Charles Mabbyalas
Océane Salmeron
ING5 - BDA2
ECE Paris

Big Data Ecosystem Project : Streaming Data with Kafka

Sommaire

Sommaire **2**

I. Project overview **3**

1. Data Streaming 3
2. Kafka 3

II. Project **4**

I. Project overview

1. Data Streaming

Data streaming is also known as stream processing. Streaming Data is data that is generated continuously by thousands of data sources, which typically send in the data records simultaneously, and in small sizes (order of Kilobytes).

We use the term “streaming” to describe continuous and endless data streams with no starting or finishing points. These data streams provide a constant feed of data that can be used without downloading it first.

By using this technology, we can process, store, and analyze data streams when generated in real-time.

It includes a wide variety of data such as traffic sensors, health sensors, transaction logs, and activity logs.

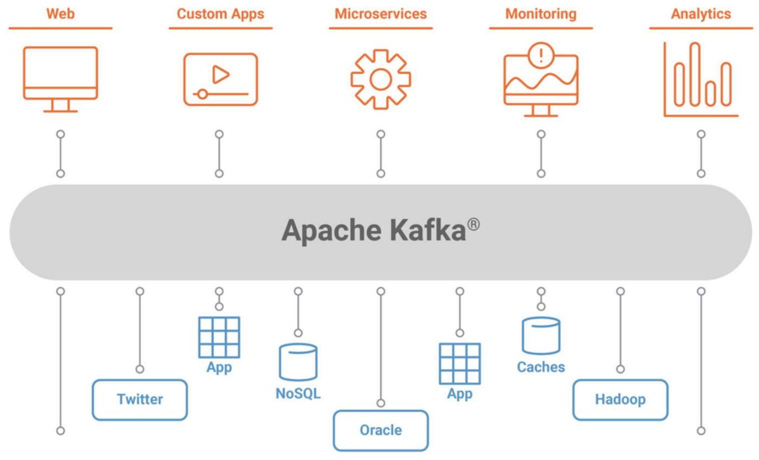
2. Kafka

Kafka is a distributed system consisting of servers and clients that communicate via a high-performance TCP protocol.

It publishes and subscribes to streams of records, similar to a message queue or enterprise messaging system, stores streams of records in a fault-tolerant durable way, and processes streams of records as they occur.

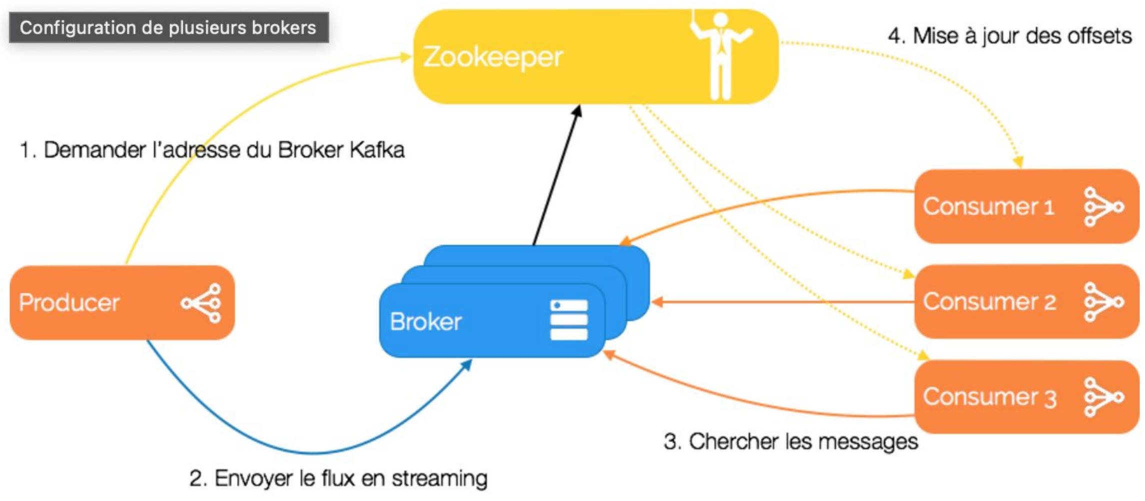
Unlike messaging queues, Kafka is a highly scalable, fault tolerant distributed system, allowing it to be deployed for applications like managing passenger and driver matching, providing real-time analytics and predictive maintenance, and performing numerous real-time services.

