your running java classes.

6 Scaling - broker side

As you saw using prometheus, in our current configuration, one kafka broker is taking the full load when others are just standby replicas (and so we will have an issue when the load will be more important than the capacity of one server !). We will create a multi partitions topic to solve this issue.

6.1 Create the topic

```
/opt/kafka/bin/kafka-topics.sh --create --topic my-topic-rf2-p4 \
--replication-factor 2 --partitions 4 --zookeeper localhost:2181
```

You can describe the topic

```
/opt/kafka/bin/kafka-topics.sh --describe --topic my-topic-rf2-p4 \
--zookeeper localhost:2181
```

Check the partitions leaders (and replicas) are well balanced between servers.

6.2 Produce

Modify edu.isep.ii3502.kafka.BasicProducer, TOPIC_NAME variable to set it to my-topic-rf2-p4 and execute it.

Check application log, especially the partition and the offset of each record sent. You can also check the number of messages per seconds using prometheus.

Question 8 :
How the producer is choosing partition of a record ?

Question 9 :
By checking prometheus, why one server is receiving more messages than others ?

You can now launch multiple producers (just click on run a second time) to simulate a second server producing records.

6.3 Consume

Modify edu.isep.ii3502.kafka.BasicConsumer, TOPIC_NAME variable to set it to my-topic-rf2-p4 and execute it.

You can see your consumer receiving records of all partitions.

Try to launch a second consumer, you can also see that both of your consumers are receiving the same records (and so each record is proccessed two times)

Question 10 :
Is edu.isep.ii3502.kafka.BasicConsumer a scalable and a fault tolerant application ?

7 Scaling - consumer side

In order of well balance the load between consumer application we will use the consumer group feature of Kafka.

Stop all java applications except one producer to my-topic-rf2-p4.

Open edu.isep.ii3502.kafka.consumergroup.ConsumerWithinAConsumerGroup.

Check ConsumerConfig.GROUP_IDP_CONFIG, compare it value with the value that was set in edu.isep.ii3502.kafka.BasicConsumer.

ConsumerWithinAConsumerGroup class is logging the partition assigned to the consumer.

Execute ConsumerWithinAConsumerGroup, observe that the application is assigned to all partitions of the topic.

Execute a second instance (with the first one still running) of ConsumerWithinAConsumerGroup class. What are now the partitions assigned to each application.

Start now a third and fourth instances of ConsumerWithinAConsumerGroup. Observe the partitions assigned.

Come back to only one instance of ConsumerWithinAConsumerGroup. Observe the partitions assigned.

Question 11 :
Explain why edu.isep.ii3502.kafka.consumergroup.ConsumerWithinAConsumerGroup is a scalable and fault tolerant application.

Question 12 :
What's happen if you launch 5 instances of the ConsumerWithinAConsumerGroup? Is all instances are reciving records ? Why ?

Stop now the last consumer and take note of the last tuple (offset, partition) read by this consumer.

Keep one producer application and wait a little bit (30 seconds). Take note of the last tuple (offset, partition) sent by your producer.

Start one consumer application. Take note of the first (offset, partition) read by this consumer. Assert there is no message lost in your consuming application.

Question 13 :
How do you think kafka is storing consumer offset ? Is the consumer application is at-least-once; at-most-once or exactly once ? How to have your consumer application exactly-once ?

8 Count records per sensor id

8.1 Complete code

Kill all consumers running.

We now want to count records by sensor id.

Open edu.isep.ii3502.kafka.consumergroup.ConsumerCountBySensorId. Complete code to produce an output like

```
sensor_id : 25   record_count : 25
sensor_id : 27   record_count : 32
sensor_id : 25   record_count : 26
(...)
```

Run now multiple instances of your application, what's the issue ?

Question 14 :
Explain how you can partition data to avoid having data inconsistency when you have multiple instances of your application ?

8.2 Add key to records

Modify edu.isep.ii3502.kafka.BasicProducer to use sensor id as Kafka record key. You can do the following change :

- Get the output of generateRecord() method in a variable
- Extract the sensor id value in a new variable
- In the send() method, add the key argument (new ProducerRecord(TOPIC, sensorID, Temperature-SensorRecord.generateRecord())

Restart the producer and rerun multiple instances of ConsumerCountBySensorId, do you still have data inconsistency ?

Question 15 :
Explain the relationship between kafka record key and kafka partition

Question 16 :
Is your application with my-topic-rf2-p4 topic, consumer group and key partitionning is scalable and fault tolerant ? Why ?

9 Data consistency inside Kafka

Question 17 :
For a topic with mulitple partitions, and a consumer reading all partitions. Do you have a global order of records (all records are read in the same order than they was sent) ? Why ? Which order guarantee do you have ?

Kafka producers allows to have different mode of acknowledgement (ack setting). Open edu.isep.ii3502.kafka.ack.ProducerW
java class.

Kafka producer have the three following ack mode :

- ack = 0 : "send and forget mode". Kafka producer are not waiting any answer from kafka server or is not doing any retry.
- ack = 1 : "just wait leader answer" : Kafka producer is waiting answer from the kafka leader handling the partition the record was sent. Some retries can be set in case of failure.
- ack = all : "wait leader and replica answers" : Kafka producer is waiting answer from the kafka leaderand replica handling the partition the record was sent. Some retries can be set in case of failure.

Question 18 :
For all ack settings (0, 1, all) indicate if it's al-least-once (so generate duplicate); at-most-once (so can generate message lost) or exactly-once ? Why ?

Question 19 :
What are the pros and cons of each acks setting ?