Kafka is run as a cluster on one or additional servers which will span multiple datacenters. The Kafka cluster stores streams of records in categories known as topics. Every consists of a key, a value, and a timestamp.



Schema of Apache Kafka terminology

3. Zookepper


Configuration de plusieurs brokers
Zookeeper
4. Mise à jour des offsets
1. Demander l'adresse du Broker Kafka
Consumer 1
Producer
Broker
Consumer 2
Consumer 3
3. Chercher les messages
2. Envoyer le flux en streaming

ZooKeeper is an open source Apache project that provides a centralized service for providing configuration information, naming, synchronization and group services over large clusters in distributed systems. The goal is to make these systems easier to manage with improved, more reliable propagation of changes.

ZooKeeper follows a simple client-server model where clients are nodes (i.e., machines) that make use of the service, and servers are nodes that provide the service. These services are used by distributed applications in general, and by Kafka in particular. To avoid the complexity and difficulty of their manual implementation, Zookeeper is used.

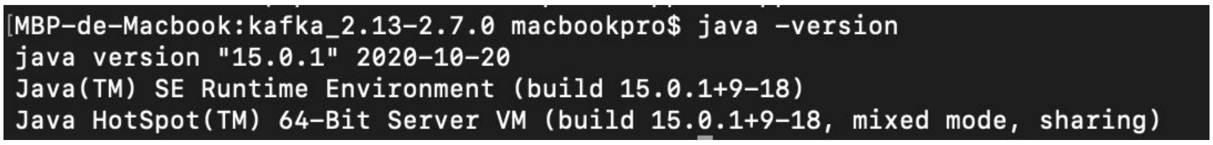
A Kafka cluster typically consists of several brokers to maintain load balancing. These brokers are stateless, which is why they use Zookeeper to maintain cluster state.

II. Project

Year 2020 was quite a difficult year with the crises that we all know and the quarantine. Like we couldn't do anything, one of the things that made us able to survive in our point of vue is video streaming web site like netflix. For this project our idea was to stream video like we can do one netflix amazon prime and others locally.

We first wanted to use kafka along with flink to build a data pipeline but we found that we only needed kafka and python to achieve what we wanted.

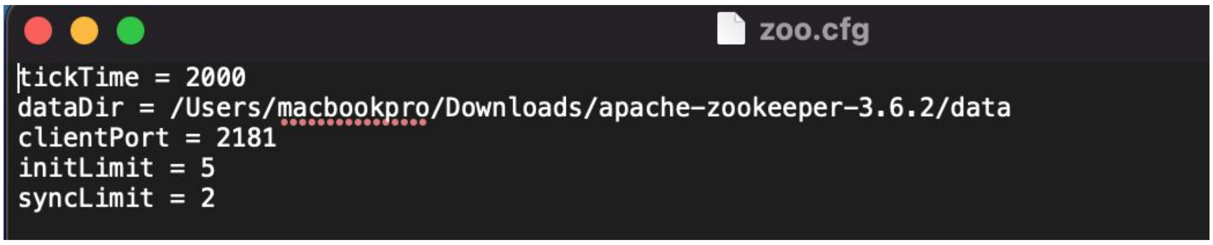
The first step for this project was to install kafka, zookeeper and kafka-python. To use kafka we need zookeeper and both zookeeper and kafka need java that we already had on our computer.



```
[MBP-de-Macbook:kafka_2.13-2.7.0 macbookpro$ java -version
java version "15.0.1" 2020-10-20
Java(TM) SE Runtime Environment (build 15.0.1+9-18)
Java HotSpot(TM) 64-Bit Server VM (build 15.0.1+9-18, mixed mode, sharing)
```

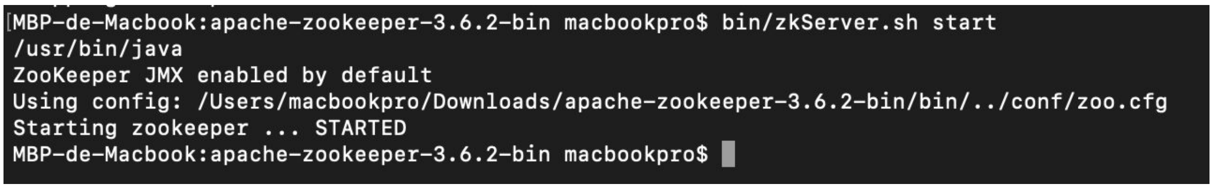
The second step was to download zookeeper and install it. That part was the one where we had the more problems because there was multiple error. After long our of search we found out that this was because java was installed on our computer via anaconda and anaconda caused rights issues that made zookeeper.

So to install zookeeper we first downloaded it from their website. Then created a configuration file where we set the "datadir", the location to store the in-memory database snapshots and, unless specified otherwise, the transaction log of updates to the database.



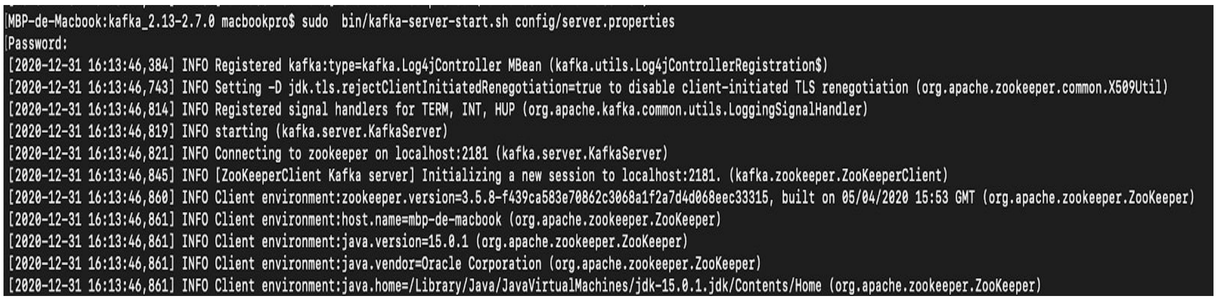
```
tickTime = 2000
dataDir = /Users/macbookpro/Downloads/apache-zookeeper-3.6.2/data
clientPort = 2181
initLimit = 5
syncLimit = 2
```

Then we used the zkServer start command to launch it on the server.



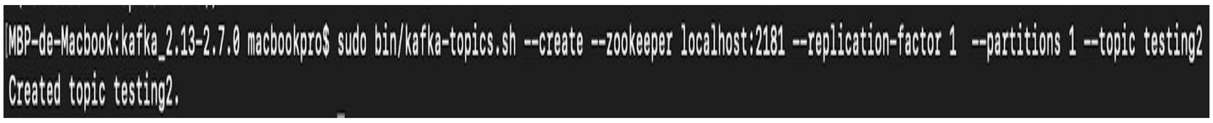
```
[MBP-de-Macbook:apache-zookeeper-3.6.2-bin macbookpro$ bin/zkServer.sh start
/usr/bin/java
ZooKeeper JMX enabled by default
Using config: /Users/macbookpro/Downloads/apache-zookeeper-3.6.2-bin/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
MBP-de-Macbook:apache-zookeeper-3.6.2-bin macbookpro$
```

Then we downloaded kafka from there website , installed it and configured server properties and launch kafka.



```
MBP-de-Macbook:kafka_2.13-2.7.0 macbookpro$ sudo  bin/kafka-server-start.sh config/server.properties
Password:
[2020-12-31 16:13:46,384] INFO Registered kafka:type=kafka.Log4jController MBean (kafka.utils.Log4jControllerRegistration$)
[2020-12-31 16:13:46,743] INFO Setting -D jdk.tls.rejectClientInitiatedRenegotiation=true to disable client-initiated TLS renegotiation (org.apache.zookeeper.common.X509Util)
[2020-12-31 16:13:46,814] INFO Registered signal handlers for TERM, INT, HUP (org.apache.kafka.common.utils.LoggingSignalHandler)
[2020-12-31 16:13:46,819] INFO starting (kafka.server.KafkaServer)
[2020-12-31 16:13:46,821] INFO Connecting to zookeeper on localhost:2181 (kafka.server.KafkaServer)
[2020-12-31 16:13:46,845] INFO [ZooKeeperClient Kafka server] Initializing a new session to localhost:2181. (kafka.zookeeper.ZooKeeperClient)
[2020-12-31 16:13:46,860] INFO Client environment:zookeeper.version=3.5.8-f439ca583e70862c3068a1f2a7d4d068eec33315, built on 05/04/2020 15:53 GMT (org.apache.zookeeper.ZooKeeper)
[2020-12-31 16:13:46,861] INFO Client environment:host.name=mbp-de-macbook (org.apache.zookeeper.ZooKeeper)
[2020-12-31 16:13:46,861] INFO Client environment:java.version=15.0.1 (org.apache.zookeeper.ZooKeeper)
[2020-12-31 16:13:46,861] INFO Client environment:java.vendor=Oracle Corporation (org.apache.zookeeper.ZooKeeper)
[2020-12-31 16:13:46,861] INFO Client environment:java.home=/Library/Java/JavaVirtualMachines/jdk-15.0.1.jdk/Contents/Home (org.apache.zookeeper.ZooKeeper)
```

Then we need check if kafka is well started by creating a topic

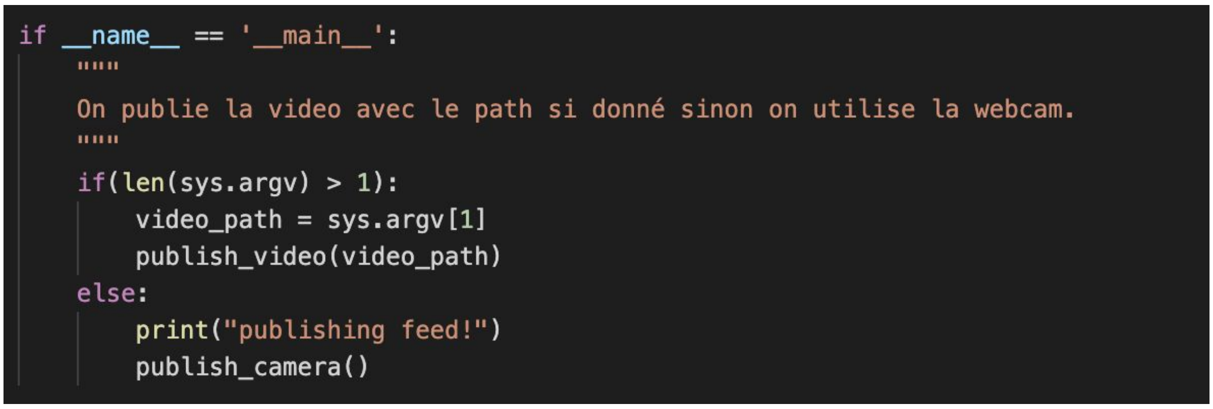


```
MBP-de-Macbook:kafka_2.13-2.7.0 macbookpro$ sudo bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1  --partitions 1 --topic testing2
Created topic testing2.
```

Now that kafka is installed correctly we can publish our video for that we now have to create a consumer and a producer, we are going to do it in python.

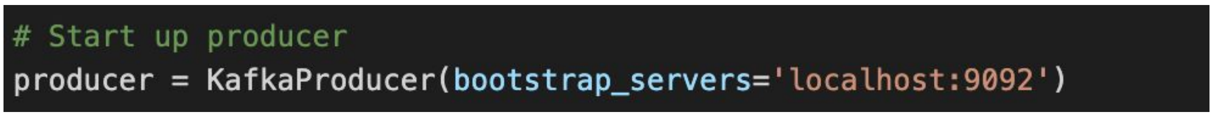
From kafka we import kafkaProducer and kafkaConsumer that will write and read the data. We will publish video by two ways : with camera straming live video from your webcam and from a video fill. The two processes are similar.

For the consumer we also need Flask that is a lightweight WSGI (Web Server Gateway Interface) web application framework to publish the video.



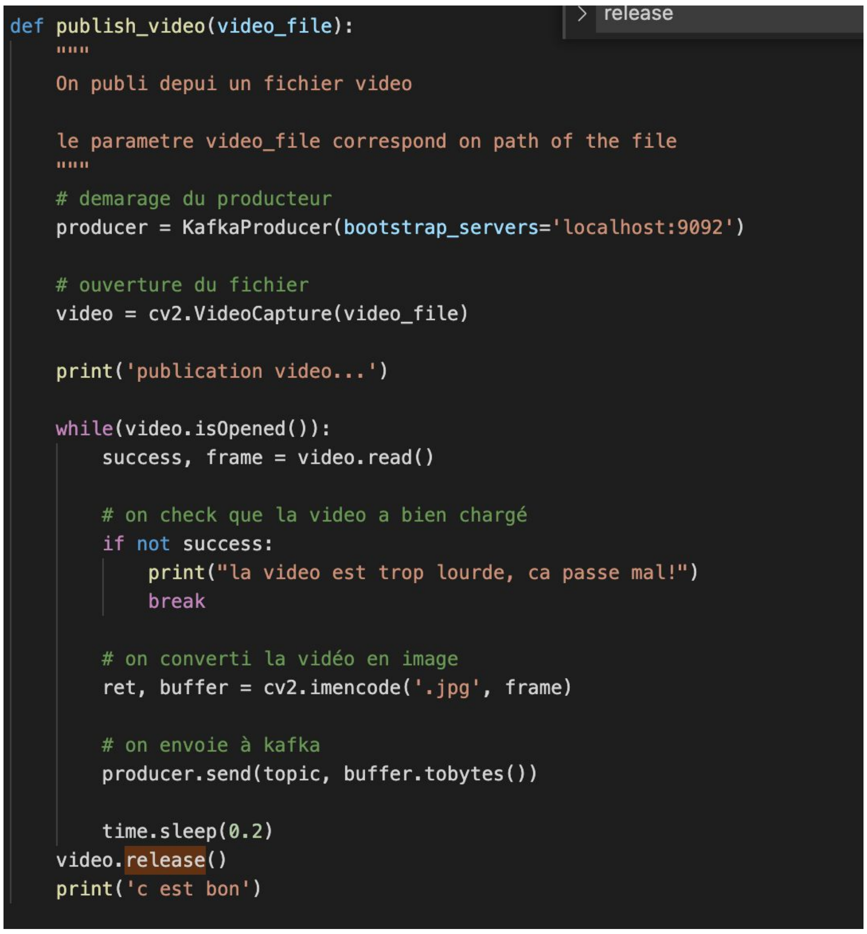
```python
if __name__ == '__main__':
    """
    On publie la video avec le path si donné sinon on utilise la webcam.
    """
    if(len(sys.argv) > 1):
        video_path = sys.argv[1]
        publish_video(video_path)
    else:
        print("publishing feed!")
        publish_camera()
```

For the producer we first start a producer



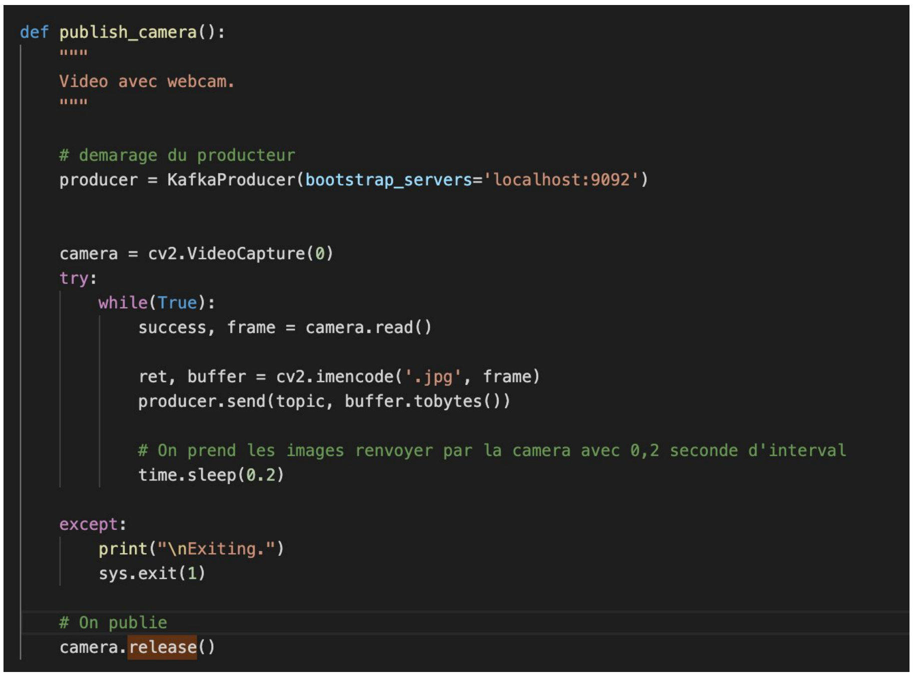
```python
# Start up producer
producer = KafkaProducer(bootstrap_servers='localhost:9092')
```

For the video we open the video, convert it into png image then send it to kafka.



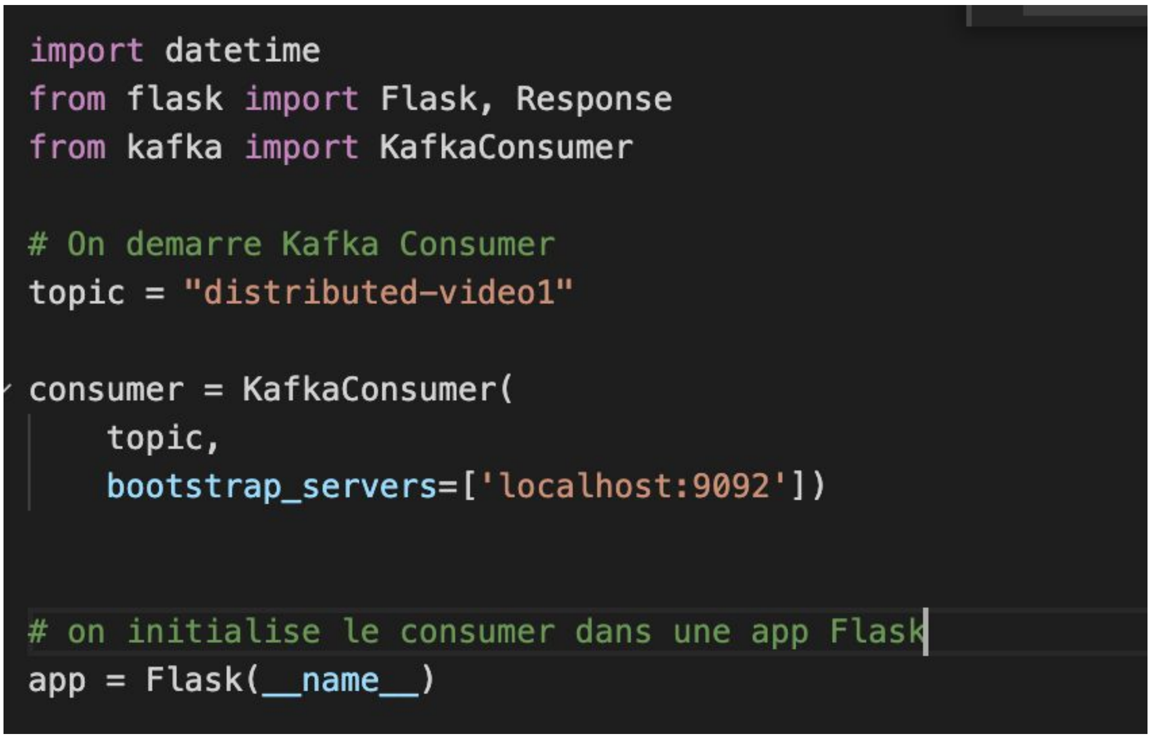
```python
def publish_video(video_file):
    """
    On publi depui un fichier video

    le parametre video_file correspond on path of the file
    """
    # demarage du producteur
    producer = KafkaProducer(bootstrap_servers='localhost:9092')

    # ouverture du fichier
    video = cv2.VideoCapture(video_file)

    print('publication video...')

    while(video.isOpened()):
        success, frame = video.read()

        # on check que la video a bien chargé
        if not success:
            print("la video est trop lourde, ca passe mal!")
            break

        # on converti la vidéo en image
        ret, buffer = cv2.imencode('.jpg', frame)

        # on envoie à kafka
        producer.send(topic, buffer.tobytes())

        time.sleep(0.2)
    video.release()
    print('c est bon')
```

When we use the camera it's the same thing except we take images from the camera with a 0.2 second interval.



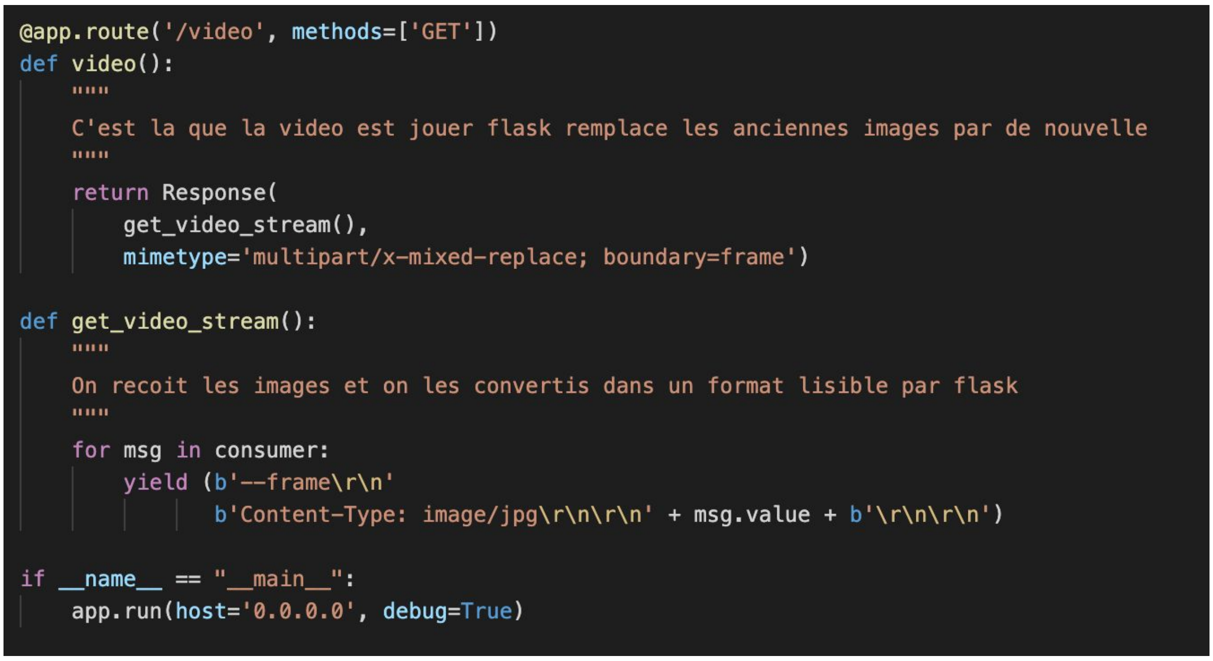
```python
def publish_camera():
    """
    Video avec webcam.
    """

    # demarage du producteur
    producer = KafkaProducer(bootstrap_servers='localhost:9092')


    camera = cv2.VideoCapture(0)
    try:
        while(True):
            success, frame = camera.read()

            ret, buffer = cv2.imencode('.jpg', frame)
            producer.send(topic, buffer.tobytes())

            # On prend les images renvoyer par la camera avec 0,2 seconde d'interval
            time.sleep(0.2)

    except:
        print("\nExiting.")
        sys.exit(1)

    # On publie
    camera.release()
```

For the consumer we start kafka and initialise a Flask App



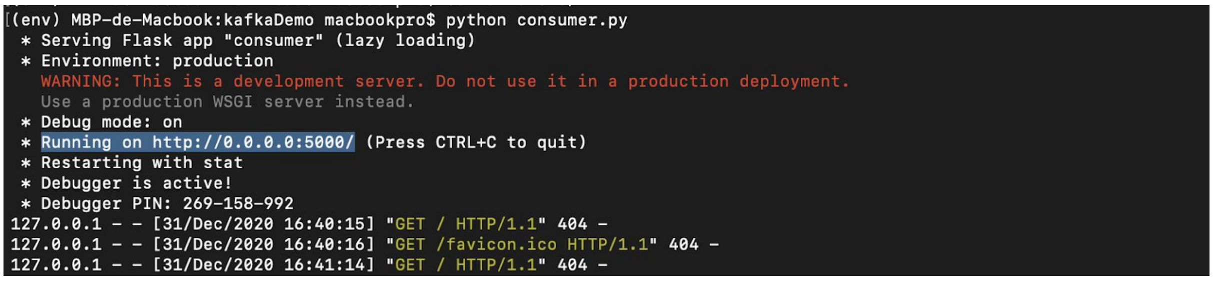
```python
import datetime
from flask import Flask, Response
from kafka import KafkaConsumer

# On demarre Kafka Consumer
topic = "distributed-video1"

consumer = KafkaConsumer(
    topic,
    bootstrap_servers=['localhost:9092'])


# on initialise le consumer dans une app Flask
app = Flask(__name__)
```

Then we just have to publish the video on the right port.

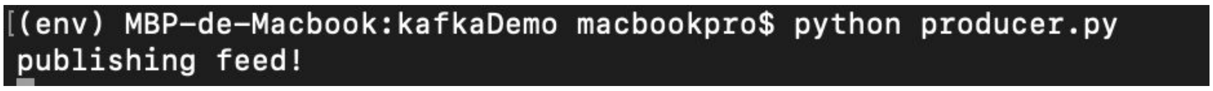


```python
@app.route('/video', methods=['GET'])
def video():
    """
    C'est la que la video est jouer flask remplace les anciennes images par de nouvelle
    """
    return Response(
        get_video_stream(),
        mimetype='multipart/x-mixed-replace; boundary=frame')

def get_video_stream():
    """
    On recoit les images et on les convertis dans un format lisible par flask
    """
    for msg in consumer:
        yield (b'--frame\r\n'
               b'Content-Type: image/jpg\r\n\r\n' + msg.value + b'\r\n\r\n')

if __name__ == "__main__":
    app.run(host='0.0.0.0', debug=True)
```

We then start the consumer and the producer (using the web cam).

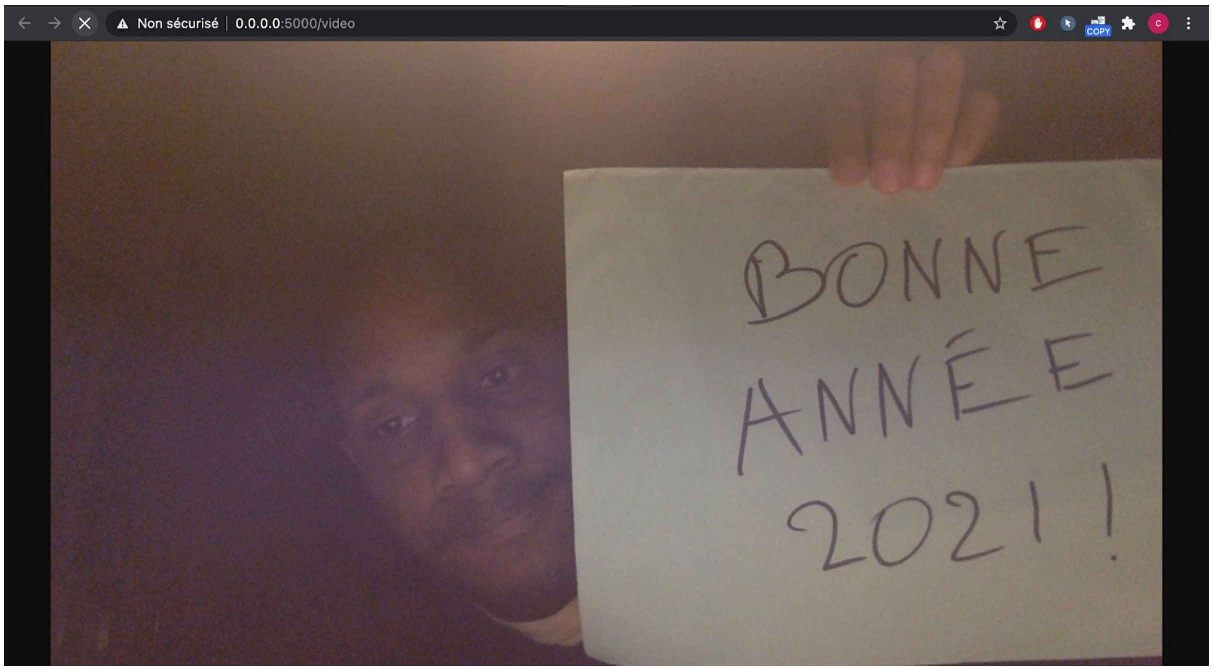


```
[(env) MBP-de-Macbook:kafkaDemo macbookpro$ python consumer.py
 * Serving Flask app "consumer" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 269-158-992
127.0.0.1 - - [31/Dec/2020 16:40:15] "GET / HTTP/1.1" 404 -
127.0.0.1 - - [31/Dec/2020 16:40:16] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [31/Dec/2020 16:41:14] "GET / HTTP/1.1" 404 -
```



```
[(env) MBP-de-Macbook:kafkaDemo macbookpro$ python producer.py
publishing feed!
```

Here is the result :



This project was very interesting to do because we had to do our own research and we were able to practice things we didn't have the time to do in class.

Ressources:

- https://www.baeldung.com/kafka-flink-data-pipeline
- https://www.tutorialspoint.com/zookeeper/zookeeper_installation.htm
- https://medium.com/@kevin.michael.horan/distributed-video-streaming-with-python-and-kafka-551de69fe1dd
- https://zookeeper.apache.org/doc/r3.3.3/zookeeperStarted.html
- https://www.lebigdata.fr/python-langage-definition
- https://insatunisia.github.io/TP-BigData/tp3/